# Abstract

Procedurally generated planets allow for an infinite amount of unique player experiences, whilst also allowing developers to focus on gameplay and less on the intricacies of planet design. This project will provide a framework to easily generate realistically sized procedurally generated planets. This project builds on previous studies across: sphere generation (Cajaraville 2019; Keinert 2015; Lague 2020), floating point precision (O’Neil, Unity), terrain generation (Vitacion and Liu 2019; Fischer 2020), atmospheric rendering (Elek 2009; Schafhitzel 2007; Michelic 2019) and level of detail systems (Hoppe 1998). By combining these works, this proposal provides an insight into how to enhance established techniques. Culminating in a C++ library for generating the planets, as well as an executable demonstrating its functionality.

Contents

[Abstract 1](#_Toc95422577)

[Introduction and Rationale 3](#_Toc95422578)

[Literature Review 3](#_Toc95422579)

[Sphere Generation 3](#_Toc95422580)

[Floating Point Precision 4](#_Toc95422581)

[Terrain Generation 5](#_Toc95422582)

[Atmospheric rendering 5](#_Toc95422583)

[Level of Detail 5](#_Toc95422584)

[Methodology 6](#_Toc95422585)

[Aims and Objectives 6](#_Toc95422586)

[Project Plan 8](#_Toc95422587)

# Introduction and Rationale

This final year project will conduct research into the most efficient and high-fidelity way of creating procedurally generated planets that are also a realistic size. These planets will have features such as: multiple biomes, level of detail system, atmospheres, and procedurally textured surfaces. Despite being researched individually, these fields have yet to be combined into a single library or experience. For example, games such as *Elite Dangerous* feature to-scale planets but lack a lot of variation on the planet’s surface, whereas *No Man’s Sky* offers lots of variation but lacks a 1:1 scale. In summary, this project plans to produce planets which are both realistically sized and diverse.

# Literature Review

## Sphere Generation

In the field of procedural planet generation, there are many methods for the creation of the base sphere mesh. The most popular methods include UV spheres, normalized cubes, spherified cubes and icosahedron (Cajaraville 2019). Unfortunately, as with any technique, there are positives (such as computation speed) and negatives (such as non-uniform point distribution) associated with each. An ideal technique would allow for the distribution of the points to be even and as close as possible to the points on an actual sphere.In addition to these techniques, a method using the Fibonacci sequence exists, called the Fibonacci sphere (Patel 2022). As remarked by Keinert et al, this is a “well-known approach to generate a very uniform sampling of the sphere” (2015, 7). As such, this would be optimal for generating procedural terrain if wrapped and triangulated correctly. Unfortunately, this is a very complex computational calculation (Lague 2020).

## Floating Point Precision

A common issue when working with numbers of this magnitude is floating point precision errors. This is because floating point values begin to lose the majority of their precision at 1000km (O’Neil 2022). There are many methods for fixing this error, one of these methods involves manipulating the view matrix. This would essentially mean moving the planet around the camera, instead of moving the camera around the planet (Unity 2013). In addition to this main fix, scaling the planet’s size and using 64 bit double precision values for position, should allow for the removal of any floating point precision errors that may occur (Unity 2013; O’Neil 2022).

## Terrain Generation

The most standard method for generating terrain is to use noise, for example Perlin or simplex, in order to change the elevation of vertices (Fischer et al. 2020). Of this noise there are 5 main types: Simplex, Perlin, cubic, value and diamond square. Of these, each method’s effectiveness can be evaluated by its computational complexity, the randomness of the noise data and the coherency of the noise generated. (Vitacion and Liu 2019). An example of terrain generation based solely on noise is the planets in *Kerbal Space Program* (Unity 2013). But as seen in Vitacion and Liu’s paper, noise alone will not allow for a detailed multi environment terrain (2019). In contrast, the work done by Fischer et al. demonstrates how to use this noise as a base, as well as adding climate simulation to define biomes and add more detailed terrain features (2020). The only downside of this implementation is that it is designed to work on a flat piece of terrain and, as such, would have to be adapted to work on a spherical world.

## Atmospheric rendering

There are several methods used in the industry for atmospheric rendering, a variety of them would be unsuitable for the application being created here. One of these methods is volumetric ray marching, as this method only allows for the rendering of the atmosphere from outside of the planet (Elek 2009). Fortunately, Schafhitzel et al. (2007) and Elek (2009) provide a method which allows for the creation of a mostly scientifically accurate atmosphere with low computational complexity. This is because the first step, computing the scattering integral, is precomputed before runtime operation (Schafhitzel et al. 2007; Elek 2009). Another compelling method is discussed by Florian Michelic (2019). They outline an implementation that would allow for atmosphere rendering through a similar method discussed above, as well as integrating the rendering of clouds into the algorithm (Michelic 2019).

## Level of Detail

A level of detail system (LOD) is crucial to the ability to run the intended executable. This is because the scale of the planets prevents a fully detailed model from being displayed at once due to the sheer number of triangles being rendered. As such, being able to switch or dynamically change the mesh’s complexity would be necessary. There are 3 main techniques that can be used to simplify an arbitrary mesh. These are: coarsen the mesh outside of the view frustrum, screen space geometric tolerances (measuring surface deviation from the original model then refining/coarsening the model based off an error value), and surface orientation (coarsening geometry not in view) (Hoppe 1998). Additionally, this technique described by Hoppe allows for the smooth transitioning between these levels of detail using geomorphs.

# Methodology

Firstly, a further literature review will need to be conducted to gather all valid equations and methods that could be used in the final implementation. Additional initial research will be conducted into the stretch goals so that they can be integrated if there is any extra time. A design, create, test pattern will be used for the implementation of the core features. The implementation for generating the sphere would be the first to undergo this process. The level of detail system will follow, as this is a required component for the rest of the features. Next, the terrain generation algorithms will be implemented, including the base terrain features and the more specific biomes. The final major step in completing the main project is the addition of the atmosphere, potentially stretching to include clouds. If time allows, the stretch features will then be completed in order of: multithreaded optimisations, oceans and, finally, terrain manipulation.

# Aims and Objectives

The main goal of this project is to create an executable that demonstrates complete procedural planet generation system. This executable should be both realistic in features and in scale. The final executable will allow the user to set parameters for the generation of the planet (such as size, colours, seed etc.) and proceed to explore the generated planet with a flying camera.

Following are all the components and problems that must be solved and researched to complete the project. These components are necessary to the projects development and must be completed ahead of the project’s deadline.

* Generating initial sphere mesh
  + Creating the vertices and geometry for the planet.
  + Fixing any floating-point precision errors that would appear due to the scale of the planet.
  + Setup a level of detail (LOD) system to simplify and coarsen the mesh of the sphere, will allow for LOD on the finished terrain.
* Generated Terrain on Sphere
  + Basic Terrain Generation
    - The basic started terrain shape that would be spread across the entire body of the planet.
  + Biome generation
    - Create the biomes on the planets, and then tweak terrain details to refine the individual biomes
* Planet atmosphere system
  + Creating the sky colour, visible atmosphere, and visualisation of the sun.

In addition to these main components, below are additional stretch goals that would ideally feature in the final version. These will only be implemented if time allows.

* Multithreaded generation
  + Using multiple cores to speed up the generation/rendering pipeline of the project.
* Terrain manipulation
  + Giving the user the ability to manipulate the generated terrain, by raising and lowering the vertices, and attaching additional geometry
* Ocean support
  + Establishes an ocean level on the planet and adds large bodies of water between the landmasses/biomes

# Project Plan

Over the page, and attached with this document (Gannt.png, Gannt.mpp), is a Gannt chart outlining the stages in which the project shall be completed. This also includes all additional and associated deadlines. This Gannt chart will be followed step by step to complete the project.

Graphical user interface, application

Description automatically generatedWorks Cited

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].

Lague*,* S., *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., 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. *In*: *Proceedings Visualization ’98 (Cat. No.98CB36276)*. Presented at the Proceedings Visualization ’98 (Cat. No.98CB36276), 35–42.

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].

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

Schafhitzel, T., Falk, M. and Ertl, T., 2007. Real-Time Rendering of Planets with Atmospheres, 8.

Unity,. *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].

Vitacion, R. J. and Liu, L., 2019. Procedural Generation of 3D Planetary-Scale Terrains. *In*: *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. Presented at the 2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT), 70–77.