GENERATING REALISTIC VIRTUAL ENVIRONMENTS USING PROCEDURAL GENERATION

INTRODUCTION

The aim for this project is to research how procedural generation is used to create realistic environments and use it to produce a video game to show what has been found in the research.

The video game will allow you to move around an endless world populated with plants that have been randomly placed and generated using algorithms. It will show how these algorithms can be used to create games that can have almost infinite worlds and environments to explore, which in turn creates a new experience every time you start a new game. It will show how you can use significantly less memory then most other games and run just as smoothly.

PROCEDURAL GENERATION

Procedural generation is term used to describe the method of creating data from an algorithm. Procedural generation can be used for many things from creating textures and terrains to more sophisticated things like loot systems and character creation.

Procedural generation uses a seed to generate a new set of data every time a new seed is used which is how you get a new experience every time you create a new world, this also means that you can use the same seed to replicate the same data every time as well.

Render distances are used to keep memory usage as minimal as possible by only rendering game chunks in a certain distance to the player, this allows for smooth gameplay and low memory consumption.

PERLIN NOISE

Perlin noise is an algorithm invented by Ken Perlin in 1983 that creates a set of pseudo random numbers. A classic random number generator produces numbers that have no relationship to each other making it very erratic which would create unrealistic terrain. Perlin noise fixes that problem by having each random number have a relationship between then, making the numbers form smooth curves and no sharp spikes which makes the terrain very realistic. The Perlin noise function returns values from 0 to 1.

In Figure 1 you can see a Perlin noise map that was generated in Unity where each value represents a colour from black to white. In Figure 2 colour thresholds have been added so that different values represent different colours to represent different parts of the terrain. Finally Figure 3 is a terrain mesh created by adding height based on the value of the point, for example if the point is the value 1 it will be a peak and between 0 and 0.4 values is the water.

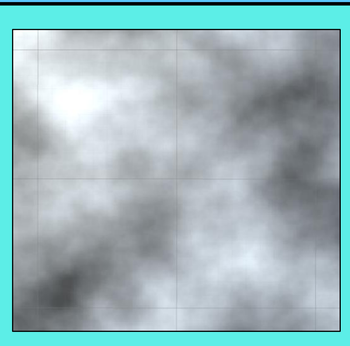

Figure 1 - noise map using Perlin's algorithm

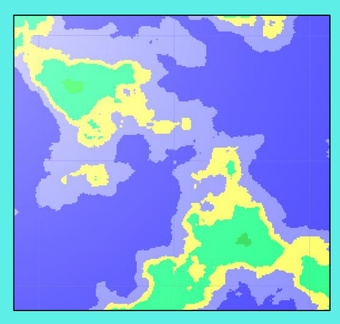

Figure 2 - colour map using Perlin's algorithm

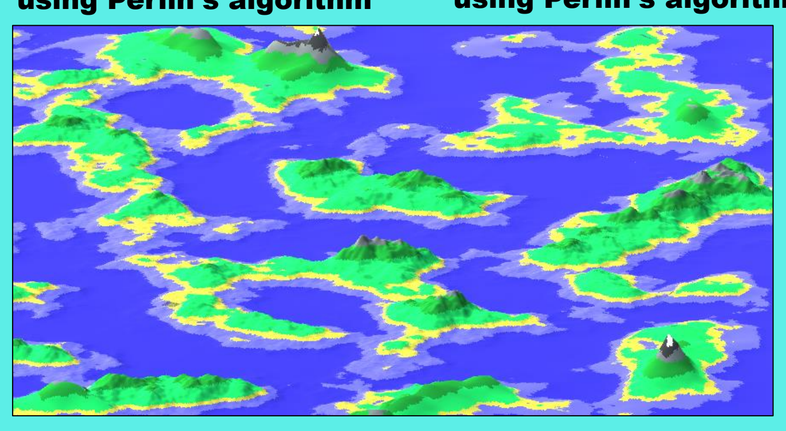

Figure 3 - virtual terrain chunks generated by using Perlin's algorithm to create a mesh

L-SYSTEMS

L-systems are a way of procedurally generating plants in a virtual space. They use a very simple algorithm where you have a string of characters and each character represents an instruction to build the plant, For example 'F' means go up and '+' means to rotate on the axis.

It starts with an axiom, usually the character 'X' and uses rules to build a long string of characters through a number of iterations, to finally reach a complete set of instructions. The rules used are very simple it looks at a character for example 'X' and then says turn the 'X' into 'FFX', it does this for every character in the string. Figure 4 and Figure 5 show how plants can look different using different amounts of iterations used.

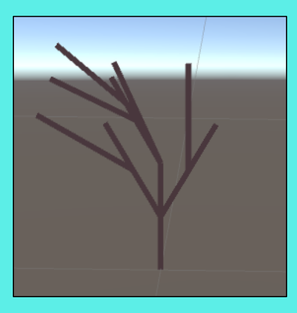

Figure 4 - plant created using L-systems rule set and 2 iterations

Figure 5 - plant created using L-systems rule set and 6 iterations

FUTURE WORK

If given more time to work on this project the game could include more algorithms to create more unique terrains such as biomes which contain unique plants and animals. Cave systems could be implemented to allow even more experience for the player by allowing exploration of procedurally generated caverns. Animals could be procedurally generated to create unique looking animals everywhere.

The player could have the tools to destroy and create new data allowing them to sculpt the world to look how they desire giving them even more experiences and allowing the game to be more fulfilling. Procedural generation can have infinite possibilities for the future of the game.

PRIFYSGOL BANGOR UNIVERSITY

Author
Adam Watson
adamw200100@gmail.com

Supervisor
Llyr Ap Cenydd
llyr.ap.cenydd@bangor.ac.uk

Project made in Unity using the C# language and Unity libraries.