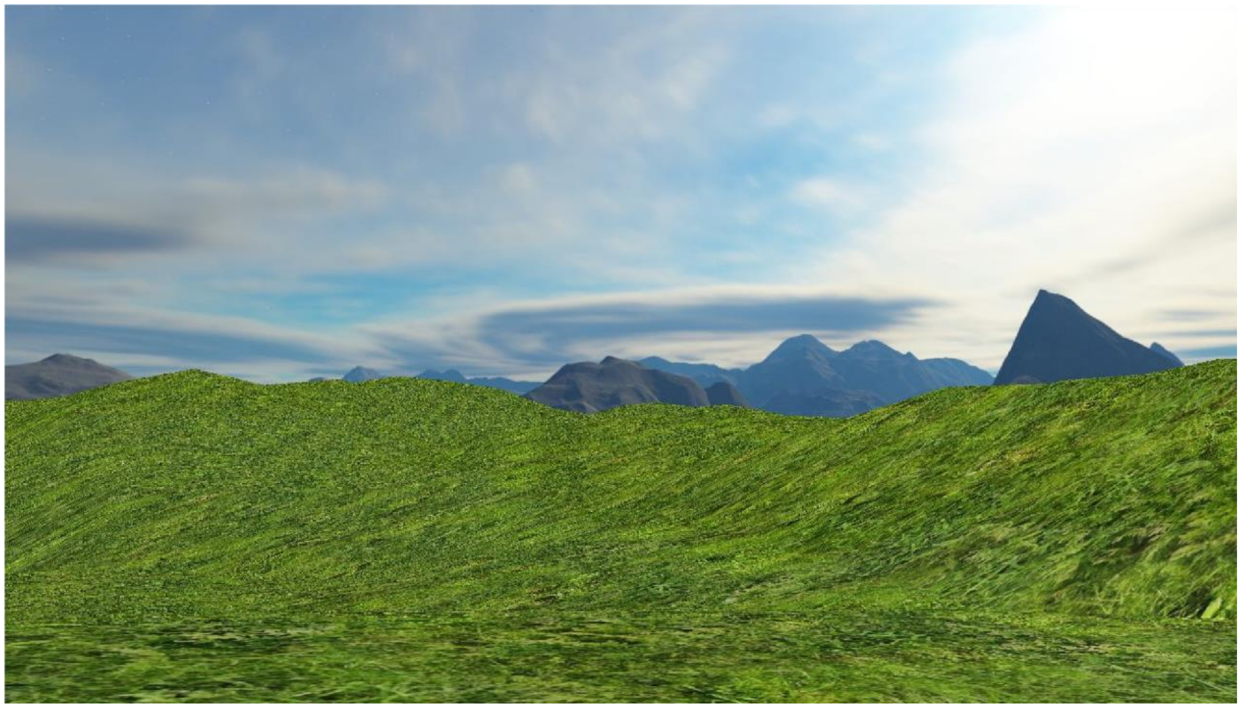


World-101110011

Final report

COMP 371 - Team # 13

13.04.2017

—

Name(s): Tarik Abou-Saddik,

Justin Velicogna, Sami Boubaker,

Vsevolod (Seva) Ivanov, Eric Morgan

Overview

The purpose of this application is to explore computer graphics using OpenGL with the C++ programming language. More precisely, we are interested in the creation process of a procedurally generated world. We decided to go with the idea of recreating the nature in a form of a forest. For the world, we attempted to underline the artificial nature of the computer generated world. Depending on the time constraints, this forest will have various “objects” on its terrain like trees, plants, bushes etc.

This world will present an artistic simulation of a forest to be explored by its user. It will be presented as an extendable base structure. World-101110011 will serve as an inspirational environment relying on a visual representation of an artificial world.

Project URL: https://github.com/sevaivanov/world-101110011

Goals

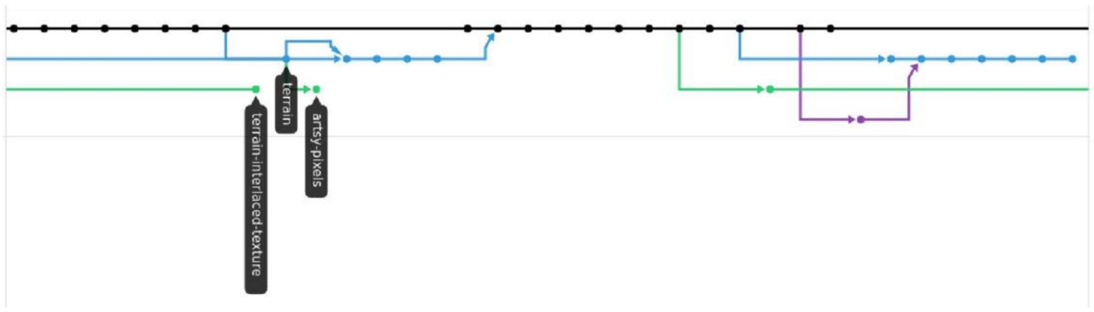
1. Create a procedural world with a terrain.
2. Load procedurally generated objects into that world.
3. Handle collisions and allow camera motion.
4. Add shadows, weather, etc.

World // Architecture

The world is procedural due to its unique representation on each creation.

At first, I started by creating a basic Shader, Camera and Window classes. From the point, I used a basic Makefile with targets to compile everything for various platforms. To simplify the merging, we decided to use an Abstract Mesh class to uniformize the rendering control and techniques across different elements of our world.

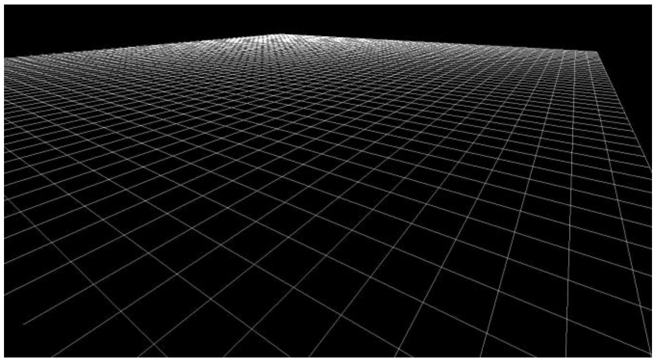
We decided to use Github revision system to help us to collaborate smoothly, avoid regression and snapshot various patches / features using different branches :



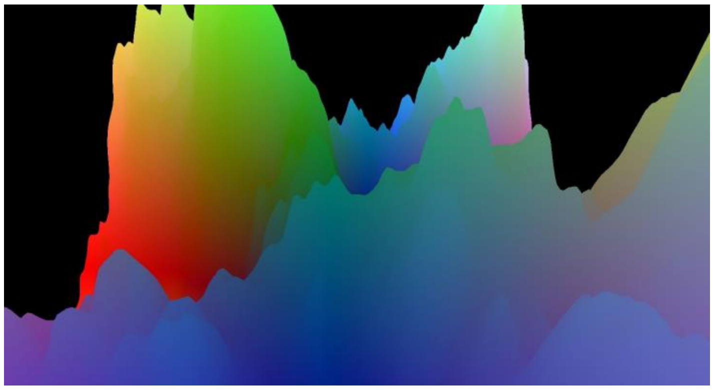
Terrain

The Terrain is procedural due to its usage of Height Map with customly written noise functions. Moreover, it uses recursion to generate its grid surface by wrapping the Height Map generator.

At first, I created a simple grid to explore a very basic layout of the cells using the dimensions of width and height of the world size input. This helped me to grasp the trivial flat terrain structure stretching along the x and z axes.

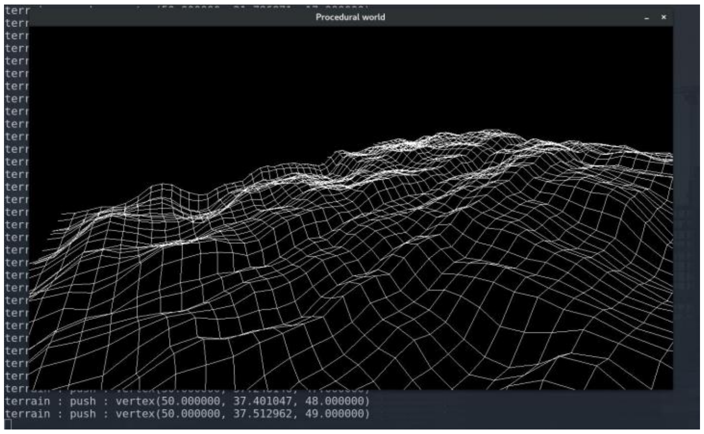


From the point, I started reading about the Height Maps and their implementation. A recurring element was the usage of the noise functions such as Perlin, Simplex and Cellular. It seemed that in order to create a realistic terrain one would need to blend many of those functions together to create a realistic irregularity of the real-world terrain. The latter brought me exploring various noise libraries.



In particular, the libnoise one seemed very promising. It was very simple to integrate and it allowed me to create a very realistic mountains. However, we wanted something personalized as a more flat terrain to allow trees and this is where everything got very complicated. In fact, this library is perfect to wrap various noise functions and their blending but the extension of those requires their utils which are written in C and are quite hard to understand and even more to personalize. I think it is a great framework which even allows you to create planets but for our project and the learning purposes it seemed more reasonable to implement basic noise functions myself.

I went with a creation of HeightMap located in the TerrainHeight class which is based on research shamelessly copied into *docs/research/PerlinNoise.pdf*. Basically, I needed to code a Perlin noise alike math functions which should be in 2D because since we are feeding an x and z values to have as a return an y elevation value.



```
terrain : push : vertex(50.000000, 37.401047, 48.000000)
terrain : push : vertex(50.000000, 37.512962, 49.000000)
```

After quite a learning curve to understand frequencies, octaves, etc., the TerrainHeight class produced an interesting looking terrain. Of course, I could have copied this code directly into the terrain vertex shader but considering the impossibility of printing values and its early stage, I decided to leave this step for the stage when everything works properly.

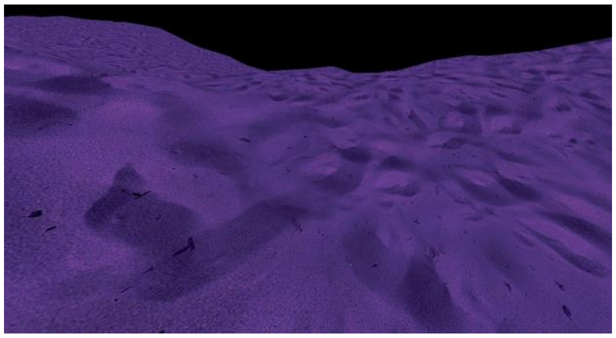
One thing that I didn't completely understand is to how properly "clamp" the output y values to stay between the reasonable "min < y < max" range without breaking my noise functions.

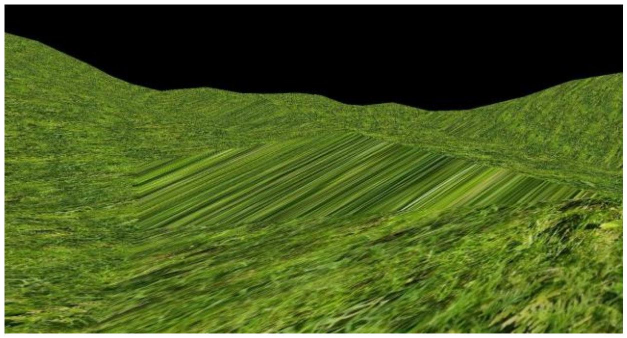
The way I understand it : to generate an infinite terrain in every directions, we have to create an illusion of movement by positioning the camera at the center of the initial terrain and then, advancing it in the proper direction by regenerating one row of a terrain on each step forward using an offset along x and z axes. However, the previous "clamping" issues to avoid excessive y heights got me stalled. Nevertheless, I programmed the architecture and logic of the advancing the terrain instead of moving the Camera forward in the world :

https://www.instagram.com/p/BS4JZwngLQO/

From this point due to lack of time, I decided to prioritise the compilation of the other world parts and the overall aesthetics of the terrain. Therefore, I left this Work in Progress under the "terrain-infinite" branch for future development.

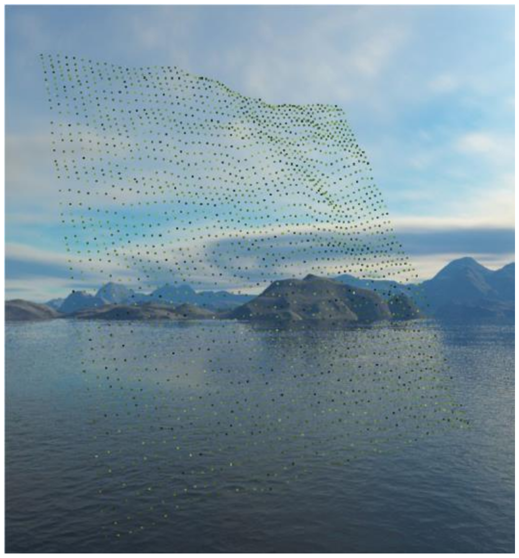
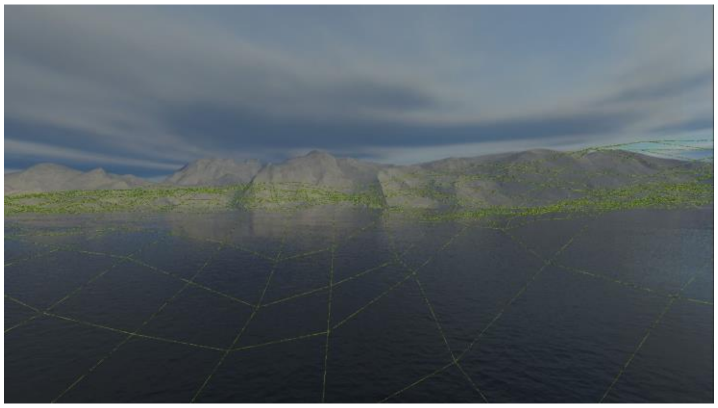
To enhance the aesthetics, I implemented textures and then I tried various looks:





It led me to discover a little anomaly on the RHS that I will solve in the foreseeable future.

Finally, I added support for multiple rendering modes such as points and lines using the Abstract Mesh implementation along with specific generation of the vertices indices using the EBO buffers to refer to the real vertices found in the VAO buffer :

Note the fascinating Window logo reproduction on the left hand side.

Ambiance

To stimulate the user's senses and re-create an illusion of the actual forest environment, a background soundscape was added using the SFML sound engine. Other engines such irrKlang were tested but SFML had an active cross platform support which is more promising in terms of support and out-of-the-box compatibility.

Skybox

…

Forest

…

Weather

…

Shadow

Shadows are added to tree objects. These shadows are created using a subclass of the GLobject class which creates a clone of a given objects which is thenand flattened it and changed it's color to black (Shadows via ray tracing were not used due to the large computational time required to calculate intersections). Ordinarily, the class would take a light source and a slope, and project the light source onto the slope, however, due to an error in the ray-line intersection function, the shadow is currently projected straight downwards onto a flat slope (however the architecture is still there, provided the ray-line intersection function is fixed.)

Documentation

The three documentation resources are:

- Doxygen code documentation: helps the understanding of the code -

  README : containing the Roadmap of what is done and not

  finished - docs/ folder: containing the resources such as:

- Help markdown file
- Process screenshots
- Research - Reports

Authors

Student ID	Name	Parts
40004286	Vsevolod (Seva) Ivanov	Project architecture, Compiling on Unix, Floating camera, Procedural terrain, Ambiance sound, Documentation
40005294	Justin Velicogna	Shadows
26417404	Sami Boubaker	Procedurally generated Objects, documentation
27518722	Tarik Abou-Saddik	Skybox, (lighting?), documentation.
26863426	Eric Morgan	Weather and reflections

Notes:

- We took Justin on board following the March 30th lecture.
- Details on the implemented parts can be found in the README > Roadmap.