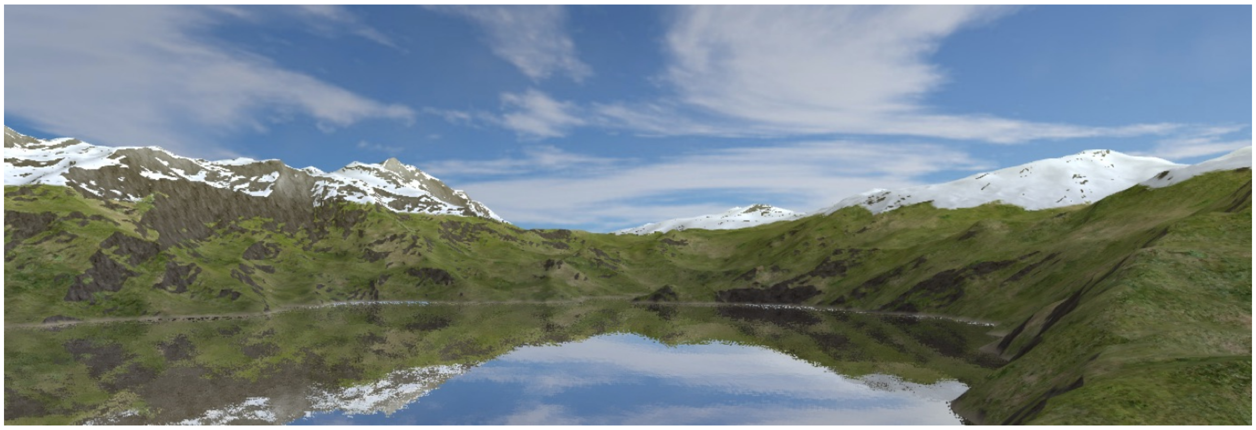
CSC305 Assignment Three
Due Date: 7 July 2017

Brian Wyvill

June 2017



1 Introduction

In this assignment you will use procedural methods to generate a virtual world. This goal can be subdivided into three parts:

- geometry (procedural generation on either CPU or GPU)
- rendering (terrain shading, texturing)
- animation (camera animation, etc..)

The sub-tasks in these three parts are subdivided into two categories: basic and advanced. Tasks in the Basic category will allow you to obtain 65% The advanced tasks can be used to improve your grade and difficulty is annotated in parenthesis beside each item. A real time world having a visual quality as shown in the teaser image above will award you 100%.

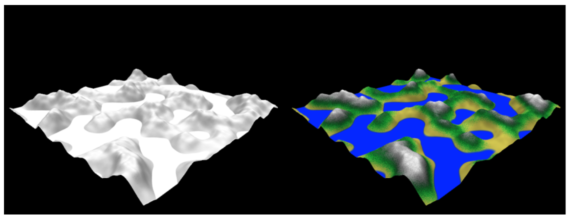
Many of the advanced tasks would require additional research to implement, and are provided as options for students who are interested. If you have other ideas for tasks you would like to try, please let us know!

You can find a video of a previous submission for this assignment showing some advanced features here: youtube.com/watch?v=WeAqQD6-GJs

Geometry

Basic

- create a triangular mesh for a quad on z=0 (compute a float[] on CPU, then upload vertex array to GPU). You cannot only load a mesh from file. Create the vertex and index buffer manually, and use GL_PRIMITIVE_RESTART to create strips of triangles (one per row).


(a) Hybrid Multifractal

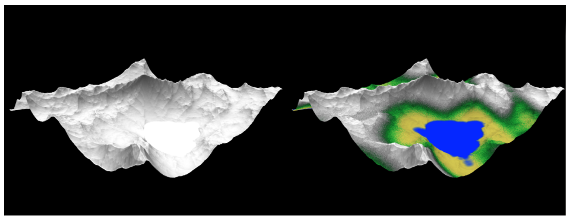

(b) Ridged Multifractal

Figure 1: Other noise combinations

- generate perlin noise on CPU and upload it to a texture
- generate a fBm texture
- setup ModelViewProjection matrices (C++) to display your 3D scene
- use your fBm texture as a displacement map for your mesh (vshader)
- color terrain according to height (fshader, use a 1D texture)

Be cautious about how you set up your textures and what texture formats you use. Default OpenGL textures are also clamped to [0, 1]. You likely want to use unbounded floating point textures, e.g., GLR32F for our height-map. Make sure to pick the right format with the correct number of parameters!

Advanced

- use ImGUI (github.com/ocornut/imgui) to control the parameters of your program (easy)
- use other combinations of noise functions to get more interesting terrains (easy)
- implement and experiment variations like Simplex Noise or Worley Noise (medium)
- use tensor product surfaces in combination with a noise function to produce smooth areas of terrain. (medium)
- use noise functions to generate clouds (and integrate participating media in fshader) (hard)
- compute the fBm noise on the GPU (fshader + framebuffer) and show how you can interactively change the parameters of the random generation (hard)
- perform erosion to create a more realistic heightmap (hard)
- use tessellation shaders to render in a LOD fashion (requires OpenGL4) (hard)
- make an infinite terrain (generate new tiles on demand, require GPU noise) (hard)
- use L-systems to add trees to your terrain (very hard)
- rather than using a flat domain, do everything on a sphere (solid noise), similar to the example video (very hard)

For these advanced tasks, this textbook might also come useful to you: Texturing and Modeling, Third Edition: A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics) 3rd Edition by David S. Ebert (Author), F. Kenton Musgrave (Author), Darwyn Peachey (Author), Ken Perlin (Author), Steve Worley (Author)

Rendering

Basic

- add diffuse shading to your terrain (you have to compute per fragment normal by finite differences on the texture that displaced the terrain)
- use the tile-able textures provided (see the github wiki) or your own to texture your terrain. In the fshader you can use the normal of a fragment (slope of terrain) and its height to decide which textures to blend. For example, snow does not deposit on very steep slopes, and happens only at a certain height.
- surround your scene with a cube, and texture this cube to color the sky of your scene.

Advanced

- use an OpenGL CubeMap to texture the sky and get rid of artifacts caused by discontinuities in the UV parameterization. (easy)
- use a normal map texture to represent waves on your water; see this texture. You can overlap multiple scaled copies of this texture and translated them over time to emulate a water effect (easy)
- add the mirroring effect of water (screen-space reflections) to your scene; 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. (medium)
- simulate the fact that reflections are affected by water movement by distorting the reflected image with a noise function; this is achieved with dudv-maps, also called uv displacement maps. (medium)
- add dynamic shadows to your scene by rendering a shadow map from the perspective of your light source (hard)
- use a fragment shader to implement some post-processing effects such as depth-of-field or SSAO (Screen Space Ambient Occlusion) (hard)
- add atmospheric scattering so that the sky color changes according to the position of the sun, and terrain fades according to distance. (hard)

Animation

Basic

- implement a controllable moving camera (WASD control forward/back and side-to-side, while mouse controls direction)
- make the sun in your scene move through the sky in a circular arc

Advanced

- draw a 3D bezier curve in your scene (easy)
- implement an FPS camera (Z-position of camera is determined by local terrain height) (easy)
- use a bezier curve to animate the camera path (in the lookAt function, evaluate one bezier curve to know the position of your camera, and another to know what you are looking at) (medium)
- parametrize your camera path with arc-length (i.e. uniform speed) and use ease in/out to avoid abrupt starts/stops; concatenate multiple curves to show off your virtual world. (medium)

- use a displacement map to offset the position of vertices for the water plane (over time). The displacement can be computed numerically, as waves are nothing but a summation of periodic functions. (medium)
- make it snow! use particles and billboards to animate 3D snow (hard)
- camera motion causes motion blur! implement screen-space motion blur (hard)

2 Getting started

You may use the assignment3 folder from the icg git repository as your starting point. Post any related questions on the GitHub issue tracker... and help each other!!

For more information on implementing some of the advanced tasks, you can check out the tutorials on the wiki resources page: github.com/drebain/icg/wiki/Resources

Remember that you must implement all graded features in your assignment submission yourself, and any code obtained elsewhere must be properly referenced.