Newcastle University School of Computing

Creating and optimising a voxel-based rendering and world generation system for a videogame world

Charles Jacob Hart (200376143)

14133 words

Supervisor: Rich Davison

A report submitted in partial fulfillment of the requirements of Newcastle University for the degree of Master of Computing with Honours in Computer Science (Game Engineering)

May 10, 2023

Declaration

I, Charles Jacob Hart, of the School of Computing, Newcastle University, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of Newcastle University and public with interest in teaching, learning and research.

Charles Jacob Hart
May 10, 2023

Abstract

Videogames utilising voxel graphics are incredibly popular, such as Minecraft, one of the best selling games of all time, as are games which produce procedurally generated content. With the audience for these games only growing, it is important that voxel rendering systems are optimised as much as possible. Both a voxel rendering and procedural world generation system were created using the Unity game engine, which is able to render a world consisting of over 400,000 voxels. The world generation system uses smooth noise functions to generate hills and caves in the terrain.

Several optimisations were added to both the procedural world generation and voxel rendering systems, including view frustum culling, object pooling, and internal face culling. It was found that object pooling reduced the load times of terrain by 30.7%, and that internal face culling provided an increase in frame rate of over 1700% to a small voxel world. Due to the view frustum culling that Unity provides by default, the results of the implemented View Frustum Culling (VFC) algorithms were inconclusive, and more research must be done to determine their effectiveness.

Acknowledgements

I would like to thank my supervisor, Rich Davison, for his guidance and support throughout this project.

Contents

List of Figures

List of Tables

List of Abbreviations

Chapter 1

Introduction

1.1 Context

Voxels are both a data structure and a way of representing data graphically in three dimensions; A voxel represents a single point in 3D space, on a regular grid. It can be considered as the 3D equivalent to a pixel in two dimensional computer graphics. The task for this project is to create both a system capable of generating a 3D voxel world for a video game, and a system capable of rendering this generated world, before further optimising both of these systems.

Most traditional videogames take place in a static world, which is designed and created by artists during production of the game. However, a procedural approach can create a game world which is unique for each player, and can be as large as the player wants it to be, as new terrain is generated programatically; The world is generated following several constraints and variables, which allow different worlds to look recognisably similar, with enough variation to create a varied player experience with little repetition. [1]

Additionally, voxel based game worlds allow a further degree of variation, as it is much simpler to create dynamically deformable terrain, where events in the game can change the shape and composition of the game world - such as is seen in the popular videogame Minecraft, which has sold over 238 million copies, [2] where players can destroy or create parts of the world at a single voxel scale.

1.2 Project Overview

In this project, a voxel based rendering and world generation system will be created, before being optimised as much as possible in the time allotted to the project. The voxel based rendering system will utilise polygonal rendering, and not direct volume rendering - where volumes are rendered directly without generating polygonal surfaces, [3] for simplicity and ease of production in the Unity game development environment.

1.3 Aims and Objectives

The broad aim for this project is to create an optimised voxel based rendering and world generation system. The systems should be sufficiently documented so that a competent member in the field could understand how it works, and then utilise it to add gameplay and turn it into a full game. Therefore, it could form the basis of a videogame.

Although the artefact created is described as a "game" throughout this report, it does not have any gameplay features at this point in time, as this was not the intent of the project, and it should instead be considered a tech demo.

Firstly, research will be carried out into voxel graphics systems and procedural content generation systems in videogames, with two or three examples being selected for further research and implementation.

Secondly, a voxel rendering system will be created. This voxel rendering system will be able to receive a list of "Blocks" - Different textured states each voxel can hold, such as air, stone, grass etc - and will turn this into a set of triangular meshes with the appropriate materials and textures in order to render the given voxels.

Next, The world generation system will be implemented. It will use smooth noise functions to generate a list of blocks to send to the rendering system, which follow given constraints and rules, creating a varying game world with features such as hills and caves.

Finally, optimisations will be added to both created systems, so that the rendering system will run at a stable frame rate of at least thirty FPS, as this is a common minimum accepted frame rate for console videogames [4], or ideally a minimum of sixty FPS, as this is a common refresh rate for modern monitors. [5] This frame rate should be achieved on a computer with at least 8GB of Random Access Memory (RAM) [6] and at least 4GB of video RAM in the Graphics Processing Unit (GPU) [7], as these are commonly stated minimum system requirements for videogames in 2023. The procedural world generation system will be optimised so that it can run in real-time; As the camera moves towards the edge of the currently generated game world, more sections of the game world will be generated. In order to maintain the illusion of a natural game world, there should be no significant drops in performance when generating new parts of the game world.

The performance metrics recorded will be taken from multiple devices, with little to no other software running in the background which may impact the performance of the game. This will provide a wider sample size for the results than would be given by a single device. Evaluation will also occur in "built" versions of the game, as a packaged executable, and not inside the Unity editor itself, as this would come with a significant reduction in performance. [8]

1.4 Overview of Report

Chapter 2 of this report will discuss the background information required in order to understand both the technologies used to complete the project, and the information used to inform the work carried out in the project.

Chapter 3 will cover the methodology including the project requirements, design, implementation and testing strategies. Chapter 4 will cover the results and evaluation of the project, before chapter 5 concludes the report with a discussion of the extent to which objectives were met, a summary of what was learned, and areas of future work which could expand upon the project.

Chapter 2

Background Review

This section presents information that may be required to understand the report, and sources which have informed the work done.

2.1 Concepts and Applications

2.1.1 The Unity game engine

Unity is a popular game engine, which can be used to quickly and easily create 3D and 2D games. [9] Below, Unity specific terminology is explained.

- **Components** are functional behaviours or scripts that can be attached to a game object. Unity has many built-in components, such as those allowing objects to interact with they physics system or render 3D geometry, but custom components can also be written, using C#. [10]
- A **game object**, often spelled gameobject, is one of the core concepts in unity. Each gameobject is a structure that by default only has a transform component, which stores information about the position, scale, and rotation of a gameobject. [11] By adding more components, gameobjects can be given more functionality, such as the ability to render a mesh, and have a 3D appearance in the game world. [12]
- A **scene** is a file which contains a collection of game objects being used in a game, in groups which will be loaded all simultaneously. [13] For example, in a simple game, both the main menu and first level of the game could be separate scenes, as they will each need to be loaded and unloaded separately from one another.
- **Instantiation** and **destruction** are processes performed to gameobjects in Unity, in which a gameobject is "spawned" into the game world, [14] or removed from the game world, [15] respectively. Instantiation can be used to create copies of a game object which has been turned into a prefab, a

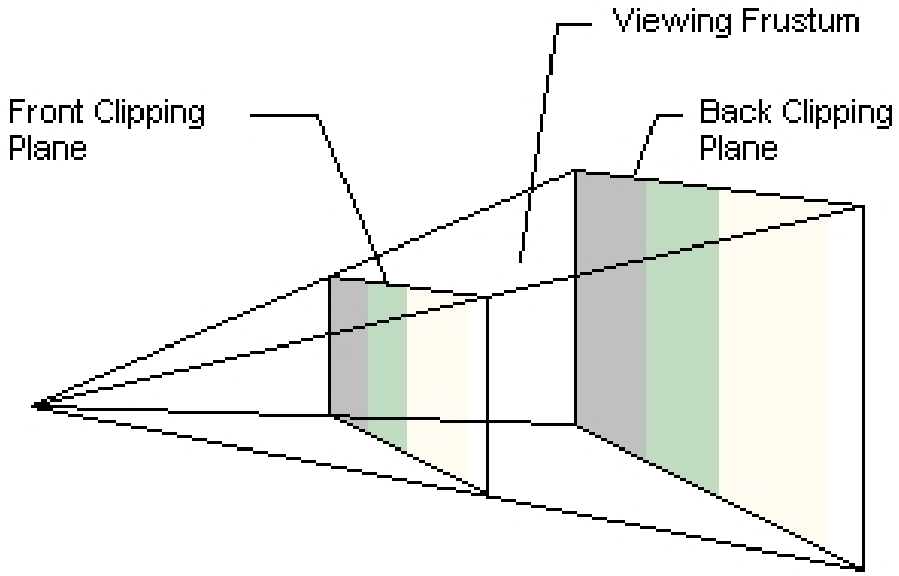

Figure 2.1: A diagram showing the view frustum of a camera in 3D space. [18]

template of a game object, which does not have to currently exist inside a scene. [16]

2.1.2 Computer Graphics Concepts

3D objects in computer graphics are typically made up of a collection of primitives, simple polygons - often triangles - to form a mesh, which can be rendered to create a 2D image of the 3D model. [17] Each triangle consists of three vertices, points in 3D space which are connected together.

2.2 Optimisations in video game software

2.2.1 View Frustum Culling

In computer graphics, the view frustum is a 3D shape which defines the area that is visible by a camera, between the near and far clipping planes. [18] See Figure 2.1. View frustum culling is the process of culling, or disabling the rendering of objects, when they are determined not to be inside the view frustum. Lowering the number of objects being rendered will reduce the burden on the GPU, and increase the frame rate of the game.

In a paper by Su, Guo, Wang, *et al.*, a VFC algorithm is proposed, which uses an adaptive binary tree structure to efficiently determine which objects are contained within the view frustum. [19] The results of the paper show it has a

high efficiency when being used in a scene with a high amount of polygons, as would be seen in a rendered voxel world, which may potentially have millions of triangles in total.

2.2.2 Occlusion Culling

Occlusion Culling is a culling method in which objects that are occluded, or hidden behind another object, are culled. [20] Several different methods are available for determining which objects are being occluded, but in a paper by Bormann, the view frustum is sliced, as an octree is created, which is used to determine which objects are being occluded. [21] The nearer objects are to the camera, the larger they will be when rasterised and projected onto the screen. Therefore, these objects are more likely to occlude other objects, and so more divisions in the octree will be found closer to the camera, so that the efficiency of checking an occlusion query involving an object close to the camera is increased.

2.2.3 Object Pooling

Object pooling is an optimisation that can be used in the Unity game engine, when many game objects are being instantiated or destroyed at once. [22] Object pooling can be beneficial because, when instantiating many game objects at once, there is a high burden on the Central Processing Unit (CPU) [23] as more memory must be allocated for these objects, and memory fragmentation can result in it taking a long time for the CPU to find a contiguous area of available memory. [22]

Therefore, when it is expected that several copies of a game object will be instantiated at once, for example the face of a voxel, it is more efficient to instead keep this memory allocated, by not destroying gameobjects that are no longer in use. Instead, these objects are simply deactivated, stopping all attached scripts, and stored in a "pool", a list of objects from which it may be retrieved and reactivated when a new face is required.

This reduces the number of times game objects are both instantiated and destroyed, lowering the burden on theCPU [23], and therefore increasing the overall performance of the game, such as an increase in frame rate.

Chapter 3

Methodology

3.1 Requirements

The overall aim for this project is to create a voxel based rendering and world generation system using the game engine Unity. In order to do this effectively, and to evaluate the project results, a number of requirements must be defined. Different requirements will be allocated different priorities, and estimated difficulty or effort to complete, so that the time available on the project can be utilised effectively.

Table 3.1: Table showing the priority of features, the desirability of implementation, and estimated difficulty of implementing each feature.

Feature	Category	Priority	Estimated Difficulty (Relative Units)
Rendering a single voxel	Mandatory	1	8
Adding textures to voxels	Important	3.5	5
Rendering a chunk	Mandatory	2	6
Rendering pillars	Mandatory	3	5
Generating world dynamically	Very Important	7	10
Rendering given list of blocks	Mandatory	4	6
Generating world blocks	Mandatory	5	7
Generating hills	Very Important	8	10
Generating caves	Important	10	15
FPS counter UI element	Very Important	6	2
Internal face culling for voxels	Important	9	12
Generating Rivers	Optional	14	18
Voxel lighting system	Optional	15	25
View frustum culling	Important	12	9
Occlusion culling	Desirable	13	25
Object pooling	Important	11	12

Categories, in ascending order of importance, are Optional, Desirable, Important, Very Important, and Mandatory.

3.1.1 Description of Requirements

Rendering a single voxel: The most basic requirement is that the game must be able to render a single voxel, made up of 6 faces to create a cube shape. This requirement has the highest priority, as without this requirement being met, almost all of the other features are unable to be implemented in any way.

Adding textures to voxels: One requirement for a complete voxel rendering system is the ability to render different textures or materials to different voxels. There may both be textures which apply to all sides of a voxel, and materials which apply different textures to each face of a voxel. In addition to making the final game world more interesting, this will help when debugging the project, as it will be much easier to distinguish between individual voxels.

Rendering a chunk: As part of the world generation, a chunk will be a 16x16x16 cube of voxels. Chunks will be utilised in many ways, including use in the view frustum culling system. Additionally, they will be the main component in the voxel pillars used by the procedural world generation system.

Rendering pillars: A pillar will be a stack of several chunks on top of each other, to create a vertical volume of voxels. This will be the main subdivision of the game world, and will be used extensively by the world generation system and rendering system.

Generating world dynamically: This is a very important requirement, which is that in a given area around the player camera, pillars will be generated and rendered; A pillar is a vertical 3d volume made up of voxels. Pillars will be generated and rendered as the camera moves towards the edge of the currently generated game world, and also removed as the camera moves to a given distance away from each pillar. This means that the player can move the camera in any direction across the world and the world will continually generate more and more pillars of terrain.

Rendering a given list of blocks: The world generation system should be able to pass a list or array of "Blocks" - states which each voxel should hold - to each chunk, in order for these blocks to be rendered. This is a simple requirement, but the two systems must be able to work together in order for this project to be successful.

Generating world blocks: This requirement is the basis for the world generation system. This system will complete a number of passes, taking the position of a voxel as input, to determine what block that voxel should turn into through the world generation process. Examples of these passes include determining the

height of hills, placing a layer of grass blocks on the surface, rock underneath, and carving cave shapes through the terrain.

Generating hills: This requirement is a part of the world generation system. Hills are an important part of terrain, and varied height can be generated using a heightmap. This heightmap could take the form of a smooth noise function, which creates pseudorandom noise, more importantly with smooth transitions between points, so the values on the heightmap and therefore hill height would not look unnatural or jagged, and instead form natural looking, rolling hills. More specifically, hills should not have an incline of more than 5 voxels on the Y axis per voxel on the X or Z axis.

FPS counter UI element: During implementation, it is very important to be able to quickly see the frame rate of the scene, as this is one of the most significant metrics by which the performance may be evaluated. Therefore, the project must have a User Interface (UI) element which displays the current FPS of the scene. In order for the value being shown to be human readable, it must not update more than approximately five or six times per second.

Internal face culling for voxels: This requirement is an optimisation for the voxel rendering system. For each voxel, faces which are not exposed to the "air", or are touching other opaque voxels, should not be rendered. This is because they will never be visible, as the player camera will never be inside an opaque voxel. As this is an optimisation, the performance cost of implementing this should not outweigh the performance benefit of the optimisation.

Generating rivers: This requirement is a part of the world generation system. Rivers are a part of most natural environments, so they should be included in the world generation system. They should consist of water blocks, and start as several thin streams at the top of nearby hills before combining together, each adding to the total width of the resulting river. This river should also be at the same level as the terrain in the area, and should not sit on top of the terrain in an unnatural manner, as liquids in the real world do not behave like this. Note that no fluid simulation should be attempted here.

Voxel lighting system: This requirement is a part of the voxel rendering system. Lighting of voxel surfaces can take place in a much simpler manner than existing lighting and shading methods, as each surface is perpendicular to the others, and lighting can be calculated for each individual face, lowering the brightness by a specified amount for each voxel in turn as they get further away from a light source. Smoothing and interpolation can then be applied, creating a lighting system which does not use features such as metallicness or smoothness, as many other shaders may use. This should result in a more performant shader, which avoids the added performance costs of unneeded features. This could however be very complex to implement, so can be seen as optional in the project.

View frustum culling: This requirement is an optimisation for the voxel rendering system. This optimisation will cull, or otherwise not render, any voxels which are not within the view frustum of the player camera. This will prevent any attempts to render geometry which will not be within the bounds of the screen, and should therefore increase performance as the load on the GPU is reduced. [24]

Occlusion culling: This requirement is an optimisation for the voxel rendering system. Occlusion culling typically culls - the early rejection of objects from the rendering pipeline - polygons which are being occluded, or obscured by a second object between it and the camera. This cuts down the number of draw calls being processed by the GPU for objects which are in frame, but not visible due to being behind another opaque object. This optimisation must not cull any polygons which are not being occluded, as this would cause undesirable changes to the graphics of the game, not rendering polygons which should be visible.

Object pooling: This requirement is an optimisation for the voxel rendering system. It will function by storing the mesh objects which a voxel is composed of, when a voxel is removed from the scene, for example when the player camera moves too far away from existing terrain. These stored mesh objects can then be used again when more terrain is generated and rendered, instead of instantiating entirely new objects. This is because instantiation and destruction of gameobjects in unity comes with impacts in performance, so object pooling can avoid both of these processes. [23]

3.2 Design

The design stage of the project covered several areas, including both the voxel rendering and world generation systems, the basis of which is the overall structure of the game world.

3.2.1 Voxel Rendering System

3.2.1.1 Voxel

The voxel is the most important component of the game world - each voxel will be of uniform size, and will be a point on a regular 3D grid. The voxel component will store references to each of its neighbouring voxels, in each of the 6 directions, X+, X-, Y+, Y-, Z+, and Z-, as well as a state, or "block", which contains information about how to render the voxel. It will also contain references to its face objects.

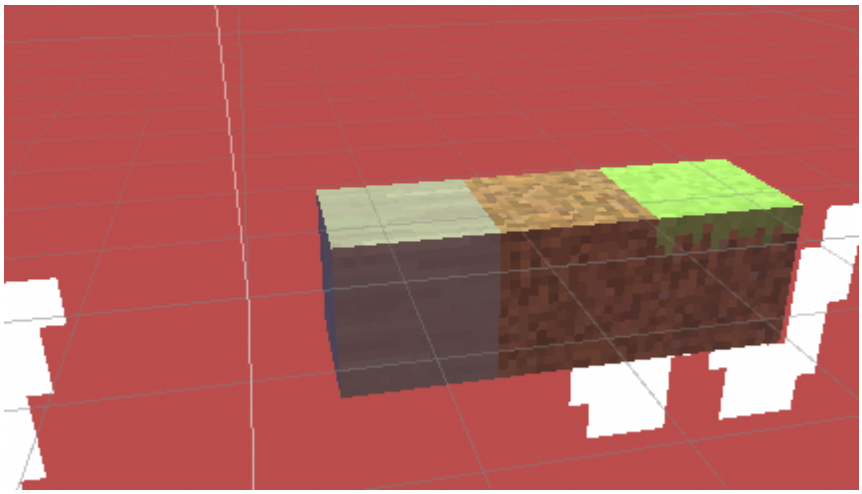

Figure 3.1: Left to right: Air, Rock, Earth and Grassy Earth blocks. Air is not a visible block.

3.2.1.2 Block

Each voxel will have a block value, which provides information for the rendering of the voxel. This includes the block name, opacity - as a boolean, and both the material assigned to the block and the format of the texture, either a single texture for all faces, or a different texture to be applied to each face of the voxel. Examples of blocks include:

- Air, a transparent block which is not rendered, and represents the block state of an empty voxel.
- Rock, an opaque block which has a single grey texture shared across all faces.
- Earth, a block similar to rock, with a brown texture.
- Grassy Earth, an opaque block with separate textures for each face, with the bottom face that of the Earth block, the top face a green grass texture, and the sides a transitional texture with both grass and earth.

3.2.1.3 Face

Each rendered voxel will have between one and six faces, depending on the transparency of adjacent voxels. Each face is an individual quad mesh, made up of two triangles; Due to this, a rendered voxel can remove or add faces independently of one another, as adjacent voxels change state and change the configuration of faces required by the voxel. Additionally, this makes applying textures to blocks with unique textures for each face much simpler, as a different

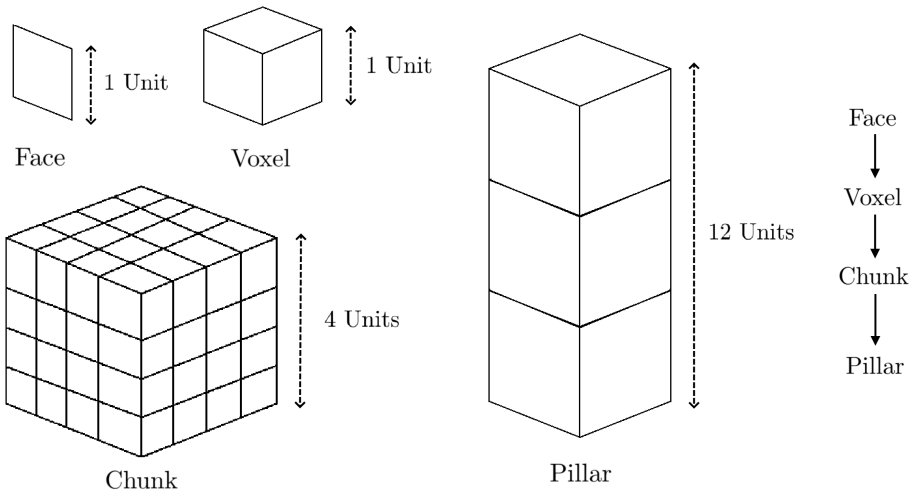

Figure 3.2: Diagram of game world structure and hierarchy - not drawn to scale, and chunks are resized to 4x4x4 voxels for illustrative purposes.

material can be set and edited for each face independently, instead of a single material being applied to an entire voxel's mesh.

3.2.1.4 Chunks and Pillars

In the context of this project, a chunk refers to a 16x16x16 cube of voxels, and a pillar is a series of chunks stacked vertically, with the height depending on the desired world dimensions. The number of chunks in a pillar depends on the world height desired, and is one of the options which can be set in the menu, before generating the world. Pillars will hold references to its chunks, and chunks will hold references to its voxels.

3.2.1.5 Textures and Materials

There will be two methods for texturing voxels, single texture materials, and multiple texture materials. Single texture materials will apply to voxels which have the same texture on each face, and where orientation of these textures is not important. Multiple texture materials however, will contain a texture array or atlas, which has several textures that the material can use. The texture that each face will use can then be chosen so that each face uses the same material, but has a different texture on each face. This approach will also be required on any block which requires a specific orientation of its textures on each face, although none are planned to be implemented in the project at this stage.

3.2.2 World Generation System

3.2.2.1 Hills

Hills will be generated across the surface of the terrain in the game world, with the height of each voxel in the hill being determined by processed smooth noise, such as perlin noise. This should ensure that hills appear to have random locations and sizes, and both the starting height and maximum height of hills can be defined in the main menu.

Unity's Animation Curves system will be used to allow fine tuning of the mapping from smooth noise samples to the heights of hills. Editing this curve in the Unity editor should allow a developer to change the profile of the hills; For example, they could become steeper, or shallower, or even be completely remapped, to create ravines or other features.

3.2.2.2 Cave Systems

Processed 3D smooth noise functions will be used to create cave systems and structures in the game world. The boolean result of this function will determine if the current voxel should be carved to form a cave - meaning it will have the block state of "air". Altering the threshold value will expand or contract the caves, making the size of generated cave systems variable.

3.2.2.3 World Floor

At the base of the game world, or all voxels with a Y coordinate of value 0, a layer of lava rock blocks will be generated. This is mostly to ensure that any caves which intersect the bottom of the world do not result in any situations where a hypothetical player could fall out of the game world.

3.2.3 Optimisations

3.2.3.1 Internal Face Culling

The internal face culling system will operate by performing a check before a voxel's faces are generated, by checking the block state of each neighbouring voxel in each direction. If the neighbouring voxel has a transparent block state, such as "air", or if there is not a voxel in that direction, for example if a voxel is at the edge of the currently generated world, then the face in that direction will be generated and rendered.

Each time a new voxel is generated, usually through generating a new pillar in the world, all voxels touching the newly generated pillar will be updated, and perform this check again, so that any faces which are now not exposed can be removed.

3.2.3.2 Object Pooling

Object pooling will be performed for both pillars and voxel faces. For pillars, when a pillar is unloaded as it becomes too far away from the camera, it will be placed in the pillar pool, which will contain pre populated pillars - pillars which already contain the chunks and voxels which make up a pillar. The faces of these voxels will be placed in six separate face pools, one for each direction - X+, X-, Y+, Y-, Z+ and Z-.

Each time a pillar or voxel face is put into a pool, any references to its position or block state must be cleared, and these values must be set again when the pillar or face is drawn from the pool.

3.2.3.3 View Frustum Culling

View frustum culling can be performed in several ways, and tests will need to be carried out during implementation in order to decide which is the optimal implementation of the optimisation.

The first method is to compare the screen space position of each vertex of a voxel face - the position on the screen which the point will be mapped onto after rendering - and if none of a face's vertices are within the screen bounds, then it may be culled. When a scene has thousands of faces, it may be more efficient to perform this check using the vertices of a chunk instead, and cull whole chunks at a time when they leave the bounds of the screen.

The second solution is to make use of Unity's physics system, which already implements bounding boxes and efficient collision detection. A box collider can be attached to each chunk, set as a trigger, and the camera can have a frustum shaped collider attached to it, with the dimensions of the collider set to match those of the camera's view frustum. Through this method, detecting which chunks are within the view frustum should have a very small performance cost, and will allow view frustum culling to be performed.

3.2.3.4 Occlusion Culling

The occlusion culling system will make use of camera depth textures, generated textures which represent how close the geometry that maps to each pixel is to the camera in world space, [25] and writes this to the red channel of a texture, typically with a value of 255 at the near plane and 0 at the far plane of the view frustum.

This depth texture can then be compared to the depth of the respective pixel in a fragment shader. If the depth in the shader is further from the camera than what the depth texture shows, this means that for the current pixel, the polygon being rendered will not be visible, as it will be occluded by another object in the scene. This means that this fragment can be discarded and does not need to be rendered.

3.2.4 User Interface

3.2.4.1 Main Menu

The game will have a user interface, primarily a main menu seen before generating the world, which will provide many options to change the world generation, and to both enable and disable features that have been added. There are several reasons for adding this, such as being able to compare the impacts of optimisations both individually and in combination with each other. Furthermore, being able to easily change the render distance, or amount of voxels the world should consist of, at runtime, is very useful when measuring performance. Additionally, it is useful to be able to reduce the amount of voxels being rendered when testing the game inside the unity editor, as performance inside the editor is typically much worse than that of a built game excecutable. [8]

3.2.4.2 FPS Counter

The game will have a small UI element at the top left corner of the screen, displaying the current frame rate. This can be used to roughly gauge the performance of the game at any time.

3.3 Implementation

3.3.1 Technologies Used in Implementation

The technologies/software used in the implementation of the project are described below.

Unity *(Version 2021.3.11f1)* is a game engine created by Unity Technologies, which supports both 3D and 2D games across several different platforms including desktop computers, games consoles, and mobile devices.

Plastic SCM *(Version 11.0.16.7739)* is a version control and source code management tool used for software development, especially with Unity, as the software integrates easily into the Unity environment as an add-on through Unity's package manager system.

Microsoft Visual Studio *(Version Community 2019 16.11.10)* is an Integrated Development Environment (IDE) made by Microsoft. It supports several languages, including C#, which will be used for all of the scripting in this project. It also supports the Visual Studio Tools for Unity plug-in, which can be used to debug games in Unity.

3.3.2 Voxel Rendering

3.3.2.1 Voxel Faces

The first element to be implemented was the voxel face. Each *VoxelFace* component has references to its parent Voxel, which stores information such as the

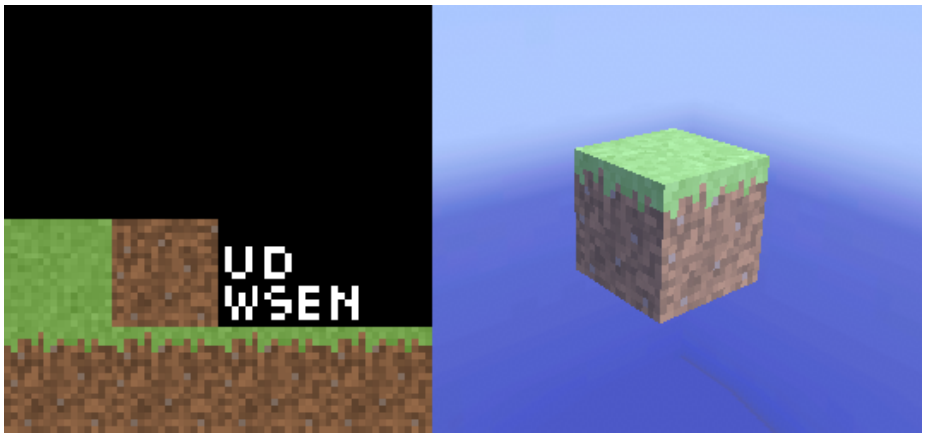

Figure 3.3: Left: The texture sheet for the grassy earth block. The black area and text are unused in game, and serve only as a guide for the directions of each 16x16 texture - Up, Down, North, South, East and West. Right: The appearance of the grassy earth block in-game.

current block assigned to the voxel. [26]

When the *renderFace* method is called on a face, it generates a mesh - a collection of triangles which form a 3D shape - of a square. The vertex positions of each triangle in the mesh are set according to the direction the face has been assigned. Finally, the correct block material is applied to the face mesh.

3.3.2.2 Multi-Texture Materials

If the block material being applied to the voxel face is a multi-texture material, such as the grassy earth block, then the correct texture is chosen from the material's texture array, depending on the direction of the voxel face.

3.3.2.3 Game World Structure

The *worldGeneration* component is the largest component of the world structure, and handles world generation, object pooling for pillars, and holds references to all pillars in the world in a dictionary, with the key as the pillar world position. [27] To generate the underlying structure for the game world, the *WorldGeneration* component instantiates empty pillar game objects, or uses and places existing pillar game objects if there are any in the pillar object pool, in a given area around the player camera, which can be configured in the menu.

The *PillarGeneration* component is part of the pillar prefab, the first subdivision level in the game world, and populates itself with several chunks vertically, according to the specified world dimensions. [28] The *PillarData* component holds references to the contained chunks, in another dictionary with world

space position keys. [29]

The *ChunkGeneration* component then populates itself with many Voxel game objects, which at this point do not have any faces. [30] As with all previous components, it holds references to all contained voxels in a dictionary.

This creates a pillar of empty voxels, which can then be used to place all of the faces required when the game world is generated.

3.3.3 Game World Generation

3.3.3.1 Generating Terrain Around the Player Camera

As part of both the world generation and rendering systems, terrain is rendered and generated in a specified radius around the player camera. The position on the Y axis of a chunk is not used for this calculation, as terrain is generated and rendered one pillar at a time.

Every frame, the pillar that contains the camera is calculated by rounding down the X and Z coordinates to the nearest multiple of 16 - the size of a pillar in these dimensions. Next, the positions of all pillars within the given radius are calculated, and the dictionary of all currently generated pillars is checked.

If there is no pillar at an expected location, one is either instantiated or taken from the pillar pool to be used. If there are any pillars in the dictionary which are not at one of the expected positions, it is then either deleted or removed into an object pool if pillar pooling has been enabled in the menu.

3.3.3.2 World Generation Script

The world generation system generates a list of blocks for each chunk, which is passed to the chunk and then each voxel within that chunk. [27] The list is a set of nested lists, to form a 3D list, where a block value can be accessed using the X, Y and Z coordinates of a voxel - relative to the chunk's origin. The *chunkGeneration* script then iterates through every voxel contained within itself, and assigns each voxel's block state from this list. [30]

In order to generate this list of blocks, the *evaluateBlock()* function is called, with the world space position of each voxel in the chunk as a parameter. The steps taken by this function are described in the below sections.

3.3.3.3 Terrain Base Layer

The first step of the *evaluateBlock()* function is to generate a layer of "Lava Rock" blocks at the bottom of the game world, to ensure that no caves create holes through which a potential player could fall out of the playable game area. To do this, the function will return "Lava Rock Block" if the voxel's Y coordinate is zero.

Figure 3.4: A 2D Perlin Noise texture. The texture can be procedurally generated infinitely in any direction, and can be scaled to match any required resolution.

3.3.3.4 Hill Generation

The second step of the *evaluateBlock()* function is the generation of hills. In order to generate hills, the system uses several variables which can be set in the main menu, which are 'Maximum hill height', 'Minimum hill height', and the hill slope curve, which is defined inside the Unity editor, using Unity's *AnimationCurve* class.

Firstly, 2D Perlin Noise is sampled, using the X and Z components of the voxel's position. This returns a value between 0 and 1, which smoothly interpolates between points and is used as a heightmap - an example of a smooth Perlin Noise texture can be seen in Figure 3.4.

The two variables 'Minimum hill height' and 'Maximum hill height' are both coordinates on the Y axis; using the value between 0 and 1 from the sampled Perlin Noise, the height of a hill at any given coordinate is interpolated between these two values. This results in hills which are randomly shaped and positioned, but still have natural looking slopes between them.

Additionally, the hill slope curve - seen in Figure 3.5 - can be adjusted in the Unity editor, and adjusts the interpolation between the minimum and maximum hill height. This can be visualised as graphically changing the shape of the average hill's slope, and so using Unity's *AnimationCurve* class is an intuitive way to represent this. For example, to linearly interpolate between the two values, a straight line passing through both (0,0) and (1,1) would be used.

After the height of a hill at a given X and Z coordinate is calculated, a voxel which has the Y coordinate matching this height returns the 'Grassy Earth'

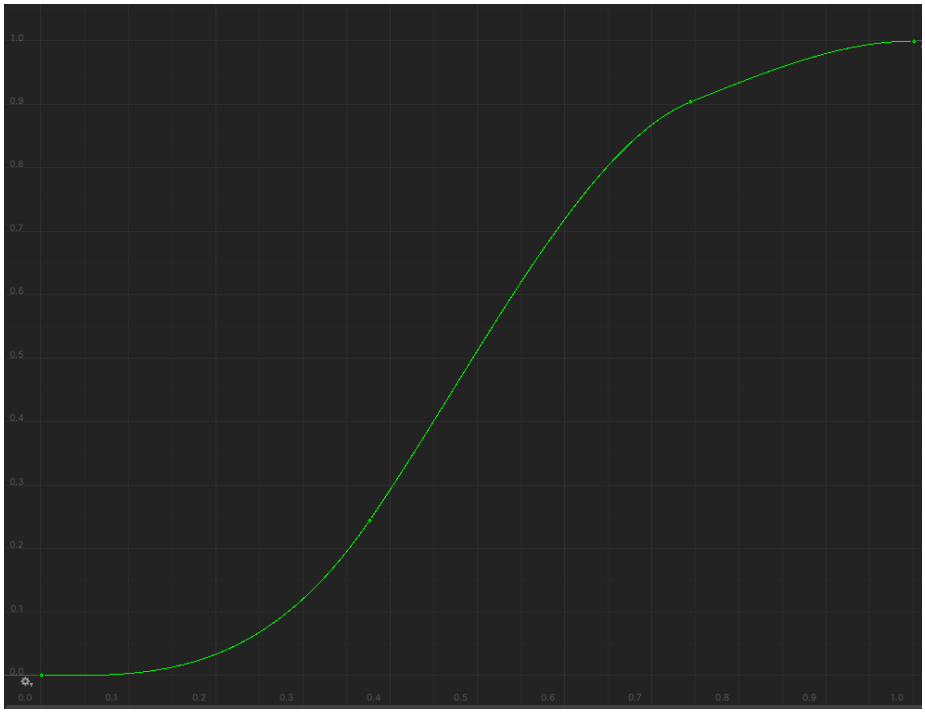

Figure 3.5: The 'hill slope curve', which can be adjusted inside the Unity Editor.

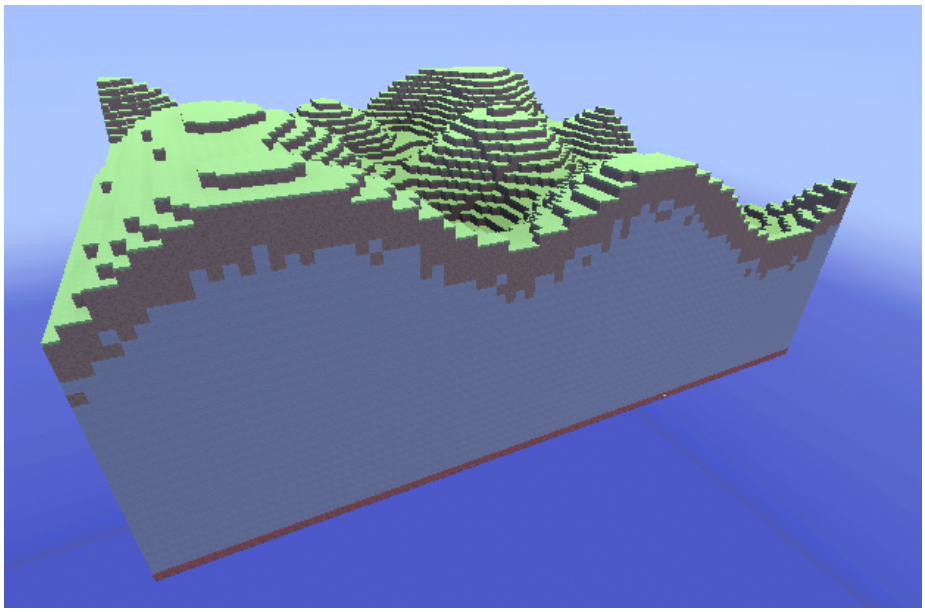

Figure 3.6: A section of the generated game world, including hills, but before caves are generated.

block, as the voxel will be a part of the surface layer of voxels, on top of the hill. Next, any voxel which has a Y coordinate placing it in the three voxels below the surface of the game world will return the 'Earth' block, to create a layer of earth before the rock and caves are generated. Any voxel below this earth layer, but not at the 'Lava Rock' layer at Y coordinate 0, will return the 'Rock' block. Voxels in the area where the rock and earth layers meet have a 50% chance to be the other block of the two, to create a smoother transition between the layers, as can be seen in Figure 3.6.

3.3.3.5 Cave Generation

The final step of the world generation algorithm is to generate cave systems and 'carve' them out - by setting the block state of voxels to 'air' where a cave should be.

The cave generation system also uses Perlin Noise, but as it requires a boolean output - either carve out a cave at a given point, or leave it unchanged - values must be compared against a threshold value.

The function used to generate caves can be seen in Figure 3.7; It was chosen due to the relatively low number of 'islands' that form - areas of one colour which are completely separated from the rest of that colour. The function chosen was $({p_1}^2 + {p_2}^2) <= threshold$ - where p_1 and p_2 are samples of 3D Perlin Noise, each using different seeds for noise generation, and *threshold* is a value which

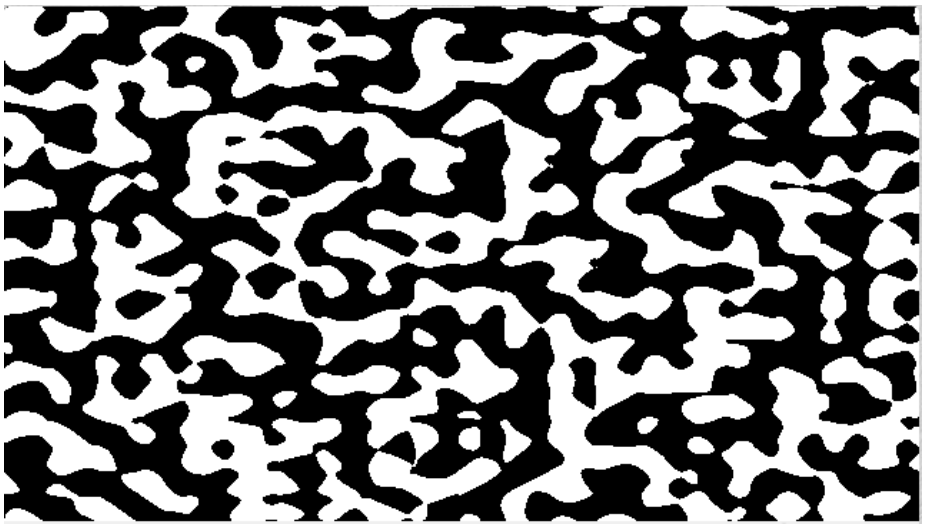

Figure 3.7: A 2D representation of the function used to generate caves. Black represents areas which are unchanged, and white represents areas which are 'carved' out into caves.

determines the width of caves.

When this cave generation algorithm was implemented, it resulted in caves with an appearance like those seen on the left of Figure 3.9. While these caves look close to the desired result, they also produced several large breaches in the surface terrain, where a cave intersects with the surface of the game world. As the Y coordinate of a voxel did not influence the generation of a cave at this point, the cave generation algorithm produced caves which had a consistent width at any Y coordinate. Ideally, the caves should become thinner as they approach the surface of the game world, so that only a small hole is formed and cave generation is less destructive to the appearance of the surface layer.

In order to do this, Unity's *AnimationCurves* were used again. Before a voxel's block state is set to air as part of generating a cave, a check is performed. This check compares the Y coordinate of the current voxel to the Y coordinate of the surface block at the same X and Z coordinates, and if it is within a certain number of blocks - eight by default - or closer to the surface, the cave threshold variable is adjusted in order to make the caves thinner as they approach the surface.

Firstly, a normalised height value within this eight block region is calculated - for example a voxel eight blocks below the surface would have a value of zero, and a voxel at the surface a value of one. Next, the cave taper curve, seen in Figure 3.8, is evaluated using this value along the X axis, and the cave generation threshold is then reduced by the resulting value. As a lower cave generation threshold produces a thinner cave, with less air voxels, this creates caves which taper as they reach the surface of the game world, as seen on the

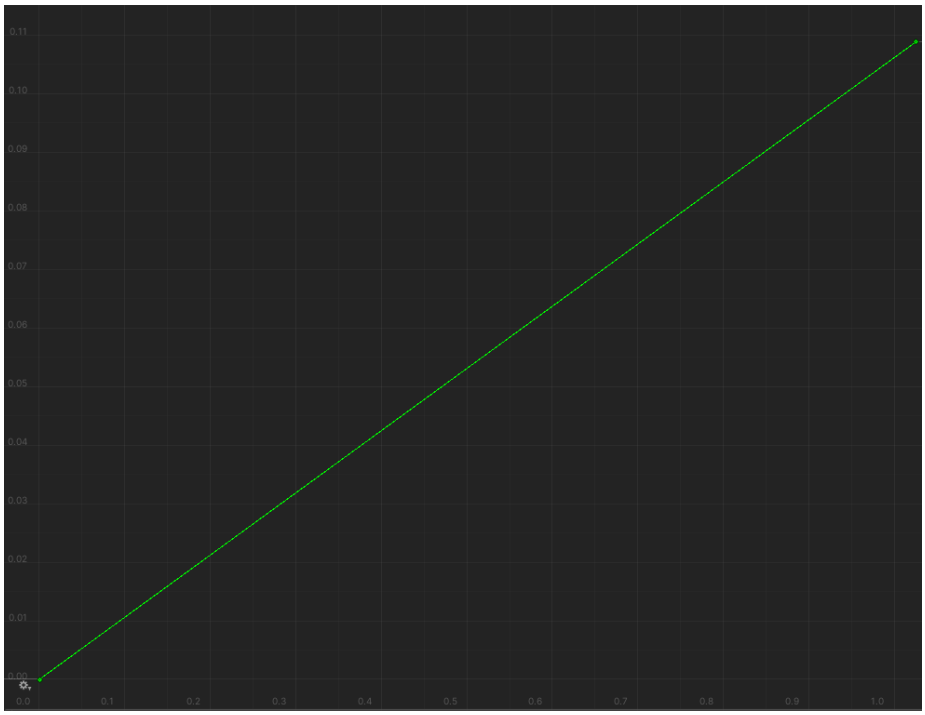

Figure 3.8: The cave taper curve, used to adjust the rate at which caves become thinner as they reach the surface of the game world.

right of Figure 3.9.

3.3.4 Optimisations

3.3.4.1 Internal Face Culling

To perform internal face culling, a check is performed at the start of the *renderVoxel* function. The *VoxelData* component holds references to all neighbouring voxels in each direction, and the block state of each of these voxels is checked. [31] If a neighbouring voxel's block state is not transparent, which includes all blocks other than air in the current implementation, then a face will not be generated in that direction.

If there is no voxel in the selected direction, for example if on the edge of the currently generated game world, then a face will be generated. This means that the only places where faces are generated are between an opaque and transparent voxel, or an opaque voxel on the edge of the game world.

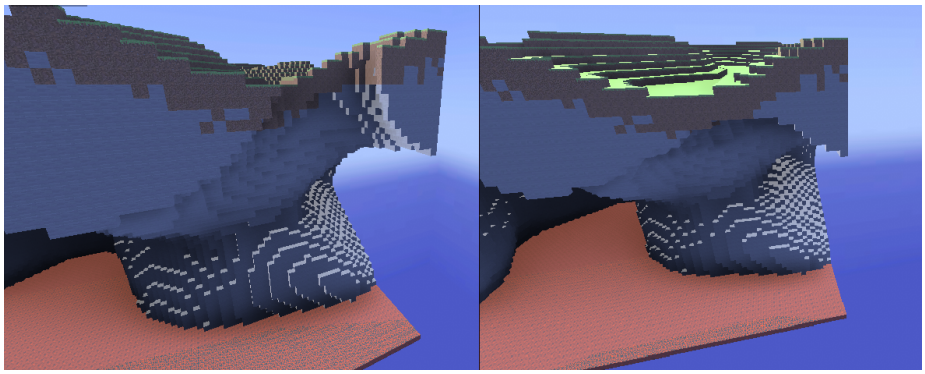

Figure 3.9: Left: A generated cave which breaches the surface of the game world Right: The same cave with cave tapering applied is less destructive to the surface of the game world.

3.3.4.2 Object Pooling

Object pooling was implemented for both pillars and voxel faces. For pillar pooling, pooled pillars are held in a list, and when the world generation system attempts to delete a pillar, it is instead disabled - meaning that none of its components, such as renderers or scripts, will run. [27] It then deletes all voxel faces within itself, or pools them if the face pooling system is enabled, before placing within the pooled pillars list. It can now be taken out of this list and placed in the world at a new location when a new pillar is needed, and it can be re-enabled. This saves the world generation script from instantiating and populating a new pillar with chunks and potentially hundreds of voxel game objects.

For voxel face pooling, six separate pools are created, one for each direction. Faces are pooled according to their direction, and when no faces are available in the pool, a new one is instantiated. It is important that when a voxel face is put into a pool, any references to the face, for example the references each Voxel holds to its faces, must be deleted. Otherwise, Voxels may attempt to render or remove faces which no longer belong to them, resulting in holes appearing in the game world.

3.3.4.3 View Frustum Culling

Several versions of view frustum culling were implemented in this project, though only the final approach met the requirements of adequate performance benefits and no reduction in graphical quality.

The first implementation of view frustum culling utilised Unity's *Camera.WorldToScreenPoint* function, which converts a world space position into the location it will be mapped to on the screen, when the current frame has been rendered. [32] Using this function, the x and y screen coordinates of a point can be compared to

the width and height of the user's display, and if the screen space coordinates are outside of this range, then the point will be outside of the camera's view frustum.

In this first implementation, each vertex of a voxel would be checked every frame, in order to find which vertices are within the view frustum. For any given voxel, if no vertices were within the view frustum, then the mesh renderer - the component responsible for rendering the geometry of a game object in Unity - would be disabled, until one or more of the vertices returned to within the view frustum.

This was not a performant implementation, as will be covered in section 4. Therefore, the second implementation of view frustum culling was largely the same, but was performed on a chunk scale, instead of for the individual voxel. [33] This was significantly more performant, but still had issues. The most significant issue was the incorrect culling of chunks when the camera was in close proximity; In many cases, the camera was not able to see any of the chunk vertices, but the voxel faces within the chunk were still within the camera's view frustum.

Another approach was needed, which would both have sufficient performance, and would not cull any chunks which were even partially visible.

The approach which worked best utilised an element of Unity's physics system, colliders and triggers. Each chunk was given a box collider, which was approximately 110% the size of a chunk. Then, the player camera was given a frustum shaped collider set to be a trigger, the dimensions of which are set to match the camera view frustum. Whenever a chunk's collider enters the view frustum trigger, the chunk's frustum culling component will enable all mesh renderers within itself, and disable all mesh renderers when the chunk exits the view frustum trigger. [34] This produces a view frustum culling system which has little overhead to run, and can efficiently cull chunks.

3.3.4.4 Occlusion Culling

Although an occlusion culling system was planned, it became clear during its implementation, that the system was outside of the scope of this project. It was partially implemented, and the progress made towards a functional occlusion culling system will be covered, though it does not currently function in any capacity.

Firstly, any game objects in the scene with a mesh renderer are given a second mesh renderer, using the same mesh but a separate material, which is only visible to a secondary camera, by editing the culling mask on both the secondary camera - referred to as the occlusion camera, and the player camera.

This material's fragment shader returns a colour, where the red channel represents depth - where a value of 255, or solid red, is the far plane, and 0, or solid black, is the near plane of the camera. [35] This is calculated using the *UNITY_SAMPLE_DEPTH* function, and reading the depth texture of the occlusion camera.

This provides the start of an occlusion culling system, as the texture seen

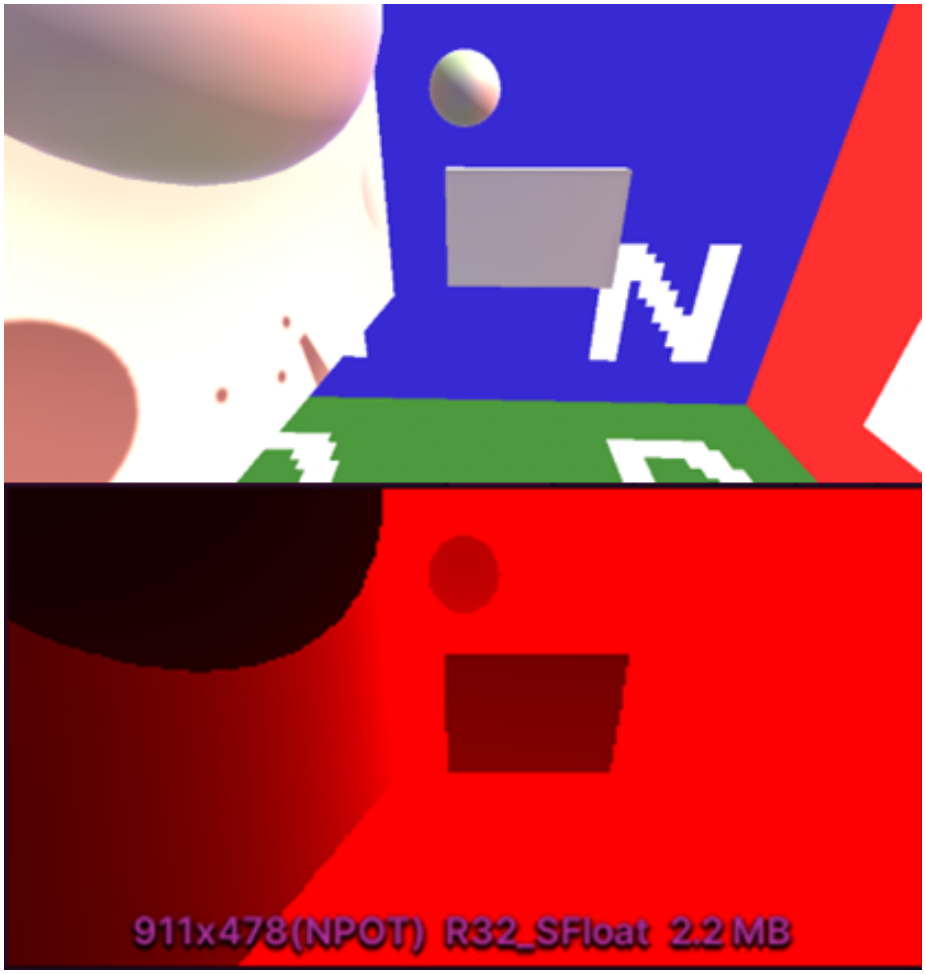

Figure 3.10: Top: A scene with several rendered game objects. Bottom: The depth texture created by the occlusion culling system. *The pink text is texture metadata, and not a part of the texture.*

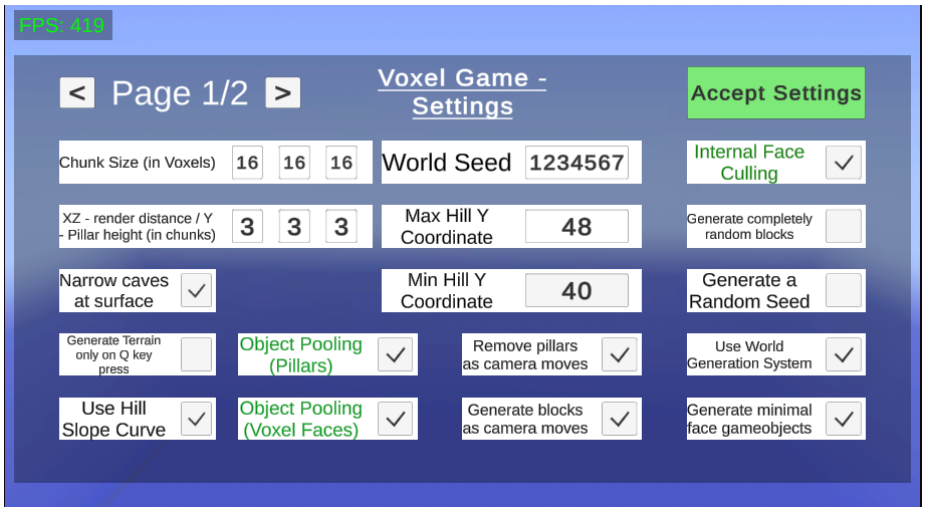

Figure 3.11: Page 1 of the game's main menu, which allows several optimisations to be enabled or disabled, and several world generation properties to be changed.

at the bottom of Figure 3.10 could then be used to determine which meshes are occluded within a scene, for them to then be culled. This is not currently implemented, and will be covered in the future work section.

3.4 Testing

To allow optimisations to be evaluated independently from one another, the game's main menu, seen in Figures 3.11 and 3.12, allows each optimisation and feature to be enabled or disabled, world generation properties to be edited, and allows for selection between the three different implementations of view frustum culling. Therefore, optimisations can be evaluated against both the fully unoptimised game, and any desired combination of optimisations and settings, although it would not be feasible to cover all possible combinations.

3.4.1 Main Menu Options

The settings menu has several features and optimisations that can be enabled and disabled, and several options to change world generation parameters.

3.4.1.1 3D Vectors

Chunk Size (in Voxels) - Defines the size of a chunk in each axis, 16x16x16 is the default.
Render Distance/Pillar Height - X and Z define the number of chunks in

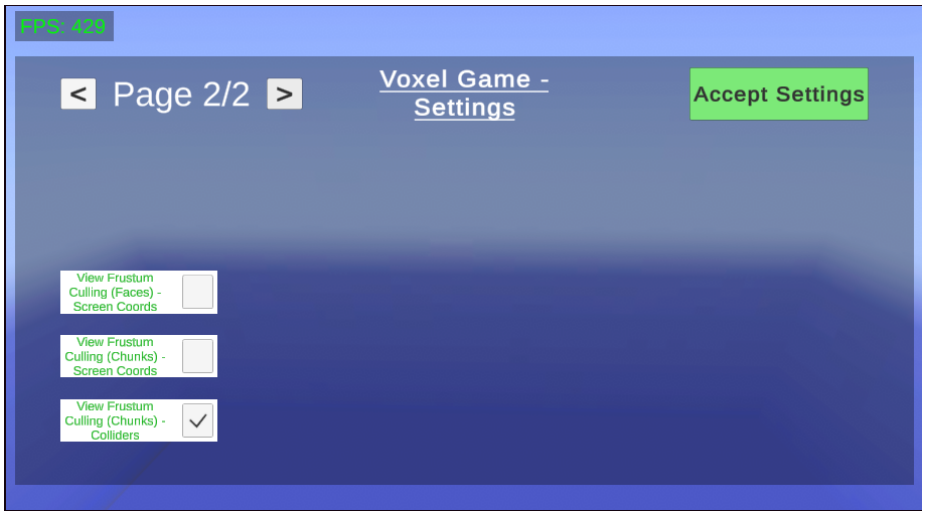

Figure 3.12: Page 2 of the game's main menu, which has options to enable or disable each view frustum culling approach. If several are enabled, only one will be used.

each direction which are generated and rendered in each axis, whilst Y defines how many chunks are in a pillar.

3.4.1.2 Integers

World Seed - A value used in world generation. Changing the seed results in a completely different world being generated.
Maximum Hill Y Coordinate - The highest point in the game world at which a hill will generate.
Minimum Hill Y Coordinate - The lowest point where a hill will generate; any voxels lower than this will be either a cave or rock.

3.4.1.3 Booleans

Internal Face Culling - Enables/disables Internal Face Culling optimisation.
Generate Random Blocks - Generates a random block, or air, at every position in the game world, used for minor tests of the rendering system.
Generate Random Seed - Produces a unique world generation seed.
Use World Generation System - Enables/disables the world generation system.
Generate minimal face game objects - Toggles a minor optimisation made

late in development, which limits the instantiation of mesh-less face game objects to voxels which have rendered faces.
Remove Pillars as Camera Moves - Pillars are unloaded/removed as they leave the rendered area around the camera.
Generate Blocks as Camera Moves - Pillars are instantiated and blocks are generated and rendered as they enter the rendered area around the player camera.
Object Pooling (Pillars) - Toggles the object pooling optimisation for pillar objects.
Object Pooling (Voxel Faces) - Toggles the object pooling optimisation for voxel faces.
Object Pooling(Pillars) - Toggles the object pooling optimisation for pillar objects.
Use Hill Slope Curve - Enables/disables the hill slope curve, which can be used to edit the shape of hills. See section 3.3.3.4.
Generate Terrain on Q Press - Disables automatic terrain generation, and generates terrain only when the user presses Q. This is used to view generated terrain from angles not possible in normal circumstances.
Narrow Caves at Surface - Caves taper as the reach the surface of the game world, to reduce the impact they have on the terrain's appearance. See section 3.3.3.5.
View Frustum Culling (Faces) - Screen Coords - Toggles the first implementation of view frustum culling.
View Frustum Culling (Chunk) - Screen Coords - Toggles the second implementation of view frustum culling, which works at a chunk scale.
View Frustum Culling (Chunks) - Colliders - Toggles the third and final implementation to view frustum culling, which uses Unity's physics system.

3.4.2 Data Recording Methods

Test data was recorded using a testing script which collects several sources of data and saves them to a JavaScript Object Notation (JSON) file, an example of which can be found in Appendix A.1. [36]

The notable information recorded is:
Environment Type - Whether the test was running in a built version of the game or the Unity editor.
CPU, Name - Name of the device's CPU.
CPU, Cores - Number of cores present on the device's CPU.
CPU, Frequency - Frequency of the device's CPU, in megahertz.
RAM Total - The size of the device's RAM, in megabytes.
Graphics, Device Name - The name of the device's GPU.
Graphics, GPU VRAM - The size of the GPU's Video RAM, in megabytes.
Test Duration - The duration of the test, from when terrain was first generated, in milliseconds.

FPS, Average - The mean average FPS, sampled twice per second.
FPS, Data - The samples of FPS taken during runtime.
Pillar Generation and Render Times, Average - The mean average duration taken to generate, populate and render a pillar of terrain, in milliseconds.
Pillar Generation and Render Times, Data - The individual data points recorded for this section of the test, in milliseconds.
Chunk Block List Generation Times, Average - The mean average time taken to generate the list of blocks a chunk should have, using the world generation system, in milliseconds.
Chunk Block List Generation Times, Data - The individual data points recorded for this section of the test, in milliseconds.
Voxel Rendering Times, Average - The mean average time taken to instantiate faces for a voxel, or retrieve them from an object pool, and set the materials of the meshes, in milliseconds. Individual data points are not recorded here, as there can often be hundreds of thousands of voxels rendered in a single test, and the file should remain human readable.
Chunk Rendering Times, Average - The mean average time taken to generate everything needed to render an entire chunk, in milliseconds.
Chunk Rendering Times, Data - The individual data points recorded for this section of the test, in milliseconds.

Also recorded is the state of all settings found in the main menu, described in section 3.4.1. Notably, this includes a list of enabled optimisations.

3.4.3 Testing Strategies

3.4.3.1 During Implementation Phase

During the implementation phase of the project, testing was carried out using the settings available in the main menu, and testing various combinations of world dimensions, world generation seeds, and features, to ensure that they had the expected result on the game.

For testing the optimisations at this stage, before the test data recording script in section 3.4.2 were implemented, values were taken from the FPS UI element. Times taken to perform actions such as loading and rendering a new pillar, instantiating a voxel face, and rendering a chunk were printed to the Unity console.

Using a third party package from the Unity Asset Store, the Unity console was made to be visible in a built version of the game, as can typically only be seen inside the Unity editor. [37] This is seen in the game by pressing the "Insert" key, and hidden with the "Delete" key. Additionally, this allows the user to see any errors or exceptions which are thrown in any script that is running.

3.4.3.2 Post Implementation Phase

After the implementation phase, when collecting the final set of results, the test data recording script in section 3.4.2 was used to record data regarding performance, optimisations, and world generation features. Tests were carried out on two different devices, one desktop computer and one laptop, the specifications of which can be found in the test data files. Both devices had a display resolution of 1920x1080 pixels.

Tests were carried out in a built version of the game, as this is the state in which a potential user would experience the game, and running a game inside the Unity editor comes with a reduction in performance. [8]

In each test, all settings and options were kept the same, aside from the feature being tested. The world generation seed was kept the same across all tests, as "1234567", so that changes in the random generation of terrain did not impact the results. In each test, the world is generated, and the user moves the camera around the game world, generating more terrain, for approximately one minute. The devices that were running the tests had no other programs running that would use a significant amount of system resources, and the laptop was connected to AC power to ensure that the power saving mode was not entered, as that would reduce expected performance.

In most cases, metrics such as FPS will fluctuate frequently, therefore the mean average over a period of a minute or more is a better indication of performance than a single frame rate sample.

Chapter 4

Results and Evaluation

This section presents the results of the project - the artefact created, the numerical data recorded, and the findings that are drawn from it.

4.1 Artefact

The artefact created in this project is a tech demo created using the Unity game engine, of a voxel rendering and world generation system. It has a main menu, seen in Figure 3.11, through which the user can define the properties of the generated world, such as the dimensions of the rendered area around the camera, and can enable or disable each optimisation added to the project.

After the user confirms the parameters they would like the program to use from the list found in subsection 3.4.1, the game world is generated. The game world is made up of vertical pillars, which are loaded in a specified area around the player camera. Each pillar contains several chunks stacked on top of each other, which in turn contains by default 4096 voxels - a 16x16x16 cube of voxels - as can be seen in Figure 3.2 and Figure 4.1.

The game world is generated using an algorithm that determines the block state of each voxel - one of several appearances that a voxel may have, including completely invisible, or "air". Figure 3.1 shows the appearance of several of these block states.

The world generation algorithm generates hills using perlin noise as a heightmap, so that hills have a pseudorandom distribution, and have inclines of between zero and four blocks vertically for every block horizontally - as can be seen in Figure 4.2. [27] Also in Figure 4.2 are the caves created by the world generation system, which use 3D perlin noise to carve cave shaped holes in the terrain. Finally, the lowermost layer of voxels consists entirely of "lava rock" blocks, whose purpose is to provide a base to the world, so that the caves generated do not put holes in the bottom of the game world.

Several optimisations were added to both the voxel rendering and world generation system. The first, internal face culling, stops any voxel faces from

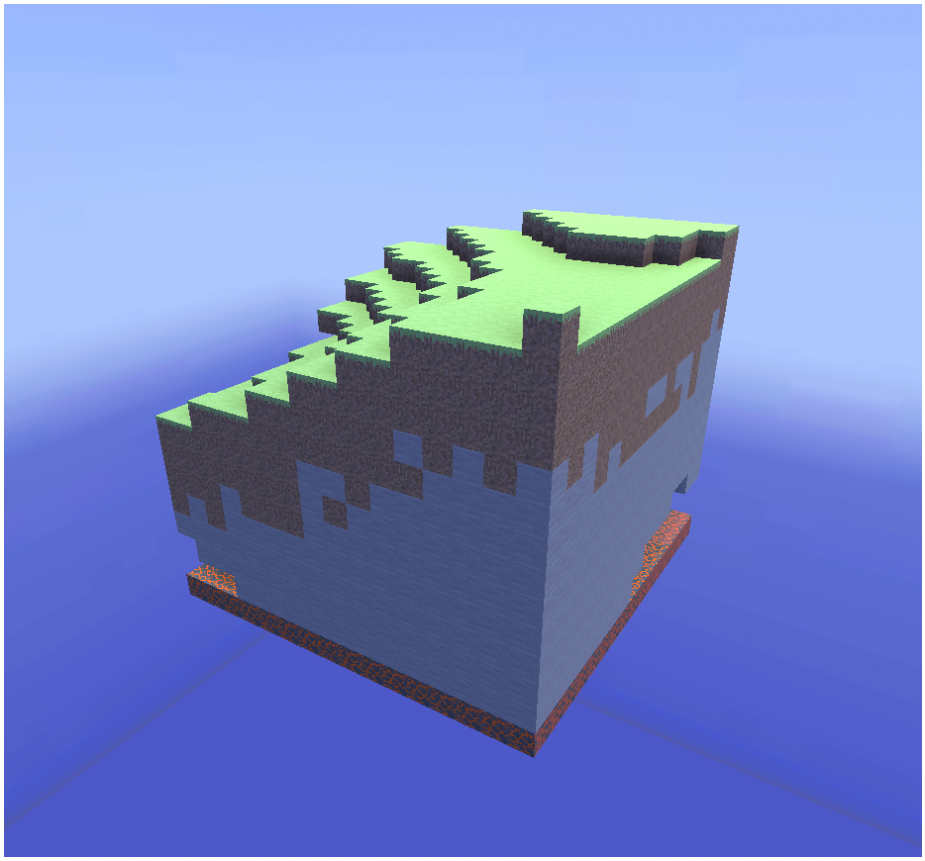

Figure 4.1: A single chunk of 16x16x16 voxels rendered in the game, with each voxel having a block state specified by the world generation system.

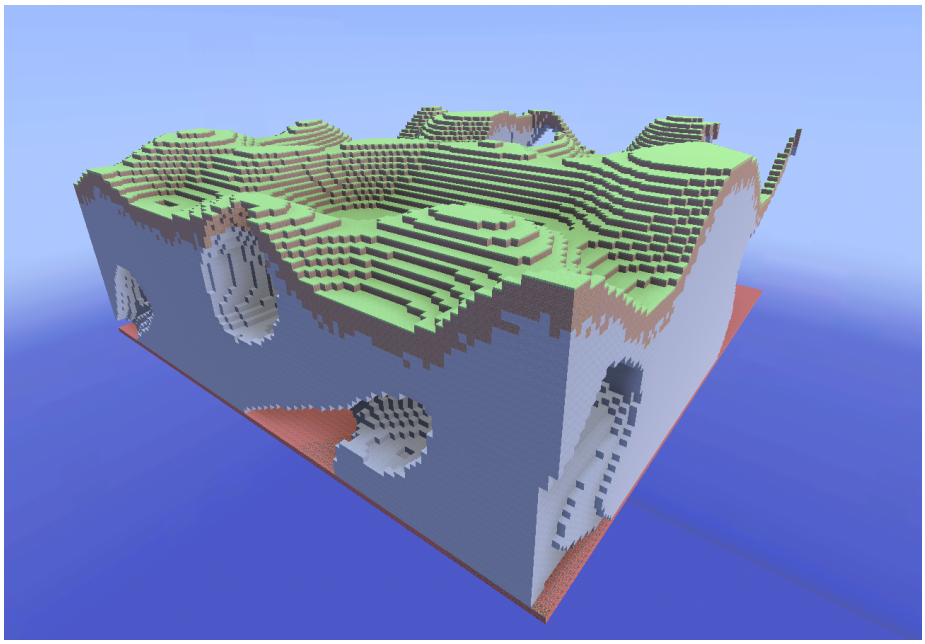

Figure 4.2: A screenshot of a 112x48x112 voxel world, using the world generation system to generate both hills and caves.

being created if the face is not between an opaque and transparent block, or facing the edge of the generated world. This stops faces from being generated inside the terrain, where they are not exposed and not visible - as can be seen in Figure 4.3.

It also has two other optimisations, object pooling and view frustum culling. Object pooling speeds up the rendering of new terrain, by keeping several objects such as pillars and voxel faces loaded when they are no longer being used, but making them invisible. Then, it uses these objects to render new terrain, instead of creating new objects from scratch. This cuts down on the amount of times new objects must be instantiated, as this takes a relatively long time. VFC disables the rendering of chunks that are not within the view frustum, as can be seen in Figure 4.4, and therefore increases frame rate by lowering the number of polygons in the scene.

4.2 Recorded Data and Findings

Data was recorded, using the data recording system explained in subsection 3.4.2, from which several findings are drawn. Data is separated into sections for each optimisation, and only notable data points are shown. Additionally, data is presented from tests on two devices, a desktop PC and a laptop, and more information about how these tests were carried out can be found in subsubsec-

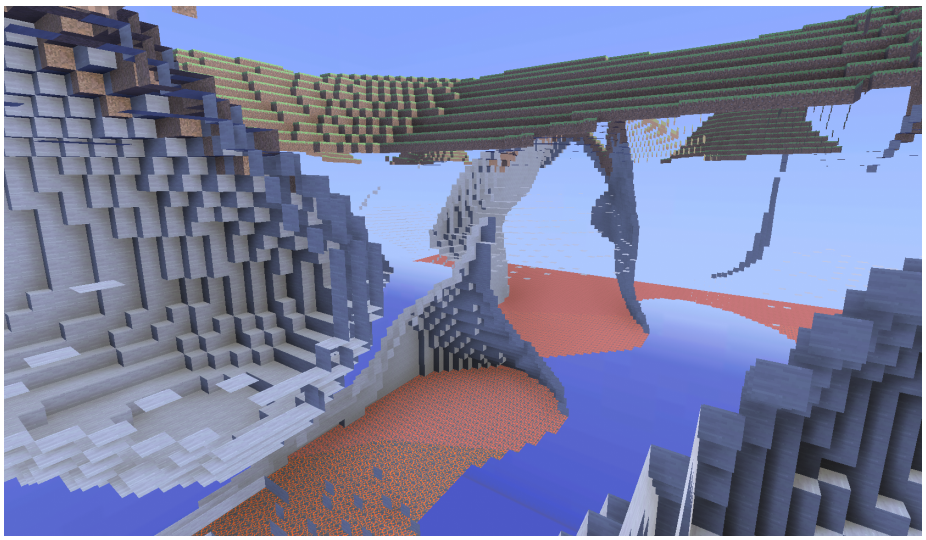

Figure 4.3: The same terrain as seen in Figure 4.2, with the camera placed inside the generated terrain.

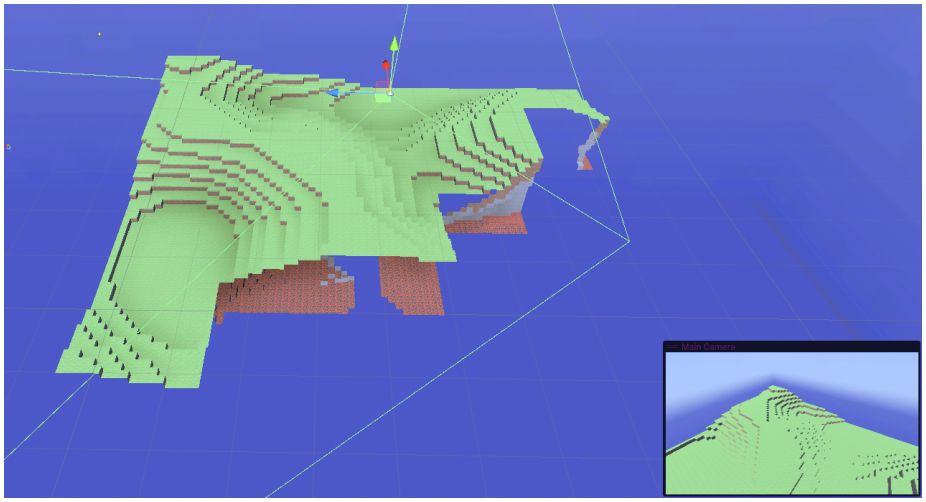

Figure 4.4: Only the chunks within the camera's view frustum, shown by the green lines, are rendered. The window in the lower right shows the view from the camera's perspective.

tion 3.4.3.2.

For each optimisation, tests were carried out on both devices on:

- A partially optimised game, where all optimisations aside from the optimisation being tested are enabled.
- A fully optimised game, where all optimisations are enabled.

A completely unoptimised version of the game will not run when rendering a 112x48x112 voxel world, as this causes the Unity player to crash. As a result, only a partially optimised and fully optimised version of the game will be tested, although this still allows each optimisation's results to be compared against one another.

As VFC has several variants, of which only one may be enabled at a time, only the collider based variant will be used, unless one of the other approaches to VFC is the focus of the test. For each set of tests, the data recorded can be found in the linked appendix, which includes the system specifications of each device, and the settings, world properties, and optimisations chosen in the main menu of the game. See subsection 3.4.1.

4.2.1 Internal Face Culling

During the tests for IFC, the rendered world size of 7x3x7 chunks would cause the Unity player to crash without the optimisation enabled. Therefore, all the data used in this test was gathered whilst running the game with a 3x3x3 chunk render area.

As can be seen in Figure 4.5, without IFC the average FPS of the scene was 5.35, and with IFC, this was increased significantly to 100 FPS. This is a 1,769% increase in FPS for a rendered world of size 3x3x3 chunks. As each voxel has six faces without IFC enabled, this means that, for any given game world, as the rendered area around the camera increases, the surface area to volume ratio of the world decreases. Therefore, a higher proportion of all voxel faces in the scene are internal faces, not exposed to the player camera, and so this optimisation is very important for rendering a large amount of voxels at once. This claim is supported by the fact that without IFC enabled, the 7x3x7 chunk world - of 112x48x112 voxels - that was used to test all other optimisations could not be loaded; The Unity player would crash before the world could be rendered, and therefore a smaller world had to be used to test this optimisation.

4.2.2 Object Pooling

As can be seen in Figure 4.6, without object pooling, the average time taken to generate and render a pillar was 235.1ms, and with object pooling, this time decreased to 162.8. This is a reduction of 30.7%, and shows that object pooling has had a significant positive effect on the loading times of the game. This is particularly important, as this loading of objects is performed on the same thread as the rest of the game, so the game will freeze, with no new frames rendered, until the generation of a pillar has completed. [38]

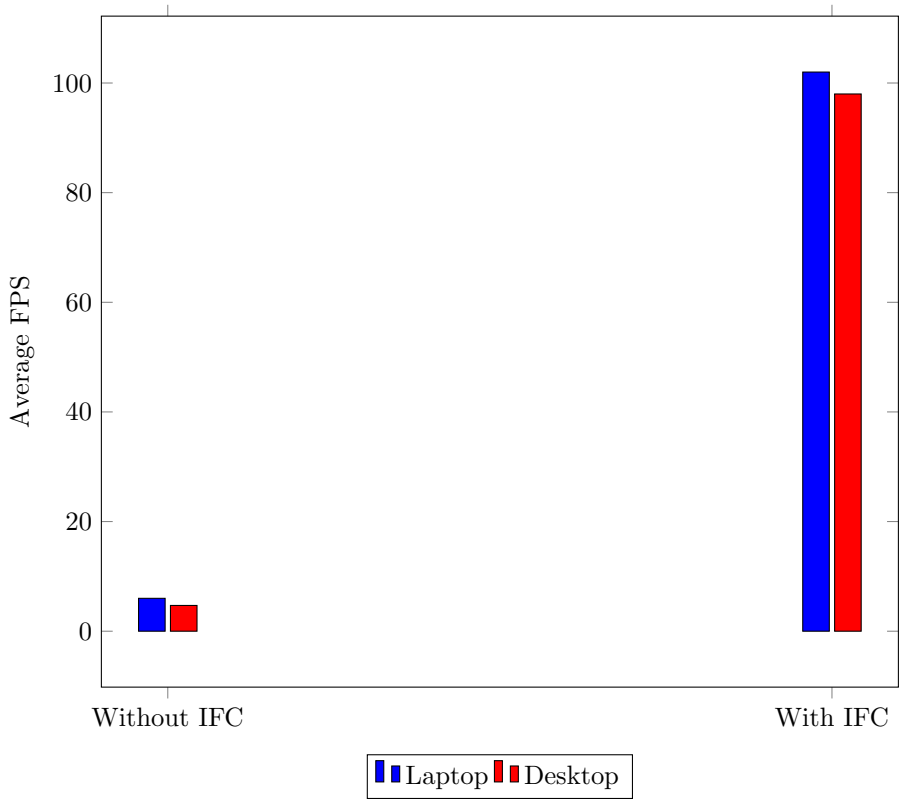

Figure 4.5: The average FPS of the game with and without the IFC optimisation. Data can be found in Appendix C and Appendix B.

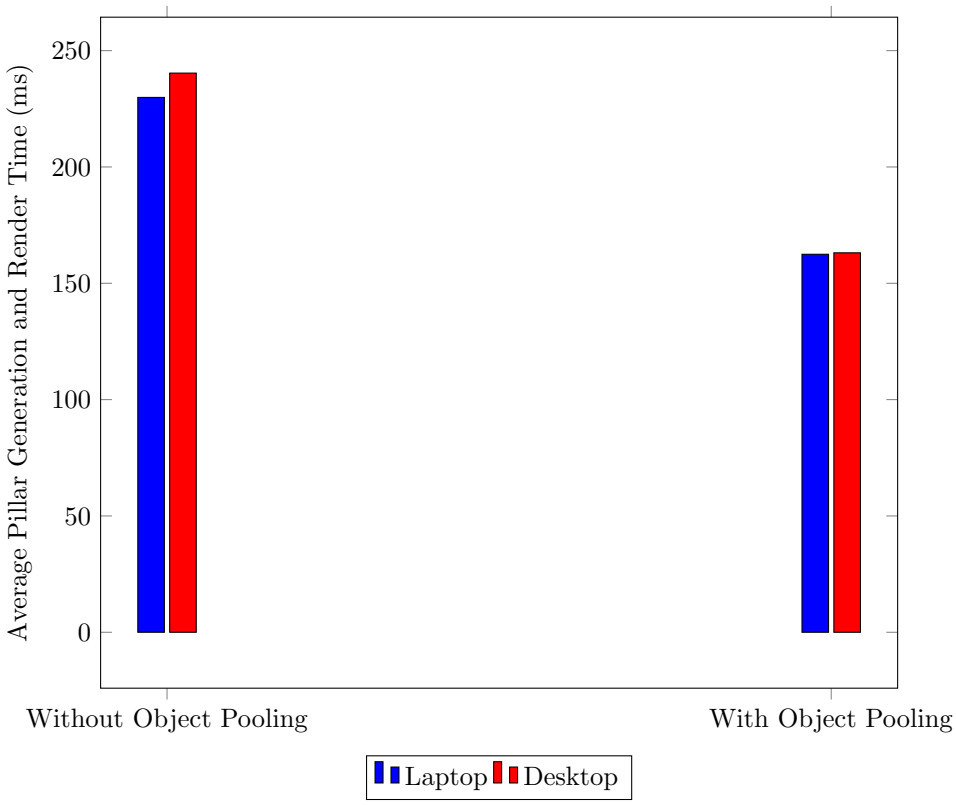

Figure 4.6: The average time taken to generate and render a pillar of 3 16x16x16 chunks, in milliseconds, with and without the object pooling optimisation for both pillars and voxel faces. Data can be found in Appendix D and Appendix B.

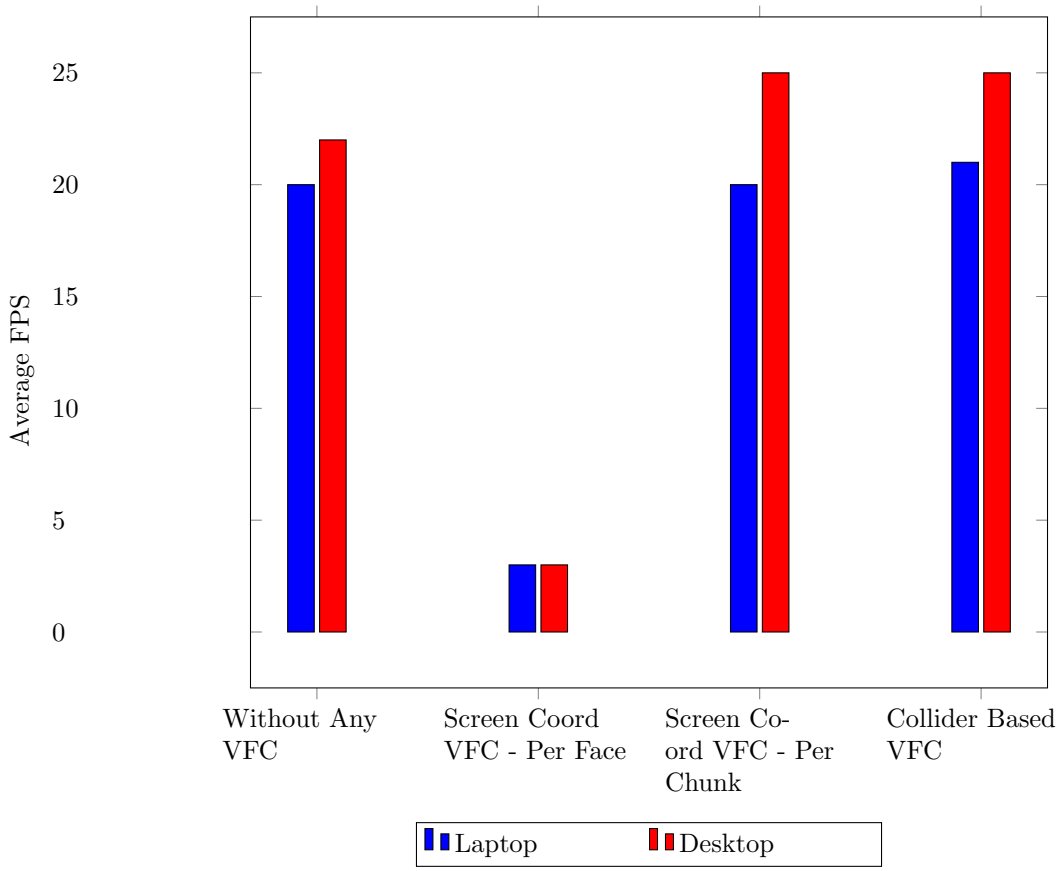

Figure 4.7: The average FPS of the game when several different approaches to VFC were used. Data can be found in Appendix E, Appendix F and Appendix B.

4.2.3 View Frustum Culling

As seen in Figure 4.7, the average FPS reduces by 85.7% from 21 to 3FPS when the fist implementation of VFC, screen coordinate based per face VFC. This is a significant reduction in frame rate, and so this implementation of VFC should not be used.

Figure 4.7 also shows that the average FPS increases by 9.5% from 21 to 23FPS when collider based VFC is used. Although this is an increase in frame rate, it should not be considered a significant optimisation, as frame rate values are rounded to the nearest integer, and so the true change in frame rate could be as little as 1FPS, which is statistically insignificant. Although this was the best implementation of VFC added to the project, it should not be used in a voxel rendering system, as the performance benefits are negligible.

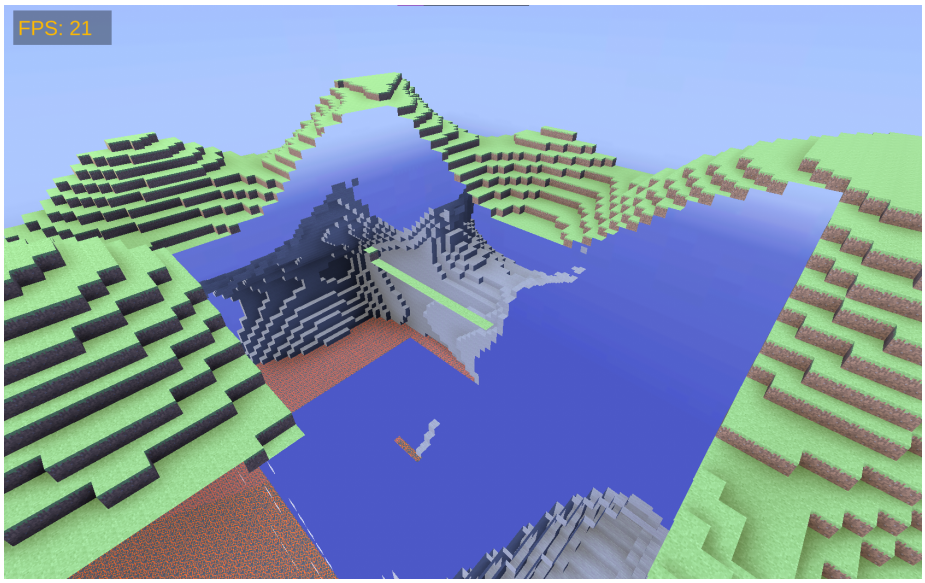

Figure 4.8: A screenshot of the terrain with screen coordinate per chunk VFC used. Several chunks are incorrectly culled, as this method is imprecise.

Similarly, Figure 4.7 shows an average FPS increases of 7.1% from 21 to 22.5FPS when screen coordinate based per chunk VFC is used. Again, this is statistically insignificant, and due to rounding of frame rate, could be a difference of less than 1FPS. Additionally, this implementation of VFC resulted in the over culling of chunks which were still visible, when none of a chunk's eight vertices were visible, as seen in Figure 4.8. This is an unacceptable graphical issue, and so this optimisation should not be used, regardless of the performance benefits.

It was later found that Unity automatically performs VFC, and so it is possible that the approaches to VFC implemented would be successful in a voxel rendering system. Despite this, the screen coordinate per face implementation of VFC would still have a significant negative impact on frame rate, as the algorithm to calculate the screen position of voxel face vertices would still run over approximately one million times per frame update, for a rendered volume of 112x48x112 voxels, which takes up far too much CPU time to grant a net positive effect to frame rate.

4.3 Limitations

- The voxel rendering system only supports fully transparent or fully opaque blocks, which are full cubes the size of a voxel.
- No appropriate lighting system was implemented; Voxel faces do not cast

shadows and are all lit from a directional light in the sky.

- No occlusion culling system was implemented fully, therefore the frame rate recorded could be improved. Due to this, the frame rate will decrease somewhat significantly when viewing the entire generated world from a distance, where all voxels are within the view frustum.
- When generating new pillars of terrain, the game will freeze until the pillars have been generated.
- In the data recorded, a frame rate of lower than 3 FPS was rounded to 3. The cause of this is currently unknown, but it may be due to rounding errors. This does not effect the findings significantly, as a frame rate of 3 FPS is considered very poor.

Chapter 5

Conclusions

This section discusses the extent to which the aim, objectives and requirements of this project have been met, summarises the findings of the the project, and identifies areas for future work.

5.1 Requirements

Definitions of requirements and importance rankings can be found in subsection 3.1.1.

Table 5.1: Evaluation of Requirements.

Requirement	Importance	Achieved
Rendering a single voxel	Mandatory	Yes. Voxels can be rendered with six faces to give the illusion of a solid cube.
Adding textures to voxels	Important	Yes. Voxels can be assigned both single texture and multi-texture materials.
Rendering a chunk	Mandatory	Yes. Chunks can be rendered as a 3D volume made of voxels.
Rendering pillars of terrain	Mandatory	Yes. Pillars are generated as a vertical stack of several chunks.
Generating world dynamically	Very Important	Yes. Pillars of terrain are generated within a specified range around the camera, and as the camera moves towards the edge of existing terrain, pillars that are too far away are removed, and new pillars are generated.
Rendering given list of blocks	Mandatory	Yes. Chunks are given a list of blocks, and can render the blocks specified.

Continued on next page

Table 5.1 – continued from previous page

Requirement	Importance	Achieved
Generating world blocks	Mandatory	Yes. There is an *evaluateBlock()* function, which can be given several steps to complete, which each influence the generation of terrain.
Generating hills	Very Important	Yes. Hills are generated using a perlin noise heightmap, and properties such as maximum hill height, minimum hill height, and hill slope shape can be adjusted.
Generating caves	Important	Yes. Caves are generated using processed perlin noise, and frequently connect together, creating large cave systems.
FPS display UI element	Very Important	Yes. FPS is displayed in the upper left corner of the screen at all times, and updates five times per second.
Internal face culling	Important	Yes. Voxel faces are only rendered if positioned between an opaque and transparent voxel, and are therefore exposed to the "air" of the game world, and visible.
Generating rivers	Optional	No.
Voxel lighting system	Optional	No.
View frustum culling	Important	Yes. Several different approaches were implemented, with an approach utilising Unity's physics system being the most performant.
Occlusion culling	Desirable	Partially. A complete and functional occlusion culling system was not able to be implemented with the resources available. Future work could complete the partial implementation that currently exists.
Object pooling	Important	Yes. Both pillar and voxel face game objects are pooled when terrain is unloaded, and these faces and pillars are then used in the generation of new terrain, instead of instantiating new game objects, which reduces terrain loading times by 30.7%.

5.2 Aim and Objectives

Definitions of aim and objectives can be found in section 1.3.

Table 5.2: Evaluation of Aim and Objectives.

Aim or Objective	Achieved	Description
Aim - Create an optimised voxel rendering and world generation system, which a competent third party could use to create a videogame	Yes	The voxel rendering system created could certainly be used to create a videogame, and the world generation system is easily expandable to add new features that may be required.
Objective - Carry out research into voxel graphics systems and procedural content generation	Yes	See chapter 2.
Objective - Create a voxel rendering system capable of rendering a list of given blocks	Yes	More information in Table 5.1
Objective - Create a world generation system with hills and caves	Yes	More information in Table 5.1
Objective - Optimise the game sufficiently	Partially	The game has been optimised, and has a significantly higher frame rate and lower load times compared to the unoptimised game, however a consistent frame rate of 25 is reached, which is just below the minimum desired frame rate, and the synchronous loading of terrain will cause stuttering as terrain is loaded. Therefore, this objective has only been partially achieved.

5.3 Summary of Output

A tech demo was created using the Unity game engine, which contains both a voxel rendering and world generation system. The user is presented with a main menu where they can change parameters such as render distance, the heights at which certain world features are generated, and which optimisations they would like to enable, and then a game world is generated. The generated game world consists of many thousands of voxels, and contains both hills and cave systems, generated using Perlin noise.

Several optimisations were added, and these include:

- Internal Face Culling, in which voxel faces which are not positioned on the boundary of an opaque and transparent block are not generated or rendered. This means that voxel faces that are inside terrain, and not visible to the player camera, are not unnecessarily generated.
- Object Pooling, where game objects such as pillars and voxel faces are not removed when a pillar is unloaded, but instead made invisible and placed in a "pool", from which they can be retrieved and used to generate new terrain at a later point in time. This reduces the amount of times new game objects must be instantiated in the Unity game engine, as this takes a considerable amount of time. [23]
- View Frustum Culling, where meshes that are not within the viewing frustum of the camera are disabled, as to not attempt to render polygons which are not within the area visible by the camera.

5.4 Summary of Findings

It was found that the IFC optimisation resulted in a 1,769% increase in frame rate for a world of 48x48x48 voxels, and that this increase in frame rate is even more significant in larger game worlds. See subsection 4.2.1. The object pooling optimisation was found to reduce terrain loading times by 30.7%, reducing the impact of freezing during the generation of new terrain. See subsection 4.2.1. Several approaches to VFC were implemented, although it was later found that Unity performs its own VFC automatically, so the effectiveness of the VFC optimisations implemented could not be determined. It was however found, that one of the approaches to VFC implemented, which used the screen space coordinates of voxel faces, was not suitable for use, as the frame rate was reduced by 85.7%. See subsection 4.2.3.

5.5 Future Work

Areas of future work to pursue are:

- More variations of blocks could be added, such as blocks that do not visually take up the entire voxel's space, voxels that have a semi-transparent texture, or that can be entered by the camera and have a visual effect, such as water blocks that add a blue hue as a post-processing effect.
- More world generation features could be added, such as trees, rivers, or foliage, for example flowers, bushes and piles of leaves.
- The partially implemented occlusion culling system could be completed, to further increase frame rate.
- A voxel lighting system could be added, in which certain blocks emit light, which attenuates as it gets further away from the source. Shaders could then change the appearance of voxel faces to reflect the light strength in an air voxel.
- A more consistent testing or benchmarking setup could be developed, in which the player camera moves according to a pre-defined path or animation, so that the same terrain is generated and viewed at any given point in the test.
- A system to import user defined worlds could be implemented, so that something other than the generated world could be shown, for example a voxel based model that the user has designed.
- Asynchronous generation of terrain could be implemented, so that the generation of terrain does not cause the rest of the game be temporarily unresponsive.

References

[1] R. Hiranand, *18 quintillion planets: The video game that imagines an entire galaxy — cnn business*, Jun. 2015. [Online]. Available: https://edition.cnn.com/2015/06/18/tech/no-mans-sky-sean-murray/index.html.

[2] "Minecraft franchise fact sheet april 2021," *Xbox*, Apr. 2021. [Online]. Available: https://news.xbox.com/en-us/wp-content/uploads/sites/2/2021/04/Minecraft-Franchise-Fact-Sheet_April-2021.pdf.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '88, New York, NY, USA: Association for Computing Machinery, 1988, pp. 65–74, ISBN: 0897912756. DOI: 10.1145/54852.378484. [Online]. Available: https://doi.org/10.1145/54852.378484.

[4] *Frames per second (fps) in tv, cinema, and gaming*, Jun. 2022. [Online]. Available: https://www.ionos.co.uk/digitalguide/server/know-how/fps/.

[5] *What are monitor refresh rates?* [Online]. Available: https://www.iiyama-monitors.co.uk/faqs/what-are-monitor-refresh-rates.

[6] *How much memory do you need for gaming?* [Online]. Available: https://www.kingston.com/unitedkingdom/en/blog/gaming/how-much-memory-for-gaming.

[7] B. Gapo, *How much vram do you need for gaming? [guide]*, Apr. 2023. [Online]. Available: https://www.gpumag.com/how-much-vram-gaming/.

[8] J. Dunstan, *Unity performance: Editor vs. standalone*, Feb. 2015. [Online]. Available: https://www.jacksondunstan.com/articles/2955.

[9] S. Axon, *Unity at 10: For better-or worse-game development has never been easier*, Sep. 2016. [Online]. Available: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/.

[10] U. Technologies, *Using components*. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/UsingComponents.html.

[11] U. Technologies, *Gameobjects*. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/GameObjects.html.

[12] U. Technologies, *Mesh renderer*. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/class-MeshRenderer.html.

[13] U. Technologies, *Scenes*. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/CreatingScenes.html.

[14] U. Technologies, *Object.instantiate*. [Online]. Available: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html.

[15] U. Technologies, *Object.destroy*. [Online]. Available: https://docs.unity3d.com/ScriptReference/Object.Destroy.html.

[16] U. Technologies, *Unity manual: Prefabs*. [Online]. Available: https://docs.unity3d.com/Manual/Prefabs.html.

[17] *How to make 3d models: The ultimate guide — aristek systems — aristek systems*. [Online]. Available: https://aristeksystems.com/blog/3d-design-for-everyone/#polygons.

[18] Aug. 2012. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/xna/ff634570(v=xnagamestudio.42).

[19] M. Su, R. Guo, H. Wang, S. Wang, and P. Niu, "View frustum culling algorithm based on optimized scene management structure," in *2017 IEEE International Conference on Information and Automation (ICIA)*, 2017, pp. 838–842. DOI: 10.1109/ICInfA.2017.8079019.

[20] T. Möller, *Occlusion culling algorithms*, Nov. 1999. [Online]. Available: https://www.gamedeveloper.com/programming/occlusion-culling-algorithms.

[21] K. Bormann, "An adaptive occlusion culling algorithm for use in large ves," in *Proceedings IEEE Virtual Reality 2000 (Cat. No.00CB37048)*, 2000, pp. 290–. DOI: 10.1109/VR.2000.840518.

[22] R. T. Bonet, *Object pooling in unity 2021+*, Mar. 2021. [Online]. Available: https://thegamedev.guru/unity-cpu-performance/object-pooling/#when-do-you-actually-need-pooling.

[23] *Introduction to object pooling*, May 2022. [Online]. Available: https://learn.unity.com/tutorial/introduction-to-object-pooling.

[24] U. Assarsson and T. Moller, "Optimized view frustum culling algorithms," Mar. 2000. [Online]. Available: https://www.cse.chalmers.se/~uffe/vfc_bbox.pdf.

[25] U. Technologies, *Cameras and depth textures*. [Online]. Available: https://docs.unity3d.com/Manual/SL-CameraDepthTexture.html.

[26] C. J. Hart, *Voxeldissertation/voxelface.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/voxel/VoxelFace.cs.

[27] C. J. Hart, *Voxeldissertation/worldgeneration.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/WorldGeneration.cs.

[28] C. J. Hart, *Voxeldissertation/pillargeneration.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/pillar/PillarGeneration.cs.

[29] C. J. Hart, *Voxeldissertation/pillardata.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/pillar/PillarData.cs.

[30] C. J. Hart, *Voxeldissertation/chunkgeneration.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/chunk/ChunkGeneration.cs.

[31] C. J. Hart, *Voxeldissertation/voxeldata.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/voxel/VoxelData.cs.

[32] C. J. Hart, *Voxeldissertation/voxelface.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/voxel/VoxelFace.cs.

[33] C. J. Hart, *Voxeldissertation/chunk.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/chunk/Chunk.cs.

[34] C. J. Hart, *Voxeldissertation/viewfrustumcullingcolliders.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/optimisations/ViewFrustumCullingColliders.cs.

[35] C. J. Hart, *Voxeldissertation/hzb-oc shader.shader at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/shaders/HZB-OC%20shader.shader.

[36] C. J. Hart, *Voxeldissertation/testing.cs at main · charliehart0/voxeldissertation*, May 2023. [Online]. Available: https://github.com/CharlieHart0/VoxelDissertation/blob/main/Unity%20Project%20Files/Assets/scripts/Testing.cs.

[37] U. Technologies, *Unity documentation - log files*. [Online]. Available: https://docs.unity3d.com/Manual/LogFiles.html.

[38] U. Technologies, *Unity - manual: What is multithreading?* [Online]. Available: https://docs.unity3d.com/2020.1/Documentation/Manual/JobSystemMultithreading.html.

Appendix A

Example Test Data


```
{
    "Test Comment" : "This is an example test of the test
        data recording system, here is where a comment or
        description of the test would go",
    "Date and Time" : "06/05/2023 13:20:41",
    "Environment Type" : "Editor",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
            3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 76310.3069000021,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : true
```

```
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 32,
                "Y" : 16,
                "Z" : 32
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
                "Max Height" : 15,
                "Min Height" : 1
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" : true
                ,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 43,
        "Data" : [
            211, 184, 288, 241, 325, 295, 288, 303, 340,
                118, 176, 241, 3, 17, 17, 16, 17, 16, 17,
                15, 16, 17, 16, 17, 16, 17, 17, 17, 14,
                16, 17, 16, 16, 17, 16, 16, 17, 17, 16,
                17, 16, 31, 33, 29, 30, 28, 20, 22, 22,
                28, 26, 26, 23, 27, 28, 26, 28, 23, 22,
                21, 18, 20, 20, 20, 20, 21, 20, 20, 21,
                24, 25, 6, 20, 18, 17, 18, 20, 20, 16, 14,
                 14, 12, 13, 12, 12, 13, 21, 30, 33, 23,
                31, 13, 19, 16, 16, 12, 14, 16, 17, 19,
```

```
                22, 18, 14, 16, 16, 15, 17, 23, 17, 16,
                20, 19, 20, 30, 33
            ]
        },
        "Pillar Generation and Render Times (ms)" : {
            "Average" : 591.359496000111,
            "Data" : [
                613, 618, 642, 635, 607, 601, 603, 597, 592,
                568, 524, 542, 631, 635, 611, 633, 585,
                580, 541, 610, 580, 541, 558, 555, 569
            ]
        },
        "Chunk Block List Generation Times (ms)" : {
            "Average" : 2.73969199985266,
            "Data" : [
                3, 2, 3, 2, 2, 1, 2, 3, 2, 1, 3, 2, 3, 2, 1,
                4, 3, 2, 2, 3, 2, 2, 1, 2, 3
            ]
        },
        "Voxel Rendering Time (ms)" : {
            "Average" : 0.0650291178318363
        },
        "Chunk Rendering Times (ms)" : {
            "Average" : 129.520135999918,
            "Data" : [
                132, 150, 182, 159, 163, 133, 137, 124, 108,
                119, 83, 73, 174, 194, 97, 187, 145, 102,
                100, 119, 131, 98, 88, 107, 120
            ]
        }
}
```

Listing A.1: An example of a JSON file created as output from the testing process.

Appendix B

All Optimisations Enabled Test Data


```
{
    "Test Comment" : "This test is on the Desktop
       computer, and has all optimisations enabled.",
    "Date and Time" : "08/05/2023 20:24:12",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
           ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
           3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 92538.0294999927,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
```

```
            "Colliders - Chunks" : true
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 25,
    "Data" : [
        4, 33, 34, 32, 32, 3, 25, 24, 28, 21, 31, 32,
            29, 19, 18, 21, 20, 23, 21, 26, 61, 62,
            48, 34, 33, 3, 26, 24, 23, 26, 28, 32, 15,
            28, 28, 26, 30, 3, 30, 28, 24, 28, 31,
            40, 10, 31, 30, 27, 29, 28, 34, 6, 27, 3,
            24, 31, 6, 24, 3, 20, 33, 43, 50, 34, 20,
            29, 22, 3, 21, 3, 23, 3, 21, 25, 24, 24,
            24, 28, 24, 33, 42, 33, 38, 29, 31, 33,
            32, 33, 32, 25, 29, 28, 26, 32, 3, 22, 24,
```

```
            3, 22, 21, 50, 44, 39, 23, 24, 23, 20,
            17, 20, 22, 19, 22, 19, 33, 20, 27, 21,
            22, 29, 28, 23, 3, 26, 27, 25, 26
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 163.100650000108,
        "Data" : [
            174, 190, 263, 239, 240, 258, 281, 244, 222,
                204, 206, 230, 240, 273, 247, 251, 241,
                204, 220, 212, 232, 198, 216, 184, 256,
                234, 240, 238, 225, 238, 282, 260, 230,
                250, 288, 228, 226, 192, 194, 220, 225,
                219, 208, 206, 205, 245, 356, 218, 218,
                143, 134, 134, 124, 131, 138, 143, 149,
                138, 127, 131, 138, 141, 138, 134, 138,
                154, 150, 154, 142, 141, 139, 122, 124,
                120, 111, 116, 114, 131, 125, 124, 115,
                120, 118, 122, 131, 117, 119, 97, 125,
                114, 122, 106, 133, 118, 133, 129, 124,
                134, 122, 124, 120, 123, 117, 135, 154,
                121, 119, 117, 119, 123, 114, 113, 143,
                123, 126, 126, 123, 120, 112, 175, 167,
                163, 157, 261, 146, 127, 141, 144, 120,
                122, 122, 117, 121, 142, 144, 130, 126,
                122, 114, 113, 127, 142, 126, 121, 126,
                131, 93, 131, 134, 137, 143, 134, 121, 110
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.41017510780653,
        "Data" : [
            5, 5, 2, 4, 4, 4, 4, 3, 2, 7, 5, 2, 4, 4, 2,
                4, 4, 1, 5, 5, 2, 4, 3, 2, 4, 4, 0, 4, 3,
                1, 4, 4, 3, 4, 4, 1, 4, 4, 1, 3, 4, 1, 3,
                3, 2, 4, 5, 1, 4, 4, 2, 4, 4, 2, 4, 4, 1,
                4, 4, 0, 3, 3, 1, 3, 4, 1, 3, 3, 2, 4, 4,
                2, 4, 4, 2, 4, 3, 2, 6, 3, 1, 4, 3, 1, 4,
                4, 2, 4, 4, 2, 4, 4, 2, 5, 4, 2, 4, 4, 2,
                3, 4, 2, 4, 4, 1, 4, 4, 1, 4, 4, 1, 3, 4,
                1, 4, 4, 0, 3, 4, 1, 4, 4, 2, 3, 4, 2, 4,
                3, 2, 4, 4, 1, 4, 3, 0, 4, 4, 2, 4, 4, 2,
                3, 4, 2, 4, 4, 2, 3, 4, 2, 4, 3, 0, 3, 4,
                0, 4, 3, 2, 4, 4, 1, 4, 4, 1, 4, 3, 1, 4,
                3, 2, 4, 3, 2, 3, 4, 1, 3, 4, 1, 4, 4, 0,
                3, 4, 1, 3, 3, 1, 4, 4, 1, 4, 4, 1, 4, 4,
```

```
                1, 3, 4, 2, 4, 3, 3, 4, 4, 2, 4, 3, 1, 3,
                4, 3, 4, 3, 2, 4, 3, 2, 4, 4, 1, 3, 4, 0,
                4, 3, 1, 4, 4, 1, 3, 4, 2, 4, 4, 2, 3, 4,
                1, 4, 4, 1, 4, 3, 1, 4, 4, 1, 4, 4, 2, 3,
                4, 2, 4, 4, 1, 4, 4, 1, 4, 4, 2, 4, 4, 2,
                4, 4, 1, 4, 3, 1, 4, 4, 2, 3, 4, 1, 4, 3,
                1, 4, 4, 1, 4, 4, 2, 3, 3, 1, 4, 4, 0, 3,
                4, 1, 3, 4, 2, 3, 4, 1, 3, 4, 2, 4, 4, 2,
                4, 4, 3, 4, 3, 2, 3, 4, 1, 4, 3, 1, 4, 3,
                1, 4, 4, 2, 3, 3, 2, 3, 4, 1, 4, 3, 1, 3,
                4, 1, 4, 4, 0, 3, 3, 1, 4, 4, 0, 4, 3, 2,
                3, 4, 1, 3, 4, 0, 4, 4, 1, 4, 4, 2, 5, 4,
                1, 5, 4, 2, 4, 4, 2, 3, 3, 2, 4, 4, 1, 3,
                4, 2, 3, 3, 2, 3, 4, 1, 4, 4, 1, 3, 4, 2,
                4, 4, 1, 4, 4, 1, 3, 3, 2, 4, 3, 3, 4, 3,
                1, 4, 4, 2, 3, 3, 2, 4, 4, 2, 4, 4, 1, 4,
                4, 2, 3, 4, 1, 4, 4, 2, 4, 3, 2, 3, 4, 1,
                4, 4, 2, 3, 3, 0, 4, 3, 0, 4, 3, 1, 4, 4,
                2, 4, 4, 1, 4, 4, 2, 3, 3, 1, 3, 3, 1
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00627778647264628
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 17.9642826839636,
        "Data" : [
            17, 2, 4, 18, 12, 8, 47, 35, 31, 26, 21, 23,
                28, 20, 22, 31, 32, 17, 39, 32, 22, 42,
                27, 25, 32, 28, 10, 18, 12, 17, 18, 10,
                20, 24, 13, 18, 19, 16, 19, 34, 31, 12,
                35, 35, 14, 27, 19, 18, 34, 20, 18, 21,
                13, 21, 25, 11, 15, 20, 17, 10, 31, 19,
                18, 23, 20, 11, 17, 4, 7, 19, 4, 7, 34,
                26, 25, 38, 18, 20, 34, 22, 5, 34, 26, 21,
                 17, 14, 24, 37, 29, 28, 40, 27, 27, 28,
                24, 21, 22, 11, 19, 21, 12, 24, 19, 70,
                17, 26, 28, 19, 28, 17, 19, 17, 15, 12,
                20, 22, 5, 29, 16, 17, 19, 11, 21, 19, 11,
                 19, 27, 10, 25, 25, 30, 5, 21, 23, 9, 25,
                 19, 26, 37, 19, 18, 18, 10, 19, 23, 12,
                17, 21, 16, 17, 22, 27, 5, 22, 20, 7, 15,
                11, 14, 16, 13, 11, 15, 17, 1, 9, 12, 22,
                19, 18, 14, 19, 13, 15, 18, 18, 7, 21, 16,
                 11, 28, 26, 9, 29, 15, 12, 25, 13, 14,
                22, 23, 4, 18, 20, 6, 19, 33, 14, 27, 14,
```

```
                14, 17, 11, 14, 17, 12, 13, 22, 13, 12,
                20, 14, 17, 16, 12, 15, 16, 11, 15, 20,
                12, 12, 15, 15, 5, 15, 14, 6, 15, 11, 11,
                18, 14, 17, 19, 12, 14, 17, 14, 9, 17, 10,
                 14, 17, 14, 12, 16, 19, 7, 16, 14, 15,
                20, 21, 13, 17, 15, 10, 19, 17, 6, 12, 6,
                4, 16, 19, 17, 17, 14, 11, 19, 24, 9, 11,
                4, 8, 21, 18, 15, 19, 18, 3, 21, 20, 17,
                23, 13, 16, 19, 15, 14, 25, 24, 11, 20,
                16, 11, 18, 15, 15, 16, 12, 15, 22, 13,
                13, 15, 12, 16, 24, 16, 19, 34, 25, 24,
                18, 22, 2, 16, 13, 11, 19, 12, 10, 24, 6,
                2, 17, 14, 14, 15, 12, 14, 15, 12, 13, 19,
                 14, 14, 17, 13, 9, 16, 19, 7, 20, 19, 10,
                 18, 11, 15, 16, 13, 10, 15, 16, 5, 31,
                25, 12, 31, 19, 16, 30, 21, 17, 18, 14,
                14, 21, 14, 124, 21, 13, 15, 17, 18, 6,
                20, 18, 15, 20, 13, 16, 16, 15, 7, 17, 17,
                 7, 15, 13, 15, 15, 12, 13, 20, 23, 4, 23,
                 17, 18, 21, 15, 16, 18, 13, 15, 16, 14,
                15, 10, 7, 21, 12, 11, 12, 16, 18, 6, 19,
                9, 17, 21, 19, 14, 17, 12, 16, 15, 10, 15,
                 20, 15, 16, 19, 16, 13, 14, 7, 2, 23, 23,
                 7, 23, 20, 11, 18, 13, 14, 23, 19, 14,
                23, 16, 15, 17, 12, 13, 15, 11, 8
            ]
        }
}
```

Listing B.1: The JSON file output from testing the fully optimised game on the desktop computer.

```
{
    "Test Comment" : "-",
    "Date and Time" : "08/05/2023 21:26:48",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
            2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
```

```
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 99443.0803000033,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : true
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 112,
                "Y" : 48,
                "Z" : 112
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
                "Max Height" : 43,
                "Min Height" : 30
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
               false,
            "Generate face objects only where needed" :
               true,
            "Narrow Caves at Surface" : true
        }
```

```
},
"FPS" : {
    "Average" : 21,
    "Data" : [
        3, 25, 20, 25, 26, 25, 26, 25, 25, 25, 26,
            24, 22, 21, 24, 24, 23, 23, 25, 24, 31,
            31, 10, 30, 31, 31, 27, 29, 30, 3, 25, 17,
             19, 21, 25, 25, 25, 25, 21, 17, 20, 20,
            22, 26, 3, 27, 15, 22, 22, 27, 30, 30, 28,
             22, 22, 19, 23, 27, 25, 30, 3, 31, 27,
            25, 20, 22, 24, 24, 21, 19, 19, 26, 32, 6,
             21, 22, 29, 21, 18, 20, 20, 22, 21, 24,
            23, 25, 7, 26, 22, 23, 28, 28, 3, 21, 20,
            23, 27, 26, 3, 28, 30, 28, 32, 31, 28, 31,
             3, 26, 22, 24, 3, 25, 27, 25, 26, 26, 26,
             6, 20, 21, 17, 23, 3, 17, 19, 17, 17, 20,
             18, 18, 18, 23, 6, 21, 25, 20, 20, 3, 20,
             21, 21, 21
    ]
},
"Pillar Generation and Render Times (ms)" : {
    "Average" : 162.481120779143,
    "Data" : [
        190, 220, 328, 285, 284, 298, 309, 280, 265,
            236, 235, 276, 295, 296, 274, 248, 248,
            247, 250, 245, 274, 244, 289, 214, 294,
            276, 276, 290, 262, 307, 315, 289, 268,
            287, 267, 245, 284, 231, 233, 261, 267,
            257, 253, 243, 238, 248, 465, 283, 281,
            107, 110, 108, 100, 109, 104, 100, 73, 87,
             120, 103, 110, 120, 123, 124, 112, 119,
            137, 139, 137, 142, 113, 106, 102, 104,
            95, 97, 99, 114, 114, 119, 123, 118, 119,
            99, 114, 114, 108, 102, 104, 112, 114,
            126, 131, 134, 137, 131, 109, 114, 111,
            105, 104, 89, 110, 101, 114, 116, 108,
            103, 111, 110, 102, 106, 113, 117, 113,
            110, 116, 100, 108, 114, 128, 98, 170, 99,
             102, 105, 122, 128, 140, 113, 102, 100,
            96, 82, 111, 100, 119, 105, 98, 109, 112,
            116, 113, 121, 109, 124, 137, 105, 104,
            106, 89, 106, 105, 100
    ]
},
"Chunk Block List Generation Times (ms)" : {
    "Average" : 3.41454610396257,
```

```
        "Data" : [
            5, 3, 2, 3, 4, 2, 4, 4, 2, 4, 4, 3, 6, 3, 3,
                4, 4, 2, 4, 4, 2, 3, 4, 2, 5, 4, 1, 4, 4,
                1, 3, 5, 2, 4, 4, 2, 4, 4, 1, 4, 3, 1, 3,
                3, 2, 4, 4, 2, 3, 3, 2, 5, 4, 3, 3, 5, 8,
                4, 4, 1, 4, 4, 0, 3, 4, 2, 4, 4, 3, 2, 3,
                1, 4, 12, 2, 4, 4, 1, 3, 4, 1, 4, 3, 2, 3,
                 3, 2, 4, 5, 2, 4, 3, 3, 4, 4, 1, 4, 4, 3,
                 4, 4, 1, 3, 3, 1, 3, 3, 1, 3, 5, 2, 4, 4,
                 2, 4, 4, 0, 4, 3, 0, 4, 4, 3, 3, 2, 1, 4,
                 4, 1, 3, 3, 0, 4, 4, 0, 4, 3, 2, 3, 4, 2,
                 4, 4, 2, 3, 5, 2, 4, 4, 2, 4, 4, 1, 3, 4,
                 2, 3, 3, 2, 4, 3, 1, 3, 3, 1, 3, 4, 1, 4,
                 3, 2, 3, 3, 0, 4, 4, 2, 3, 2, 3, 2, 4, 2,
                 3, 3, 1, 3, 3, 0, 4, 3, 2, 3, 4, 1, 4, 4,
                 2, 5, 4, 1, 4, 4, 2, 3, 4, 2, 4, 5, 2, 3,
                 4, 3, 3, 3, 1, 4, 2, 2, 3, 2, 0, 4, 4, 0,
                 3, 3, 0, 4, 3, 1, 4, 4, 2, 4, 3, 3, 3, 3,
                 2, 3, 3, 1, 4, 4, 2, 3, 2, 1, 3, 3, 1, 4,
                 4, 0, 3, 4, 3, 3, 4, 1, 4, 3, 0, 4, 3, 0,
                 4, 4, 2, 4, 4, 1, 2, 4, 3, 3, 3, 1, 3, 3,
                 3, 3, 5, 2, 3, 3, 3, 4, 4, 2, 4, 3, 3, 3,
                 3, 2, 3, 3, 1, 4, 4, 0, 5, 3, 1, 3, 3, 2,
                 4, 2, 1, 5, 4, 0, 4, 4, 2, 3, 5, 2, 3, 4,
                 1, 4, 4, 2, 4, 3, 1, 3, 3, 0, 4, 3, 2, 3,
                 4, 2, 4, 3, 1, 4, 3, 2, 4, 3, 1, 3, 4, 2,
                 3, 3, 1, 3, 4, 1, 3, 3, 2, 4, 4, 1, 4, 3,
                 2, 4, 4, 2, 3, 3, 1, 3, 3, 1, 3, 4, 0, 4,
                 4, 2, 3, 4, 1, 3, 5, 2, 4, 4, 3, 3, 3, 2,
                 3, 4, 1, 3, 4, 2, 3, 3, 1, 3, 4, 0, 4, 3,
                 0, 4, 4, 0, 3, 4, 1, 3, 2, 1, 3, 3, 1, 4,
                 3, 1, 3, 3, 3, 4, 3, 1, 4, 4, 2, 4, 3, 3,
                 4, 2, 2, 4, 3, 1, 4, 3, 1, 3, 4, 1, 4, 4,
                 1, 3, 4, 2, 3, 2, 2, 4, 5, 2, 3, 4, 2
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00588738640596217
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 17.1701692641104,
        "Data" : [
            20, 1, 5, 21, 14, 9, 59, 38, 37, 32, 23, 38,
                32, 22, 26, 33, 33, 18, 43, 33, 24, 46,
                29, 31, 36, 34, 12, 19, 12, 20, 21, 10,
                23, 23, 12, 20, 20, 17, 23, 34, 35, 15,
```

```
                44, 30, 15, 24, 16, 19, 22, 13, 20, 24,
                15, 26, 28, 11, 17, 23, 18, 11, 37, 21,
                23, 29, 22, 13, 16, 4, 6, 21, 4, 7, 42,
                30, 30, 41, 15, 20, 30, 24, 5, 42, 29, 26,
                 16, 12, 22, 46, 31, 31, 42, 31, 29, 31,
                24, 25, 24, 11, 25, 20, 11, 26, 22, 20,
                18, 27, 28, 13, 35, 33, 23, 20, 15, 14,
                23, 24, 5, 34, 16, 21, 21, 11, 26, 21, 11,
                 23, 32, 11, 30, 28, 31, 6, 21, 25, 10,
                23, 16, 26, 38, 25, 23, 19, 10, 22, 26,
                21, 38, 11, 16, 15, 24, 19, 5, 17, 11, 12,
                 13, 8, 12, 15, 11, 7, 13, 12, 7, 13, 13,
                5, 8, 1, 4, 10, 8, 5, 21, 16, 17, 13, 9,
                13, 13, 9, 13, 13, 14, 8, 14, 15, 12, 16,
                15, 17, 17, 16, 7, 26, 14, 12, 29, 28, 10,
                 28, 20, 12, 25, 14, 15, 34, 16, 14, 16,
                13, 14, 13, 11, 14, 14, 10, 13, 18, 11,
                12, 13, 13, 4, 12, 12, 6, 14, 10, 10, 20,
                18, 7, 15, 10, 14, 17, 12, 10, 15, 17, 1,
                10, 11, 13, 16, 15, 12, 15, 11, 9, 18, 20,
                 6, 18, 10, 16, 17, 11, 12, 16, 13, 9, 17,
                 10, 14, 16, 15, 16, 17, 23, 9, 19, 18,
                17, 23, 15, 15, 15, 11, 16, 20, 13, 13,
                26, 13, 12, 15, 11, 11, 17, 14, 15, 17,
                13, 12, 17, 12, 10, 18, 16, 6, 11, 6, 3,
                15, 16, 15, 15, 11, 10, 16, 17, 8, 17, 13,
                 15, 13, 10, 14, 15, 10, 13, 19, 13, 14,
                17, 12, 12, 16, 16, 4, 16, 12, 12, 17, 13,
                 13, 16, 13, 8, 15, 13, 7, 15, 15, 4, 15,
                10, 14, 13, 12, 9, 17, 19, 7, 19, 16, 11,
                24, 20, 6, 15, 8, 5, 67, 18, 14, 16, 12,
                9, 18, 16, 7, 18, 16, 10, 16, 13, 15, 14,
                14, 9, 15, 17, 12, 14, 10, 16, 13, 10, 13,
                 13, 9, 13, 15, 11, 9, 9, 3, 6, 18, 16,
                13, 17, 16, 3, 21, 18, 14, 20, 10, 14, 15,
                 11, 10, 20, 19, 8, 20, 17, 12, 19, 16,
                16, 17, 13, 15, 24, 14, 12, 14, 13, 15,
                24, 15, 19, 32, 21, 20, 17, 21, 1, 15, 13,
                 11, 18, 12, 10, 18, 5, 1, 17, 13, 13, 15,
                 11, 13, 15, 11, 13
            ]
        }
}
```


Listing B.2: The JSON file output from testing the fully optimised game on the laptop device.

```
{
    "Test Comment" : "This is a test with all
       optimisations active, but with a smaller scale
       world of 3x3x3 chunks",
    "Date and Time" : "08/05/2023 20:50:02",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
           3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 68039.3869000077,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : true
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 48,
                "Y" : 48,
                "Z" : 48
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
```

```
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 98,
    "Data" : [
        110, 146, 145, 141, 130, 114, 125, 160, 135,
            154, 144, 62, 89, 97, 3, 70, 75, 74, 126,
            126, 112, 93, 83, 74, 77, 76, 80, 81, 88,
            85, 89, 82, 83, 98, 115, 120, 128, 117,
            107, 69, 69, 65, 78, 99, 124, 81, 89, 117,
             153, 25, 111, 104, 86, 59, 60, 104, 104,
            95, 84, 70, 91, 95, 143, 3, 104, 86, 89,
            94, 92, 71, 128, 130, 111, 146, 112, 88,
            92, 115, 3, 105, 86, 105, 135, 158, 185,
            38, 141, 181, 104, 3, 107, 134, 22, 102,
            116, 166, 164, 3, 117, 27, 122, 154, 3,
            145, 3, 110, 138, 3, 149, 3, 101, 151,
            163, 138, 74, 126, 147, 119, 19, 107, 115,
             108, 109
    ]
},
"Pillar Generation and Render Times (ms)" : {
    "Average" : 145.580472222357,
    "Data" : [
        244, 235, 225, 181, 270, 229, 256, 211, 220,
            141, 130, 141, 134, 133, 150, 153, 141,
            156, 140, 133, 139, 129, 113, 109, 149,
            138, 112, 134, 132, 116, 118, 137, 124,
            122, 125, 147, 120, 123, 111, 112, 114,
            114, 120, 127, 125, 127, 126, 118, 120,
```

```
                117, 119, 127, 124, 120
            ]
        },
        "Chunk Block List Generation Times (ms)" : {
            "Average" : 3.52556481534316,
            "Data" : [
                4, 4, 1, 3, 4, 2, 4, 4, 1, 4, 4, 2, 4, 4, 2,
                    7, 4, 1, 4, 4, 2, 4, 4, 2, 3, 3, 2, 4, 4,
                    1, 4, 4, 1, 4, 4, 2, 6, 3, 2, 4, 4, 1, 4,
                    4, 2, 3, 3, 1, 4, 3, 1, 4, 3, 1, 4, 4, 1,
                    11, 3, 1, 3, 3, 0, 3, 4, 2, 3, 3, 1, 4, 4,
                     1, 4, 4, 1, 4, 4, 3, 4, 3, 3, 4, 4, 1, 4,
                     4, 1, 4, 4, 0, 4, 3, 1, 3, 3, 1, 3, 3, 1,
                     4, 4, 1, 4, 3, 4, 3, 4, 2, 3, 4, 1, 4, 4,
                     2, 4, 3, 2, 4, 4, 1, 4, 4, 2, 3, 4, 2, 3,
                     3, 0, 4, 3, 1, 4, 4, 1, 4, 3, 1, 4, 4, 1,
                     4, 4, 1, 3, 4, 2, 4, 4, 2, 3, 4, 2, 3, 3,
                     2, 3, 4, 0, 3, 3, 1
            ]
        },
        "Voxel Rendering Time (ms)" : {
            "Average" : 0.00626238362250032
        },
        "Chunk Rendering Times (ms)" : {
            "Average" : 16.6111074075287,
            "Data" : [
                43, 27, 21, 34, 22, 25, 32, 23, 17, 16, 4, 6,
                     42, 30, 31, 35, 13, 17, 40, 28, 26, 24,
                    20, 19, 20, 10, 22, 17, 16, 7, 20, 17, 3,
                    15, 10, 14, 17, 13, 14, 15, 11, 13, 17,
                    22, 11, 20, 17, 8, 20, 16, 13, 32, 17, 21,
                     22, 20, 6, 19, 14, 9, 23, 27, 11, 17, 13,
                     15, 15, 11, 13, 18, 13, 11, 21, 19, 13,
                    23, 12, 15, 14, 11, 14, 9, 10, 15, 17, 15,
                     12, 20, 12, 9, 9, 15, 5, 13, 15, 11, 17,
                    12, 12, 10, 18, 7, 16, 19, 7, 39, 12, 17,
                    15, 17, 14, 17, 15, 15, 16, 13, 11, 14,
                    17, 8, 8, 14, 13, 10, 18, 17, 9, 19, 3,
                    13, 14, 10, 14, 13, 14, 15, 22, 4, 23, 16,
                     11, 12, 16, 17, 15, 16, 14, 10, 15, 14,
                    15, 15, 14, 18, 14, 15, 20, 14, 12, 15,
                    16, 13
            ]
        }
}
```

Listing B.3: The JSON file output from testing the fully optimised game on the desktop computer, with a 3x3x3 chunk render area.


```
{
    "Test Comment" : "All optimisations, in 3x3x3 chunk
        render distance",
    "Date and Time" : "08/05/2023 21:28:37",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
            2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 64595.9759999961,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : true
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 48,
                "Y" : 48,
                "Z" : 48
```

```
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
                "Max Height" : 43,
                "Min Height" : 30
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
                false,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 102,
        "Data" : [
            162, 160, 153, 159, 156, 155, 165, 141, 165,
                161, 161, 28, 125, 176, 3, 115, 156, 148,
                97, 100, 132, 115, 123, 172, 3, 143, 161,
                120, 57, 76, 79, 70, 93, 148, 122, 73, 65,
                 83, 127, 39, 153, 95, 71, 3, 119, 114,
                92, 107, 70, 111, 3, 3, 96, 103, 146, 101,
                 116, 3, 128, 140, 144, 156, 150, 3, 113,
                123, 187, 35, 150, 3, 174, 3, 147, 125, 3,
                 16, 120, 184, 186, 178, 159, 165, 3, 126,
                 3, 112, 141, 93, 88, 120, 3, 94, 114, 3,
                102, 133, 20, 96, 187, 14, 105, 90, 102,
                93, 157, 3, 88, 102, 103, 106, 78, 47, 98,
                 39, 98
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 130.074177026749,
        "Data" : [
            243, 231, 237, 193, 237, 235, 267, 227, 228,
                119, 116, 132, 126, 125, 125, 120, 126,
                126, 128, 123, 124, 131, 140, 111, 106,
                110, 103, 139, 136, 102, 136, 117, 101,
```

```
            103, 112, 115, 130, 122, 104, 105, 116,
            126, 104, 108, 108, 117, 122, 106, 110,
            109, 124, 116, 110, 115, 136, 114, 112,
            115, 112, 112, 107, 108, 105, 102, 107,
            100, 109, 98, 109, 117, 115, 100, 101, 109
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.33863918993387,
        "Data" : [
            4, 3, 2, 3, 4, 2, 5, 5, 1, 3, 5, 0, 3, 4, 2,
            4, 5, 2, 3, 4, 1, 3, 4, 1, 3, 3, 3, 4, 3,
            0, 3, 4, 2, 4, 4, 1, 4, 3, 1, 3, 4, 0, 2,
            4, 1, 3, 3, 1, 5, 4, 0, 3, 2, 1, 4, 3, 3,
            3, 4, 2, 4, 4, 1, 4, 3, 1, 4, 4, 1, 3, 4,
            1, 3, 3, 1, 4, 3, 2, 4, 4, 1, 4, 4, 2, 3,
            4, 2, 4, 3, 2, 3, 4, 0, 3, 4, 1, 2, 3, 1,
            4, 3, 0, 3, 4, 2, 4, 4, 2, 3, 4, 2, 3, 4,
            3, 4, 4, 1, 3, 4, 0, 3, 3, 1, 3, 3, 3, 3,
            4, 1, 4, 4, 1, 4, 3, 1, 4, 3, 3, 4, 3, 1,
            4, 4, 1, 3, 2, 0, 4, 4, 1, 3, 4, 1, 4, 4,
            1, 4, 3, 2, 3, 3, 1, 4, 3, 2, 4, 3, 1, 3,
            3, 2, 4, 2, 2, 3, 4, 2, 12, 4, 1, 4, 3, 1,
             4, 3, 2, 3, 4, 2, 4, 4, 0, 4, 4, 1, 4, 3,
             1, 3, 4, 3, 4, 3, 1, 4, 3, 0, 4, 4, 0, 4,
             4, 2, 3, 4, 1, 2, 4, 0, 3, 4, 0
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00509783939030949
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 15.0089581081191,
        "Data" : [
            38, 24, 19, 29, 19, 23, 27, 22, 20, 13, 4, 5,
             31, 23, 22, 32, 11, 16, 38, 28, 26, 23,
            18, 17, 17, 9, 17, 15, 15, 6, 19, 14, 3,
            16, 10, 16, 19, 14, 11, 19, 12, 16, 15, 9,
             14, 22, 15, 9, 22, 25, 8, 26, 14, 12, 15,
             11, 13, 13, 10, 12, 18, 12, 14, 19, 17,
            6, 25, 15, 11, 17, 13, 9, 16, 10, 12, 20,
            13, 12, 13, 14, 4, 17, 14, 15, 20, 19, 11,
             14, 12, 6, 17, 13, 14, 20, 13, 10, 14, 9,
             13, 16, 17, 3, 13, 10, 13, 32, 11, 8, 15,
             12, 17, 12, 10, 13, 14, 14, 6, 16, 16, 2,
             15, 14, 14, 18, 13, 14, 15, 15, 10, 16,
```

```
                12, 9, 16, 15, 8, 18, 14, 15, 18, 12, 13,
                15, 12, 13, 16, 14, 7, 14, 15, 3, 17, 13,
                21, 18, 15, 10, 13, 7, 16, 16, 14, 11, 17,
                 29, 10, 15, 13, 10, 16, 11, 10, 19, 16,
                11, 19, 13, 10, 14, 10, 11, 18, 12, 12,
                15, 13, 10, 18, 15, 8, 16, 12, 9, 19, 12,
                12, 15, 18, 5, 17, 14, 15, 10, 8, 16, 19,
                21, 2, 23, 16, 11, 21, 9, 12, 15, 12, 13,
                13, 15, 10, 21, 23, 2
            ]
        }
}
```

Listing B.4: The JSON file output from testing the fully optimised game on the laptop device, with a 3x3x3 chunk render area.

Appendix C

No Internal Face Culling Test Data


```
{
    "Test Comment" : "This is a test without internal
        face culling. It had to be done with a 3x3x3 world
         due to crashes at full world scale",
    "Date and Time" : "08/05/2023 20:47:30",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
            3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 119873.412899986,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : false,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
```

```
            "Screen Coordinates - Chunks" : false,
            "Colliders - Chunks" : true
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 48,
            "Y" : 48,
            "Z" : 48
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 4.7,
    "Data" : [
        3, 5, 5, 5, 5, 4, 3, 3, 3, 3, 5, 5, 5, 5, 5,
            5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 4, 3, 4, 5,
            5, 5, 6, 6, 5, 4, 4, 4, 5, 4, 4, 4, 4, 4,
            4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 6,
            4, 4, 4, 4, 4, 4, 5, 5, 5, 6, 8, 12, 11,
            3, 10, 10, 10, 10, 12, 3, 4, 5, 3, 5, 5,
            3, 4, 5, 3, 4, 4, 3, 4, 5, 5, 7, 5, 5, 4,
            6, 7, 3, 7, 3, 3, 4, 4, 4, 4, 5, 4, 3, 3,
```

```
                5, 5, 5
            ]
        },
        "Pillar Generation and Render Times (ms)" : {
            "Average" : 1418.09652799964,
            "Data" : [
                264, 870, 1601, 1468, 257, 1108, 1235, 1451,
                    1588, 1730, 2237, 251, 1317, 2551, 1658,
                    1334, 1518, 831, 1527, 1903, 1349, 1730,
                    2537, 1338, 1786
            ]
        },
        "Chunk Block List Generation Times (ms)" : {
            "Average" : 3.46644666651885,
            "Data" : [
                5, 3, 1, 3, 3, 1, 4, 3, 2, 4, 3, 1, 4, 4, 1,
                    4, 3, 2, 4, 3, 1, 3, 4, 2, 4, 5, 1, 4, 4,
                    2, 4, 4, 1, 4, 3, 2, 4, 4, 3, 4, 4, 1, 4,
                    3, 1, 4, 4, 2, 4, 4, 0, 3, 3, 1, 4, 3, 2,
                    4, 4, 2, 3, 3, 1, 4, 4, 2, 4, 4, 1, 4, 4,
                    1, 4, 4, 1
            ]
        },
        "Voxel Rendering Time (ms)" : {
            "Average" : 0.165285684743903
        },
        "Chunk Rendering Times (ms)" : {
            "Average" : 433.545434667269,
            "Data" : [
                38, 43, 6, 314, 365, 121, 567, 584, 284, 501,
                     582, 222, 40, 13, 12, 471, 264, 225, 308,
                     577, 191, 509, 498, 272, 584, 561, 267,
                    625, 666, 285, 1063, 958, 123, 81, 40, 43,
                     343, 497, 395, 1074, 1093, 301, 504, 830,
                     243, 645, 448, 151, 774, 603, 41, 409,
                    326, 5, 572, 605, 247, 844, 748, 212, 768,
                     314, 169, 824, 677, 119, 1143, 1172, 140,
                     813, 371, 73, 734, 808, 149
            ]
        }
}
```


Listing C.1: The JSON file output from testing the partially optimised game without internal face culling on the desktop computer, and a 3x3x3 chunk render area.

```
{
    "Test Comment" : "test without internal face culling.
        had to be performed with 3x3x3 chunk render area
       as the unity player will crash without this
       optimisation at a larger world scale",
    "Date and Time" : "08/05/2023 21:37:17",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
            2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 82239.7492000014,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : false,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : true
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 48,
                "Y" : 48,
                "Z" : 48
            },
            "World Seed" : 1234567,
```

```
            "Hill Properties" : {
                "Max Height" : 43,
                "Min Height" : 30
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
                false,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 6,
        "Data" : [
            3, 7, 7, 7, 6, 5, 4, 5, 4, 3, 3, 3, 3, 6, 6,
                6, 6, 6, 6, 5, 6, 4, 3, 3, 3, 3, 3, 3, 3,
                3, 5, 3, 5, 8, 10, 11, 13, 14, 14, 3, 5,
                5, 5, 6, 6, 5, 6, 5, 6, 6, 6, 3, 8, 9, 9,
                8, 9, 9, 8, 8, 8, 8, 9
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 1572.28782857174,
        "Data" : [
            934, 1490, 1351, 216, 1074, 1186, 1269, 1479,
                 1518, 2103, 215, 1140, 2710, 2897, 2398,
                1895, 2640, 1961, 2994, 855, 681
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.41452222258326,
        "Data" : [
            4, 4, 1, 4, 4, 2, 4, 7, 2, 4, 4, 2, 4, 4, 1,
                4, 2, 2, 4, 3, 2, 4, 3, 1, 4, 3, 2, 4, 4,
                1, 3, 4, 2, 4, 3, 3, 3, 3, 0, 2, 4, 1, 3,
                3, 2, 3, 3, 2, 4, 3, 3, 3, 4, 2, 4, 3, 0,
                3, 3, 1, 3, 3, 2
        ]
```

```
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.200840376237849
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 487.173669840845,
        "Data" : [
            320, 340, 109, 511, 522, 280, 465, 553, 167,
                33, 6, 11, 432, 256, 229, 297, 531, 189,
                423, 436, 252, 536, 507, 259, 528, 547,
                259, 1019, 897, 110, 74, 37, 39, 307, 447,
                 322, 1286, 1282, 72, 1356, 1200, 270,
                934, 877, 500, 497, 732, 602, 1119, 926,
                525, 967, 713, 214, 1301, 1468, 161, 257,
                480, 53, 85, 201, 333
        ]
    }
}
```


Listing C.2: The JSON file output from testing the partially optimised game without internal face culling on the laptop device, and a 3x3x3 chunk render area.

Appendix D

No Object Pooling Test Data


```
{
    "Test Comment" : "This is a test on a 7x3x7 chunk
       world with object pooling  disabled for both
       pillars and voxel faces.",
    "Date and Time" : "08/05/2023 20:53:04",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
           3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 105189.921399996,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : false,
                "Faces" : false
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
```

```
            "Screen Coordinates - Chunks" : false,
            "Colliders - Chunks" : true
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 23.349,
    "Data" : [
        4, 31, 31, 32, 32, 32, 26, 26, 16, 22, 21,
           22, 26, 30, 27, 25, 22, 22, 26, 26, 29,
           29, 28, 27, 30, 30, 28, 26, 23, 24, 20,
           26, 24, 23, 25, 26, 26, 3, 22, 21, 21, 22,
            19, 22, 3, 22, 20, 3, 21, 24, 22, 18, 29,
            3, 23, 22, 23, 24, 3, 23, 20, 24, 3, 24,
           3, 20, 17, 22, 3, 20, 17, 23, 22, 20, 45,
           46, 30, 38, 38, 40, 23, 28, 26, 37, 29,
```

```
            27, 28, 32, 33, 27, 3, 27, 33, 3, 27, 27,
            3, 25, 26, 28, 3, 24, 26, 3, 27, 27, 16,
            19, 26, 28, 34, 33, 28, 24, 20, 20, 24,
            27, 27, 27, 26, 26, 26, 27, 25, 26
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 240.347384506968,
        "Data" : [
            247, 111, 166, 193, 275, 261, 252, 264, 251,
            252, 224, 269, 248, 225, 261, 272, 234,
            237, 219, 223, 220, 297, 327, 226, 208,
            188, 273, 260, 282, 270, 217, 242, 254,
            306, 225, 246, 291, 206, 225, 212, 213,
            238, 246, 318, 207, 216, 263, 247, 274,
            238, 239, 262, 262, 230, 240, 247, 266,
            364, 238, 220, 221, 240, 249, 245, 232,
            217, 209, 222, 222, 218, 233, 216, 229,
            225, 201, 237, 240, 266, 292, 231, 216,
            202, 216, 206, 217, 232, 234, 213, 219,
            236, 248, 224, 435, 274, 229, 235, 223,
            233, 244, 229, 216, 215, 217, 210, 214,
            219, 222, 203, 213, 272, 282, 283, 354,
            291, 222, 203, 200, 213, 247, 228, 243,
            221, 206, 225, 230, 272, 369, 219, 220,
            210, 210, 224, 231, 232, 244, 221, 211,
            202, 249, 286, 277, 238
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.55859084525299,
        "Data" : [
            4, 3, 0, 3, 4, 2, 3, 3, 2, 3, 4, 0, 4, 5, 2,
            3, 4, 3, 3, 4, 2, 4, 4, 1, 4, 4, 2, 3, 4,
            2, 3, 4, 1, 4, 4, 2, 5, 4, 2, 4, 4, 1, 4,
            4, 1, 4, 4, 1, 4, 4, 0, 4, 5, 1, 4, 4, 1,
            4, 4, 2, 4, 4, 1, 4, 4, 1, 5, 5, 1, 5, 4,
            1, 4, 3, 3, 4, 4, 2, 3, 4, 2, 4, 4, 2, 4,
            3, 0, 4, 4, 1, 4, 4, 2, 3, 4, 3, 5, 4, 2,
            4, 4, 2, 4, 4, 2, 4, 4, 1, 7, 4, 2, 4, 4,
            0, 4, 3, 2, 4, 4, 1, 4, 3, 0, 4, 3, 1, 4,
            4, 2, 3, 4, 1, 4, 3, 2, 3, 3, 0, 4, 5, 1,
            4, 3, 2, 4, 4, 2, 3, 4, 2, 4, 4, 2, 3, 4,
            2, 4, 4, 1, 4, 4, 1, 4, 3, 1, 3, 3, 1, 4,
            3, 1, 4, 5, 2, 4, 4, 2, 5, 4, 2, 4, 4, 2,
            4, 4, 1, 4, 5, 0, 4, 4, 1, 4, 4, 1, 3, 4,
```

```
            1, 4, 4, 2, 4, 4, 1, 4, 4, 1, 4, 3, 0, 4,
            4, 1, 4, 4, 1, 3, 4, 0, 4, 4, 1, 4, 4, 4,
            4, 4, 3, 5, 4, 1, 4, 4, 1, 4, 4, 1, 4, 3,
            2, 4, 4, 1, 4, 4, 1, 3, 4, 3, 4, 3, 1, 4,
            4, 1, 4, 4, 2, 4, 4, 2, 3, 4, 1, 4, 3, 1,
            4, 4, 1, 4, 4, 2, 3, 4, 0, 5, 4, 1, 4, 4,
            2, 4, 4, 1, 4, 3, 2, 3, 4, 1, 4, 3, 1, 4,
            4, 3, 3, 4, 2, 4, 4, 1, 4, 4, 2, 4, 4, 3,
            4, 4, 0, 4, 4, 0, 4, 4, 1, 3, 4, 1, 4, 4,
            1, 4, 4, 2, 4, 4, 3, 4, 4, 2, 4, 4, 2, 4,
            4, 2, 4, 4, 2, 4, 3, 1, 4, 4, 0, 4, 4, 1,
            4, 4, 0, 4, 3, 1, 4, 3, 1, 4, 4, 1, 3, 4,
            1, 4, 4, 0, 4, 4, 0, 4, 4, 4, 4, 4, 1, 4,
            4, 1, 4, 4, 1, 4, 4, 3, 4, 4, 1, 4, 4, 0,
            3, 4, 2, 4, 4, 2, 3, 4, 2, 4, 3, 1, 4, 4,
            1, 4, 4, 3, 4, 4, 1, 4, 4, 2, 5, 5, 3, 4,
            4, 3, 3, 4, 1
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.0101398462865041
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 25.61623309843,
        "Data" : [
            41, 38, 6, 16, 17, 4, 15, 1, 4, 16, 12, 8,
                45, 42, 32, 36, 28, 31, 31, 27, 23, 40,
                29, 15, 34, 31, 22, 36, 26, 25, 37, 33,
                10, 32, 23, 22, 26, 23, 22, 26, 17, 20,
                33, 23, 19, 34, 39, 17, 39, 28, 14, 42,
                23, 15, 32, 19, 20, 29, 20, 24, 32, 19,
                18, 34, 70, 13, 49, 31, 22, 26, 24, 11,
                21, 7, 10, 17, 4, 6, 41, 45, 28, 48, 25,
                21, 47, 31, 4, 44, 32, 25, 17, 15, 24, 34,
                 26, 25, 39, 27, 27, 33, 27, 21, 27, 18,
                22, 26, 19, 23, 28, 33, 35, 23, 30, 8, 34,
                 24, 20, 26, 23, 14, 28, 29, 6, 35, 22,
                20, 26, 19, 24, 27, 18, 23, 28, 11, 25,
                30, 32, 6, 32, 34, 9, 29, 24, 23, 40, 24,
                22, 26, 18, 23, 31, 20, 21, 32, 26, 27,
                33, 29, 11, 37, 25, 15, 44, 37, 12, 42,
                27, 17, 42, 24, 31, 53, 29, 24, 30, 23,
                28, 25, 18, 22, 26, 18, 21, 40, 23, 21,
                36, 27, 7, 28, 45, 11, 29, 21, 19, 31, 23,
                 17, 27, 17, 17, 30, 22, 13, 28, 24, 11,
                28, 27, 7, 29, 19, 22, 28, 21, 13, 34, 31,
```

```
                    11, 30, 23, 14, 20, 7, 9, 31, 25, 28, 35,
                    26, 22, 32, 37, 6, 66, 37, 8, 32, 23, 23,
                    34, 21, 12, 23, 23, 12, 20, 12, 25, 24,
                   24, 10, 25, 11, 25, 24, 29, 27, 41, 22,
                   23, 25, 20, 19, 27, 19, 22, 31, 25, 20,
                   34, 24, 17, 28, 29, 5, 29, 28, 26, 39, 24,
                    20, 37, 21, 21, 37, 27, 26, 35, 29, 13,
                   41, 30, 16, 41, 31, 25, 39, 19, 23, 32,
                   22, 17, 35, 17, 21, 31, 20, 24, 27, 25,
                   12, 35, 27, 6, 29, 20, 21, 28, 23, 23, 17,
                    15, 22, 25, 21, 23, 37, 27, 28, 35, 25,
                   23, 35, 28, 22, 40, 33, 20, 40, 30, 21,
                   30, 27, 18, 21, 21, 15, 16, 14, 26, 28,
                   32, 8, 42, 35, 18, 28, 30, 18, 34, 25, 20,
                    32, 24, 19, 30, 30, 3, 37, 38, 6, 35, 27,
                    15, 36, 26, 24, 76, 22, 14, 27, 20, 20,
                   29, 21, 25, 28, 20, 18, 31, 25, 11, 29,
                   20, 25, 28, 21, 24, 31, 22, 23, 34, 29,
                   25, 32, 25, 19, 22, 21, 20, 16, 20, 19,
                   37, 34, 26, 35, 24, 26, 24, 7, 59, 27, 30,
                    26
            ]
    }
}
```


Listing D.1: The JSON file output from testing the partially optimised game without object pooling for pillars or voxel faces on the desktop computer.


```
{
    "Test Comment" : "test performed without pillar or
        voxel face object pooling",
    "Date and Time" : "08/05/2023 21:39:39",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
            2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 92781.808799997,
```

```
"Settings" : {
    "Optimisations" : {
        "Object Pooling" : {
            "Pillars" : false,
            "Faces" : false
        },
        "Internal Face Culling" : true,
        "View Frustum Culling" : {
            "Screen Coordinates - Faces" : false,
            "Screen Coordinates - Chunks" : false,
            "Colliders - Chunks" : true
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 20,
```

```
        "Data" : [
            4, 26, 26, 22, 21, 22, 22, 24, 23, 23, 24,
                24, 23, 24, 26, 26, 27, 26, 24, 25, 25,
                26, 25, 23, 25, 25, 24, 22, 28, 23, 22,
                22, 3, 23, 27, 33, 13, 20, 23, 27, 6, 17,
                19, 20, 3, 21, 22, 3, 22, 20, 23, 3, 21,
                14, 23, 26, 30, 13, 28, 21, 21, 22, 19,
                22, 27, 3, 22, 23, 23, 23, 23, 24, 25, 8,
                23, 23, 21, 20, 24, 3, 22, 20, 20, 18, 23,
                 25, 21, 23, 3, 21, 21, 20, 22, 28, 6, 14,
                 21, 3, 22, 21, 22, 3, 20, 20, 19
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 229.909165753383,
        "Data" : [
            187, 195, 263, 241, 241, 264, 269, 241, 232,
                218, 233, 235, 246, 258, 249, 240, 226,
                234, 222, 228, 285, 203, 202, 203, 235,
                247, 261, 251, 226, 266, 267, 237, 279,
                261, 253, 204, 227, 209, 210, 234, 239,
                234, 303, 249, 210, 222, 242, 264, 235,
                233, 229, 250, 215, 231, 247, 232, 221,
                203, 209, 223, 225, 211, 228, 221, 243,
                271, 256, 268, 274, 231, 224, 234, 221,
                218, 210, 222, 301, 207, 213, 214, 212,
                220, 223, 220, 209, 210, 171, 191, 258,
                223, 418, 206, 203, 212, 175, 190, 197,
                208, 208, 195, 220, 229, 216, 224, 217,
                224, 237, 237, 256, 220, 234, 216, 219,
                208, 203, 215, 231, 223, 214, 216, 226,
                216, 218, 334, 301, 198, 184, 218, 219,
                216, 212, 208, 220, 208, 186, 215, 225,
                217, 231, 254, 199, 215, 214, 256, 211,
                216
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.27142237409201,
        "Data" : [
            5, 3, 2, 4, 4, 1, 4, 3, 2, 2, 3, 2, 4, 3, 3,
                4, 3, 2, 3, 4, 1, 4, 4, 2, 3, 3, 0, 4, 3,
                2, 4, 3, 2, 3, 5, 2, 3, 4, 1, 4, 4, 1, 3,
                4, 1, 4, 4, 1, 3, 4, 0, 4, 4, 2, 3, 4, 1,
                3, 4, 0, 3, 4, 1, 4, 4, 2, 3, 2, 3, 4, 4,
                2, 3, 4, 2, 4, 4, 2, 3, 3, 0, 3, 4, 2, 3,
```

```
                7, 3, 4, 4, 3, 3, 3, 2, 4, 3, 1, 4, 4, 2,
                4, 3, 3, 3, 3, 2, 3, 4, 0, 3, 3, 1, 3, 3,
                1, 3, 3, 1, 4, 3, 2, 3, 4, 2, 3, 3, 1, 4,
                2, 1, 3, 3, 1, 3, 4, 1, 4, 4, 2, 4, 3, 2,
                4, 3, 2, 2, 4, 1, 3, 4, 2, 3, 3, 0, 4, 4,
                2, 4, 4, 2, 4, 4, 1, 4, 4, 1, 4, 3, 1, 4,
                4, 3, 2, 4, 0, 3, 4, 1, 3, 3, 0, 4, 3, 2,
                4, 3, 1, 4, 4, 1, 3, 3, 1, 3, 3, 2, 4, 5,
                3, 5, 3, 2, 5, 4, 1, 4, 2, 2, 5, 3, 2, 3,
                3, 1, 4, 4, 2, 4, 3, 1, 3, 2, 1, 4, 3, 2,
                2, 3, 2, 3, 4, 1, 4, 3, 0, 4, 4, 2, 3, 3,
                2, 3, 3, 1, 3, 3, 1, 3, 2, 3, 3, 3, 2, 4,
                3, 1, 3, 4, 2, 3, 3, 2, 4, 3, 2, 3, 3, 3,
                4, 3, 2, 4, 4, 2, 3, 3, 0, 3, 4, 0, 3, 3,
                1, 3, 3, 2, 2, 4, 3, 2, 3, 2, 3, 2, 1, 3,
                4, 1, 3, 4, 0, 2, 4, 2, 3, 3, 1, 3, 4, 1,
                3, 4, 1, 4, 3, 2, 3, 3, 3, 5, 4, 1, 4, 4,
                1, 4, 3, 1, 3, 3, 0, 3, 3, 1, 3, 3, 1, 3,
                2, 2, 3, 3, 3, 4, 4, 2, 3, 2, 1, 3, 3, 2,
                4, 3, 1, 3, 4, 1, 3, 4, 2, 4, 4, 0, 4, 4,
                2, 3, 4, 1, 4, 4, 1, 4, 4, 2, 3, 3, 1, 4,
                4, 2, 5, 4, 2, 4, 4, 1, 3, 4, 2, 2, 3, 1,
                4, 3, 2, 3, 4, 2, 3, 3, 3, 3, 4, 1, 3, 4,
                0, 3, 4, 1, 3, 4, 2, 3, 3, 2, 3, 4, 0, 4,
                4, 3, 4, 4, 0, 3, 3, 1, 4, 4, 1, 4, 3, 1,
                3, 3, 0
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00895925812519834
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 22.0572668949151,
        "Data" : [
            17, 1, 4, 17, 11, 7, 42, 32, 31, 27, 21, 23,
                26, 22, 21, 32, 31, 15, 30, 29, 20, 31,
                23, 24, 34, 30, 10, 23, 17, 18, 22, 15,
                20, 23, 14, 16, 31, 19, 18, 29, 30, 12,
                38, 28, 14, 30, 19, 14, 27, 15, 18, 29,
                24, 21, 29, 16, 16, 24, 21, 10, 34, 22,
                57, 22, 19, 10, 19, 7, 9, 21, 4, 13, 35,
                24, 24, 53, 19, 17, 61, 25, 4, 36, 24, 21,
                 17, 13, 25, 33, 26, 25, 35, 25, 24, 29,
                24, 19, 23, 16, 22, 24, 16, 22, 21, 16,
                17, 21, 22, 9, 29, 21, 24, 22, 19, 13, 24,
                 23, 4, 30, 19, 17, 22, 16, 21, 23, 16,
```

```
                20, 28, 11, 25, 28, 43, 5, 22, 26, 7, 25,
                19, 21, 36, 21, 19, 21, 15, 20, 25, 17,
                18, 21, 26, 23, 35, 31, 7, 27, 19, 17, 22,
                 15, 20, 28, 22, 12, 25, 22, 11, 22, 22,
                7, 32, 19, 20, 21, 15, 18, 22, 15, 19, 25,
                 20, 18, 27, 20, 14, 23, 24, 2, 24, 21,
                22, 33, 20, 16, 37, 21, 24, 37, 28, 24,
                35, 29, 12, 41, 30, 16, 42, 31, 24, 32,
                16, 21, 30, 20, 17, 31, 17, 22, 25, 17,
                22, 25, 23, 12, 22, 23, 6, 27, 18, 19, 26,
                 22, 84, 26, 22, 8, 26, 16, 20, 28, 15,
                20, 26, 23, 9, 25, 16, 20, 23, 16, 22, 22,
                 16, 18, 27, 27, 1, 24, 21, 14, 14, 1, 5,
                18, 11, 8, 35, 29, 30, 23, 16, 24, 27, 18,
                 178, 28, 24, 6, 26, 17, 12, 30, 19, 14,
                16, 5, 1, 19, 11, 6, 23, 13, 11, 34, 16,
                9, 30, 24, 9, 21, 19, 9, 32, 17, 23, 31,
                22, 23, 30, 30, 6, 28, 25, 18, 23, 16, 21,
                 29, 21, 24, 29, 19, 16, 33, 24, 10, 32,
                20, 15, 35, 25, 6, 29, 25, 3, 31, 28, 8,
                28, 24, 21, 22, 18, 15, 24, 22, 6, 26, 22,
                 13, 29, 23, 21, 31, 16, 19, 24, 23, 10,
                24, 16, 20, 29, 29, 10, 24, 18, 21, 24,
                16, 21, 100, 19, 19, 25, 17, 22, 14, 18,
                13, 15, 13, 7, 17, 16, 4, 28, 26, 3, 28,
                16, 17, 24, 17, 21, 14, 17, 24, 29, 25,
                17, 27, 20, 13, 14, 17, 7, 35, 27, 4, 30,
                17, 22, 17, 24, 21, 32, 23, 19, 29, 20,
                17, 13, 12, 20, 28, 25, 10, 25, 22, 10,
                38, 32, 28, 23, 17, 17, 27, 17, 18
            ]
        }
}
```


Listing D.2: The JSON file output from testing the partially optimised game without object pooling for pillars or voxel faces on the laptop computer.

Appendix E

No View Frustum Culling Test Data


```
{
    "Test Comment" : "This is a test without any form of
       VFC",
    "Date and Time" : "08/05/2023 21:02:09",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
           ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
           3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 102544.9965,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
```

```
            "Colliders - Chunks" : false
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 22,
    "Data" : [
        4, 27, 28, 28, 27, 26, 28, 28, 24, 27, 31,
            35, 29, 29, 29, 29, 26, 26, 16, 29, 31,
            22, 33, 35, 24, 23, 23, 29, 29, 9, 24, 23,
             21, 22, 22, 21, 20, 22, 21, 20, 19, 21,
            21, 26, 27, 25, 26, 28, 27, 27, 28, 3, 24,
             23, 31, 30, 29, 36, 30, 23, 22, 34, 37,
            9, 29, 37, 33, 29, 40, 14, 35, 36, 28, 26,
             17, 24, 3, 34, 35, 27, 30, 3, 34, 33, 29,
             38, 3, 34, 35, 3, 25, 24, 25, 24, 21, 20,
```

```
            21, 20, 24, 24, 23, 21, 23, 3, 24, 18,
            22, 3, 22, 18, 10, 17, 19, 22, 6, 21, 21,
            22, 3, 17, 3, 33, 20, 18, 18, 18, 17, 14,
            3, 8, 3, 14, 14, 15, 14, 14, 15, 14
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 157.881996825574,
        "Data" : [
            175, 189, 281, 224, 239, 269, 264, 237, 221,
                200, 196, 248, 261, 259, 246, 205, 218,
                208, 220, 209, 231, 198, 224, 191, 244,
                233, 237, 240, 224, 260, 282, 260, 236,
                238, 221, 202, 247, 196, 194, 221, 223,
                219, 207, 203, 200, 217, 360, 253, 217,
                128, 132, 124, 135, 139, 128, 142, 154,
                151, 145, 148, 140, 136, 139, 150, 153,
                201, 146, 144, 134, 138, 155, 153, 142,
                149, 138, 135, 136, 128, 132, 124, 124,
                124, 114, 114, 154, 155, 145, 147, 140,
                136, 137, 148, 133, 132, 125, 153, 120,
                115, 133, 127, 134, 125, 115, 116, 131,
                152, 148, 145, 139, 137, 136, 112, 141,
                113, 117, 121, 135, 119, 127, 131, 125,
                126, 128, 135, 130, 138, 117, 119, 134,
                125, 136, 134, 126, 145, 145, 153, 137,
                146, 144, 147, 129, 134, 124, 122, 123,
                118, 122, 131, 132, 133, 131, 122, 121,
                116, 138, 140, 124, 130, 122, 123, 122,
                130, 127, 137, 128, 133, 126, 129, 128,
                120, 120, 112, 117, 117, 117, 125, 125,
                116, 117, 120, 126, 129, 133, 130, 131,
                123, 101, 145, 127
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.35714179926554,
        "Data" : [
            4, 3, 2, 4, 3, 0, 4, 4, 2, 3, 3, 2, 4, 3, 2,
                4, 4, 1, 5, 3, 1, 4, 3, 2, 4, 4, 1, 3, 3,
                1, 3, 4, 2, 4, 4, 2, 4, 4, 2, 4, 3, 1, 4,
                4, 0, 4, 3, 1, 4, 3, 1, 4, 4, 2, 3, 3, 1,
                3, 4, 1, 4, 4, 1, 4, 3, 1, 4, 4, 3, 4, 4,
                2, 4, 4, 2, 4, 4, 1, 4, 3, 0, 4, 3, 2, 3,
                4, 2, 3, 3, 3, 5, 4, 3, 4, 5, 2, 4, 3, 2,
                4, 4, 2, 3, 4, 1, 4, 4, 0, 3, 5, 2, 4, 3,
```

```
            1, 3, 3, 0, 3, 3, 1, 3, 4, 2, 4, 4, 1, 5,
            3, 1, 4, 4, 1, 3, 3, 1, 4, 4, 1, 4, 4, 3,
            4, 3, 2, 4, 4, 2, 4, 4, 1, 4, 3, 1, 3, 4,
            1, 3, 3, 1, 4, 4, 2, 4, 4, 1, 4, 4, 1, 3,
            4, 1, 3, 4, 1, 4, 3, 1, 3, 4, 1, 4, 4, 1,
            4, 4, 2, 3, 3, 2, 3, 4, 2, 4, 4, 2, 4, 4,
            2, 4, 5, 2, 4, 4, 3, 4, 4, 2, 4, 5, 2, 3,
            3, 2, 4, 3, 1, 3, 3, 1, 4, 4, 1, 3, 3, 1,
            3, 3, 1, 3, 4, 2, 4, 4, 1, 3, 4, 2, 4, 4,
            1, 3, 4, 1, 4, 4, 2, 3, 3, 1, 4, 4, 1, 4,
            3, 2, 4, 4, 1, 4, 4, 1, 4, 3, 1, 4, 4, 1,
            4, 3, 2, 3, 4, 2, 4, 5, 2, 4, 3, 0, 3, 4,
            1, 4, 3, 2, 4, 4, 1, 4, 3, 0, 3, 3, 1, 4,
            4, 2, 4, 3, 2, 4, 4, 2, 4, 4, 2, 4, 4, 0,
            3, 4, 0, 4, 4, 2, 4, 3, 0, 4, 4, 1, 3, 4,
            1, 4, 4, 1, 4, 4, 1, 3, 3, 2, 3, 3, 1, 4,
            3, 2, 4, 4, 1, 4, 4, 2, 4, 4, 0, 4, 3, 1,
            3, 3, 1, 4, 3, 1, 4, 4, 3, 4, 4, 1, 3, 3,
            1, 4, 3, 1, 4, 3, 1, 4, 3, 1, 4, 4, 3, 4,
            4, 1, 4, 3, 1, 4, 3, 1, 4, 4, 1, 3, 4, 1,
            4, 4, 1, 4, 4, 1, 4, 3, 0, 3, 4, 2, 5, 4,
            2, 4, 4, 1, 4, 4, 3, 3, 4, 2, 4, 4, 1, 4,
            4, 1, 3, 4, 0, 4, 3, 0, 3, 4, 1, 3, 3, 2,
            4, 4, 0, 4, 3, 1, 4, 4, 3, 4, 4, 2, 3, 4,
            1, 4, 4, 1, 4, 4, 2, 3, 3, 1, 3, 4, 0, 3,
            4, 2, 3, 4, 2, 4, 4, 2, 4, 4, 2, 4, 4, 2,
            4, 4, 1, 3, 4, 2, 4, 3, 0, 3, 3, 0, 4, 4,
            2, 3, 4, 2, 3, 4, 2, 4, 4, 1, 4, 4, 1, 4,
            3, 0, 3, 4, 1, 4, 3, 1, 4, 4, 0, 3, 4, 1,
            3, 3, 2, 4, 3, 2, 4, 4, 2, 3, 3, 1, 4, 4,
            1, 3, 4, 1, 4, 4, 1, 4, 3, 2, 4, 3, 2, 4,
            4, 2, 4, 4, 1, 4, 4, 2, 4, 4, 1, 3, 4, 1,
            4, 4, 2, 3, 3, 2
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00625716833206067
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 17.6719541444823,
        "Data" : [
            17, 2, 4, 18, 12, 7, 53, 34, 33, 27, 20, 22,
            27, 20, 22, 33, 31, 16, 37, 30, 20, 38,
            26, 29, 32, 31, 10, 18, 12, 17, 18, 10,
            20, 21, 13, 19, 20, 16, 19, 31, 33, 11,
            38, 26, 12, 20, 15, 15, 25, 12, 18, 21,
```

```
17, 21, 25, 11, 16, 20, 16, 10, 32, 19,
19, 22, 20, 10, 20, 5, 7, 17, 4, 6, 34,
25, 25, 38, 15, 17, 35, 21, 5, 35, 25, 21,
 17, 13, 24, 38, 29, 26, 39, 27, 28, 28,
24, 22, 31, 12, 21, 17, 11, 21, 18, 17,
15, 22, 24, 11, 30, 31, 19, 18, 14, 12,
20, 22, 5, 29, 15, 18, 18, 10, 21, 19, 11,
 19, 27, 9, 25, 24, 28, 5, 19, 23, 8, 22,
15, 21, 45, 22, 22, 18, 10, 19, 23, 12,
17, 19, 18, 10, 22, 17, 16, 22, 12, 13,
25, 17, 16, 27, 22, 16, 25, 23, 9, 30, 28,
 13, 21, 20, 15, 21, 20, 9, 22, 16, 13,
23, 15, 14, 15, 12, 11, 16, 12, 13, 18,
12, 13, 26, 27, 13, 30, 19, 19, 26, 71,
17, 28, 17, 18, 25, 17, 19, 24, 20, 9, 26,
 20, 12, 21, 19, 15, 20, 20, 9, 22, 15,
13, 30, 15, 14, 15, 12, 11, 15, 11, 13,
17, 12, 13, 21, 20, 11, 23, 16, 15, 22,
12, 12, 22, 14, 14, 19, 14, 15, 18, 15, 8,
 19, 14, 9, 21, 19, 15, 20, 19, 10, 22,
15, 13, 23, 15, 14, 16, 12, 10, 16, 11,
14, 16, 12, 12, 15, 20, 18, 26, 21, 7, 21,
 16, 15, 16, 13, 15, 43, 13, 8, 15, 15, 7,
 16, 16, 5, 24, 16, 16, 23, 12, 13, 26,
15, 16, 20, 14, 16, 19, 16, 8, 21, 15, 9,
20, 21, 18, 24, 20, 9, 23, 16, 13, 24, 16,
 14, 15, 12, 11, 15, 12, 14, 17, 12, 13,
16, 16, 5, 32, 15, 16, 16, 12, 11, 15, 12,
 13, 17, 15, 12, 19, 18, 10, 15, 17, 2,
17, 15, 14, 22, 15, 17, 18, 15, 13, 18,
15, 13, 20, 17, 13, 19, 15, 13, 19, 14,
13, 22, 14, 17, 11, 9, 19, 18, 21, 6, 25,
20, 11, 18, 15, 13, 19, 14, 13, 19, 15,
12, 17, 8, 16, 27, 18, 12, 23, 14, 17, 25,
 15, 14, 17, 13, 14, 20, 15, 16, 20, 16,
16, 22, 20, 14, 22, 19, 9, 17, 12, 12, 16,
 14, 9, 16, 12, 14, 15, 13, 15, 18, 18, 6,
 18, 16, 9, 20, 16, 15, 16, 13, 16, 18,
13, 15, 18, 14, 16, 15, 12, 14, 17, 15, 9,
 16, 14, 9, 19, 18, 15, 18, 14, 15, 13, 4,
 17, 14, 16, 16, 16, 14, 12, 14, 15, 11,
14, 14, 15, 19, 17, 8, 17, 17, 8, 19, 15,
17, 17, 12, 16, 18, 13, 15, 17, 14, 9, 20,
 17, 10, 22, 20, 6, 16, 12, 12, 17, 14,
13, 16, 17, 6, 17, 15, 9, 15, 12, 14, 16,
14, 13, 14, 17, 15, 14, 14, 12, 9, 13, 12,
```

```
                13, 18, 8, 13, 17, 14, 16, 18, 20, 17,
                19, 18, 19, 15, 17, 18, 14, 11, 17, 17,
                15, 19, 13, 12, 12, 12, 3, 31, 18, 19, 20,
                15, 15
        ]
    }
}
```

Listing E.1: The JSON file output from testing the partially optimised game without any form of view frustum culling on the desktop computer.

```
{
    "Test Comment" : "a test performed without any form
        of view frustum culling",
    "Date and Time" : "08/05/2023 21:41:49",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
            2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 96743.9174000025,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : false
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
```

```
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 20,
    "Data" : [
        4, 26, 26, 25, 26, 28, 28, 20, 15, 22, 25,
            28, 21, 22, 23, 28, 28, 31, 10, 26, 18,
            21, 21, 22, 24, 27, 26, 29, 3, 26, 32, 3,
            20, 21, 28, 9, 25, 27, 3, 26, 30, 3, 29,
            3, 23, 23, 31, 24, 23, 23, 27, 3, 23, 26,
            26, 25, 25, 27, 32, 32, 29, 8, 23, 27, 7,
            22, 24, 24, 5, 18, 17, 18, 21, 3, 20, 18,
            18, 22, 22, 26, 10, 22, 21, 20, 25, 3, 25,
             3, 23, 25, 31, 28, 28, 15, 23, 25, 3, 22,
             12, 24, 3, 23, 19, 22, 21, 22, 18, 21,
            12, 17, 21, 3, 23, 25, 9, 20, 17, 19, 3,
            19, 32, 19, 19, 3, 21, 19, 21
    ]
},
"Pillar Generation and Render Times (ms)" : {
    "Average" : 140.6861633332,
```

```
        "Data" : [
            184, 191, 259, 234, 232, 251, 269, 240, 221,
                205, 197, 226, 241, 241, 233, 230, 235,
                206, 217, 213, 236, 201, 222, 180, 251,
                230, 236, 237, 232, 234, 264, 244, 225,
                241, 269, 229, 222, 196, 197, 219, 223,
                217, 216, 205, 206, 235, 342, 221, 218,
                128, 114, 124, 123, 124, 117, 126, 78,
                113, 109, 97, 93, 93, 102, 126, 126, 110,
                118, 112, 117, 136, 106, 101, 103, 118,
                101, 101, 100, 103, 107, 105, 80, 86, 90,
                98, 120, 111, 107, 113, 110, 114, 121,
                104, 116, 116, 108, 103, 102, 108, 116,
                123, 117, 108, 115, 116, 127, 101, 114,
                110, 107, 102, 104, 120, 113, 112, 120,
                126, 128, 116, 126, 115, 111, 125, 135,
                131, 127, 131, 106, 124, 119, 135, 127,
                129, 118, 91, 103, 107, 112, 117, 113,
                115, 127, 114, 109, 120, 126, 117, 110,
                90, 121, 112, 105, 96, 99, 117, 124, 110,
                108, 106, 111, 102, 109, 119, 115, 110,
                109, 102, 111, 117, 123, 116, 113, 113,
                110, 112, 128, 113, 115, 109, 131, 110,
                116, 105, 172, 124, 111, 103, 103, 98,
                103, 121, 117, 127, 136, 98, 107, 99, 118,
                 114, 110, 111, 137, 127, 126, 136, 125,
                119, 99, 107, 132, 120
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.33077952372176,
        "Data" : [
            5, 3, 2, 3, 4, 1, 3, 3, 2, 6, 3, 4, 3, 3, 3,
                4, 3, 1, 5, 3, 2, 3, 4, 2, 4, 4, 0, 3, 4,
                2, 3, 3, 2, 3, 3, 1, 4, 3, 1, 4, 2, 1, 3,
                4, 1, 3, 4, 0, 5, 4, 1, 3, 3, 2, 3, 3, 2,
                4, 3, 1, 4, 3, 2, 4, 4, 2, 3, 4, 2, 3, 4,
                1, 4, 4, 2, 4, 3, 1, 4, 3, 1, 3, 4, 2, 3,
                4, 1, 3, 3, 3, 4, 3, 2, 4, 3, 3, 4, 4, 2,
                3, 4, 2, 3, 3, 1, 4, 3, 1, 4, 3, 1, 3, 3,
                1, 4, 3, 0, 4, 3, 1, 3, 4, 3, 4, 4, 1, 3,
                4, 2, 3, 3, 1, 3, 3, 1, 4, 4, 2, 4, 5, 2,
                4, 4, 2, 2, 3, 2, 3, 4, 3, 3, 3, 0, 4, 3,
                0, 4, 4, 1, 3, 3, 0, 4, 4, 1, 3, 4, 1, 2,
                3, 2, 2, 3, 3, 3, 3, 1, 3, 5, 2, 2, 3, 2,
                2, 4, 1, 3, 3, 0, 5, 4, 2, 3, 4, 3, 4, 4,
```

```
                1, 3, 4, 2, 4, 3, 3, 3, 3, 0, 3, 4, 2, 3,
                3, 2, 4, 4, 0, 4, 3, 1, 3, 4, 2, 3, 4, 2,
                3, 3, 1, 4, 4, 2, 3, 2, 1, 3, 4, 1, 4, 4,
                2, 4, 4, 2, 4, 3, 1, 4, 3, 1, 4, 3, 1, 4,
                4, 1, 3, 3, 1, 3, 3, 1, 4, 4, 2, 3, 3, 2,
                3, 4, 1, 4, 3, 1, 3, 3, 1, 4, 3, 2, 4, 3,
                1, 3, 4, 0, 3, 4, 0, 3, 3, 1, 3, 3, 2, 4,
                4, 1, 4, 3, 2, 4, 4, 2, 4, 3, 1, 5, 4, 1,
                3, 5, 1, 5, 4, 1, 2, 4, 0, 3, 4, 2, 2, 3,
                3, 3, 4, 2, 3, 4, 2, 4, 4, 1, 3, 4, 1, 4,
                3, 2, 4, 4, 0, 4, 3, 0, 3, 3, 0, 3, 4, 0,
                4, 3, 1, 3, 4, 1, 4, 4, 0, 4, 5, 0, 5, 4,
                0, 4, 4, 1, 3, 4, 2, 3, 3, 2, 4, 4, 1, 3,
                3, 2, 4, 4, 1, 3, 4, 2, 4, 5, 2, 3, 3, 1,
                4, 3, 1, 3, 3, 1, 2, 3, 1, 3, 3, 0, 3, 3,
                0, 3, 4, 2, 4, 3, 1, 3, 3, 2, 2, 3, 2, 3,
                3, 1, 3, 4, 0, 4, 3, 1, 4, 3, 1, 4, 4, 1,
                3, 3, 2, 3, 3, 1, 4, 4, 2, 3, 3, 1, 4, 2,
                2, 4, 2, 1, 2, 3, 3, 3, 3, 1, 4, 4, 1, 4,
                4, 2, 2, 3, 2, 2, 3, 1, 5, 3, 1, 4, 4, 0,
                3, 4, 0, 4, 3, 2, 4, 3, 1, 3, 3, 1, 4, 3,
                1, 3, 3, 2, 4, 3, 0, 4, 3, 1, 3, 4, 2, 3,
                3, 0, 2, 3, 1, 4, 4, 0, 3, 3, 0, 3, 3, 0,
                3, 4, 1, 4, 3, 2, 4, 3, 1, 4, 4, 1, 3, 3,
                3, 4, 4, 2, 4, 3, 2, 4, 3, 1, 3, 4, 1, 6,
                5, 3, 4, 3, 2, 4, 3, 2, 3, 4, 2, 3, 4, 2,
                4, 4, 1, 4, 4, 1, 5, 4, 2, 4, 3, 1, 3, 3,
                1, 4, 5, 2, 4, 3, 3, 3, 4, 2, 3, 3, 2, 3,
                3, 2, 4, 3, 2, 3, 3, 1, 4, 3, 1, 5, 5, 3,
                3, 3, 2, 4, 4, 0, 3, 4, 2, 4, 4, 3, 4, 4,
                2, 3, 3, 1, 3, 3, 0, 3, 3, 2, 4, 4, 1
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00603910686606581
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 15.3525457141655,
        "Data" : [
            17, 2, 4, 15, 11, 8, 41, 31, 28, 23, 18, 20,
                24, 17, 18, 24, 25, 14, 33, 28, 20, 35,
                24, 21, 27, 24, 9, 15, 10, 15, 14, 9, 17,
                14, 9, 16, 17, 14, 16, 23, 26, 10, 31, 24,
                 11, 21, 16, 14, 24, 12, 23, 18, 11, 19,
                28, 9, 12, 17, 16, 8, 26, 16, 16, 20, 16,
                11, 15, 5, 6, 16, 3, 6, 30, 24, 21, 34,
```

```
13, 14, 30, 18, 4, 30, 20, 20, 14, 13, 20,
 31, 24, 23, 32, 23, 22, 23, 20, 18, 19,
9, 17, 16, 10, 20, 16, 51, 15, 23, 24, 15,
 22, 15, 17, 15, 12, 11, 18, 18, 3, 25,
13, 16, 16, 9, 18, 15, 9, 17, 23, 9, 23,
21, 24, 4, 16, 20, 7, 20, 16, 22, 32, 15,
16, 15, 9, 18, 20, 10, 15, 16, 14, 16, 16,
 16, 6, 22, 14, 9, 23, 25, 8, 24, 16, 12,
21, 11, 12, 28, 13, 11, 9, 1, 4, 21, 12,
14, 20, 13, 8, 14, 12, 7, 10, 6, 14, 13,
13, 5, 14, 6, 15, 22, 23, 11, 24, 16, 17,
21, 11, 12, 25, 14, 13, 20, 15, 14, 23,
21, 9, 38, 25, 11, 21, 19, 1, 13, 14, 7,
16, 9, 12, 18, 10, 15, 14, 11, 13, 18, 13,
 7, 16, 16, 4, 18, 17, 4, 18, 11, 10, 19,
12, 9, 11, 3, 2, 12, 6, 5, 15, 8, 7, 20,
12, 6, 17, 12, 14, 13, 10, 13, 17, 10, 11,
 23, 11, 12, 14, 10, 13, 17, 11, 11, 20,
11, 14, 17, 16, 7, 20, 11, 17, 19, 16, 16,
 21, 20, 3, 17, 17, 6, 13, 9, 13, 14, 11,
10, 21, 17, 10, 22, 11, 16, 22, 11, 15,
18, 15, 8, 18, 17, 10, 17, 17, 11, 21, 15,
 12, 20, 19, 1, 22, 12, 14, 18, 11, 16,
13, 11, 19, 12, 7, 14, 14, 10, 11, 16, 15,
 14, 10, 14, 14, 15, 15, 7, 15, 18, 1, 14,
 14, 9, 14, 12, 12, 15, 12, 7, 25, 19, 4,
10, 14, 12, 12, 16, 5, 12, 12, 17, 20, 18,
 13, 19, 18, 8, 21, 14, 12, 24, 16, 16,
11, 17, 15, 21, 19, 14, 21, 15, 14, 25,
18, 15, 22, 15, 13, 23, 22, 7, 11, 21, 13,
 9, 9, 11, 17, 16, 6, 17, 13, 9, 16, 12,
15, 18, 10, 14, 14, 10, 15, 13, 13, 17,
18, 17, 15, 14, 19, 4, 9, 17, 4, 14, 17,
11, 21, 17, 13, 16, 11, 13, 17, 15, 7, 14,
 4, 6, 20, 13, 15, 17, 10, 12, 13, 10, 13,
 13, 4, 12, 18, 2, 14, 22, 17, 12, 22, 14,
 17, 16, 10, 15, 17, 14, 10, 11, 18, 7,
21, 11, 11, 14, 12, 9, 13, 10, 15, 19, 11,
 18, 17, 14, 12, 12, 14, 13, 10, 14, 13,
12, 17, 7, 18, 20, 8, 18, 10, 22, 20, 19,
10, 17, 16, 13, 22, 18, 4, 20, 18, 8, 17,
19, 8, 15, 15, 12, 22, 16, 19, 17, 18, 7,
19, 14, 10, 16, 15, 9, 21, 16, 16, 8, 10,
17, 21, 19, 8, 15, 14, 9, 31, 23, 24, 17,
11, 16, 15, 14, 11, 12, 12, 13, 10, 13,
13, 10, 16, 6, 15, 16, 7, 18, 17, 7, 19,
```

```
                15, 10, 25, 20, 16, 17, 13, 12, 9, 7, 15,
                17, 17, 6, 13, 13, 6, 18, 13, 16, 15, 14,
                12, 17, 13, 10, 18, 22, 1, 19, 15, 17, 21,
                 14, 14, 17, 15, 14, 21, 18, 17, 18, 10,
                15, 19, 19, 10, 14, 17, 2, 16, 14, 11, 24,
                 19, 19, 16, 11, 13
            ]
        }
}
```

Listing E.2: The JSON file output from testing the partially optimised game without any form of view frustum culling on the laptop computer.

Appendix F

Alternate View Frustum Culling Approaches Test Data


```
{
    "Test Comment" : "This is a full optimisation test,
        using the screen coord voxel faces VFC
        optimisation",
    "Date and Time" : "08/05/2023 20:59:47",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
            3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 58124.3504000008,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
```

```
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : true,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : false
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
            },
            "Render Distance (in voxels)" : {
                "X" : 112,
                "Y" : 48,
                "Z" : 112
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
                "Max Height" : 43,
                "Min Height" : 30
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
                false,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 3,
        "Data" : [
            3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 3
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
```

```
        "Average" : 230.907487999797,
        "Data" : [
            261, 79, 182, 251, 245, 230, 258, 258, 274,
                218, 197, 215, 218, 231, 243, 249, 247,
                216, 243, 200, 211, 231, 202, 253, 166,
                227, 230, 238, 243, 276, 222, 223, 211,
                206, 228, 262, 251, 263, 221, 211, 239,
                236, 244, 220, 230, 218, 241, 331, 230,
                243
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.47665800005198,
        "Data" : [
            5, 3, 1, 3, 3, 2, 4, 4, 1, 4, 4, 2, 4, 3, 3,
                3, 4, 1, 4, 3, 0, 5, 4, 1, 3, 3, 2, 3, 3,
                1, 4, 4, 1, 4, 3, 2, 4, 4, 2, 4, 4, 1, 4,
                4, 1, 3, 4, 1, 4, 4, 1, 3, 4, 1, 4, 4, 2,
                3, 4, 1, 4, 4, 0, 4, 3, 1, 3, 3, 1, 4, 4,
                2, 4, 4, 1, 3, 4, 1, 3, 3, 2, 3, 3, 1, 3,
                4, 2, 6, 4, 3, 4, 3, 3, 4, 3, 2, 3, 3, 2,
                4, 4, 2, 4, 4, 2, 3, 4, 1, 4, 5, 1, 4, 4,
                2, 4, 5, 1, 4, 3, 0, 4, 4, 1, 4, 3, 2, 4,
                3, 2, 4, 4, 2, 4, 4, 0, 4, 4, 1, 4, 4, 2,
                3, 4, 2, 4, 4, 2, 4, 4, 2
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00702266004732423
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 21.7302299998204,
        "Data" : [
            36, 41, 5, 9, 1, 3, 19, 10, 7, 29, 34, 33,
                33, 24, 29, 26, 21, 23, 30, 32, 15, 36,
                31, 21, 42, 30, 30, 32, 31, 11, 18, 12,
                16, 18, 10, 40, 20, 16, 16, 19, 18, 18,
                27, 30, 12, 43, 30, 14, 25, 24, 17, 29,
                13, 20, 21, 13, 21, 25, 11, 15, 20, 17,
                10, 32, 19, 19, 23, 20, 12, 16, 5, 7, 15,
                4, 5, 35, 25, 25, 39, 15, 17, 34, 23, 4,
                38, 25, 22, 20, 16, 34, 32, 22, 22, 32,
                22, 22, 23, 21, 18, 20, 11, 18, 17, 11,
                22, 49, 14, 18, 27, 31, 9, 31, 23, 24, 20,
                 17, 13, 25, 23, 5, 32, 17, 19, 19, 11,
                23, 20, 14, 22, 28, 11, 26, 32, 35, 6, 20,
```

```
                26, 9, 23, 16, 23, 42, 19, 20, 19, 11,
                21, 26, 13, 19
            ]
        }
}
```


Listing F.1: The JSON file output from testing the fully optimised game with the view frustum culling approach of comparing screen space coordinates of voxel faces, on the desktop computer.


```
{
    "Test Comment" : "full optimisation, with screen
       coord face VFC mode",
    "Date and Time" : "08/05/2023 21:33:30",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
           2.30GHz",
        "Cores" : 8,
        "Frequency (MHz)" : 2304
    },
    "RAM Total (MB)" : 16126,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX 3050
            Laptop GPU",
        "GPU VRAM (MB)" : 3979
    },
    "Test Duration (ms)" : 51432.2342000008,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : true,
                "Screen Coordinates - Chunks" : false,
                "Colliders - Chunks" : false
            }
        },
        "World Properties" : {
            "Chunk Dimensions (in voxels)" : {
                "X" : 16,
                "Y" : 16,
                "Z" : 16
```

```
            },
            "Render Distance (in voxels)" : {
                "X" : 112,
                "Y" : 48,
                "Z" : 112
            },
            "World Seed" : 1234567,
            "Hill Properties" : {
                "Max Height" : 43,
                "Min Height" : 30
            }
        },
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
                false,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 3,
        "Data" : [
            3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 3, 3, 3, 3, 3, 3
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 187.768574999939,
        "Data" : [
            186, 195, 260, 233, 234, 251, 268, 241, 221,
                205, 201, 227, 244, 242, 232, 232, 234,
                205, 220, 214, 232, 201, 222, 183, 252,
                230, 238, 253, 229, 233, 266, 246, 233,
                239, 266, 229, 223, 195, 196, 219, 230,
                219, 208, 205, 206, 236, 342, 218, 217,
                123, 112, 109, 124, 126, 122, 127, 106,
                110, 106, 112, 117, 115, 112, 112, 110,
                122, 113, 102, 118, 111, 101, 107, 102,
                98, 108, 100
```

```
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.38814999999707,
        "Data" : [
            4, 3, 2, 4, 3, 0, 3, 3, 2, 7, 4, 2, 4, 3, 1,
                3, 3, 1, 3, 3, 1, 4, 4, 3, 3, 4, 1, 3, 3,
                1, 3, 3, 3, 3, 4, 2, 4, 4, 2, 4, 4, 2, 4,
                3, 0, 4, 4, 1, 5, 5, 3, 4, 4, 3, 3, 4, 2,
                4, 4, 1, 4, 4, 2, 3, 3, 1, 5, 3, 3, 3, 3,
                2, 4, 3, 2, 4, 3, 1, 4, 4, 0, 4, 4, 1, 4,
                4, 2, 4, 3, 2, 3, 4, 3, 5, 3, 2, 4, 4, 3,
                3, 4, 1, 4, 4, 2, 4, 3, 0, 4, 4, 1, 3, 3,
                2, 4, 3, 1, 3, 4, 1, 3, 4, 2, 4, 4, 2, 4,
                3, 0, 3, 3, 0, 4, 3, 1, 3, 4, 3, 4, 4, 1,
                4, 4, 3, 3, 3, 2, 3, 4, 1, 3, 3, 1, 2, 3,
                0, 4, 3, 1, 4, 4, 0, 3, 4, 2, 4, 4, 1, 3,
                2, 1, 3, 4, 0, 3, 4, 2, 3, 4, 1, 3, 3, 0,
                4, 4, 2, 3, 3, 1, 4, 3, 0, 3, 3, 1, 3, 3,
                2, 4, 3, 1, 4, 4, 2, 4, 4, 1, 3, 4, 1, 3,
                4, 1, 2, 4, 2, 4, 3, 2, 3, 3, 1, 4, 3, 1,
                3, 3, 0
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00553164396941718
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 16.672872368181,
        "Data" : [
            18, 2, 5, 16, 11, 7, 42, 30, 28, 23, 19, 20,
                24, 17, 20, 24, 25, 15, 33, 28, 19, 36,
                23, 22, 27, 24, 9, 14, 10, 15, 15, 8, 17,
                15, 10, 16, 16, 14, 16, 23, 26, 10, 31,
                25, 10, 20, 16, 16, 23, 13, 24, 17, 11,
                19, 27, 9, 12, 18, 15, 8, 27, 16, 17, 21,
                17, 9, 16, 4, 6, 15, 3, 6, 29, 22, 22, 33,
                 12, 14, 30, 19, 3, 35, 25, 22, 15, 12,
                21, 31, 24, 23, 33, 23, 23, 23, 20, 18,
                18, 9, 17, 17, 9, 20, 16, 48, 14, 24, 23,
                15, 24, 15, 17, 15, 12, 10, 18, 17, 5, 24,
                 13, 16, 15, 9, 18, 16, 9, 17, 23, 9, 22,
                21, 23, 4, 16, 20, 7, 20, 16, 21, 32, 17,
                15, 15, 8, 18, 19, 10, 15, 17, 15, 15, 17,
                 16, 6, 18, 13, 9, 22, 23, 9, 23, 16, 12,
                21, 10, 14, 28, 14, 12, 20, 20, 3, 16, 16,
```

```
                7, 14, 9, 13, 15, 10, 9, 13, 15, 2, 8, 9,
                13, 14, 16, 11, 18, 18, 7, 15, 17, 8, 21,
                14, 9, 15, 15, 6, 10, 14, 6, 14, 15, 11,
               20, 13, 10, 15, 11, 8, 13, 9, 14, 12, 10,
               13, 14, 8, 13, 21, 11, 14, 15, 16, 4
        ]
    }
}
```

Listing F.2: The JSON file output from testing the fully optimised game with the view frustum culling approach of comparing screen space coordinates of voxel faces, on the laptop computer.

```
{
    "Test Comment" : "A test with full optimisations, and
        screen coordinate chunk VFC. Important to note
       that this form of VFC culls chunks far more often
       than it is supposed to, and so often large
       sections of the game world are culled. This
       improves frame rate dramatically at the cost of
       unacceptable visual issues",
    "Date and Time" : "08/05/2023 21:06:31",
    "Environment Type" : "Build",
    "CPU" : {
        "Name" : "AMD Ryzen 5 5600G with Radeon Graphics
            ",
        "Cores" : 6,
        "Frequency (MHz)" : 3893
    },
    "RAM Total (MB)" : 65380,
    "Graphics" : {
        "Graphics Device Name" : "NVIDIA GeForce RTX
           3060",
        "GPU VRAM (MB)" : 12142
    },
    "Test Duration (ms)" : 104384.161200002,
    "Settings" : {
        "Optimisations" : {
            "Object Pooling" : {
                "Pillars" : true,
                "Faces" : true
            },
            "Internal Face Culling" : true,
            "View Frustum Culling" : {
                "Screen Coordinates - Faces" : false,
                "Screen Coordinates - Chunks" : true,
```

```
            "Colliders - Chunks" : false
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
    "Misc Options" : {
        "Generate Random Blocks" : false,
        "Randomise Seed" : false,
        "Use World Generation System" : true,
        "Remove Pillars Dynamically" : true,
        "Generate Pillars Dynamically" : true,
        "Use Hill Shape Curve" : true,
        "Generate Terrain only when Q pressed" :
            false,
        "Generate face objects only where needed" :
            true,
        "Narrow Caves at Surface" : true
    }
},
"FPS" : {
    "Average" : 25,
    "Data" : [
        4, 25, 25, 26, 26, 25, 26, 26, 26, 18, 25,
            20, 22, 25, 24, 24, 23, 19, 21, 23, 26,
            24, 28, 22, 26, 27, 24, 26, 27, 26, 26,
            26, 21, 24, 25, 24, 23, 23, 8, 21, 18, 18,
             24, 23, 3, 27, 28, 25, 23, 26, 23, 26,
            40, 35, 35, 3, 27, 11, 24, 25, 25, 21, 30,
             27, 3, 22, 28, 22, 28, 29, 3, 33, 19, 6,
            23, 28, 34, 24, 18, 20, 26, 26, 23, 27,
            14, 21, 22, 36, 26, 20, 19, 34, 28, 30,
```

```
            30, 34, 36, 28, 29, 46, 42, 42, 38, 38,
            43, 21, 20, 18, 19, 3, 40, 3, 37, 3, 33,
            33, 32, 32, 33, 29, 16, 33, 31, 33, 32,
            33, 32, 33, 33, 38, 34, 35, 35, 32, 24,
            26, 26, 28, 13, 20, 34, 42, 3, 3, 30, 30,
            27, 30, 32
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 160.204940993778,
        "Data" : [
            176, 189, 264, 236, 250, 245, 286, 224, 224,
            208, 196, 237, 271, 260, 246, 210, 219,
            203, 221, 212, 230, 198, 217, 182, 254,
            229, 236, 237, 223, 235, 283, 269, 239,
            246, 223, 195, 231, 207, 196, 224, 230,
            226, 207, 204, 201, 217, 349, 258, 217,
            146, 139, 127, 151, 156, 141, 157, 151,
            132, 128, 118, 111, 110, 113, 147, 150,
            150, 154, 157, 148, 125, 137, 142, 153,
            152, 148, 139, 114, 139, 126, 116, 115,
            120, 119, 120, 129, 132, 136, 115, 127,
            112, 113, 125, 124, 99, 123, 117, 114,
            136, 142, 152, 151, 135, 122, 121, 117,
            98, 129, 114, 131, 120, 112, 125, 132,
            122, 128, 112, 115, 117, 117, 126, 143,
            144, 120, 120, 119, 131, 140, 138, 142,
            128, 125, 113, 126, 141, 133, 132, 135,
            129, 128, 132, 141, 139, 147, 129, 117,
            122, 116, 130, 140, 138, 129, 133, 125,
            129, 132, 102, 128, 116, 118, 117, 126
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.36322360203503,
        "Data" : [
            4, 4, 2, 4, 4, 1, 4, 4, 2, 3, 4, 3, 4, 4, 2,
            4, 4, 1, 3, 4, 2, 4, 4, 2, 3, 4, 1, 4, 4,
            1, 3, 4, 2, 4, 4, 2, 4, 4, 2, 3, 3, 1, 4,
            4, 1, 4, 3, 1, 4, 4, 1, 4, 4, 2, 3, 4, 2,
            3, 4, 0, 3, 4, 1, 3, 3, 1, 4, 4, 2, 4, 3,
            2, 4, 4, 2, 4, 4, 2, 3, 3, 1, 4, 4, 1, 3,
            4, 2, 3, 4, 2, 4, 4, 2, 5, 5, 2, 4, 4, 2,
            4, 4, 2, 4, 3, 1, 3, 4, 1, 4, 4, 2, 4, 4,
            1, 4, 4, 1, 4, 4, 0, 4, 4, 2, 4, 5, 1, 4,
            4, 1, 4, 3, 1, 3, 4, 1, 4, 4, 1, 4, 4, 3,
```

```
                4, 4, 2, 3, 4, 2, 3, 3, 2, 3, 3, 0, 3, 4,
                1, 3, 3, 1, 3, 4, 1, 4, 4, 2, 3, 3, 2, 4,
                4, 2, 4, 3, 1, 3, 3, 1, 3, 4, 1, 3, 4, 1,
                4, 3, 1, 3, 3, 1, 4, 4, 1, 4, 5, 3, 5, 4,
                2, 4, 4, 1, 4, 4, 2, 4, 4, 2, 4, 4, 1, 5,
                5, 1, 4, 4, 2, 4, 4, 2, 4, 3, 2, 3, 4, 2,
                3, 4, 2, 4, 3, 1, 4, 3, 2, 4, 3, 2, 4, 3,
                1, 3, 4, 1, 3, 3, 1, 4, 3, 0, 4, 3, 2, 3,
                3, 1, 4, 4, 1, 4, 4, 1, 4, 3, 0, 3, 4, 1,
                3, 4, 1, 3, 3, 1, 3, 3, 1, 3, 3, 0, 4, 4,
                2, 3, 4, 2, 3, 4, 1, 4, 4, 0, 4, 3, 0, 4,
                3, 2, 4, 4, 1, 4, 4, 2, 4, 4, 2, 3, 4, 3,
                4, 4, 1, 4, 4, 3, 4, 3, 2, 3, 3, 1, 4, 4,
                1, 4, 4, 1, 3, 4, 1, 3, 4, 0, 4, 4, 0, 4,
                3, 1, 3, 3, 0, 4, 3, 2, 4, 4, 1, 3, 3, 1,
                3, 3, 0, 3, 3, 2, 4, 3, 0, 4, 4, 1, 4, 4,
                2, 4, 4, 1, 3, 3, 1, 3, 4, 2, 4, 3, 2, 3,
                4, 1, 4, 3, 3, 4, 3, 2, 4, 3, 2, 3, 4, 2,
                4, 4, 2, 4, 4, 2, 4, 4, 1, 4, 3, 2, 3, 4,
                1, 4, 3, 1, 4, 3, 1, 3, 3, 1, 3, 4, 1, 3,
                4, 1, 4, 4, 1, 4, 3, 1, 4, 4, 1, 3, 4, 1,
                3, 3, 1, 4, 4, 1, 4, 4, 1, 3, 4, 2, 4, 3,
                1, 4, 4, 1, 3, 3, 2, 4, 4, 2, 4, 3, 1, 4,
                3, 0, 3, 3, 2, 3, 4, 2, 4, 3, 1, 4, 4, 1,
                4, 3, 1, 4, 3, 2
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00588309352488921
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 17.3540890272977,
        "Data" : [
            18, 1, 5, 17, 12, 8, 47, 35, 31, 27, 20, 23,
                32, 24, 22, 25, 26, 18, 37, 30, 21, 34,
                27, 24, 32, 27, 10, 18, 12, 17, 17, 10,
                19, 21, 12, 19, 22, 16, 20, 31, 32, 12,
                40, 25, 12, 23, 19, 14, 29, 19, 18, 21,
                13, 21, 25, 11, 15, 19, 17, 10, 31, 19,
                18, 23, 20, 10, 16, 5, 7, 19, 3, 7, 35,
                25, 25, 38, 15, 17, 34, 22, 5, 34, 24, 22,
                 17, 13, 24, 36, 28, 27, 39, 27, 28, 29,
                25, 22, 20, 10, 19, 18, 10, 22, 18, 17,
                16, 22, 24, 9, 30, 19, 20, 17, 15, 12, 20,
                 23, 5, 30, 16, 18, 18, 11, 21, 19, 11,
                20, 26, 10, 25, 25, 28, 5, 19, 24, 8, 26,
```

```
                15, 22, 44, 23, 22, 19, 11, 19, 23, 12,
                18, 19, 15, 16, 19, 22, 7, 21, 15, 9, 26,
                27, 14, 28, 25, 17, 25, 13, 15, 40, 21,
                14, 24, 15, 16, 16, 12, 15, 16, 11, 15,
                20, 13, 12, 14, 15, 5, 15, 14, 6, 15, 11,
                10, 26, 24, 9, 20, 15, 18, 21, 17, 14, 20,
                 22, 2, 16, 14, 18, 22, 19, 15, 20, 15,
                12, 20, 21, 6, 20, 16, 17, 26, 16, 14, 17,
                 13, 16, 18, 14, 14, 24, 14, 13, 16, 12,
                11, 19, 15, 16, 20, 12, 13, 17, 14, 9, 17,
                 11, 13, 18, 14, 12, 16, 19, 7, 16, 14,
                13, 18, 14, 14, 17, 14, 9, 15, 14, 8, 15,
                17, 4, 15, 11, 14, 15, 13, 8, 15, 17, 7,
                20, 17, 11, 20, 17, 6, 12, 6, 5, 16, 16,
                15, 17, 13, 11, 17, 17, 7, 24, 23, 13, 19,
                 14, 16, 18, 18, 10, 19, 19, 14, 17, 12,
                17, 16, 12, 15, 17, 12, 15, 17, 14, 13,
                10, 4, 7, 20, 17, 14, 18, 17, 2, 20, 19,
                14, 21, 11, 14, 16, 13, 10, 22, 22, 10,
                18, 17, 8, 15, 16, 5, 16, 12, 14, 15, 13,
                8, 15, 17, 7, 18, 17, 12, 17, 11, 16, 19,
                19, 2, 23, 15, 15, 19, 13, 16, 16, 15, 6,
                16, 17, 7, 15, 14, 16, 22, 15, 19, 19, 15,
                 14, 19, 14, 17, 20, 13, 14, 17, 12, 15,
                15, 13, 15, 9, 6, 18, 15, 16, 17, 29, 18,
                15, 17, 12, 15, 20, 13, 13, 19, 14, 13,
                17, 14, 8, 17, 14, 11, 18, 14, 12, 26, 15,
                 14, 19, 17, 8, 23, 18, 13, 22, 16, 10,
                18, 12, 13, 19, 16, 8, 21, 19, 7, 21, 16,
                17, 24, 19, 19, 22, 17, 15, 21, 22, 5, 23,
                 16, 14, 17, 13, 15, 21, 15, 13, 25, 24,
                7, 13, 7, 5, 18, 18, 16, 18, 15, 11, 20,
                19, 7, 20, 17, 10, 15, 15, 16
            ]
        }
}
```

Listing F.3: The JSON file output from testing the fully optimised game with the view frustum culling approach of comparing screen space coordinates of chunk vertices, on the desktop computer.

```
{
    "Test Comment" : "This test has all optimisations and
         screen coord chunk VFC. This usually has
        significantly better performance but this is due
        to too much culling, removing chunks which should
```

```
    be visible, which is unacceptable.",
"Date and Time" : "08/05/2023 21:31:20",
"Environment Type" : "Build",
"CPU" : {
    "Name" : "11th Gen Intel(R) Core(TM) i7-11800H @
        2.30GHz",
    "Cores" : 8,
    "Frequency (MHz)" : 2304
},
"RAM Total (MB)" : 16126,
"Graphics" : {
    "Graphics Device Name" : "NVIDIA GeForce RTX 3050
        Laptop GPU",
    "GPU VRAM (MB)" : 3979
},
"Test Duration (ms)" : 70920.2775000036,
"Settings" : {
    "Optimisations" : {
        "Object Pooling" : {
            "Pillars" : true,
            "Faces" : true
        },
        "Internal Face Culling" : true,
        "View Frustum Culling" : {
            "Screen Coordinates - Faces" : false,
            "Screen Coordinates - Chunks" : true,
            "Colliders - Chunks" : false
        }
    },
    "World Properties" : {
        "Chunk Dimensions (in voxels)" : {
            "X" : 16,
            "Y" : 16,
            "Z" : 16
        },
        "Render Distance (in voxels)" : {
            "X" : 112,
            "Y" : 48,
            "Z" : 112
        },
        "World Seed" : 1234567,
        "Hill Properties" : {
            "Max Height" : 43,
            "Min Height" : 30
        }
    },
```

```
        "Misc Options" : {
            "Generate Random Blocks" : false,
            "Randomise Seed" : false,
            "Use World Generation System" : true,
            "Remove Pillars Dynamically" : true,
            "Generate Pillars Dynamically" : true,
            "Use Hill Shape Curve" : true,
            "Generate Terrain only when Q pressed" :
                false,
            "Generate face objects only where needed" :
                true,
            "Narrow Caves at Surface" : true
        }
    },
    "FPS" : {
        "Average" : 20,
        "Data" : [
            4, 23, 22, 22, 21, 13, 17, 15, 19, 18, 17,
                19, 17, 18, 17, 22, 21, 18, 3, 16, 18, 3,
                18, 16, 21, 19, 31, 30, 32, 3, 18, 22, 3,
                18, 18, 18, 17, 20, 18, 17, 19, 5, 15, 3,
                22, 20, 25, 17, 16, 34, 37, 33, 32, 24,
                23, 37, 36, 35, 34, 30, 33, 20, 27, 21,
                19, 19, 34, 37, 3, 22, 23, 24, 26, 3, 20,
                23, 25, 22, 21, 25, 21, 22, 3, 23, 26, 30,
                 28, 28, 29, 29
        ]
    },
    "Pillar Generation and Render Times (ms)" : {
        "Average" : 160.014987050522,
        "Data" : [
            191, 199, 278, 232, 250, 262, 278, 233, 231,
                203, 209, 239, 239, 253, 264, 223, 209,
                230, 210, 219, 237, 209, 267, 182, 235,
                240, 243, 247, 268, 266, 255, 228, 226,
                245, 267, 235, 215, 202, 209, 230, 230,
                229, 224, 214, 237, 252, 323, 224, 225,
                167, 134, 128, 136, 140, 131, 137, 134,
                120, 116, 121, 106, 108, 110, 125, 122,
                121, 114, 113, 120, 106, 124, 114, 115,
                119, 118, 119, 113, 124, 124, 98, 131,
                118, 120, 129, 117, 109, 131, 114, 108,
                113, 119, 125, 116, 123, 116, 115, 116,
                110, 110, 107, 135, 119, 106, 98, 101,
                122, 116, 122, 114, 113, 120, 117, 126,
                146, 141, 120, 122, 123, 122, 171, 121,
```

```
            101, 107, 105, 110, 110, 113, 105, 114,
            117, 102, 104, 119, 95, 121, 109, 111,
            101, 114
        ]
    },
    "Chunk Block List Generation Times (ms)" : {
        "Average" : 3.3988095925366,
        "Data" : [
            4, 5, 2, 4, 3, 2, 2, 4, 2, 3, 4, 3, 3, 4, 2,
            4, 4, 2, 4, 4, 1, 3, 3, 2, 4, 4, 1, 4, 4,
            1, 4, 3, 2, 4, 4, 1, 3, 4, 1, 4, 4, 1, 3,
            4, 1, 4, 5, 0, 4, 3, 1, 3, 4, 3, 4, 4, 1,
            3, 3, 2, 3, 4, 2, 3, 4, 1, 4, 3, 3, 4, 4,
            2, 4, 4, 2, 4, 3, 2, 3, 3, 1, 3, 4, 1, 4,
            5, 2, 4, 4, 3, 3, 4, 2, 3, 4, 2, 3, 4, 2,
            4, 4, 2, 4, 4, 1, 3, 4, 1, 4, 4, 2, 3, 4,
            2, 4, 3, 1, 4, 4, 2, 3, 4, 3, 3, 4, 2, 4,
            3, 2, 3, 4, 0, 4, 4, 2, 4, 3, 2, 3, 4, 2,
            4, 4, 2, 4, 3, 2, 7, 6, 4, 4, 4, 2, 4, 4,
            2, 5, 3, 0, 3, 3, 1, 3, 4, 1, 3, 4, 2, 4,
            4, 2, 3, 3, 2, 4, 3, 2, 3, 4, 0, 3, 4, 1,
            3, 3, 2, 4, 4, 1, 4, 3, 2, 3, 4, 2, 3, 3,
            2, 3, 2, 1, 3, 4, 0, 4, 3, 2, 3, 3, 2, 3,
            4, 2, 3, 4, 1, 4, 2, 1, 3, 3, 1, 4, 4, 0,
            4, 3, 1, 4, 3, 1, 4, 4, 1, 3, 4, 0, 3, 3,
            1, 4, 3, 3, 3, 4, 1, 4, 4, 0, 3, 4, 2, 5,
            4, 2, 3, 4, 1, 4, 4, 2, 3, 3, 1, 3, 3, 2,
            4, 3, 2, 5, 4, 3, 4, 4, 2, 4, 3, 1, 3, 4,
            1, 4, 3, 2, 3, 3, 2, 3, 3, 2, 2, 3, 0, 3,
            4, 2, 3, 3, 0, 4, 3, 2, 3, 4, 2, 4, 3, 0,
            4, 3, 1, 3, 3, 2, 4, 3, 1, 3, 3, 3, 4, 3,
            1, 4, 4, 2, 4, 4, 1, 4, 3, 3, 4, 3, 1, 4,
            3, 1, 3, 5, 2, 4, 4, 3, 4, 4, 2, 4, 4, 0,
            4, 3, 2, 3, 4, 0, 4, 4, 0, 3, 3, 1, 3, 3,
            1, 2, 3, 1, 2, 4, 2, 4, 4, 1, 3, 2, 1, 3,
            4, 2, 3, 3, 3, 4, 3, 2, 3, 3, 2, 2, 3, 0,
            4, 4, 0, 4, 4, 0, 4, 3, 2, 3, 4, 2, 4, 4,
            1, 4, 3, 1, 4, 4, 0, 4, 3, 1
        ]
    },
    "Voxel Rendering Time (ms)" : {
        "Average" : 0.00556917193337758
    },
    "Chunk Rendering Times (ms)" : {
        "Average" : 16.340770263609,
        "Data" : [
```

```
            17, 2, 4, 16, 11, 7, 47, 31, 28, 24, 18, 21,
                27, 20, 22, 24, 24, 14, 36, 32, 18, 32,
                23, 22, 28, 25, 9, 16, 10, 15, 16, 8, 19,
                15, 9, 14, 17, 13, 18, 24, 29, 11, 35, 25,
                 12, 18, 13, 13, 22, 11, 17, 26, 11, 19,
                21, 10, 13, 18, 15, 8, 27, 16, 18, 24, 17,
                 10, 28, 5, 8, 13, 4, 5, 31, 23, 22, 34,
                13, 16, 29, 21, 4, 31, 21, 23, 17, 15, 24,
                 33, 23, 24, 29, 21, 20, 21, 19, 17, 18,
                9, 17, 17, 9, 21, 16, 38, 15, 21, 22, 7,
                23, 16, 18, 16, 12, 10, 19, 19, 5, 27, 13,
                 17, 15, 9, 20, 16, 9, 18, 24, 9, 23, 22,
                25, 4, 20, 22, 9, 20, 14, 20, 33, 17, 16,
                15, 8, 18, 20, 11, 15, 20, 17, 20, 20, 18,
                 8, 21, 16, 10, 24, 26, 9, 26, 18, 14, 23,
                 13, 13, 30, 14, 14, 19, 15, 18, 15, 11,
                16, 16, 10, 15, 20, 13, 11, 14, 14, 5, 14,
                 12, 9, 14, 11, 11, 21, 16, 12, 15, 11,
                13, 17, 13, 7, 15, 14, 7, 19, 15, 5, 16,
                10, 13, 14, 12, 8, 22, 15, 14, 17, 13, 9,
                18, 10, 13, 17, 14, 12, 16, 19, 8, 17, 15,
                 14, 15, 19, 6, 21, 17, 12, 21, 18, 6, 12,
                 7, 5, 17, 19, 17, 18, 14, 12, 20, 21, 8,
                25, 22, 13, 20, 16, 12, 17, 19, 3, 22, 20,
                 15, 21, 11, 16, 16, 13, 11, 16, 19, 10,
                22, 10, 18, 18, 13, 13, 17, 11, 13, 25,
                13, 12, 14, 10, 12, 16, 11, 11, 16, 12,
                12, 14, 11, 8, 19, 15, 12, 16, 15, 3, 24,
                21, 15, 21, 11, 14, 15, 11, 9, 13, 14, 6,
                15, 9, 12, 18, 12, 14, 17, 11, 13, 23, 13,
                 14, 14, 10, 13, 15, 11, 11, 17, 12, 12,
                15, 12, 7, 26, 25, 3, 29, 26, 19, 37, 13,
                19, 22, 16, 11, 21, 20, 9, 22, 11, 18, 21,
                 20, 2, 15, 62, 15, 18, 11, 13, 13, 13, 6,
                 16, 16, 7, 13, 12, 14, 19, 12, 15, 18,
                13, 11, 20, 12, 11, 13, 11, 14, 21, 12,
                15, 24, 13, 15, 16, 11, 10, 17, 20, 3, 21,
                 15, 11, 17, 4, 1, 18, 15, 14, 15, 12, 13,
                 14, 10, 14, 14, 15, 4, 19, 15, 11
        ]
    }
}
```


Listing F.4: The JSON file output from testing the fully optimised game with the view frustum culling approach of comparing screen space coordinates of chunk vertices, on the laptop computer.