Abertay University
Student Services
The student author of this work has been assessed as dyslexic. Typically, the presentation of this work is unlikely to represent the quality of the content. The student should not be penalised for poor spelling, minor innacuracies or any other reason attributable to dyslexia.
If there are any queries, please contact Richard Costella in Student Services.

Procedural Generation of Virtual Reality Environments

Eoghan K. Mulvenna

BSc (Hons) Computer Game Applications Development, 2018

School of Design and Informatics
Abertay University

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

I'd like to thank my supervisor Karen Meyer for all her help and advice with this honours project and dissertation. I'd also like to thank my partner Katie and flatmates Lauren and Jonny for all their support. Additionally I'd like to thank all my friends, my parents Gerry and Jacqueline and the lecturers and fellow students of the Abertay University School of Design and Informatics for all their help and support throughout the year.

Abstract

Context

A defining problem upon the consumer release of virtual reality hardware was a lack of content in Virtual Reality games. One way that the content within these games could be improved is the use of Procedural Content Generation.

Aim

Investigate whether the use of PCG is a viable and effective technique for improving VR games.

Method

In order to look into the applications of procedural generation in terms of VR and test if it is a viable method for the generation of VR content an application was created in which you can explore a procedural generated environment in VR. This environment was created through a combination of Marching Cubes, Cellular Automata and Simplex Noise to produce an interesting and immersive world. The application was then user tested to assess the effectiveness of the research project. Qualitative and quantitative data was gathered through user testing and performance testing so that the feedback and results from these tests could be used to determine how viable procedural environments are for use in VR games is.

Results

The results of this project demonstrated that while not a perfect answer to VR's content problems procedurally generated environments where indeed very effective at encouraging exploration, discovery and immersion as well as greatly increasing replayability. However, the results also revealed that a lot of thought and work need to be put into VR performance and gameplay in order to achieve good results using procedural environments.

Conclusion

Using the techniques and methods described in this paper PCG can be achieved that works very well in VR as on top of the expected result of being able to create large interesting environments, it also creates environments that encourage natural exploration and invoke a fantastic sense of wonder and discovery.

Abbreviations, Symbols and Notation

VR – Virtual Reality.

PCG – Procedural Content Generation.

CA – Cellular Automata.

MVP – Minimum Viable Product.

1. Introduction

Two of the stand out trends of the modern games industry have been a huge increase in procedurally generated games as well as the rise of virtual reality. However, there are very few games that combine the two despite the many theoretical advantages. This project aims to investigate whether there is any merit in utilising procedural generation techniques to improve virtual reality games and to record and document the benefits and drawback of doing so.

1.1 What is Procedural Content Generation

Procedural content generation (PCG) in games refers to the creation of game content automatically using algorithms. (Togelius et al., 2011, p. 1*).* But why would creating games content algorithmically be useful? There are many reasons for using procedural generation in a game; from reducing the number of designers and artists needed, to allowing for whole new types of games like Rogue (A.I. Design, 1980) and the Roguelike genre that followed. But most importantly in the context of VR with procedural generation *there is in principle no reason why games need to end.* (Shaker, 2016, p. 3).

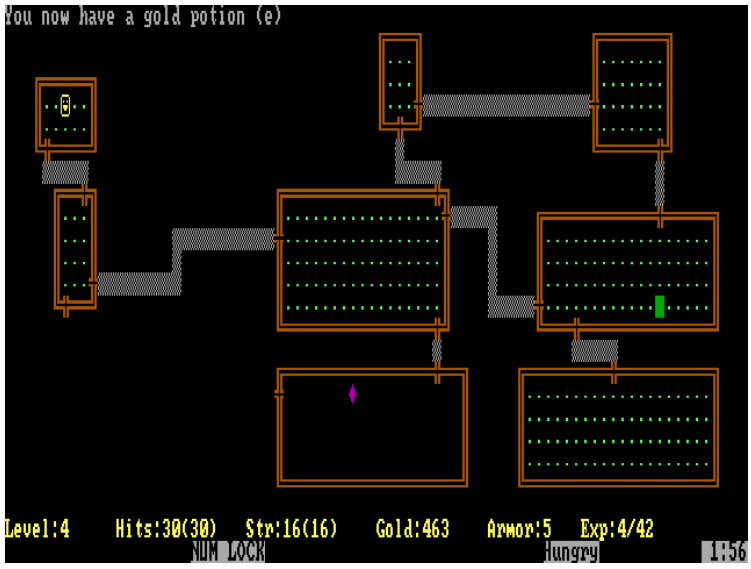

Figure 1: Rogue Screenshot (Rogue, 1980)

Procedural generation has been used in game development for decades. One of the first known instances of a game making use of procedural generation is Beneath Apple Manor (Worth, 1978) where levels made up of various rooms and connecting pathways would be randomly generated. While Beneath Apple Manor was well received at the time of its release it is largely forgotten today with the namesake for the genre that followed going to Rogue instead, despite its later release (Scorpia, 2018). Since then procedural generation has become a cornerstone in game development, being used for everything from simple levels to entire games, such as Dwarf Fortress (Adams, 2006) where every element right down to the world, characters, story, and quests is procedurally generated.

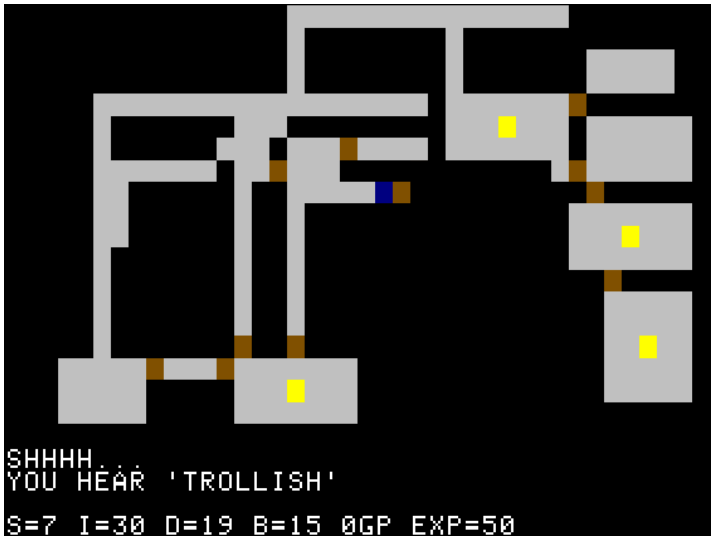

Figure 2: Beneath Apple Manor Screenshot (Beneath Apple Manor, 1978)

1.2 Virtual Reality

On the 28th of March and the 5th of April 2016, the Oculus Rift and HTC Vive respectively began being rolled out to the general public and their releases polarised many people on the topic of Virtual Reality (VR) and its place in the games industry. Although the Oculus Rift had been in development for years beforehand and had released several development kits to the public, this was the first time that completed VR headsets were on the market. Many people hailed VR as the future of gaming and worth the steep price (Adebayo, 2016) while others named it a gimmick that would soon pass (Iwaniuk, 2016). Despite the many difference of opinions on the place for VR in the games industry there was one unifying attitude held by all who had the chance to try a VR headset: the games lacked content (Durbin, 2016). It was true that many of the games were

much closer to short experiences than games and even the more fleshed out and bigger titles at release such as Vanishing Realms (Indimo Labs LLC, 2016) and Job Simulator (Owlchemy Labs, 2016) still only had around three hours of playtime.

Almost two years later and this is still very much a problem for VR games. While hundreds of games have been released for the available VR headsets most are quite short and lacking in both content and replayability. This is not inherently a bad thing and there have even been many great short games that do exactly what they were intended to do such as Justin Roiland and William Pugh's game Accounting (Crows Crows Crows; Squanch Games, 2016) which was created as a free short narrative based experience. Short games and experiences can be great, especially in VR, however there is a great lack of larger scale and longer games available and this project looks to see if one possible solution to this could be to use procedural content generation.

1.3 Aims and Objectives

- To review current examples of procedural generation in VR games as well as other games and identify which technique or techniques are most appropriate for the project.
- To prototype a procedural generation system for the creation of 3D explorable environments.
- To determine if procedural generation can create appropriate content suitable for virtual reality.
- To obtain and evaluate player feedback and data
- Gather performance data of procedural generation in VR
- Identify other methods of procedural generation that could also improve VR games.
- To recommend future work for the project.

1.4 Research Question

How can Procedural Content Generation be used to create suitable environments for use in Virtual Reality?

2. Literature Review

For this project to be carried out, appropriate knowledge of various procedural techniques as well as virtual reality is required. To achieve this, research was conducted into PCG techniques that were contenders for the project as well as into virtual reality as a platform and the current state of VR games using procedural generation. The following are the relevant results of this research.

2.1 Procedural Techniques

2.1.1 Noise

One of the most notable techniques for procedural generation is Perlin Noise which came about through the work Ken Perlin did for the film Tron (Lisberger, 1982). Perlin noise was initially used by Ken Perlin to achieve much more realistic looking textures and since has become an immensely important part of game development and is an excellent tool with a vast number of uses. At the time Perlin was creating the algorithm it would have been considerably more difficult and expensive and in some cases impossible, to generate textures of the same standard by any previously known techniques (Perlin, 1985, p. 296). Perlin noise and other types of noise that followed such as Simplex noise, also created by Ken Perlin, are now used in many different forms of PCG including landscape generation where noise is used to create a heightmap which is then used to generate 3D terrain.

2.1.2 Lindenmayer Systems

A Lindenmayer system (L-system) is a grammar based system of procedural generation, this means that it uses symbols to construct strings that with defined rules applied to them, can generate quite interesting patterns. Each symbol will mean something different to the ruleset and cause the string to change and grow. Because of the recursive nature of an L-Systems, it can create quite organic and natural looking patterns and thus is often used for the creation of biological patterns like branching trees and other plants. L-systems can also be used to generate caves by using their ability to branch of and grow to carve out cave tunnels.

2.1.2 Agent Based Dungeons

An agent based approach to creating caves and dungeons is based on placing a single agent in a map that will “dig” rooms and tunnels. The look of these rooms and tunnels is largely based on the how the agent’s behaviour is set up. A “blind” agent will create a very stochastic dungeon that will seem chaotic and may not always make sense, whereas an agent with more information about how a dungeon should look will create a more ordered and less stochastic dungeon (Shaker et al., 2016).

2.1.3 Cellular Automata

Cellular Automata is a technique that consists of a grid of cells each with one or more states. These states are changed based on the cells surrounding each cell, referred to as a neighbourhood. The way in which a cell’s neighbourhood affects the cell is defined by a set of rules that modify the cell’s state or states based on the states of the neighbourhood cells (Shiffman, 2012). There are two main types of neighbourhood used in Cellular Automata: Von Numen and Moore. On a 2D grid of cells a Von Numen neighbourhood consists of the cells above, below, left and right of the cell. A Moore Neighbourhood consists of the eight cells surrounding the centre cell. Although not that different the choice of neighbourhood changes how the cells change and how the rules are implemented and designed. In 3D Cellular Automata however, the neighbourhoods work a little differently as instead of having a flat 2D grid of cells you now have a 3D grid of cells, which means that the neighbourhoods are a lot bigger and more complicated to create rules for. So for example while a Moore neighbourhood had 8 Neighbours in a 2D grid it would have 26 in a 3D grid.

2.2 Marching Algorithms

There are many ways to take procedurally generated data and convert it into geometry such as heightmaps and voxel cube based systems, which have seen mass success ever since Minecraft (Mojang, 2009) hit the mainstream as Minecraft uses voxels for to create it’s terrain. While marching algorithms can be used for voxel systems the algorithms themselves have many uses and can be used simply to create terrain from data. The concept behind a Marching algorithm is that you have a grid of shapes and a collection of data, often called

an isosurface, which is used to create geometry based on where and how the isosurface intersects with the shapes. These shapes are made up of vertex nodes and edge nodes. Based on which vertex nodes are inside or outside the isosurface, triangles are created at the nodes to form geometry.

2.2.1 Marching Squares

The simplest implementation of this kind of algorithm is the Marching Squares algorithm. While not the first Marching algorithm created it is the simplest as it is used with 2D data rather than 3D data. This type of marching algorithm consists of four vertex nodes and four edge nodes forming a square. This allows for the 15 different configurations as well as the case where no nodes are active and nothing is rendered. This can be worked out by treating each of the 4 vertex nodes as a bit in a 4-bit number. So, if only one of the top left node was active the configuration would be 0001 (top left, fig. 3) and if all of them are active then 1111 (bottom right, fig. 3) which would form a complete square. This means that a binary value can be assigned to each of the configurations making them easier to identify and work with.

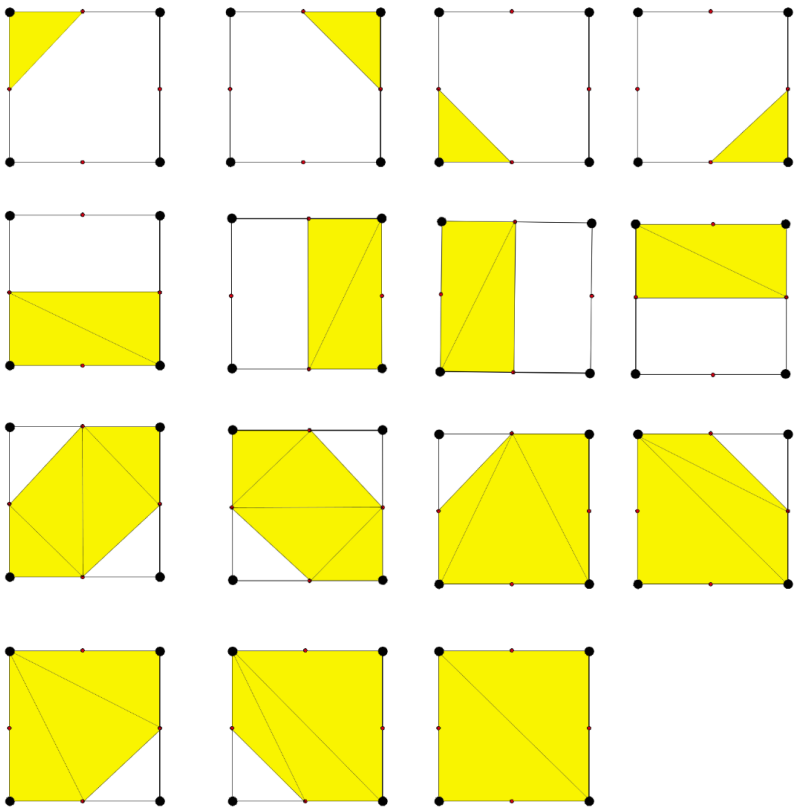

Figure 3: Marching Squares Configurations

2.2.2 Marching Cubes

Marching Cubes was the first of the Marching algorithms, it was first developed for use within medicine (Lorensen, Cline, 1987). Since its initial conception it has seen wide use among many different fields and disciplines such as engineering and computer aided design but has also seen ample success within the games industry. This is because of its ability to render geometry from data easily which proved extremely useful for game development.

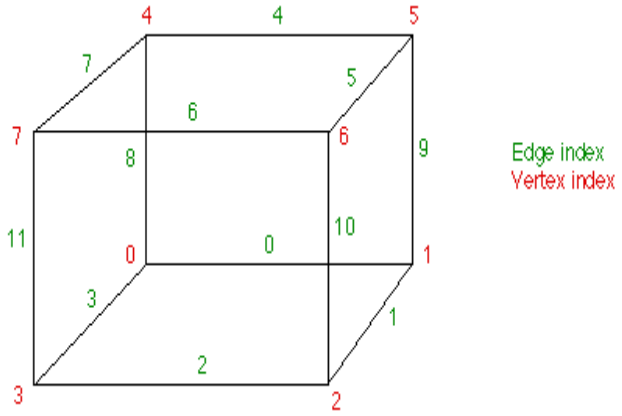

Figure 4: Cube Node Example (Bourke, 1994)

It works much the same way as Marching Squares described above, using the intersection of vertex nodes with a data field to render geometry. However, as a cube is a 3D shape there are a lot more nodes: 8 vertex nodes, one at each corner of the cube and 12 edge nodes, one in the middle of each edge. Since Marching Cubes has 8 vertex nodes instead of 4 an 8-bit number is required, this is where the increased complexity of Marching Cubes comes in as now instead of just 15 different configurations there are 255. Luckily due to the symmetrical nature of a cube the majority of these configurations are the same and can be narrowed down to 15 unique configurations (fig. 5).

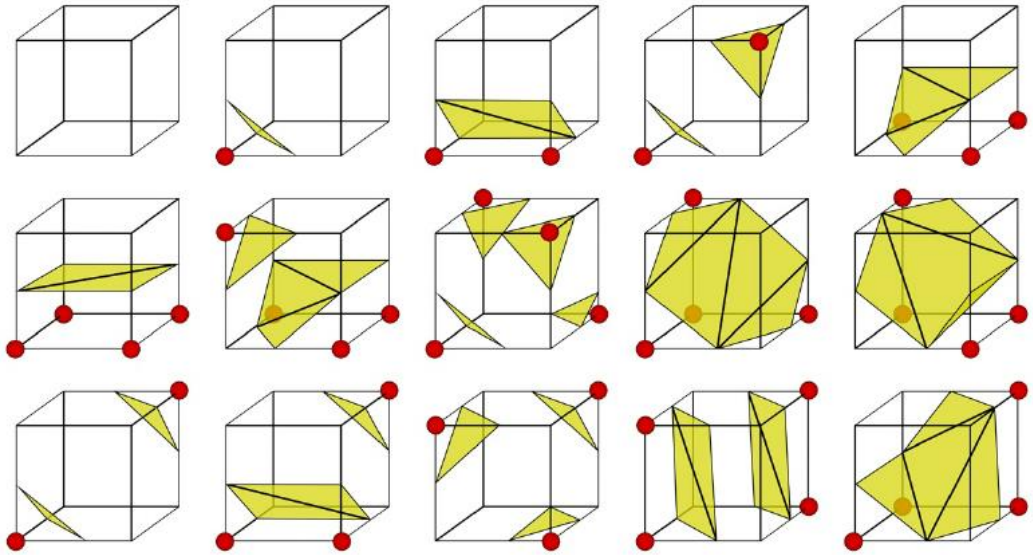

Figure 5: Marching Cubes Configurations (Laprairie, Hamilton, 2017)

2.2.3 Marching Tetrahedrons

There is another kind of marching algorithm named Marching Tetrahedrons. Building upon Marching Cubes this algorithm takes it a step further and divides each of the cubes into a number, often six, of tetrahedrons. These tetrahedrons are then checked individually to see if each of their vertex nodes are inside of the isosurface. In Marching Tetrahedrons each tetrahedron has four vertex nodes and six edge nodes which allow for eight unique configurations (Bourke, 1994).

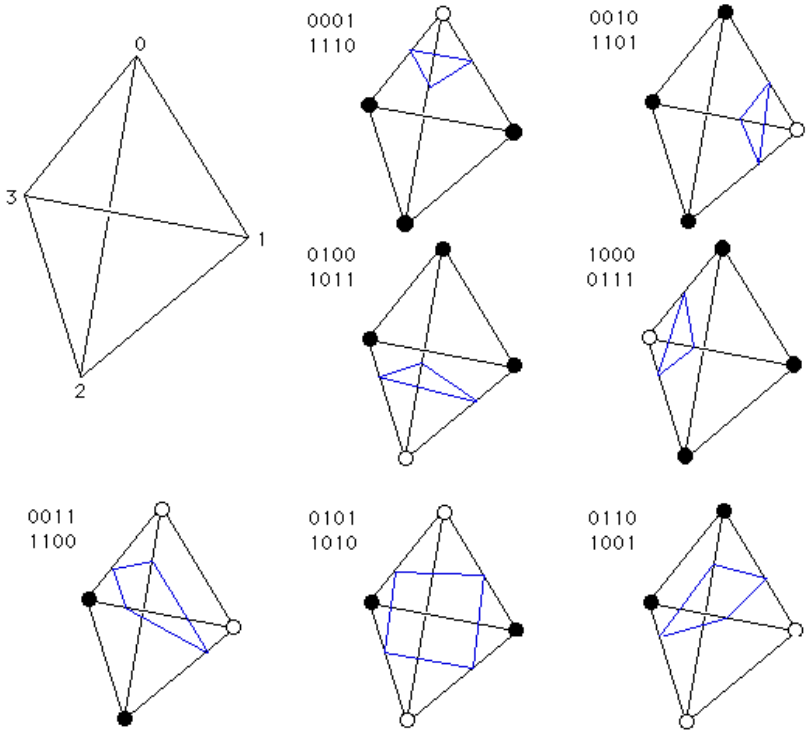

Figure 6: Marching Tetrahedrons (Bourke, 1994)

This has both advantages and disadvantages over Marching Cubes as it can lead to more exact polygonization of data with less ambiguity than Marching Cubes, however depending on the data it is possible that Marching Tetrahedrons may miss parts of the isosurface. On top of this as a tetrahedron is not symmetrical in every direction like a cube the rotation of each of the configurations is important which, depending on the implantation may mean that some cases may need to be handled differently to each other. There are even more variations on Marching algorithms building on top of this idea with different shapes using different number of tetrahedrons such as Marching Rhombic Dodecahedrons (Sahillioglu, 2008) using 4 tetrahedrons, each of the variations building upon the same systems used by the algorithms discussed here.

2.3 Virtual Reality

2.3.1 History of Virtual Reality

The Idea of being totally immersed in a virtual world has long been seen by many as the ideal way to play games and experience media. Although we are not at the stage dreamed about by countless Sci-Fi stories, with today's VR headsets we are closer than ever. However, VR is not a new idea, people have been imagining and pursuing it for almost one hundred years. The earliest known recorded reference to what we would now call VR is the short story "Pygmalion's Spectacles" (Weinbaum, 1935) in which the author describes a set of goggles that allow you to be completely immersed in a film and experience all the sights sounds and other sense as if you were really there. But it's not just the idea that has been around for a long time, people have been trying to create virtual reality systems since the 1960s such as the Sensorama and The Sword of Damocles which while technically impressive for the time were still incredibly primitive.

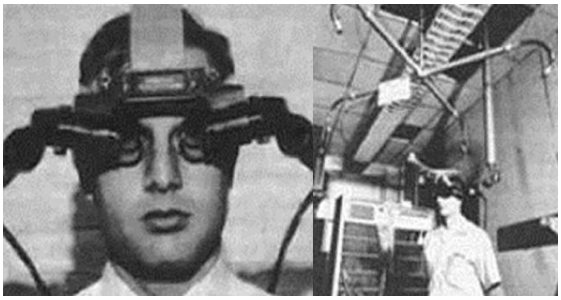

Figure 7: The Sword of Damocles (Sutherland, 1966)

Over the next few decades more research and interest arose and many more basic VR devices were created some of which being used for purposes like military, medical and flight simulations. By the 1990s the games industry had grown to be a huge force in technology and the focus of the development of VR shifted onto video games and the first wave of VR systems began to be developed. Sega were the first company to announce a VR headset in 1991, starting a VR arms race and the opinion at the time was that affordable VR would be on the market by 1994 since *VR promises big payoffs to the companies who can come out with a successful home system. Already there are, at least, half a dozen companies with plans in the works* (Engler, 1992). Most of these headsets never saw a commercial release and Sega in the end only managed to release an arcade version of their VR system. Nintendo on the other hand did release their system the Virtual Boy however it was a commercial disaster and VR development slowed to a near halt (Langshaw, 2014).

Most people thought that would be the end of VR and it was to a degree but interest was still there and VR remained a big topic in Sci-Fi media with such modern examples as Ready Player One (Cline, 2011) and Black Mirror (Brooker, 2011) gaining a lot of mainstream attention and discussion. Eventually as the years went on, VR once again became a goal for the games industry as technology had moved along a great deal since the 90s. The Oculus Rift saw several development kits released and interest amongst gamers and the media grew. Both Sony in 2014 and Valve in 2015 announced they were working on VR projects and a second wave of VR began. All three of these VR headsets were released in 2016 and unlike their 90s predecessors actually provided an incredibly immersive experience. However, despite the technology working well there was a relatively low adoption of these headsets. This was for a multitude of reasons, some of which being the price of the headsets especially the HTC Vive at US $799, as well as reports of some people experiencing motion sickness. But many people at launch believed it was a lack of content or lack of a “killer app” that was holding VR back from going fully mainstream. Despite finally having the hardware to make VR possible the main sentiment from the consumer was that *the next major obstacle will be content (Scherba, 2016).*

2.3.2 Procedural Generation in VR

In terms of VR games making use of PCG, there are very few. While there are some procedural based games that have been adapted for VR they often feel like it, as movement and/or gameplay can be very awkward and can lead to nausea; such as games like 4089: Ghost Within (Phr00t's Software, 2016) which can be played without VR but has VR support. There are some games that were designed from the ground up for VR that make use of PCG though. Games such as Trickster VR (Trickster Games, 2016) and Dreadhalls (White Door Games, 2017) make effective use of procedural generation to add to the replayability of the games as they both have set levels or tasks that remain the same but the game world for each level is different each time they are played. While there are a few smaller games like these that do use procedural generation the majority of bigger games do not and even the games mentioned use of procedural generation is still limited as they don't generate geometry procedurally. While there are some great VR games that are executed well there are also many that would benefit with at least elements of the game being created procedurally.

2.4 Summary

This project sets out to see if procedural generation is a viable way to create environments for virtual reality. Through the research that was conducted it appears the best way of investigating this would be to create a cave system using Marching Cubes, 3D Cellular Automata and Simplex Noise. The reasons for these choices are that Cellular Automata is fantastic for creating more natural looking shapes and patterns and is particularly good for cave and land type formations. As well as this Marching Cubes will be used as it is a great technique for translating generated data into geometry. Simplex noise will be used to generate the starting data for Cellular Automata to work from and is able to be seeded so by using the same inputs every time the same outputs will be achieved. By combining these techniques a cave system suitable for VR should be able to be created and can be tested to gauge whether or not techniques like these would work for VR games.

3. Methodology

3.1 Project Overview

To begin investigating the research question, an application needed to be made utilising procedural generation that could be played in VR. This meant that several steps needed to be taken, firstly an application of this scale would need to be properly designed so a design document was created that detailed the methods and techniques to be used as well as the approach to be taken to achieve the MVP as well as stretch goals that could be implemented if the MVP was achieved. Secondly a prototype was created to test out the design of the application and make sure that it was feasible. Thirdly once a working prototype was created and the design of the project updated based how it went, the prototype could be built upon into a full application that met the criteria of the MVP. After that further gameplay and interactivity was added to build upon the application to create a full VR experience.

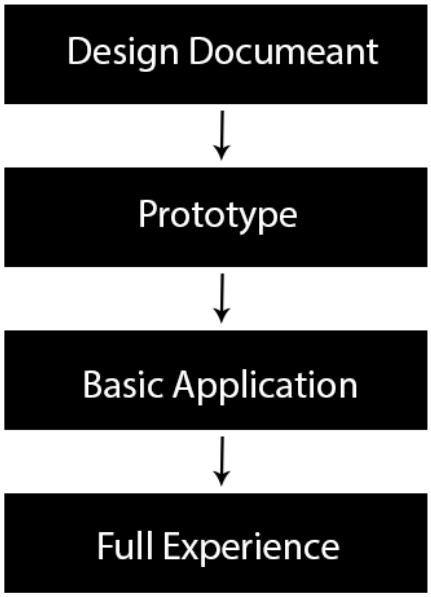

Figure 8: Project Millstones

3.2 The Application

3.2.1 Development Environment

There were three main criteria for choosing which development environment to use: the ability to generate geometry, the ability to be used with VR and finally support development in C++. Unreal 4(Epic Games, 2014) was chosen due to it meeting the above requirements, prior experience with the engine, as well as its use meaning the area of research could be focused on without having to worry about other areas of code such as graphics pipeline programming. The HTC Vive was selected to be the target headset that the application was developed for. This headset was chosen due to access and familiarity with developing for it.

3.2.2 Prototype

A prototype was designed to test the feasibility of the project. Marching Squares was used as it requires a similar methodology to Marching Cubes, but is less complex due to being in 2D rather than 3D. To create Marching Squares several base classes were created that could be built upon later for Marching Cubes. Both algorithms require features such as data generation, nodes, grid systems and procedural mesh generation, so the prototype was constructed in a manner that allowed it to be expanded from 2D to 3D at a later stage.

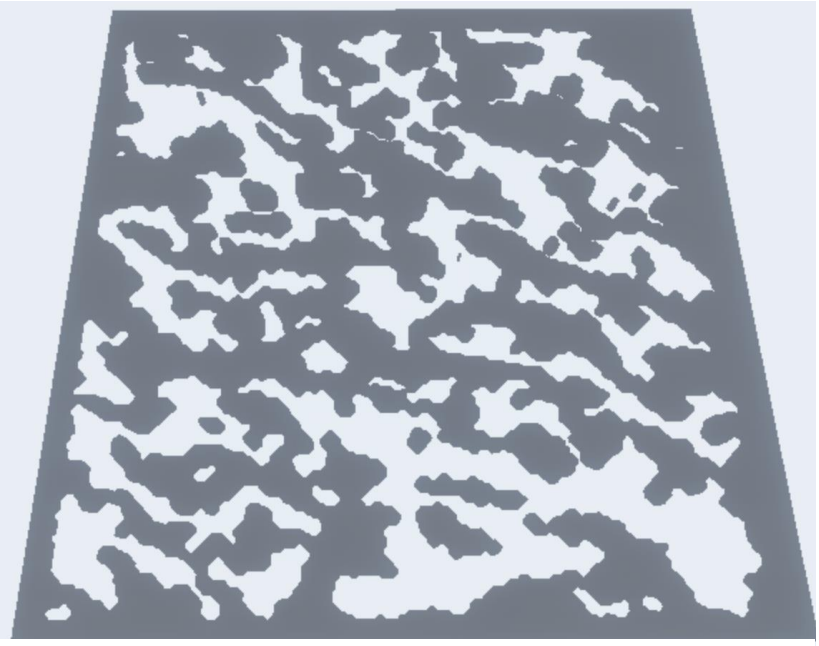

Figure 9: Prototype Cave Generation

The finished prototype consisted of map data generation being done by randomly selecting if a part of the grid was empty or a wall. This basic noise was smoothed using a simplistic version of Cellular Automata that made the data resemble caves. The Marching Squares algorithm then created vertices and triangle arrays that could be used by Unreal 4's *UProceduralMeshComponent* to create geometry in the scene. This resulted in a good representation of what the full application would be without a lot of the complexity by showing of simple versions of the main algorithms that would be used. The prototype demonstrated effectively that the techniques could work together and produce good looking results and showed that the project would indeed be feasible.

3.2.3 Marching Cubes Implementation

Once the project had been both designed and prototyped the full application was developed. The first step was to turn the Marching Squares algorithm that had already been implemented into the Marching Cubes algorithm. The main way these Marching algorithms work is they take data either generated or collected and translate it into geometry. In this case an isosurface is created and then used with Marching Cubes to create the cave environment. A simple set of test data was created for the initial development of the Marching Cubes algorithm. The pseudocode below illustrates the design and key steps of this implementation of the algorithm:

Marching Cubes Pseudocode

1. *Initialise grid of cubes.*
   1.1.Create and place vertex nodes.
   1.2.Create and place edge nodes.
2. *Get isosurface.*
   2.1.Find which nodes intersect issosurface.
   2.2.Calculate configuration 8-bit value based on node intersections.
3. *Polygonise data.*
   3.1. For each cube in cube grid.
   3.1.1. Create triangles based on cubes intersection.
   3.1.1.1. Check cube configuration in triangle look up table.
   3.1.1.2. Create vertices at nodes listed in look up table.
   3.1.2. Use vertices to create triangles.
   3.1.2.1. Use number of vertices to determine number of triangles.
   3.1.2.2. Add each set of 3 vertices to a tringle.
   3.1.2.3. Add each tringle to the triangles array.
   3.2 Create Normals and UVs.
4. *Render generated geometry.*

As illustrated above, there are four main steps to the algorithm. First, a grid of cubes containing the eight vertex nodes and twelve edge nodes was created. This was done by creating two three dimensional vectors; one for the vertex nodes and one for the cubes. Each vertex node object in the vector contained information about its position and if the node is active or not (intersecting the isosurface) as well as the edge node objects surrounding that vertex and their position and active values. Each cube object in the vector contained all twenty of its nodes, the edge nodes worked out from the information contained in the vertex nodes and the eight-bit configuration value of the cube. The second step is to pass in the isosurface data and for each node, check whether that node is inside or outside of the isosurface. Once the node's active value has been found the configuration of each cube is calculated. Each the vertex nodes counts as a bit in an eight-bit number here referred to as the configuration value. Based on a node being in or out of an isosurface its active variable is set either true or false which correlates

to either a one or a zero in the configuration number. After each of the vertex nodes are checked the configuration value can be created for a cube by taking each of the active nodes and adding them up. For example in the cube below vertices 0, 4, 5 and 7 intersect the isosurface. This would result in a binary number of 10110001 which equates to the value of 177 in decimal. This value is directly linked to a configuration of triangles constructed between these vertices.

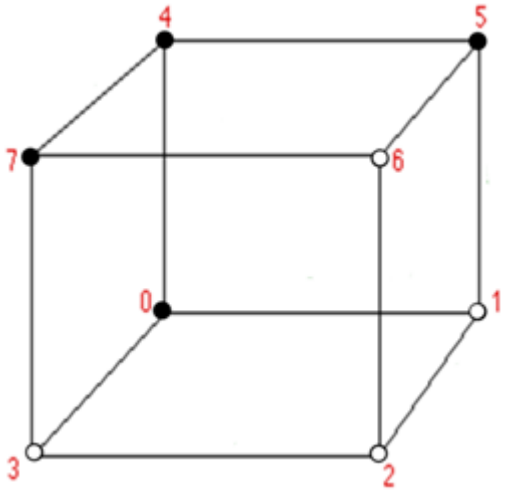

Figure 10: Node Example

Now that the grid of cubes has been created and the cubes configurations calculated using the isosurface, the third and most important step of translating the isosurface into geometry can be carried out. Herein the lies the issue, while in the 2D prototype there were only 15 possible configurations, in the full application with Marching Cubes there are 255. As mentioned in the literature review since a cube is symmetrical in every direction there are actually only 15 unique configurations. However, while this does make things easier there is still the issue of knowing what part of the cube the configuration of triangles needs to be drawn at. After looking into how others have implemented Marching Cubes it was observed that the majority used a look up table to check either the triangles or the edges of each configuration. This method also seemed the most appropriate for this project so a 256 by 16 table was created so that a configuration value could be input and lead you to each of the points in sets of threes that would give the vertices for triangles.

The data then needed be stored in a manner that could be rendered as a mesh. The way Unreal 4 handles the procedural creation of meshes is through a procedural mesh component that can be added to an object that when passed the correctly formatted mesh data will create a mesh attached to the object in the scene. Four *TArrays* (similar functionality to a *std::vector*) are needed to be passed to the procedural mesh component in order to render the mesh.

```
TArray<FVector> vertices;
TArray<int32> triangles;
TArray<FVector> normals;
TArray<FVector2D> uvs;
```

Figure 11: Procedural Mesh Data

In order to now create the mesh itself the cube grid is looped through and for each cube the configuration is checked against the lookup table. Each of the nodes listed at that entry contained the triangle look up table are added to a node array called *points*. Next the position of each node in *points* is added to the vertices array, then based on the number of nodes in the array between 1 and 5 triangles are constructed by using the index of the vertices. Since the vertices are added into the array in the order they are taken from the look up table it means they are retaining their correct order, so the triangle array simply needs to take the index of the vertices that were just added and it will have the correct vertices to draw a triangle, all in groups of three.

Once all the cubes have been looped through and the vertices and triangles placed in the arrays the normals and UVs can be generated. A normal is created for each tringle by taking the cross product of two of the triangles edges and then simple uvs are created using the normals. That then results in all four of the arrays that are needed to render geometry so all that is left is simply to pass these arrays from the Marching Cubes to the object with the procedural mesh component attached and give the component the arrays. Then once this object is placed in a scene it successfully generates geometry from the data it is given and render it in the scene.

3.2.4 Isosurface Generation

While Marching Cubes was now implemented and working with the test data it did not look very nice as simply adding a 3rd dimension to the prototype's test data produced a very strange and messy output (fig. 12). The isosurfaces required for the final application were instead created using a combination of Simplex Noise and Cellular Automata as these two techniques together were able to achieve natural and interesting looking caves whilst avoiding predictable and repeating patterns. The use of Simplex Noise also allowed for the project to utilise seeds so that by using the same inputs you could get the same results every time which is useful for both functionality and testing.

Figure 12: Strange Generation Using Prototype Data

3.2.4.1 3D Cellular Automata

Cellular Automata (CA) has been used for many different purposes over the years from reproducing real world patterns such as snowflakes and seashells to full games such as the Game of Life (Conway, 1970). These examples and the majority of uses of Cellular Automata differ from this project in one key aspect: they are usually 2D. While not being used as often for 3D as it is for 2D, CA is very good at creating cave like patterns in 3D. A 3D vector of integers was created to store the state of each cell. The CA algorithm was implemented as illustrated in the following pseudocode:

Cellular Automata Pseudocode

1. *Get inputs*
2. *Create map*
3. *Fill map*
   2.1 For size of map
   2.2. Randomly set cell start state
4. *For smoothing value*
   4.1. For size of map
   4.2. Get cell neighbours.
   4.2.1. Get surrounding cells
   4.2.2. Check if cell valid
   4.2.3. Increase wall count if cell has wall state
   4.3. Check cell against rule set
   4.4. Perform appropriate rule on cell
5. *Return filled smoothed map*

Firstly, the inputs needed to be passed into the algorithm, these are values such as the width, height and depth as well as parameters like the seed so that the algorithm could be easily customised and changeable. This is useful as it means that the algorithm is reusable, flexible and robust as well as meaning that variables can be opened up to the editor for quick and easy testing. Once the inputs have been received, a 3D grid of cells can be created using the width height and depth values for the size of the grid in the x, y and z dimensions respectively.

Once the grid was set up it could be filled with random starting cells that could be smoothed into cave shapes by CA. This is achieved by using nested *for loops* to loop through each cell so that a state can be applied to it. Initially the states were assigned by essentially flipping a coin giving a static like effect but this raised several problems such as an inability to reproduce results as well as unpredictable and greatly differing results that often didn't look very good. This eventuality was expected as it was predicted that a more complex solution would be required to achieve decent results. Simplex noise was chosen to solve this problem and will be discussed more in the next section.

In CA a cell can be any number of states and the algorithm could be made advanced and interesting with the use of many different states however for this project only two states were required: alive and dead. Each cell is either assigned a one or a zero; one being alive and zero being dead. These states are then used to tell where the walls of the cave are and where is empty space. Now that a grid of cells all with states has been created the cells can now be smoothed into more cave like shapes by changing the states of each cell based on the rule set. The smoothing value is customisable to allow for different results to be achieved and tested. The lower the smoothing the rockier and jagged the caves will be, the higher the smoothing the cleaner and carved the caves look. However, when the smoothing is too high you can end up with floating sections or possibly no geometry at all.

To smooth the cells nested *for loops* are once again used to loop through the grid and check each cell's neighbours using a Moore neighbourhood. This means that for each cell the twenty-six surrounding cells will be looked at. A wall count value is created and for each neighbour cell with a wall state the value is increased. Next each rule is tested based on the cells wall count. Three rules seemed to be the best way of approaching the problem: Death, Birth and Stasis. Then it was a case of getting these rules right but this proved problematic as getting the process took some time. This was because slight changes in the rules produced dramatically different results, some semi cave like and some very un-cave like, even producing strange puzzle box like shapes and rock like shapes.

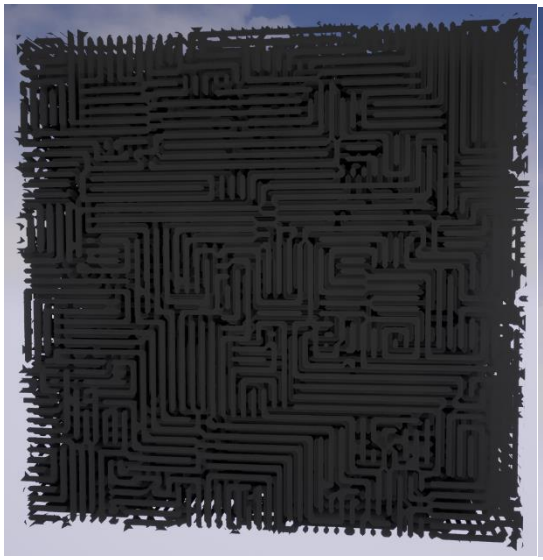

Figure 14: Puzzle Box Ruleset Result

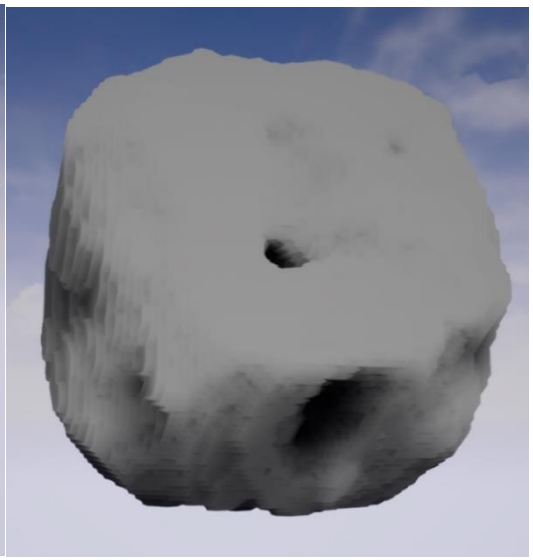

Figure 13: Rock Ruleset Result

The death rule meant if a cell was alive it would become dead, the birth rule meant that if a cell was dead it would become alive and stasis meant the cell would stay the same. The rule set details how many alive neighbour cells cause which rule to happen. After much testing and experimentation, the below rule set was found to produce the best-looking caves. It was mostly a matter of figuring out the ratios of values between rules as well as making sure not to generate too much or too little geometry. Now that the working ruleset had been found all that was left to do was convert the data to geometry using Marching Cubes. This was simply a case of passing the grid from Cellular Automata into the Marching Cubes algorithm instead of the prototype testing data

State	Death	Birth	Stasis
Cell	1,2,3,4,5,6,7,8,9,10,11,12,13	17,18,19	14,15,16,20,20,22,23,24,25,26

Table 1: Cellular Automata 3D Cave Ruleset

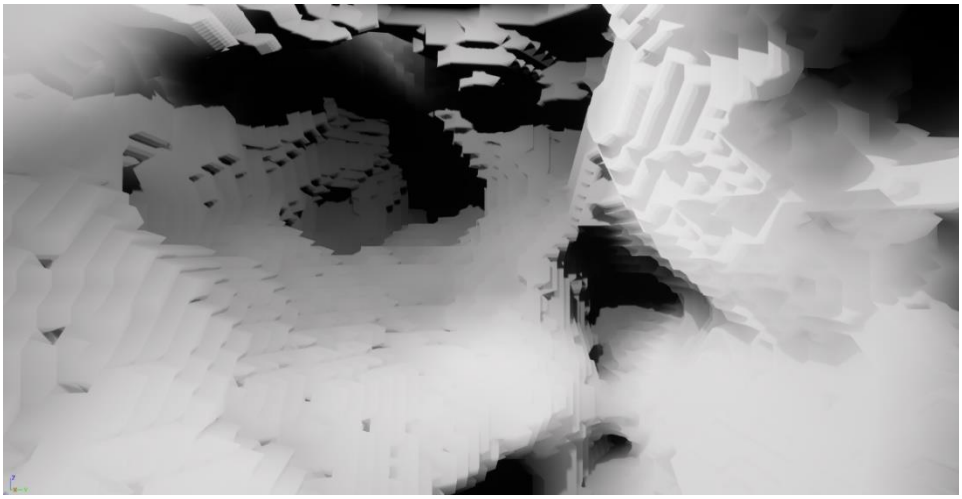

Figure 15: Marching Cubes and Cellular Automata

3.2.4.2 Simplex Noise

Although the completely random state assignment method worked to a degree, as discussed above it didn't produce the best results so a more complex solution was required. Simplex Noise, created by Ken Perlin in 2001, is similar Perlin's original noise but has several benefits over its predecessor. Some of these benefits include less directional artefacts and less computational overhead for multiple dimensions such as 3D noise which is what will be used for this project. Simplex Noise also has less complexity with, in big O notation where n is dimensions, $O(n^2)$ verses $O(n2^n)$ complexity for the original Perlin Noise. So, it seemed like using Simplex Noise would be a very good fit for this project as a

more complex but not too slow method for assigning states. So Simplex Noise was implemented to replace the old fill percent system and was designed so that it would be very customisable. This was because having high control over the noise will allow for more interesting results as well as being able to fine tune and change the noise with ease.

Once the application was implemented and all the separate parts were complete and working together a 100x100x100 size cube of cave could be generated quite quickly at an average speed of 2.52 seconds. A 200x200x200 cube could be created but would take much longer, taking on average 25.73 seconds. Much bigger than that however took far too long to generate and since the goal was to create an environment that could be used for a theoretical game some performance solutions and optimisations would need to be carried out if larger areas of cave were to be achieved.

3.3 Multi-Threading and Chunk Map

The first and most obvious solution was to use multi-threading and a chunk map system to divide the environment up into smaller segments called chunks that would each be rendered on their own thread instead of one after the other. This was carried out by creating a chunk manager that would handle the chunks and create and delete them. This chunk manager had a chunk width that defined how many chunks around the current chunk would be rendered at one time and a chunk map to store each of the chunks. In order to make sure each chunk is rendered the chunk manager would loop through each of the neighbouring chunks in the chunk map to the one the players in and generate any that hadn’t been generated yet. This was done by creating a queue of chunks and adding each of the un-generated chunks to the queue. The cube object at the front of the queue would pass its inputs by reference onto a thread that would then generate the isosurface and run Marching Cubes for that chunk. Each frame a render function would check all the chunks in the chunk map and the ones that have finished generating are rendered.

The map storage type was used to store the chunks around the player as well as chunks that have already been generated based on how far away the player is from them so that if the player went back to them quickly the chunks wouldn't need to be regenerated. Once the player goes too far away from the already generated chunks they will be deleted and removed from the chunk map and will need to be regenerated when the player returns to that chunk, but due to the work put into the noise generation the exact same chunk will be generated as the one that was created before since the inputs and seed are the same and the noise is offset based on the chunk position in the chunk map. This ensure that the chunks always lines up with each other and that they are always the same chunk when they need to be regenerated.

3.4 VR Movement

Movement is one of the biggest aspects to think about for any VR project. Due to the nature of this project, a lot of movement systems may not have worked because of the unevenness of the procedurally generated ground (fig. 16) and the nausea that occurs when a player directly moves vertically in VR. Some thought was put into designing an interesting approach to movement by using a mine cart to ground the player and moving that through the caves as grounding the player using a non-moving object has been shown to ease nausea induced through locomotion (Whittinghill, Ziegler, Case and Moore, 2015). This method however is outside the scope of the project so a simpler way to move around was needed.

Teleportation was chosen as the method of movement for several reasons. Most importantly since a player may teleport instantaneously from one point to the other without moving between the points, the unevenness of the caves was no longer an issue. Using teleportation is also useful as most people who have used VR before will be familiar with it and it is intuitive for those who haven't as the concept is familiar to most people. Teleportation is also a good fit as it is much less complex than many other methods that would have been appropriate such as the jog on the spot method used in Climbey (Lindenhof, 2016).

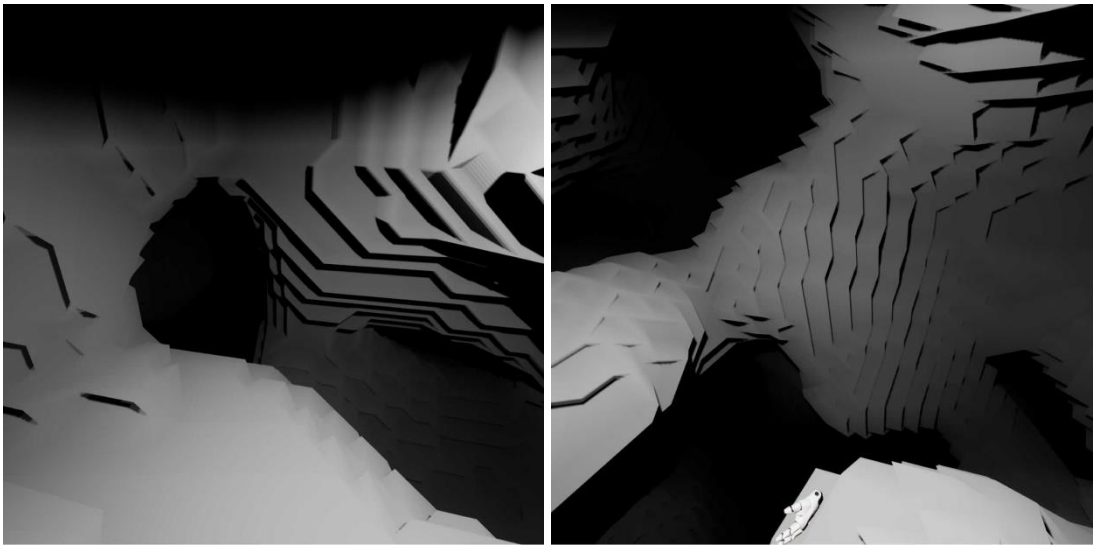

Figure 16: Cave Screenshots in VR

3.5 Gameplay Considerations.

As well as movement to accurately test if these kinds of procedural environments would work for VR games, a very basic level of interaction is required. While creating an entire game around the environment generation was not required some level of gameplay is needed, just enough to allow the player to explore and see the possibilities and uses of procedural VR environments. As discussed, a teleporter was implemented so that the VR player would be able to navigate through the caves and explore. Upon experimenting with the teleporter, as was the intention of this project the caves ended up being very interesting and fun to explore meaning that the exploration would be able to act as the core gameplay. To facilitate the exploration the rest of the experience was designed around encouraging it, by adding ambient music and giving the player a torch.

3.6 User Testing

To fully answer the research question of "How can Procedural Content Generation be used to create suitable environments for use in Virtual Reality?" user testing was carried out to gather data on how and if the generated environment works as the environment in a virtual reality game. This was done by using questionnaires to gather both quantitative and qualitative data. Collecting both quantitative and qualitative data was important as the former allows for direct correlations to be drawn from data and later allows for opinions as well as more direct and meaningful data to be collected. Quantitative performance data was also collected to analyse the performance of the application and record the benefits multithreading the chunk system had.

The testing process consisted firstly of the tester reading through the information sheet (Appendix B) that outlined the project, its purpose and the fact they could stop participating at any time. If the tester wished to participate after reading the information sheet they would next sign a consent form. Once the tester had been briefed and consented to take part in the testing they were instructed on how to use and put on the Vive and the controls were explained to them. After they were set up in the Vive they were free to explore the caves, which they could do for as long as they wished, usually between five and fifteen minutes. Once they were satisfied they had explored enough they were given a questionnaire to fill out (Appendix A). The questionnaire consisted of three sections, the first was a few questions intended to gather information about the demographic so that the data gathered from the tests could be compared by a range of factors like age group and whether they had used VR before. The second was designed to gather the bulk of the data to be used in answering the research question. It contained multiple questions about their experience that they could select a number representing how much they agreed with each question. The third section contained several questions that allowed testers to write in their own answers so as to gather more direct opinions and observations about the application.

4. Results

During the testing each participant filled out a questionnaire where they were asked some demographic questions, eight questions in which they could choose a number between one and five to answer the question and three in which they could write an answer. In the numeric answer questions, the one was a low answer and five was a high answer. Below are the results of these questionnaires. The written answers can be found in Appendix C.

4.1 Questionnaire Data

Q1	How immersed did you feel in the environment?
Q2	How easy was it to move around the environment?
Q3	How enjoyable did you find the environment?
Q4	How much did you like how the environment looked?
Q5	How much did you want to explore the environment?
Q6	How repetitive did you find the environment?
Q7	How likely would you be to play a game that used similar techniques to create the environment?
Q8	How much do you feel that VR game content would be improved by using techniques like this?
Q9	What was the thing you liked the most about the experience
Q10	What was the thing you liked least about the experience
Q11	Any other comments

Table 2: Reference Table of Each Question in the Questionnaire

ID	Age	VR	Headsets	Gameplaying	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
1	18-24	Yes	V	Everyday	4	4	4	3	5	2	3	4
2	18-24	Yes	V, R, M	Few Times	4	4	5	4	5	2	5	5
3	18-24	Yes	V	Everyday	3	5	3	5	5	2	5	3
4	18-24	Yes	V, O	Few Times	5	4	4	4	5	2	4	4
5	18-24	No	No	Everyday	4	5	4	5	5	2	5	5
6	18-24	Yes	P,R,M	Everyday	4	5	4	4	4	3	4	4
7	18-24	Yes	V,R,M	Everyday	3	4	4	2	5	2	5	4
8	18-24	Yes	V	Few Times	2	3	5	5	5	4	5	4
9	18-24	No	No	Everyday	3	3	2	4	3	4	3	3
10	18-24	No	No	Few Times	4	4	5	3	4	2	3	4

Table 3: Raw Questionnaire Data

This table contains the raw data taken directly from the questionnaires. The first notable thing that can be easily seen is that coincidentally all participants ended up being in the 18-24 age range. This is likely because it was university students surveyed and so are more likely to be in that age range. Next a question on whether or not the participant had used VR before. More people who took part had used VR before than people who had not. This is good as combined with the results of the next question on how often they played games, which showed everyone who took part played games either every day or a few times a week, it shows that the data gathered is close to what would be the target demographic of a game using this kind of technique e.g. someone who plays games frequently and has used VR.

The results of numeric section of the questionnaire are more clearly displayed on the below graph detailing the number of times each of the answers was chosen for each question. Overall the data seems to support the hypothesis that the procedural environment would encourage exploration and be an enjoyable experience. Most of the questions most frequently being answered with a 4 or a 5, both being on the positive side of the responses possible. The notable outlier is the Q6 where 2 is the most frequent answer. This however is due to a mistake in writing the question as the question was phrased in a way where lower numbers were good and higher numbers were bad, due to this mistake the numbers have been flipped in the graphs so that while still keeping the same results can more accurately be compared the other questions.

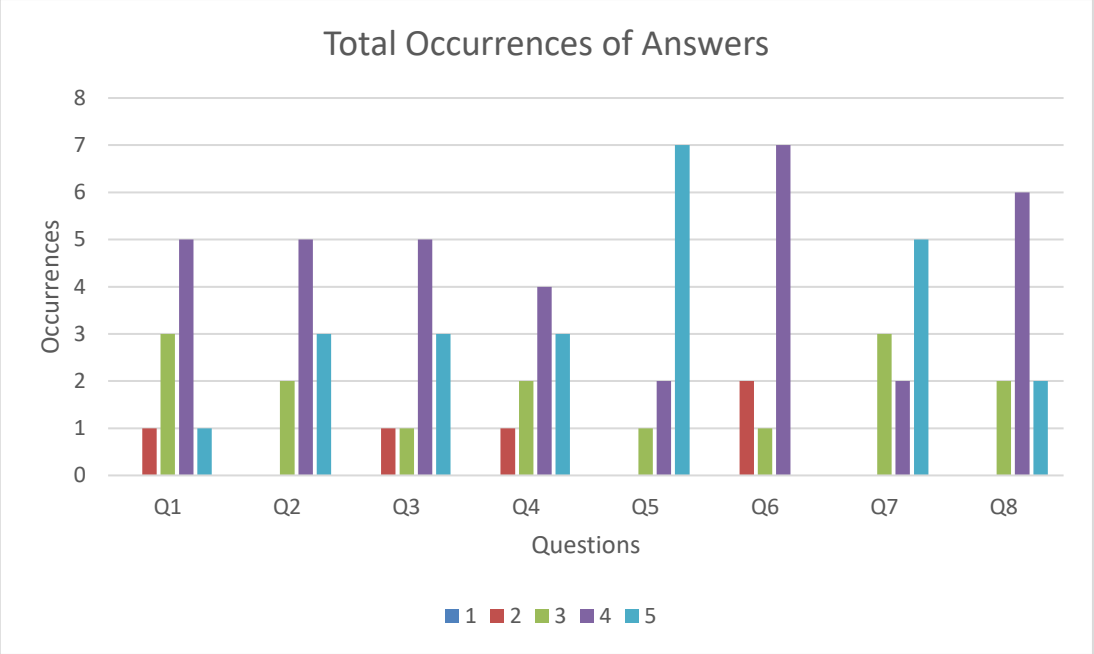

Figure 17: Total Occurrences of Each Answer in Each Question

4.2 Breakdown of Data

Breaking down each of the questions into its own pie chart shows even more clearly the distribution of answers and the overall feeling of participants about the project.

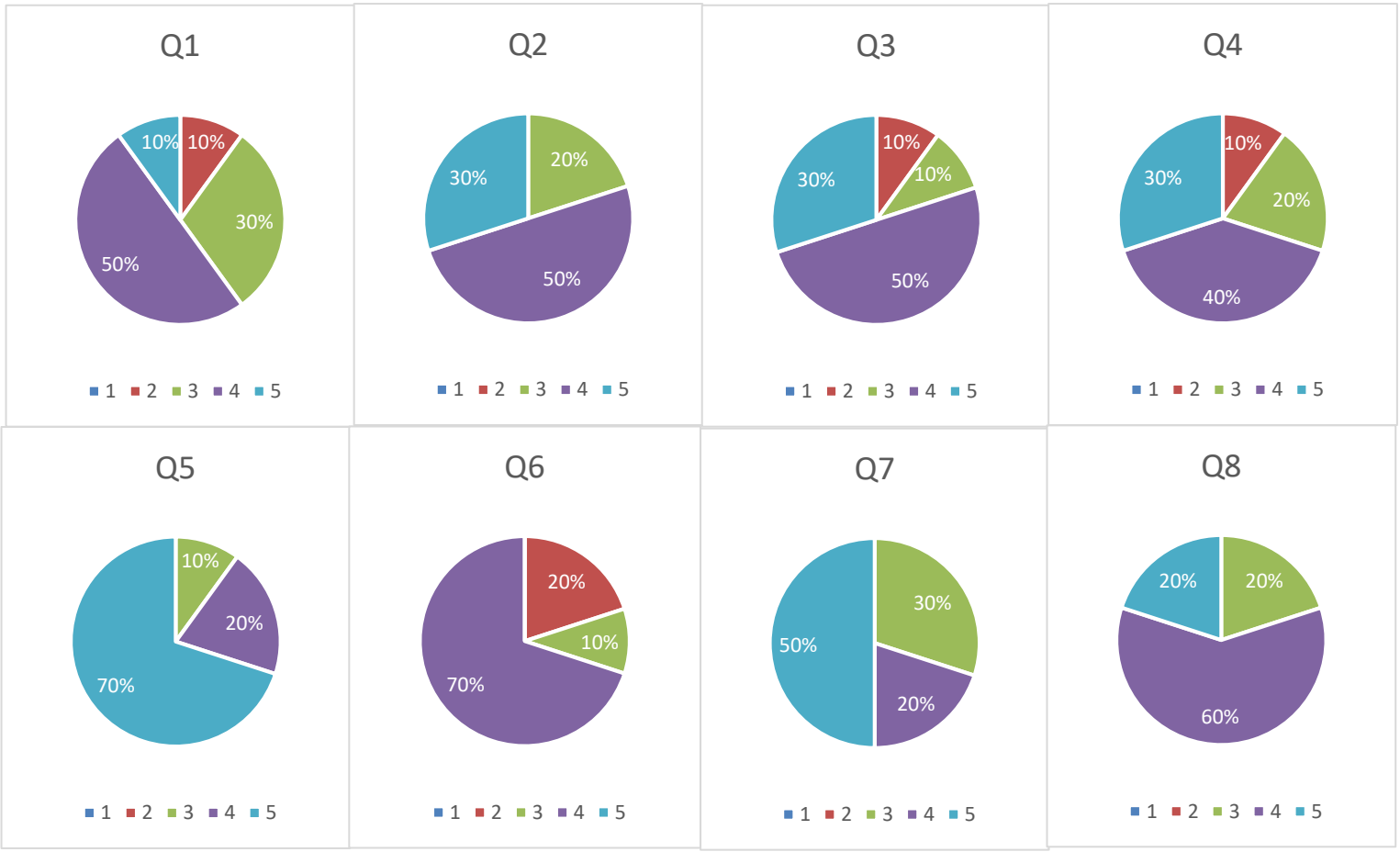

Figure 18: Pie Charts Showing a Breakdown of Each Question's Results

4.3 Comparison between VR Users

As this project is looking to see if using procedural generation is a viable way to create VR environments it is useful to see if having used VR before effects your opinion on the application. From the comparison graph below, it can be seen that there is a slightly higher average answer for VR over non-VR users however it is only slight higher by a very small margin and not on every question. The only question where the average is higher by a notable margin is Q5, which was the question about exploration. This means that on average people who have used VR before are more likely to want to explore the environment than those who have not. However due to the small sample size and the fact that only three participants had not used VR this may not be representative of a larger sample.

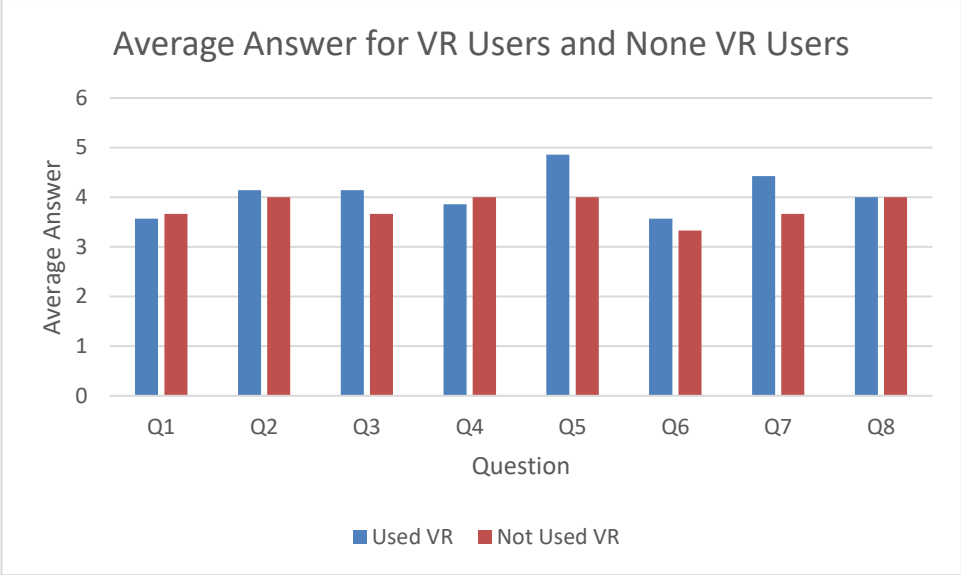

Figure 19: Bar Graph Showing a Comparison of Average Answers between VR Users and Non-VR Users.

4.4 Written Answers

The third section of the questionnaire consisted of three questions that a response to the question could be written in a box below: Q9, Q10 and Q11. The responses to these questions can found in Appendix C. The main trends that can be observed from these responses are firstly that for Q9, the majority of the comments where about enjoying the exploration. For Q10 most of the negative comments were talking about teleporting into the walls and only one comment stating that they didn't feel like exploring the caves. For the final question Q11 most people left it blank but the ones who didn't mostly left positive or encouraging feedback such as "Nice job" and "lots of potential for cool features" as well as comments with suggestions such as "Adding features for the player to mark their way through the level would add a lot to the experience.".

4.5 Performance Data

To test the performance the same inputs where given to four different test cases. The first three test cases being a single chunk and the fourth used multiple multithreaded chunks. The first chunk was 100x100x100, the second 150x150x150 and the third being 200x200x200. The time it takes to generate the chunks increases exponentially as the size increases which is to be expected. The fourth test case used twenty-seven multi-threaded 50x50x50 chunks.

Sample	100^3	150^3	200^3	(50^3)*27
1	2.62	9.04	29.63	7.35
2	2.55	8.62	27.59	7.46
3	2.48	9.37	25.14	6.68
4	2.56	8.74	25.41	6.78
5	2.53	9.01	24.63	5.27
6	2.51	9.18	25.38	6.43
7	2.54	8.54	24.60	6.43
8	2.54	8.88	24.39	7.50
9	2.41	8.49	24.35	8.41
10	2.45	8.50	26.22	7.08
Average	2.52	8.84	25.73	6.94

Table 4: Performance Results

The multi-threaded chunks are equivalent to 150x150x150 in combined size. So, comparing the results of the multi-threaded chunks to the second single chunk test case gives a good estimate of the differences in speed between the two. As the results show on average the multi-threaded chunks are around 2 seconds faster. The breakdown of the speeds of all 27 chunks for each sample can be found in Appendix D.

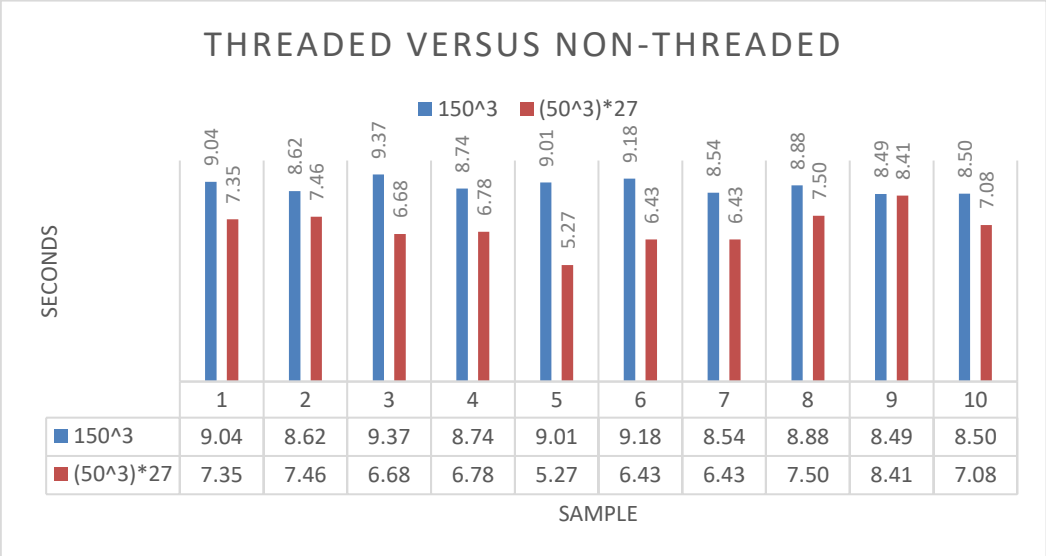

Figure 20: Comparison of Multithreaded Performance

5. Discussion

Throughout this section the results of the testing as well as the overall development of the project will be analysed. The project as a whole will be examined on how well it was able to answer the research question and aims as well as how the results of testing fitted with the hypothesis that players would enjoy a procedural VR game and be encouraged to explore by the unique environments.

5.1 Analysis of Testing Data

5.1.1 Numeric Question Analysis

Part 2 of the questionnaire contained the most important questions as they allowed for the opinions and feedback of the tester to be quantified and analysed. This allows for a direct way to see at a base level if the project was able to answer the research question effectively. The total occurrences of answers for each question (fig. 17) shows that there was a majority positive response to every question.

By using fig. 18 we can look at each of these questions individually. Q1 measured how immersed the player felt and it can be seen that players on average felt immersed with 50% of players reacting positively and 10% very positively. However, 30% of players only felt somewhat immersed and 10% did not feel very immersed. Overall this is a positive result as most players did indeed feel immersed though it is not as positive as expected. Most likely the bare bones nature of the application and occasional problem with players putting their head through walls lead to this slightly lower than expected result.

Q2 looked to see how players found navigating around the procedural environment. The results of the testing show that people found navigating procedural environments easy and straightforward with 70% of people reacting positively or very positively and no one reacting negatively to the question. This is very promising as movement is a key issue in VR and would be a big problem if procedural environments caused issues with movement as it would make them

unsuitable for use within VR. Luckily moving around the environment proved not to be an issue.

The third question considered if the players enjoyed the experience or not. Knowing this is important, because if a games feature or its environment isn't fun then that's a fundamental problem for the game. However, from the responses it shows that even just the environment and no gameplay beyond having a light and exploring was in fact enjoyable even on its own.

The next question asked about how much players liked the look of the environment. This was to gather data on if people liked the look of the procedurally generated environment as virtual reality games benefit from visually pleasing worlds. Overall players enjoyed the appearance of the environment though the feedback was slightly less positive than other questions and thus demonstrates that the visual side of the application would need some work to be used as part of the game. However, there are a multitude of ways that the aesthetics and visuals could be improved for a real VR game such as procedural texturing of the environment, placement of objects and flora throughout the caves and other such methods to enhance the look of the cave without changing the generation itself.

The next two questions, Q5 and Q6, look at how much people wanted to explore the caves and how repetitive the caves were. These two questions are linked as it was hoped that the repetitiveness of the caves would directly link to how much people would want to explore it. The reason for using procedural content is to make your game less repetitive and unpredictable so it was important to make sure that this was indeed the case. Q5 had the highest positive response of any question with 70% of answers being very positive and 20% being positive. Q6 also had a high rate of positive response with 70% being positive. From this it can be concluded that the procedurally generated environments in this case are interesting and different enough to greatly promote exploration without any other factors encouraging it.

The penultimate question was used to see how willing people would be to actually play a full game making use of this kind of environment generation. The very positive response to this question was again very encouraging as that indicates that after playing through the application people wanted to play more and see it fully fleshed out. The final question asked how much this kind of technique could improve VR games so as to get a direct response to the core of the research question. The feedback on this was very good as 80% of people thought it would improve VR games and nobody who tested responded negatively to the questions. This once again reaffirms the general response to the user testing, that people enjoyed the use of procedural generation and believed it worked for the experience.

5.1.2 Written Question Analysis

The third part of the questionnaire allowed for the participants to write out their thoughts in a more direct manor. The first of these questions was on positive thoughts they had about the application. Overall the written responses to this question greatly align with the hypothesis that people would be encouraged to explore the environment purely by the interestingness of the procedural generation. Almost every response to this question related to how enjoyable it was to explore the caves as well as how interesting the procedural generation was and that it encouraged them to explore. Such comments stating the best part of the application being that they "*really enjoyed the exploration and discovering strange parts of the cave*" and "*how unlikely you were to see the same thing twice*" reinforce the data from part two that shows just how much the players liked the generation and the exploration and completely support the hypotheses. The immersion of the cave system also came up a few times as being the thing testers liked most with comments such as "*the different levels of heights and sizes of areas really made me feel like I was in a cave system*" showing the procedural caves manage to capture the feeling of being in caves and underground really well.

The second question asked about what the players disliked about the experience to see what could be improved about the application. Most of the responses to this question described problems relating to teleporting and then being half in a wall. This is very much a valid issue that was not predicted but arose because of a combination of procedural generation and virtual reality. What these comments are describing is that if

you teleport too close to a wall you could end up standing inside the wall. This happens as the teleportation system teleports your VR play space not just you so if the player is standing near the edge of their space even if they teleport to beside a wall they could end up inside of it. This is a big issue and something that was not anticipated however it does not result in the complete discrediting of procedural generation for VR as it is a problem that can be fixed. This feedback highlights that there are a lot of factors that make creating VR content different from regular content that need to be considered and these factors are not always obvious at first.

The final question in the section and in the questionnaire, was if there were any other comments. This was so that if the player wished to give feedback on something that wasn't covered by the questionnaire so far, they could. Half the participants left this section blank but the ones who did fill it out left either encouraging statements such as "nice job" or useful advice like suggestions of future gameplay features. This was good to see as it meant it got people thinking about how procedural generation could work in the context of VR gameplay and showed that they saw a lot of potential in the project.

5.1.3 Performance Analysis

After the user testing, performance testing also took place. While the performance was less important for answering the research question and only needed to be good enough to run the game well, it is still useful to document and examine the performance of the application. During development it was identified that multi-threading was required to meet acceptable performance and generation speeds and this breakdown of the different performances of single large chunks and smaller multithreaded chunks demonstrates just how necessary the multithreading was.

The results show that on average the multithreaded test case was 2 seconds faster than the single large chunk of the same size. Which is a great deal faster in terms of having to generate chunks at runtime. It is important to note that 27 chunks are only loaded in at the start and usually only 9 chunks are generated when moving around in the game. While the performance is definitely improved over the original single large chunks, the generation could benefit from being slightly faster as on a non-high-end machine the performance would be a lot worse.

5.2 Project Findings

5.2.1 Summarised Results of Project

Overall the results of this project were very positive, while not an outright success on all counts, the research carried out shows huge potential and if persisted could amount to be a valid answer to some of the problems with VR games. The techniques employed in this project have been shown to be a feasible way to achieve diverse, interesting and enjoyable environments for VR.

5.2.2 Research Question

The research question that this project set out to answer was “How can Procedural Content Generation be used to create suitable environments for use in Virtual Reality”. The results of this project show that by combining appropriate 3D procedural generation with considerations for the VR movement, gameplay and performance, environments that not only work well for VR but also allow for a unique sense of exploration and discovery can be achieved.

5.2.3 Real World Applications of Research

In terms of actual uses for procedural VR environments while their use won’t solve all of VRs content problems, there is certainly a lot to be gained using procedural generation for whole environments or even just parts of an environment. As established in this project, when implemented with VR in mind, procedural environments can add a scale and unpredictability to a game’s world that would take a large amount of time to achieve manually by artists. Creating game worlds that are different every time greatly increases replayability, which is especially important in VR due to many games’ short playtimes. If a game were to use techniques similar to those laid out in this project, some very interesting and fun games and projects could be achieved.

5.3 Critical Evaluation of Project

5.3.1 Development

Though the result was a success, the project itself hit several speed bumps throughout development. Far more time was spent on optimisation and performance than was expected and thus none of the stretch goals described in the design document such as procedural textures and interspersing objects like rocks and flora throughout the cave system were able to be included in the project. The teleport system used to move around could benefit from some improvements as the main criticism of the application was players would sometimes teleport and appear in a wall. This could have been remedied by adding a collision check when teleporting to make sure none of the players VR play space is intersecting with the wall. It also would have been good to have some more gameplay features so as to investigate procedurally generated environments VR games more effectively.

There are also areas where the application itself could be improved. Elements like performance, while it was brought to an acceptable point, could still be greatly improved as the application would not run very well on a computer with lower specs than the build and test environment. It also would have been beneficial to the application if any form of texturing was done to the caves as though the standard grey texture does look fine it would improve immersion if there was some degree of diversity in the walls to break up the solid colour.

5.3.2 Testing

Having a larger and more diverse sample size when carrying out the user testing would have improved the effectiveness and impact of the results of this research. While the testing went well, it was carried out on students who were all in the same age range which is not a very diverse group, so the project would have benefited from a much larger more varied group to gain a better understanding of a wider range of people's opinions. The testing could also have benefited from testers also playing a non-procedurally generated VR game so that they could compare the experience and be asked further questions about the disparity between the two applications to provide further information of the benefits and weakness of procedural generation for VR environments.

6. Conclusion

6.1 Overall Conclusions

Throughout the development and testing of this project, the main conclusion that that can be drawn is that procedural generation works very well in VR as the environments are well suited to creating interesting environments that players want to explore. Procedural environments especially encourage natural exploration as it can invoke a fantastic sense of wonder and discovery, at least when using the techniques described and carried out in this project. It also shoes that using procedural generation is definitely a viable way to improve and increase the content of a VR game. That being said it certainly is not the only way and many VR games may not benefit from procedural environments due to incompatible gameplay or inappropriate movements system as they may limit or completely counter the effectiveness of procedural environments for VR. Overall, though not perfect and not the only solution, generating a VR game's environment procedurally can certainly improve a VR game's content and even add additional benefits and gameplay stemming from the unpredictability of PCG.

6.2 Implications

This research project serves as an example and case study that explores the benefits of creating a VR game's environment procedurally. The methods and results explored throughout this paper can be used both as a demonstration that VR procedural environments can work, as well as an example and basis for those looking to create VR environments of their own. On top of this, as VR is still quite new there is a lot of research still to be done into what works and what does not and hopefully this project will be able to be used as a starting off point for further research into similar areas.

6.3 Future Work

There is much further work that could be carried out if this project were to be continued. Primarily building a real game around the environment generation would be a fantastic continuation for this project, as it would allow for further insight into what benefits PCG has for VR, especially when it comes to what kind of gameplay works well and what doesn't. Additionally, distinct kinds of environments and procedural techniques could be looked at to see if they achieve similar or better results, such as creating world terrain from noise and height maps or the creation of interconnected dungeon rooms procedurally.

Appendices

Appendix A – Questionnaire

Questionnaire

Part One: Circle the appropriate answer(s) to each question.

Age range:	18-24	25-34	35-44	45+
Have you used Virtual reality before?	Yes		No	
If yes which of the headset have you used?	HTC Vive	PS VR	Oculus Rift	Mobile VR (Gear, Cardboard, Daydream etc.)	Other
How often would you say you play video games?	Every day		A few times a week	Occasionally	Never

Part 2: Circle one number between 1 and 5 where 1 is low and 5 is high.

How Immersed did you feel in the environment?	1	2	3	4	5
How easy was it to move around the environment?	1	2	3	4	5
How enjoyable did you find the experience?	1	2	3	4	5
How much did you like how the environment looked?	1	2	3	4	5
How much did you want to explore the environment?	1	2	3	4	5
How repetitive did you find the environment?	1	2	3	4	5
How likely would you be to play a game that used similar techniques to create the environment	1	2	3	4	5
How much do you feel that VR game content would be improved by using techniques liked this?	1	2	3	4	5

Part 3: Please answer the questions in the box below.

What was the thing you liked the most about the experience or any positive thoughts?

What was the thing you liked the least about the experience or any negative thoughts?

Any other comments?

Appendix B - Information Sheet

Information

This project is an investigation into the use of procedural content generation for creating virtual reality environments.

If you choose to participate in this research you will use the HTC Vive to explore a procedurally generated cave system. After you are content with the amount of time you have played you can then fill out a quick survey. The full process will take around 10 minutes.

The questionnaire is anonymous and the data collected will solely be used in the dissertation about this project.

You may stop participating in this research at any time and when in the Vive can remove the headset and exit the game at any time without needing to give a reason.

This is not a complete experience or game but simply a research project meant to demonstrate and test the combination of Virtual reality and Procedural Generation.

Upon finishing reading this sheet if you wish to participate you may now sign the consent form and begin.

Appendix C – Written Responses

ID	Responses to Q9 (Any Positive Comments)
1	Exploration of the Cave network, trying to reach certain points in the caves
2	Exploring and the way the environment looked
3	how unlikely you were to see the same thing twice
4	whenever I try a VR experience I am always instantly immersed in the environment so using this and procedural generation makes for a unique experience every time
5	the different levels of heights and sizes of areas really made me feel like I was in a cave system
6	exploring stuff
7	Moving through the cave was very enjoyable. It made me want to continue exploring the environment
8	I like the idea of the procedural level generation it seems very useful to make games less repetitive
9	being near edges of holes and sharp drops gave me a sense of unease
10	really enjoyed the exploration and discovering strange parts of the cave

ID	Responses to Q10 (Any Negative Comments)
1	points where I could see through walls, occasional weird jittering
2	getting stuck in walls, teleport arrow too bright, environments could be brighter
3	not much to encourage you to explore
4	the only thought is that sometimes after a teleport the player's head is inside the wall making it slightly confusing
5	sometimes teleport into walls
6	other than the environment there was nothing else to explore
7	the clipping/bugs and the open spaces of the cave were annoying
8	There were some bugs. I would have also liked to see a way to re-generate levels immediately in game
9	
10	seeing the inside of the walls was disorientating

ID	Responses to Q11 (Any Other Comments)
1	lots of potential for cool features
2	
3	
4	
5	
6	nice one
7	Adding features for the player to mark their way through the level would add a lot to the experience.
8	good job
9	only rated some parts 2 answers e.g. immersion and repetitiveness as I did because of how bare bones the test is right now, fleshed out I believe it could make for a good system but I have nothing to compare as I have never played VR
10	

Appendix D – Chunk Generation Performance Data

Time in Seconds

Chunk	1	2	3	4	5	6	7	8	9	10
1	0.594	0.559	0.478	0.465	2.098	0.548	0.521	0.492	0.456	0.517
2	0.827	0.573	0.63	1.875	2.15	0.55	0.522	2.152	2.423	0.595
3	1.231	0.628	0.896	2.163	1.586	0.663	0.826	1.938	2.936	1.611
4	1.387	1.439	0.717	2.351	1.555	1.293	1.31	2.302	3.026	2.026
5	2.116	1.546	1.303	2.441	1.679	1.235	1.311	2.705	3.27	2.014
6	1.976	1.329	1.531	2.871	3.147	1.401	1.582	2.531	3.586	2.375
7	2.572	2.065	1.834	2.846	3.309	1.453	1.882	2.745	4.264	2.384
8	2.737	3.03	1.981	3.105	1.966	2.093	2.112	3.244	4.49	2.82
9	2.856	3.383	2.049	3.334	3.537	2.147	2.379	3.314	4.485	3.16
10	3.061	3.638	2.504	3.48	3.481	2.682	2.433	3.501	4.972	3.26
11	3.618	3.669	2.659	3.633	3.747	2.895	2.551	3.52	5.506	3.442
12	3.515	4.232	2.963	3.83	3.674	3.131	2.684	3.972	5.659	3.712
13	3.848	4.549	3.068	4.012	3.778	3.565	2.777	3.894	5.651	3.625
14	4.035	4.562	3.213	4.213	3.91	3.686	2.75	4.458	5.798	3.598
15	4.414	4.69	3.541	4.439	4.073	3.794	3.057	4.378	5.752	3.962
16	4.609	5.047	3.691	4.466	4.351	4.163	3.109	5.026	6.043	4.39
17	4.812	5.228	3.934	4.889	4.483	4.296	3.104	4.8	5.992	4.605
18	5.031	5.542	4.583	5.364	5.112	4.705	3.202	5.489	6.617	4.68
19	5.116	5.915	4.618	5.511	5.271	4.785	4.947	5.302	6.505	4.973
20	5.283	6.128	4.919	5.771	3.643	4.911	5.409	5.468	7.125	5.206
21	5.557	6.469	5.278	5.749	5.431	5.038	5.385	5.636	7.468	5.265
22	5.602	6.476	5.292	5.825	3.558	5.229	5.246	6.166	7.475	5.928
23	6.59	6.891	5.643	5.856	3.614	5.924	5.922	6.199	7.762	6.072
24	6.485	7.163	6.234	5.984	4.722	5.81	6.087	6.606	7.661	6.489
25	6.727	7.455	6.13	6.451	3.808	5.721	6.044	6.982	7.9	6.527
26	7.346	7.458	6.231	6.539	4.258	6.273	6.433	6.939	8.409	6.628
27	6.892	7.267	6.675	6.78	4.341	6.387	6.419	7.5	8.364	7.082

A chunk's time is the time since the start of the generation, not since the last chunk.

List of References

Addams, T (2006). *Dwarf Fortress.* [Video Game]. Bay 12 Games.

Adebayo, T (2016). *We Took The HTC Vive VR For A Spin, Is This The Future Of Gaming?* Available at: https://www.unilad.co.uk/technology/we-took-the-htc-vive-vr-for-a-spin-is-this-the-future-of-gaming[Accessed: 20 October 2017]

A.I. Designs (1980) *Rogue* [Video Game]. Epyx.

Beneath Apple Manor (1978). *Beneath Apple Manor*[fig. 2]. Available at: https://crpgbook.wordpress.com/review-index/1978-beneath-apple-manor/ [Accessed 20 March]

Bourk, P (1994) Polygonising A Scaler Field. Available at: http://paulbourke.net/geometry/polygonise/ [Accessed: 3 November 2017]

Brooker, C (2011). Black Mirror. [TV Show]. Zeppotron.

Cline, E. (2011). *Ready Player One*. Random House

Conway, J H (1970). Game of Life [Video Game]. Conway, J H.

Crows Crows Crows; Squanchtendo (2016) *Accounting* [Video Game]. Crows Crows Crows.

Durbin, J. (2016). *VR/AR's biggest obstacle: Lack of content.* Available at: https://venturebeat.com/2016/09/18/vrars-biggest-obstacle-lack-of-content/ [Accessed: 12 October 2017]

Epic Games. (2014). *Unreal Engine 4* [Software]. Epic Games

Indimo Labs LLC (2016) *Vanishing Realms* [Video Game]. Indimo Labs LLC.

Iwaniuk, P. (2016). *How, why, and when VR will fail.* Available at: https://www.pcgamesn.com/how-why-and-when-vr-will-fail?amp [Accessed: 20 October 2017]

Lang, M (2014). *Virtual Boy Retrospective Nintendo's disastrous foray into VR*. Available At: http://www.digitalspy.com/gaming/retro-gaming/feature/a562419/virtual-boy-retrospective-nintendos-disastrous-foray-into-vr/ [Accessed: 19 October 2017]

Laprairie, M, Hamilton, H (2017). *Marching Cubes Configuration* [fig. 5]. Available at: https://www.researchgate.net/figure/Look-up-table-for-the-Marching-Cubes-algorithm-13_fig2_228454597

[Accessed 20 January]

Lindenhof, B (2016) *Climbey* [Video Game]. ShadowBrain Games.

Lorensen, W.E. and Cline, H.E. (1987). 'Marching Cubes: A high resolution 3D surface construction algorithm.', *Computer Graphics*, 21(4), pp. 163-169.
doi: 10.1145/37402.37422
Mojang. (2009). *Minecraft* [Video Game]. Mojang.

Owlchemy Labs (2016) *Job Simulator* [Video Game]. Owlchemy Labs.

Perlin, K. (1985). 'An Image Synthesizer', *Computer Graphics*, 19(3), pp. 287-296.
doi: 10.1145/325165.325247

Phr00t's Software (2017) *4089: Ghost Within* [Video Game]. Phr00t's Software.

Rogue (1980). *Rogue*[fig. 1]. Available at:
https://static.giantbomb.com/uploads/original/1/15568/537945-rogue_006.png
[Accessed 12 October 2017]

Scorpia. (2018). *1978 – Beneath Apple Manor*. Available at:
https://crpgbook.wordpress.com/review-index/1978-beneath-apple-manor/
[Accessed 15 February 2018]

Shiffman, D. (2012). *The Nature of Code.* Available at:
www.natureofcode.com/book/chapter-7-cellular-automata/
[Accessed 28 January 2018]

Sahillioglu, Y. (2008) *Marching Rhombic Dodecahedra.* Available at:
https://www.cise.ufl.edu/~ysahilli/vis/ [Accessed 9 March 2018]

Shaker, N. (2016) *Procedural Content Generation in Games.* Springer International Publishing: Springer.

Togelius, J., Kastbjerg, E., Schedl, D. and Yannakakis, G. (2011). *What is procedural content generation?* pp. 1.

The Sword of Damocles(1966). The Sword of Damocles [fig.7] Available at:
http://www.blakesleeadv.com/wp-content/uploads/2017/01/http-2F2Fmashable.com2Fwp-content2Fuploads2F20122F092Fthe-sword-of-damocles.jpg [Accessed: 15 October 2017]

Trickster Games (2016). *Trickster VR: Co-op Dungeon Crawler*. [Video Game]. Trickster Games.

Tron (1982) Directed by Steven Lisberger [Film] Burbank, Calif: Buena Vista Distribution.

Weinbaum, S G (1935). *Pygmalion's Spectacles*. Kessinger Publishing.

White Door Games. (2017). *Dreadhalls*[Video Game]. White Door Games.

Whittinghill, D.M., Ziegler, B., Case, T. and Moore, B. (2015). *Nasum virtualis: A simple technique for reducing simulator sickness.* Games Developers Conference (GDC).

Worth, D. (1978). Beneath Apple Manor [Video Game]. The Software Factory.

Bibliography

Anderson, B (1994). *An Implementation of the Marching Cubes Algorithm.* Available at: http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html

Cui, J (2011). *Procedural Cave Generation.*

Mark, B, Berechet, T, Mahlmann, T, Togelius J (2015) *Procedural Generation of 3D Caves for Games on the GPU.*

Shaker, N, Togelius, J, Nelson, M J (2016). Procedural Content Generation in Games. Available at:http://pcgbook.com/