Introduction to Computer Graphics

Project report

Kim Lan Phan Hoang, Tim Nguyen, Lucie Perrotta

Part 1 - Procedural Terrain

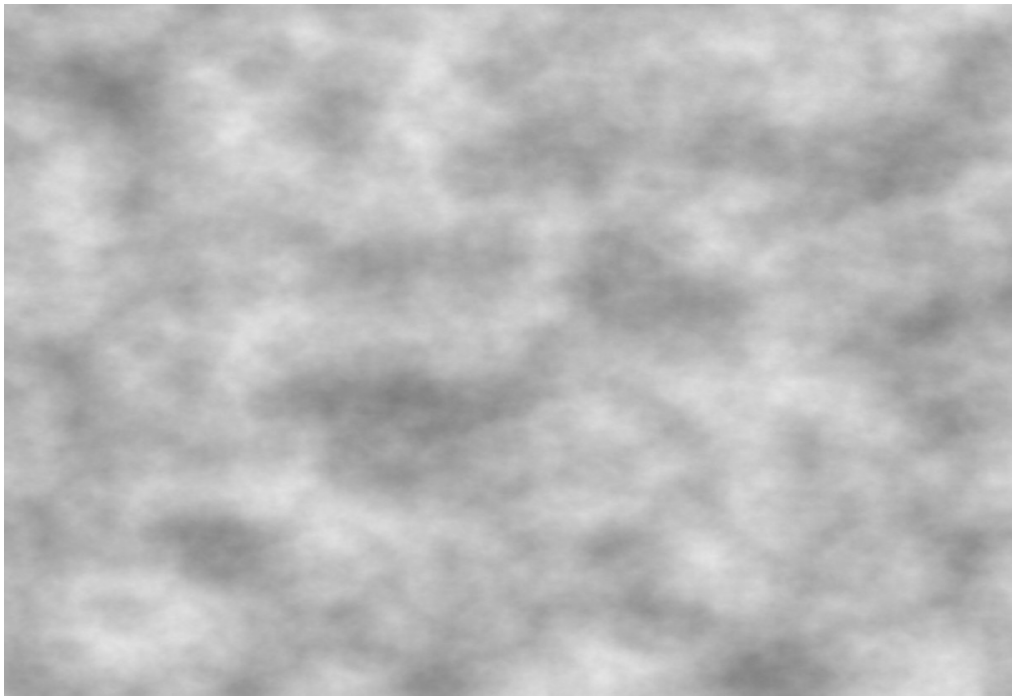
Part 1.1 - Height Map Generation

We use as basis the last homework, keeping the screenquad (used to create Perlin Noise) and adding a grid object (which generate the terrain).

For the perlin noise generation, we followed the explanations given in some slide in the course. In a function named perlin, which take as input the position of the pixel, we wrote the folloing instruction. We first divide the screen into NxN squares, and recompute the coordinates of each points relatively to the local square it is in. We generate a pseudo random vector for each corner of the squares, in a certain fashion so that the values of the random vectors will always be the same for a given point.

Perlin Noise

Then we also compute the vectors between the point and the corners of the square it is in. Then we compute the dot product between the 4 random vectors and the 4 corner-to-point vectors. Finally, we use the given mix function to mix the 4 computed scalars and hence compute the gray value of the color of the point. We just need to normalize this value between 0 to 1 before returning it.Now we can use this function to compute the actual fractal brownian motion. We used a slightly different version from the one given in class, inculing a gain parameter, changing the brightness of the noise at each iteration. For each iteration we go to a more precise variant of the fractal and add it to the previous fractal render. The "shrinkage speed" is determined by the lacunarity parameter, and the H parameter determines the intensity reduction speed between 2 iterations. Then we return a new float gray value for the brownian noise, with arbitrary chosen parameters.



Brownian Noise

The final color value is simply this gray value. We then send it to the frame buffer in onder to use it as a texture.

Part 1.2 - Terrain Rendering

Once the Perlin Noise (and fBm) are generated, we can use it to display a terrain. For that, we bind the framebuffer texture to the grid object. The grid vshader will then be able to use it and compute "random" height which will form mountains.

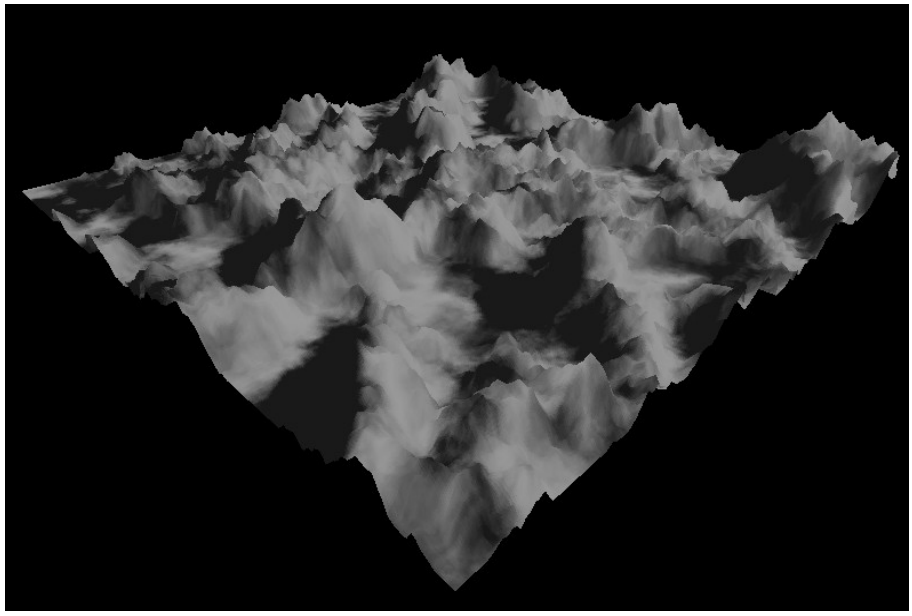
For the terrain coloring, we wanted to map the colors according to the height.
We separated the coloring into two parts : the lake part and the forest-mountains part.
Our first step was to create a texture of colors (yellow, dark green, gray/white) to map the height of the forest and mountains areas. After binding this texture, we used it to color these parts.

For the lake part, we just apply the same color without any gradient effect.
Instead, we added a cloud-mirrored effect, by using the diffuse term we saw in our previous lessons, that was used for the shading.

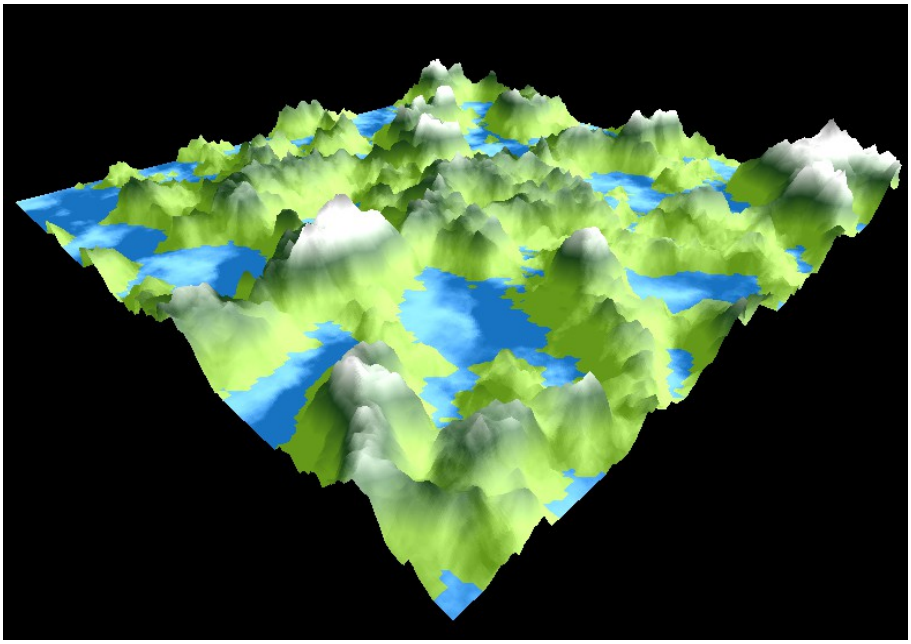
Phong shading is added to the terrain. The implementation is inspired from the previous homework, using same computation and same structures. Unless that normal vectors are computed in the fshader, using neighbor coordinates and cross function. Specular effect is also removed (since moutains don't shine).

Introduction to Computer Graphics

Before coloring



Final rendering



Implementation

Perlin Noise & fBm - Lucie
Terrain generated with Perlin Noise - Kim
Shading - Kim
Coloring terrain depending on height - Tim

Fraction of the workload of each group member

Kim : 33%
Lucie : 33%
Tim: 33%