

Proposal For Senior Project

Brandon L Mendoza-Gaytan

Earlham College

CS388

blmendo22@earlham.edu

Abstract

This paper will present my proposal for my capstone project. I will create an algorithm in Unity3D that procedurally generates terrain along with an element of user interaction to give the user a better understanding of the inner workings. This is a remedy to the problem of procedurally generated terrain being challenging to understand. I will introduce the general field of procedural generation, provide a survey, and then lay out a design for my project and other details.

1 Introduction

When you hear procedural generation (PCG), you probably think of video games. Indeed, it has its uses in other forms of media like movies, animation, and visual effects, but video games utilize PCG the most out of any digital medium. PCG is nothing new. Games like *Beneath Apple Manor*(1987) and *Rogue*(1980) are some of the first to use PCG to generate their content. Since then, PCG has grown both in complexity and reach, as well as in research and development. PCG is inherently a complicated and hefty algorithm that takes decent computational power, and research mainly deals with making it more efficient or expanding its use.

2 Review

2.1 What is Procedural Generation?

In video games and video game development, two primary ways exist to create a video game's content: By painstakingly handcrafting these elements or having the computer generate it through an algorithm. The latter is known as procedural generation. A PCG algorithm can be very complicated, but at its core, it is an algorithm that contains several rules that dictate what the result

should look like. Minecraft uses it to generate its infinite number of worlds, as does *No Man's Sky* [4]. *Spelunky* uses PCG to create levels while incorporating game mechanics, increasing replayability.



Figure 1: Minecraft and Spelunky

2.2 Handcrafting or Generating Content?

During the development of video games, there is a problem that all developers eventually encounter: What should be done about the content of a video game? Content can refer to various things in game development, but at the end of the day, it can be solved either by creating it algorithmically(PCG) or handcrafting it [6]. Handcrafting content gives the developers complete control in the design, implementation, and deployment, but it comes at the cost of time and money. Depending on the content, it can take multiple people and groups to accomplish this. Not to mention efficient collaboration and communication. On the other hand, using PCG alleviates some of these concerns, at the cost of control over the content's more intricate designs. But for some developers, this trade-off is worth it. [8]. It should be noted that for some, PCG is not a replacement for handcrafting content but rather a tool to be used alongside human creativity [7].

2.3 Using Procedural Generation

As mentioned, PCG is used in video games and movies, and within these, in different areas, but

they are used more in some than others. Liu Y. in [6] provides an excellent overview of PCG in action. Algorithms can be created to create different elements of a world. Procedurally generating vegetation is possible through an L system, a formal grammar system that lays out a potential shape for vegetation. An algorithm can produce the architecture of buildings and other structures to a believable degree. It's even possible to use PCG to program the behavior of entities within a game. Separation steering [2] is a unique algorithm created by Craig Reynolds to recreate animal behavior. It was tailored for boids, a bird-oid object, and simply put, it mimics a bird flock's behavior. These are examples of how PCG is used in practice and can be applied to different media.

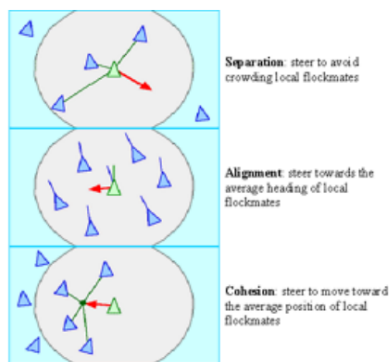


Figure 2: A visual representing separation steering

2.4 Techniques and Conventions

PCG has many applications in the real world, which means it has a lot of techniques and concepts that underpin it. Some are shared between applications, and some are not. In video game development, games that use PCG to generate terrain have similar processes. It all starts with a map of noise [9]. These maps form the basis of what the world will look like. A height map is created using this. Height maps dictate the heights of different points of a world. The whiter a pixel is, the higher it will be when generated, and the opposite is true for darker pixels.

From here, the algorithm can go in many directions. Occupancy-regulated extension(ORE) is an algorithm meant to remedy the issue of conventional algorithms only working within their domain, that is, the specific environment to which they are tailored, considering their game mechanics. ORE is more general and creates geometry without the limitations of the domain in mind. It

should be noted that this algorithm is to be used in a 2D setting. Seeding [10] is a convention used universally within PCG. The seed is usually a string of numbers that plays a big part in the algorithm, as it is used as instructions to generate content. With this, it is possible to generate the same thing. Using seeds is typically recommended as it helps with debugging and testing once the algorithm reaches a particular stage. Another fairly used convention is chunking. PCG is resource-intensive and becomes more intensive as you add graphics and other features. To minimize the load on hardware, you split the content, usually a level or terrain, into equally sized portions called chunks. Only a certain number of chunks are generated, loaded, and unloaded based on conditions specified by the algorithm. These are just some examples of concepts within PCG.

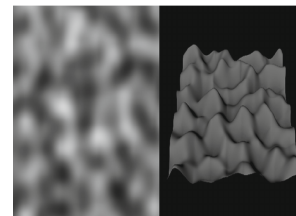


Figure 3: The left is a noise map, and the right is the result of using that noise as a height map

2.5 Problems with Procedural Generation

PCG, as powerful and useful as it is, does not come without its weaknesses. It isn't easy to implement PCG in a meaningful and practical way. The benefits that PCG offers assumes a proper algorithm. Generated content typically looks generic [5], and integrating a PCG algorithm within its domain [7] adds detail and character. This adds complexity and more difficulty in creating a working algorithm, but this is practically a necessity to avoid a generic outcome. Otherwise, it can be worth it to create content by hand. Another limiting factor of PCG is that there are very few general algorithms. What I mean by this is that few algorithms exist that can be used as a foundation and applied to different projects. There are algorithms like ORE [8] which are made to be general and then built upon by others, but it is the odd one out. Any successful implementation of PCG is very specialized and complex to translate to other domains.

2.6 Review Closing

Procedural generation's applications are numerous and can be flexible and tailored to any experience. Algorithms can be created that generate levels and terrains for video games, create structures and architecture, imitate animal behavior, and so much more, and bring a world to life. As is the case with a lot of technologies, PCG is headed in the direction of AI and machine learning integration. Trends for the future involve enhancing the content generated by using AI, making worlds, stories, art, and aesthetics more tailored to players' actions, making PCG more accessible to users without the need for a deep understanding of algorithms, and real-time generation reflecting a user's actions and decisions. The future looks promising. PCG is a powerful tool that can be used alongside human creativity to enhance digital media and create a better experience.

3 Procedural Level and Terrain Generation

I want to hone in on one PCG application: procedurally generated levels and terrain, which is commonly used in video game development. I want to distinguish between levels and terrain, as the former tends to be multiple small bits that make up a game, and the latter tends to be a much larger portion. Regardless, these both have the potential to be some of the most challenging and resource-intensive aspects of a game. I would be willing to go as far as to say that a game's quality is predominantly decided by the variety, depth, and overall quality of its levels. Minecraft is a game that has enormous replay value due to its use of PCG to generate unique terrain every time a player creates a world. Spelunky generates different levels and incorporates the game mechanics into the algorithm, leading to a longer enjoyment of the game. While PCG is not a requirement nor a guarantee that if you implement it, your game will be a hit, it can elevate it if used properly.

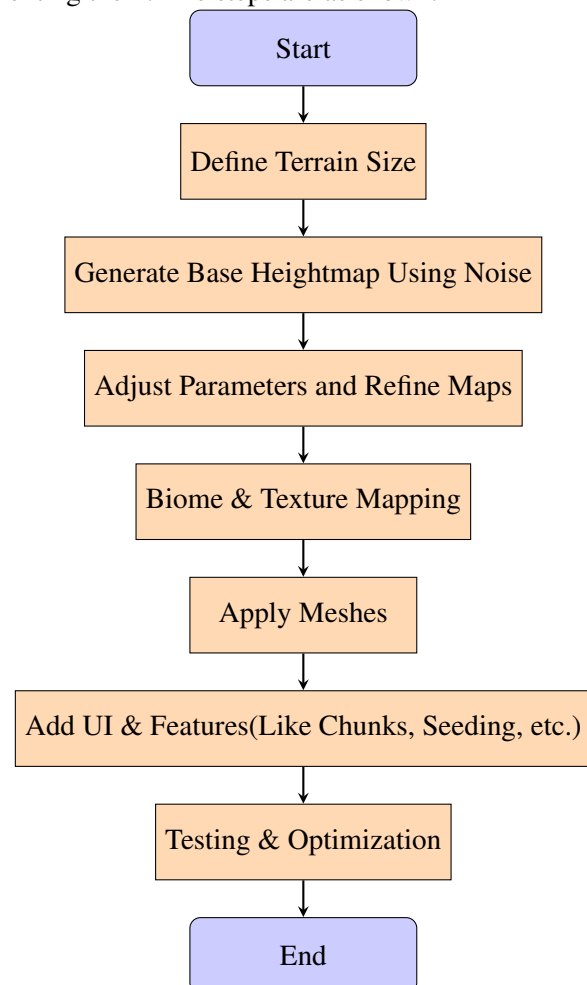
I will focus on terrain generation because that will be the centerpiece of my senior capstone project. I decided to do this as a project because I've played games like Minecraft, which use terrain generation for its worlds, which the developers have dubbed "World Generation". I have always wondered how it works and decided to take this as an opportunity to learn and implement an algorithm. I will not be creating an algorithm on

the same scale as Minecraft. That is an algorithm with years and years of development behind it, and certainly not feasible in the time for this project, but I can create a simple one in Unity3D for the purposes of this project.

4 Design

I want to incorporate an element of user interaction. I mentioned one of the issues that PCG has is the difficulty in understanding what is going on. This extends to procedural terrain generation. To help with this, I want to create an algorithm that users can interact with through a friendly user interface. My inspiration came from Patel in [9]. Here, he adds an aspect of interactivity and explains how noise, height maps, and other variables affect the terrain. I wish to do something similar.

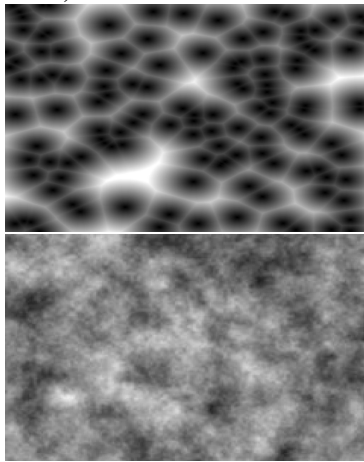
The basic steps to creating a terrain-generating algorithm will be the same as all others. I want this to be easy to understand; as such, I will not be using complicated features or concepts and implementing them. The steps are as shown:



First, I'll have to decide on the size of the terrain. I will start relatively small, but as the project

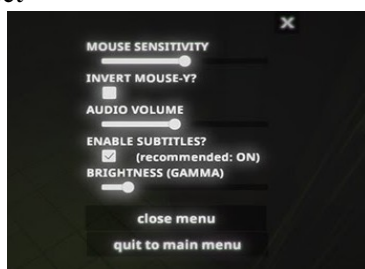
takes shape, I will expand it. Next comes choosing the kind of noise to use for the height map. I will begin with Perlin noise, which is most common when it comes to noise maps. However, many options exist, such as Worley, Simplex, Fractal, etc [1]. Eventually, I want to implement the ability to change the noise used. Then comes adjusting noise parameters, like frequency and octave, and perhaps layering maps on top of each other for a more unique result. The resulting noise map will be used as a height map. Assigning biomes will

Figure 4: Worley/Voronoi Noise (top) and Fractal Noise (bottom)



come next, but I will keep it simple initially. For example, if the terrain is below a certain height, it will be water. It will be a snowy peak if it is above a certain height. To wrap up the foundation, I will add a mesh, which is the land in this case. Finally, I will add a UI that can change the values of the algorithm itself. I want the terrain to change in real time, but I do not know how much this would impact performance. From here on, I will add more features and adjust as needed. Throughout the whole process, I will be testing and optimizing.

Figure 5: Sliders in Unity, which will be used in my project



5 Evaluation Plan

When it comes to evaluating my project, there are a couple of things I will want to keep in mind. The first and most significant part is ensuring the terrain generation algorithm generates a believable and usable terrain. Generating a mess of meshes and land is not the goal here. Next is the consistency and performance of the algorithm. Generating one believable piece of terrain is great, but if the algorithm can't consistently do this, there is a problem. Performance may or may not be a significant consideration. On the one hand, I am not making an extremely complex algorithm, nor do I plan to use high-quality and realistic meshes and assets requiring powerful hardware. I want to make this as accessible as possible.

Figure 6: Odd world generation in Minecraft



On the other hand, this project won't be the simplest thing to run. My design and implementation decisions will impact the performance. Being aware that a design choice won't brick the performance of a device will need to be kept in mind. In the end, though, I think I will have done a good job as long as my laptop can comfortably run this project. If I want to push it, I could use an old machine and test it.

For the user interface side of things, there are three criteria I want to get right—simplicity, ease-of-use, and easy to understand. I must be careful because I wish to find a good balance. I want it to be easy to understand, but without skimping out on the learning, I want it to be easy to use and intuitive, but without taking too much control from the user, and I want it to be simple but not lacking foundational features.

I could ask people unfamiliar with this topic to try out my project to gauge if I have accomplished these goals. It would not hurt to ask those who are familiar for their feedback. I want to note that this project won't be one-size-fits-all regarding a user's prior experience. If I can get someone who

has absolutely no idea about what PCG is to learn, that would be ideal. But I am going in with the assumption that the user at least knows what PCG is.

6 Contributions

When I was looking into PCG and the specifics of terrain generation, I found a good number of references to use. These references were papers from a journal, mainly because that was all that showed up. You had to specify if you wanted a PCG algorithm already implemented. Github has PCG and terrain generation projects, like [3]. You can find projects where people played around with and explored PCG, and some who created a terrain generator that creates terrain based on fields that the user can specify. Still, these assume you are already familiar with the concept. One resource that stood out to me was [9] because of its interactive nature. It started as text and images, but then came the first slider. The slider gave me control over the values of the code snippets. These snippets then controlled a 3D model that resembled the early terrain stages generated by a height map. I could not find anything like this on GitHub, so I decided to take this idea and recreate it with Unity3D as a learning resource to introduce the concepts of terrain generation.

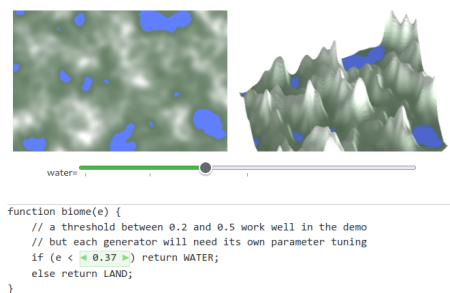


Figure 7: Here is a slider that changes the value that determines where water level starts in [9]

7 Risks

Time will be a risk. I want to think it will not have a significant impact on my creation of this project because of its scalability. I can start small and increase the scope and complexity as I see fit. But unexpected things can happen and throw off my pace, so it would be smart to be conservative with my goals starting out. Get the project's foundation going; have a working terrain generation algo-

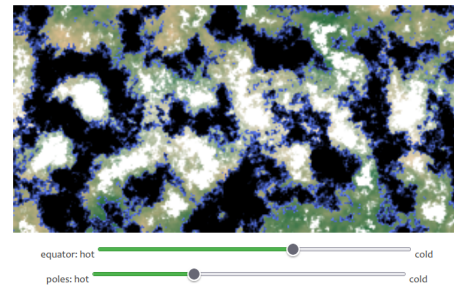


Figure 8: A slider that changes the climate of the poles and equator in [9]

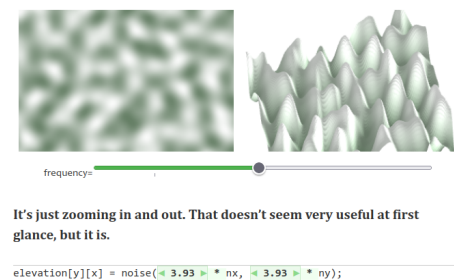


Figure 9: A slider that changes the frequency of hills in [9]

rithm and a user interface to interact with it. From there, I can expand and implement additional features like seeding, so I can directly control what the resulting terrain looks like, chunking if I want to increase the size of the terrain, and other features. Resources, I do not think, will be a significant risk. This is not a project that relies too heavily on external paid resources. Nonetheless, there are some resources to keep in mind.

8 Resources

It is doubtful that I will need an outside source of data. The only resources I have planned to use are the references that I have collected in preparation for this project, the software to be used, which in my case is Unity3D, hardware that is powerful enough to be able to run Unity3D and my project, and the people I will confide in whenever I need help, namely Niel Nicholson as he did a similar project on PCG. Currently, I have no concerns with these resources. Unity, while not open source, is free for personal use, and I have experience with it. As for the hardware, I am pretty confident that my personal computer at home has enough power to use Unity3D and run my project, but I'm sure my laptop can handle it as well. As

for Unity assets, all will be from the Unity store. There are a lot of free assets available on there, which will be sufficient for this project.

9 Timeline

Before starting this project for my capstone, I wish to take some time over the summer to polish up my Unity knowledge. I have experience using Unity and know my way around the environment for basic use at least, but I am far from proficient. It's been a while since I last created a project using Unity, and more knowledge of the environment would be helpful to know before tackling my capstone project.

It is hard to know exactly how to allocate fourteen weeks for this project. As such, this is a tentative timeline.

- Week 1-2: Plan and implement a terrain generation algorithm. Test it afterwards.
- Week 3-4: Have an algorithm to work with, start planning and creating a basic UI
- Week 5-6: Link the UI to the terrain generation algorithm so that it can be modified.
- Week 7-8: Clean up the UI, start planning additional features to implement into the algorithm.
- Week 9-10: Begin implementing and testing additional features into the algorithm
- Week 11-12: Test the overall project, fine-tune, and clean up anything that may need it
- Week 13-14: Implement new features or improve current ones depending on time.

References

- [1] A. LAGAE, S. LEFEBVRE, R. C. T. D. G. D. D. E. J. L. K. P. M. Z. A survey of procedural noise functions. *The Eurographics Association and Blackwell Publishing Ltd.* (2010).
- [2] BOND, L. Procedural generation: An algorithmic analysis of video game design and level creation. *Honors Theses.* 249 (2017).
- [3] CLYENS, A. Procedural terrain generator. <https://github.com/aidan-clyens/TerrainGenerator>.
- [4] ENGELSTEIN, G. Procedural generation. *Gametek* (2023).
- [5] J. TOGELIUS, A.J. CHAMPANDARD, P. L. M. M. A. P. M. P. K. S. Procedural content generation: goals, challenges and actionable steps. *ResearchGate* (2013).
- [6] LIU, Y. An overview of procedurally generating virtual environments. *ResearchGate* (2024).
- [7] MAWHORTER, P., . M. M. Procedural level generation using occupancy-regulated extension. *Proceeding of the 2010 IEEE Conference on Computational Intelligence and Games* (2010).
- [8] NICHOLSON, N. Procedural content generation using occupancy regulated extension. *CS/DS student portfolios* (2018).
- [9] PATEL, A. J. Making maps with noise. *Red Blob Games* (2015).
- [10] SAMPAIO, P., B. A. F. B. . L. M. A fast approach for automatic generation of populated maps with seed and difficulty control. *ResearchGate* (2017).