PROCEDUAL GENERATION REPORT

By Taylor Ring

INTRODUCTION

For this assignment, we were asked to create a program that would generate procedurally generated worlds with features at runtime. these programs and algorithms provide numerous benefits to the entertainment industry, like creating a unique world for the player to run around or increasing the quality of an element. Along with this, we should also manipulate the scene with various properties, like materials/lighting and camera control.

ELEMENTS

SKYBOX

A Skybox is a graphical object that surrounds the player and the world. Its main purpose is to fill the void of the scene with textures. It is also used as an illusion to the player to show that the world is infinite. There is another object that does the same effect called a Skydome, which uses a sphere to surround the objects.

The process of creating a skybox is the same as creating a box in OpenGL. This means that vertex data must be calculated to form the shape of the box, and then it's sent using Vertex array objects (VAO) and vertex buffer objects (VBO). Finally, to "see" the cube map, glDrawArrays is used to tell OpenGL what to draw and how much of it to draw.

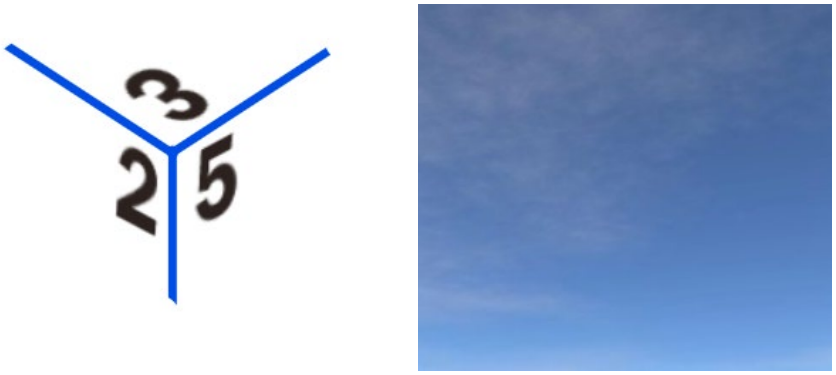

Figure 1: The skybox is made up of different panels which are placed to create an endless world.

TERRAIN

There are many algorithms and structures that offer different ways to create terrain. For the project, the diamond-square algorithm is used. This is done in three steps:

- With a defined grid size of (x*2 +1), the four corners heights are adjusted with random and HEIGHT_MAX.
- The diamond step is used which adjusts the middle point of the four corners with an average height and some random value.
- The Square step is used which calculates the height of 4 points based on the middle point created earlier.

The last two points will repeat with a smaller step size until it's one, meaning that all points of the grid have been adjusted to their surroundings.

While the terrain is done, it still has a basic material which doesn't make it appealing. To fix this, texture blending is used. This process uses multiple textures and lerping (a mathematical function to find the value between two values and a percentage) to generate the correct texture for that point. For the project, the height was used to determine which texture or textures to use for blending. This creates terrain with various levels from sand to snow.

Finally, shading and lighting is applied to the terrain to give a 3D effect in the scene. Materials and lights are sent to the shaders and using the textures normals, calculates how the shading is applied.

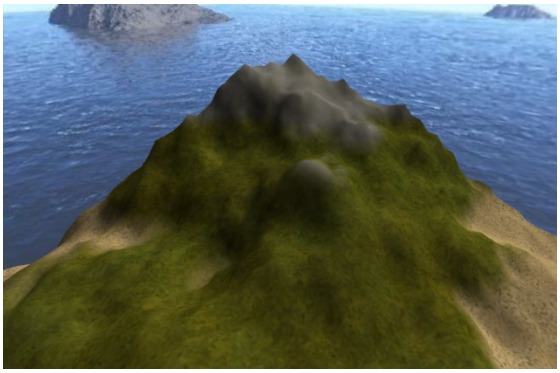

Figure 2: Blending of textures doesn't give the texture an hard edge

WATER

While the creation of water is the same as terrain, it operates differently to how it's displayed. the issue with water is that it's never still and always moves. To combat this, the vertex shader is used. Shaders in a graphics API allows the programmer to create interesting adjustment to the values they receive. The vertex shader deals with the physical coordinates while the fragment shader deals with the colouring of the point. Changing the y value of water on the CPU every frame is very inefficient. Instead, the original coordinate can be sent to the vertex shader and use trigonometric functions to create waves.

In the fragment shader, the transparency of the water can be changed so the ground can be seen underneath. To make the water look like its "flowing", the same effect of waves in the vertex shader can be done in the fragment shader as well. This time, it will change the texture coordinates back and forth, making the texture move in run-time.

Figure 3: the water's height is adjusted to create a wave

TREES

Trees is a type of procedural element which requires additional data to be created properly. It needs terrain's data to adjust itself accordingly to the ground. When creating the tree, a phenomenon was used called fractals. Fractals are endless patterns that have the same pattern no matter the scale. This was used to create smaller branches.

The scene needs to be filled with hundreds of trees to simulate forests and to add detail to the scene. However, this can cause an issue with draw calls. A draw call is a message to the GPU to draw the data, but it is expensive. As the tree is made from 6 separate draw calls, the amount of draw calls can build up to large and numbers the more trees there is. The solution to this is instancing, which is the giving information to the GPU and state how many times to draw in one draw call. This means that thousands of trees can be created just using 6 draw calls. This allows additional data to be sent to the GPU to customise the trees, like scale, rotation and position offset.

Finally, using the same technique as water, the trees are also animated to simulate them reacting to the wind in the scene.

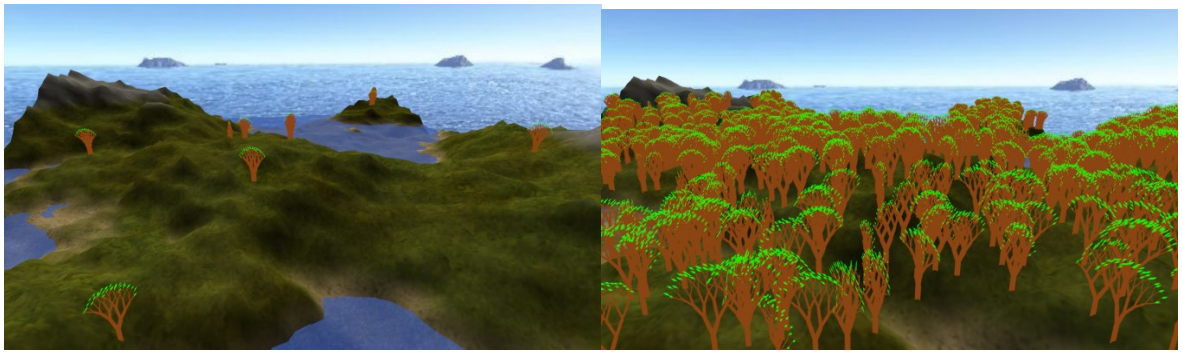

Figure 4: (left) 10 trees via a for loop, (right) 1000 trees via instancing with very little consequence as it uses fewer draw calls

GRASS

As the ground look plain, some additional detail can be added, which is grass. Grass, or a blade of grass, is made of 5 triangles with 7 points and a triangle strip. The advantage of this is that parts of the trees instancing can be used with how vertices are linked in the skybox class. This means that the five triangles can be sent normally by using gldrawArrays. The way the trees are instanced is applied so that 1000's blades of grasses are produced. Finally, a small offset is added to make it random.

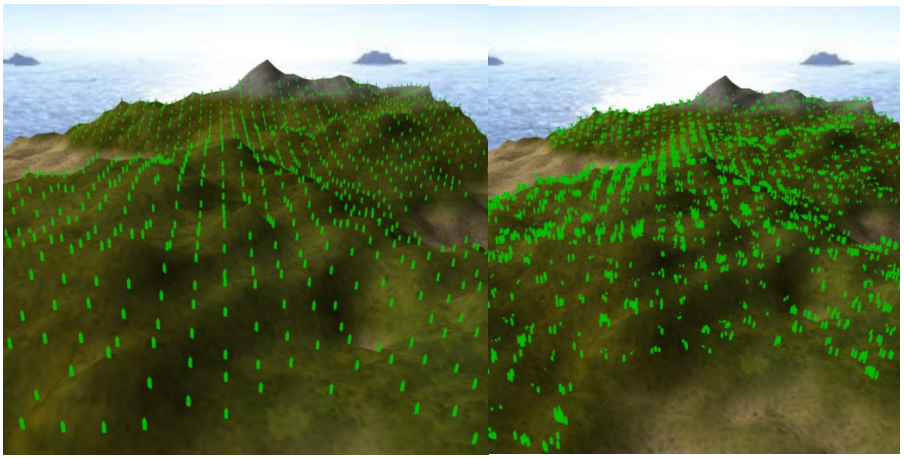

Figure 5: (left) even through there is 10000 blades, the only positions available are the grid points. (right) some randomisation has been implemented to "clump" grass

CODE AND SCENE MANAGEMENT

It's crucial that the code is optimised and factored in such a way so that it's quick to. Each of the features are in their own class and can be drawn independently. The program allows multiple shaders with different objects, while making sure that there is no interference. To support texture-loading of multiple different images, a library called SOIL was included. This means that Jpegs (the image type used for the skybox) can be used.

To allow the user to see the terrain and move around, a camera class was created. The class is responsible of producing projection and view matrices that the shader uses to adjust the coordinates. With additional keyboard and mouse input, the user can move the camera in any direction.

CONCLUSION

WHAT WENT WELL

Given that we had considerably less time on this assignment, I'm impressed what the amount features that were done and completed in a nice and organised fashion. The product looks impressive knowing that most of the product is from code and vertex points. The best feature is the trees. Instancing is an advanced feature that opened the possibility of thousands of objects with little consequence. With a function to calculate points in a certain region, the trees were able to stick to the correct land area and not in the ocean.

Upon doing the grass, I discovered a solution to a previous issue. Before, only some of the triangles in the leaves and grass were show at different angles. The problem was that the back-face culling caused the triangles to disappear depending on the view. Once this was disabled for the grass and trees, all the triangles are shown.

WHAT DIDN'T GO WELL

There have been a few issues that did and didn't make sense. One issue was how shading was done to the objects. For now, the shading ability is applied to the terrain and water, but the water does not show any signs of shading. On creation, the normal for the water is all up vectors (0,1.0,0), which provide no shading result. As the y values are being adjusted in the vertex shader, and it only looks at one vertex at a time, it's impossible to recalculate the normal for that coordinate. the two solutions for this are the move the y adjustment to the CPU (which is expensive) or use a geometry shader (which is not enough time).

While the trees look nice, they still feel 2D. an improvement to this is to create a 3D object like how the Skybox was done. Then, the vertices are adjusted using the same branch calculations and the data is sent. As the trees won't move a lot, their normal can be calculated to add shading.

Finally, while the trees do add some detail to the land, it's not enough. It would have been nice to add additional elements like another tree, coral for underwater trees and flowers to make the scene colourful.

IMPROVEMENTS

As stated, the scene needs more elements to make it nicer to look at. This includes trees, clouds and flowers. The blending of the textures on the terrain need to be defined more using better values. The sand and grass textures work, but the rock and snow textures do not.

Another improvement is the option to completely generate the world using a button in runtime. This would show the true power of how procedural generation can be used quickly to create new worlds.

REFERENCES

SOIL. (2014). [Multi-type image loading for C++].

https://github.com/paralin/soil