Santiago Bonada, 100525252

Advanced Graphics Assignment One

Terrain Specification:
Terrain is specified through the following format within a text file:
Size in OpenGL space
Final Resolution → N
Initial Resolution --> M
Beginning H value for the fractal algorithm
MxM height values to seed the final NxN grid

Terrain Generation:
The Diamond-Square algorithm covered in class is used to generate the terrain. First we set aside memory for the final grid and read in the seed values into their correct positions. After this we perform the iterative form of the Diamond-Square algorithm to generate new height points, using the H value from the terrain file to control the random number distribution (and halving it every iteration). std::normal_distribution(0.0,1.0) from the random header in C++ was used (as by specification in the assignment) to get normally_distributed random values. After generating the terrain we also store the maximum and minimum height values which is later used in the fragment shader. Terrain generation all takes place within the terproc.cpp.

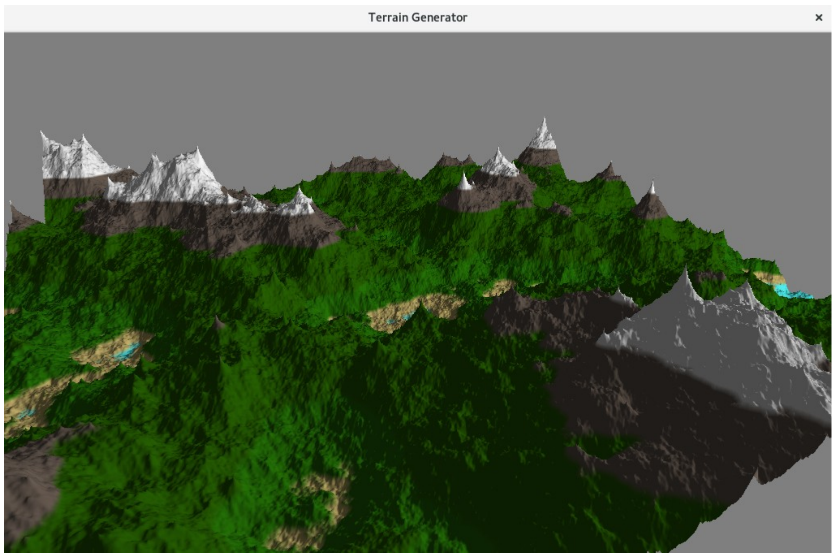
Texture Generation:
Next we generate a 1-Dimensional texture to texture the terrain, where height maps to a colour in the texture. Generation is done by specifying colours as normalized floating point values within glm::vec4's, as well as specifying the height ranges those colours take up. The colours and ranges in my generator are currently hardcoded in my main function, but they can easily be specified within a text file. The texture is then simply the colour values set over the ranges. Part of the texture creation is implemented in Texture1D.cpp.

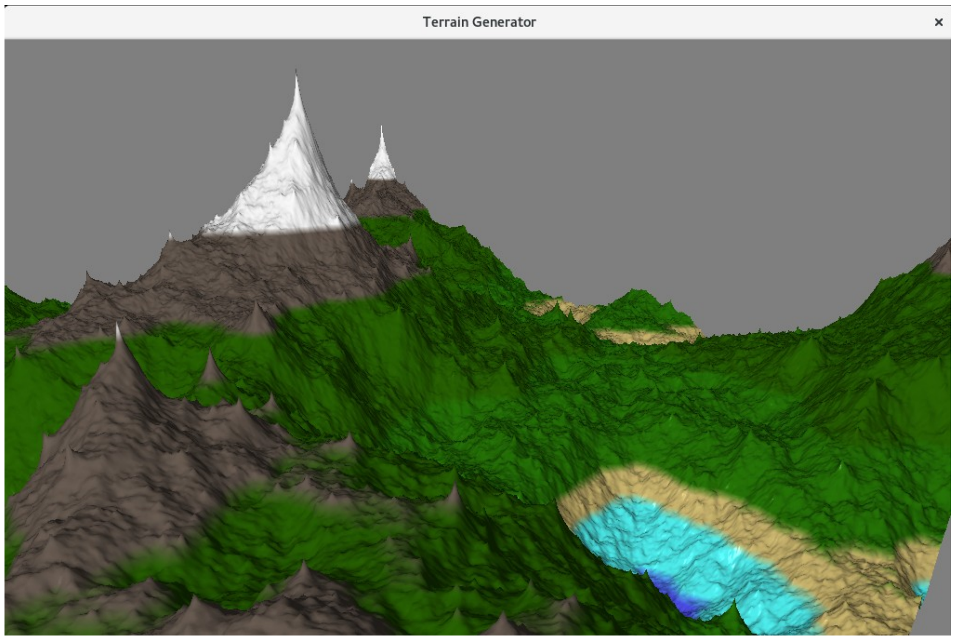
Vertices,Indices,Normals,etc:
There's a class which encapsulates a VAO for Terrains called TerrainVAO declared in TerrainVAO.hpp and implemented in TerrainVAO.cpp. It's pretty standard stuff, we parse the terrain generated and create vertices, normals, and link the triangles with indices and pass them to OpenGL through buffers. The texture is also put into the buffer, but no additional processing is required for it after the functions in Texture1D.cpp. We clamp the texture so that the lower and upper bound colours in the texture just continue (it makes sense in the context of Terrain).

Display and Shaders:
Finally, we display the terrain. The CPU code to do so does not differ much from the code in the labs, and is defined within gl_helper.cpp (at the time envisioned as containing helper functions but the file should really just be like gl_funcs.cpp or something but that doesn't matter). The vertex shader is essentially a copy paste from the labs, specifically the ones that leave phong shading to the fragment shader. The fragment shader is also pretty much the same as the ones from the lab, but here we get the base colour from the 1-dimensional texture indexed with the height passed from the vertex shader. The minimum/maximum height values of the terrain mentioned earlier are used to normalize the height of the vertex to lie between (0,1) so that the texture can be applied agnostic to the scale of the terrain, so there's also an additional function in the frag shader to normalize values within ranges. The shaders are named ter.vs and ter.fs respectively.

Camera:
Finally, the camera code. Not actually my own, I seeked out the modern OpenGL tutorials and discovered that the camera code was distributed under a free as in freedom license. The camera code is contained within Camera.hpp and Camera.cpp along with the license statement. I did not change the code within the implementation files, but in my display func I rotate the camera so that the coordinate system has Z in the up-direction (the camera is implemented where Y is the up-direction. The camera is controlled with WASD, with Z and X to raise and lower it respectively. Dragging with the mouse changes the angle of the camera in the direction of movment. To do so the mouse is stuck at the center of the window and the delta is calculated when moving the mouse; I was unable to find out how to query if the window had focus in glut so the mouse will stay there in the center, so I also threw in the ESC key closing the window/program.

A pair of screenshots of the final output:


Terrain Generator
×


Terrain Generator
×