Historic Urban Modelling and Automatic Road Extraction From Hand Drawn Maps

Johnson Yick

A thesis submitted for the degree of
Master of Science
at the University of East Anglia
April 2010

Historic Urban Modelling and Automatic Road Extraction From Hand Drawn Maps

Johnson Yick

Abstract

This thesis is comprised of two parts, covering the modelling of the Cunningham map and Brighton map in three-dimensions using Autodesk 3ds Max and the development of a novel road extraction algorithm.

Extraction of roads from images is of major interest in image analysis. Road extraction is possible because they have unique features that make them stand out in an image. Once extracted, the road data can be represented in many forms, such as in a Global Positioning Satellite (GPS) system for calculating lengths of roads.

The Cunningham map was drawn by William Cunningham in 1558; it shows Norwich from the east side. The Brighton map was created in 1778, the map is colour coded and unlike the Cunningham map it is drawn from a top down perspective. While creating these virtual models, it was found that much of the modelling was repetitive and predictable, for example the creation of buildings and roads. This pattern allowed the development of a novel road extraction algorithm. By passing a circle through the road network and recording its radius and position, it is capable of finding the skeleton of the road network and the width of the roads within the network. An advantage of this algorithm over others is that it only searches for major roads and can ignore small errors within an image such as noise, without slowing down the algorithm drastically.

From the testing completed on both synthetic and real-world cases, it is shown that the proposed algorithm can extract a skeleton from an image. This skeleton can also be imported into Autodesk 3ds Max for the re-creation of the road network in three-dimensions.

Acknowledgements

I would like to thank the following people who helped in making this study a success.

Prof. Andy Day (CMP, UEA) and Dr. Robert Laycock (CMP, UEA) for their advise and expertise throughout the study.

Mr. David Drinkwater (UMG, UEA) for the three-dimensional models of the cathedral, St Andrews, cross building and the castle.

Prof. Carole Rawcliffe (HIS, UEA) for her expert historical input for this study.

My wife Hengdan Chen for her support and motivation throughout this study, who without this would not have been possible.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Background

In the past, many maps were drawn by hand. These maps hold vital information for cultural heritage sites, such as locations of specific buildings or road layouts. This information in turn can explain how the people lived within those areas.

The Cunningham map (Figure 4.1) was created by William Cunningham in 1558; this map shows how Norwich appeared from his perspective, on top of a hill viewing from the east.

The Brighton map (Figure 4.3) is a more recent map created by Thomas Budgen in 1788, this map depicts the layout of Brighton at the time. This map differs from the Cunningham map in that it is drawn in colour and is a top-down view showing the building footprints.

Building three-dimensional models on computers was pioneered by General Motors since they created the first modelling package in 1959, called DAC-1 [Kru94]. They invited IBM to become a partner as at the time IBM was the leading computer developer and the DAC-1 ran faster and had more flexibility on the IBM 7090. The DAC-1 was limited to what it could accomplish because of the hardware it ran on. For example, the IBM 7090 had 50,000 transistors [IBM10] while modern day central processing units(CPU) have hundreds of millions of transistors. In addition, more

than a single CPU can coexist within a single computer. For instance, servers using Intel Xeon processors can bring the transistor count to well over a billion.

1.2 Motivations and Research Objectives

Substantial research has been completed in developing techniques to extract buildings and roads from two-dimensional images. There are two main sets of road extraction algorithms, "Road tracing", for instance, G. Vosselman and J. Knecht [VK95] and "Dynamic programming", for example, A. P. CAl Poz and G. M. DO Vale [PV03].

This thesis proposes a new type of algorithm of the "Road tracing" type. The proposed algorithm extracts road geometry from images or hand-drawn maps. Unlike previous techniques, this algorithm is based on the principle that if there is a two-dimensional tunnel and a circle is placed inside, if that circle touches two unique edges then the centre of the circle is at the mid-point between the two opposite tunnel walls.

The proposed algorithm has two main advantages over current road extraction algorithms. Firstly it can record width at every point within each section of road accurately. Secondly it is able to ignore sections of the map that do not fit the specification for a road. For instance, when all road paths are greater than 40 pixels, but a path is found with just 2 pixels, in this case the 2 pixel wide road would be ignored.

1.3 Problem Definition in the Cunningham and Brighton Maps

Modelling within Autodesk 3D studio Max (3ds Max) is an intuitive way to create a virtual environment. Using the tools available within the program it is possible to create any models, this is why it was used for the modelling of both the Cunningham and the Brighton maps.

The Cunningham map model is a joint collaboration between the Urban Modelling Group (UMG) and Professor Carole Rawcliffe (History Department, University of East Anglia). There are many problems involved when developing a virtual model with limited information, for example, how the buildings appeared on the reverse of what W. Cunningham drew on the map (Figure 4.1), as there is only one map from his one perspective.

The aims of the Cunningham map model, is to show audiences how the city appeared in 1558 from W. Cunningham's perspective. Using the specifications given by Professor C. Rawcliffe, a single video should be rendered fly over the city with a pan around the St Andrews hall. As well as the video the 3ds Max file containing the entire scene needs to be provided, so that if there are any future improvements or implementations it can be done without remaking the entire model.

The Brighton map model, although based on a more modern map with more defined footprints for the buildings and roads, has its own set of unique problems. These problems include how the buildings are built within the map - none of them are uniform or similar to each other in any way - this means that each building has to be modelled bespoke for each footprint. A more fundamental problem is that within the map there is no way to find the heights of any of the buildings on the map.

The aim for the Brighton map is similar to the Cunningham map in that a video is to be provided but only a pan around the virtual model is needed. The map is also be used within the road extraction algorithm, so that manually modelled roads can be tested against the algorithms results.

Within both maps it is clear that modelling the buildings will be easier than modelling the roads, because, unlike the buildings, the roads appear to be everything that is not either a building or grassland; this is especially true for the Cunningham map model where it is unclear where some roads start and end because of buildings

obstructing the view. Within the Brighton map it is not clear which types of roads are used, because obviously larger roads are used to accommodate vehicles while smaller alleys cannot, both of these roads would appear very different.

1.4 Problem Definition in the Road Extraction Algorithm

There are many ways to model real-life objects in virtual environments. For example, the simplest way, is to use a three-dimensional modelling package, such as Autodesk 3D studio Max (3ds Max) which supports the creation of any virtual object. Procedural modelling is a method for creating a virtual model, with little or no interaction with the operator. The advantage of procedural modelling is that it speeds up the process of creating large complex scenes, such as cites. Using procedural modelling algorithms, such as that developed by R. G. Laycock et al. [LD03a], it would be possible to model a large city many times faster than using a traditional three-dimensional modelling package.

Being able to extract accurate road geometry from two-dimensional sources, such as scanned images, photographs and diagrams quickly, has been the goal of many researchers such as S. O. Elberink and G. Vozzelman [EV07]. Having the data represented as a set of Cartesian coordinates allows the road network to be separated from the image. The data can then be imported into a modelling package, such as 3ds Max, where the data can be used to create a three-dimensional road network within a virtual environment.

Current algorithms for road tracing are based on a two step principle, segmentation of the road from a map and then thinning of the road. Thinning algorithms developed so far can be classified into two main types, sequential and parallel algorithms. All thinning algorithms work by removing all non-critical pixels. A critical pixel is one

that lies on the medial axis of the road network.

Parallel algorithms calculate the position of all non-critical pixels and then remove them all at once. Sequential algorithms calculate the boundary pixels while removing non-critical pixels. Although research has gone into removing non-critical pixels in groups to speed up the process [SES95], removing non-critical pixels individually remains a computationally expensive process, because the algorithm has to be recursive. This thesis proposes a new algorithm which instead of using a thinning algorithm to obtain the skeleton, passes a dynamic circle through the road network to find the skeleton of the road network.

Chapter 2

Cultural Heritage and Virtual Reconstruction

2.1 Abstract

The use of Virtual Reality by museums such as the Louvre in France allows prospective visitors and scholars to view their exhibits without actually going there. The Louvre's website attracted an audience of nearly six million visitors between 2002 and 2004. This is nearly the number of visitors the actual museum attracted during the same period [Lou10]. With the development of modern technology and increasing number of people with high speed internet, it can only be assumed that these numbers are now substantially higher than they were back then.

With Virtual Reality becoming ever more popular, with uses ranging from museums like the Louvre to video games, it is apparent that the immersion in a virtual world is the future. Cultural heritage sites will only last as long as we protect them, but some sites like ancient Rome are already gone. Using existing modelling techniques, it is possible to recreate such sites as virtual models. These models can be used to show audiences what the cultural heritage sites looked like, but also to test existing and new theories about them. For example, B. Frischer [Fri10] describes the possibility of using a virtual model to simulate visitors entering the "Trajan Forum" in ancient

Rome.

2.2 Applications for virtual heritage sites

2.2.1 Dudley Castle

Being at the forefront of technology, Colin Johnson [Joh10a] developed a virtual tour system that allowed visitors to interact with a three-dimensional reconstruction of Dudley Castle as it was in the year 1550. This system was launched in 1994, at the visitor's centre of Dudley Castle in England, with the first user being H.M. the Queen of England [Joh10b].

2.2.2 Rome Reborn

Rome Reborn is run primarily by the University of Virginia's Institute for Advanced Technology in the Humanities (IATH) [Vir10b]. Since 1997, IATH has been working towards the creation of a high quality digital model of Rome as it appeared in 320 A.D, which is close to the climax in the development of the city, in terms of its population [GFS+05]. Their secondary goal is to create the cyber-infrastructure needed for updating, correcting and augmenting the model. This involves knowledge about how the city was reconstructed digitally, such as bridges, walls and streets. Sources of archaeological information or speculative reasoning are made available to the public when necessary. Beyond its primary functions, the model can be used for teaching the general public or students about how ancient Rome looked.

In the latest iteration, version 2.0 [Vir10c], Procedural and Mental Images were invited to join the Rome Reborn project. Rome Reborn 2.0 further develops the quality of the virtual model, by using procedural techniques to create features, such as using geometry to model windows and doors. Previous versions used textures to show position of windows and doors. Using Mental Image's RealityServer platform

Figure 2.1: Shows a render from the RealityServer, of the Rome Reborn 2.0 model (Image provided by IATH).

[Ima10c], it is possible for users to view the model on the internet, unlike previous versions, where the model needed to be on their workstation. Although demonstrated [Vir10d], it has not been implemented for general public use, Figure 2.1 shows the quality of the images produced from RealityServer's solution.

With the collaboration of Google [Vir10a] in 2008, the model was imported into the Google Earth platform, as shown in Figure 2.2. Using Google Earth allows anyone with access to the Google Earth application to view the Rome Reborn model in three dimensions. The buildings within the city are of variable quality, from high quality models for larger models like the Colosseum, to low polygon models for the houses. The buildings have been textured on the inside as well as the outside; this goes to show the quality of the model and the time taken to create it.

2.2.3 Treasures of King Tutankhamum

Heritage Key [Key10a] allows users to view cultural heritage sites, such as the treasures of King Tutankhamun. This site is built by modelling each item using a manual

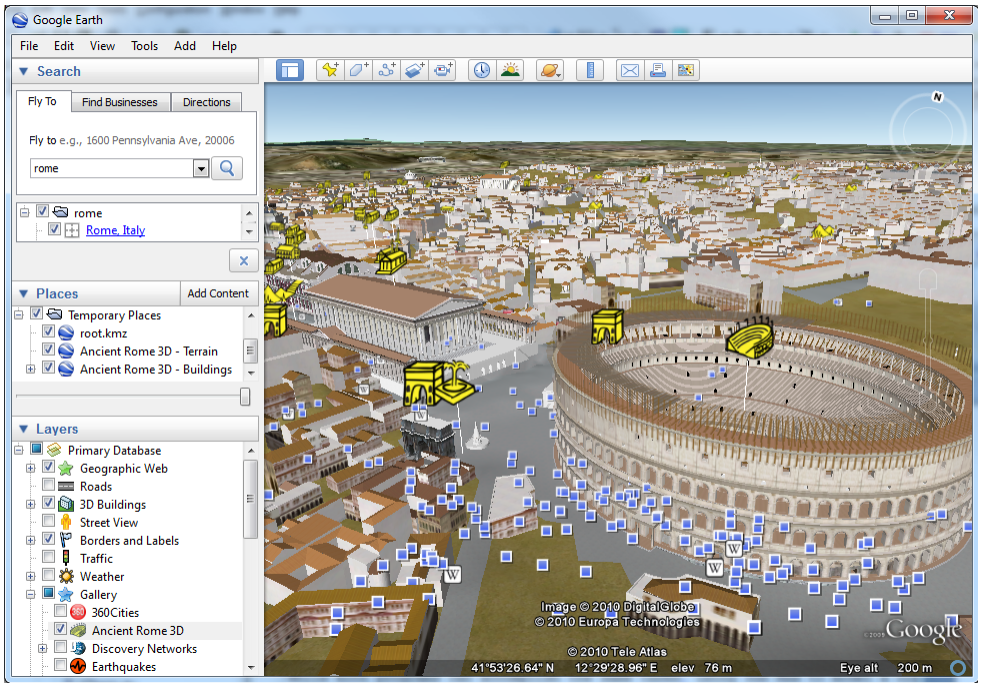


Figure 2.2: Illustrates how the Rome Reborn model can be viewed using the Google Earth application. The view is of the Colosseum.

modelling package [Key10b], then using an open source Application Program Interface (API) called OpenSimulator [Ope10] to create the virtual environment, so that it can be accessed by anyone. Using OpenSimulator is not as user-friendly as other solutions, such as Google Earth, because before the user can view the cultural heritage sites, they need to download specialist software to view it; although users have to download Google Earth, Google's platform is widely used and well known unlike OpenSimulator.

2.3 Manual Modelling

2.3.1 Manual modelling packages

Modelling packages have come a long way since General Motors invented DAC-1. Current modelling packages such as Autodesk 3D Studio Max 2010 (3ds Max) [Aut10a] and Autodesk Maya 2010 (Maya) [Aut10b] contain features that make modelling fast and easy, such as intuitive design for modelling. Although created by the same

company, 3ds Max and Maya are designed for two different types of modelling.

3ds Max is an excellent program for creating virtual environments such as cities, because of the outstanding features it offers to the graphics designer . For instance, the material library contains a special set of materials called "architectural and design". These materials are programmed shaders that allow for accurate representation of surfaces such as copper or masonry, when combined with the Mental Ray rendering engine. Using these materials means it is easier to represent a more realistic scene, because the materials react correctly to light. 3ds Max offers features for speeding up the creation of virtual environments. For example, the "Day light" system can accurately calculate the colour and intensity of sunlight when given a date and location.

Although Maya has similarities to 3ds Max, it offers more features geared towards the creation of dynamic models, such as characters and non-uniform environments like a jungle. Special effect groups use Maya instead of 3ds Max when working on movies such as Avatar [Aut10c] and Monsters Vs. Aliens [Ani10]. One reason why special effect artists working on films prefer Maya to 3ds Max is because of the animation suite that comes with Maya.

While both programs can accomplish the same tasks, using 3ds Max to model a city would be easier than using Maya but, to model an animal, Maya would be better suited. An open source modelling package called Blender [ble10] was developed as an alternative to what Autodesk has to offer. Although Blender is not as popular as Autodesk's offerings, it is still a viable alternative to professionals. For example, ProMotion's studio created a series of short animations called Kajimba using it [Stu10].

The reason for creating virtual models of an environment instead of building physical models is the greater flexibility offered by virtual models. For example, if

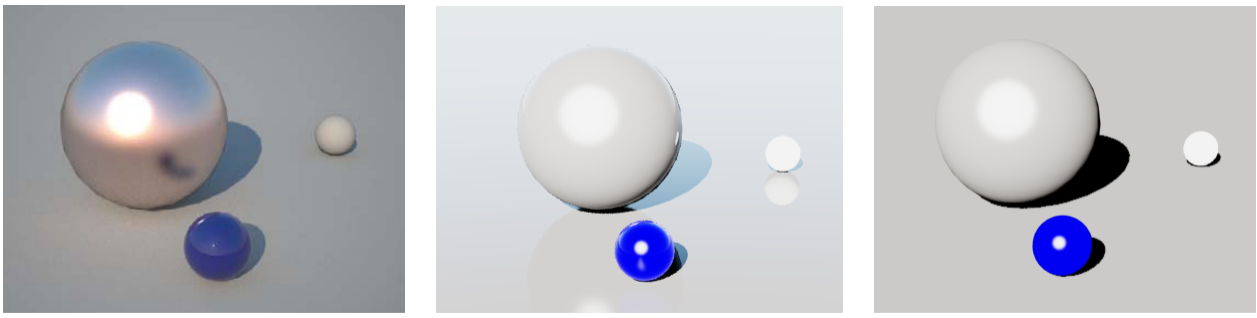


Figure 2.3: a) Illustration of a scene rendered using Maxwell. (b) The same scene rendered using Mental ray. (c) The same scene rendered using Scanline.

a model of a city needs to be passed to another person to work on: if the model is virtual it can be sent by email, this would not be possible for a scaled physical replica model of the city.

Once a virtual model has been created it is possible for the model to be imported into another modelling package. This is possible because mainstream modelling packages, such as 3ds Max and Maya, can save the geometry of a scene in the ".3ds" format which can be read by all modelling packages.

2.3.2 Rendering engines

A rendering engine can be defined as a system used to calculate the effects of light within the scene. This includes lighting effects such as reflections, refractions and caustics. There are many types of rendering engines in existence. For example, Mental Ray [Ima10a], Vray [Sof10], Maxwell [S.L10] and Brazil [Spl10]. All of these rendering engines can be grouped into three different types: unbiased, biased and hybrid.

Unbiased rendering engines initially start off with more noise within the image but generally produce more realistic renders. For example the rendering equation

developed by J.T. Kajiya [Kaj86]. Unbiased algorithms work produce better results the more iterations of the algorithm can complete this is because more light rays are traced back to the camera, leading to more of the initial noise is removed. This means that the longer a single image is left to render using an unbiased algorithm, the clearer the resulting image. Having to wait for all the pixels in the image to be filled in by random ray tracing means that the rendering of a single image using unbiased rendering engines such as Maxwell can take up to 40 minutes for a small image. This is especially true if we set up a scene where a light source and camera are separated by an opaque screen with only a single hole wide enough for one photon to pass though. Such a scene would take a very long time to render using an unbiased rendering engine.

Biased rendering engines represent light by using approximation algorithms, Using approximations of light can produce artefacts when unexpected situations occur within the scene. For example, where both edges of the wall overlap, light can leak out. Biased rendering engines can render images at a much faster speed than unbiased rendering engines, because they do not try to calculate the exact paths that light beams would take. Using a biased rendering engine, such as the Default Scanline or standard Ray Tracing rendering provided by 3ds Max, allows the same scene to be rendered in 7 seconds (Scanline) compared to 40 minutes on an unbiased rendering engine. Although faster, the results do not look as impressive when compared with other renders 2.3(b).

Hybrid rendering engines like Mental Ray, take the best from both biased and unbiased designs. They incorporate unbiased algorithms such as the one proposed by J.T. Kajiya [Kaj86]and to speed up the process, they use other algorithms such as final gather (FG) or global illumination (GI). Using techniques like these can decrease the rendering time of a scene, while keeping the lighting effects realistic.

The main problem with using a hybrid engine is that they must be set up correctly to be effective. If set up incorrectly, they can take longer or the rendering engine can even crash in extreme cases. When using the correct settings, it is possible to render the same test scene in 7 minutes for the first frame, and then subsequent frames are rendered much faster at 10 seconds per frame. The long render time for the first frame is due to the calculation of the FG and GI maps which can be re-used in subsequent frames. Mental Ray produces high quality renders while keeping the render times reasonable, as shown in Figure 2.3(b).

Chapter 3

Procedural Modelling techniques

3.1 Introduction

A procedural modelling algorithm is a form of automatic modelling using a set of predefined rules. Many procedural modelling packages are now available, for example CityEngine and CityScape both come with comprehensive tools for creating urban environments. Using these programs it is possible to drastically cut down the amount of manual modelling that needs to be done. For instance, Figure 3.1 shows an entire city created in CityEngine, consisting of over 24,000 buildings; creating such a scene manually would have been a colossal task taking weeks, instead of minutes.

There are limitations to what a procedural modelling algorithm can do, because each algorithm is designed to handle a single aspect of procedural modelling, so cases that fall outside the scope of these algorithms will fail. A good example would be key buildings that define a city, such as the London Eye and Big Ben, in London, these buildings would have to be created using manual modelling techniques, so that they can be placed within the city generated by procedural modelling.

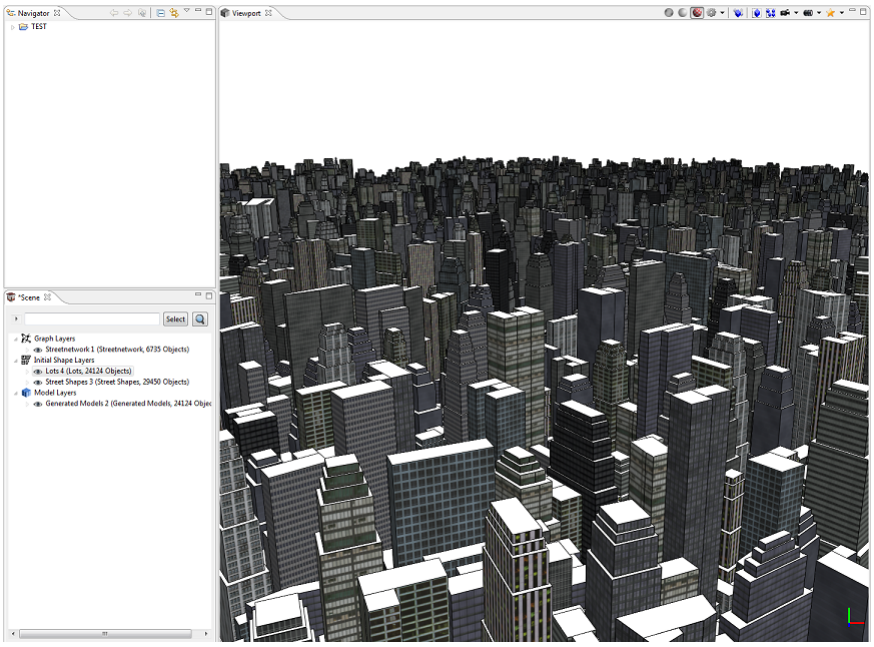


Figure 3.1: Shows a screen dump from CityEngine. The model consists of 24,124 buildings and an extensive road network.

3.2 Automatic / Semi-Automatic Modelling

3.2.1 Procedural Generation of Cities

Using aerial photography as the only source of input can lead to problems for procedural algorithms. For example, the heights of individual buildings are unknown. Research conducted by T. K. Javzandulam et al. [JL07] shows how the heights of a building can be extracted from their shadows when using an aerial photograph. If the geometry of the roof is unknown, then it would be modelled with a flat roof, but this is not true for the majority of buildings. Techniques researched by R. G. Laycock and A. M. Day [LD03a] use a straight skeleton of the building's footprint to recreate the roof, this allows unknown roof geometry to be calculated.

Having more information about the site to be modelled means fewer unknown variables have to be estimated or calculated from the image. Using a combination of a map based on the footprints of the buildings in Koblenz and a height map for the

buildings, R. G. Laycock and A. M. Day [LD03b] developed a technique to extract the buildings in three-dimensions. Firstly, each building footprint on the map was drawn with a single dot in the middle to represent a building, using these as seeds the areas where flood filled in and their two-dimensional polygons extracted. Secondly using the heights from the height map, the two-dimensional polygons were extruded into three-dimensional polygons representing the buildings. In the final stage, a generic set of textures was used to texture each side of the building. The result is a complete set of buildings that can accurately represent the buildings on the Koblenz map.

Although much research has been conducted into procedural modelling, there is still a large difference between models created using procedural modelling algorithms and models created using manual modelling packages such as 3ds Max. The main differences are caused either by failure of the algorithm, or insufficient quality of data supplied to the algorithm. For instance, when modelling from an aerial photograph, some features such as tall buildings or bridges can obstruct the view, leading to gaps in the map that can be problematic. One example, is the resultant breaks in the connectivity of a road network when a tall building obscures the view from where the image was taken. Another problem with the procedural generation of cities is that textures and models are reused many times over, giving the scene an artificial appearance, because all objects seem too uniform, as can be seen in Figure 3.1.

3.2.2 Procedural.inc

Founded by P. Muller et al., Procedural.inc [Pro10a] developed CityEngine which is capable of generating virtual cities procedurally. Although creating virtual environments can be accomplished in more detail when using manual modelling packages such as 3ds Max, it requires more time to create the scene. CityEngine is already used by major games developers, like Blizard and Square Enix for the creation of their

virtual environments. Although CityEngine can only take three types of input data [Geographical Information System (GIS), AutoCad (".dfx" export format) and OpenStreetMap data [Coa10]] for procedurally creating roads, it can export to many types of modelling packages, such as 3ds Max or Maya. CityEngine is an excellent tool for speeding up the development process of modelling cities, but with a high cost (entry price of $3450 [Pro10b]) and with the ability to only model urban environments, it is clearly targeted at professionals needing to create large virtual cities. To expand up on the work done by pioneers like P. Muller, others developed algorithms for extracting road data from other sources such as satalite photography, this is discussed within the next Chapter.

3.3 Procedural Road Junction Extraction

There are two main approaches to road extraction: road tracing algorithms and dynamic algorithms. Both will be explained in subsequent chapters.

3.3.1 Road Tracing Algorithm

G. Vosselman and J. Knecht [VK95] propose a typical road tracing algorithm. The algorithm requires an initial section of the road network to be defined by the operator. Once this section has been defined, the algorithm uses a single observation Kalman filter to create the initial state of the road profile. From the state profile, it is possible to predict the next section of road. Applying the Kalman filter recursively to update the state of the road profile, the entire road network can be found. G. Vosselman and J. Knecht made some assumptions about roads, which are more suited to aerial photography, such assumptions [VK95] are incorrect when working with some maps and hand drawn diagrams, for example:

- Roads are elongated (geometry)
- Roads have a maximum curvature (geometry)

When using G. Vosselman and J. Knecht's algorithm for road extraction, problems can occur. For example, where the curvature of the road is greater than the predicted value from the road profile, this would cause breaks in the connectivity of the road network. J. Zhou et al. [ZBC05] developed their work further by using a particle filtering system, which estimates the current profile and past profile states, this is used to predict the next position to move towards. J. Zhou et al.'s algorithm requires that an operator be present while the algorithm is running, this is because human interaction is needed to select the correct particle filter when the current one fails. Having the operator present while the algorithm is running would increase the accuracy of the algorithm, but this makes the algorithm semi-automated, rather than fully automatic like the original algorithm of G. Vosselman and J. Knecht.

The outcome of procedural modelling varies between algorithms and quality of input data. Having a cleaner source of input data can increase the quality of the outcome. For instance, using a flat bed scanner instead of a digital camera to take a picture of a map would increase the quality of the input.

C. Zhang et al. [ZMB99] passed a grey scaled aerial photo through a histogram to determine the approximate intensity values of the road. Once these ranges of values are found, the roads could go through segmentation in order to be extracted. Although segmentation extracts the roads from an image, it also extracts noise with it, because anything with a colour intensity similar to the road is also extracted. Every extracted object is then tested using a minimum ellipse along their longest axis (see Figure 3.2), in this way it is possible to remove the smaller features within the map. As roads are elongated, they would be preserved but smaller features such as houses would be ignored. If the road network has gaps, C. Zhang et al. also created a method to connect such roads to preserve the connectivity of the road network. After connecting all parts of the network, the image is passed through a thinning algorithm

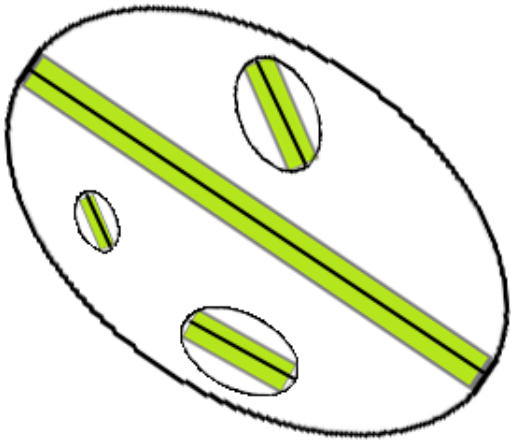


Figure 3.2: Illustrates a set of objects with their minimum bounding ellipses along the long axes.

to produce a skeleton of the road network.

3.3.2 Road Extraction using a Dynamic Programming

A dynamic programming approach like the one proposed by A. Gruen and H. Li [GL97] uses a set of control points to define the skeleton, as shown in Figure 3.3 (a). As only some of the control points are within the road network, each control point scans perpendicularly to the vector of the line in a one-dimensional grid, to check where the road is in relation to the centre of the road 3.3 (b). When a control point is found to be outside the road, it is interpolated and moved within the threshold boundaries, so it becomes closer to the centre of the road 3.3 (c). The algorithm then iteratively repeats until all nodes are within threshold boundaries.

3.4 Mathematical morphology

These techniques are designed to change the geometry of the objects within the image as explained in this book by J. Serra [Ser83], the sub-chapters of this thesis will discuss ones relevant to road extraction.

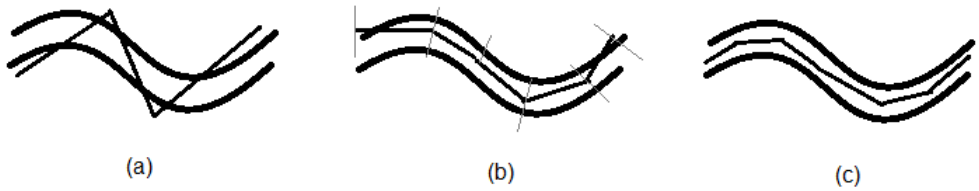


Figure 3.3: Diagram showing A. Gruen and H. Li's approach using dynamic programming.

3.4.1 Binary Dilation

This is a mathematical morphology technique which is designed to grow objects within a binary image. A structured object is chosen such as a square or circle to grow the object, then a point is searched for on the boundary of the object. The centre of the structured object is moved to the boundary pixel, where it traverses around the boundary edge until it reaches back to the beginning. When the structured object stops moving, all the marked pixels become part of the object, as shown in Figure 3.4.

3.4.2 Binary Erosion

This can be seen as an opposite operation to the Binary dilation in that instead of adding the pixels that have been marked it removes them. This is shown in Figure 3.4. Using a combination of dilation followed by erosion can leave the image the same size, but smooth out any rough edges. For example Figure 3.5.

3.4.3 Skeletonization

There are two main types of algorithms that can produce a skeleton from an image, they are called sequential and parallel skeletonization. A skeleton is the complete set of critical pixels; a critical pixel is a pixel that is located either within the centre of

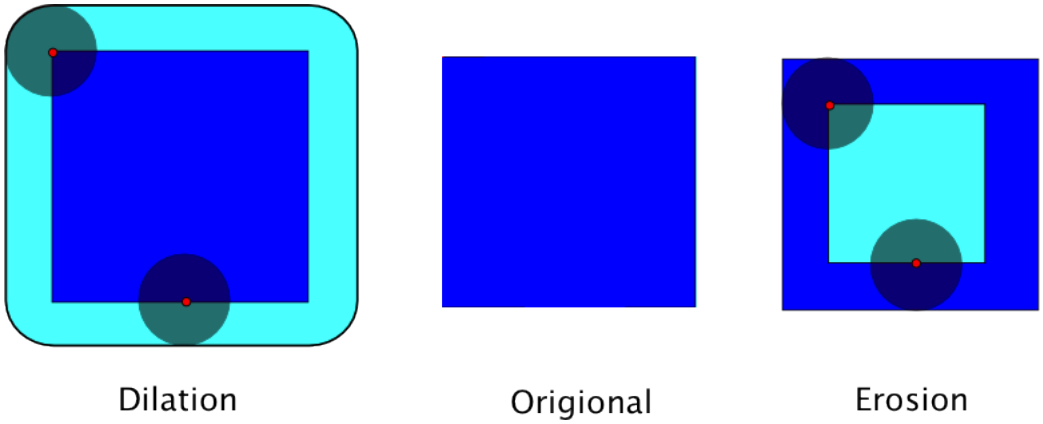


Figure 3.4: Diagram showing how erosion and dilation affects the shape of the origional shape, using a circle as the structured object.

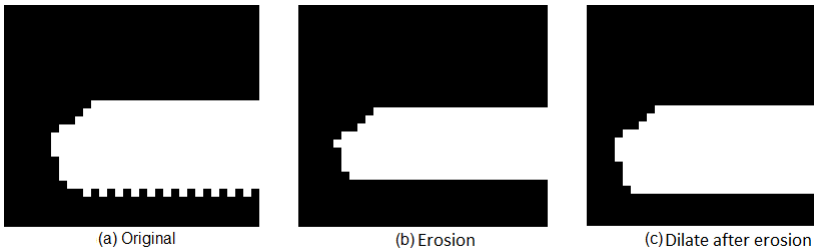


Figure 3.5: Figure showing how an image can be dilated then eroded to remove noise.

the object or to preserve connectivity. Sequential algorithms remove one pixel at a time, while calculating non-critical pixels within the image. Parallel algorithms work in two stages, firstly by recording all non-critical pixels then in step two removing all non-critical pixels.

Sequential algorithms are easier to implement as the most basic uses a 3x3 matrix, which moves around the boundary of the object removing all pixels under the matrix. While the boundary layer is removed recursively, if there exists only critical pixels in a single iteration then the algorithm terminates.

Examples of parallel algorithms are those due to R. W. Hall [Hal89] and C. M. Holt et al. [HSCP87]. Although C. M. Holt et al. developed an improved parallel thinning algorithm that manages to cut down the number of iterations the algorithm must complete, there is still a form of recursion until the skeleton is found. Single pass algorithms such as the algorithms of C. Neusius and J. Olszewski [NO94] firstly label all pixels within the object noting their distance from an edge. This initial pass forms the preliminary skeleton; this skeleton is not deemed complete because connectivity is not preserved. To improve connectivity, C. Neusius and J. Olszewski developed an algorithm to join the sections of the skeleton to form a completely connected skeleton.

Chapter 4

Theory and Design for Cunningham Map and the Brighton Map

4.1 Introduction

This Chapter discuses the specification of both the Cunningham map and the Brighton Map and the foreseeable problems with their implementations. The Cunningham map is a hand-drawn map created in 1558 and showing how Norwich looked at that time, while the Brighton map was created more like a modern map from a top down view showing only the footprints of the houses and roads.

4.2 Cunningham Map Specification

The Cunningham map model was produced in collaboration with Professor Carole Rawcliffe (School of Histroy, University of East Anglia). The main purpose of the model is to provide the audience with a fly-through of the city according to the view that W. Cunningham saw back in 1558. For the model to be able to meet this objective, a set of specifications were established as follows:

- Represent the Cunningham map as accurately as possible.

- Video of a fly-through from the view drawn by W. Cunningham in St. Andrew's Hall.
- Low polygon count (So that if needed it could be expanded on to work in real time).

4.2.1 Cunningham Map Analysis

To be able to create an accurate virtual environment, accurate data for the site is needed, for example photographs showing the scene from different perspectives or data about the positions of buildings. The problem with the Cunningham map (see Figure 4.1) is that there is only one perspective of the scene and that perspective is from a hand-drawn source. This means that the virtual model is based on an interpretation of W. Cunningham's representation of Norwich. The problem of the map being hand-drawn can be seen, for instance, by looking at the right-hand side of the Figure 4.1, where sheep can be seen having the same size as a wall.

4.2.2 Cunningham Map Model Design

The virtual model will concentrate on recreating all the architectural features of the Cunningham map such as buildings, churches and road network. This is because the map drawn by W. Cunningham focuses mainly on these aspects.

To render a fly-through video of the Cunningham map, a virtual 3D model of the drawn Cunningham map would have to be created to scale. Modelling of the Cunningham map leads to problems such as the skewed height of walls that can be seen in Figure 4.1, the walls would have to be much taller on the east side of the city compared to the west, as the east side wall appears to be of the same height as the west even though it is farther away. A decision was taken in order to keep the virtual model realistic and features such as oddly skewed wall heights were made uniform.

Figure 4.1: Presents the full Cunningham map.

The Cunningham map does not contain much information about the site. For instance, the dimensions of houses and their colours within the Cunningham map. The problem with a lack of detail led to the styles of the houses being taken from other known historical buildings within the same time period; this method was approved by Prof C. Rawcliffe.

The only accurate information within the Cunningham map is the position of churches and the general shapes of buildings. Although some buildings look out of proportion compared to real life, such as the cathedral which seems much larger than it actually is in real life.

The Cunningham map only offers a single view of the site, a lot of detail cannot be seen, for example features behind houses. Things that are missing from the map need to be modelled by reference to other sources in order to keep the model realistic. An example of where detail is missing can be seen in Figure 4.2. In this figure it is

Figure 4.2: Shows the small road gaps within the Cunningham Map.

hard to see if the road ends or if it goes behind the houses.

4.2.3 Cunningham Map Modelling Approach

Creating the entire virtual environment in three-dimensions would take a long time due to the high level of detail needed in the scene. 3ds Max was chosen as the modelling package because of the excellent features it contains for modelling architectural scenes. For the creation of textures, Adobe Photoshop CS3 (Photoshop) was chosen for its comprehensive tools for image manipulation and editing.

4.3 Brighton Map Specification

The Brighton map model unlike the Cunningham map model does not need to show the entire area of the map, because only a section of the map was requested. This cuts down the development of the virtual model drastically, as unlike the Cunningham map model each building has to be modelled individually owing to their irregular shapes.

The Brighton map follows similar specifications to those of the Cunningham map; they are both low polygons and should represent the map as well as possible. A video is shown flying around the virtual model, instead of flying over it, as shown by the Cunningham map.

Figure 4.3: Illustrates the entire Brighton map.

4.3.1 Brighton Map Analysis

The Brighton map (Figure 4.3) was created by Budgen in 1788: the map is drawn in colour whereby yellow represents roads, blue represents buildings, green represents fields and the pale orange colour represents gardens and open space. This map has many similarities with the Cunningham map (Figure 4.1) in that both are of residential areas and are historical maps. However the differences between the two maps are also apparent, for example, the Brighton map is drawn in colour while the Cunningham map is drawn in black and white.

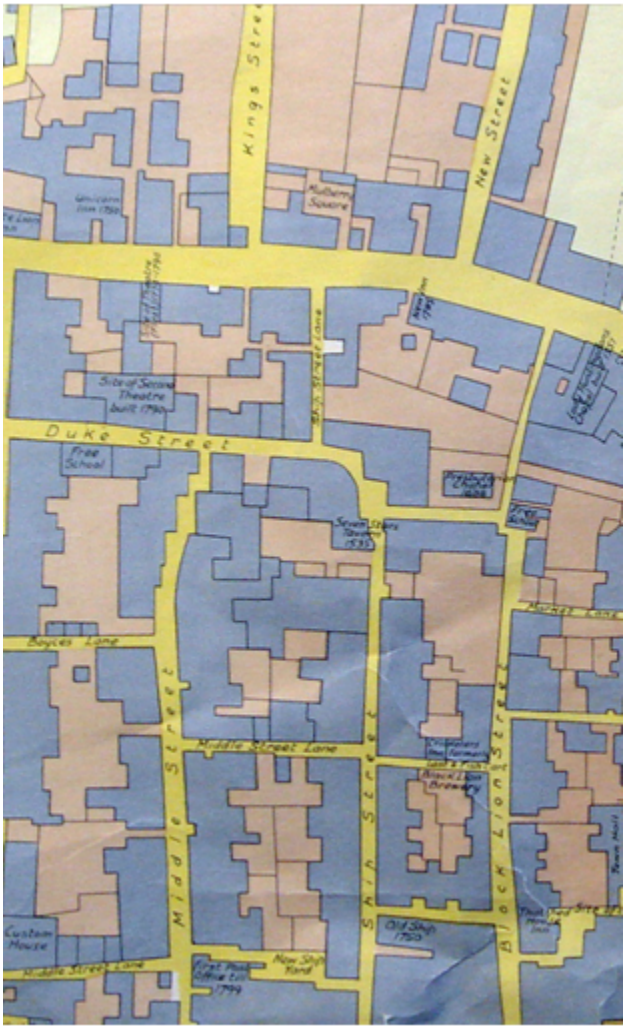


Figure 4.4: Illustrates the section of the Brighton map modelled.

The section of the map that is actually modelled is shown in Figure 4.4. This section of map is a good representation of the entire map, in that it covers a large area where roads, buildings and open space are included.

Towards the bottom right of Figure 4.4 there is a lot of noise. This is because when the Brighton map was photographed, it was not completely flat so the creases are clearly visible. Although this does not affect the modelling of the Brighton map model it could affect the automatic road extraction on such a map, as shown in

Chapter 5.5.

4.3.2 Brighton Map Model Design

As regards the design of the buildings it can be seen that the majority of them have irregular shapes, which means that each building has to be bespoke for that single building footprint. As there is no information as to the height of the buildings they are approximated by reference images; this is explained in greater detail in Chapter §5.5.

4.4 Lighting Design for the Cunningham Map and the Brighton Map models

After creating the virtual model, it needs to be illuminated, 3ds Max by default illuminates a scene using a global light source, which does not cast any shadows or do anything else with the light. Such a solution is inadequate for illuminating a scene that is supposed to be realistic.

Within 3ds Max there are two types of lights, called “Standard” and “Photometric”. “Photometric” lights allow an accurate representation of light sources such as a “35 watt halogen” light blub, as shown in Figure 4.5. Using “Photometric” lights allow the operator to easily set up realistic lighting. There are two types of “photometric” lights, “Target light” and “Free light”, although they are basically the same type, one contains a target while the other has no target point. “Standard” lights have three different types, Spot light, Directional light and Omni light. Although they have fewer functions, these are good enough for simple scenes. By using a combination of these lights, it is possible to create any illumination required within a scene.

When creating a virtual model, there are two main factors that contribute to the amount of time a scene takes to render, the complexity of the scene and number of

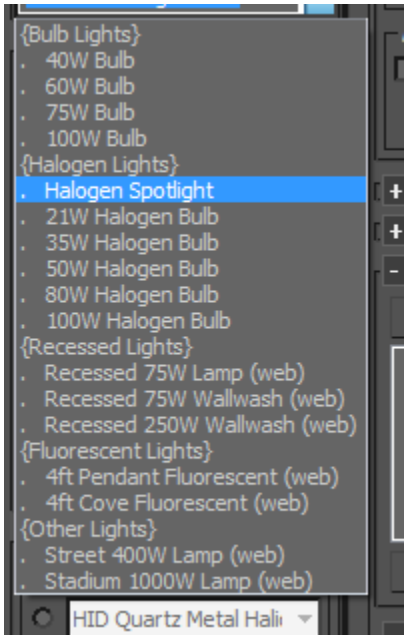


Figure 4.5: Illustrates photometric lighting options.

light sources. If a single frame takes many hours to render, a video running at 30 frames per second (FPS) could take many days to render. This is why when creating the virtual model it has to be designed to be illuminated with the smallest number of light sources. Within the Cunningham map, there is only a single main source of illumination, the sunlight. To recreate sunlight, a system is built into 3ds Max called the "Day light" system, which when set to "Photometric", can accurately recreate the sunlight's colour and intensity. Using the "Day light" system does not increase the render time significantly because it only contains two light sources, the sun and sky light.

4.5 Rendering Design for Cunningham Map and Brighton Map

Once a virtual environment has been created and its lighting set up, the next step is to render the scene. 3ds Max has Mental Ray V3.5 built into it. Mental Ray was chosen because it is fast to implement, provides good results and does not require additional software to be installed. Although Mental Ray does not need special materials like Vray or Maxwell, it does have a special set of materials called the "architectural and design materials", these materials have been optimised to represent metals, matt surfaces and water.

Mental Ray incorporates many algorithms for advanced lighting, it also has programmable shaders that can be added in the form of C or C++ code. Writing code for shaders can be time-consuming. An alternative is to use existing shaders written by others, which can be bought or downloaded freely at [Mat09].

GI is used to calculate the indirect illumination of a scene. Indirect illumination is usually more important than direct lighting, because, without indirect illumination, an object within a scene is either lit or not, causing very sharp shadows. Many GI algorithms exist such as the one proposed by X. Granier [GD04].

FG is a method used by Mental Ray to create accurate renderings. It works by tracing one ray from the camera onto the scene for every pixel within the output image. When a ray reaches any geometry within the scene, it traces more rays from that point called final gather rays, see Figure 4.6, these rays are traced out in all directions away from the surface of the object. Once the final gather rays reach a light source, using the distance of the ray and other variables it is possible to calculate the intensity of the ray. Any light calculated from the final gather rays is added to the direct illumination to form the total illumination. As can be seen, calculating FG is a very time-consuming process, tracing numerous rays and finding intersections,

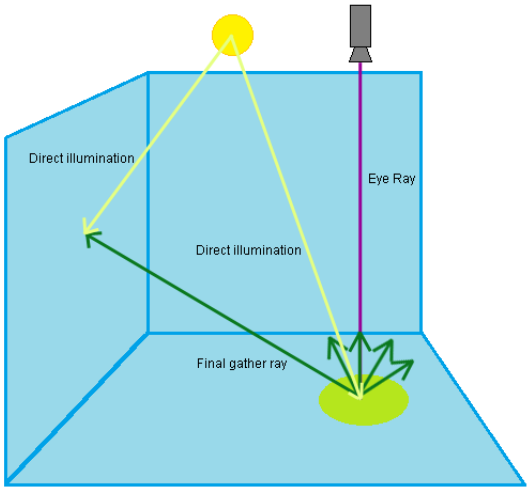


Figure 4.6: Illustrates how the "final gather" feature works.

because of this Mental Ray lets the operator save the FG map. The FG map should only be saved if the scene is static, because any movement in the scene would change the effects of indirect illumination causing the FG map to be useless.

When enabling features such as GI and/or FG, the rendering time for a single frame is increased, this is because of the extra computational power needed to calculate such effects.

Photon mapping is an algorithm that is used for calculating the interaction of light and objects within a scene. Developed by H. W. Jensen [Jen96], it is possible for photon mapping to simulate refraction, diffuse reflection and caustic. Photon mapping is a two-pass algorithm. The first pass is for emitting photons from the light sources within the scene. After all photons have been emitted, the positions of the intersections between the photons and objects are recorded. If a ray is reflected off a diffuse surface, the direction of the reflected ray is calculated using the surfaces' bidirectional reflectance distribution function (BRDF). Once the first pass is completed, all intersections and related data are saved in a file called the photon map. In the

second pass, rays are traced out for every pixel of the output image, these rays are traced until they meet a surface. When a ray finds a surface, the radiance is calculated in a three step procedure:

- 1: Gather all photons within a sphere "S", let these nearest photons be called "N".
- 2: For each photon, divide the amount of flux (real world photons) that the photons represent, by the volume of S and multiply it by the BRDF applied to it within the first pass.
- 3: The sum of the results for each photon represents the total surface radiance.

After calculating this, the rendering equation [Kaj86] is used to calculate the light leaving the intersection point. A caustic is a pattern that is focused on the surface of a translucent object, where the original light path has been altered. Photon mapping can recreate realistic caustics, by using the two pass method.

Rendering complex virtual environments consumes large amounts of resources. If a single frame takes 30 minutes to render, a single second of video running at 30 FPS would take 15 hours to render. To reduce render times, Mental Ray supports multithreading with near 100% scaleability [Ima10b]. To further decrease render time Mental Ray also supports network rendering, where more than one computer can work on a single rendering job. Larger jobs can be outsourced to others with faster computers. Using companies with render farms such as "Rebus"[REB10], who have 250 workstations available for network rendering, can greatly speed up render times.

Chapter 5

Implementation of Cunningham Map and Brighton Map

5.1 Introduction

The Cunningham map model was created for the UMG (Urban Modelling Group), who specialise in the creation of three-dimensional virtual models from either two-dimensional reference images or real world sites.

The specification for the Cunningham map model was that it needed to have a low polygon count and look visually attractive, for use in a fly-pass render. The Cunningham map model represents a whole city. As most of the buildings are duplicates of themselves, the geometry of each house is similar, with small changes in their X,Y,Z skews. To make the buildings more unique a large set of predefined textures are applied to them. The roads within the Cunningham map can only be described as all the areas where no building can be seen or grass because there are no road edges defined.

After developing the Cunningham map model the Brighton map model was created. The Brighton map was created by T. Budgen in 1788, so it looks more like a modern day map compared with the Cunningham map. The main difference between the two maps is that the Brighton map only offers a two-dimensional view from top-down while

the Cunningham map offers a three-dimensional perspective view. Another difference between the two is that the Brighton map is in colour while the Cunningham map is drawn in black and white.

Unlike the Cunningham map model only a section of the Brighton map would be modelled, this would follow the same specifications of the Cunningham map model, of using low polygon models. Because the roads are defined better within the Brighton map it is possible to create them more accurately.

5.2 Cunningham Map Preparation

There are three main problems when using photographs as textures within a virtual model, they are lighting, perspective of image and lens distortion.

Lens distortion is apparent when a camera is used to take pictures of a standard grid of lines as shown in Figure 5.1. To correct lens distortion, there are filters within Photoshop to reverse the process.

As the image is two-dimensional, it contains no depth data. This can cause problems if the surface within the image is not directly in front of the camera and is perpendicular to the camera. To correct the perspective of the image, Photoshop has a tool which uses the four corners of the map to realign and skew the image correctly.

Lighting is much harder to correct using post-processing techniques. When taking a photograph of a surface to be used as texture, the best source of lighting would be a uniform diffuse light source, such as a diffused flash. Although hard to accomplish when using a point shoot camera, a modern Digital Single-Lens Reflex (DSLR) camera can have an adjustable flash.

The only texture taken from the real life source was of the original Cunningham map, as illustrated in Figure 4.1. This image was provided by the UMG.

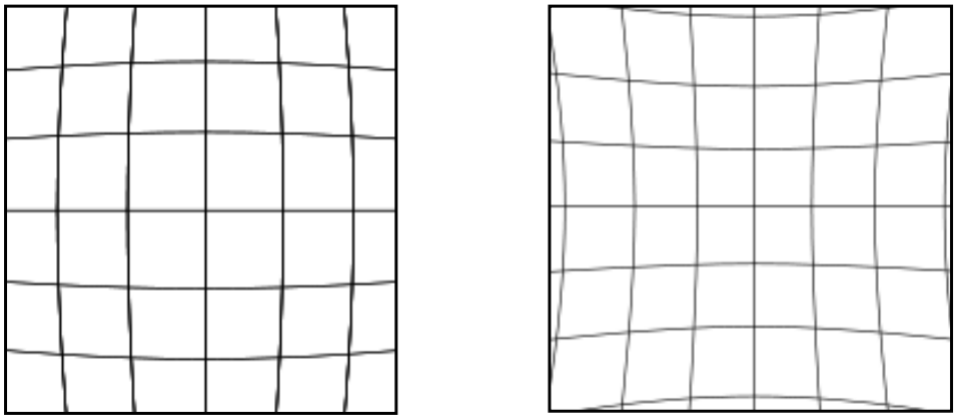


Figure 5.1: Illustrates the effects of lens distortion. (a) Barrel distortion. (b) Pin cushion distortion.

5.3 Modelling Cunningham Map

5.3.1 House Modelling

Each house consists of nine polygons, representing a cube with a prism merged on top, this compound object forms the basic house as shown in Figure 5.2. Using these basic houses, additional features can be added to the house such as chimneys or loft extensions with windows, these are the two predominant features of the houses within the Cunningham maps. To make the geometry of the houses more unique, each house is skewed slightly differently to represent the fact that houses built within this time period where initially created straight but changed overtime.

A generic set of ten master houses were created, these houses could accommodate the majority of houses within the Cunningham map. These master houses are unique to each other in that their geometries or textures are different. Once these master houses were placed, all the houses were skewed individually to further increase the diversity of the houses. The houses not covered by the generic set were all created as bespoke houses in each situation.

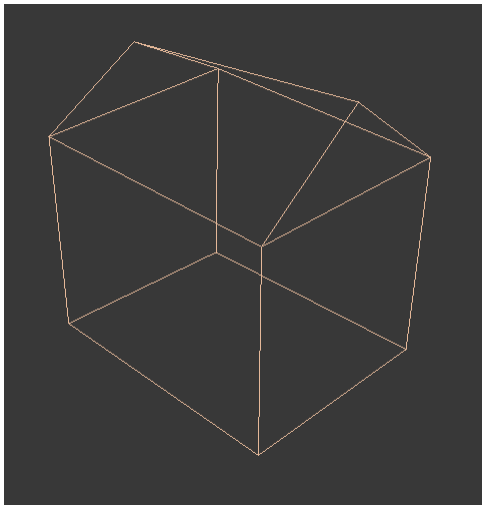


Figure 5.2: Illustrates the simple use of polygons within each building. Note the irregularities in the roof shapes, to make the houses more unique.

5.3.2 Modelling of Landmark buildings

The Cathedral is one of the main buildings within the Cunningham map and as the UMG already has a model of the cathedral it was requested for this project, but their model was found to have a very high polygon count of over 280,000 polygons as shown in Figure 5.3 (a). After heavy editing of the model's geometry to delete hidden polygons and simplify the model, its polygon count was reduced to 85,000 polygons as shown in Figure 5.3 (b).

The cathedral is one of four models which if left unedited would have increased the complexity of the scene drastically, the other three models being, St Andrew's Hall, the Cross building and the Castle, these models like the cathedral were edited to have low polygon counts. All of these models had to have the complexity of their geometry reduced. After reducing the complexity of these models, they could fit within the virtual environment without affecting the overall polygon count as much as they did when unaltered.

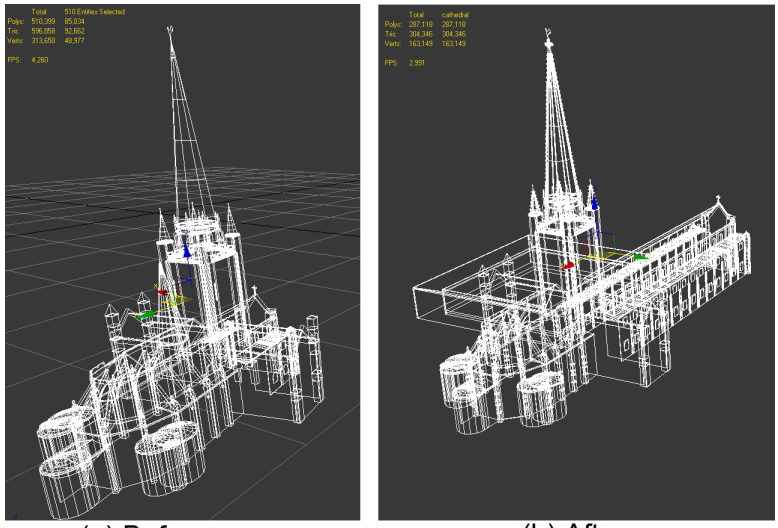


Figure 5.3: Diagram showing the high polygon cathedral (a) and the modified low polygon version (b). To remove unnecessary polygons, any internal polygons that could not be seen from the outside were removed first. After this any areas of high details, for instance, the roof was simplified then any geometry that could be replaced with textures was done.

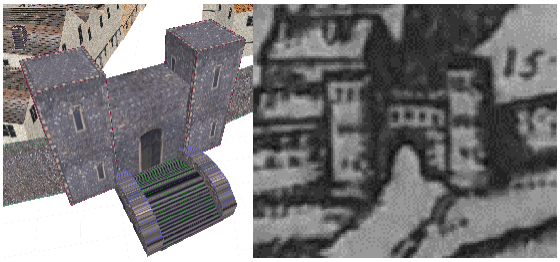


Figure 5.4: Diagram comparing the difference between the three-dimensional model of the gate house and the original Cunningham map gate house.

5.3.3 Modelling Walls

To create the walls, the dimensions of the walls must be measured, but because the whole image is warped and the height of the wall is uniform throughout the image, which would mean that the wall would have to be substantially taller at the east side than the west. To compensate for this effect, the Cunningham map was only used as the basis of where the walls should be placed, the height of the outer wall was kept constant so that the model would look realistic from all perspectives. The wall is created by first creating a single section of wall which is a cube, this single section of wall is then extruded to create the next section and then that section is extruded to create the next, repeating this process until the whole wall is created. By this method, the number of polygons used to create each section of wall can be monitored and each wall section can have more detail when the wall is curved and less detail when the wall is straight.

Other more unique sections of wall such as keeps and gate buildings require more complex modelling, these buildings were created to look as they did within the Cunningham map, as shown in Figure 5.4.

5.3.4 Modelling Trees

A lot of research has gone into the modelling of trees using L-systems, but for the purposes of this model only low polygon trees are required so a much simpler solution was needed to keep the polygon count low.

Previous work on this problem was conducted prior to taking on of this project, with the outcome being that it could be possible to model low polygon trees using basic shapes. The solution is to use two ellipsoids with one inside the other, and a cylinder for the tree trunk as illustrated in Figure 5.5, to represent the branches and leaves of the tree. The two ellipsoids are textured using a texture created in Photoshop, then this texture is used in conjunction with an opacity map so that parts of the ellipsoid seem to become transparent. Using this low polygon model which consists of 856 polygons, it is possible to create entire forests at a relatively low polygon count unlike a tree system such as the one built into 3ds Max which would use more than ten thousand polygons per tree.

A low polygon tree was needed because on the far side of the Cunningham map there is a forest as shown in Figure 4.1. Using a high polygon model to represent the trees would have been a waste of polygons but using a static back drop would have meant that the model could only be viewed from a single perspective. For these reasons, the entire forest was created using low polygon count trees. Much like the set of master houses used to create all the houses within the virtual environment, a set of master trees was used, with different heights and sizes. As only the dimensions of the tree were changed, not their complexity, the polygon count remains the same for both large and small trees.

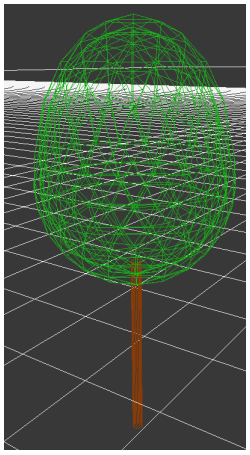


Figure 5.5: Diagram showing use of polygons within a single tree.

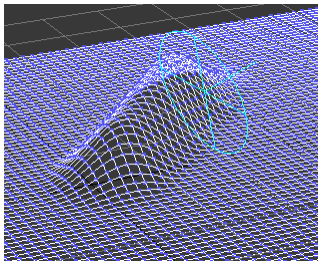


Figure 5.6: Diagram showing how the pull modifier can create simple hills.

5.3.5 Ground Modelling

The ground model is created to represent the ground within the Cunningham map. This model has to represent the texture of the ground as well as the hills shown in the Cunningham map. There are three main sources of height deformation in the map, these are located in the bottom right corner where the sheep are, where the castle is and in the top left corner where there are two windmills (see Figure 4.1).

To create these height deformations, the "Paint Deform" tool built into 3ds Max was used, this allows for the pushing or pulling of vertices to create smooth mountain ranges quickly and effectively, see Figure 5.6.

5.3.6 River Modelling

Within the Cunningham map, there is a river that runs from the bottom left of the image in an "S" pattern towards the top right corner. As well as a river, there are another two sources of water, the moat that runs around the outside of the city and a smaller moat that runs around the castle. As the sources of water are not viewed close up and there is no detail about how deep they are, they are modelled using a spline which has been extruded into a two-dimensional polygon. After creating the river geometry, they are textured using a texture that replicates the surface of water, this is accomplished using the built-in materials library within 3ds Max.

5.3.7 Road Modelling

A major problem was found when creating the road network within the Cunningham map. This is because, unlike other features within the image, the roads have no defined size, the road network seems to be the area where there is an absence of a building or wall or farm. Another problem with the road network is that parts of it are hidden from view when the Cunningham map was created as illustrated in Figure 4.2. After consulting with C. Rawcliff, a historian, it was determined that the main roads would be paved with a rubbish gutter running down the middle, but smaller roads would just be dirt tracks. Although this was not shown in the Cunningham map, it was modelled in this way.

5.4 Texturing Cunningham Map

5.4.1 House Texturing

As the specification of the virtual environment stated that it should have a low polygon count, most of the detail in the model would have to be created with textures. Textures being images also use up memory so the smaller the texture the faster the

render time will be, but using small textures that are tessellated across a large area looks very unrealistic. When using larger textures, more memory is required, also the time taken to render the scene is increased although not as much as with an increase in geometry. The textures used for the houses have to be of sufficiently high resolution as a single texture must be stretched across the whole face of each house, this image cannot be tessellated because otherwise two doors end up on the same side of the house. The actual resolution of textures used range from 120 x 130 pixels, to 1800 x 1300 pixels. Using large textures and with each house requiring four large textures, and four small textures with the bottom plane left un-textured, each house has a large memory footprint when rendering. To compensate for this increase in memory, rendering of the virtual environment is done using an Intel Xeon quad core with 8 GB of RAM, although this scene could be rendered on a much more modest computer, using a faster computer meant that videos could be rendered in a matter of hours.

The textures are created in Photoshop, then saved as bitmap images to be used in 3ds Max. The textures are applied to the model by either using “Face Definition” or “UVW mapping”. “Face Definition” is when each face is assigned a value from “1” to “n”, these values represent the texture ID linked to that face. For “UVW mapping”, the texture is first mapped onto an intermediate object, then this object is mapped onto the object that is to be textured, using this method it is possible to create seamlessly textured surfaces.

To create normal maps, a third party program called CrazyBump [Rya10] is used, this tool allows for real-time manipulation and creation of other maps such as displacement, occlusion, specular and diffuse.

While a normal map would normally contain RGB channels, displacement, occlusion and specular maps are all monochrome, this is because they are a measure of intensity

per pixel of the texture instead of vector per pixel of texture as in the case of a normal map.

5.4.2 Textures of Other Buildings

The texturing of the walls is also done using "UVW mapping", using a stone texture similar to the church texture. Using this simple method, it is possible to keep the complexity of the model to a minimum, so that if the model needs to be changed later on by another user, it can be done without learning many techniques.

5.4.3 Tree Texturing

The leaves of the tree are simulated using an opacity map on a green/brown texture, the opacity map allows the camera to see though some parts of the ellipsoid but not others and, since the two ellipsoids overlap each other, they form what look like leaves from a distance, see Figure 5.7.

5.4.4 Ground Texturing and Other Figures

The ground was textured using a grass texture that was tessellated across the whole ground, then the roads were textured with paved stones if it was a main road, otherwise they were textured using a brown mud texture to represent a dirt track road.

5.5 Modelling the Brighton Map

The Brighton map has a different set of problems compared with the Cunningham map, the main one being that there is no accurate reference to the heights of the buildings or what the buildings look like in this area as the map only shows the footprints of the buildings. To assess the heights of the buildings measurements were taken from images such as those shown in Figure 5.9. If we estimate the average height

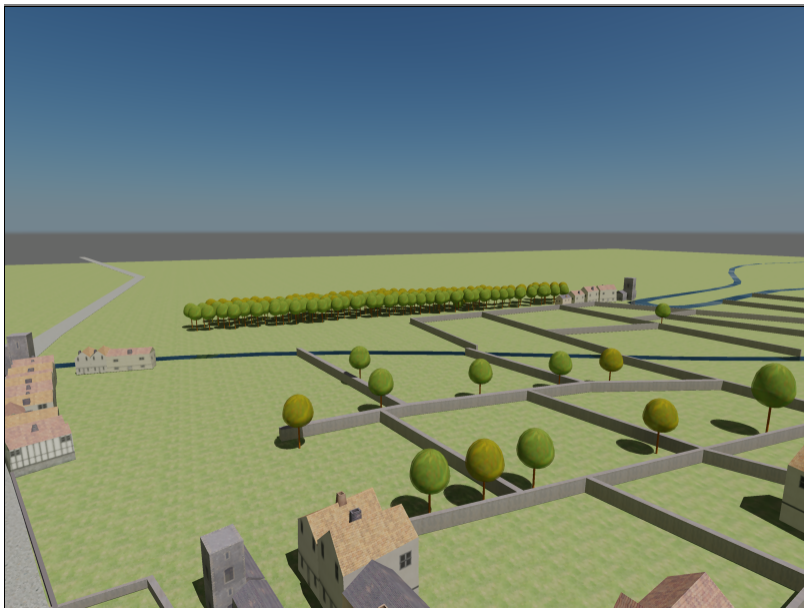


Figure 5.7: Diagram showing how texturing can be used to create a tree that is acceptable when viewed at a great distance.

for a tall door is 250.00 cm, it is possible to calculate the approximate height of the building as 1632.00 cm; although it is not possible accurately to relate the dimensions of the heights acquired from the image and the dimensions of the footprint, it gives a rough guide to how tall the buildings should appear.

To model the houses a simple block modelling technique was used because the majority of the buildings within the map are irregular shapes with convex as well as concave parts. Using this method gives a fast and accurate result. Starting with a cube, extra sections are extruded out and then the vertices moved onto the perimeter of the footprint, as shown in Figure 5.8. This is repeated until the whole building is created. The height is easily modified by simply moving the top faces of the building.

This technique is different from the one used in the Cunningham map in that each building has to be created specifically for that purpose owing to its irregular shape. This meant that only a section of the map was modelled unlike the entire

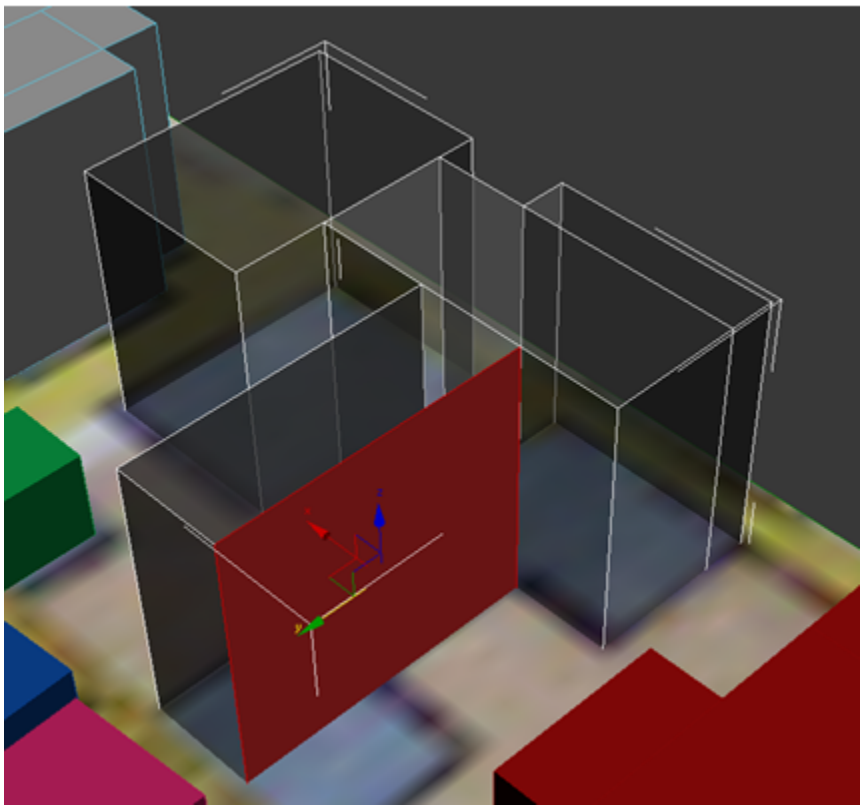


Figure 5.8: Screen dump illustrating how a building is created using box modelling.

Cunningham map.

5.6 Texturing the Brighton Map

Due to insufficient data on the appearance of buildings on the Brighton map, several scanned images were requested from Philip Brown (UEA Researcher). These scanned documents show buildings of a similar time period, and some of the images such as those in Figure 5.9 are good enough to be used as textures within the Brighton map model.

Other scanned images show generally how uniform the buildings appear as shown in the appendix A.1. This means that the buildings were probably constructed at the

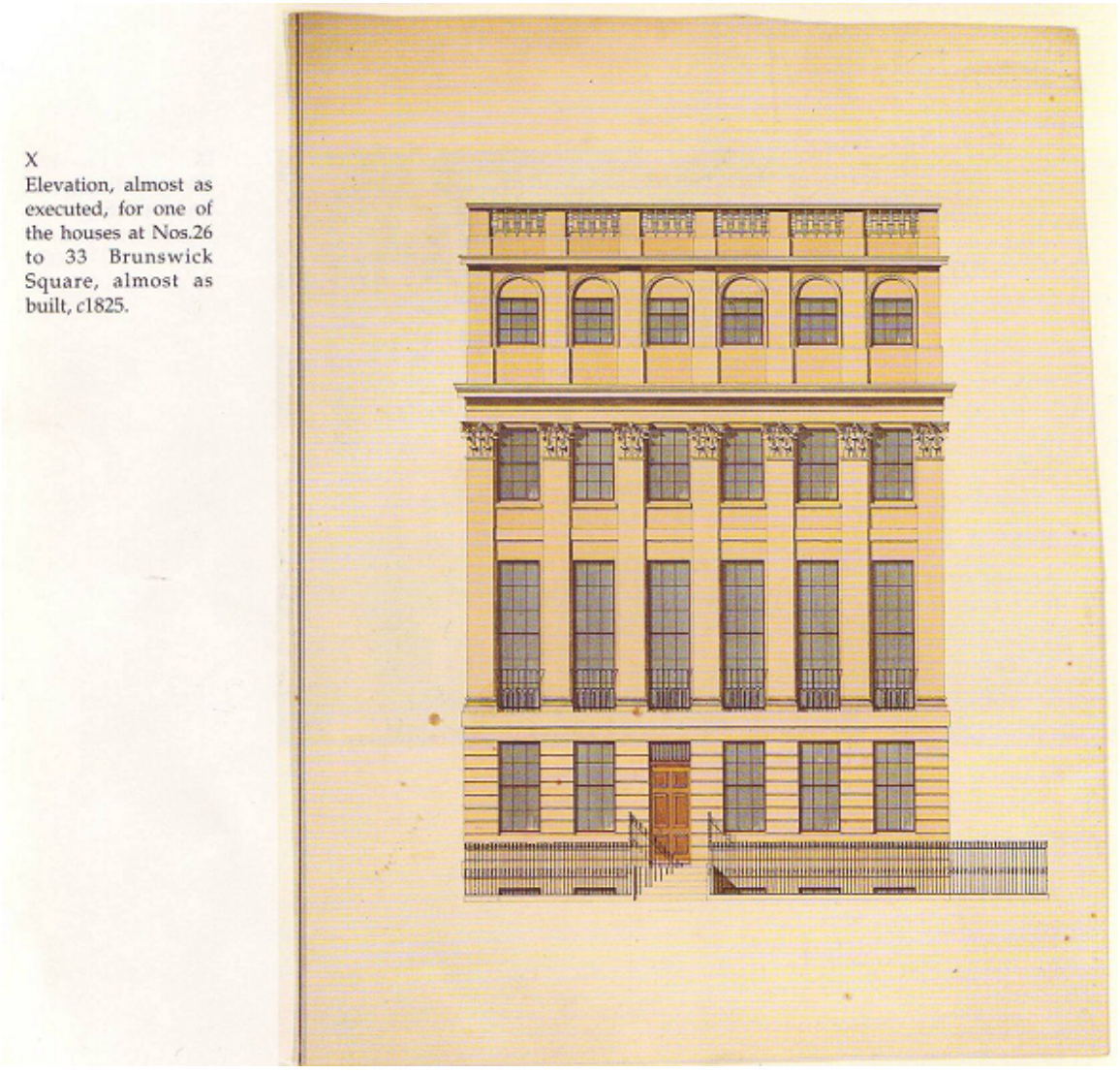


Figure 5.9: Screen dump illustrating how a building is created using box modelling.

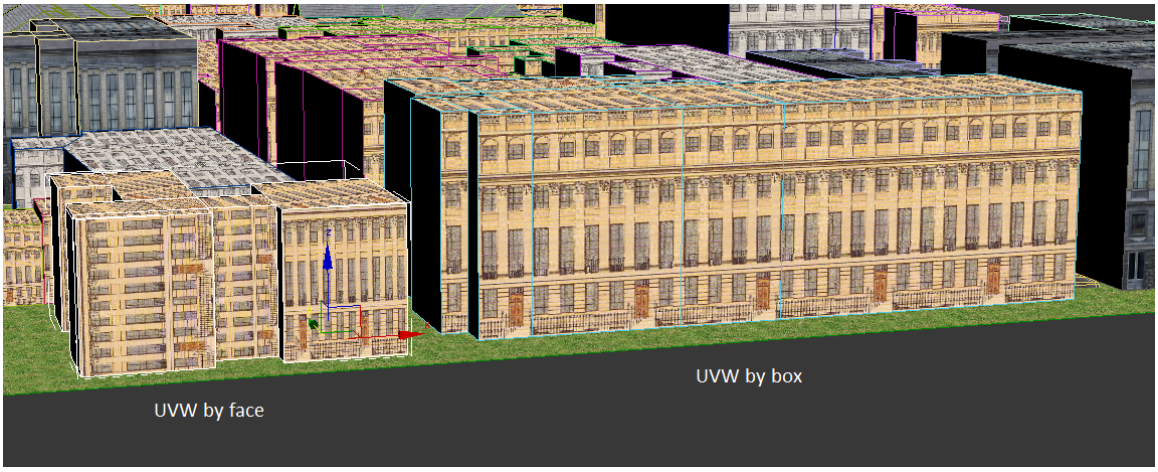


Figure 5.10: Screen dump illustrating how a building is created using box modelling.

same time and therefore were painted in similar colours to each other, a supposition further supported by a photo as shown in the Figure A.2.

Applying the textures to the buildings requires a different approach from that in the Cunningham map because each face within the model is of different size, which causes problems if each face is assigned a single texture because it could cause the texture to appear squashed on one face and elongated on another, producing an unrealistic appearance. The way in which the face of the polygon was created also matters when UVW mapping is used, as this can cause problems as shown in Figure 5.10. To solve these problems a different UVW setting is used, UVW mapping by box, which uses the texture first to map onto an intermediate object, in this case a cube, then mapping this cube onto the surface of the model. This approach is perfect for texturing a model which contains many flat sides which lie perpendicular or parallel to each other. The results of UVW mapping by box as compared with UVW mapping by face are shown in Figure 5.10

The modelling of the roads was accomplished in a similar way to how they were created in the Cunningham map. This gives a two-dimensional road network;

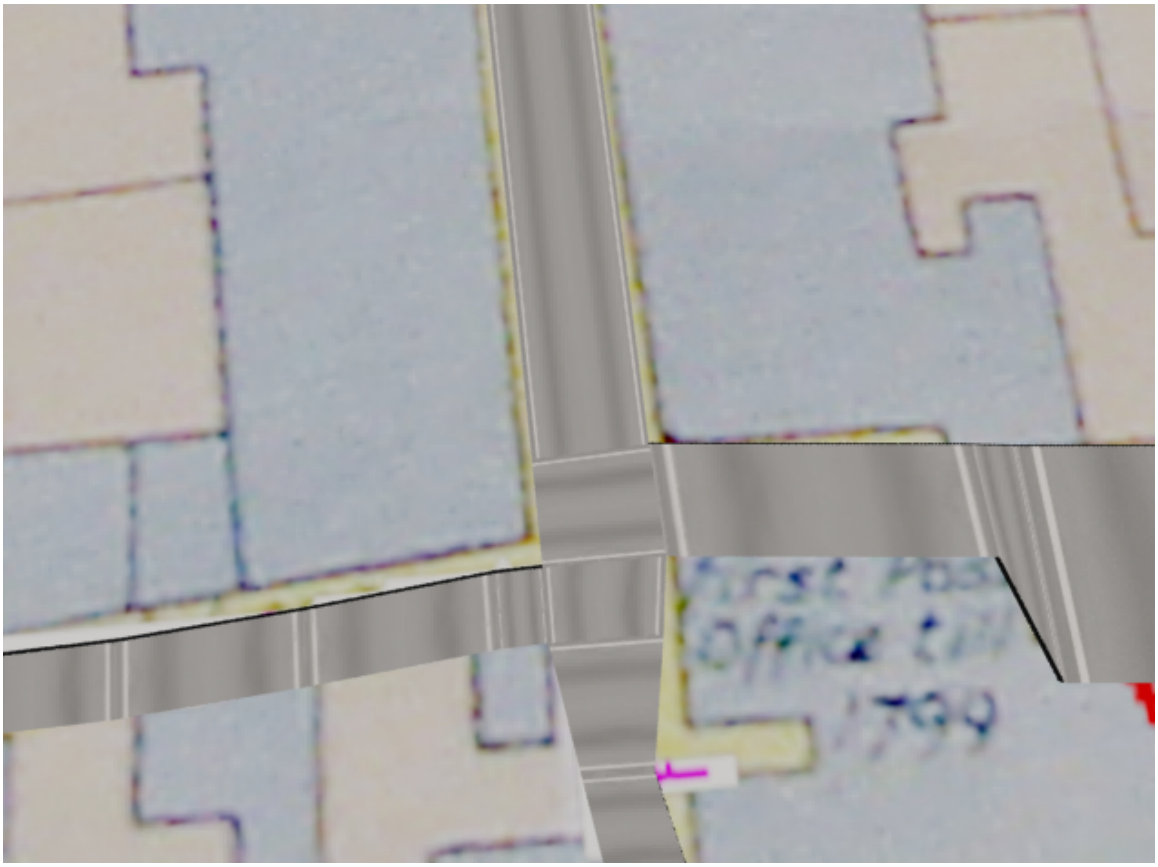


Figure 5.11: Screen dump showing how the texture errors appear when UVW mapping is applied per face, on a joined road network modelled by hand within 3ds max.

creating a three-dimensional road network would require the box modelling method as described in Section §5.5. Instead of modelling a building for a road network, doing this would take a very long time because all the vertices of the road have to be positioned by hand. Although using a process like this where the entire road network is joined together limits how the roads can be textured, because polygon orientation causes textures to be rotated 90 degrees in some sections compared with others, see Figure 5.11. Using an automated approach for extracting and modelling the roads would speed up the modelling process, and this is discussed in Chapter §3 and implemented in Chapter §8.

5.7 Lighting Cunningham Map and Brighton Map

5.7.1 Day Light Illumination

The "Day light" system is based on two light sources, the sun and the sky. Using this system offers many advantages over setting up the lights manually, for example there is a built-in system that allows the user to specify a time and location so that the "Day light" system automatically recreates the lighting effects, by positioning the sun in the correct place as well as the colour of the sun and sky.

5.7.2 Omni Light illumination

Omni lights are needed within the scene because, once a light source is introduced into the scene, the standard global illumination is turned off. To get an even spread of light on all surfaces, the use of four Omni lights positioned at various heights at the four corners of the scene, with their decay rates set to infinity and set to cast no shadows, give a uniform global illumination to the model, much like how a scene is lit when the sky is cloudy. Using Omni lights like this is a fast and effective way of increasing the realism within a scene, because calculating the exact paths of light using an unbiased rendering algorithm would only increase the time taken to render the scene, also the models are not of very high detail so the result would be similar to using Omni lights.

5.7.3 Shadows

There are many ways to calculate shadows within 3ds Max, the method chosen for this virtual environment is called ray tracing. Based on the ray tracing rendering technique, it calculates the shadows by tracing rays. Although this takes longer than other methods, it can produce accurate shadows and this is important so as to increase the realism of the scene without greatly affecting the complexity of the scene.

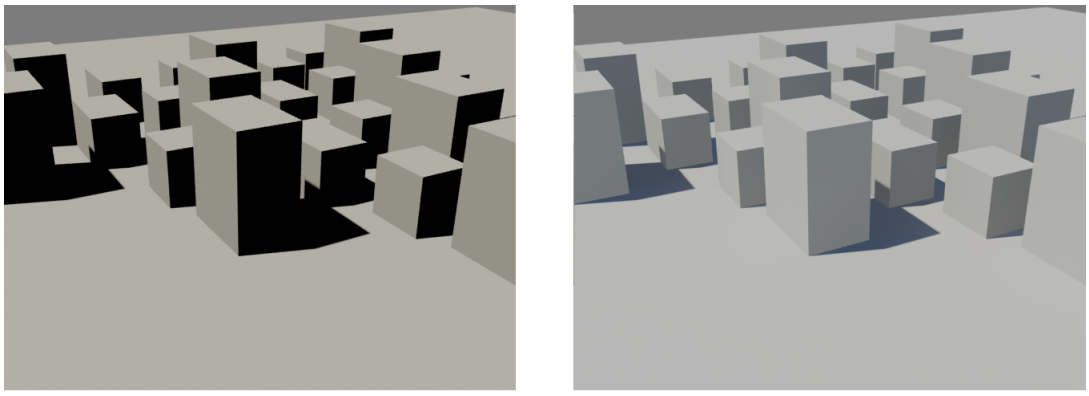


Figure 5.12: Illustration of the difference between using a daylight system with and without FG and GI used and not. (a) Scene rendered without FG or GI (b) Scene rendered using FG and GI.

A problem when using ray tracing shadows is that the shadows it produces although accurate in shape are unrealistic in tone. As shown in Figure 5.12(a), ray tracing shadows are of pure black, this is an unrealistic representation of how shadows look in real life, because there is illumination from other sources that needs to be taken into account. To calculate the other sources, both FG and GI are used so that the shadows are more realistic as they are no longer pure black, as illustrated in Figure 5.12(b).

Chapter 6

Results of the Cunningham and Brighton Maps

6.1 Cunningham Map Renders

With the entire Cunningham map model and lighting set up, a single image at a resolution of 640 X 480 takes 50 seconds to render on an Intel Quad Core 2.4 GHz with 6 GB of RAM, this is a reasonable amount of time as this is a worst case situation where the camera can see the entire model. This worst case situation render is shown in figure 6.1.

As shown in this top-down render, all the features of the Cunningham map have been recreated in three dimensions. Figure 6.2 shows how the model fits on top of the original Cunningham map. Within this render it can be seen that the position of the models do not fit exactly with their positions on the Cunningham map, this is because of various factors. For instance, if the model were recreated exactly as the map looks, then houses would appear on top of each other due to the skewed perspective of the original Cunningham map.

To render the final video, a series of bitmap images are rendered first, because a video is really just a sequence of images shown at a speed greater than 30 FPS. The main reason for rendering out bitmap sequences instead of the video file straight

Figure 6.1: Render showing the entire Cunningham map virtual model, from a top down view.

Figure 6.2: Render showing the entire Cunningham map virtual model, super imposed onto the original Cunningham Map.

away, is that 3ds Max tends to crash frequently when rendering large scenes in the final gather. The most common crash is when Mental Ray runs out of memory, using bitmaps the render sequence can continue from where it crashed while rendering a video file would mean restarting again from the beginning. The final video lasts 1 minute and 25 seconds, it comprises three separate cameras:

1. Initial splash screen and fading from map to virtual model.

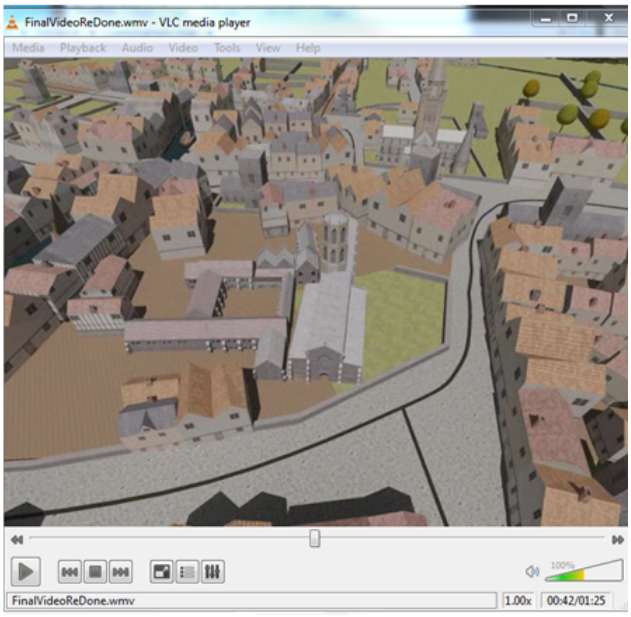
2. Fly across Cunningham map.

3. Around St. Andrews building in a 360 degree arc.

The total number of frames needed for this video is 2251, because the first 10 seconds is a single frame with text. With close-up renders of St. Andrews taking 43 seconds, while long distance renders take 50 seconds, using a single computer it took over 29 hours to render all the bitmaps. The sequence of bitmap images was stitched together using RadVideo [RAD10], this program allows for the stitching of images into a video sequence and the compression of such files. Compression is needed as the uncompressed video size is well over 2.7 GB. Using compression options available in RadVideo, it is possible to compress it down to 22.5 MB, although there is a drop in the quality of the video compared with the uncompressed video which is really just playing all the bitmap images in sequence. Although it is possible to compress the file to an even smaller file size, doing this would dramatically reduce the quality of the video.

As can be seen in a comparison of the two images in Figure 6.3 (a) and (b) there is a slight difference between the compressed screen dump of the video and the uncompressed bitmap image, the main difference is the washout of colours and the image appears to be more blurred even at the native resolution of the video. Although slightly blurred, the video still works really well in showing the geometry

(a)



(b)

Figure 6.3: (a) Shows how the original bitmap render looks like (note how sharp the image appears). (b) Shows how the same scene appears within the video after compression.

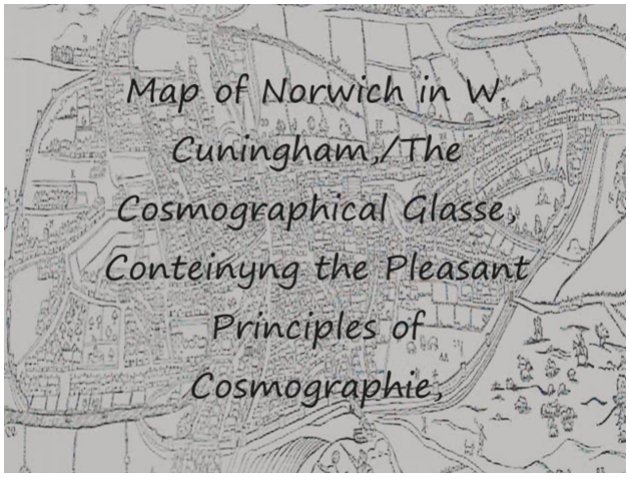


Figure 6.4: Illustrates how the text appears during the initial 10 seconds of the video.

of the houses, their colour and also the position of each building. Using the mixture of low polygon houses and the reduced polygon version of the important buildings like St. Andrews, it creates a scene where these buildings can blend in, as shown in Figure 6.3 (a) where in this render both St. Andrews and the cathedral model can be seen.

As three cameras are used, there are three short video sections that have to be laced together to form the final video. To do this, Windows movie maker was used, using this also allows for the text in the splash screen to be added easily as shown in Figure 6.4.

6.2 Brighton Map Renders

Like the Cunningham map model the Brighton map model was rendered by the same settings, and on the same computer, but since the scene is much less complex it only takes five seconds per frame in the worst case, as shown in Figure 6.5.

Comparing the two pictures, one given by the automatic road extraction and the other not, we can see in Figure 6.6, the geometry in the manual modelled roads is

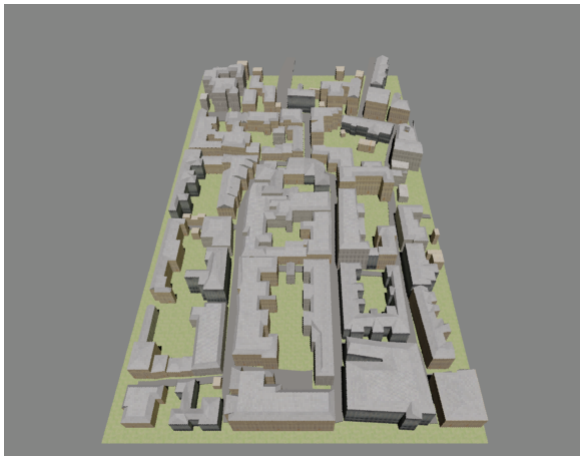


Figure 6.5: Showing the entire virtual model of Brighton

more complete as there are gaps within the automatically created ones.

The time taken to create the manually extracted roads was 30 minutes, and running the automatic road extraction algorithm (Chapter §8) and fixing any polygon overlaps took 20 minutes; on a larger map the gap between manually modelling the road network and using this semi-automated approach would increase even more, this is because fixing small errors is easier than creating new sections of roads and manually texturing them. If we used a method that allows for texturing like the automated approach the time gap would increase further because such roads would have to be individually built, much like the way in which the max script code reconstructs each section of the road independently.

Another way to compare the two sets of roads is to see how accurate the automatic road extraction algorithm is, this is discussed in Chapter §9.

The final video for the Brighton map model lasts 20 seconds, and the video pans around the perimeter of the Brighton map showing how the model appears to be superimposed on the actual map, much like the Cunningham map video. As the video path is less complex than the Cunningham map model it is possible to produce

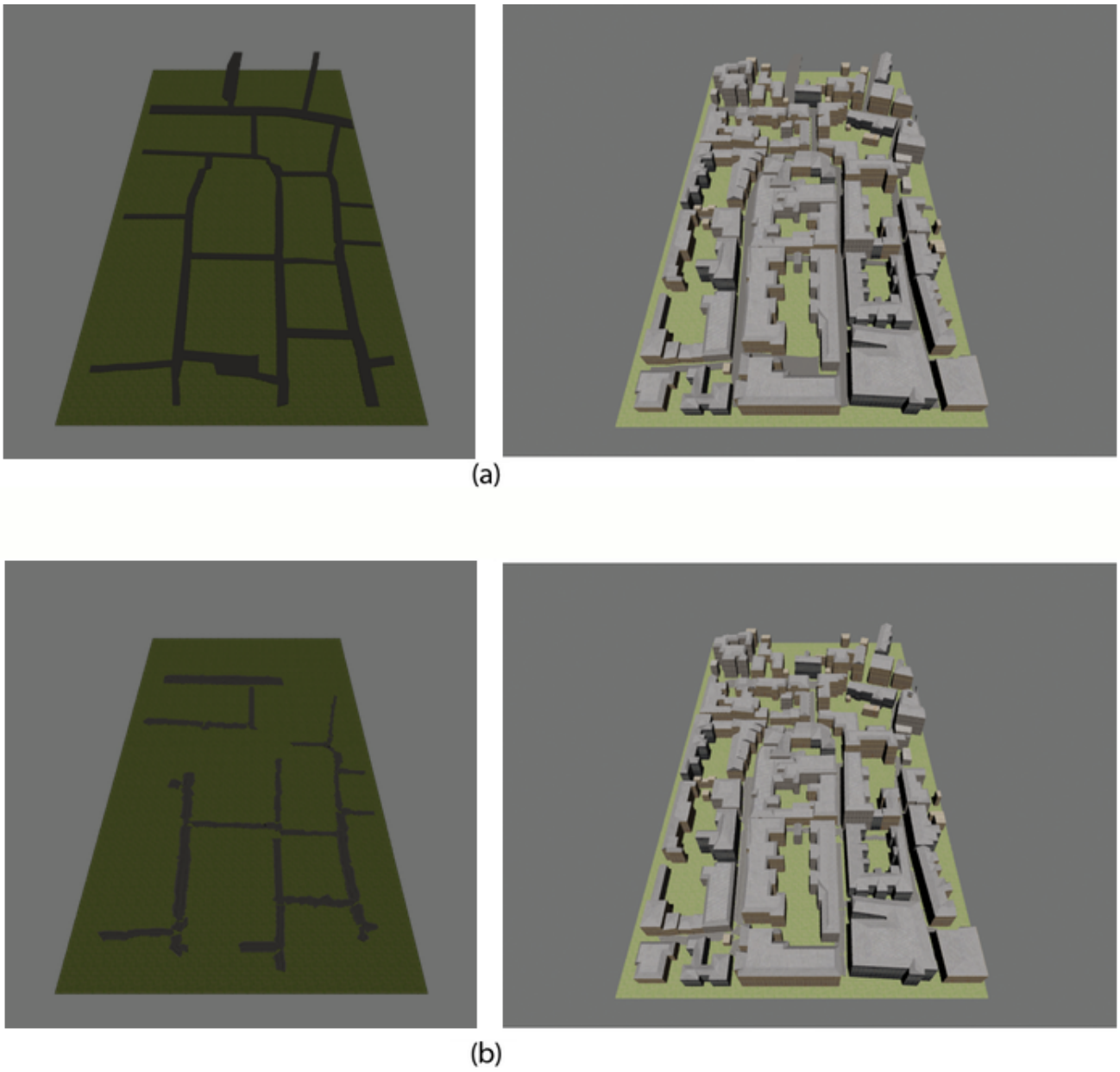


Figure 6.6: Illustrates the difference between manually modelled roads (a) in 3ds Max and automatically generated roads (b) using the road extraction algorithm and the max script described in Section §8.5.

the entire video with a single camera locked to a single path, leading to a faster post-production time. As noted in Section §6.1, compression can cause a loss of picture quality. This is also the case with the Brighton video, because it is not easily possible to use a multi-gigabyte; compression is needed.

Chapter 7

Theory and Design for the Road Extraction Algorithm

7.1 Introduction

This Chapter discusses the design of a novel algorithm for road extraction.

7.2 Possible Solutions to Road Extraction

7.2.1 Manual Creation

Modelling road networks using three-dimensional modelling packages such as 3ds Max is the most intuitive way to recreate road networks. Although using manual modelling packages gives a high degree of accuracy, it takes a much longer time to recreate large road networks compared with procedural algorithms.

7.2.2 Procedural Creation

Procedural modelling of a road network is done in two stages. Firstly, the road is separated from the rest of the image through segmentation. Secondly, the straight skeleton of the segmented road section is searched. Doing this would result in a set of points representing the centre of the roads. In real world applications, the algorithm may be deflected by noise within the image, as well as varied thickness within the

road. A new algorithm is proposed in this thesis which is based on a circle expanding and contracting. The skeleton of the road network is found by moving the circle around the road network. By limiting the input variables, it is possible to create a solution more quickly:

- Binary coloured image, where roads are white (colour images can be used, if the roads are significantly lighter in colour than the background. The image the be passed though a filter to make it binary, making the roads white).
- Buildings surrounding the roads are filled in.
- Small picture size 256 x 256, this is to reduce computational time as well as reduce memory footprint. Larger maps can be processed by simply dividing them up into sections.

Applying the above rules to the input data, the initial position of the circle must be on a white pixel. From this starting position, the circle would expand until it finds the adjacent edge of the road network. As the circle moves, the position and width of the circle are recorded. These values represent the skeleton of the road network and the width of the road at that point.

The benefit of the proposed algorithm over ones such as C. Zhang et al. [ZMB99] is that it always tries to preserve connectivity, as such they will inherently accept all widths of roads as legitimate road sections, as long as they are over a certain length defined within their algorithm. The algorithm proposed within this thesis is also more robust when compared with straight skeleton algorithms because, as can be seen in Figure 7.1, the circle moves through noise with minimal effect on its path. Current road extraction techniques also do not take into account the widths of roads, this means that when reproduced the roads would have a uniform width.

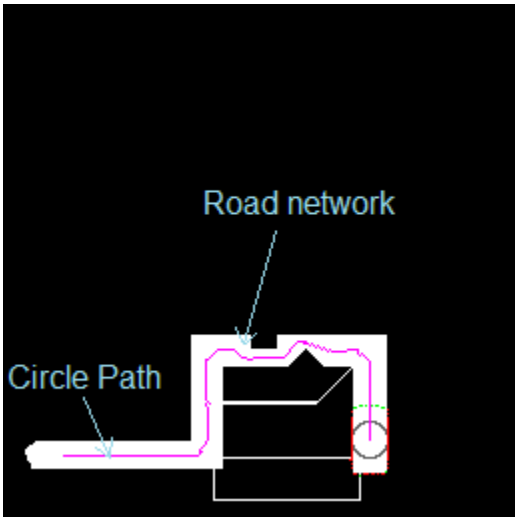


Figure 7.1: Shows how the circle moves though the road network avoiding narrow pathways.

7.3 Procedural Road Extraction Algorithm Design

The road network can be thought of as a complex polygon that is undefined in size and shape, until it has been extracted. Using a straight skeleton for the road network would be possible, if the roads do not contain much noise or errors. Any errors within the input map would cause errors within the output data.

The algorithm proposed in this thesis can be split into three distinct sections:

- Circle dynamic size.
- Junction / large area handling.
- Circle movement.

Each section handles a specific aspect for extracting the skeleton of the road network.

7.3.1 Circle Size

The size of the circle is directly proportional to the size of the road, as the road gets wider the radius of the circle increases, if the road gets thinner the radius of the circle decreases. The circle can track changes of road size, because at the start of every iteration, the algorithm performs a size check to make sure the radius of the circle is correct.

A circle is said to be at the correct width when the edge of the circle touches two unique sites of collision. If the edge of the circle goes over the boundary of the road edge, the circle should shrink to remain inside the road network. If only a single edge is found, it should expand to search for other road edges. To prevent it growing and shrinking out of control, a limit is placed on the maximum and minimum size of the circle.

7.3.2 Junction Handling

Junctions are defined as where the circle checking can see more than two unique paths. Once a junction is found its location, as well as details about all the paths leading away from this junction is recorded within the rooted linked list structure. Once all the paths have been recorded the circle is moved to a new empty path to start mapping out the road section from there.

Using a rooted linked list structure, allows for the algorithm to efficiently map out all the junctions within the map. If a new junction is found that exists already within the data structure, this new junction is not recorded instead the existing junction should be updated with the relevant information from the path leading towards it.

7.3.3 Large Area Handling

Large open areas cause algorithms that perform sequential thinning to slow down dramatically. This is because thinning an area twice the size takes approximately four times as long, because the area it covers contains four times as many pixels. Parallel algorithms would also slow down, but only because they have to mark extra pixels for removing. The new approach taken in this thesis can be broken down into three steps:

- Check to see if the previous six iterations were growing. Six was chosen by use of trial and error to find the optimal number of iterations to capture the find large areas, but enough iterations to prevent false events.
- Scan area for possible unique exits.
- Mark area as a junction and all possible exits as new paths.

A special case exists when no exits are detected. When the circle detects this situation, instead of classifying it as a junction it classifies it as a dead end.

7.3.4 Circle Movement

The path of the circle represents the shape of the skeleton of the road network. As the circle moves around the map, it increases the size of the skeleton. During each iteration of the algorithm, the circle checks to find the next possible directions. Legal paths for the circle to take can be calculated in four ways:

- If two unique collision sites are found, combine the two unique collision site angles to form a resultant angle, see Figure 8.2.
- If the circle is too close to a road edge, then move perpendicular to the road edge.

- If the circle has only one collision site, predict the next possible move by checking what the circle can see along its line of sight.
- If the circle has just moved to a junction, force it to move by the predetermined angle saved when the junction was calculated.

Using these methods it is possible to guide the circle around the map and keep the circle as close to the centre of the road as possible.

7.3.5 Circle Visibility of Surroundings

When the circle moves around the map, there will be places where it can only be in contact with a single unique collision site. In this situation, the circle will check its surroundings. The circle would start by expanding the circle and testing to see if two or more unique collisions exist. Once two unique collision sites are found, the resultant vector can be calculated. The radius of the circle is then restored back to its original radius.

Chapter 8

Implementation of the Road Extraction Algorithm

8.1 Introduction

This chapter describes the implementation of the individual parts of the novel road extraction algorithm.

8.2 Image Pre-processing

8.2.1 Binary Filter

To convert a colour image into a binary image, every pixel in the image is sequentially converted into its grey scale counterpart then into its binary counterpart. Using a scanline method initiated from the bottom left to the top right, each pixel is analysed using its RGB channels to determine their grey scale values (Grey Value = 0.3*Red + 0.59*Green + 0.11*Blue). To convert into binary, the final step is to perform a simple test to see if the grey scale value is greater than 100, if it is then the pixel is said to be white (255) otherwise the pixel is taken to be black (0). A threshold of 100 was chosen to be the threshold as this was the value used within the Brighton map.

1	2	3
8	0	4
7	6	5

Table 8.1: 3x3 Matrix cells labelling

8.2.2 Dilation and Thinning

Using a combination of dilation and thinning algorithms, it is possible to smooth out the results of an image, removing small errors on the edges of the objects within the image. An object's edge can be defined by using a 3 x 3 matrix (see Table 8.1) with the centre of the matrix "0" on a black (0) pixel, if three or more adjacent pixels are found to be white (255), then pixel "0" is said to lie on the object's edge.

Both dilation and thinning algorithms have much in common, both algorithms affect the edges of objects within the image, while dilation increases the size of the object, thinning will make the object decrease in size.

Both dilation and thinning algorithms have been implemented using a 3 x 3 matrix which moves around the image, by a scanline method, from the bottom left to top right. If an edge is found, then it records its initial position and follows the edge of the object, recording all pixels it passes through, this continues until it reaches the edge of the map or returns to its initial position. When it finds the end of that specific edge, it carries on moving through the map by the scan line method, searching for new edges. It ignores all edges that have been marked, to prevent edges being thinned multiple times within a single iteration.

8.2.3 Text removal

As text removal is beyond the scope of this thesis, all text that was found within an image was manually removed using Photoshop before the image was used.

8.3 Procedural Road Extraction

Previous road extraction algorithms worked in two stages; first they segmented the road network from the image. Next, an algorithm (usually a thinning algorithm) is used to search for a skeleton of the road network. The novel road extraction algorithm presented here works differently from previous algorithms, in that it does not use a thinning algorithm to search for the skeleton.

The novel road extraction algorithm can be broken down into three main sections as shown in Figure A.3. The three sections are: "Circle size checking and angle calculation", "Junction calculation and hierarchy storage and usage" and "Circle movement and data storage". The algorithm works sequentially, but could be used in a multi-threading manner if there were multiple start points and they used the same data tree.

8.3.1 Road Network Data Structure

Because the road extraction happens sequentially, there is a possibility that a section of the road that has been searched through could be found again. To prevent this, there needs to be a data structure that can store all the sections of roads that have been found, the most efficient way of storing this data is to use a rooted tree of linked lists. This type of tree is used because it allows for the easy insertion, deletion and rearrangement of any of the nodes. Each node within the tree represents a single junction within the road network. The data stored at each node is the position of their parent node as well as data relating to that node, such as the number of paths from that node.

Using this structure allows the algorithm to efficiently manage the memory used and speeds up the insertion of road sections. This approach makes the algorithm run faster than if structures such as static arrays were used, as these would need resizing

each time a new road section was added.

The data structure actually consists of a set of four lists each designed for their own data handling, these are as follows:

- Junction Cartesian coordinates.
- "Junction paths to take" in the form (0, 90, 180, 270) where each value represents a legitimate path direction in degrees.
- "Junction paths taken or not", this holds a list of equal size to the "Junction paths to take" but instead holds data in binary form where "0" represents a path that can be taken while "1" means the path has already been taken.
- Junction parent node, this is to search for a parent node when the child node has no other paths that can be taken.

As linked lists are defined in the standard library within C++, there is no need to rewrite these base functions (push, pop, insert, etc.)

By combining the lists "Junction paths to take" and "Junction paths taken or not", all the data needed to prevent the circle from entering a route that has already been taken are stored.

8.3.2 Circle Dynamic Size

The "Circle size checking and angle calculation" part of the algorithm, as illustrated by the light blue area in the flow chart shown in Appendix A.3, handles the dynamic size of the circle as well as predicting the next possible position to move to.

First, it must determine the number of collision sites on the perimeter of the circle (see red part of the circle on Figure 8.1). Each set of unique collision sites has a set of pixels that make up that site; these pixel locations are stored within a two-dimensional

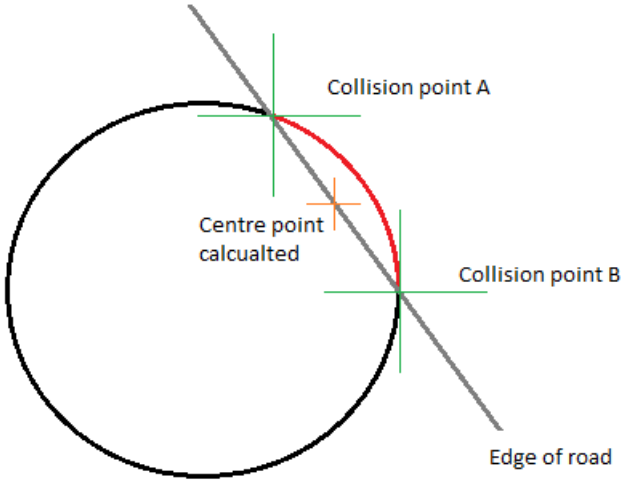


Figure 8.1: Diagram showing how the centre point is calculated using the minimum and maximum points

list. To determine the centre of each collision site the list that relates to that collision site is searched through, to find the maximum (X, Y) and minimum (X, Y) positions. Knowing this, it is possible to triangulate the centre point, by calculating the half waypoint between the minimum and maximum X and then to do the same with their Y components (see Figure 8.1). Doing this will give a centre point which can be used for calculating if the collision is unique, a unique collision site is when:

1. The angle between each unique collision site is greater than 20 degrees.

2. Each unique collision site has a normalised size value greater than 0.1.

Another rule is that if there are more than two collision sites, then the sites that are the closest are merged until there are at most two remaining collision sites. Using the two collision sites, it is possible to calculate whether the circle should grow or shrink, this is illustrated in the Algorithm 1. As shown, the circle can only change its size by one pixel, this is to prevent the size of the circle from changing dramatically

within a single iteration, which is possible if this limitation is not in place. Using a one pixel limit is appropriate for processing sections of maps that are 256 x 256 pixels, because the roads being processed would also be limited in there width. After calculating the width of the circle, it has to calculate in which direction to move next. There are two ways of calculating this direction, if there are less than two collisions then the "Circle sight" algorithm is called as shown in Section §8.3.4, otherwise the direction is calculated using the "Direct calculation" algorithm explained in Section §8.3.3.

Algorithm 1 Calculate $CircleWidth$

```
1: if NumOfUniqCircleEdgeCollisions ≥ 1 then
2:    CircleWidth = CircleWidth − 1
3: else if NumOfUniqCircleEdgeCollisions ≤ 1 then
4:    CircleWidth = CircleWidth + 1
5: end if
6: return CircleWidth
```

8.3.3 Direct calculation

As the circle moves round, ideally the circle would be in constant contact with two unique collision sites. If this is true, it is possible to calculate the resultant direction by using the collision sites and the centre of the circle. The resulting direction is halfway between the two unique collision sites, by default the smaller of the two angles is chosen to be the resultant direction initially, as shown in Figure 8.2. After the resultant direction is calculated, it is checked to see if it is pointing towards an area it came from, this is explained in Section §8.4. When a direction is found to be facing the wrong way, the larger of the two angles is used to calculate the new resultant direction.

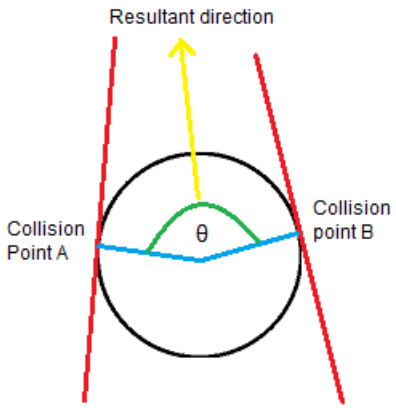


Figure 8.2: Diagram showing how the circle moves when in contact with two unique edges.

8.3.4 Circle Sight

When the circle moves round the road network, there is a chance that the circle is in contact with a single contact site or no contact site at all. This is caused by the circle being too small to touch both edges of the road at once. If this situation occurs the "Circle sight" algorithm (see Algorithm 2) is called to search for the two unique collision sites.

Algorithm 2 Calculate $NumberOfUniqueCollisions$

```
1: while NumberOfUniqueCollisions ≤ 2 do
2:     SearchingRadius + 1
3:     check perimeter for number of unique collisions
4: end while
```

As shown in the pseudo code, the algorithm only stops when more than one unique collision site is found. If more than two collision sites are found, the angular gaps between each of the unique collision sites midpoints are calculated. The two sites with the smallest angular gap between them are merged and a new midpoint is calculated. This process is repeated recursively until only two unique collisions sites remain.

To calculate the exact direction, two factors are taken into account, the size of

each collision site and the location of its midpoint. Using the two midpoints and the centre of the circle, it is possible to calculate the resultant direction as shown in Figure 8.3. Once the maximum and minimum angles are calculated, the sizes of the collision sites are used to calculate the exact resultant *direction*.

The size of the unique collision is important because having a large site means that more of the *SearchingRadius* pixels overlap non-road areas, while a small size means that it is further away from the centre of the circle than the larger site. Using the normalised values of the two unique sites, it is possible to calculate the exact resultant direction which must lie between the maximum and minimum already calculated, this is shown in the Algorithm 3.

Once the exact resultant direction is found, it is checked to see if it faces the correct direction, this is explained further in Section §8.4, if the direction is found to be facing the wrong direction then the maximum and minimum values are swapped and the calculation of the exact resultant angle is repeated.

Algorithm 3 Calculate the $finalDirection$

1: $maximum$ belongs to $collisionSite0$, $minimum$ belongs to $collisionSite1$
2: normalise $collisionSite0$ and $collisionSite1$ so that we can find the actual direction, the circle should move towards.
3: **if** $collisionSite0$ *leq* $collisionSite1$ **then**
4: $finalDirection = maximum - collisionSite0$ / ($maximum$-$minimum$)
5: **else**
6: $finalDirection = minimum + collisionSite1$ / ($maximum$-$minimum$)
7: **end if**

8.4 Check backtracking

It is necessary to check whether a resultant direction is legitimate because both algorithms for calculating the resultant direction can produce two different directions in each case. To check whether the resultant direction faces towards the area that the circle came from, Algorithm 4 is used.

Algorithm 4 Calculate if *movingBackwards*

1: **for** $i = 0$ to 7 **do**
2: itt = NumberOfIterations-i
3: **if** $tempX[itt] \leq minX$ **then**
4: $minX = tempX[itt]$
5: **else if** $tempX[itt] \geq maxX$ **then**
6: $maxX = tempX[itt]$
7: **else if** $tempY[itt] \leq minY$ **then**
8: $minY = tempY[itt]$
9: **else if** $tempY[itt] \geq maxY$ **then**
10: $maxY = tempY[itt]$
11: **end if**
12: **end for**
13: $calculatedX = minX + ((maxX - minX)/2)$
14: $calculatedY = minY + ((maxY - minY)/2)$
15: $deltaX = calculatedX - currentX$
16: $deltaY = calculatedY - currentY$
17: $deltaX2 = calculatedX - futureX$
18: $deltaY2 = calculatedY - futureY$
19: $deltaX3 = futureX - currentX$
20: $deltaY3 = futureY - currentY$
21: $tempDeltaSumH = \sqrt{deltaX1 * deltaX1 + deltaY1 * deltaY1}$
22: $tempDeltaSumO = \sqrt{deltaX2 * deltaX2 + deltaY2 * deltaY2}$
23: $tempDeltaSumA = \sqrt{deltaX3 * deltaX3 + deltaY3 * deltaY3}$
24: $top = (tempDeltaSumA^2) + (tempDeltaSumH^2)$ - $(tempDeltaSumO^2)$
25: $bottom = 2 * tempDeltaSumA * tempDeltaSumH$
26: $theta = \arccos(top/bottom) * 180/PI$
27: **if** $theta \leq 60$ **then**
28: $movingBackwards$=true
29: **end if**
30: **if** $movingBackwards$ **then**
31: run "Direct calculation" or "Circle sight" again depending on which one gave input
32: **end if**

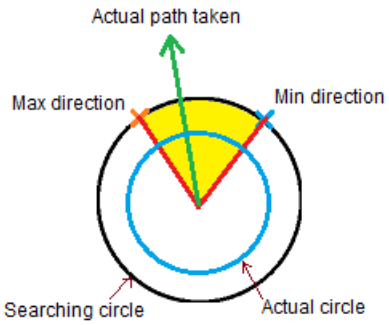


Figure 8.3: Diagram showing how the circle picks next direction from circle sight.

The algorithm can be broken down into three stages. First, it calculates the maximum and minimum (X, Y) coordinates, using these coordinates it is possible to get the midpoint as shown in lines (13-14). The second step is to use the COSINE rule to calculate the angular difference between the newly calculated midpoint and the predicted direction which has been converted into a set of (X, Y) coordinates. Finally, if the calculated angle is less than 60 degrees, then the direction is classed as backtracking. When a direction is found to be backtracking, it calls the algorithm which sent the input data, either "Direct calculation" §8.3.3 or "Circle sight" §8.3.4. Because there are only two possible directions, and the first is found to be backtracking, the next direction calculated from the algorithm is forced to be correct, so it skips the backtracking step once it has calculated the direction.

8.4.1 Junction Detection

To search for a junction during each iteration, rays are traced at 5 degree increments around the centre of the circle, the rays only stop once they reach the edge of a road or their maximum length which is twice the radius of the circle. Each adjacent ray which successfully traces to its maximum length stores the position of that point; a

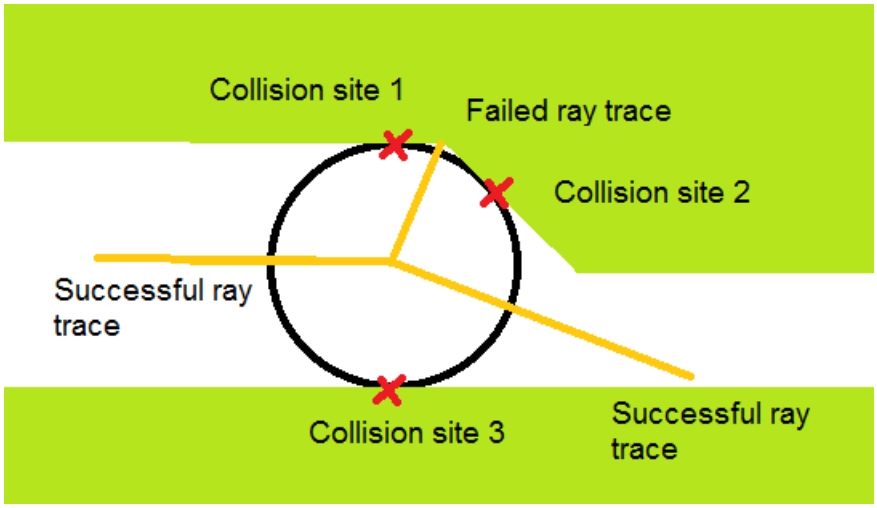


Figure 8.4: Illustration of a situation where verification of path legitimacy is needed.

new list is created once a ray is found that does not trace its maximum length. Doing this creates a set of lists that holds the maximum coordinates (X, Y) of each ray that traced its maximum length, each set of coordinates are used to triangulate the centre point of that set, in a similar way to how the centre point is calculated in Section §**??**. A ray with a length three times the size of the radius is passed through this point to verify that a path is legitimate. This is needed because, as shown in Figure 8.4, it is possible for a junction to be detected even when there clearly isn't one.

If a junction is found, this is entered into the junction list, with the paths visible from that point and the path taken to get to that junction, this is to prevent the circle from taking the same path twice.

8.4.2 Large Area Detection

Large areas can throw off the circle expansion code which is designed for moving along roads and meeting junctions. A large area is defined as a section of the road network which suddenly grows three times its width over six iterations of the algorithm. When this situation is triggered, the algorithm reacts by tracing rays from the centre of the

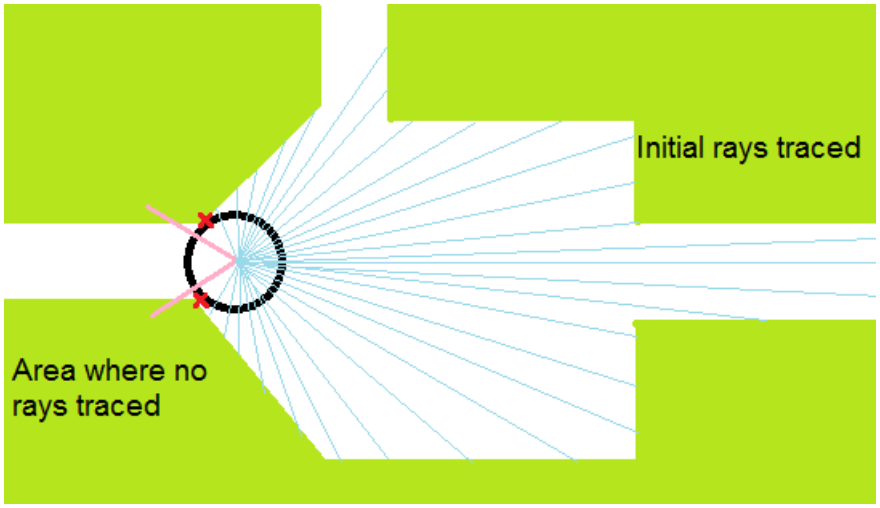


Figure 8.5: Figure showing how the initial rays are traced; the pink lines mark the area where no rays are traced.

circle to the edges of the roads at one degree increments; to prevent the circle from tracing rays to positions it has already been to, they are excluded from an area defined by a 60 degree cone, so a total of 300 rays are traced. Using these preliminary rays it is possible to calculate the centre of the large area approximation from the view of the entrance. This is shown in Figure 8.5.

When this preliminary centre point is found, a second set of rays are traced as can be seen in Figure 8.6, this time using the entrance point of the large area to define the direction of the cone. To prevent rays being traced in that direction, the length of the rays are recorded and checked against their neighbouring rays to check for any sudden changes in their lengths, as this indicates an exit. Once all the exits are found, any errors that may have been produced are corrected by calculating the angle between the rays, if the angles are too close together the exit points are merged together forming a new exit point. These exit points are entered into the data tree, but they only contain a single path which can be used. The predetermined path which the circle should take is calculated by expanding the exit point until two unique collision

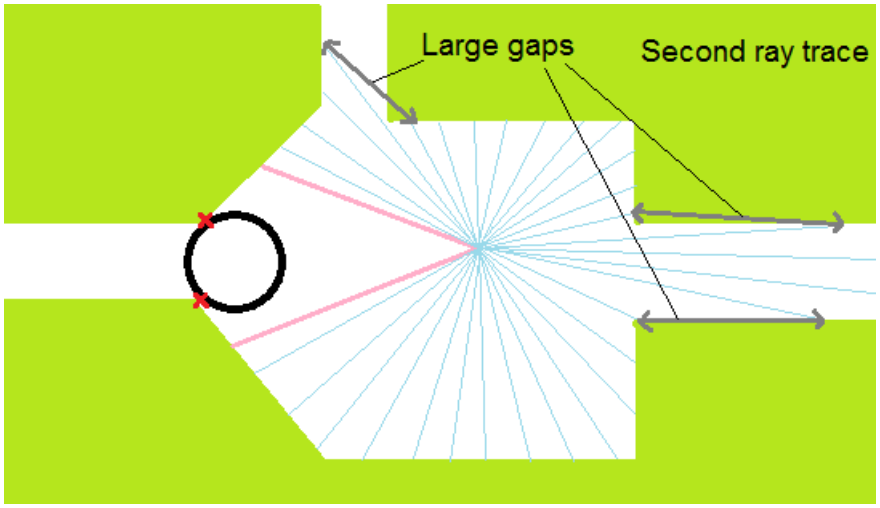


Figure 8.6: Diagram showing how the second set of rays are traced and how the large gaps relate to the exits from the junction. The pink area shows where rays are not traced.

sites are found, then the smallest non collision site is the path which should be taken.

Large areas can also be dead ends, such as cul-de-sacs. To take this into account, if a large area has no other path it can take besides the one that entered the area in question, then it is declared as a dead end and marked accordingly in the data tree.

8.4.3 Dead End Handling

Much like junctions, dead ends have an algorithm to handle this special case. A dead end is defined as a point within the path where there is only a single contact point on the circle and the circle has expanded so that it is touching both opposite edges of the path edge walls. Only when these circumstances are met is a dead end declared.

Once a dead end is declared, the current (X, Y) points of the circle become the dead end junction, such a junction is denoted within the data structure as having junction paths (0, 0, 0), and junction paths available as (1, 1, 1).

After a dead end is found, the algorithm searches for other nodes within the data

StartOfLine
29 30 4
30 31 5
33 31 4
36 31 4
StartOfLine
48 31 4
51 31 4
54 31 4
57 31 4

Table 8.2: Output saved data

tree which have a path that has not been taken yet. If a node with a path cannot be found, then the algorithm terminates, as the whole network is found, this is the legitimate way for the program to terminate.

8.4.4 Exporting Coordinates

As each road length section is calculated, it is assigned a unique ID representing its position within a two dimensional array which holds all points of all paths within the road network, a separate array is used to store the "heads" and "tails" denoting which path joins with which path. The points are then saved into a special format which can be read by the Max Script code, the format allows the saving of multiple 2D splines. A sample of the format is shown in Table 8.2.

The term "StartOfLine" is placed at both the start and termination point of a path, so when the Max Script scans through the .txt file and comes across such a term, it initiates the creation of a line, when it comes across the next "StartOfLine" it knows it should stop adding any more points to the spline and instead create a new spline. The three values represent the X, Y, W values for the road to be represented in 3ds Max, where X and Y are the Cartesian coordinates within 3ds Max and the W represents the width of the road at that point. Because of how the Max Script works

there are two copies of this .txt, with one of them omitting the "StartOfLine" when a new line is created, as only one of these files need to declare where a spline should start and end.

8.5 Max Script Importer

To recreate the roads in 3ds Max, a script was written that can import the data produced by the C++ program. The script works by firstly setting up pointers to start reading from both text files, the main text file being the one with the "StartOfLine" commands present, this text file is used to position the roads. The second text file is used for calculating the widths of the roads. Algorithm 5 explains briefly how the Max Script works. There are two loops one for creating the number of road sections, the second is for going through each road section calculating the width of the roads.

Algorithm 5 Max Script pseudo to calculate road position and widths

```
1: pointerA = positionDataFile(begin)
2: pointerB = widthDataFile(begin)
3: faceA = 2
4: faceB = 4
5: while pointerA ≠ positionDataFile(end) do
6:    if pointerA = "StartOfLine" then
7:       faceA = 2
8:       faceB = 4
9:       Create new Spline
10:   else
11:      Read X, Y
12:      Add X, Y to spline
13:      pointerA move next line
14:      Select faceA and faceB
15:      Move faceA and faceB by amount pointerB in the direction of their
         normals
16:      pointerB move next line
17:      faceA+ = 2
18:      faceB+ = 2
19:   end if
20: end while
```

Chapter 9

Results for the Road Extraction Algorithm

9.1 Procedural Road Extraction Algorithm

To test the road extraction algorithm a combination of synthetic and real-world inputs where used. By using these test cases it is possible to find the strengths and weaknesses of the proposed algorithm. There are three main aspects to the algorithm that need to be tested: circle movement, circle growth and junction handling. The Cunningham map is not used to test this algorithm because, the Cunningham map is not from a top down view and there are many obstructing objects that cause breaks within the road, this is why the Koblenz and Brighton maps are used.

9.1.1 Circle Movement

To test for circle movement, all possible directions in which the circle can move were taken into account, this can account for all roads that have a linear width, but having a road that changes its width when it makes a turn can cause the circle to deviate from the centre of the road as shown in Figure 9.1. Although these deviations initially cause the road to appear to be incorrectly traced, when the road is recreated in 3ds Max it is able to reduce this effect by the sampling frequency it uses, as the max script only uses every fifth point that the C++ program outputs. Different values

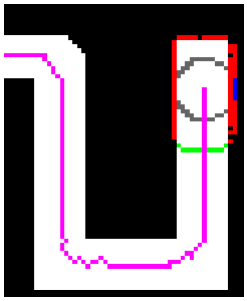


Figure 9.1: This image shows how the circle can move away from the true centre of the road network in some cases.

were tested and for the optimal results, the fifth point was chosen be the parameter because it gave the best results, by smoothing out noise within the road and still keeping enough detail about the path of the road. Although this value is set as a default, the user could modify.

Having many functions to move the circle means that all types of situation are covered and that there are no situations where the circle cannot move, as it will always move towards an empty space and away from where it came from.

To show that the circle can travel around a road network with turns in the road, a test image was used to force the circle to move in 90 degree increments so as to test the tracking ability of the circle as it moves through the image, the results are shown in Figure 9.2

As the circle will not always be moving and turning in 90 degree increments, another test map was used to test the function to see how it would perform when the circle approaches a turn less than 90 degrees (see Figure A.4) and greater than 90 degrees (see Figure A.5).

Another common problem with input data is the clarity of the image. If the image has blurred edges, this can cause the edges of roads to appear very irregular, the worst case being when a thin road edge bordering two sections of road is completely lost.

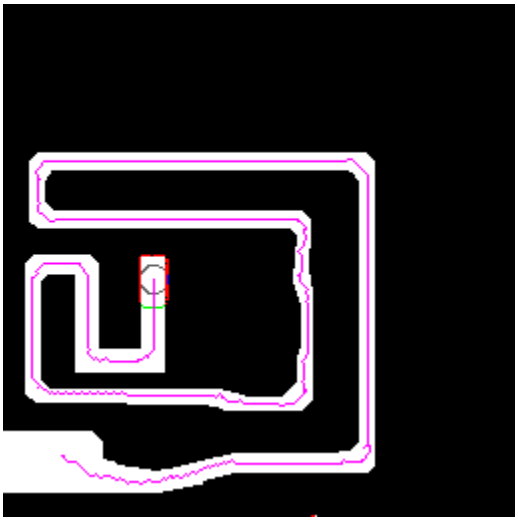


Figure 9.2: Image illustrating how the circle tracks around a road network with 90 degree turns within it.

Although the problem can be reduced by using a threshold when converting the image from colour (RGB) into a binary image, if any noise gets through it will affect the straightness of the road traced as shown in Figure 9.3(b). Figure 9.3(a) shows the original image where there is a layer of 1 pixel thick noise around the edge of the road. Because of the small size of the road with an average width of 10 pixels having 2 pixels worth of noise, this means that the road width can only be accurate to plus or minus 20%, which is a substantial amount of the road width.

9.1.2 Circle Growth

While this is a small part of the entire algorithm, this part keeps a record of the current width of the road, which can be used by max script in the creation of three-dimensional roads. As explained in the design and implementation chapters, the width of the circle expands and contracts depending on the number of unique contact points of the circle and depending on whether the circle perimeter overlaps the edges of the road network.

To test the circle to see how the circle changes size, a test input with a single road

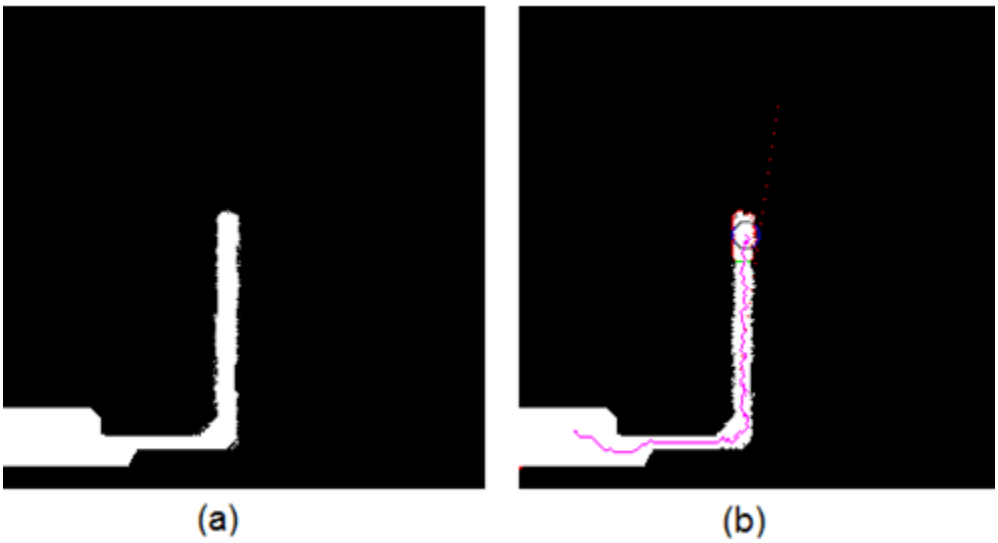


Figure 9.3: (a) Shows the input image for the algorithm. (b) The pink line shows the path of the circle and that it managed to extract the road successfully.

of varying road widths was used, the initial size of the circle is shown in Figure 9.4(a), the contraction stage in Figure 9.4(b) and finally the expansion stage in Figure 9.4(c).

9.1.3 Junction Handling

This part of the algorithm is the most important because missing a junction can mean that entire sections of the map can be missed or traced over twice or more. To minimize the chance of tracing over road sections twice, there are many dedicated functions such as the back tracking function which prevents the circle moving back on itself. To search for junctions, the circle uses the circle sight technique as explained in the implementation chapter. When all these techniques come together, they form the junction handling part of the algorithm. As shown in Figure A.6 a junction has been detected with three exits, not all junctions are detected as clean as this one, this can be caused by factors such as the existence of numerous road entrances at odd angles (see Figure A.7) or different road widths (see Figure A.8).

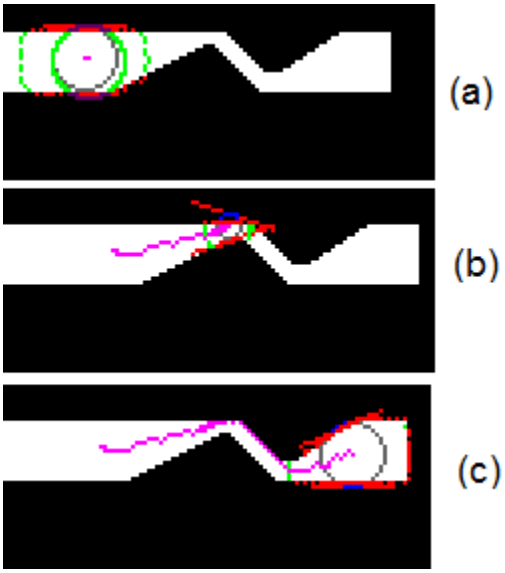


Figure 9.4: (a) Shows the original starting width of the circle. (b) Shows the circle contracting as it moves through a small gap within the road network. (c) Shows the circle expanding as the road width increases, note how the pink line remains close to the centre of the road network.

Sometimes, the algorithm will miss a junction for one of two reasons:

- The connecting road is much smaller in width compared with the current road.
- The connecting road is of insufficient length.

This can be seen clearly in Figure 9.5. In cases such as these, the user would have to either reposition the circle to start off where it missed off or create the section of road in 3ds Max after the virtual model has been created.

Another main part of the junction handling section is the ability to detect and handle large areas, as shown in Figure A.12. It can be seen that the algorithm can detect these large areas then mark them accordingly. As these will leave large gaps within the road network, the user would have to fill in these gaps once the roads have been reconstructed in 3ds Max.

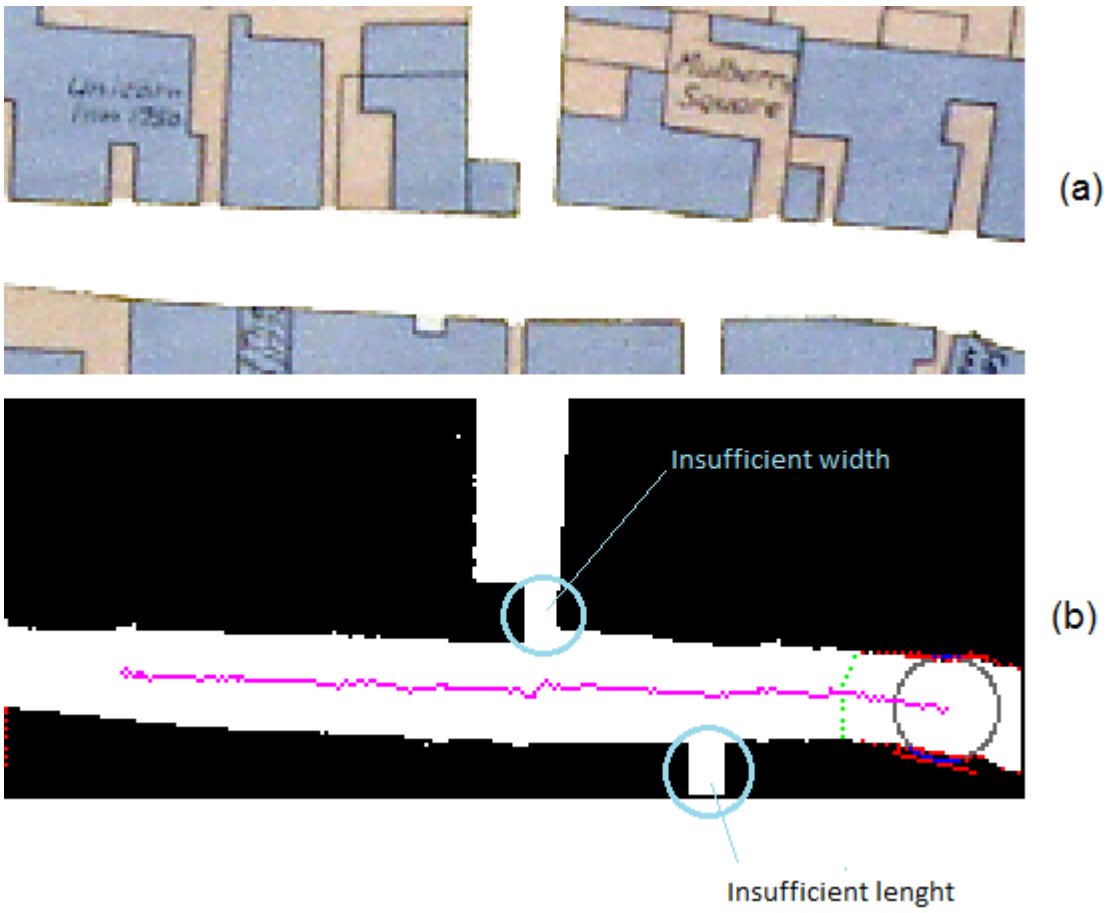


Figure 9.5: (a) Illustrates the original input for the algorithm. (b) Shows the algorithm can miss junctions if they are substantially smaller than the current road width or of insufficient length.

9.1.4 Synthetic Input Examples

Figure A.9 shows how the circle traces the paths around a map with many junctions, which do not interconnect.

Figure A.10 shows how the circle reacts when it comes into contact with a junction that it has already encountered.

With an image such as shown in Figure A.11 a conventional technique such as those proposed by G. Vosselman and J. Knecht [VK95] or J. Zhou et al. [ZBC05] would include all the thin single pixel lines as part of the road network but, as the figure shows, the proposed algorithm is only slightly affected by all the noise within this image.

9.1.5 Koblenz Map and Brighton Map as Proof of Concept

As a proof of concept, the algorithm was used on two maps, the Koblenz map which started off as a black and white image and the Brighton map which is in colour.

To input the data into the program correctly, as the map is at a much larger resolution than the program can accept, the image is subdivided into sections where a section of road lies at the starting position of the circle. These sections of road networks once found are superimposed onto the final image, pink lines represent the path followed by the circle, while a blue line represents a section of road that the algorithm has missed. Gaps exist between sections of the pink traced lines because of cutting up the map into sections.

9.1.6 Koblenz Map Road Extraction

As shown in Figure A.13, the areas that were processed by the algorithm are all shown in pink. The input data for the program was created based on this map. Because this map has houses in the same colour as the road, the algorithm is unable to use

segmentation techniques to extract the road only from this map, which is why each section of the map was individually coloured in using a paint bucket tool and paint brush tool in Photoshop to make the image compatible with the algorithm, a typical input image is shown in Figure A.14. Although filling in the houses manually is not an optimal technique for the Brighton map because it is in colour, the algorithm is able to automatically segment the road from the background.

Using this algorithm, the majority of the roads where traced, although four roads were missed towards the top of the Koblenz map with blue lines showing where the roads should be. These problems can be identified by the positions of the junctions and the width of the entrance of the roads at the junctions, this is shown in Figure 9.6. These errors occur because of the inherent nature of the algorithm, which ignore roads that are much smaller in width this is shown in Figure 9.6(a) and (b). Figure 9.6(c) shows a unique case where the circle enters a much large junction than the width of the road causing the circle to expand and therefore miss calculate the "Cross junction" as a single section of road. Errors like these would have to be corrected by the user after running max script.

9.1.7 Brighton Map Road Extraction

The Brighton map (see Figure A.16) was taken using a digital camera, this is evident by looking at the image as the brightness of the image changes from bottom to top, also there is more noise within the image in the form of creases within the paper map. As the image is coloured and segmentation can extract the road from the background, the raw image of the map can be used as input for the algorithm, as shown in Figure A.17. Within the Brighton map, it can be seen that there are two paths that are missed by the algorithm marked by the two blue lines, towards the top of the image. This case is used as an example of junction handling.

Figure 9.6: This figure shows three cases where the algorithm failed within the map. (a) and (b) Illustrates a junction where the entrance is too small. (c) Shows a "Cross" junction where neither roads leaving the current path is detected.

To compare the manual modelling methods to the automatic extraction method there are two things to take into account:

- Speed (Time taken to create the virtual road model).
- Accuracy (How many roads are accurately traced and number of errors within the roads).

Using the Brighton map as the test case, it took over two hours to accurately model and texture the road using manual modelling methods. Using the automatic extraction method it is possible to cut down the time taken to develop the model as the sections that are extracted by the algorithm can be easily used within the final virtual model with little editing. This is shown in Table 9.1; the map names correspond to Figure A.15, where each individual part of the image is numbered. Each section was used as an input for the road extraction algorithm sequentially.

The resultant time taken to construct the whole map was 240 seconds. The reason for such a long execution time, is because of the 'cout' and 'println' statements within the code which, prints between 20 to 100 lines text for each pixel of the road extracted.

Map name	run 1	run 2	run 3	average run time
Brighton 1	13	12	13	13 sec
Brighton 2	13	13	15	14 sec
Brighton 3	3	3	2	3 sec
Brighton 4	8	8	8	8 sec
Brighton 5	8	9	9	9 sec
Brighton 6	8	9	8	8 sec
Brighton 7	5	5	4	5 sec
Brighton 8	3	4	3	3 sec

Table 9.1: Table showing time taken to extract each section of map.

This is useful when debugging and finding out more specific information about the roads extracted, for example lengths of each road.

Once the roads have been extracted using the proposed algorithm it is possible to use the max script, as described in Section §8.5. Using this combination it is possible to create the entire road network, as shown in Figure 9.7. This complete process takes 20 minutes from the divisions of the original map to having the complete three-dimensional model. Most of the time is spent dividing the map into appreciate sections so that the algorithm can read it and correcting the errors produced by the max script by overlapping polygons; this is explained in more detail in Section §9.2.

9.2 Max Script output

Using the max script proposed in Section §8.5, it is possible to import the two .txt files from the C++ program and these text files can be used to reconstruct the road network in three-dimensions. The roads are reconstructed in three-dimensions to allow for future development, such as kerb sides and other features of the roads to be added easily. Figure 9.9 shows how the roads appear when rendered using Mental Ray in 3ds Max. Figure 9.8 shows how the roads are extracted if no width modifiers are applied to the road sections. When a width modifier is applied, small errors can occur when polygons overlap, as shown in figure 9.9.

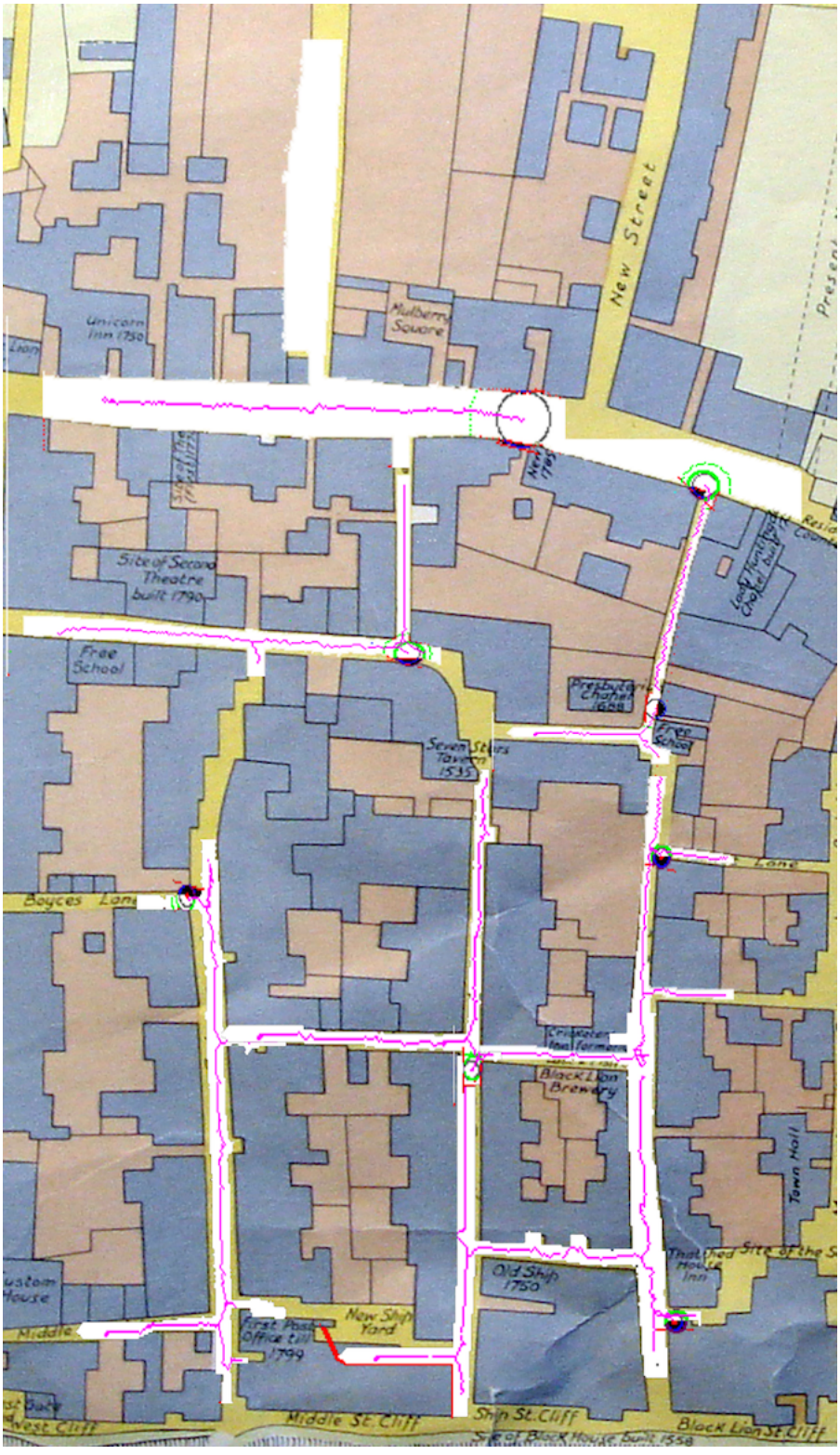


Figure 9.7: This screen shot within 3ds max, shows the road network once the max script has finished running. Pink lines indicate where the centre of the roads are.

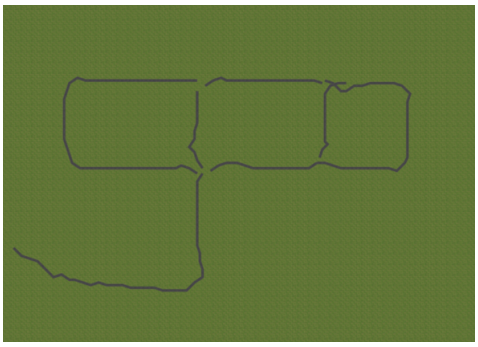


Figure 9.8: Illustrates how the road network looks before calculating road widths.

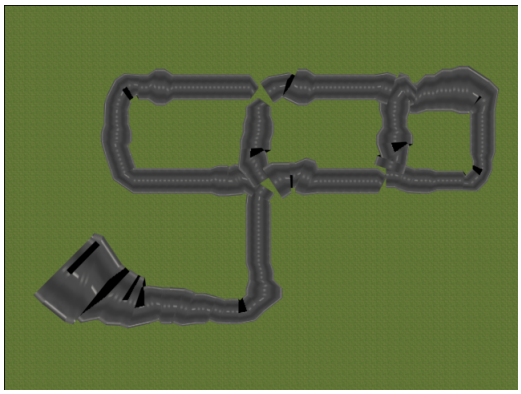


Figure 9.9: Illustrates how the road network looks after adding in real road widths, the black areas show overlapping polygons.

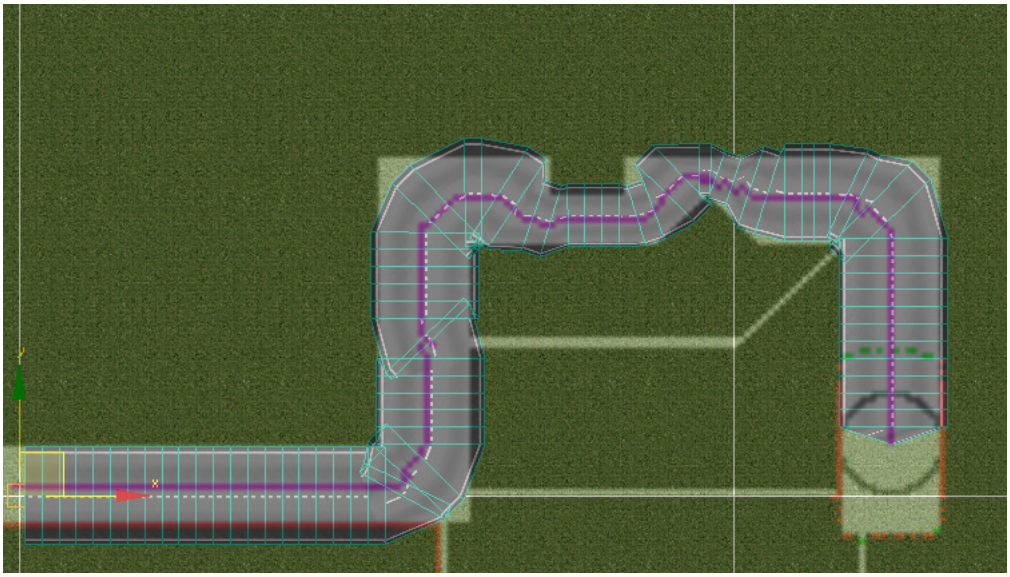


Figure 9.10: This image shows how the 3ds Max virtual road model appears when superimposed on top of the original input image.

To show how effective the algorithm is, the output data from Figure 7.1 was used to generate a three-dimensional road network, the result can be seen in Figure 9.10 where the virtual model of the road network is superimposed on top of the original input image. Although there are a few geometrical errors with some polygons overlapping, the whole road section remains a single object without any holes within it and the UVW texture maps correctly. The blue lines that lace through the entire road network show the edges of each polygon.

To reconstruct larger maps, they are broken down into smaller sections which are then joined back together by the user within 3ds Max. Letting the user join the sections of roads is important because if the input size of maps have different aspect ratios it can cause the model to become skewed in a one or more axes. A section of the Koblenz map was used to show how max script is able to reconstruct large maps as well as smaller ones, the results for the Koblenz map are shown in Figure A.18 showing how the polygons are used within the model and in Figure A.19 which shows

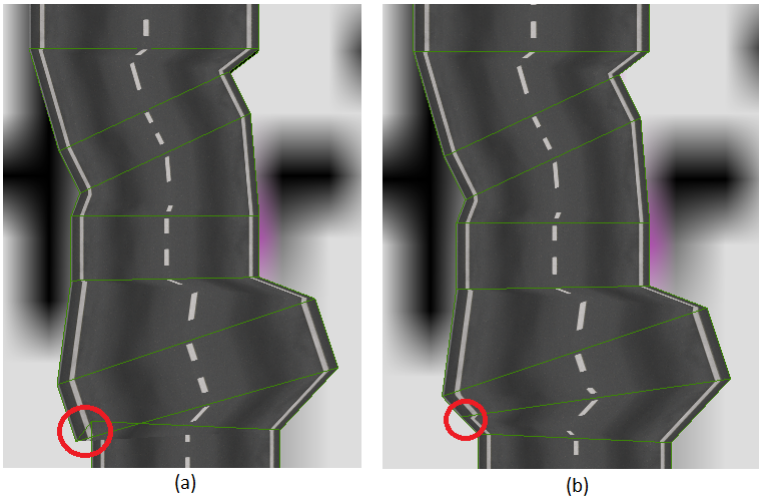


Figure 9.11: (a) This image shows how the polygons overlap after calculating road widths. (b) This image shows how the vertices can be moved to avoid overlapping polygons.

a render of the entire Koblenz map using Mental Ray.

The total polygon count for the Koblenz map is 4830 polygons which consist of 9556 triangles. This is a low number considering that this is for the entire road network built in three-dimensions and such a model could be used within a virtual environment scene with a few changes where polygons overlap, this is a simple change which involves moving a few vertices so that the polygons no longer overlap, as shown in Figure 9.11 (a) where the highlighted area shows where polygons overlap. Figure 9.11 (b) shows how the road looks after two vertices have been moved so that the polygons no longer overlap.

To further show how effective the max script code is at converting the C++ output into three-dimensional models, it was tested on the Brighton map data set, the results for this can be seen in Figure A.20 which again shows where polygons are used and Figure A.21 which shows how the model looks when rendered and superimposed on top of the original map with the roads traced marked on it.

Chapter 10

Conclusion

10.1 Summary

The aim of this study was two-fold as described in Chapter §1, one aim was to recreate the Cunningham and Brighton maps in three-dimensions, this lead onto the development and implementation of the novel road extraction algorithm.

Current applications of virtual cultural heritage sites are explored in Chapter §2, many well-known examples already exist such as Rome Reborn [Vir10b] which was developed by IATH and show-cases Rome as it was in 320 A.D. This chapter also explores the benefits of manual modelling while Chapter §3 discusses procedural modelling techniques and their benefits, such as using mathematical morphology for segmentation and skeletonization.

Chapter §4 goes though the design and justification of the design of both the Cunningham map and Brighton map, because there are many similarities between them, such as both have roads and houses. But there are also many differences between them, for example, the Cunningham map is a map drawn from the perspective of W.Cunningham while the Brighton map was created using a top down view like modern day maps.

The implementation of the Cunningham map is detailed in Chapter §5 which goes into detail about how each aspect of the model that was created, such as roads, houses

and rivers. Because the final model had to have a low polygon count, this chapter also discusses the polygon reductions performed manually on some buildings, such as the Cathedral model. As well as modelling, this chapter also explains how the lighting was set up to represent how the lighting would have appeared back in the time when Cunningham drew the map, as shown in Section §5.7.1.

The implementation of the Brighton map is explained in Chapter §5.5 which describes each aspect of how the model was created. As the final model is a section of the map that is used to compare manual modelling methods and the efficiency of the automatic road extraction algorithm.

Chapter §6 goes though the results of both the Cunningham map and Brighton map models. The final products of the Cunningham map model are a single video lasting 1 minute and 25 seconds and a 3ds Max archive which stores all the data about the scene, for example geometry and textures.

Chapter §7 goes though in detail how the road extraction algorithm was conceived and developed. All the core aspects of the road extraction algorithm are explained such as the circle sizes in.

The implementation of the road extraction algorithm is explained in Chapter §8. The algorithm is split into two parts, pre-processing and the actual run time of the algorithm. During pre-processing, any imperfections in the image are removed, such as converting an image into a binary image from a RGB colour image, this is necessary to reduce the number of variables before running the image though the algorithm to minimize the number of errors. The algorithm can be broken down into three separate sections, expansion, movement and data recording, all of these parts are explained in Section §8.3 and §8.4.

A max script was developed in order to import C++ data to be converted into three-dimensional geometry within a virtual environment; this is explained in Section

§8.5. The script works by reading two files, one used for the positions of roads while the other is used for calculating road widths.

Chapter §9 shows the results from the road extraction algorithm with many test cases including two real-world cases. This Chapter also shows how the data from the road extraction algorithm can be used to recreate the roads in three-dimensons.

10.2 Performance Overview

As shown in Chapter §6 it can be seen that the Cunningham map model was created within the specification specified by C. Rawcliffe. This model has been used by the UMG on one of their websites [UMG10] showing the public how Norwich looked within the past.

The results of the road extraction algorithm show that although the algorithm is restricted in what it can detect because of how the algorithm works, some roads can be missed because the algorithm deems them to be errors, this is shown in Section §9.1.3 where roads are missed because their widths are much smaller than the width of where the circle currently is. Even with this restriction, the proposed algorithm can calculate the widths of roads and, when combined with the max script, can create three-dimensional roads which is not possible using algorithms such as that of G. Vosselman and J. Knecht [VK95] because these do not determine the widths of roads, only the position of the road skeleton.

10.3 Future Work

The development of the Cunningham map went according to specification, any future modifications of the model can be completed by editing the original 3ds Max file and such an edit could be completed by anyone with a copy of 3ds Max. From the research and the results, there are a number of improvements that could be made to the road

extraction algorithm, these are listed below:

- Increasing the robustness of the algorithm so that it never misses a section of road would be the next step. Using a combination of skeletonization techniques that preserve connectivity and the proposed algorithm for calculating road widths, it may be possible to achieve a complete road network which preserves connectivity and gets all road widths as well as ignoring small errors.

- Currently the algorithm is designed to work with images that are 256 x 256 pixels large, this could be changed by letting the user can adjust the parameters of the algorithm. For example, the minimum width of the roads that are accepted as real roads.

- Working through the image moving sequentially from pixel to pixel is time consuming. To speed up the process, the algorithm could be seeded at multiple times, either at random or at predetermined locations. Each seed could then run on a single CPU core, this could be implemented using OpenMP or using CUDA for implementation on an NVIDIA GPU. Each thread would operate independently but use a central storage for data to store centrally the data tree and the paths that each circle has traced over.

10.4 Final Thoughts

This study presents two final products, the model of the Cunningham map which is already being used within the commercial world and a novel approach to extracting roads. The road extraction algorithm has been demonstrated to work within its confined input, when roads do not dramatically change in size. The algorithm proposed is shown to be viable for extracting roads from any map then converting these data into a three-dimensional model within 3ds Max, no current algorithm

provides such an easy way to create a virtual model from an image. In this regard, the objectives of this study have been completed as the results in Chapter §9 show how well the algorithm performs and how it is a viable option for users who wish to extract roads from an image where the road size does not change dramatically.

Chapter 11

Nomenclature

3ds Max: Autodesk 3D studio max.

API: Application program interface.

BRDF: Bidirectional reflectance distribution function.

CPU: Central processing unit.

DSLR: Digital single-lens reflex.

FG: Final gather.

FPS: Frames per second.

GIS: Geographical information system.

GI: Global illumination.

GPS: Global positioning satellite.

IATH: Institute for advanced technology in the humanities.

Maya: Autodesk maya.

RGB: Red, green, blue.

UMG: Urban modelling group.

Appendix A

Appendix

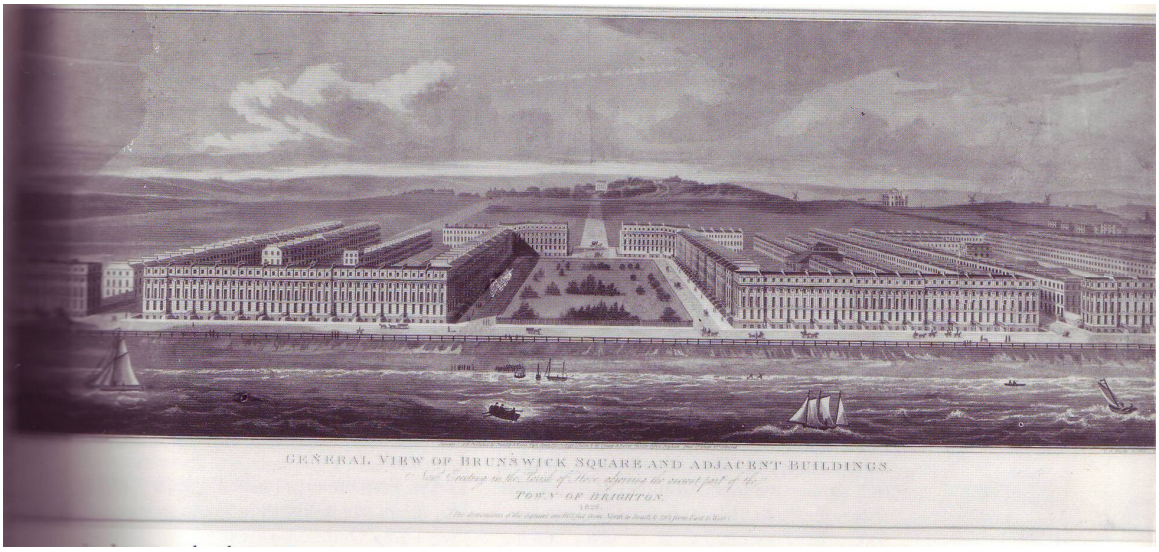


Figure A.1: Image showing Brunswick, a part of Brighton appeared in 1820.

Figure A.2: Image showing how houses appeared in Brunswick square.

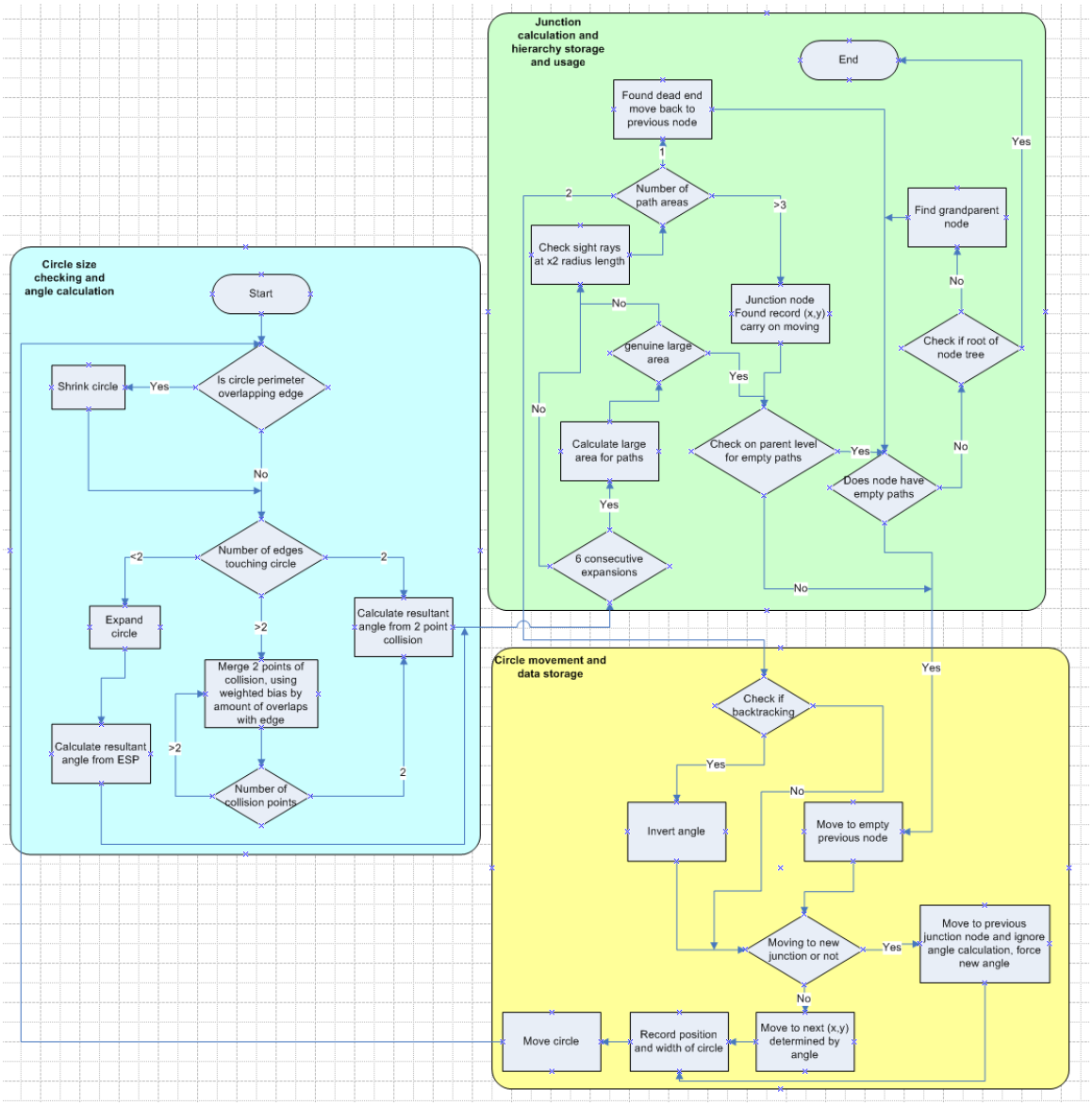


Figure A.3: Diagram showing how the algorithm finds paths.

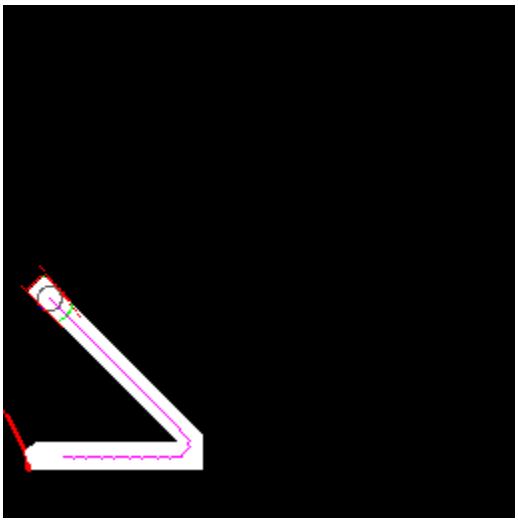


Figure A.4: Image illustrates how the circle performs when there is an acute angle within the road network.

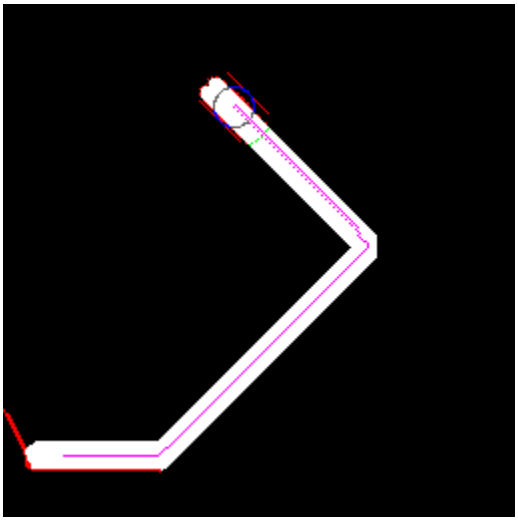


Figure A.5: Image illustrates how the circle performs when there is a reflex angle within the road network.

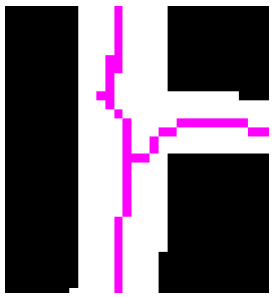


Figure A.6: Illustrates a single “T” junction, where the pink line is the path traced by the algorithm.

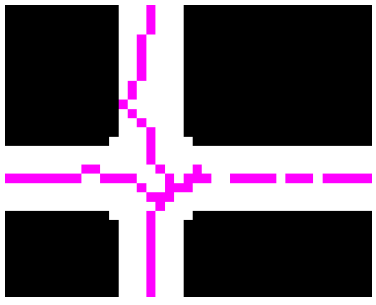


Figure A.7: Shows a “Cross” junction, where the pink line is the path traced by the algorithm.

Figure A.8: Illustrates a more complex junction, where the pink line is the path traced by the algorithm.

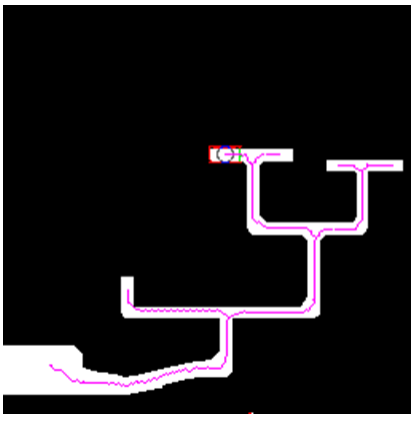


Figure A.9: Shows how the algorithm traces though an image where it needs to go back through its data structure to find previous open paths.

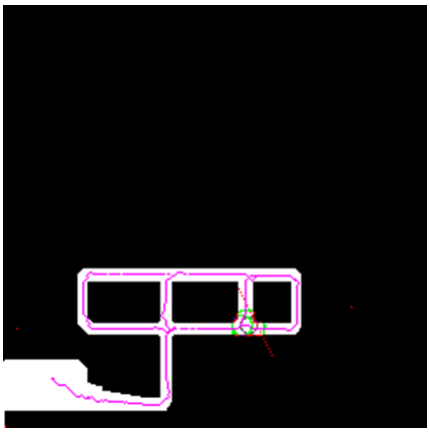


Figure A.10: Illustrates how the algorithm moves through a grid network of roads.

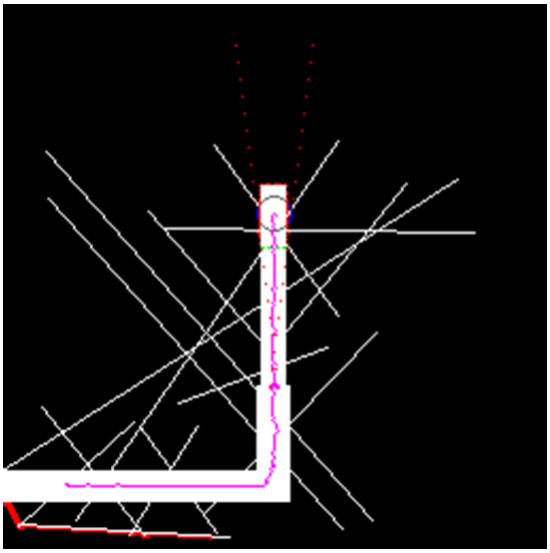


Figure A.11: Illustrates how the proposed algorithm performs where the current road extraction algorithms would fail.

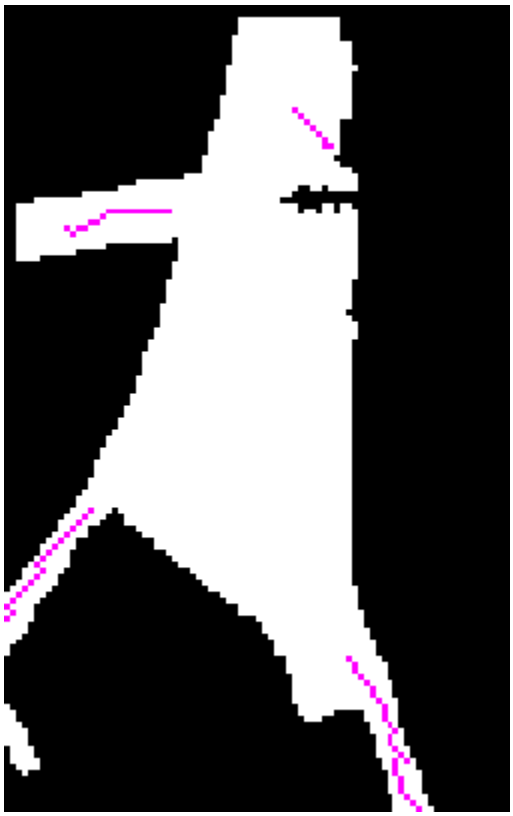


Figure A.12: Image showing how a large area is traced, with the pink paths being found leading away from the large area.

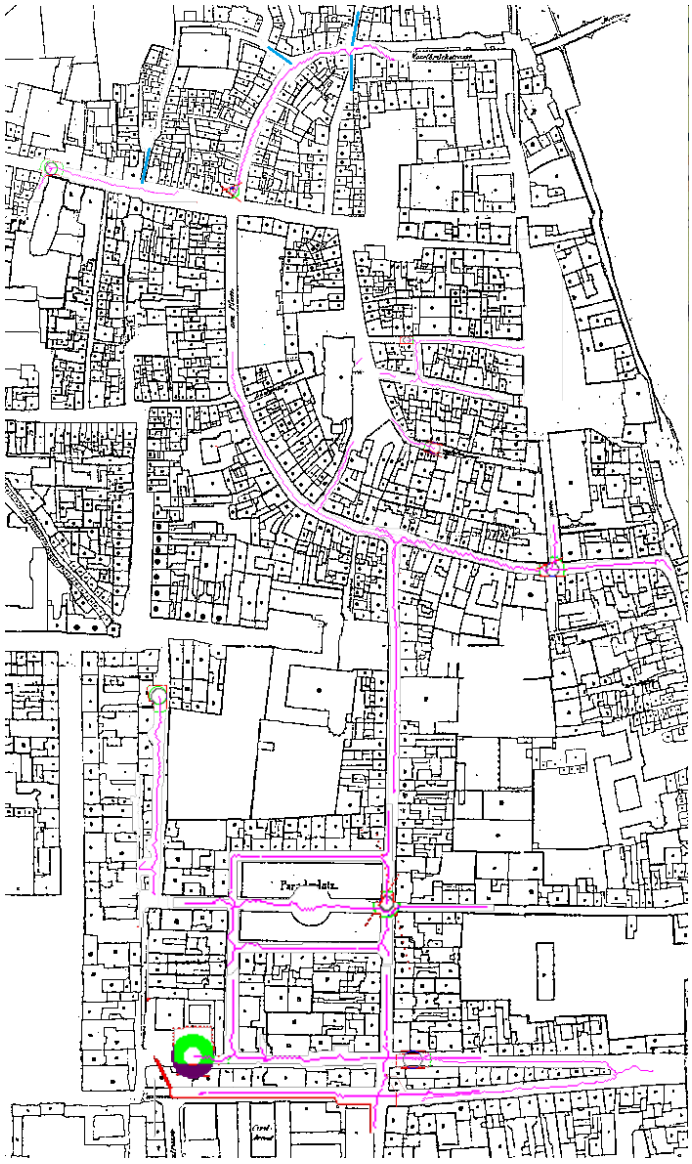


Figure A.13: This image shows how the algorithm performs on the Koblenz map and the paths found.

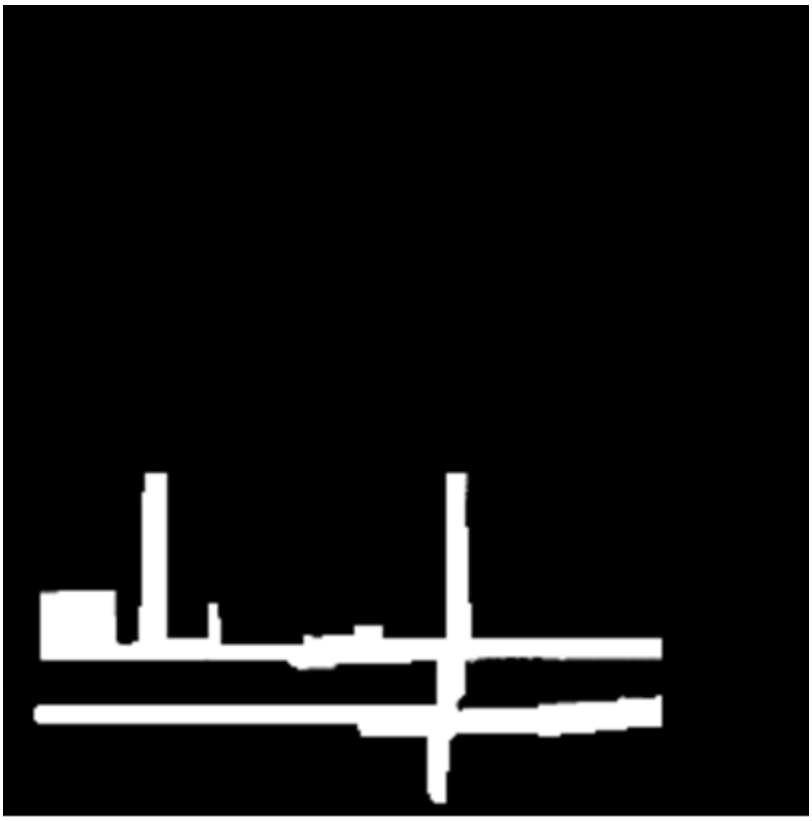


Figure A.14: This illustrates a typical input image for the Koblenz map. As the map contains many imperfections such as houses with the same colour as the road surface, To make the map suitable for the algorithm the houses where blacked out manually within Photoshop.

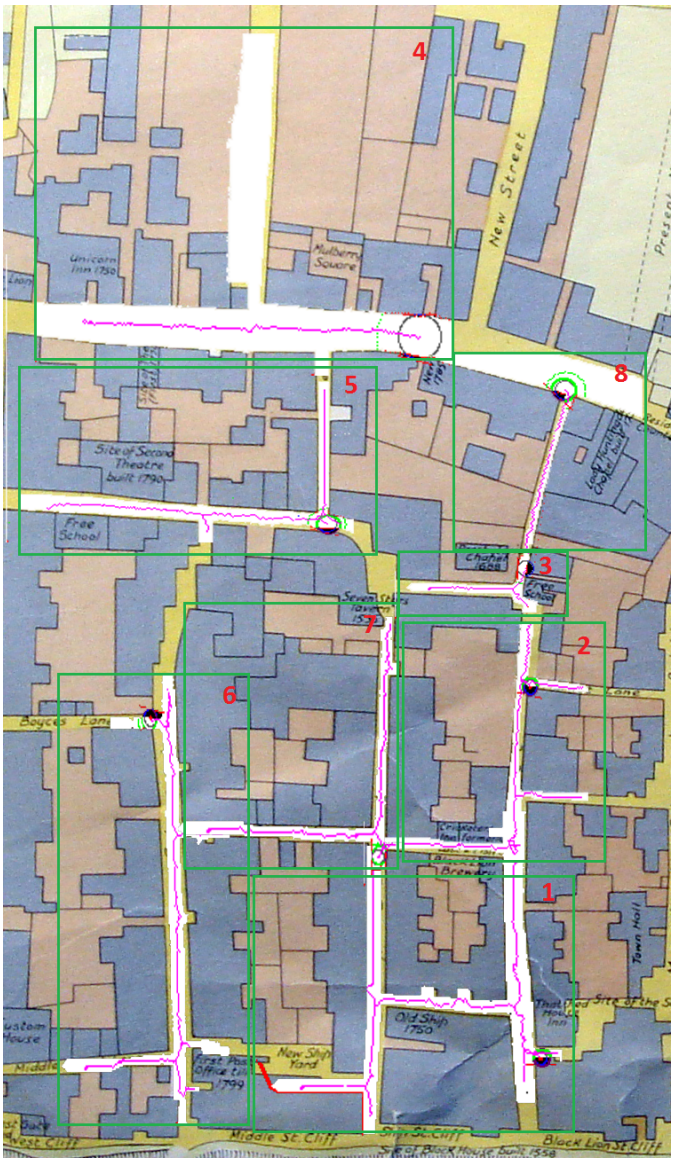


Figure A.15: This image depicts how the Brighton was divided into sections before using as input for the algorithm.

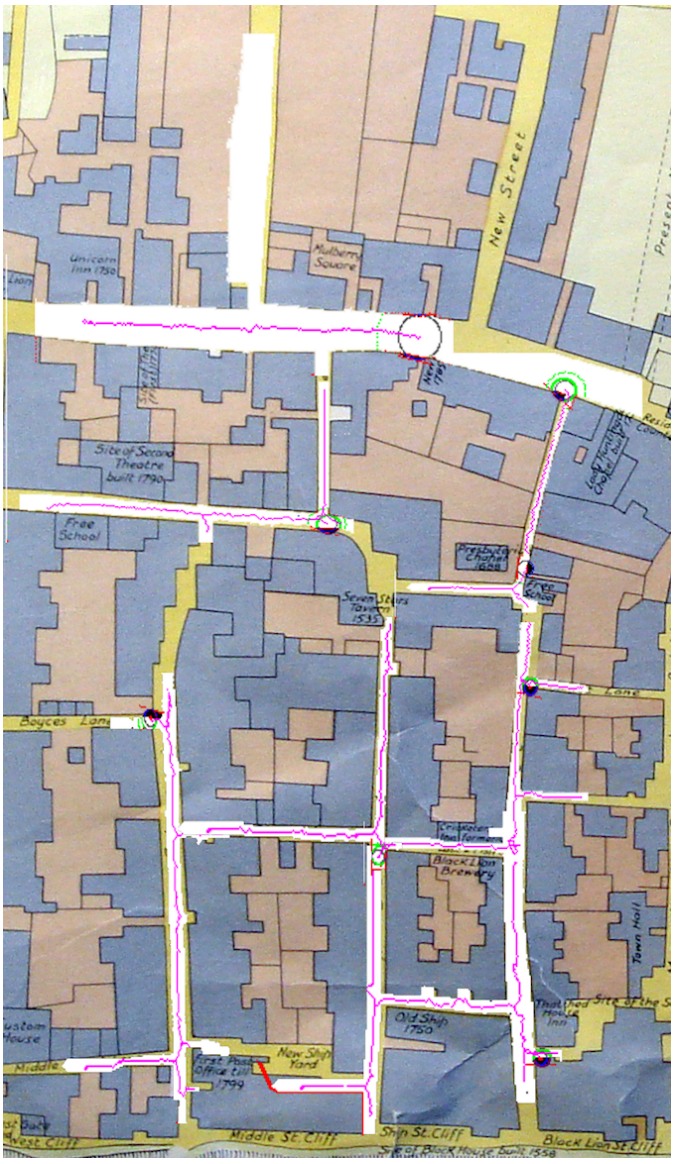


Figure A.16: This image shows how the algorithm performs on the Brighton map and the paths found.

Figure A.17: Shows a typical input image from the Brighton map.

Figure A.18: This image illustrates how the Koblenz road network appears in 3ds Max.

Figure A.19: This image shows how the 3ds Max model of Koblenz appears superimposed on top of the original map.

Figure A.20: This image shows how the Brighton road network appears in 3ds Max.

Figure A.21: This image illustrates how the 3ds Max model of Brighton appears superimposed on top of the original map.

Bibliography

[Ani10] DreamWorks Animation. Autodesk - film - dreamworks animation skg. http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=13186278&linkID=10164097, [accessed 2010].

[Aut10a] Autodesk. Autodesk - autodesk 3ds max. http://www.autodesk.co.uk/adsk/servlet/index?siteID=452932&id=12341413, [accessed 2010].

[Aut10b] Autodesk. Autodesk - autodesk maya. http://www.autodesk.co.uk/adsk/servlet/pc/index?siteID=452932&id=13742919, [accessed 2010].

[Aut10c] Autodesk. Autodesk - film - avatar. http://usa.autodesk.com/adsk/servlet/item?id=14270942&siteID=123112, [accessed 2010].

[ble10] blender.org. blender.org - home. http://www.blender.org/, [accessed 2010].

[Coa10] Steve Coast. Openstreetmap. http://www.openstreetmap.org/, [accessed 2010].

[EV07] Sander Oude Elberink and George Vosselman. Quality analysis of 3d road reconstruction. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36:305–310, 2007.

[Fri10] Bernard Frischer. The rome reborn project. how technology is helping us to study histroy. http://www.romereborn.virginia.edu/rome_reborn_2_documents/papers/Frischer_OpEd_final2.pdf, [accessed 2010].

[GD04] Xavier Granier and George Drettakis. A final reconstruction approach for a unified global illumination algorithm. *ACM Trans. Graph.*, 23:163–189, 2004.

[GFS+05] Gabriele Guidi, Bernard Frischer, Monica De Simone, Andrea Cioci, and Alessandro Spinetti. Virtualizing ancient rome: 3d acquisition and modeling of a large plaster-of-paris model of imperial rome. *Proceedings-spie the international society for optical engineering*, 5665:119–133, 2005.

[GL97] Armin Gruen and Haihong Li. Semi-automatic linear feature extraction by dynamic programming and lsb-snakes. *Photogrammetric Engineering and Remote Sensing*, 63(8):985–995, 1997.

[Hal89] R.W. Hall. Fast parallel thinning algorithms: parappep speed and connectivity preservation. *ACM, Commun*, 32(1):124–131, 1989.

[HSCP87] Christopher.M Holt, Alan Stewart, Maurice Clint, and Ronald H Perrott. An improved parallel thinning algorithm. *ACM, Commun*, 30(2):156–160, 1987.

[IBM10] IBM. Ibm archives: 7090 data processing system. `www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7090.html`, [accessed 2010].

[Ima10a] Mental Images. Mental images: home. `http://www.mentalimages.com/index.php`, [accessed 2010].

[Ima10b] Mental Images. Mental images: Technical specification. `http://www.mentalimages.com/products/mental-ray/technical-specifications.html`, [accessed 2010].

[Ima10c] Mental Images. Realityserver 3.0 white paper. `http://www.mentalimages.com/fileadmin/user_upload/PDF/RealityServer_White_Paper1212.pdf`, [accessed 2010].

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30. 1996. 3-211-82883-4.

[JL07] Taejung Kim Javzandulam and T. Tae-Yoon Lee. Semiautomatic reconstruction of building height and footprints from single satellite images. *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 4737–4740, 2007.

[Joh10a] Colin Johnson. About exrenda and colin johnson. `http://www.exrenda.net/about_exrenda.htm`, [accessed 2010].

[Joh10b] Colin Johnson. Exrenda - dudley castle c1550 visualisation. `http://www.exrenda.net/dudley/index.htm`, [accessed 2010].

[Kaj86] James T Kajiya. The rendering equation. *ACM SIGGRAPH Comput. Graph*, 20(4):143–150, 1986.

[Key10a] Heritage Key. Heritage key — unlock the wonders. `http://heritage-key.com/`, [accessed 2010].

[Key10b] Heritage Key. A reason for rezzing: how and why we built king tut virtual. `http://heritage-key.com/blogs/pavig-lok/reason-rezzing-how-and-why-we-built-king-tut-virtual`, [accessed 2010].

[Kru94] Fred N Krull. The origin of computer graphics within general motors. *IEEE*, 16(3):40–56, 1994.

[LD03a] Robert.G Laycock and Andy.M Day. Automatic generating roof models from building footprints. *Proceedings of WSCG*, Poster presentation, 2003.

[LD03b] Robert.G Laycock and Andy.M Day. Automatically generating large urban environments based on the footprint data of buildings. In *SM*

'03: Proceedings of the eighth ACM symposium on Solid modeling and application, pages 346–351. ACM, 2003. 1-58113-706-0.

[Lou10] Louvre. History of the project — louvre museum. http://www.louvre.fr/llv/apropos/fiche_apropos.jsp?CONTENT%3C%3Ecnt_id=10134198673232603&CURRENT_LLV_FICHE%3C%3Ecnt_id=10134198673232603&FOLDER%3C%3Efolder_id=9852723696500916&bmLocale=en, [accessed 2010].

[Mat09] MR Materials. Welcome to mr materials! http://www.mrmaterials.com/, [accessed 2009].

[NO94] Christian. Neusius and Jan. Olszewski. A noniterative thinning algorithm. *ACM Transaction on Mathematical Software*, 20(1):5–20, 1994.

[Ope10] OpenSimulator. Main page - opensim. http://opensimulator.org/wiki/Main_Page, [accessed 2010].

[Pro10a] Procedural.Inc. Procedural - features. http://www.procedural.com/cityengine/features.html, [accessed 2010].

[Pro10b] Procedural.Inc. Procedural - purchase. http://www.procedural.com/purchase/purchase.html, [accessed 2010].

[PV03] A.P.Dal Poz and G.M.do Vale. Dynamic programming approach for semi-automated road extraction from medium-and hig-resolution images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34:87–91, 2003.

[RAD10] RAD. Rad game tools. http://www.radgametools.com/default.htm, [accessed 2010].

[REB10] REBUS. Render farm 3d renderfarm rebusfarm render farm service. http://www.rebusfarm.com/, [accessed 2010].

[Rya10] Ryan. Crazybump. http://www.crazybump.com/, [accessed 2010].

[Ser83] Jean Serra. *Image Analysis and Mathematical Morphology.* Academic Press, Inc., 1983.

[SES95] Zhu Y. Seneviratne, L.D. Earles, and S.W.E. New line-based thinning algorithm. *Vision, Image and Signal Processing, IEE Proceedings*, 142(6):351–358, 1995.

[S.L10] Next Limit S.L. Maxwell render :: The next generation in rendering technology capable of simulating light exactly as it behaves in the real world. `http://www.maxwellrender.com/`, [accessed 2010].

[Sof10] Choas Software. Choas group / software offical website offical website - home. `http://www.chaosgroup.com/en/2/index.html`, [accessed 2010].

[Spl10] SplutterFish. Splutterfish: Brazil rendering system for 3ds max — home. `http://www.splutterfish.com/sf/WebContent/Index`, [accessed 2010].

[Stu10] ProMotion Studios. Kajimba. `http://www.kajimba.com/`, [accessed 2010].

[UMG10] UMG. Virtual past - brining histroy to life. `http://www.virtualpast.co.uk/`, [accessed 2010].

[Vir10a] Univeristy of Virginia. Rome reborn press release. `http://www.romereborn.virginia.edu/rome_reborn_2_documents/project_news/google_earth_ancient_rome_release_final.pdf`, [accessed 2010].

[Vir10b] University of Virginia. Rome reborn. `http://www.romereborn.virginia.edu/index.php`, [accessed 2010].

[Vir10c] University of Virginia. Rome reborn 2.0. `http://www.romereborn.virginia.edu/rome_2.0.php`, [accessed 2010].

[Vir10d] University of Virginia. Rome reborn 2.0 interface. `http://www.youtube.com/watch?v=PhzzjBPB25s`, [accessed 2010].

[VK95] George Vosselman and Jurrien de Knecht. Road tracing by profile matching and kalman filtering. In *Automatic Extraction of Man-Made Objects from Aerial and Space Images*, volume 1, pages 265–274. Birkhauser Verlag Basel, 1995.

[ZBC05] J. Zhou, W.F. Bischof, and T. Caelli. Robust and efficient road tracking in aerial images. In *In International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume 36, pages 35–40. 2005.

[ZMB99] Chunsun Zhang, Shunji Murai, and Emmanuel Baltsavias. Road network detection by mathematical morphology. *Bulletin of SFPT (French Soc. of Photogrammetry and Remote Sensing)*, 103:3–14, 1999.