Introduction to Computer Graphics
Project Part 2 - Rendering

Group 23 - Florian Junker, Louis Séguy, Mathieu Monney

May 5, 2014

1 Overview

In this project part, we added textures to the terrain. Moreover, we refactored the code using object-oriented programming to allow for better modularization. We also improved our terrain generation shaders to have a more realistic landscape. Then we created a skybox to fill the background. Finally, we added some shadows using a shadow map. See Figure 1 for a rendering.

2 Implementation

2.1 Texturing

In order to add some textures to the terrain, we created a `Texture` class which allows us to load TGA textures conveniently. Then we had to modify the fragment shader of the terrain. As suggested in the instructions, we compute and use the slope of the terrain to blend the grass and rock textures to achieve a more realistic look. If we are below or just above the water level, we interpolate this grass and rock texture with the sand texture. To do this, we use the `mix` and `smoothstep` functions. To add some snow, we're also using the slope so that there is always snow above a certain threshold but also a bit below if the slope is not too high. See Figure 2 for a screenshot.

2.2 Modeling the sky

To render the sky, we used a skybox. At first, we drew 12 triangles (2 per face) to model the skybox. We textured them using a single 2D texture and UV mapping, but this led to seams between the different faces. To fix this issue, we used a cube map texture (see Figure 3) instead of a regular 2D texture. It also made it easier to draw the skybox since we can now use indexed drawing instead of drawing 12 distinct triangles. Furthermore, the coordinates for texture mapping are also easier to figure. We can compute them in the vertex shader directly instead of having a list of precomputed UV coordinates.

2.3 Self shadowing

We are using a shadow map to add shadows to our terrain. In the first pass, we're moving the camera to render from the light point of view. We render the

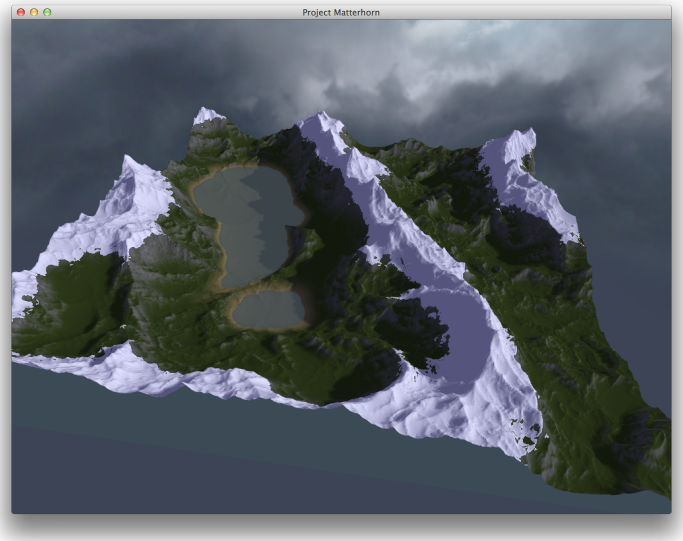

Figure 1: A rendering of our current project state

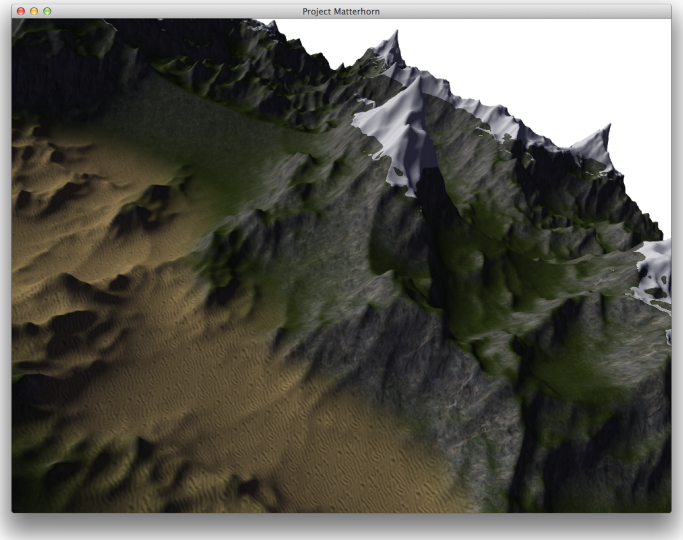

Figure 2: After adding some textures

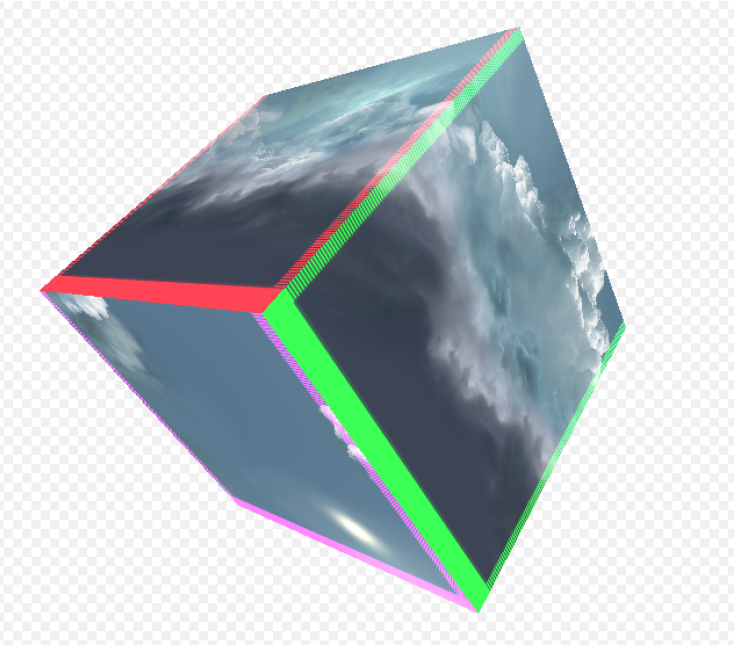

Figure 3: The cube map texture of the skybox

z-buffer in the shadow map texture using an orthogonal projection (since the sun is a directional light). Then, in the second pass, we go back to the usual point of view. We render the terrain as usual, but we sample the shadow map and test if the fragment should be shadowed or not (comparing its distance to the one stored in the shadow map). We had to solve a couple issues to have a decent result. The first one was to remove shadow acne. The solution we chose is to use a bias when comparing the distance. The second issue is the strong aliasing of the shadows. We solved this problem in some different ways. Firstly, we used a `sampler2DShadow`, we increased the size of the shadow map texture and finally we added percentage-closer filtering (PCF) using a Poisson disk. The PCF filtering is quite effective and not too expensive when using 4 samples. Finally, the keys from 1 to 9 allow to choose the position of the sun (from sunrise to noon). See Figure 5 and Figure 4.

2.4 Code modularization

In order to have a better modularization of our code, we refactored it using object-oriented programming. For instance, we have a `Texture` class to load texture from image files, a `HeighMap` class to generate the height map, a `Grid` class to render the terrain, etc. Using destructors and dynamic memory allocation allows for a much easier resource management.

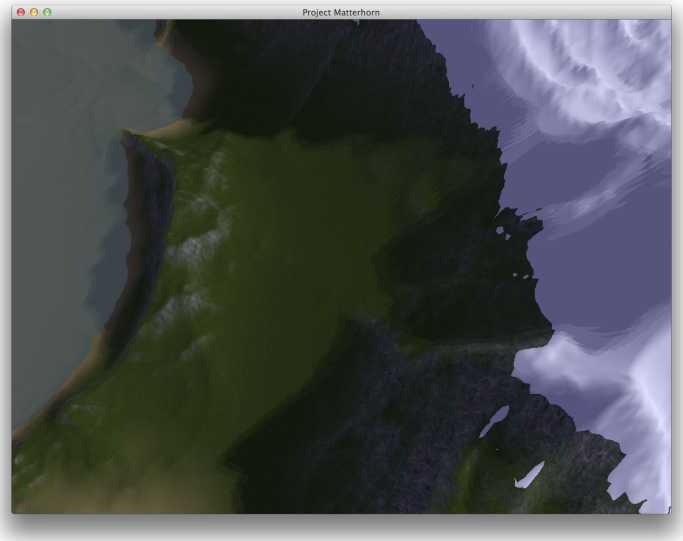

Figure 4: Shadows with PCF filtering

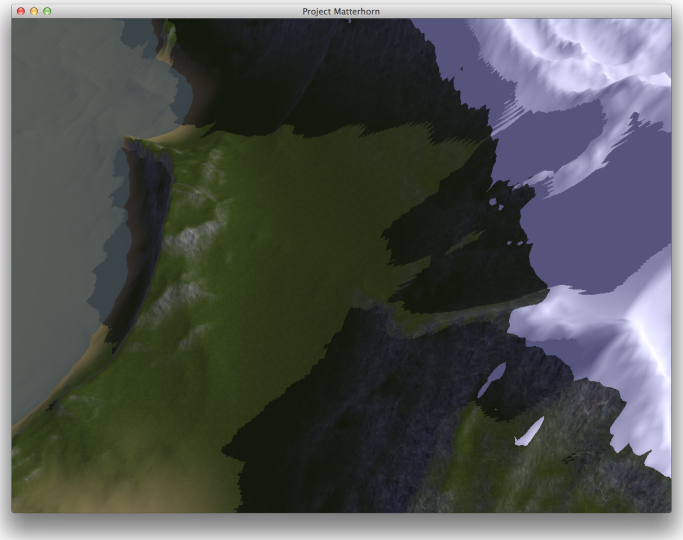

Figure 5: Shadows without any filtering

3 Results

3.1 Texturing

To chose the values for the height at which we display sand, grass, etc, we did the same way as for the first part. Once we had a pleasant terrain we tested different values to have a realistic landscape. Loading the textures is time consuming and we might store them compressed and decompress them on the fly when needed to improve performance.

3.2 Modeling the sky

Generating the skybox is instantaneous. To get a good looking result we had to get a big enough cube so that the sky looks far away. Thus, we also had to tweak the view frustum.

3.3 Self shadowing

The shadowing algorithm is quite costly. Actually, we have to add a new rendering pass to compute the shadow map. Even more costly is the PCF filtering. Indeed, we sample each fragment multiple times (4 times by default in our shader). We also had to find the smallest bias to remove shadow acne without adding a too big offset.

4 Improvements

4.1 Multifractal

To have a more heterogeneous landscape, we implementend a multifractal noise. It allows the terrain to be more diversified – this way we obtain lakes, hills and moutains (see Figure 6 for a screenshot). Though, this kind of terrain doesn't look very realistic, it's too smooth. That's why we added a ridged multifractal noise.

4.2 Ridged multifractal

To overcome this hilly and smooth look, we implemented multifractal. Using an absolute of the perlin noise, we achieve a much better look. This is the noise we're using for our final rendering (as seen in Figure 1).

4.3 Snow level

We have found that a basic line in function of height for the snow limit wasn't really nice or realistic. We added a tiny algorithm that decides where to place snow. This simple implementaion takes into account height and slope.

4.4 Water transparency

We came up with a nice water transparency rendering, using the difference between the depth and the water plane to set the transparency level of the

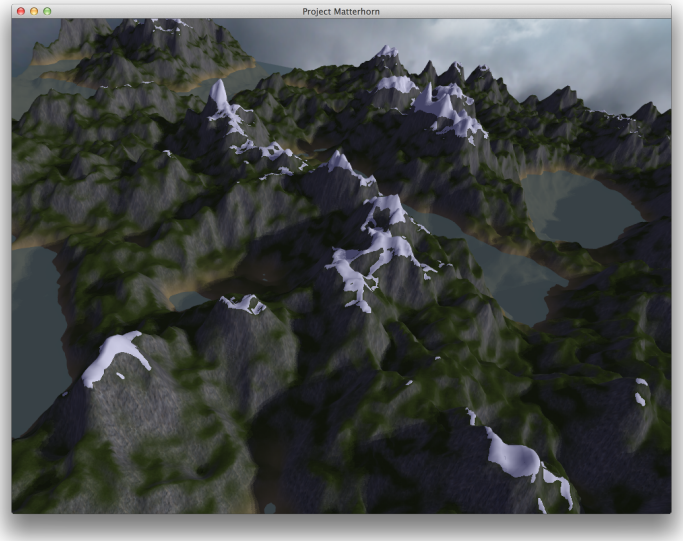

Figure 6: A terrain generated with multifractal noise

water. However, to do this we had to sample the height map in the water shader. See Figure 7.

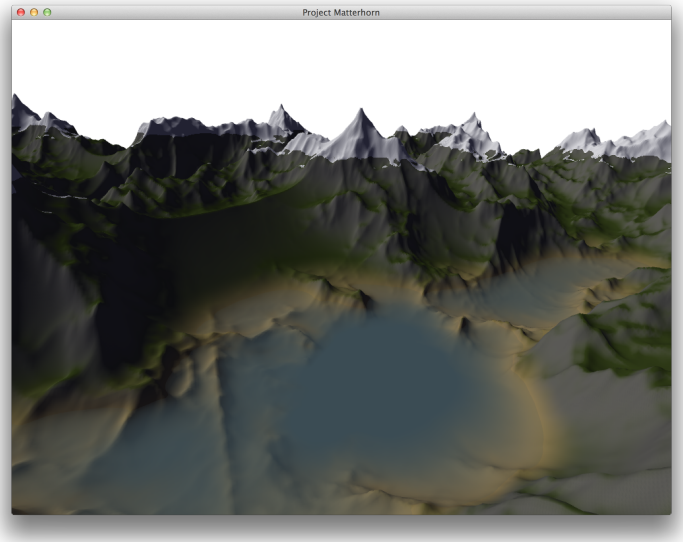

Figure 7: After implementing water transparency