Procedural 3D Terrain Generator

Overview

For my final project, I created an infinite, procedural terrain generator using the Unity game engine. There is a player controller that can navigate the terrain which is made up of terrain chunks. As the player moves around the map, terrain chunks spawn and de-spawn around the player. The landscape of each terrain chunk is determined using randomly generated Perlin noise to determine the height of parts of the chunk. There are also two shading modes that can be used, Gouraud and Flat shading.

Details

Noise Maps

Perlin Noise Map

Perlin noise is the foundation of the random generating aspect of the game. A Perlin noise map is created with colour values ranging from white to black. I added support for multiple octaves which allows for smaller random variations in the noise map. As more octaves are added the wavelength increases and the amplitude decreases, allowing for more and smaller variation in the noise map produced by that octave.

Falloff Map

To also add to the variety of noise maps I added support for a falloff map. This is a map that contains black and white values that are determined by a mathematical function. The values near the edge of the map are all white, and values become closer to black as they near the center of the map, and then finally become completely black at the center of the map. This map is then subtracted from the Perlin noise map generated earlier, this maintains the random noise values in the center of the map and creates only values of white around the edges of the map.

Meshes

World coordinates are created using the x, and y and colour value of each point in the noise map. The colour value between white and black determines what the height of the mesh will be at that point. This information is stored in an array where it is then used to create triangles between each point (vertex). This uses the Perlin noise map to create a mesh which is the foundation of each terrain chunk.

Shading

There are two different types of shading in the project, Gouraud and Flat shading. The Gouraud method calculates and interpolates a normal vector at each vertex based on the triangle(s) that border it. This creates smoother shading, blending the seams between triangles more as seen in some of the provided executables. The Flat shading method calculates a normal vector at each vertex, independently of the triangle(s) surrounding it, if any. This means that if a vertex is shared between two or three triangles, multiple normal vectors will be calculated for that single vertex. This causes the shading to only take into effect the normal vectors of a single triangle, creating a consistent shading over each triangle face, not taking into consideration the surrounding triangles.

Endless Generation / Terrain Chunks

The Perlin noise map, meshes and shading are all combined to create an individual terrain chunk. Each chunk contains a mesh to give it a shape as well as a physics collider for the player to collide with. Only a few chunks surrounding the player are visible at once, which helps to improve efficiency by reducing the number of vertices/colliders rendered at once. As the player moves new chunks are spawned in front of them and chunks behind them are hidden.

Mesh LOD

To help improve efficiency I implemented a LOD (level of detail) system for the chunk meshes as well as the physics collider. Changing a chunks LOD causes the number of vertices in that chunk's mesh to be decreased, creating geometry that is less demanding to render. The categories of LOD are created and specified in the Unity Inspector. The LOD of a chunk is determined by calculating the distance from each chunk to the player, checking to see which LOD threshold that distance is in, and then setting the chunk to that thresholds LOD value.

Physics Collider LOD

The physics collider mesh also uses a similar LOD system where the mesh LOD used for colliders can be chosen from one of the mesh LOD levels created. I used a LOD one less than the max LOD for the physics collider. This is to improve the efficiency of calculating the physics collider mesh which can be computationally difficult. To further improve the performance, I also only created a collider mesh for chunks that are close to the player and not chunks farther in the distance.

Seamless Borders

To make the seamless border between different terrain chunks I took into consideration bordering triangles when calculating the heights in a chunk. Knowing the surrounding triangles height values, I could make sure that the edges had height values that lined up with its bordering triangles.

Threading

I took advantage of threading by creating two threads, one to generate noise maps, and the other to generate meshes. These threads are used to generate/update terrain chunks when changes need to be made, like when the player walks within range to spawn a new terrain chunk. Using threading allows the calculations to be done when needed without causing the game to freeze or stall. This was challenging for me because I had never used threading before. Now I understand how threading can be used to complete processes and computation independently from main processes to avoid locking up a program.

Environment

I added a few things to the environment to improve its atmosphere and feeling. One of them being a sun and moon that I added which orbits around the center of the map, opposite of each other. They both contain a directional light which I made to always point towards the center of the map. These lights create nice looking shadows across the landscape. I also added a grey fog in the distance. This creates a greater feeling of realism, making mountains in the distance feel farther away and has the added benefit of hiding some of the lower detailed terrain chunks. Lastly, I added a spotlight to the player which can be used during night time to see how lighting interacts with the terrain mesh and other generated shadows.

Mesh Texture

The texture that is applied to the mesh of each terrain chunk can be modified and customized in the Unity Inspector. Multiple textures can be used with a mesh by defining the starting point of a texture within the range 0 and 1. The 0 in the range refers to the lowest point on the mesh and 1 refers to the highest point. The value chosen for that texture will be the height at which the texture starts being used on the mesh. That texture will continue being used until either the top of the mesh is reached or another texture starts above it. Besides defining the starting point of a texture, I added a few other customizability options. One of them is the ability to change the tint colour and tint strength of the texture. This allows the texture to be overridden by a different colour to achieve a different appearance. Another option I added was the ability to blend between two adjacent textures, this allows for smooth and natural blending between water and sand for example. The last customizable choice I added is texture scaling. This is crucial for adjusting textures to help reduce the tiling effects created by some seamless textures.

Shader

The shader is what creates and applies the texture to the meshes. I use tri-planar texture mapping to ensure that textures on steep slopes of the mesh do not appear stretched. I did this in the shader by calculating a projection of the texture onto each plane and then combining those projections. This is also where I calculate the “strength” of the texture that is drawn. This strength value is used for blending between two textures.

Difficulties / Issues

Due to computational limitations, I set the LOD of the collider meshes to be one step lower than the most detailed LOD mesh. This is because setting the LOD of the collision mesh to the most detail introduces stuttering when the collision mesh is calculated for nearby chunks. This leads to some side effects like the camera clipping through the mesh in some rough terrains such as mountain peaks, and I believe it is what may lead to the player occasionally falling through the map.

I made the moon by creating a sphere, placing it a far distance away from the center of the map, and having it rotate around the center of the map. Because I used a Unity GameObject the moon occasionally disappears and reappears from view. I think this is caused by the moon object being clipped by the cameras far clipping plane. I didn’t get the moon and sun to both rotate around the player while they move around the world. Without this, if the player runs far enough in one direction they can run past where the moon and sunset and rise.

Topics Learned

I learned many different new techniques related to graphics as well as many methods of programming while working on this project. I learned what tri-planar texture mapping is and how to implement it, the basics of delegates and events as well as how to use callbacks. I also learned how to use threading and how it can be useful for improving computational tasks as well as what octaves are and how they can be useful using values of lacunarity and persistence. I am not new to the Unity interface but I am a beginner when it comes to programming in Unity. This project has taught me many useful programming practices related to unity such as how to show and modify variables I created in the Unity Inspector window, new

methods and properties of the C# language, as well as how to override the default Unity Scene View to update and see changes to variable without entering “Play Mode” in Unity.

Instructions

Builds

The “Builds” folder contains five different executables of the program. Each executable contains slightly different settings to demonstrate the variety of options available. The keys for controlling the player are written at the top left of the screen in every build. Notes that I have added are displayed on the top right of the screen.

“Default Map”

This build uses the Gouraud shading method along with the terrain generated using standard Perlin noise.

“Falloff FlatShaded Map”

This build uses the Flat shading method along with terrain generated using the falloff noise map which creates a more island filled terrain.

“Falloff Map”

This build uses Gouraud shading as well as a falloff noise map

“FlatShaded Map”

This build uses the Flat shading technique along with terrain generated using the standard Perlin noise map. In this build shadows from the sun’s directional light are disabled. This makes it much easier to observe how the light affects the lighting on each individual triangle of the meshes.

“NoDayNight Map”

This build is the same as the “Default Map” build except that it has no day and night cycle. This is so you can explore and observe details of the map, techniques, etc. without darkness and shadows.

Unity Project

If you open the project in Unity and navigate to the “Terrain Assets” folder under the project pane, along with all the assets, you can see the different settings used for each executable of the game. Selecting any of these “Noise”, “Terrain” or “Texture” objects will show their respective settings in the Inspector panel. If any of these settings are modified the changes can be seen by ensuring the “Auto Update” box is checked, by clicking the “Update” button at the bottom of the Inspector or by running the game inside unity. I encourage you to make changes to see how the generation works but to see any changes made the same scene must be loaded as the folder in which the Noise, Terrain and Texture objects you are modifying are located in. A video I recorded showing some of these things is included in the project submission.

Noise Data

Inside this object are settings for the persistence, lacunarity, scale of the noise, the number of octaves, the random generation seed and the x and y offset of the noise map.

Terrain Data

Inside this object are settings for the scale of the whole world, the option for flat shading and falloff map, a multiplier to change the height scale of the map as well as a mesh height curve. Selecting this mesh height curve allows the height values of the map to be modified for a specific range of height values.

Texture Data

Here the texture layers can be modified by adding new textures, changing tint colours and values, changing the starting height of each texture, the scale or the blend strength.

External Resources

I used the Unity3D game engine to develop this program. The player controller that I used is a standard asset created by Unity that I downloaded from the unity asset store. This includes everything in the “Standard Assets” folder in the project assets folder.