Final Project | Team 7

A magical Journey Through a Procedurally Generated Forest

Johnny Mak	40002140
Helen Tam	27845537
Kevin Ye	40000926
Amal Zemmouri	27384335

COMP 371 - Computer Graphics

C. Poullis & S. Mudur

Concordia University

Winter 2017

April 18th 2017

1) Project Objectives

The objective of our project is to create a fully procedurally generated world, and to fully understand procedural generation. We decided on creating a magical forest where a user can control the camera to navigate through the world. We chose to generate a forest because it was very simple to visualize how the world would be. Everything that appears in our world is procedurally generated: maple/pine trees, bushes, rocks, fireflies, clouds, terrain height and texture. The only two things that aren't procedurally generated on each run are the skybox and the sunlight. That decision was made because those assets only appear once in our world.

2) Interests and Motivation

Both project options seemed very interesting but we decided to work on a procedurally generated world due to the experience and knowledge it will give us. Most game engines nowadays are able to provide all the functionality required to load and move an object. On the other hand, the knowledge of procedural generation cannot be done by game engines, hence we have chose this option to expand our knowledge in video game development.

Another reason why we chose procedural generation is because of the abstractness of the topic. Unlike loading objects, we have to create our own assets in real-time when we run our program. This forced us to make rules to prevent our generation from creating unwanted results.

3) Objective Accomplishment

For the generation of assets, we used predefined shapes that are scaled and transformed across the terrain to give more variety. Although the topology is the same, every shape of the same class has a different geometry. To achieve this, randomly generated values are used.

Bushes and rocks are created from the same class, which uses a predefined icosahedron that is scaled and translated according to our needs. To vary each bush and rock, the randomly generated values take into account the desired range for scaling and for translating along our terrain.

Trees come in two varieties, maple trees and pine trees. Each maple tree consists of an icosahedron as the tree top and a tree trunk, whereas a pine tree is made of a pine tree top and a trunk. Both pine tree tops and tree trunks are created by rotating polylines, similarly to our first assignment. Trees also scale and translate with randomly generated values, but these values are generally much bigger so as to not have a tree the size of a pebble.

Clouds are made of three icosahedrons. The generation of clouds follows the same concept as any asset mentioned above.

As we chose to aim for a low polygon art style, normals of each face are computed within each object and are passed into the vertex shader with flat interpolation. This creates the flat shading effect we desired.

To fit with the “magical journey” part of our project, we made extensive use of various types of lighting to achieve magical effects. A general point light is used to light up the terrain. Some smaller orange point lights make up the sunset effect and even smaller light blue point lights act as fireflies in the forest. Little cubes are generated to represent these fireflies. The cube and its according light have the same position, make it seem as though the firefly cube was glowing. Each light is calculated with an attenuation, an ambient, and a diffuse component. Specularity was not needed as it clashed with the visuals.

In addition, we used a grid as the base of our terrain. The terrain is the very first thing that gets generated in our world because it allows every other object to have a relative position by using the vertices on the grid as an anchor point. We set a value

that defines the cell size of the grid so that when we create our heights, we can maintain the low polygon look. Having the vertices too close to each other will create a highly smooth and detailed terrain since the height generation (See below) only makes slight changes to the adjacent vertices height. Hence, a highly detailed terrain will go against our low polygon theme. Our grid (terrain) has all the required functions to return the necessary information about that specific point. (e.g height value, color, texture coordinate, etc.)

For the height generation, we used the idea behind the perlin noise which is a type of gradient noise developed by Ken Perlin. It is widely used in computer graphics as it allows procedural generation of texture. The Perlin noise algorithm is fairly simple; you need to define the grid, find the gradient vector of each node, and interpolate these nodes. The reason behind why we chose this algorithm instead of the random algorithm is that the perlin noise algorithm has a pattern in the randomness they generate while as a random algorithm would generate totally random heights and produce an uneven terrain.

Having this Perlin Noise as a base, we were able to use its algorithm to assign a y-value to every vertex on the grid to create a random plain. With the algorithm, we are also able to control the amplitude of the terrain, where higher amplitude would result in larger hills and holes, while smaller amplitude would result in flatter plains.

4) What we learned

During assets generation, we learned how to limit the positioning of each object with respect to the terrain size by also taking into consideration the numerous factors that we used such as unit and scaling values. Fitting tree tops onto trunks was a big hurdle to go over as they are separate entities. In addition, they needed to be manipulated as a single entity for translations along the terrain. Setting their height was only a slight problem since we have a function that returns the changed height of the terrain due to Perlin Noise. We also learned about different types of lights that we

could use in our scene, and how they can be manipulated in order to create the effect that we want.

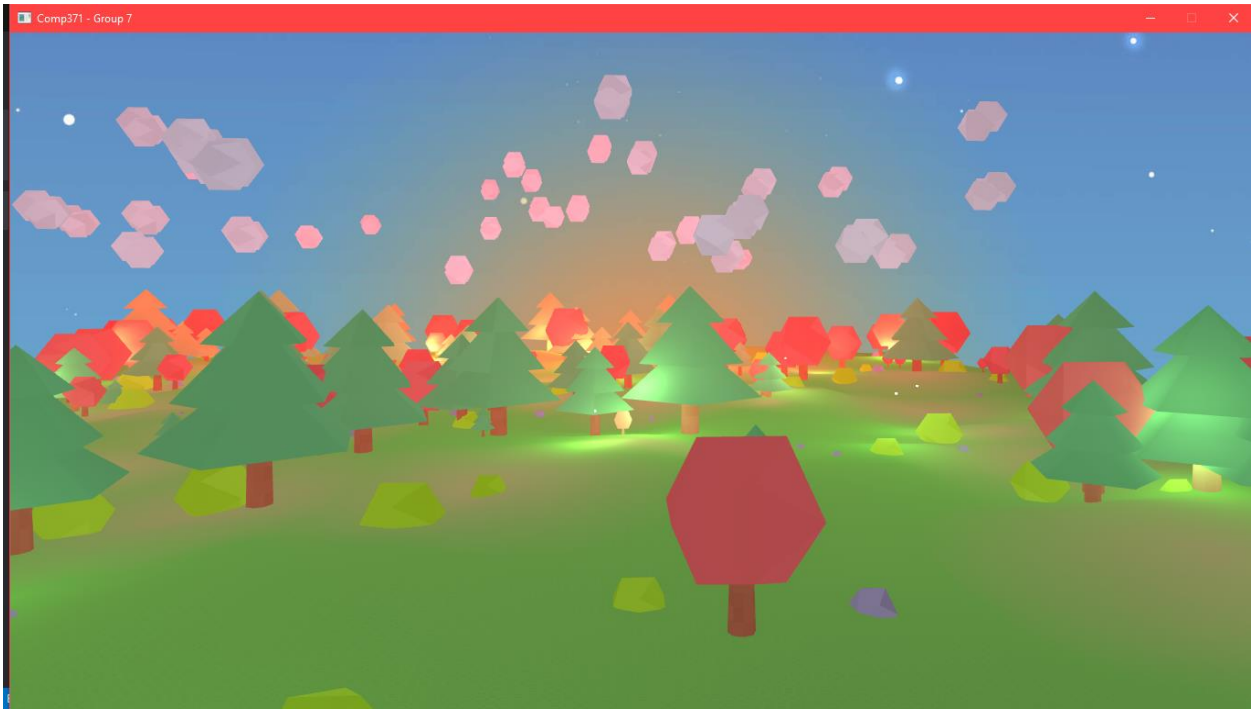
During the generation of heights, we learned how to “control” random values to a certain degree which allowed using them effectively to create the terrain we envisioned.

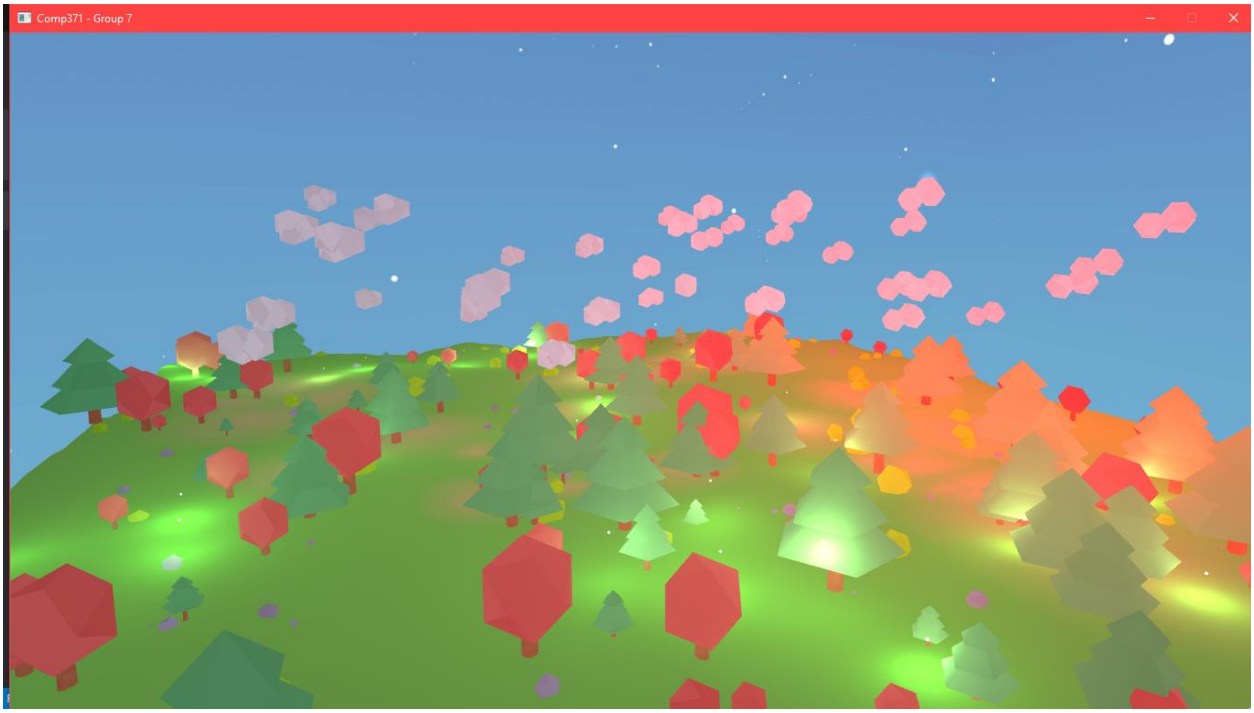
Even though we were not able to implement camera collision because of time constraint, we learned how to calculate the camera’s height position using the Barycentric algorithm and checking terrain’s height for collision.

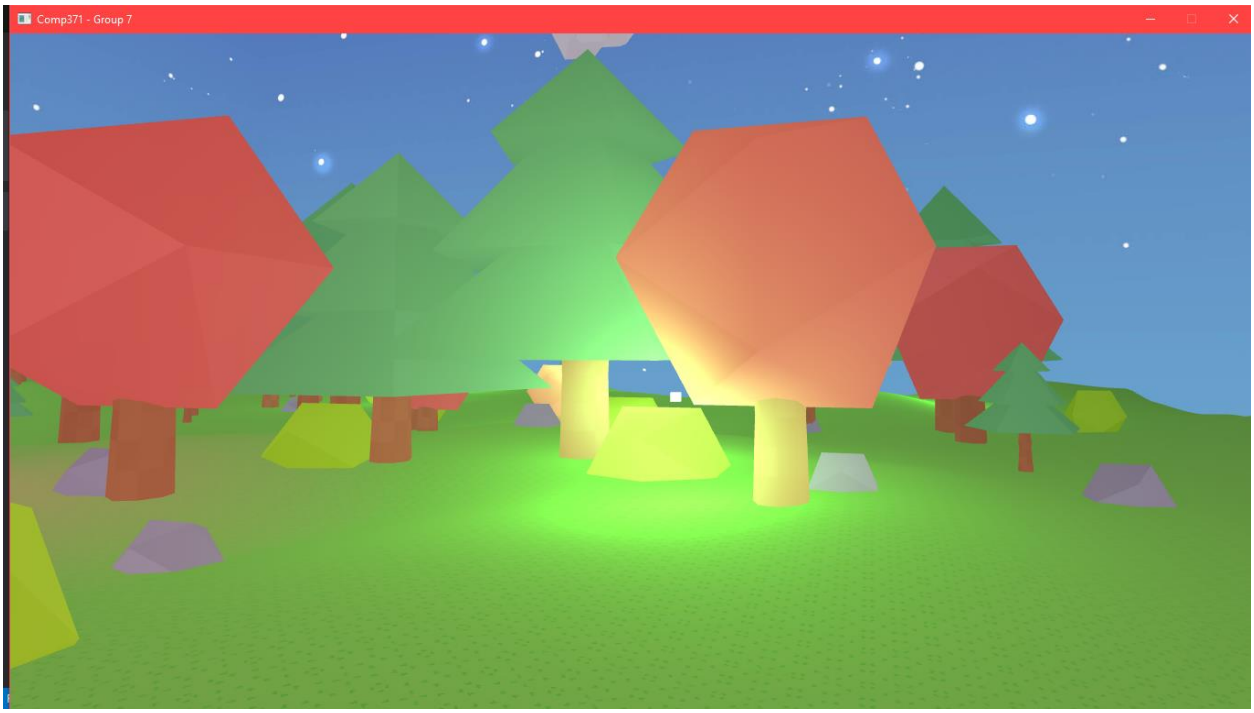
During the skybox creation, we learned how to use a different kind of texturing called Cubemaps. Although similar to 2D textures, cubemaps use 6 faces composed of 2D textures. We learned the basics of cubemaps but wanted to further extend its functionality. We came up with the idea to use this concept in order to texture the complex shapes such as the icosahedrons. The icosahedrons currently only have a color values assigned to it but thanks to the lighting, it has a layer of shading making it the shape more noticeable. Due to time constraints, we were not able to implement this idea, but if we had to do it in the future, we would definitely deploy this way of texturing.

In addition, we referred to many online sources and tutorials to deepen our knowledge on OpenGL. We have learned that the concepts covered in class are merely just the surface of it and there are still many built in functionalities that we have not seen yet which can provide additional features. For example, we can enable OpenGL to do Multisample Anti-Aliasing which forces the program to keep four different color values for one specific pixel. The rasterizer will then calculate and display the average of these colors, creating a much smoother image with reduced aliasing.

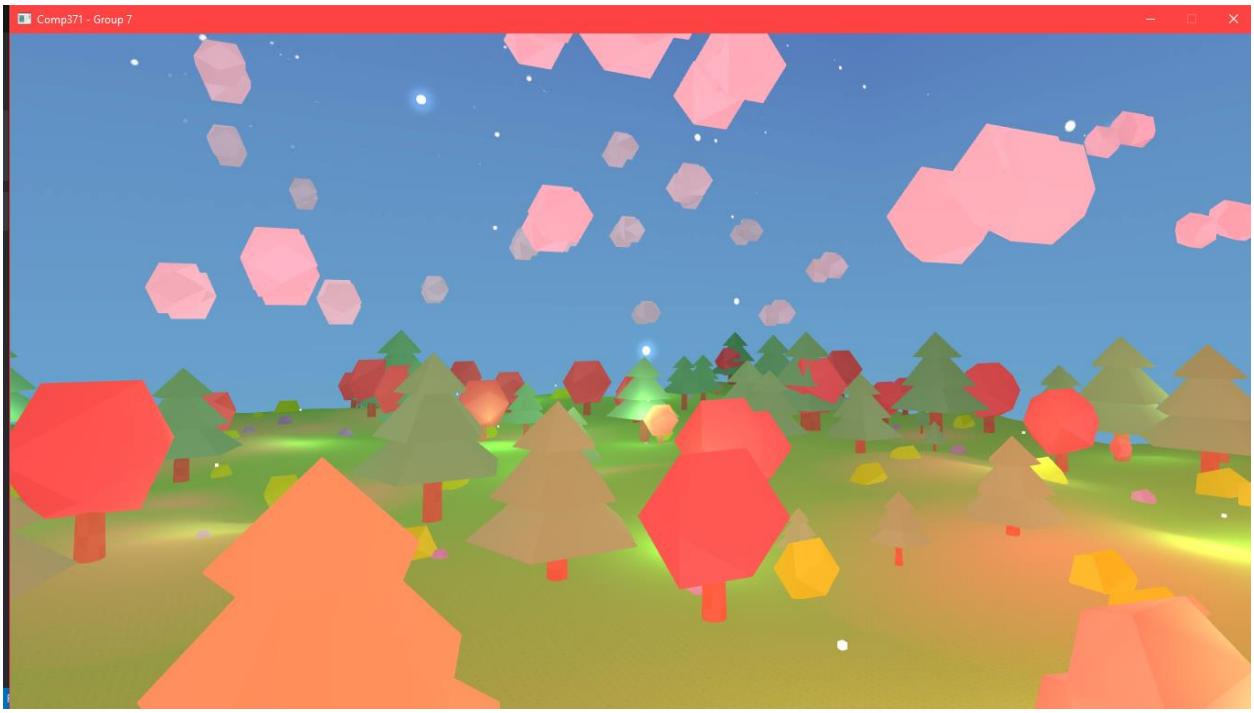
5) Screenshots






Comp371 - Group 7


Comp371 - Group 7



6) References and Resources

https://learnopengl.com/

https://www.youtube.com/watch?v=-kbal7aGUpk&list=PLRIWtICgwaX0u7Rf9zkZhLoLuZVfUksDP&index=17

http://stackoverflow.com/questions/14592925/opengl-multitexturing-and-blending

https://www.khronos.org/opengl/wiki/Multitexture_with_GLSL

http://vterrain.org/Textures/spherical.html

https://schneide.wordpress.com/2016/07/15/generating-an-icosphere-in-c/

https://learnopengl.com/#!Lighting/Multiple-lights
http://flafla2.github.io/2014/08/09/perlinnoise.html

http://libnoise.sourceforge.net/tutorials/tutorial4.html#persistence

https://www.youtube.com/watch?v=qChQrNWU9Xw

https://www.youtube.com/watch?v=IDkO8YT04bE&t

https://www.youtube.com/watch?v=AU98nWv6JWo

https://www.youtube.com/watch?v=MJ3bvCkHJtE&t

https://www.youtube.com/watch?v=Or19ilef4wE&t