# Background and Related Work

## Introduction

This project aims to make use of procedural generation to create terrain and terrain effects using the DirectX library. The content discussed within this chapter will be around the related topics surrounding this subject and will be split into three sections. The first section will delve into what procedural generation is and why it is used in the creation of video games within the games industry and will discuss the advantages and disadvantages of using this method. The second section will go into depth about the various techniques that can be used to apply procedural generation. It discusses various methods for generating the terrain procedurally and will look at how this was done early on within the industry and compare how these methods stand up to the newer ones introduced that are currently being used. The final section will go into an evaluation on the end user and provide information of what the end user thinks of the application of procedural generation within games and how to keep the user invested while using the project.

## What is procedural generation and why is it used within games?

Procedural generation is a method for creating data algorithmically, without too much input from the creator or developer. Procedural generation is normally used to create content for video games or for animated movies, such as landscapes, animation of 3D objects.

Procedural content generation has become a more frequently used method within video games over the past couple of years, furthermore it has also seen a use in the early history of the industry in such games as *civilisation* (Sid Meier, 1991) and *Rogue* in the 80’s. Video game developers are increasingly using procedural generation for everything within their game; however, the focus has been in creating visually appealing content for the user while using less development resources.

Hendrikx et al (2013) talks about how procedural generation can be used to fight the “new scalability challenges due to the exponential growth over the last decade” . Video games have been increasing in their popularity year over year as reported in the annual report released by the Entertainment Software Association (Entertainment Software Association, 2021) that claims in the United States there are nearly 227 million people playing video games. Due to the increasing popularity of games, the games industry has become very competitive in the development of triple A games, leading them to become larger and more expensive, allowing procedural generation to provide a practical method at reducing these costs. The main cost that is reduced is the time and effort taken to create levels and the artwork associated with these levels. The use of procedural generation allows many similar levels to be created with as few art assets created as necessary. By having an artist create a handful of resources that can be reused by the procedural algorithm allows for lower costs (Green, 2016).

While this method provides several benefits such as speeding up and reducing the costs of video game development there are several downsides to using this method. One of the largest downsides is the requirement of optimal high-end hardware to consistently generate content efficiently. Creating games with procedural generation is done through algorithms that can be intensive and require a lot of computing power (Green, 2016). Due to this, more effort needs to be put into optimising these algorithms so that the game created can be played by anyone and does not require a high-end computer. Another big downside of procedural generation is that the content created can begin to start feeling repetitive. When creating large worlds but only a few basic algorithms are used will inevitably create a lot of repetitive areas being generated (Green, 2016).

### Generation of plants and animals

A picture containing tree, sky, outdoor, plant

Description automatically generatedFreiknecht (2020) discusses how the procedural creation of objects occurring in nature belongs to one of the most explored areas within procedural content generation. Plant recreations within graphical programs have been done extensively for the past several decades, even as early as 1968 where *L-Systems* (Lindenmayer Systems ) were introduced to describe the structure of plants (Freiknecht, 2020). L-Systems are created by using a system of production rules that are then applied to a chosen collection of symbols that usually describe how a plant grows. The rules applied to L-Systems can be as extensive as required. Figure 1 shows how L-Systems can be utilised over several iterations to produce a realistic looking plant. Fornander (2013) describes how a third-party plug-in called SpeedTree that implements L-Systems is used for the procedural creation of vegetation, some examples include *Resident Evil Village* (Capcom, 2021) and *Ghost of Tsushima* (Sucker Punch Productions,2020*).*

Figure – several plants generated with stochastic L-System (Freiknecht, 2020)

Graphical user interface

Description automatically generatedFreiknecht (2020) states that research on procedurally generation of creatures is “few and far between” (p. 29). However, a video game called *Spore* (Maxis, 2008), introduced game-play featuring the ability for players to create a microscopic organism that could develop into a highly intelligent creature. Designer Will Wright proposed the approach of using procedural generation to create thousands of assets that would be needed when developing the game, Figure 2 shows this method in effect as the player is able to change the colour and tecture of the creature to anything that they would like and the new assets are created during runtime. Freiknecht (2020) discusses how based on the development of *Spore,* Hecker et al, introduced a system to animate creatures with unknown body types during runtime by getting a set of variables from the creature generation and then translating them into a visual model and then animating it. A more modern game that uses procedural generation is No Man’s Sky (Hello Games, 2016), which uses this technique to generate trillions of planets and the creatures that are found within these planets. This reintroduces the problem of repetitive content when using procedural generation for the bulk of the content within a game, however, it is still an amazing technical feat.

Figure – Spore, an example of the creature creation screen (Beale, 2008)

Procedural generation of animals and creatures in video games has generally been about the placement of the creatures within the game world based on the parameters created when the in-game world was created, such as adding deer anywhere there are large areas of grassland or adding fish to areas of water. Furthermore, the design of animals and creatures have largely still been left to the artists working on the game with the only part of the process that procedural generation can sometimes affect is the generation of textures needed for the animal object.

### Generation of levels and worlds

Procedural generation of content has almost always been used with the generation of dungeon levels within video games with some examples being Diablo (Blizzard Entertainment, 1998) and Don’t Starve (Klei Entertainment, 2013), one of the biggest reasons for the use of this method is the replayability it provides to the end user. The use of this method allows for the reduction in production costs as described earlier in this chapter and it can increase the creativity of the designers, moreover, allowing them to focus on other aspects of the game instead of level design.

A picture containing text

Description automatically generatedOne method of procedurally generating dungeon levels is through Cellular Automata, which is a grid of cells that reference their neighbours, rule sets are then applied to the current state of the cell along with its neighbouring ones. Patterns will begin to form after several generations/iterations through the grid (van der Linden, Lopes, & Bidarra, 2014). This method allows for the easy manipulation of each cell through parameters and constraints, an example would be checking whether all the neighbouring cells are empty and assuming that the current cell is empty as well. Figure 3A shows what the result of Cellular automata generates based on certain parameters and how it resembles what a dungeon level would look like with the grey areas being the level floor and the white area representing the walls of the level.

Figure A – Map Generated using Cellular automata (van der Linden, Lopes, & Bidarra, 2014)

The procedural generation used in *dwarf fortress* (Bay 12 Games, 2006) has not been explicitly explained by the creators of the game, however, based on Figure 3B, it would be reasonable to assume that cellular automata could have been used when creating the game world as it looks very similar to the map in Figure 3A. Furthermore, dwarf fortress has also used procedural generation for the creation of the history and religions of the generated world, with these being completely different each playthrough of the game. This goes to show that procedural generation can be very helpful in the creation of video games as dwarf fortress started out as a two-person indie project and the use of procedural generation allowed them to focus on other aspects of the game required for them to release the game.



Figure 3B – Dwarf Fortress an example of procedural content generation of worlds (Grant 2012)

## Procedural Generation Techniques

### Introduction

Procedurally generated content has been implemented in dozens of different games in a variety of ways throughout the past couple of decades, with No Man’s Sky’s’ entire concept, relying on the procedural generation of planets and creatures and dwarf fortressusing procedural generation to generate nearly all aspects of the game. This section aims to discuss the several different techniques that are used within the industry to create this content, how they can be applied to this project and whether they are practical to use. Techniques that will not be used will also be discussed and researched to provide a better understanding of the other techniques that could have been used within the project.

One of the most basic methods of storing terrain data are heightmaps, which are made up of a two-dimensional array with indices indicating the X and Z coordinates and the value indicating the Y value of the terrain. Mijailovic (2015) describes heightmaps as being fast, in terms of data access, and intuitive to use, going on to state that they “became the backbone of many algorithms” (p.7). However, Heightmaps have one big disadvantage of only storing one height value for each coordinate pair, leading to the generation of very simple terrain without any complex structures, such as caves or overhanging cliffs. Voxels – volumetric pixels - have been introduced as a way to counter this and leading to easier generation of complex terrain structures. Each method introduced within this next section, take heightmaps as an input and fill all the values within the heightmap with generated points.

### Outdated techniques used

#### Midpoint displacement

Midpoint displacement was one of the first algorithms introduced for the procedural generation of content in 1982. This method of procedural generation is implemented by splitting the image into four quadrants and adjusting each corner by a random value, each quadrant is then treated as a new image and the process is repeated, while this is happening the range to affect the corners is decreased each time a new image is identified (Snook, 2003). Figure 4 show each numbered point is set to the average of the two corners they are connected to and how this process is done recursively with every additional quadrant created. This method of procedural generation of terrain is very effective at creating random terrain and can have constraints added to allow the programmer to have more control of the structure created,

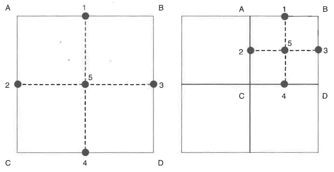
One problem that occurs during the use of the midpoint displacement method is that patterns can start emerging and can be very noticeable when looking at the terrain from different angles (Losh, Terrain Generation with Midpoint Displacement, 2016). While natural environments can have some repeating patterns, normally this is done through human involvement, it will be important to take into consideration the end user when making use of the midpoint displacement method, and repetitive terrain can be jarring or annoying for the end user to look at.

Figure 4 – stages of the midpoint displacement method. (Snook, 2003)

within the context of this project, this method of procedural terrain generation would not be effective when creating the terrain as it will become repetitive quickly and the recursive nature of this method can be computationally taxing when generating a large environment. On the other hand, this repetitive pattern is not guaranteed to occur due to the random nature of procedural generation, however, this is not enough to justify using this method within the context of this project.

#### Diamond square

Chart, line chart

Description automatically generatedThe Diamond square method is an improvement on the midpoint displacement method through ensuring that every point uses four sources of data (Losh, Terrain Generation with Diamond Square, 2016). This method is implemented similarly to the midpoint displacement method; however, diamonds are used instead of the square. This method will start out by giving the corners of the image random values and calculating the centre of the image from the average of these values. The algorithm will then iterate through the image and create diamonds to help with the generation of height values. Figure 5 shows an example of this algorithm working when given a 5x5 grid, and shows how the algorithm is also done recursively, similarly to the midpoint displacement algorithm.

Figure 5 – Example of terrain generation on a 5x5 heightmap (Mijailovic, 2015)

Miller (1986) performed an analysis on the diamond square method and found that this method of terrain generation was flawed due to the ‘Tell-tale vertical streaks’ indicating a persistent creasing problem due to the calculations taking place within a rectangular grid. Another big disadvantage of using the diamond square method is that non-repeating infinite terrain cannot be created easily (Mijailovic, 2015), as each heightmap would search for an additional heightmap, that doesn’t exist, next to it for lines near the boundaries to break.

The observation from Miller along with the recursive nature of the algorithm are the reasons why this method will not be implemented within the scope of the project. Despite this, the diamond square method is a major improvement on the midpoint displacement method when generating terrain without patterns appearing.

### Newer techniques used

Commonly used algorithms for the procedural generation of terrain are noise algorithms. These algorithms are commonly used for the generation of height maps that would then be used for terrain generation. These algorithms are great for creating a 2D height map describing how high everything in the environment should be, however, when trying to create areas with overhanging structures such as cliff faces, these 2D versions would not provide enough detail. 3D versions can be created; however, they are used for describing how densely populated areas are within the environment.

#### Perlin Noise

Perlin noise is the most used algorithm for the procedural generation of terrain, several game engines even have plug-ins or built-in implementations of this algorithm, such as Unity since it is the most popular method. Noise is the random unstructured number generator of computer graphics, and the random patterns are often described in terms of frequency (Lagae, et al., 2010). Due to the nature of this method, the amount of terrain that could be generated is nearly endless, while only changing certain parameters and constraints. Two different noise maps can be generated or constructed with this noise algorithm as mentioned earlier, height and density maps. For this project, the height maps are more important as the gradient of the terrain will be generated and the focus will not be on how populated areas are within the terrain.

To generate Perlin noise in 2 dimensions, a pseudo-random 2D normal vector – unit length pointing in a random direction - is placed on each grid point (Snook, 2003), as shown in Figure 6a. Snook (2003) goes on to describe how the height value for each point, within a grid square is calculated by getting the vector to the point for each corner of the grid and performing a dot product to get the resultant height value, Figure 6b demonstrates this. This is how Ken Perlin originally introduced this algorithm and it proved to be an effective way to generate random noise and has since been a very popular method to procedurally generate content. This project aims to implement the Perlin Noise algorithm this way to generate the height map for the terrain that will be generated.

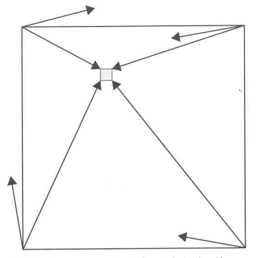
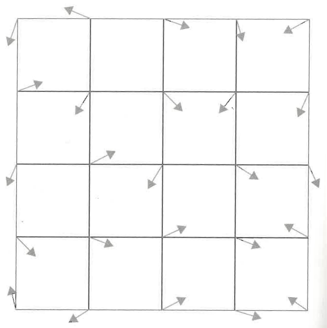


Figure 6B – Calculation of the height value for a pixel (Snook, 2003)

Figure 6A– set up of the Perlin noise function (Snook, 2003)

#### Simplex Noise

A picture containing text, mollusk, cowrie

Description automatically generatedSimplex noise was created in 2001 when Ken Perlin introduced it as an improvement on his noise algorithm to overcome certain limitations (Gustavson, 2005). Simplex noise involves the use of simplex grids - which is the simplest and most compact shape that can fill an entire space – to handle *N* dimensions. Gustavson (2005) tells us that simplex noise was designed to have a lower computational complexity with fewer required multiplications than Perlin noise. Figure 7 shows how Simplex noise compares to Perlin noise when generating noise within a 2D environment, showing how Simplex noise generates an image with more detail demonstrating clearer peaks and troughs while Perlin noise generates an image with less detail it does, however, show more variation. Moreover, the simplex noise generated an image with sharper gradients between the points, as seen with super dark spots surrounded by a white area. This could lead to the terrain being generated looking less realistic to the user.

Figure 7 – comparison between Perlin noise and Simplex noise - (Gustavson, 2005)

While simplex noise is less computationally complex, it was designed to handle a higher number of dimensions, allowing for the quick generation of large structures. Furthermore, the generation of sharper gradients between points within the noise image would be useful when generating large mountains with huge cliff faces which this project does not aim to do. An additional point is that the implementation of Simplex noise in 3 dimensions and higher is currently patented by Ken Perlin (United States Patent No. US6867776B2, 2001), with it currently being expected to expire in early 2022, I do not plan to take this risk when generating terrain in the project.

## User Evaluation

While this project is aimed to create realistic environments, it should be important to consider what would help engage the user. Adding water to generated terrains that produce valleys could be a way to increase the realism of the project. When producing procedurally generated content that would be used within video games, it should be important to consider what would be realistic within the given context, otherwise the player would get bored or uninterested due to the complete randomness of the environment with very little detail. A study found that out of a group of 41 people, 60% preferred Procedurally generated reefs compared to static reefs made by artists (Korn, et al., 2017), figure 8 shows an example of what the participants of the study were shown. This justifies the use of procedurally generated content within video games as the end users prefer the dynamically changing look of environments within video games, however, if the game has a detailed story and focuses heavily on this storyline, the use of procedural generation would be a drawback as it would take away from the story.

A picture containing text, ocean floor

Description automatically generated

Figure 8 – Reef structures designed by an artist (top) and generated procedurally by Perlin noise (bottom)

(Korn, et al., 2017)

Another important aspect of realistic environments when done through procedural generation is the use of texturing and visuals of the environment. If the same texture is used throughout the whole environment, especially in complex environments, the end user would become annoyed or uninterested to explore the environment as it would look the same everywhere, they go. End users are getting used to playing games that are visually appealing to them with exciting environments to explore due to the advancements in graphical technology, leading to higher quality and larger textures being produced.

This project will aim to create an environment that would be as realistic as possible without sacrificing the accuracy found within nature, furthermore, trying to create a realistic environment that is also fun to explore for the end user. A balance will have to be struck between these two points depending on what the project would be intended for. If this project were to be used for the generation of game levels or gameplay, artistic creativity would have to take priority compared to the realism of the generation. Although, if this project were to be used for the simulation of realistic environments, the accuracy of the environment would have to take priority over the artistic direction of the generation.

## Summary

With the information that has been researched and discussed within this chapter, I should be able to apply the appropriate methods and techniques within the scope of the project. The information found about the techniques that will not be used has helped considerably and will help towards important decisions throughout the development cycle of the project.

References

Beale, S. (2008). Spore Creature Creator. Retrieved from https://www.flickr.com/photos/laughingsquid/2568570641

Entertainment Software Association. (2021, July). 2021 Essential Facts About the Video Game Industry . Retrieved from https://www.theesa.com/resource/2021-essential-facts-about-the-video-game-industry/

Fornander, P. (2013). *Game Mechanics Integrated with a Lindenmayer System.* Karlskrona: Blekinge Institute of Technology.

Freiknecht, J. (2020). Procedural Content Generation for Games.

Grant, M. (2012). My first Dwarf Fortress World. Retrieved from https://www.flickr.com/photos/mylesdgrant/7689242534

Green, D. (2016). *Procedural Content Generation for C++ Game Development.* Packet Publishing Ltd.

Gustavson, S. (2005). *Simplex noise demystified.* Linkoping University.

Hendrikx, M., Meijer, S., Iosup, A., & Van Der Velden, J. (2013). *Procedural Content Generation for Games: A Survey.* ACM Trans. Multimedia Comput.

Korn, O., Blatz, M., Rees, A., Schaal, J., Schwind, V., & Gorlich, D. (2017). Procedural Content Generation for Game Props? A Study on the Effects on User Experience. *Computers in Entertainment*, 1-15.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. S., . . . Zwicker, M. (2010). A survey of Procedural Noise Functions. *COMPUTER GRAPHICS forum*.

Losh, S. (2016, June 27). *Terrain Generation with Diamond Square*. Retrieved from Stevelosh.com: https://stevelosh.com/blog/2016/06/diamond-square/

Losh, S. (2016, February 19). *Terrain Generation with Midpoint Displacement*. Retrieved from Steve Losh: https://stevelosh.com/blog/2016/02/midpoint-displacement/

M. F. Viana, B., & R. dos Santos, S. (2019). A Survey of Procedural Dungeon Generation. *18th Brazilian Symposium on Computer Games and Digital Entertainment*, pp. 29-38.

Mijailovic, V. (2015). *A Graph-Based Approach to Procedural Terrain.* KTH Royal Institute of Technology.

Miller, G. S. (1986). The Definition and Rendering of Terrain Maps. *ACM SIGGRAPH Computer Graphics*, 39-41.

Perlin, K. (2001). *United States Patent No. US6867776B2.* Retrieved from https://patents.google.com/patent/US6867776

Snook, G. (2003). *Real-time 3D terrain engines using C++ and DirectX 0 (1st ed.).* Charles River Media.

van der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions of Computational Intelligence and AI in Games*, 78-89.