CSC305 Assignment Four
Due Date: 7 April 2018

March 2018

Figure 1: Example implementation with some advanced features

1 Introduction

You will use **procedural methods** to generate a virtual world. This goal can be subdivided into three parts:

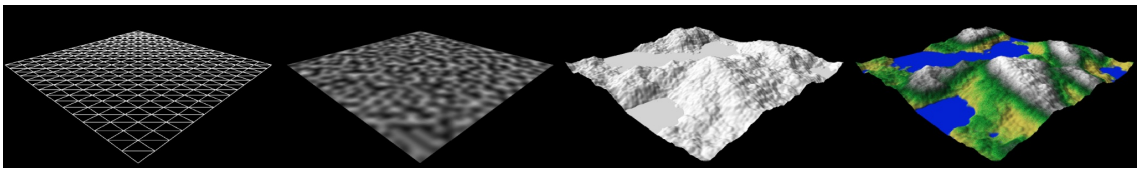
- geometry (mesh generation, procedural height generation)
- rendering (terrain shading, texturing)
- animation (camera animation)

Each of these parts are subdivided into basic and advanced tasks. Completion of the basic tasks will reward a grade of **80%**. Starter code has been provided to guide you towards completing the basics, see the **Getting Started** section below. Where an advanced feature replaces a basic feature, the basic feature will not be required (e.g. implementing a spherical world instead of a flat world). If you want to implement a feature not given below, just consult with the instructor or TA. Any nontrivial features will receive credit.

2 Geometry

2.1 Basic (35%)

- create a flat (z=0) triangular **mesh**. Do so using GL_ TRIANGLE_ STRIP , making use of GL_ PRIMITIVE_ RESTART
- implement Perlin noise on the CPU (see noise.h)

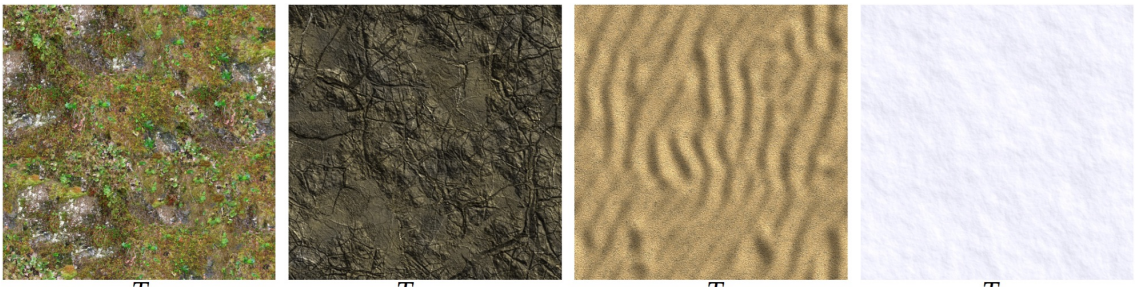


- generate a heightmap texture using **fBm** (see noise.h, and this **tutorial**)
- use the heightmap texture to **displace vertices** in the vertex shader

2.2 Advanced

- implement noise texture generation in the fragment shader 5%
- implement other noise functions to generate terrain (e.g. hybrid multifractal and ridged multifractal) 5%
- instead of generating a plane world, create a spherical world 10%
- create an infinite world (new tiles on demand, requires GPU noise) 15%
- use L-system to add trees to your terrain 15%
- use ImGUI to control parameters live 5%

3 Rendering



T_{grass} T_{rock} T_{sand} T_{snow}

3.1 Basic (35%)

- calculate surface normals, add diffuse and specular shading
- texture according to height and slope (e.g. snow will not deposit on steep slopes, grass won't grow at altitude)
- implement the skybox texture using OpenGLs cubemap textures

3.2 Advanced

- use the **normal map** texture (water.png) to represent waves. Make them translate over time. (5%)
- add a mirroring effect to the water; this is achieved by mirroring the camera position with respect to the water plane, render your scene in a framebuffer, and placing the texture back in a second step. You can also simulate refraction by blending the mirrored and non-mirrored images according to the incidence angle of your camera w.r.t. water (15%)
- use noise to distort the reflected image (5%)

4 Animation

4.1 Basic (10%)

- implement WASD camera controls

4.2 Advanced

- use a bezier curve to animate the camera path (5%)
- implement a FPS camera, camera height is determined by terrain height (5%)
- use a displacement map to animate waves; the map can be computed numerically as a sum of periodic functions varying over time (10%)
- use particles and billboards to animate 3D snow (10%)

5 Getting Started

The starter code is filled with **TODO** comments. The following order of completion is recommended:

- genTerrainMesh() in main.cpp
- drawTerrain() in main.cpp
- lerp(), perlin2D(), and fBm2DTexture in noise.h
- terrain_vshader.glsl
- drawSkybox() in main.cpp
- KeyEvent listener callback in main.cpp
- terrain_fshader.glsl