Introduction to Computer Graphics
Project Part 1 – Terrain Generation

Stefan Lienhard, Minh Dang

Handout date: Tue Apr 1
Submission deadline: Mon Apr 14 (8pm)

Late submissions are not accepted

This is the first stage of the project. You will use procedural methods to generate a terrain. The terrain generation can be subdivded into two parts: first you have to generate a heightmap for a terrain on the GPU, and then you render it.

Figure 1 is an inspiring image that shows the powerful possibilities of procedural landscapes.

Figure 1: Procedural landscape generated by Iñigo Quílez.

General Hints

- Start early. Debugging OpenGL code can be a lengthy process. Tools such as gDEBugger can be helpful.
- Google is your friend. There are good tutorials online.
- Even though we already covered many OpenGL functions in the practicums and homeworks, you might still have to read up on additional OpenGL functionality by yourself. We recommend that you revisit the practicums if you feel uncomfortable using C++, OpenGL, and glsl.
- Read the full handout before starting, both steps are interleaved: you can't visualize the heightmap (step 1) without being to display (step 2). In practice it might make more sense to

start with the second step. Be sure to have a look at the figures below, especially Figure 2 presents a possible order of the implementation steps.

1 Height Map Generation

A heightmap is an image with only one color component. Imagine it as a topograpical map: the (x,y) coordinates span that map while the pixel value corresponds to the elevation at that point. We will use a texture to store the heightmap.

Before starting the main loop to render our scene to the screen, we will use a shader program to generate the heightmap (for example in the `init` function). This is done by rendering a full screen quad into an *framebuffer object* FBO (which has the heightmap texture attached to it)[1]. Since the quad covers the entire viewport, the fragment shader will get executed once for every pixel in the FBO. It will calulate the elevation for that pixel and write it to the heightmap texture.

1. Create a texture with only one color component.
2. Create an FBO with that texture attached to it.
3. Create a shader program for rendering to that FBO.
4. Create a *Perlin noise* function in the fragment shader[2]. The fragment shader will write the values into the heightmap texture.
5. Render a full screen quad (two triangles), reaching from [-1, -1, 0] to [1, 1, 0]. Don't forget to resize the viewport to the texture size for this to work correctly (and to reset it after heightmap generation).
6. Implement *fractal Brownian motion* (fBm)(see Figure 2e)[3].
7. Don't forget to unbind the FBO after the terrain generation, otherwise you keep rendering to it instead of the screen.

Hints

- If you follow the suggested tutorial on Perlin noise on the GPU, you will have to use several textures at once (known as multitexturing[4]) to store the gradient and permutation tables. Make sure to read up on that.
- Be extremely cautious about how you set up your textures and what texture formats you use[5]. Default OpenGL textures are also clamped to [0, 1]. You likely want to use unbounded floating point textures, e.g., `GL_R32F` for our heightmap. Make sure to pick the right format with the correct number of parameters.

2 Displaying the Terrain

In the second step we create a flat, regular triangle grid (see Figure 2a) that will represent the terrain. We will create two VBOs containing the vertices and indices for the triangles. Then a vertex shader can be used to change the height of the vertices according to the heightmap. We'll

[1]Practicum 6 used an FBO to render the shadow map. If you feel unfamiliar with FBOs, there is a good tutorial here.

[2]A good tutorial on how to do this can be found here: here. The code for the tutorial can also be found online, you will have to port it to glsl so that you can use it.

[3]A description can be found in Ken Musgrave's paper on fractal terrains.

[4]A small tutorial for multitexturing is here.

[5]More information about that can be found here and here.

create a second shader progam that is used for rendering the terrain to the screen in every iteration of the mainloop (build the shader program in `init` and use it for rendering in `display`).

1. Create a VBO for the triangle grid (at least 256x256 vertices).
2. Create a shader program for rendering the terrain.
3. The vertex shader samples the heightmap texture generated in step 1 and displaces the vertices according to the height value.
4. Implement diffuse shading of the terrain in the fragment shader. For this you have calculate the normal at the given position in the fragment shader. One way to do this is by using finite differences: you can find the gradient at the current pixel position by comparing the elevation with the elevation of the neighboring pixels.
5. Come up with some simple schema to color the terrain depending on the height (e.g., see Figure 2f). We'll look into how to properly texture and color the terrain in the second stage of the project.

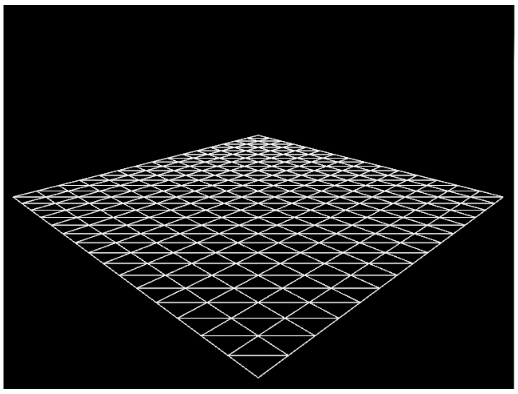
3 Advanced Topics

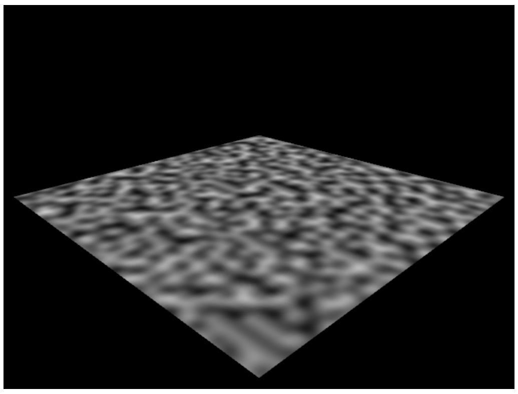
Here are some ideas (with estimated difficulty) of what you can implement to improve your grade. You are not limited to the suggestions here, you can also come up with your own extensions as long as you stay within the domain of procedural geometry generation.

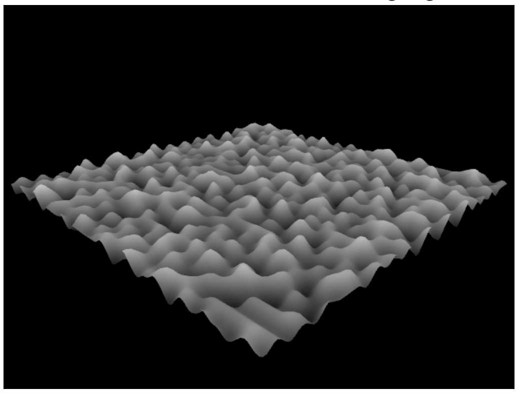
- **easy** – Use other combinations of noise functions to get more interesting terrains such as hybrid or ridged multifractals[6]. Some examples are in Figure 3.
- **easy** Implement and experiment with different basis noise functions such as Simplex or Worley noise.
- **medium** – Use tesselation shaders to do *level of detail* (LOD) rendering of the terrain[7].
- **medium** – Instead of using a flat domain, extend the noise function to 3D and map it to a sphere in order to render a procedural planet.
- **hard** – Make an infinite terrain that you can navigate through.
- **hardcore** – Use *L-systems* to populate your terrain with trees.
- Your own idea.

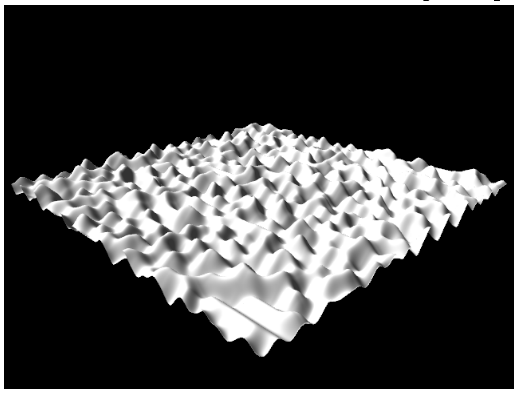
[6]In addition to Ken Musgrave's paper, also Ebert et al.'s Texturing & Modeling book might be useful. Beware that fractal terrain functions are extremely sensitive to the parameters. A correct implementation with bad parameters can look awefully wrong.

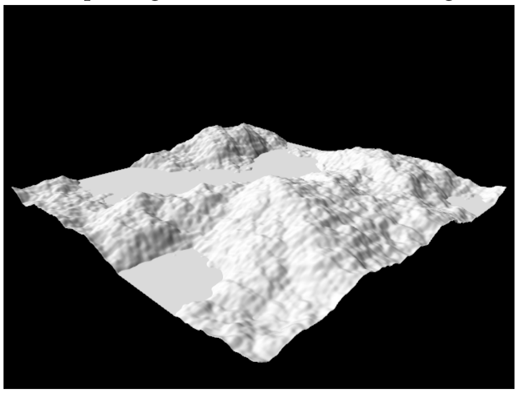
[7]Note that for this you need a GPU that supports at least OpenGL 4. Unfortunately the lab machines **don't** support that.


(a) Wire frame of a (16×16) triangle grid.


(b) Grid textured with a Perlin noise height map.


(c) Displacing the vertices (on 512 × 512 grid).


(d) Diffuse shading.


(e) Fractal Brownian motion (fBm).

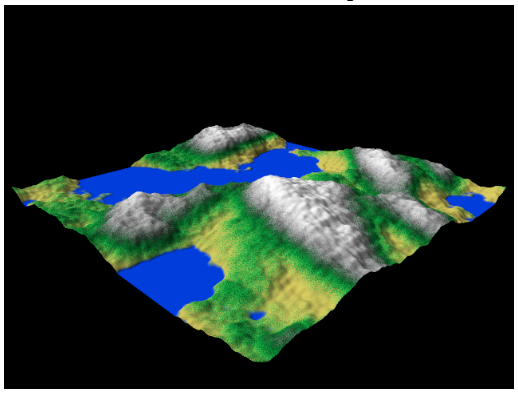
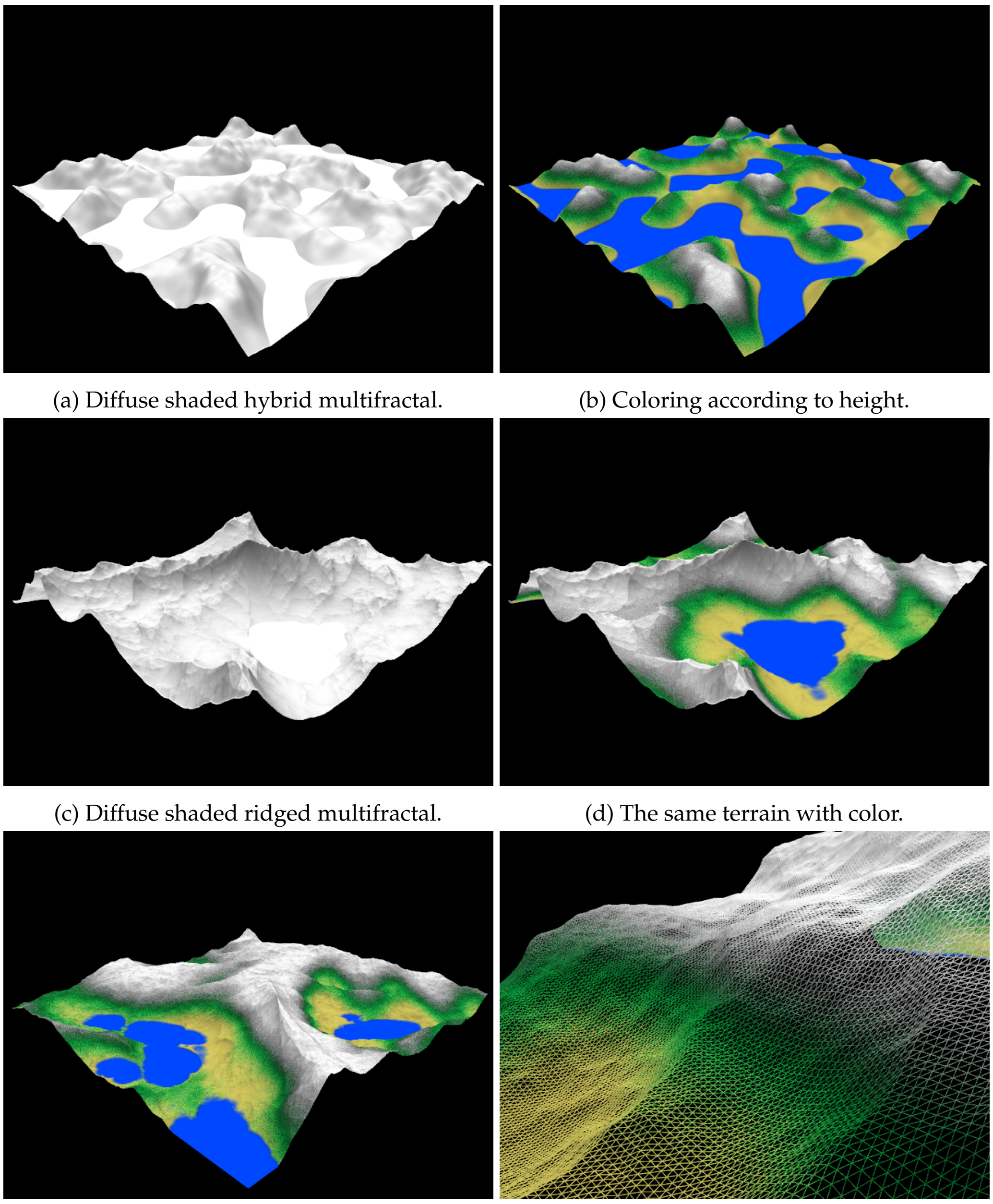

(f) Coloring according to height.

Figure 2: Triangle grid, Perlin noise, Vertical vertex displacement, diffuse shading, fBm, and coloring.



(a) Diffuse shaded hybrid multifractal.

(b) Coloring according to height.

(c) Diffuse shaded ridged multifractal.

(d) The same terrain with color.

(e) Another ridged multifractal.

(f) Wire frame close-up.

Figure 3: Advanced fractal terrains.