Supervisor: Dr. Julien Cordry

Clariana Justes, Oriol Marc

V8086261

Procedural Terrain Generation in UE4

BSc. Computer Games Programming

Contents

Abstract 2

Acknowledgements 2

Introduction 2

Methodology 2

Research 2

Perlin Noise 2

Level of Detail 3

Tile / Chunk 4

Design 4

Implementation 5

Optimization 5

Testing 5

Conclusion 5

Bibliography 5

# Abstract

Procedural Generation is a feature becoming more popular each year in the videogame industry and for the infinity possibilities that it grants. In this project, using Unreal Engine 4 and the Procedural Algorithm Generation it has been created a Tool with a variety of options to customize the final result used in conjunction with a series of techniques to create high quality procedural terrain in real-time and pre-bake.

To showcase this, an easy-to-understand and well-structured interface was built and all necessary elements to visualize the product were created. This report explains the classes that were built in order to visualize the procedural terrain, how it is generated and the parameters to customize the terrain.

# Acknowledgements

I would like to thank to my project supervisor Julien Cordry who provided support and advice on the product all throughout the development of this project. I would also like to thank my peers and friends on the course who provided support and feedback during the development process.

# Introduction

## Procedural Generation

Procedural Generation is becoming more popular each year between videogames. This is mainly because it does the work of designing and creating new assets and environments based simply on mathematical functions. This has its positive and negative side. For example, if a little and scape is needed for a game, it is better to get an actual designer to build in detail, but if what is needed is a gigantic landscape for an open world game it is probably best and more efficient to generate it procedurally. The power of making the procedural content generated more unique resides in how complex the mathematical functions that generate that content are and the customization of the options for that.

Many games have already used Procedural Generation to create new amazing environments because the huge scale of the map. Tittles like Minecraft (Mojang, 2009) use Procedural Generation to generate the game maps. No Man’s Sky (Hello Games, 2016) goes even further by creating planets and galaxies. Other games use it to create procedural items and weapons like Borderlands Series (Gearbox Software).

## Product

The product is a custom tool developed in C++ in Unreal Engine 4 that use Perlin Noise to make possible the generation and visualization of a procedural generated terrain in in real-time and pre-baked with asset placement and semi-biomes.

The product has been built from scratch to have a higher control over all the components that generate and visualize the terrain, enabling a high efficiency and optimization. The only dependence is a Plugin called Runtime Mesh Component which help to create and modify meshes at run-time more fasten than Unreal Engine 4 do. Also using basic technique in videogames called Instanced Static Mesh for an optimization with the assets on the terrain.

## Inspiration

The main reason and inspiration for choose this particular project is the immensity of possibilities give us the procedural generation whether in terrain generation, creatures generation, difficulty of game, materials and more. Also, have a great knowledge of this can open me doors in the industry because the procedural generation is in this time more used in the industry and speed up the content generation. For example, the pre-baked option accelerates the velocity of launch a new game generating a terrain adapted to the game specifications. Since I study how to do videogames, I am was really appassionato about this complexity of procedural content generation.

## Existing Technologies and Dependencies Used

Due to great interest in Unreal Engine and the potential in the industry I was decided to make the whole project was coded and compiled using Unreal Engine 4 and Microsoft Visual Studio 2017.

The Perlin Noise Algorithm is my own implementation and interpretation to avoid the number of dependencies and also has a really easy structure to change it for another different algorithm implementation.

The Plugin Realtime Mesh Component it is easy to change by the equivalent in Unreal Engine 4 called Procedural Mesh Component.

# Research

## Perlin Noise

Perlin noise is a type of gradient noise developed by Ken Perlin in 1983. The most common used ones are one, two and three dimensions. The frequency of the noise controls the distance at which samples are taken and it is used to control the scale. The amplitude controls the maximum and minimum values that the noise can take. In graphics programming Perlin noise can be used to generate two dimensional textures which later can be used as heightmaps for terrain generation or as a regular texture.

Heightmaps can be used to describe terrains by interpreting the values of the noise texture as heights. Perlin can be effectively used to generate a basic random terrain. From here, different approaches can be taken to modify the Perlin noise equation so as to make a terrain feel more realistic.

|  |  |
| --- | --- |
| Imagen relacionada | Resultado de imagen de perlin noise |
| Figure 1 | Figure 2 |

The Figure 1 show a regular sample of the 2D texture of Perlin Noise.

It is possible to make the result of Perlin Noise more realistic / different by iterating several times over its values using different types of algorithms such as Fractals or Hydraulic erosion.

## Level of Detail

The Level of Detail (LOD) involves decreasing the complexity of a 3D model representation as it moves away from the viewer or according to other metrics such as object importance, viewpoint-relative speed or position.

This technic is used to optimize the 3D environments to reducing the complexity of the models at different distances changing the number of triangles rendered an object as its distance from the camera increases. It allows reduce the consuming a huge number of resources on models far of the camera vision.

|  |
| --- |
| Resultado de imagen de level of detail camera distance |
| Figure 3 – Level of Detail |

As seen in Figure 3, the LOD change the density of polygons is changed but also trying to keep the initial shape of the object. This technique can be used in terrain rendering due to the fact in a scene that is viewing a terrain, both close and far terrain can be viewed at the same time and do not require to have the same polygon quality. The further away the objects are the easier it is to describe with lesser polygons than the initial shape. And this simple concept allows to save resources at the rendering moment.

## Tile / Chunk

The Tile/Chunk involves split the terrain generation in small pieces to reduce the calculation time and the number of polygons rendered.

Combining with Perlin Noise and the Level of Detail (LOD) makes possible generate an infinity number of pieces of the terrain using the Perlin Noise value and make it works faster reducing the LOD on the far chunks.

Also allows the possibility to use a Pooling System to reduce the number of chunks on the world modifying the chunks not rendered. In this way allowing to avoid creating more memory for the generation of the land using the memory already created but not used.

## Futures in Multithreading

Futures are a high-level mechanism for passing a value between threads, and allow a thread to wait for a result to be available without having to manage the locks directly.

Using the Futures and Promises allow me to manage a lot of different tasks asynchronous to speed up the terrain generation, the mathematical operations and calculations. In exception of the creation of the mesh because this not allow reserve the memory in a different thread of the main.

## Mesh Instancing

In real-time computer graphics, mesh instancing is the practice of rendering multiple copies of the same mesh in a scene at once. This technique is primarily used for objects such as trees, grass, or buildings which can be represented as repeated geometry without appearing unduly repetitive, but may also be used for characters. Although vertex data is duplicated across all instanced meshes, each instance may have other differentiating parameters (such as color, or skeletal animation pose) changed in order to reduce the appearance of repetition.

When rendering static meshes, a common CPU cost is the actual function call to render the mesh.  Under the covers, this draw call often contains a reference to a VBO already loaded on the GPU, as well as transform information where to render the mesh.  The call itself does not have much overhead as the real work is done by the GPU, but making this call a large number of times certainly adds up.  Doing this thousands of times every frame causes CPU driver overhead.  For example, imagine a wall that is made of 10000 individual bricks.  This would result in 10000 draw calls.

Using the Instanced Static Mesh allow to store on the GPU memory in a Vertex Buffer the actual vertices of a static mesh. The draw call consists of telling the GPU to render the mesh at a specific transform and orientation.  When rendering a batch of identical meshes, a further possible optimization would be to store the actual transforms on the GPU as well.  Using this approach, we can render all of them with a single draw call.  An instanced static mesh leverages this approach.

|  |
| --- |
|  |
| Figure 4 – Trees Instanced on the project |

All the Trees/Bushes/Rocks on the Project are using this technique and allow me to have more than 3000 rendering on the screen without affect the framerate of the project.

# Design & Implementation

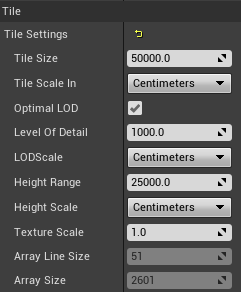
## Overview

The Implementation has been made using a SCRUM methodology to keep in mind the task that should be achieved and to make development dynamic.

## Terrain Generator

|  |  |
| --- | --- |
|  |  |

## Tile



## Biomes

|  |  |
| --- | --- |
|  |  |

## Runtime & PreBake

|  |  |
| --- | --- |
|  |  |

# Optimization

# Testing

# Conclusion

# Bibliography