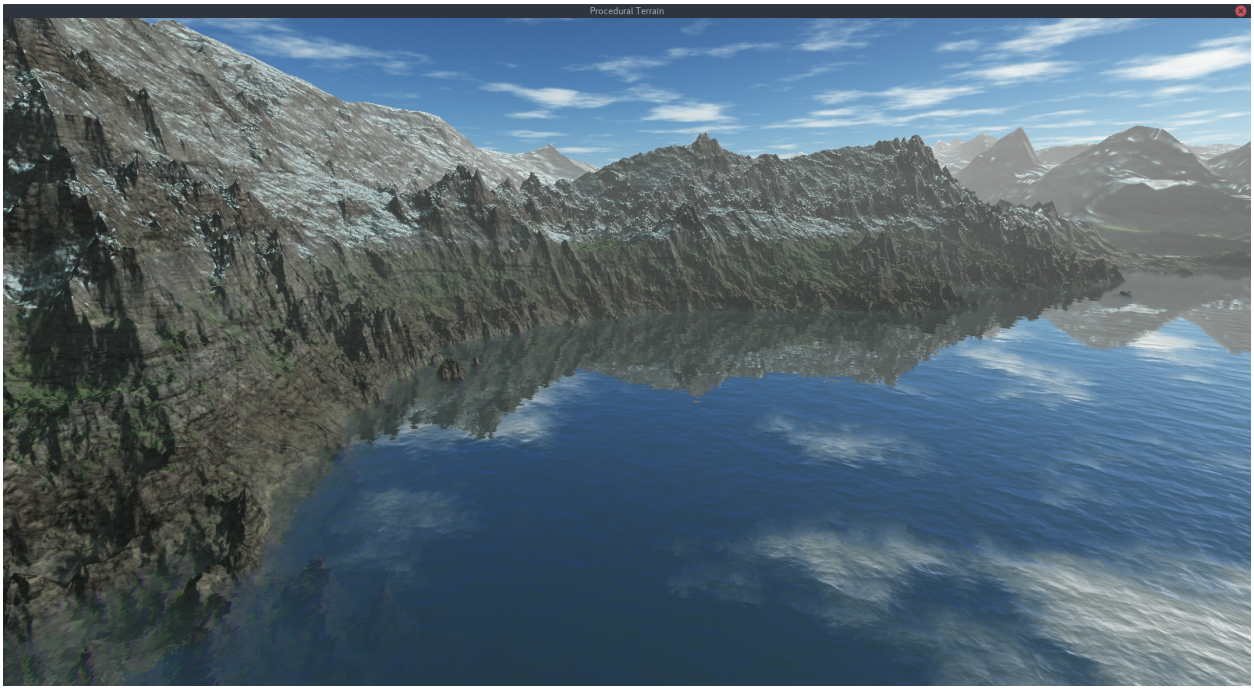
Report: Part 2

Grégoire Hirt
Marc Bickel
Raphaël Barman

May 2, 2017

1 State of the project

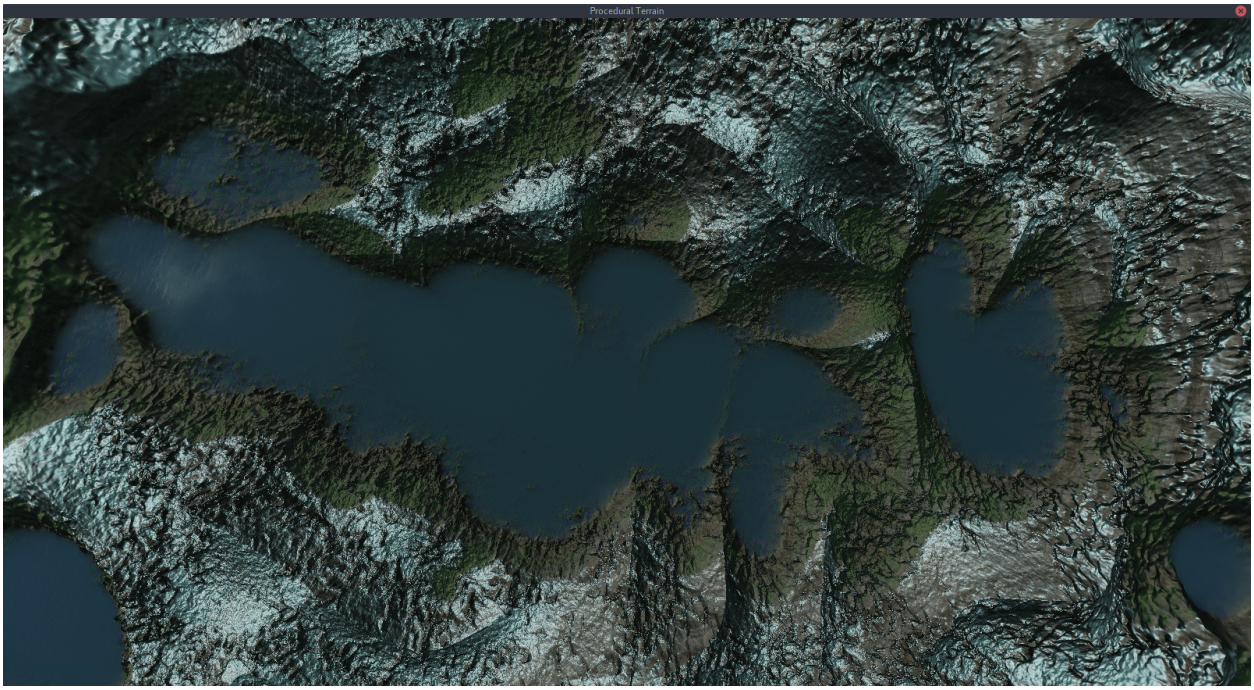
Since an image is better than a thousand words, here is one:



Current state of the project

What we implemented during this second part was:

- Textures
- Skybox
- Water
- Multithreaded noise generation



Top view

We will now explain in details the implementation of the previously mentioned parts.

1.1 Triplanar texturing

We updated the terrain fragment shaders to feature a decent phong lighting and triplanar texturing. This also include fragment-wise evaluation of the heightmap to add bump-mapping to the terrain. This allow to achieve fairly detailed landscape without pushing the vertex count to high. We then use a power of the dot product between the up-vector and the terrain normal in model space to compute the 'slope' of the ground. Based on this slope we then map either the flat surface texture or the sloppy surface texture (top or rock). The top texture also change depending on the altitude this is done by re-using the 1D texture with the colors at different height from the previous part and them mapping colors to the amount of each texture.

1.2 Water rendering

1.2.1 Reflective water

To achieve reflexion, we used the same technique as in one of the assignments, rendering the seen flipped upside-down by multiplying view matrix by a mirror matrix. We then use the resulting texture as a color for the water surface. The problem with this approach is that underwater terrain also gets rendered and seems to hover over water. To address this, we added a clip plane at the surface of the water to avoid underwater terrain to render in the mirror.

1.2.2 Full fresnel model water

The water was great this way but we wanted to have some more details. So we now render the terrain in a dedicated framebuffer instead of the window directly. This allow to use this back texture as a color for adding refractions to the water. The water color is then blended between reflection

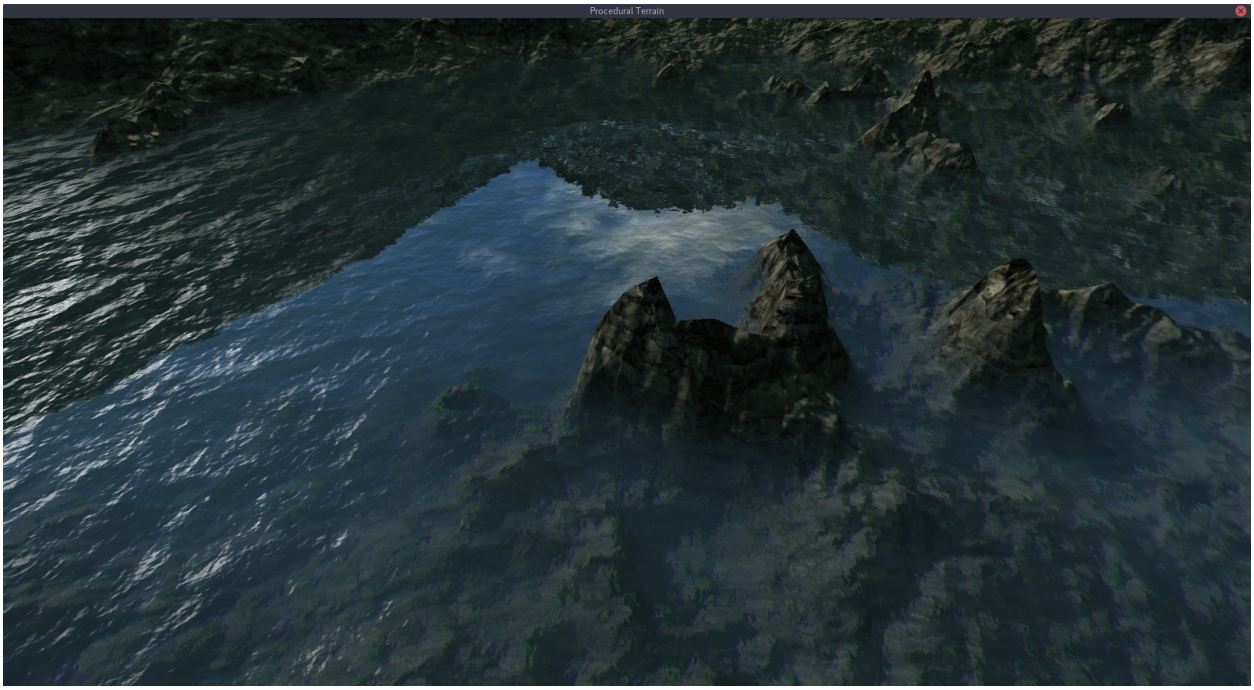
and refractions using an approximated fresnel model based on the normal of the water surface and the view angle. Another thing that we added is a water fog. Meaning that when we see the water from above, the luminosity of the refractions are decreasing with depth, and underwater becomes almost black from a certain threshold. Of course, deformation of the reflexion and refraction images are approximations since we have planar texture and not envirronement maps that would be less efficient. So we use the difference between flat reflected vector and distorded reflected vector as an offset in the texture lookup. Same goes for the refraction.
A small bonus feature is the chromatic abberation based on the water depth. Meaning red green and blue components don't get distorted exactly the same way and give this impression of colored blur.
Of course the water surface is given some specular factor to add some shine over the waves.

1.3 Skybox

The skybox was implemented in a quite canonic way. By loading 6 faces of a cube map in a CubeMap Texture and then using the normalized vertex coordinates in a zero centered cube to lookup the texture. We also disable depth writing to prevent the skybox, which is drawn first, to occlude the terrain. To keep the skybox centered on the camera we simply zero-out the last column of the view matrix when drawing the skybox. This cancels only translation and thus preserve rotation and scale (scale not needed but since we don't scale the view it does not matter).

1.4 Multithreaded noise generation

Since noise generation became a big part of the project, noise sampling was taking down the framerate when we were switching back and forth. At first, we implemented a job queue to process each chunk update on a different frame. This way the payload was distributed across many frames and we saw fps improvement, but this was not enough. So we decided to take the noise generation to a shared Opengl context that's running in another thread. This way we can generate textures without disturbing terrain rendering. The chunks then simply wait for future textures to be ready and cross-fade them with the previous one (that could be higher or lower resolution depending on the way we are moving the camera). This system gave some trouble to tune, because we didn't want to have the worker thread to busy-wait on an empty job queue, so we had to find the best heuristics to make the thread sleep.



Another view of the current state

2 Work distribution

2.1 Mandatory part

	Marc	Grégoire	Raphaël
Framebuffer from noise		✓	✓
Perlin noise	✓	✓	
FBM		✓	
Phong	✓		
Grid with height			✓
Coloring	✓	✓	✓

2.2 Extra part

	Marc	Grégoire	Raphaël
Simplex noise	✓	✓	
Ridged hybrid multifractal		✓	✓
Chunk system		✓	