# Abstract

Contents

[Abstract 1](#_Toc101348943)

[1.0 Introduction 3](#_Toc101348944)

[2.0 Literature Review 3](#_Toc101348945)

[2.1 Mesh Generation 3](#_Toc101348946)

[2.2 Level of Detail 4](#_Toc101348947)

[2.3 Floating Point Errors 4](#_Toc101348948)

[2.4 Procedural Terrain Generation 5](#_Toc101348949)

[2.4.1 Noise 5](#_Toc101348950)

[2.4.2 Data Structures 5](#_Toc101348951)

[2.5 Atmospheric Rendering 6](#_Toc101348952)

[3.0 Design and Implementation 6](#_Toc101348953)

[4.0 Testing 6](#_Toc101348954)

[5.0 Evaluation 6](#_Toc101348955)

[6.0 Conclusion 6](#_Toc101348956)

[7.0 Works Cited 7](#_Toc101348957)

# Introduction

The subject of this dissertation is the generation of procedurally generated, and realistically sized planetary bodies. The finished implementation will be a unity package, that is able to easily create highly detailed, scale and interesting three-dimensional planets, with many customizable parameters.

This work was inspired by the increasing quantity of procedurally generated content within games, specifically the Sci-Fi genre. Games such as *Space Engineers* and *Elite: Dangerous* feature procedural generation as a supplementary system to their core gameplay mechanics (maybe citation needed). On the other hand, *No Man’s Sky* takes the concept of procedural generation and applies it to every aspect of their game, making it part of the players core gameplay loop (citation needed). These are all games that feature procedurally generated planets, and each have their own techniques for generating them. Of these techniques, each have their pros and cons. For example, even though *elite dangerous* prides itself on its realistic scale, their planets could be considered boring and desolate. On the flipside, *no man’s sky* features highly detailed, varied, and interesting planets, but sacrifice size. As a result, the implementation of this project aims to combine all the benefits of these game implementations into one and give the developer a tool to use in the game creation process.

This paper features a literature review, discussing all the major components needed to complete the implementation. These components include mesh generation, level of detail systems, floating point precision errors, procedural terrain generation and atmospheric generation. Following this is the details and design describing how the implementation works, as well as how the research conducted was used in the final product. This product is then tested for its effectiveness at completing the goal set out from this project, by qualitatively measure the output of the code, and quantitatively measure the code’s efficiency and performance. Finally, a brief conclusion is featured, outlining how the project has gone, along with some notes for future refinements or implementations.

# 2.0 Literature Review

## 2.1 Mesh Generation

The most basic component of the system that will be implemented as part of this dissertation is the creation and generation of a sphere mesh. This is crucial as all planets are typically a spherical shape, due to gravitational forces pulling material to the centre of the planet (Sears 2022). For this element of the project, there exists a wide variety of techniques and algorithms to make this initial sphere. Such techniques include: UV spheres, normalized cubes, spherified cubes and icosahedron (Cajaraville 2019). These algorithms can have their effectiveness evaluated based on their: computational efficiency, distribution of vertices, and how close the generated vertices are to the unit sphere. A benefit of the both the cube algorithms, is the ease to implement a Quadtree, which can be used as a level of detail system for changing the mesh’s complexity (Schneider 2006).

One additional method is the Fibonacci sphere (Patel 2022). This algorithm allows for more evenly distributed vertices compared to the previously described methods and, as remarked by Keinert et al, is a “well-known approach to generate a very uniform sampling of the sphere” (2015, 7). Unfortunately, due to the non-linear generation of the vertices, triangulating these points would prove computationally difficult (Lague 2020).

Another downside of this approach would be the difficulty for implementing a level of detail systems, caused directly by the generation method of the vertices. One promising technique is called the marching cubes algorithm. The method uses voxels, which is defined as “a value on a regular grid in three-dimensional space”(Anon. 2019). The algorithm works using a set of 8 voxels to form a cube, then generating a triangle based off these 8 values (Sin and Ng 2018). This technique is typically used on flat terrain, however a paper written by Sin and Ng demonstrates a method to transform the voxels into the unit sphere, allowing for the creation of spherical objects (2018). Unfortunately, the algorithm is known to be significantly slower than the other techniques described, due to the original algorithm having to traverse all the data to generate the mesh (Newman and Yi 2006). Although efforts have been made to speed up and improve this algorithm, a more traditional approach would work best for something of the scale intended for this project.

## 2.2 Level of Detail

The rendering of a highly detailed planets would require the generation and rendering of billions of vertices every frame if a level of detail system is not implemented. This is additionally important, as the max mesh size in unity (using a 32 bit index buffer) is 4 billon vertices (armDeveloper 2022). One technique for implementing a level of detail system is a data structure called a quadtree. Raphael Finkel, the creator of the quadtree, defines them as, “a data structure appropriate for storing information to be retrieved on composite keys” (quadtree citation). This is perfect for use in storing predefined heightmap of varying levels of detail. Unfortunately, to implement this algorithm is computationally complex, and difficult to implement with a procedural generation technique (quadtree citation).

Geometric Clip maps are an additional technique to implement a level of detail system. This is a LOD system that, “caches the terrain in a set of nested regular grids centered about the viewer” and is similar to the algorithm implemented with texture clipmapping (Hoppe 2004). The ideas in this paper are then further discussed and implemented by Mike Savage. This blog also discusses further methods of expanding this technique, such as using Geomorphing to transition between level of details more smoothly, as well as how to add features such as terrain skirts to more traditional plane based terrain approaches (Savage 2017). Due to this algorithm relative simplicity, and the fact it is designed to be used with terrain visualization, this is what will be featured in the final product (Savage 2017).

## 2.3 Floating Point Errors

A visual and programming issue that will begin to occur, when dealing with numbers at the magnitude of planets, is the accuracy of floating-point numbers. In the context of this project, due to floats only having 6 digits of accuracy, once you go further than 1000 kilometres out, you start to lose that accuracy (O’Neil 2022). Since the earth is approximately 6378 kilometres, a method of alleviating this issue is required (Anon. 2022). Symptoms of this inaccuracy can be seen in a talk at Unite 2013 concerning the game Kerbal Space Program. This talk demonstrates a “Jitter” that occurs, which is a vibrating of the game object, that worsens the further out they bring the test spaceship (Unity 2013). In order to amend theses issues, the Kerbal Space Program developers then describe a solution that moves the player camera and game objects into different game spaces, depending on the current scale that is being dealt with (Unity 2013). Additional methods for dealing with these errors include: using doubles in place of floats, manipulating the view matrix and scaling the planets depending on their distance to the camera (Unity 2013; O’Neil 2022). As such, a combination of these previously discussed methods will be implemented.

## 2.4 Procedural Terrain Generation

### 2.4.1 Noise

In Computer graphics, there are many methods for procedural content generation. One of the more popular techniques within this field is the use of noise functions (Reference needed). Noise is defined as, “the random number generator of computer graphics” (Lagae et al. 2010). Of these noise functions (such as Perlin, simplex and anisotropic) each function has their own characteristics, such as coherency and distribution. This noise is then used to generate a floating-point value, to be mapped mapped to a relevant evaluation for a terrain mesh to use, but it does have its draw backs. As described by Fischer et al. noise is, “inherently unintuitive way to adjust noise parameters and consequently, the difficulty to create genuinely realistic looking terrain” (2020). Additionally, without using multiple layers of noise, also called fractal noise, the terrain that would be generated would very unrealistic and somewhat repetitive, as shown by the end product of Michelic’s work (2019). However, combining fractal noise with basic weather simulation, as used in autobiomes, can create both realistic and diverse terrains (Fischer et al. 2020). This technique uses an initial layer of fractal noise, then applies some weather simulation to this data to break it up into different biomes. After this step is complete, the algorithm then applies more fractal noise to the terrain, which is specific to the generated biomes. Due to the high-quality output of this method, this is the way in which the terrain will be generated as part of the implementation.

### 2.4.2 Data Structures

When noise is used in conjunction with flat terrain, the data is then typically stored within a two dimensional heightfield which is, “the most common data structure used for storing and rendering of terrain” (Becher et al. 2017). The data stored within these heightmaps are altitudes, used a mesh builder to construct or manipulate a mesh to represent these different heights. Although this method is prevalent across many games and papers, heightmaps suffer from not being able to represent multiple level terrain, such as cliffs and overhangs, due to the 2D nature of how the data stored (Becher et al. 2017). Another additional method that could be used to store the data, that would allow for these more advanced terrain features, are voxels. As previously discussed, voxels can be used to store volumetric 3D data, and essentially works as a three-dimensional heightmap. However due to both these methods requiring a data set, the method that will be implemented as part of this project will be a more algorithmic approach, generating the appropriate terrain at runtime to allow for cooperation with level of detail techniques.

## 2.5 Atmospheric Rendering

To create a more realistic and immersive planetary environment, atmospheres would be a great addition to the framework that is being built. Many source, such as Elek and Schafhitzel et al. all feature a similar technique that solves the problem of efficient atmospheric rendering (2009; 2007). This method works by creating an effect. The core functionality of this algorithm is derived from pre-calculating the light scattering integral, and storing all of this data in a lookup texture or table, to be then used by a GPU shader as a post processing effect or as part of the fragment shader (Elek 2009). The scattering integral can be computed using two different techniques, Rayleigh and Mei (O’Neil 2005). Rayleigh scattering is the scattering of smaller particles within the atmosphere, whereas Mie is relevant to the much larger airborne particles within the atmosphere.

# 3.0 Design and Implementation

# 4.0 Testing

# 5.0 Evaluation

# 6.0 Conclusion

# 7.0 Works Cited

Anon., 2019. What is a voxel? [online]. Available from: https://www.megavoxels.com/2019/08/what-is-voxel.html [Accessed 4 Mar 2022].

Anon., 2022. *Imagine the Universe!* [online]. Available from: https://imagine.gsfc.nasa.gov/features/cosmic/earth\_info.html [Accessed 24 Apr 2022].

armDeveloper, 2022. *Real-time 3D Art Best Practices: Geometry* [online]. armDeveloper. Available from: https://developer.arm.com/documentation/102448/0100/Triangle-and-polygon-usage [Accessed 12 Apr 2022].

Becher, M., Krone, M., Reina, G. and Ertl, T., 2017. Feature-based volumetric terrain generation. *In*: *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* [online]. Presented at the I3D ’17: Symposium on Interactive 3D Graphics and Games, San Francisco California: ACM, 1–9. Available from: https://dl.acm.org/doi/10.1145/3023368.3023383 [Accessed 20 Apr 2022].

Cajaraville, O. S., 2019. Four Ways to Create a Mesh for a Sphere. *Medium* [online]. Available from: https://medium.com/@oscarsc/four-ways-to-create-a-mesh-for-a-sphere-d7956b825db4 [Accessed 2 Feb 2022].

*Coding Adventure: Procedural Moons and Planets*, 2020. Available from: https://www.youtube.com/watch?v=lctXaT9pxA0 [Accessed 10 Feb 2022].

Elek, O., 2009. Rendering Parametrizable Planetary Atmospheres with Multiple Scattering in Real-Time, 8.

Fischer, R., Dittmann, P., Weller, R. and Zachmann, G., 2020. AutoBiomes: procedural generation of multi-biome landscapes. *The Visual Computer*, 36 (10–12), 2263–2272.

Hoppe, H., 2004. Geometry Clipmaps, 8.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. s., Lewis, J. p., Perlin, K. and Zwicker, M., 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum*, 29 (8), 2579–2600.

Michelic, F., 2019. Real-Time Rendering of Procedurally Generated Planets, 8.

O’Neil, 2022. *A Real-Time Procedural Universe, Part Three: Matters of Scale* [online]. Available from: https://www.gamasutra.com/view/feature/131393/a\_realtime\_procedural\_universe\_.php [Accessed 4 Feb 2022].

O’Neil, S., 2005. *Chapter 16. Accurate Atmospheric Scattering* [online]. NVIDIA Developer. Available from: https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-16-accurate-atmospheric-scattering [Accessed 6 Feb 2022].

Patel, A., 2022. *Delaunay+Voronoi on a sphere* [online]. Available from: https://www.redblobgames.com/x/1842-delaunay-voronoi-sphere/ [Accessed 4 Feb 2022].

Savage, M., 2017. *Geometry clipmaps: simple terrain rendering with level of detail - mikejsavage.co.uk* [online]. mikejsavage.co.uk. Available from: https://mikejsavage.co.uk/blog/geometry-clipmaps.html [Accessed 12 Apr 2022].

Schneider, J., 2006. GPU-Friendly High-Quality Terrain Rendering. *Journal of WSCG*, 8.

Sears, D., 2022. *Why are planets round?* [online]. Scientific American. Available from: https://www.scientificamerican.com/article/why-are-planets-round/ [Accessed 2 Mar 2022].

Sin, Z. P. T. and Ng, P. H. F., 2018. Planetary Marching Cubes: A Marching Cubes Algorithm for Spherical Space. *In*: *Proceedings of the 2018 the 2nd International Conference on Video and Image Processing* [online]. Presented at the ICVIP 2018: 2018 the 2nd International Conference on Video and Image Processing, Hong Kong Hong Kong: ACM, 89–94. Available from: https://dl.acm.org/doi/10.1145/3301506.3301522 [Accessed 21 Feb 2022].

*Unite 2013 - Building a new universe in Kerbal Space Program*, 2013. Available from: https://www.youtube.com/watch?v=mXTxQko-JH0 [Accessed 10 Feb 2022].