SWANSEA UNIVERSITY

Digitising the Campus

Author:
Robert Fletcher

Supervisor:
Dr. Gary Tam

Project Dissertation submitted to the Swansea University in Partial Fullment for the Degree of Master of Engineering (HONS) - Computing

May 2014

Declaration of Authorship

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

Date:May 2014

Signed:

Statement 1 This dissertation is being submitted in partial fullment of the requirements for the degree of a BSc in Computer Science.

Date:May 2014

Signed:

Statement 2 This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are specically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

Date:May 2014

Signed:

Statement 3 I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Date:May 2014

Signed:

Contents

Chapter 1

Introduction

The inspiration for wanting to digitize the campus is to create a more sophisticated map of the campus. The sophisticated map would give the user a better appreciation of the scale of buildings and distances than what is available with the current software.

There are many different software programs on the market that a user can use to see what the campus is like, for example; Google maps or Swansea Universitys own map. Both of these options do allow a person who is new to the campus to be able to familiarise themselves with the campus from the comfort of their own home. When the user comes to the area they have been looking at, they experience a sense of deja vu . They will be able to navigate the real world as if they have been there thousands of times before. But these maps are lacking in some way, the Swansea Universitys own solution is just having a snapshot of one or two places in key spots around the campus. This makes it hard for a new person to the campus to work out the relation for those points.

With the building of the new Swansea University Bay Campus and with plans to change the existing campus, maps can go out of date fairly quickly. This means that an auto update feature is required so that the user does not have to remember to check if there is an update, the program instead would check and download in the same way browsers such as Chrome Firefox do.

In parallel to the consumer programs for mapping and navigation great strides have been made in the field of computer vision, which is able to generate 3D reconstructions of objects from only a set of 2D photographs. This technology is called Structure from motion (SFM). By having a 3D build of the campus I will be able to give users a sense of scale and a higher detail of what the building will look like.

SFM works by looking at a photograph and identifying key points in the photograph. This could be the edge of a window. It does this identification step for every photograph in the image set. After the identification step it compares all the photographs to each other, computing if the other images have the same similar points. If there is a match the system works out the deviation of the points and then works out where the camera was in 3D space. Once all positions of the camera have been worked out it aligns all the points in 3D space making a reconstruction of a building.

There have been various studies about computer use and what devices are being used. The UCAS Survey [1] among University Students showed that 80% of them carry a smartphone. With that large install base it would be beneficial that a version of the application be made available on smartphones and tablets so that users can look at the app and correlate images and buildings that they see in the app with the real life buildings.

My main motivation for this project is to be able to combine the high detail 3D reconstruction from SFM with the ease of use that products like Google maps have when navigating from one point to another. The final step would be to embrace the mobile world and build the app to work on Smart phones.

1.1 Aims of the project

From the introduction it is clear to identify the aims of the project. To summarize, the main aims are:

- Firstly generate 3D buildings from 2D photographs of the building. This will be accomplished using the Structure from Motion techniques
- Display Models to the user. Using Graphics programs the resulting model will be able to be viewed by the user.
- The project will also have navigation so that the user does not aimlessly walk around the campus but is directed to locations they desire.
- With the campus changing so frequently and better models being produced, a way to keep the map updated without the user having to check for the latest version.
- Create mobile apps for the user so that the program is available on Smartphones or tablets so that users can look at the project on the go.
- For the project to continue a way of converting the campus into a 3D model will be produced so that the project could extend easily to the new campus.

Chapter 2

Background

2.1 Related work

This section will cover related work to this dissertation. It will mainly cover various scene reconstructions, rendering methods and newer techniques in improving path finding algorithms over large open spaces.

2.1.1 Multiple view geometry

Multiple view geometry written by Richard Hartley and Andrew Zisserman [2] is a very comprehensive guide about computer vision. They discuss the fundamental techniques of how Structure From Motion (SFM) can reconstruct a 3D object.

The book splits up the reconstruction into four parts; Fundamental mathematics, 2D transformations that set out the ground work for the next section and Projective geometry of 3D space. The final chapters are about estimation, how using the geometry of the 2D images you can compute where that point is in 3D space.

A Structure From Motion (SFM) explains how from a set of photographs of one of the buildings on campus it can convert it to a 3D model. The first thing is that it determines what are the key points in the photograph, for example the corners of a window. After getting the set of points in the building it compares them with the points in all the other photographs in the image set. Once all the comparisons have been made it can start the reconstruction process. It looks at the deviation between the points and estimates how far apart the cameras were. Now that relative locations for both cameras are known the depth of that point can be calculated and placed. Do this for every point in the set and a full reconstruction can be achieved.

By studying how a 3D reconstruction from Structure from motion techniques is made, it helps me understand what sort of photographs are needed for the best reconstruction possible.

2.1.2 3-D HUMAN NAVIGATION SYSTEM

It is an interesting paper by Shohei Koide and Masami Kato [3] about maps for people who are walking, as they say in the paper; Walking is the most fundamental means of human transportation. Unlike travel by car, walking is not planar, but rather stereoscopic

Their approximation to navigation is to use RFID tags (Radio Frequency Identification) to give the device indoor navigation data. Each tag can contain information such as the location of which floor they are on. RFID can work over 1-2 metres depending on the chosen radio frequency. This means that the device does not need to be in direct contact with the transmitter but only needs to be in the range of the tag. The device such as a smart phone can estimate distances by using their inbuilt gyroscope compass and accelerometer which gives the device an estimation of movement and direction. The RFID tags along the route gives the device the data so that the device can correct itself.

Since the tags can contain more information than just location they can be more useful. In their example the campus main gates are closed from 8pm and open at 8am. Depending on the time, the tag can tell the device whether the shortest route for navigation is through the campus or if the gates are closed that the shortest route is to avoid the campus.

The tag for correcting the navigation is a good idea but comes at a cost. The cost depends on the accuracy you want and over what sort of area how many tags you want. Each tag does not cost very much but at around £1 and over a large area the price adds up and you will have to program each one individually, which takes time.

2.1.3 Osnap tool

Most Structure from motion software produces a point cloud (A point cloud is a set of data points in some coordinate system). Depending on the building size the point cloud could contain more than a million points. This would make it very hard to render and restrict it to higher end graphics cards. To make the render simpler it needs to convert it into a mesh. There are many different meshing algorithms available but Osnap takes a novel approach. Their approach is to use an ”automatic polygonal reconstruction that can then be interactively refined by the user”[4]

The tool looks at the point clouds and divides it up into plainer segments. It then calculates the boundary of these segments, which are then used in their optimisation algorithm. This algorithm tries to ”snap” the plainer segments together. The result is a simpler mesh but with all the geometry preserved. Depending on the point cloud that was submitted to the program there may be gaps in the mesh. This is where the user interaction comes in. In 3D modelling programs, for example Maya, the user can draw and edit polygons. The Osnap tool also uses

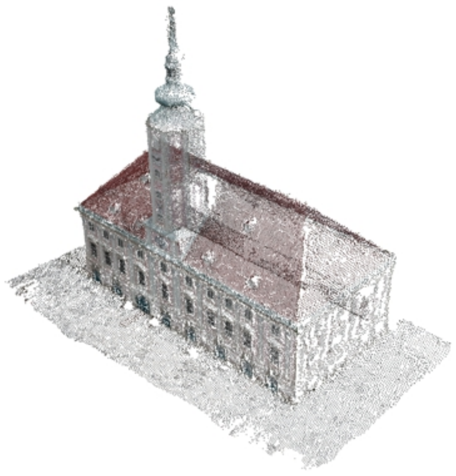


FIGURE 2.1: Osnap tool aligning point cloud

this approach where the user can snap the polygons together where the automatic tool did not, the user can also create new polygons in areas where the point cloud has not much data.

The result is that this tool can create a simpler mesh, which will make the rendering of it be easier to complete. The mesh would still be at a higher quality and retain the accuracy of the geometry.

2.1.4 Processing and interactive editing of huge point clouds from 3D scanners

This paper [5] discusses a new out-of core data structure and efficient editing of large point clouds. The out-of-core data structure references to an algorithm where the data is too large to fit in the computers memory for processing, but will need to be loaded in from external data sources for example hard drives or tape drives.

The data structure they propose is an octree. This is very efficient for storing a large multi-resolution point cloud. By having multiple resolution for the point cloud it will make rendering of the point cloud easier. As further away the point cloud is the less points you need to show and only when you are close to the building is when you need to show the high detailed version.

The octree is great data structure for storing this information. All the point cloud data is stored in the leaf nodes of the tree while all the inner nodes describe the resolution and the root node is the entire scene.

The approach is efficient because the data can be changed without sacrificing GPU utilization. This means that there is no waiting time for data being loaded and then being processed.

2.1.5 Scene Reconstruction from High Spatio-Angular Resolution Light Fields

This paper from Disney research [6] is about reconstructing a scene using a different methodology. In previous reconstruction it has all been completed with Structures from motion and computing a 3D point cloud. In this paper the construction is from 3D light fields.

Taking a dense set of photographs along a linear path creates the light fields. This produces a high detailed light field, which can then be used to compute the depth of the objects and segment the object from the background. Triangulating individual depth maps and merging them into a single model produces the meshes. To make the depth more obvious the mesh has been colour coded with pink for near and blue for further away.

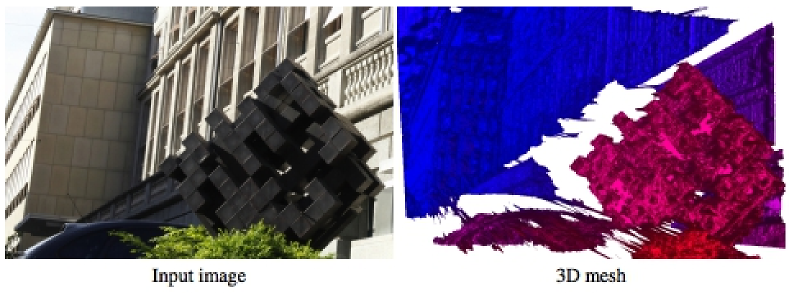


FIGURE 2.2: Visualisation of 3D meshes (colour coded)

Using 3D light fields for reconstruction leads to a higher detailed result. Also with technology progressing the stereo based depth images are unable to handle the required resolution that some fields need. For example, movies need resolutions greater than 4k (4096 pixels, 2160 pixels) or moving from 2 megapixels images to 9 megapixel.

2.1.6 Scalable Real-time Volumetric Surface Reconstruction

This paper [7] discusses an approach for creating volumetric objects in real time. The process uses a stereo camera set up in the Microsoft kinect camera. The kinect has an infrared depth sensor along with a RGB camera. This makes it a perfect device for real time depth scanning

The system starts with assuming a regular dense 3D grid. The kinect is then initialised in the centre of the plane. The system then takes in one frame of a noisy depth map from the kinect sensor. The surface observations from the sensor are integrated into the 3D grid. Then it is ray casted to find the surface points and normals. When the next frame is submitted the new camera position is estimated and the data submitted into the grid. By doing this the model is quickly built up and any initial noisy areas with not much detail can be easily built up.

The paper discusses the data structure that is being used. The data structure is as a sparse pointer structure in GPU memory. This makes it easy to scale and the reconstruction grows.

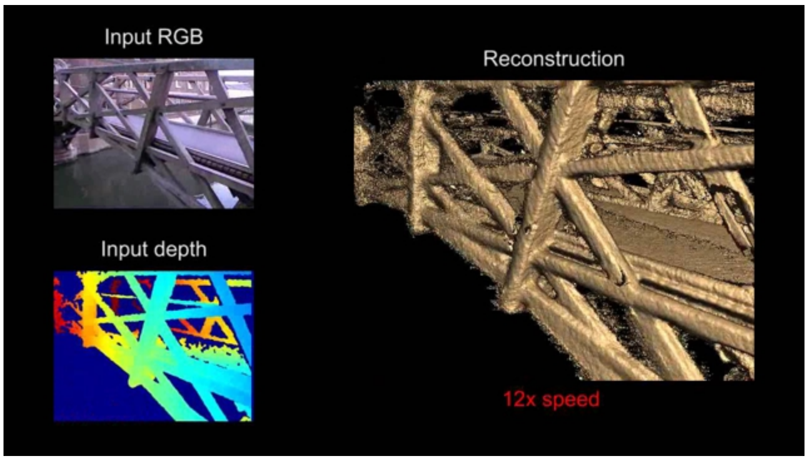


FIGURE 2.3: Showing the three different views, colour from the camera depth and the full real time reconstruction

Since this reconstruction is GPU intensive it needs a powerful machine to run on. The paper has resolved this by making it a simple client server application. The client is very simple, all it needs to do is send the depth maps from the kinect sensor to the server which has the GPU architecture capable to do the reconstruction. The server then sends back the ray casted image for the client to display. At this moment only the new specification of wireless routers are capable for constant colour and depth images.

This technology would make capturing larger scenes such as the campus very quick and in real time. The only limitation is having a capable pc for the reconstruction and having a reliable wireless signal around the campus to stream the depth map to the server.

2.1.7 Path finding in Open Terrain

Path finding is the method of finding the shortest path from a start state to an end state. Many solutions to this type of problem use graph based approaches while this paper looks at the search with a geometrical solution.[8]

There are many different search algorithms available. Most are heavily reliant or based on Dijkstra's algorithm for finding the shortest path on a weighted graph. The most common one is the A* algorithm

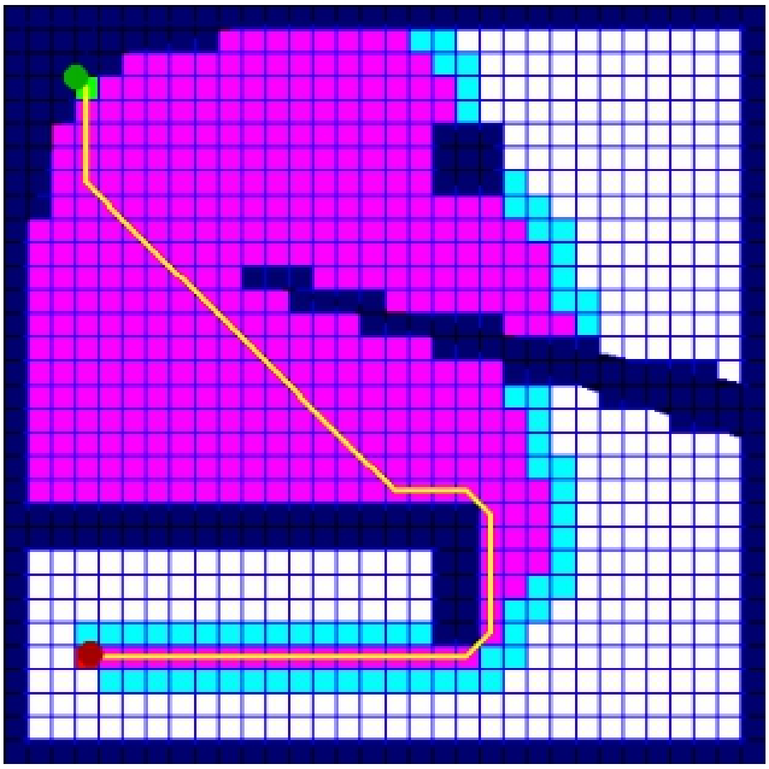


Figure 2.4: Grid Representation

In most applications of path finding mainly found in games programming the 2D scene gets split up into a grid (figure 2.4). The grid is good since it can support random access lookup. The compromises are that you trade off the size of the search space with the quality of the path found. In large open space designs this trade off is quite acute.

This papers[8] method to fix this problem is firstly to take the edges off the polygons. Then depending on the resolution that is required you split the edges up. You then take the outpoints in all the edges as the nodes in the search graph. This approach gives more flexibility in the detail than just using a grid. This does add a new problem, as the resolution increases the search cost at a greater rate.

To solve this problem they use boundary point solver (based on Snells Law) that eliminates the need to evaluate individual boundary points while retaining path quality. This algorithm is then integrated with the A* search algorithm to produce a more efficient and accurate search algorithm objects with large open spaces.

2.1.8 jMonkeyEngine 3.0 Beginner's Guide

This book by Ruth Kusterer describes the jMonkeyEngine SDK [9](Software developer kit) and its corresponding IDE (Integrated developing environment). It also describes the process from start to finish of how to create a fully functional program.

The book extensively covers building a game with AI mechanics and characters. Chapters 2 and 3 cover key areas with the latter explaining how to create a scene and interact with a user.

Chapter 2 teaches the various methods of creating a scene, including the inbuilt scene editor. Or you can uses the Orge exporter from many 3D modelling programs to export the entire scene and convert it into a jm3 binary file which can be rendered by the SDK. The other method is to simply render a common object format (obj).

Chapter 3 discusses the basics of interacting with the scene. Here too there are different methods such as implementing a simple camera, which the user controls with the standard WASD keys. There are also alternative methods discussed for implementing a follow camera and defining a motion path for it to follow.

Chapter 4 adds in information covering adding GUI (Graphics user interface) to the program where the user could interact with settings that are not directly connected to the render of the scene.

This book gives a good comprehensive knowledge to a novice on how to simply build and render a scene with user input.

2.2 Similar applications

2.2.1 Photo City Project

The photo city is a crowd source data-gathering (the method of using volunteers to get the data) project to get the data for reconstruction of the University of Washington campus.

The reconstruction process that is being used is the Structures from motion technique. For the best types of reconstruction using this method it is necessary to have a large dataset of the building with photographs from many different angles.

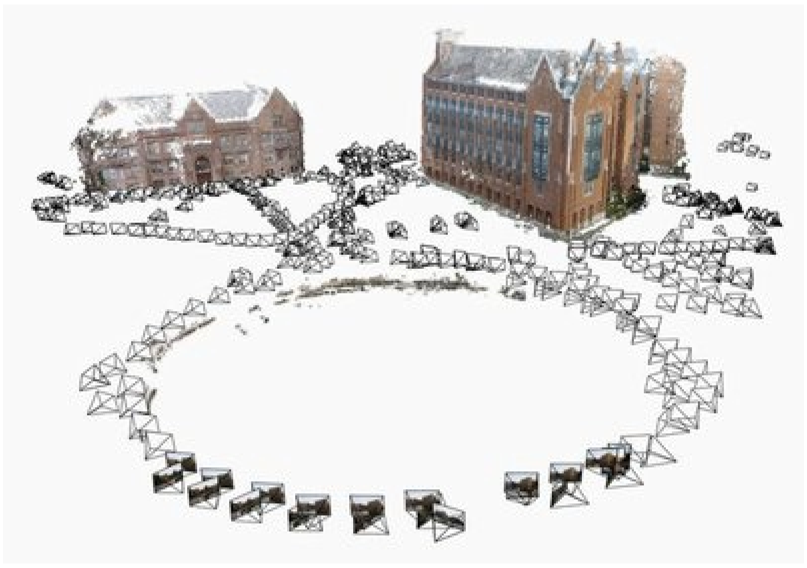


Figure 2.5: Photocity Project

The crowd sourcing is achieved by having people walking around the campus to take photographs of the buildings that are then aggregated on the server. To encourage people to take photos of the campus they have made a mobile application (app), which uses the techniques of gameification. Gameification is the method of using game mechanics for example point scoring and competition to areas outside of gaming. In this application the app has pictures of the buildings that need to be captured. Users take and submit photos of the building, only when there have been enough photos for a full reconstruction does the user get the building. The project has turned the method of getting the data into a huge capture the flag game.

By making the aggregated data fun to the volunteers it makes them more willing to capture more buildings. This produces a much higher dataset, which can be used in the reconstruction.

2.2.2 Building Rome in a Day

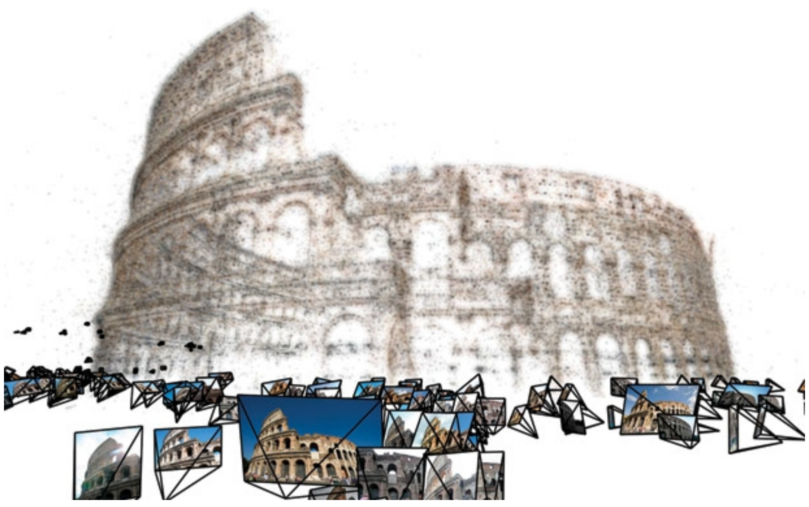


FIGURE 2.6: Small section of the Building Rome in a Day model

This is one of the first projects that used a large dataset to reconstruct a large scene. The project was designed to reconstruct Rome in under a day

The first problem was gathering enough photos that would cover the desired area. This was accomplished by using a photo-sharing network called Flickr. When this project was released back in 2009 Flickr had 40 million members [10] and 4 billion photos stored. This is a large dataset, which is excellent to call on by searching terms like Rome or Coliseum. This can produce a large enough dataset that the software can run on.

The dataset for the Rome project comprised of over 150,000 images, this would take a long time to compute the SFM on a single machine. They centre the research on using the parallel processing to the fast matching of similar structures in the image data set. A cluster was able to render all the points in only 13 hours, however the issue is the cost. To render a cloud cluster that would compute it would cost around £400 [11] (500 nodes running for 13 hours)

2.2.3 Science museum 3D model

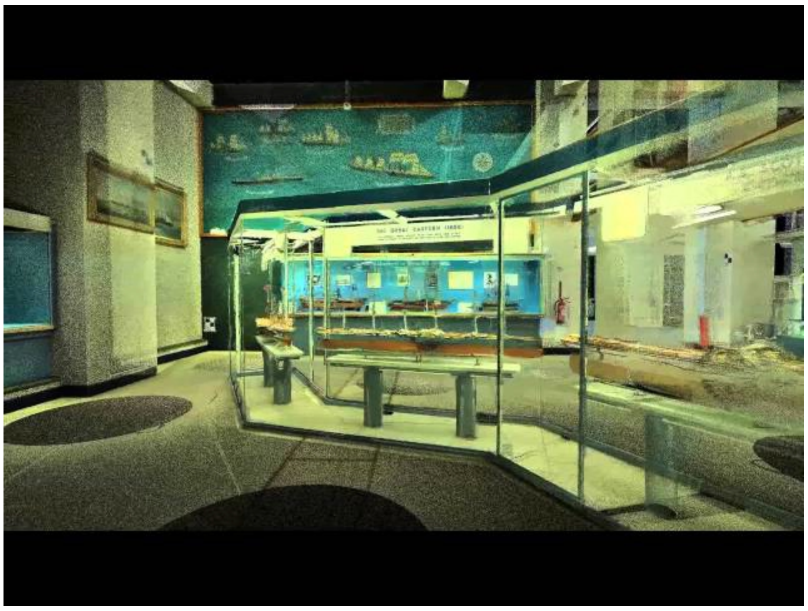


Figure 2.7: Very detailed reconstruction

The science museum London was going to change one of their galleries from displays all about the industrial revolution to displays about the informational revolution. To have a record of what the old gallery looked like the museum wanted a 3D model.

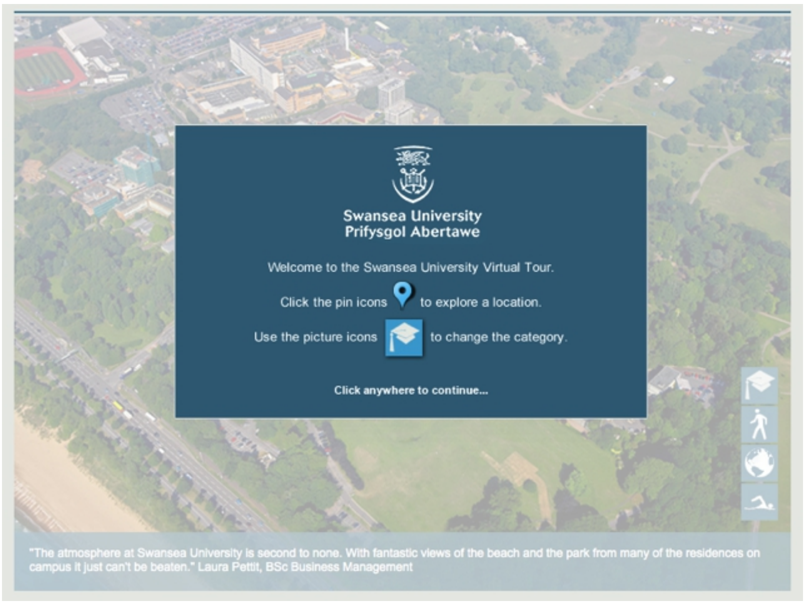
The 3D model is a point cloud that was created by Scan LAB Projects. They used 275 laser scenes gathering 2 million measurements of the gallery. By using laser scanners they could very accurately capture all the details of the gallery and produced a very accurate model.

2 million points is very hard to render on average available pc hardware. To make the 3D gallery available to everyone they have made video tours but only using 10% of the data.

2.2.4 Swansea university virtual tour

The university has its own mapping application done by Revolution Viewing. This mapping application is available on the Universities website. It allows users to select a location based on an area such as a sporting facility or accommodation and it will show on a 2D map the various locations based on the users own choice. When a user selects that location the user is presented with a panorama of that location. The user can pan around and zoom in to get a detailed look at the area.

The technology is mainly based on taking several photos from one single spot and then stitching the images to produce one single image that can be wrapped around the user giving them a 360° experience.



Swansea University
Prifysgol Abertawe

Welcome to the Swansea University Virtual Tour.

Click the pin icons to explore a location.

Use the picture icons to change the category.

Click anywhere to continue...

"The atmosphere at Swansea University is second to none. With fantastic views of the beach and the park from many of the residences on campus it just can't be beaten." Laura Pettit, BSc Business Management

FIGURE 2.8: Screen capture of the online university tour

The advantage of this approach is that the user can get a high detailed image of the area. But the problem is that the pictures don't always correspond with the location on the map. Another problem is that some areas (for example the mall) don't show the entire 360° image.

2.2.5 Google Street View

Google Street view is one of the most popular mapping and view technologies that are out on the internet today. The user selects a location, drags a yellow man icon onto the road and the map shows them a panorama of that exact spot. In the panorama view the user can move around and see a 360° degree view. Also in that view the user can select a spot and move down the road as if they were actually walking along.

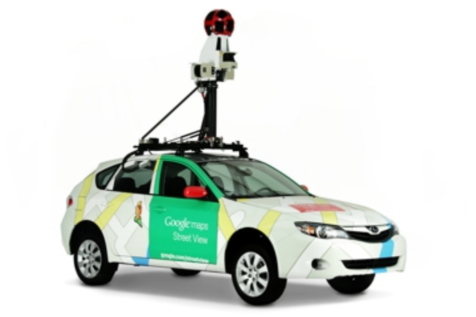


FIGURE 2.9: Google Streetview Car

Driving a customized vehicle down all the worlds roads makes the panoramas. The vehicle has 9 directional cameras pointed in all different directions. These 9 photos at each point are stitched together to form a panorama. On board there is GPS to accurately record the position the panorama is taken from. Also there are laser scanners that record a basic model that is used in the transitions between two panoramas to give the user some perspective.

2.2.6 AutoDesk123

Autodesk 123 [12] is a free web app that allows new users to 3D modelling to easily create objects in the 3D world. It has many different features with the most interesting one for this project being 123-Catch. 123 Catch uses either your smartphone or photos you upload to it to generate a 3D model.

The models it can produce have a high quality texture. From the tutorial it suggests taking around 20 photos from different angles to get the best result. Looking at how the software asks for images from different angles the assumption is that it is using the Structure from motion algorithms in the process of stitching all the images together. Since this software is being run by a large company Autodesk it is able to put large clusters designed to match and generate the model quickly.

From their gallery most of the images that get produced in this way are of small objects such as little toys. Most people have used this software with smaller objects for example a toy car, but large objects have also been made for example Casa Pojnar building in Romania.

2.3 Proposed tools

2.3.1 Image and modelling processing

2.3.1.1 Adobe Bridge

Adobe Bridge is a photo management tool. It has numerous useful features to quickly and efficiently mange the photosets that will be used in the Structure from Motion software. Most of the photos will be taken in RAW format. RAW is a lossless (has very little compression) image file where the image contains all the information the camera sensor collects with little or no compression. Since most Structure from motion software cannot read RAW as most camera manufactures have slightly different formats, the photos will need to be converted into a readable format. Adobe Bridge has a good tool for quickly rating photos so any photos in the

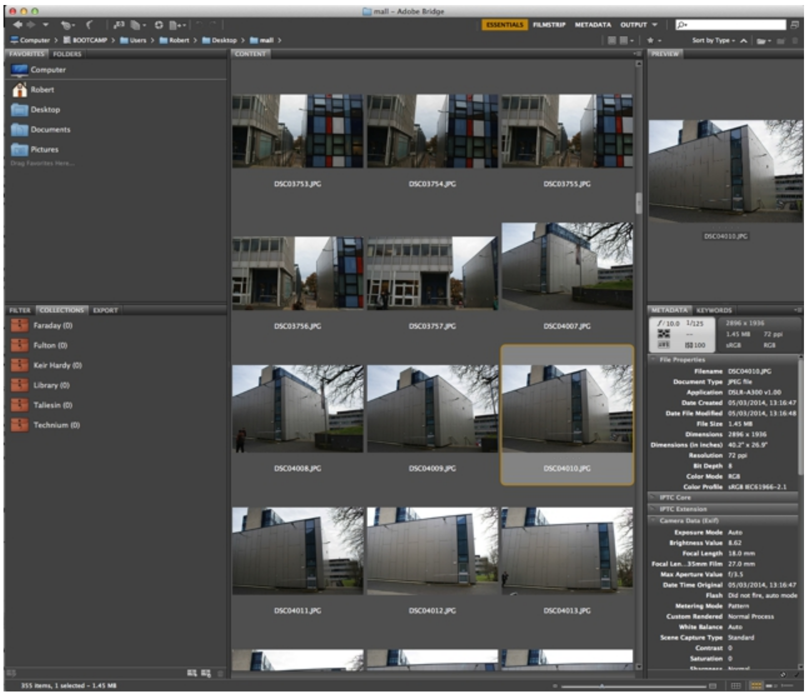


Figure 2.10: Adobe Bridge

photoset that are not good quality, for example very blurred, can be quickly discarded. With the collections feature of the software it is also very simple to move photos into the collection about one building. Once organised it is a simple step to convert the collection of photos into a format like PNG, which the Structures from motion software can read.

2.3.1.2 VSFM

VSFM is the Structure from motion software I will use to do the main reconstruction. VSFM is GUI implementation of several projects including the Bundler toolkit [13], the SiftGPU, a GPU implementation of Scale Invariant and Clustering Views for Multi-view Stereo (CMVS). All these tools in this software are utilised to convert the photoset into a dense point cloud.

Depending on the photoset this system will take 1-5 hours to produce a dense point cloud that can be used in the final software. The system can utilise the GPU of the hardware to make the feature detection much faster then just relying on the CPU.

2.3.1.3 MeshLab

MeshLab is a free Open source project that contains many different algorithms that can modify point clouds and meshes. This software will be mainly used in the post process of the point clouds that is generated from the VSFM software. The software will clean up the point cloud to remove irrelevant data. The main algorithms will be a position reconstruction that will be used to convert the mesh into a point cloud. Another algorithm will be usefto assign the faces of a mesh a texture that is derived from the original point cloud.

Another algorithm that can be used is a tool to fix minor holes in the mesh where the reconstruction algorithm was unable to make a face. A further algorithm that can be used is Quadric based edge collapse which will be used to simplify the mesh. This will make the geometry the same but will reduce the number of faces i.e. make the faces larger.

The software can then export the resulting mesh with a texture to many different object formats that can be used in different rendering or editing programs. The main format this project will use is the universal obj format.

2.3.1.4 Blender

Blender is a free modelling program that can create various projects such as 3D effects, animation and 3D models.

The VSFM computes the geometry of the buildings relative to the photos. This means that all the building is not at the same scale. Blender is used to create the 3D scene that will ultimately be rendered by my software. By placing all the buildings in their locations that can be scaled so that they are all at the right size and the campus appears realistic.

Blender can also be used to fix some problem with the models when output by Meshlab. In some occasions the texture is not correctly assigned to the model so when being rendered the models appear without a texture.

2.3.2 Programming environments

2.3.2.1 Android Studio

The Android Studio is a customised IDE from Google based on the IntelliJ platform. This means that along with all the features IntelliJ gives you with development, for example code suggestions, it also has features developed by Google specifically for Android. These include the Android GUI, where you can see your UI that you have designed on multiple screen resolutions and sizes all at the same time.

This development environment also makes it very easy to go through the steps to compile the project and build the application. It also has a tool to sign the application for it to be submitted into the Google play store.

2.3.2.2 jMonkeyEngine

JMonkeyEngine is another IDE based on the Netebeans platform. It combines the developing environment and code compiling options that Netebeans offer with the JMonkeyEngine SDK. This SDK contains all the necessary libraries that make writing java OpenGL code much easier. Since the main language is java it is designed to do quick deployment of the application to multiple platforms including desktop, web, and mobile.

The IDE also has reasonable connections with the Blender modelling software. With a plugin installed in Blender a scene can be exported and converted into a format that can be rendered in game. Since OpenGL has been extracted into the SDK writing code to manipulate object in 3D space has become very trivial and does not require a very large learning curve.

2.3.2.3 Unity3D

Unity3D is a game engine with its own IDE. The game engine is designed to compile down either DirectX or a version of OpenGL depending on what platform the application is compiled for. This makes it highly optimised to be able to render high-quality 3D meshes.

The IDE has many features; it supports many different languages thanks to Mono Development. Mono Development is an open source implementation to the Microsoft .Net framework and C# language. This means that the IDE supports code written in a number of languages; C#, JavaScript and BOO language (similar to python). Along with the different languages it has support for many different modelling applications including Blender, Maya and 3DSMax. This makes converting a 3D scene from one of the applications into an application that a user can walk about very easily. Since it is primarily a game engine it has many features that game developers want such as navigation and collision detection.

2.3.3 Languages

2.3.3.1 Java

Java is well suited for developing on multiple platforms. This is because it compiles down to java byte code that is run on java virtual machines. These virtual machines can be installed on many platforms including Windows, Mac OSX and many distros of Linux. This makes it a perfect language to develop a cross platform application since you only need to develop and compile once. Also it is the main language for the jMonkeyEngine IDE.

2.3.3.2 C#

C# is a Microsoft language with a similar syntax to java. Mono Development has worked hard to make most of the C# language open source so it can be compiled on Mac OSX and Linux. It is one of the main developing languages for the Unity3D IDE.

2.3.4 Hardware

2.3.4.1 Camera

To gather the photoset the camera that WAS used is a Sony Alpha 300 DSLR (Digital Single lens reflect camera). It haD a resolution of 3872 x 2592 pixels with a max 10 megapixels. It will have an 18- 75mm f3.5 lens, but photos were taken with a focal length of 18mm to give the widest field of view.

2.3.4.2 Desktop

This will be the primary machine that will do all development and image reconstruction and re-meshing. The pc is a 2011 iMac with a 2.7 GHz Intel Core i5, 8GB of Ram and an AMD Radeon HD 6770M 512 MB.

2.3.4.3 Mobile Device

The tablet which was used in the development of this project is the original Nexus 7 tablet from Google that had 7 inch 1280x800 display, an NVIDIA Tegra 3 quad-core processor and 1 GB of RAM and running Android 4.4 (Kit Kat).

2.4 Development model

There are many different types of development models available but this project will use the agile software development model. This model has a number of features that are beneficial to this project.

Agile Development works in the following way

1. First you find out what are the features the application may have. Speaking to the client generally does this.
2. Once you have all the features they are prioritised to find out which ones are the most urgent to complete.
3. With the priorities set you work on the features for 2-3 weeks.
4. At the end of the time the feature is reviewed in order to see if it works. If the feature is in working order then it is submitted to the application

Steps 2-4 are repeated until all the features have been added to the application or the release date for the application has been reached.

This method has been chosen for this project, as it is more flexible than other models, for example the waterfall. In that methodology the developer needs to know all the features and plan ahead of programming. It can also lead to a program that the user did not want because they changed their mind. Agile software development is more flexible. In between steps 2-4 the user can come to the developer, change a feature or request new ones. These alterations are then assessed at step 2, and the new high priority features can be developed.

Agile Development has been adopted in many different types of organisations large and small alike. Some include:

- Google
- Microsoft
- IBM
- Carbonite

For this project since all the features are not known at the very beginning of the development cycle and since many people may use this software for viewing the campus getting feedback as early as possible would help decide what features are needed.

2.5 Project Plan

There are many ideas that this project can include and could go into many different directions. To solve this problem the project features have been separated into major features and minor features. Major; these will be the high priority ideas that would be implemented first and are the most important to the project. Minor; this would be a feature that I would like to implement if I have enough time. Major features: produce a 3D model of the campus and provide a navigation system for the model. Minor Features: Create an Android app. Create some small-scale auto update.

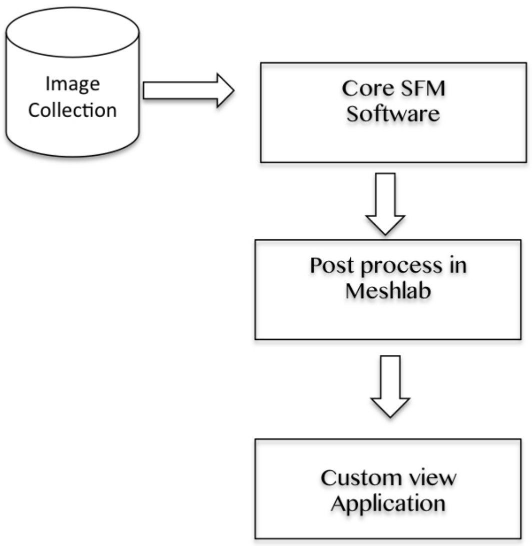


Figure 2.11: Basic version Pipline

The core plan to the project is simply turning a set of images into a 3D model of the University campus. This can be done using this simple pipeline: Input: Set of photos with a large overlap between them processed by a 3D reconstruction software like bundler

Output: a 3D point cloud that represents the model of the University campus.

The core can be achieved using software currently available on the market. The core can be added upon by using a photoset taken with a camera as the input source. The photoset will have images with large overlaps. These images will be used in the 3D reconstruction software to generate an improved model. A further addition to the core would be to build a simple application that would be able to use the model and do simple navigation for the user.

Some advanced features to the project would be building an Android application. Since Android devices have different requirements to a desktop, for example a touch native User interface and they have lower hardware specifications some work to change the code from the desktop application will be needed. Another advanced feature is the auto update. This will work by storing new revisions to the program on a server. When the user launches the program, it checks to see if there is a new version available, download it and run it. This is done because the campus changes quickly and newer models will be added to the scene, A method to get

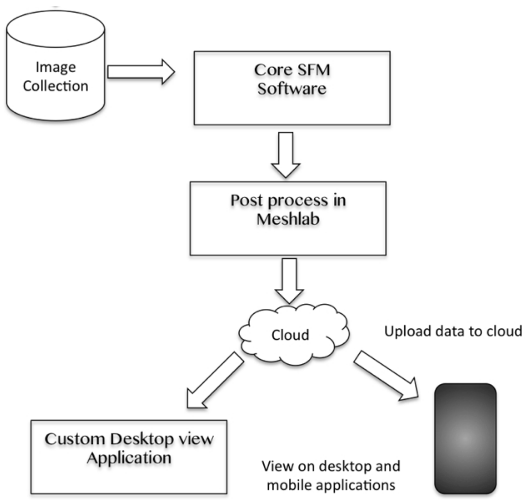


FIGURE 2.12: Advance version Pipline

the new scene without a large user involvement is needed, so that the user does not have to keep checking online whether there is a new one and go through the process of installing the application.

Unified Modelling Language (UML) is a useful tool for planning class hierarchy. This is planning what are the necessary classes and how these classes interact. It helps to make an efficient program because you will not have duplicate function is classes. It also makes it easier to develop since you know ahead of time what each class needs to do.

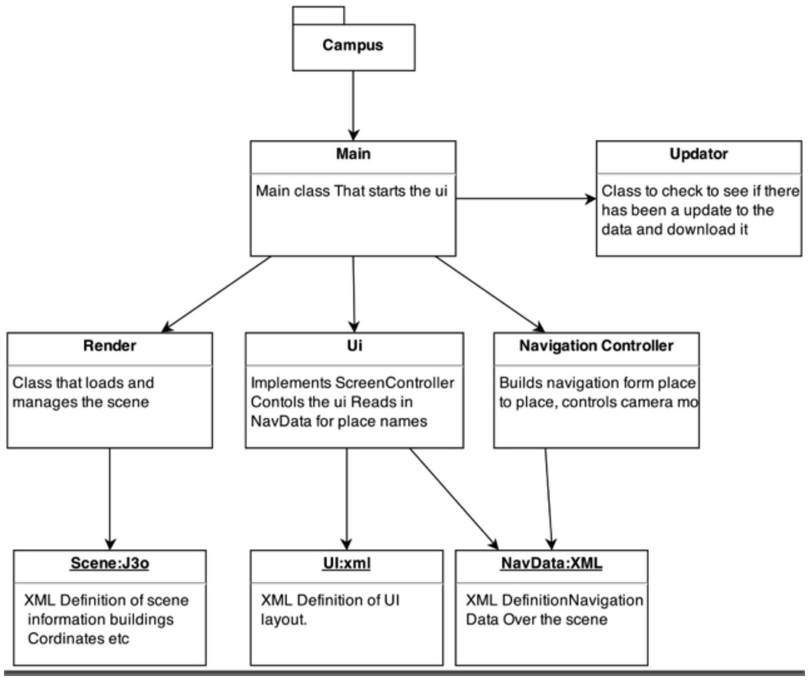

Campus
Main
Main class That starts the ui
Updator
Class to check to see if there has been a update to the data and download it
Render
Class that loads and manages the scene
Ui
Implements ScreenController Contols the ui Reads in NavData for place names
Navigation Controller
Builds navigation form place to place, controls camera mo
Scene:J3o
XML Definition of scene information buildings Cordinates etc
UI:xml
XML Definition of UI layout.
NavData:XML
XML DefinitionNavigation Data Over the scene

FIGURE 2.13: Class hierarchy

2.6 Time management

With the chosen development model being agile development it allows the ability to track the progress of the feature being developed. That being the case, software called xPlan[14] can be used to monitor the development of each feature. In Figure 2.14 you can see the output of the software in a Gnatt chart. I have colour coded the different tasks to be able to quickly see the different types.

- Red are deadlines,
- Orange is coursework
- Green is Feature Development
- Blue is research
- Grey is writing the dissertation
- Yellow is holidays
- Black are Exams
- Light blue is presentations

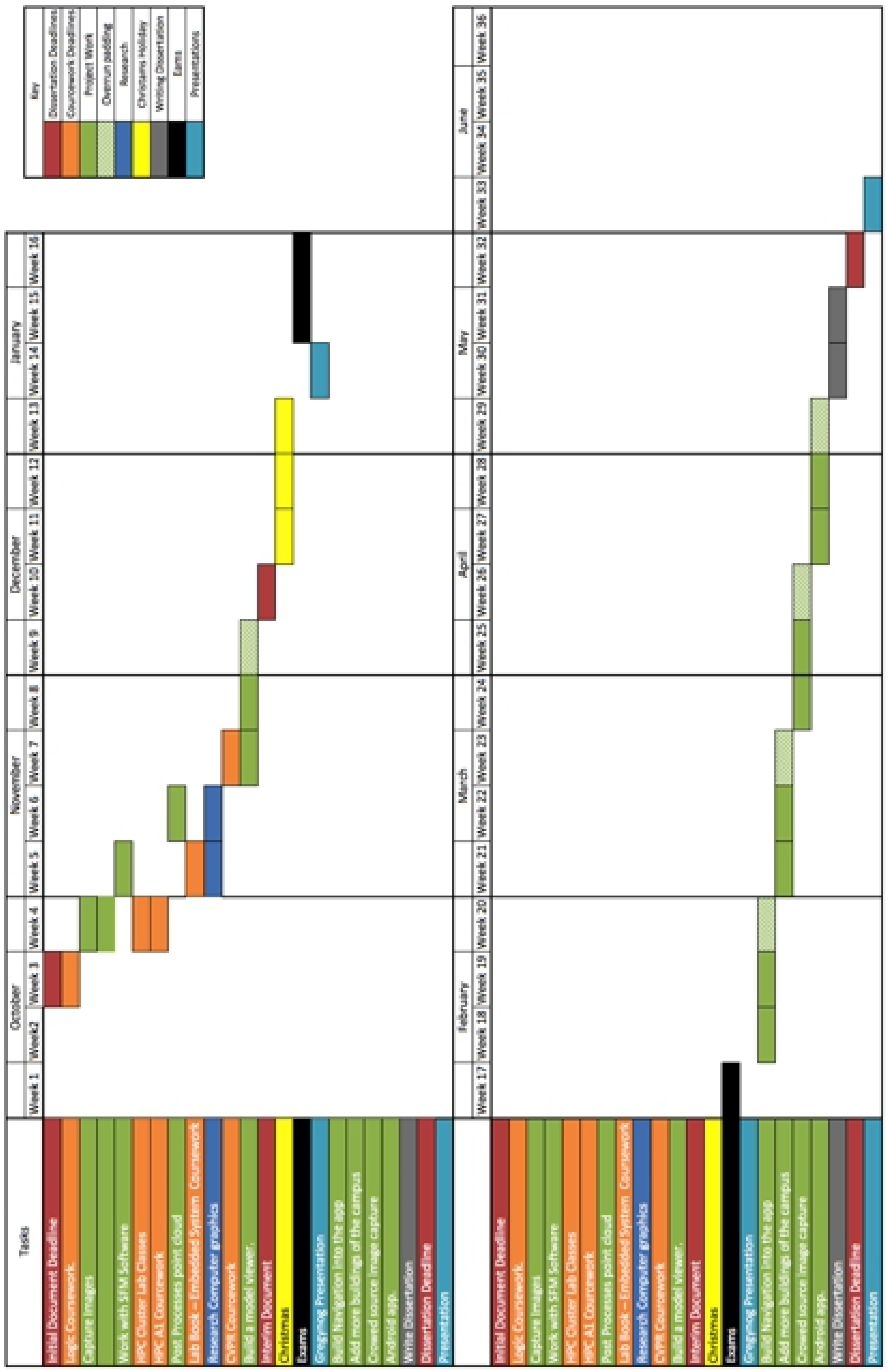


FIGURE 2.14: timetable

2.7 Risk assessment

In the process of building this project there are a number of problems and other risks that could occur. This chapter will identify the possible risks and how they could be avoided or if they are unavoidable, how to overcome them. List of possible problems

Illness. This can be resolved by simply following the timetable (viewed in chapter 2.6). The Green hashed area includes overrun times so if a feature takes longer than expected to develop the timetable automatically gives contingency before it. Therefore if a problem does occur it can be resolved quickly and with less stress on the developer.

The model cannot run on the tablet. Depending on the possible size of the model and the hardware specification of the tablet, it may not be able to render all the points on the point cloud. If this does happen, a possible solution to it would either split up the model into separate buildings that are more manageable to render. Or produce video flybys that can then be viewed on the tablet.

The software does not work. As you can see in the project pipeline the core of the project centres around the 3D reconstruction software which does the transformation from the set of images to the 3D point cloud. If the piece of software chosen to do this does not work it will be one of the largest problems since it is the core operation in the pipeline. To solve this problem I have looked into some alternatives to the chosen software.

Limited computer graphics background. One of the challenges involved in doing this project is the limited computer graphics background I have, so the learning curve on this project will be very high. Due to the steep learning curve the risk for problems associated with my inexperience could occur. To resolve this problem I will refer to books on the computer graphics background that are designed to teach someone on how to build an application that will be able to visualize the model using some of the well know libraries

Data loss. One of the most common problems that could occur is data loss or corruption. This problem can easily be solved by following Peter Kroh from digital photography best practices, a division from America society of media photographers [15]. He has created the 3 2 1 backup strategy which is to have 3 copies of the data on 2 different types of media and one to be off site. This will be implemented for this project. For the project there will be a local back up of the computer the project is being worked on, onto an external hard drive and also one copy of the data will be stored in Dropbox so that there is also a copy stored on drop box servers.

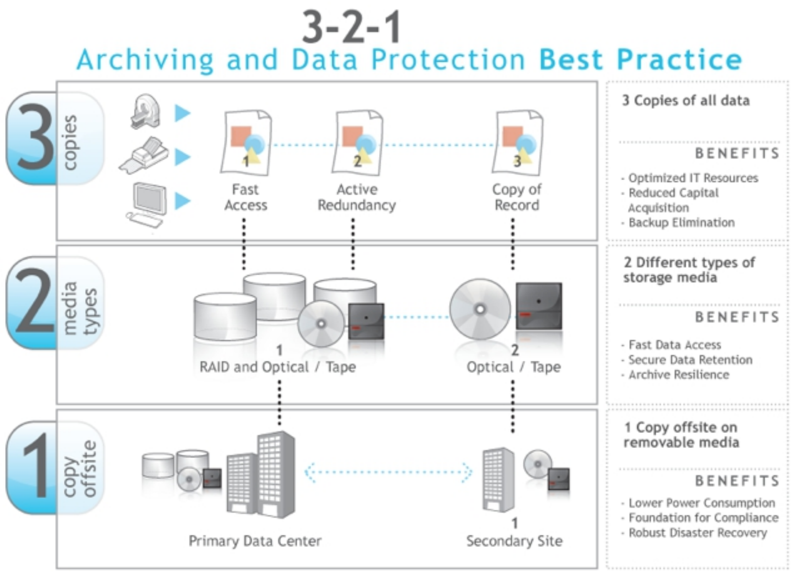


FIGURE 2.15: 321 Backup Strategy

Image Capture. One of the main features of the program is the models that are needed. If the camera used to get the model is damaged then no models can be produced. This can be resolved by having a spare camera that can be used if the primary camera does not work. An alternative strategy is to use photos from sites like Flickr or Google Plus to get the images that are needed.

Chapter 3

Creating the Campus

3.1 Experiments with SFM Software

VSFM [16] is a GUI (Graphics user interface) version of Structure from motion software that will be used in this project. I tested it first with an example dataset of a small Greek temple. I started to experiment with taking different photos to see how they make a point cloud. I started off working with a small model of a train. I tried having the same approach with taking the images that I would if I was making a panorama. This caused some unexpected results. Since the overlap was insufficient and only a limited number of photos were taken from each side of the train, i.e. front view, side view back view and top view, the software was unable to make enough matches on the train to produce an acceptable representation of it.

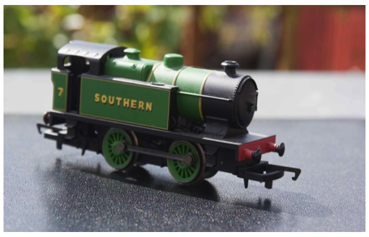
		

TABLE 3.1: 3 out of 13 images used for reconstruction

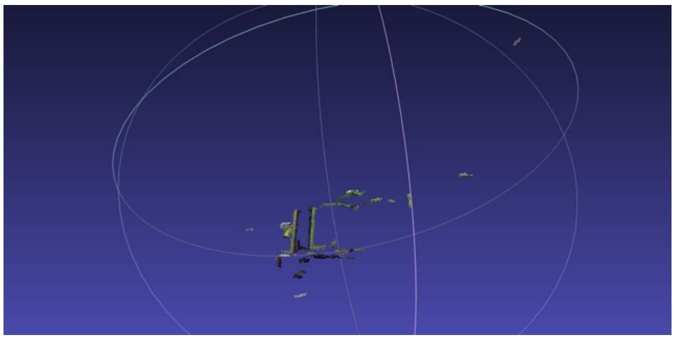
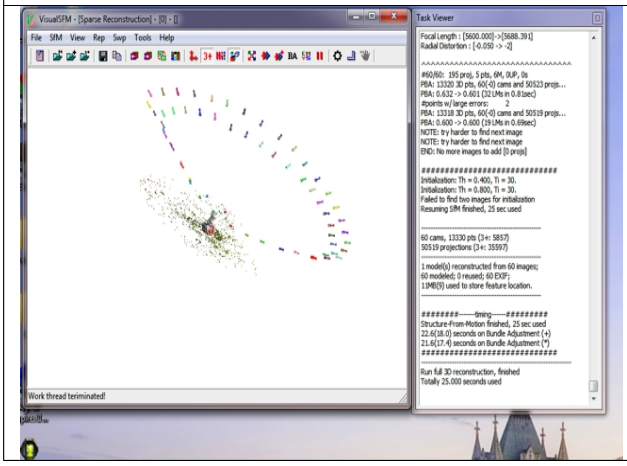
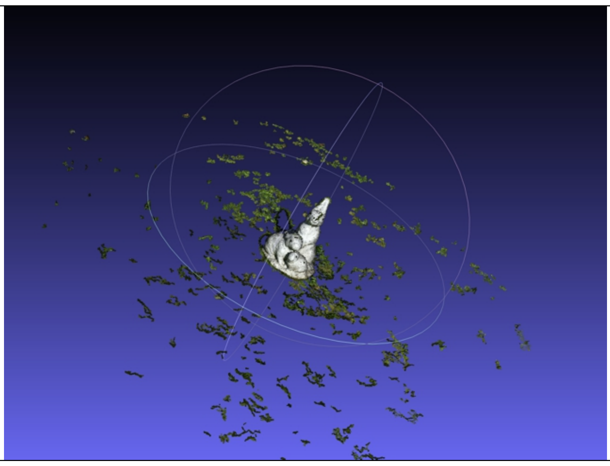


FIGURE 3.1: Result of the VSFM Software with only 13 images

I next tried a larger model. I chose a 60cm tall garden ornament of a rabbit. Here I took more photos with a larger overlap than before. The software was able to get enough matches to reconstruct the fence in the background of the images but not enough matches for the ornament. After viewing some videos online and seeing the image sets they were producing I understood I needed more overlap between the images so that the software would get more matches of the ornament and produce an improved representation of the ornament. A further exercise was undertaken whereby more photographs were taken of the ornament. This resulted in the generation of a convincing 3D model which only had a small amount of discrepancy.

Table 3.2: 3 out of 60 images used for reconstruction

Sparse Reconstruction	Dense Reconstruction
	

```
Focal Length : [5600.000]->[5688.391]
Radial Distortion : [-0.050 -> -2]

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
#60/60:  195 proj, 5 pts, 6M, 0UP, 0s
PBA: 13320 3D pts, 60(-0) cams and 50523 projs...
PBA: 0.632 -> 0.601 (32 LMs in 0.81sec)
#points w/ large errors:      2
PBA: 13318 3D pts, 60(-0) cams and 50519 projs...
PBA: 0.600 -> 0.600 (19 LMs in 0.69sec)
NOTE: try harder to find next image
NOTE: try harder to find next image
END: No more images to add [0 projs]

##################################
Initialization: Th = 0.400, Ti = 30.
Initialization: Th = 0.800, Ti = 30.
Failed to find two images for initialization
Resuming SfM finished, 25 sec used

----------------------------------------
60 cams, 13330 pts (3+: 5857)
50519 projections (3+: 35597)
----------------------------------------
1 model(s) reconstructed from 60 images;
60 modeled; 0 reused; 60 EXIF;
11MB(9) used to store feature location.
----------------------------------------

########------timing------#########
Structure-From-Motion finished, 25 sec used
22.6(18.0) seconds on Bundle Adjustment (+)
21.6(17.4) seconds on Bundle Adjustment (*)
##################################
------------------------------------------------
Run full 3D reconstruction, finished
Totally 25.000 seconds used
```

VisualSFM - [Sparse Reconstruction] - [0] - []
File SfM View Rep Swp Tools Help
Task Viewer
Work thread teriminated!

Table 3.3: Sparse and dense reconstruction

3

3.2 Getting the photoset

The first step in digitising a building is to get the required images that the Structure from motion software can use to turn it into a point cloud. From the experiments I did with the software describe in 3.1 I knew that the image set needed to have large overlaps in order to get a satisfactory reconstruction.

To try and get the whole building in view and therefore have the maximum amount of detail, we need the widest focal length possible. Also we need everything in focus. This requires having a small aperture. On the camera chosen (Sony alpha 300 DSLR) these settings can be set manually in order to get similar images. By having full control we can make sure that all the images have the same exposure (have the same brightness). In order to have photos that were at a high enough quality they must not have any motion blur from camera shake or people passing in front of the lens. To achieve this the shutter speed of the camera must be set fast enough not to produce blurry images but not too fast that it makes the photo under exposed, a value of 1/100 - 1/250 depending on the light conditions. The final consideration is what the resolution of the photos will be. The smaller the resolution the faster the processing but due to the smaller size there is less detail and therefore a poor reconstruction. The maximum the camera could do was 10megapixels but after experimenting in the VSFM software a resolution of 5.6 produced detailed reconstructions in a fast enough time.

The final camera settings that were decided are:

Resolution	5.6 megapixel
Focal length	18mm
Aperture	f22
Shutter speed	1/200

The process of photographing buildings starts with a birds eye view of the campus from Google maps and plotting a route that would encompass the building. I traversed my plotted route taking images approximately one metre apart from each other to create the photoset. This photoset is then reviewed in Adobe Bridge where any photos that would not be useful in creating a good reconstruction are then removed.

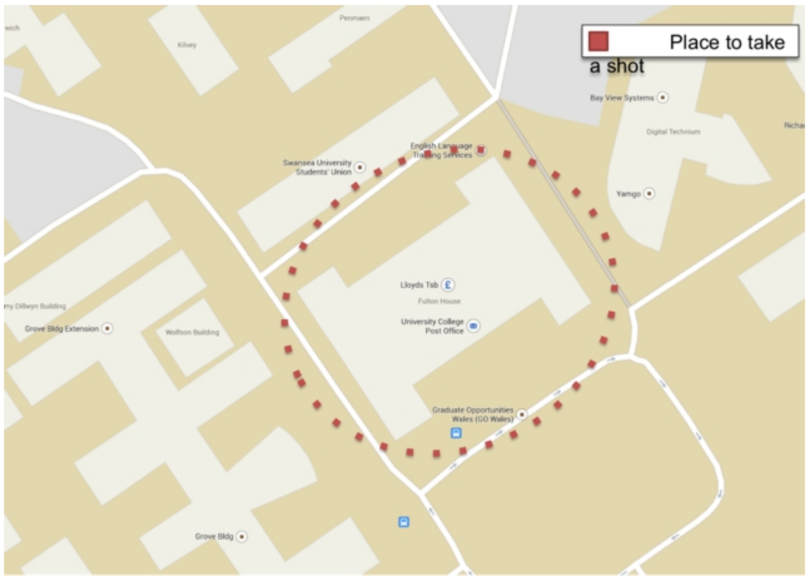


FIGURE 3.2: Plot of where to take photos

3.3 Creating the point cloud

After gathering the photoset, the next step is to generate the 3D point cloud. This will turn our 2D photos into a 3D representation of the building. The easiest way of doing this is to use Structure from motion software. Using VSFM we import the photos and let it do the feature detection, which computes all the points in the image. After the feature detection has completed we can do the reconstruction. The reconstruction takes the entire feature points from the image and compares them with the feature points in the other photos. Once all the matches have been completed it uses those matches to compute the geometry and aligns all the points generating a 3D reconstruction of the building. This gives a good representation of the 3D geometry of the building but not enough detail in the sides of the building. VSFM are another feature, Clustering Views for Multi-view Stereo (CMVS) [17]. This feature allows for a dense reconstruction to happen which produces a very dense point cloud, which has all the detail in the faces of the building.

When making the model I discovered how time consuming the process is. I decided to do some experiments to see how long it took to make a point cloud on different image sets.

Number of Pictures	Time (Seconds)
56	500
85	1000
120	1600
130	2000
140	2400
150	2800
160	3100
170	3400

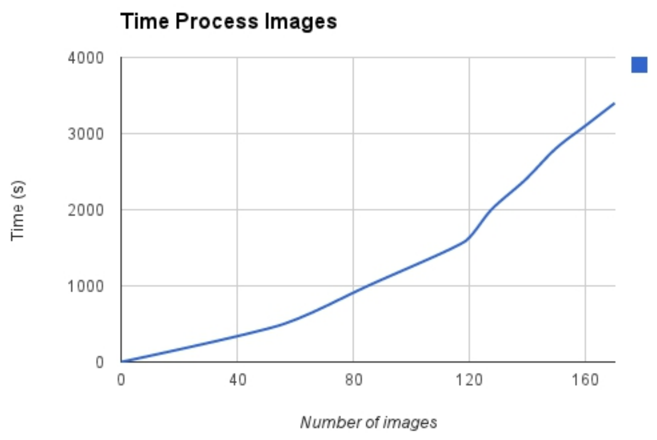


FIGURE 3.3: Graph of performance

As you can see, the time is a linear function to the number of image sets. For a good model it needs an image set of 250 -300 so that the model is detailed enough. Extrapolating the function the time for an image set of 250 images will take somewhere around 3-4 hours.

3.4 Making a model

The conversion process is a multistep process in a program called Meshlab. The reason to do the conversion is because dense point clouds will contain 0.5 - 2 million vertices depending on the size of the building. With 7 buildings having over 7 million vertices in total to render would make the scene only possible to render on higher end computers. In order to make the models easier to render we have to turn it into a mesh. This makes it easier because the graphics card only needs to compute the position, transform for one face instead of the number of vertices that make up this face. This dramatically reduces the computational time per building.

Steps to convert a point cloud to a mesh are:

1. Clean up the point cloud.
2. Use a meshing algorithm to convert the point cloud
3. Giving the mesh a texture.

3.4.1 Cleaning up the point cloud

The resulting point cloud from VSFM will have a small amount of irrelevant data that is generally from other buildings near by our main building that the Structure from motion software has picked up. Since we dont need this in the final mesh we need to remove it. This is a simple process of selecting the data and deleting it.

3.4.2 Meshing the point cloud.

Now that we have a clean point cloud to work with we need to convert it into a mesh. There are many different re-meshing algorithms that Meshlab offers. The three re-meshing algorithms that are available in Meshlab are Ball pivoting, Marching cubes, and Poissons reconstruction. The main algorithm that will be used is Poisson Algorithm. The reason why this was the algorithm of choice was it was resilient to noise, it maintains detail in the reconstruction and is more efficient in memory than other reconstruction algorithms.

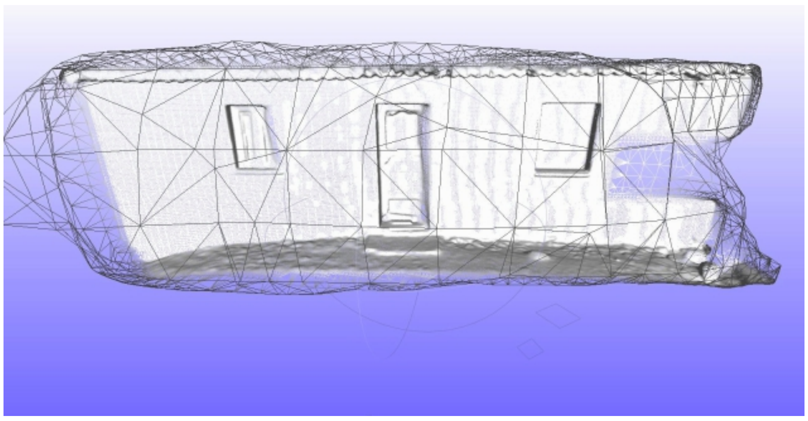


Figure 3.4: Meshing a point cloud

The Poisson Algorithm has a few settings that can be changed to improve the detail in the re-meshing. The two main settings are octree depth (higher the value greater precision in the mesh) and solver divide (reduces memory usage with an increase in time). From experiments a value of 8 for both of these settings improves the mesh detail without using up all the memory on the computer.

3.4.3 Giving the mesh a texture

At this moment the mesh does not have any texture assigned to it so it look like a grey blocky mesh. To do this the mesh needs to be given texture co-ordinates so it knows where the textures are to be placed. After that has been done you use a function in Mesh lab to transfer attributes from the point cloud to the mesh. This merges all the points from the point cloud onto the mesh giving it the colours from the images. The final result is a 3D building from the photos.

3.5 Creating the scene

One of the problems of using Structures from Motion is that every model that is produced is at a different size than before. This means that you cannot simply just place the buildings in a scene and render it. They must first be edited to make sure that they are at relative size to each other. This has been accomplished by using a tool called Blender. Using Blender the meshes are firstly imported, then scaled and finally are transformed so that they are in the right place and look like the campus.

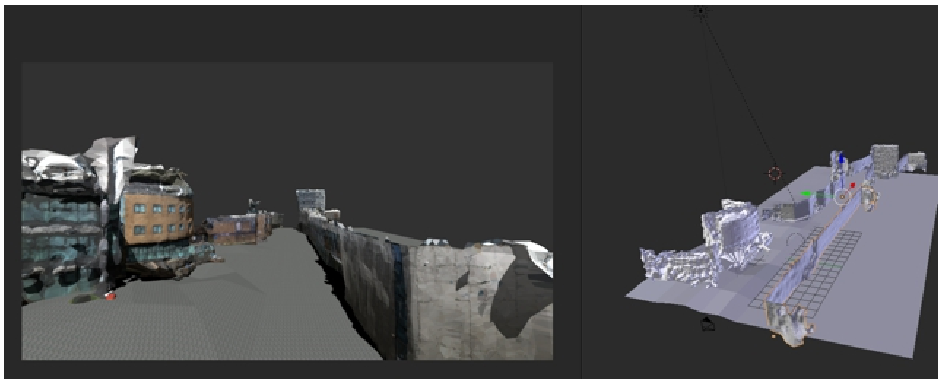


FIGURE 3.5: Composing a scene in blender

Since we are in a modelling program custom models can be imported to make the campus feel more realistic. A simple floor can be added and a pavement texture can be assigned to it. Also trees and other foliage can be added which adds more life to the campus.

For use in the jMonkeyengine SDK the best format to export is ORGE XML,a special scene file that contains all the transform rotation and scale values for each of the meshes. In the ORGE exporter it is necessary to convert the orientation setting. This is because Blender and jMonkeyengine use different co-ordinate systems. If there was not a conversion the campus would look very different in jMonkeyengine than in Blender.

Chapter 4

Developing the Application

4.1 Basic Applications

The JMonkeyengine SDK and the IDE is essentially used for designing games, but does mean it has all the libraries already integrated in for viewing and manipulating objects in 3D. A further advantage is that the language for programming in this IDE is Java. I have a broad knowledge of Java from Software engineering in Level 2 and programming 1 and 2 in Level 1. This means developing the application will not be limited by inexperience in the language. My first step in this environment was to explore the hello world program that is available. This program would build a cube, which you can move around with 1st person controls I.E keys WASD and the mouse pointing the direction. I played around with changing the control speed and adding a large light source.

```java
// libiraies need
import com.jme3.light.AmbientLight;

/* To create lightsource and add to the canvas */
AmbientLight al = new AmbientLight();
al.setColor(ColorRGBA.White.mult(5f));
rootNode.addLight(al);
```

The next step of the program was to try and import a model I had made. I used the IDE import, to import the model and used the following code to place the model into the view.

```java
// libiraies need
import com.jme3.scene.Spatial;
import com.jme3.app.SimpleApplication;

class Veiwer extends SimpleApplication {
@Override
  public void simpleInitApp() {
    Spatial fulton = assetManager.loadModel("Models/save-mesh/save-mesh.obj");
    rootNode.attachChild(fulton);

  }
}
```

I noticed that the output was very dark so from the first code I added a large omnidirectional light source that improved the output. I also experimented with the player speed to make the movement around the object fast enough. At the moment the program is just a full screen with no controls.

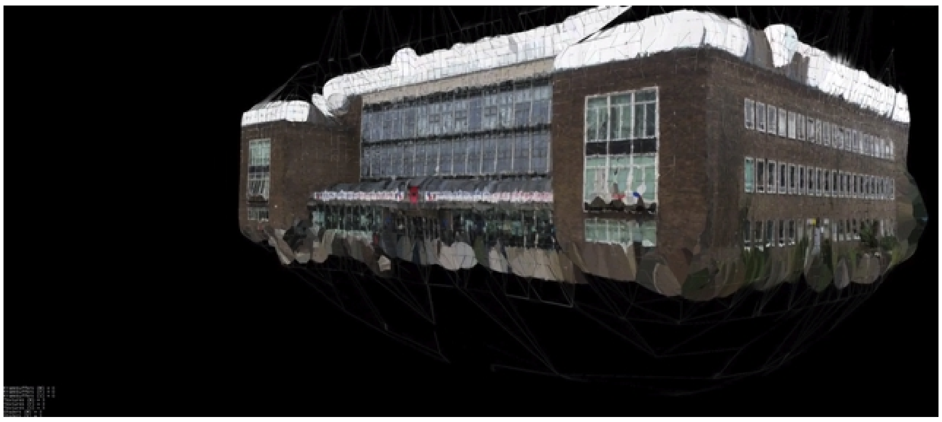
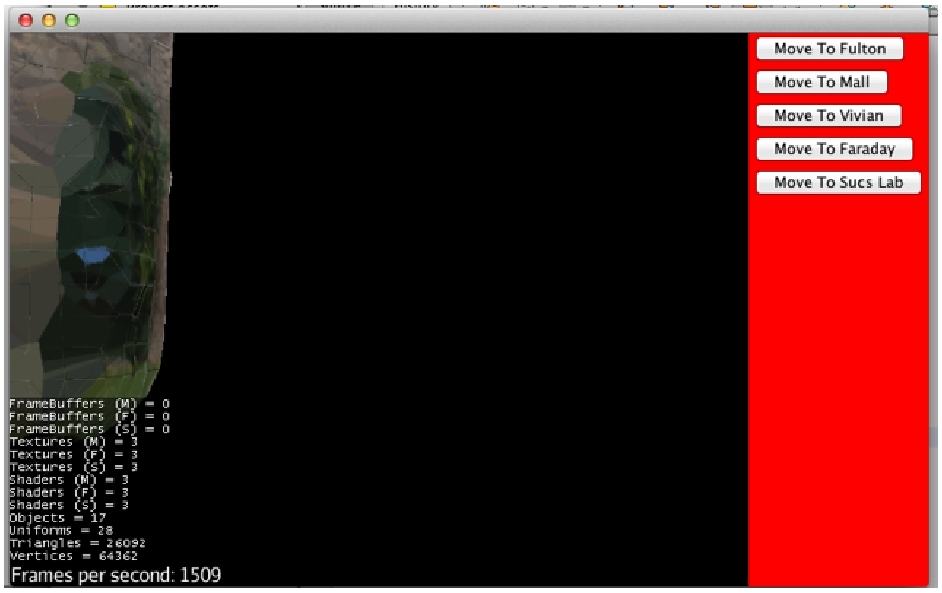


FIGURE 4.1: Screenshot of the program running

4.2 Advanced view

After getting one single building to be viewed the next step is multiple buildings. My first attempt to do this was to have a simple UI on the left from which user selects a building, for example Fulton house and the building is displayed to the user.



```
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 3
Textures (F) = 3
Textures (S) = 3
Shaders (M) = 3
Shaders (F) = 3
Shaders (S) = 3
Objects = 17
Uniforms = 28
Triangles = 26092
Vertices = 64362
Frames per second: 1509
```

Figure 4.2: Screenshot Version 1 with options to change model

Although the application is simple for the user to use it does not give them the experience I wanted, as it does not show the whole campus. To do this all the buildings would have to be loaded at the same time. To move to the different buildings in the campus a version of navigation would have to be implemented and a User Interface is needed to make it clear to the user how to use the program.

4.2.1 Importing the scene.

JMonkeyengine has very good blender integration, this makes importing a scene from the Blender modelling program easy. In Blender you export as an ORGE XML file. This keeps each building as an obj format but also has a scene file that contains all the buildings translate, rotation and scale operations. In JMonkeyengine the ORGE XML can be converted into a jm3 binary file that has all the models and scene information in one file. Once converted using it in the program is as simple as a couple of lines of java code.

```
// libiraies need
import com.jme3.scene.Spatial;
import com.jme3.app.SimpleApplication;

class Veiwer extends SimpleApplication {
@Override
  public void simpleInitApp() {
    Spatial scene = assetManager.loadModel("Scenes/Scene6/small.j3o");
    rootNode.attachChild(fulton);

  }
}
```

Since in Blender we set up lighting and have assigned textures the model looks identical to what it did in Blender

Using jMonkeyengine default camera setup the user can already navigate around the campus using the default game key controls WSAD (W - move forward, S - move backwards, A - move left, D, move right) and use the mouse to look around.

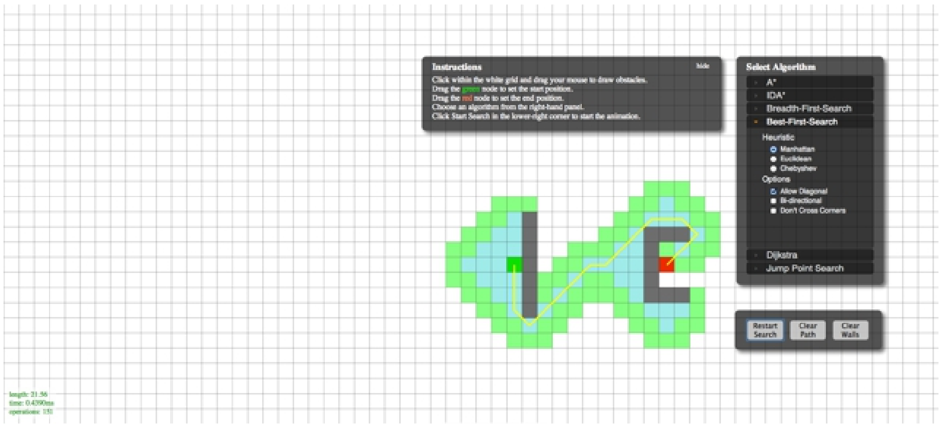
If a user is not familiar with these controls it would make moving around the campus very difficult so an alternative is needed.

4.2.2 Navigation

As previously discussed a way of moving around the campus is needed where the user is not in control of the movement and only controls the destination they want to go. The simplest method of doing this is just having a set of co-ordinates, the co-ordinate of the starting point and another for the destination. Using these two coordinates we can just simply create a vector between them and move along that vector. This would create a straight-line movement from the initial positions to the destination. It would go through the buildings since the mesh does not have any collision (All objects can pass through the mesh as if it does not exist). This creates a very unnatural movement and is not the sort of movement the user expects.

The simple movement of moving in a straight line is unnatural so we need a way of moving around buildings. This has been accomplished in video games using a method called path finding. It is used to find the shortest path between two points in things like mazes. There are many path-finding algorithms available which mostly split up an area into tiles or polygons and put them into a graph and use graph search.

To test the performance of these algorithms there is a useful website that demonstrates how the most popular algorithms work. http://qiao.github.io/PathFinding.js/visual/


Instructions

Click within the white grid and drag your mouse to draw obstacles.
Drag the green node to set the start position.
Drag the red node to set the end position.
Choose an algorithm from the right-hand panel.
Click Start Search in the lower-right corner to start the animation.

length: 21.56
time: 0.4390ms
operations: 151

FIGURE 4.3: Navigation map that the tests were run on

	Length:	Time	Operations
A*	19.07	6.3374ms	339
Breath first search	19.07	8.9512ms	1580
Best first search	21.56	2.5601ms	151
Dijsjker	21.56	2.5601ms	151
Jump point search	19.07	8.7899ms	4692

From experiments the best algorithm is A* because it computes the shortest path the quickest and with a reasonably small number of operations. This why it is used in most path finding algorithms

4.2.3 Implementing Navigation

The first step in implementing navigation is to create a mesh that can be used in the graph search; this is commonly called a navmesh. Although JMonkeyengine has support for making a navmesh I found that it did not work well detecting the buildings. Blender also has support for making a navmesh. In Blender I created the navmesh, I then edited the navmesh removing areas where the user should never be able to go (for example behind or inside the buildings). Once the navmesh was made it was exported along with the rest of the scene and imported into jMonkeyengine.

Now we have a navmesh that can be used for navigation we need the navigation library. The library is not in the initial install of jMonkeyengine but can be found in the plugins section.

The path finding library needs the geometry of the navmesh in order to compute the shortest path around the buildings. JMonkeyengine loads all the models into a tree object with a root node at the top that contains the scene information. So to get the navemsh we need to find which node in the tree it is and then get the geometry of the mesh

```
Node submesh = (Node) rndr.getRootNode().getChild("small-1-ogremesh");
        System.out.println(rndr.getRootNode().getChildren());
        Geometry g = (Geometry) submesh.getChild("_missing_material_");
```

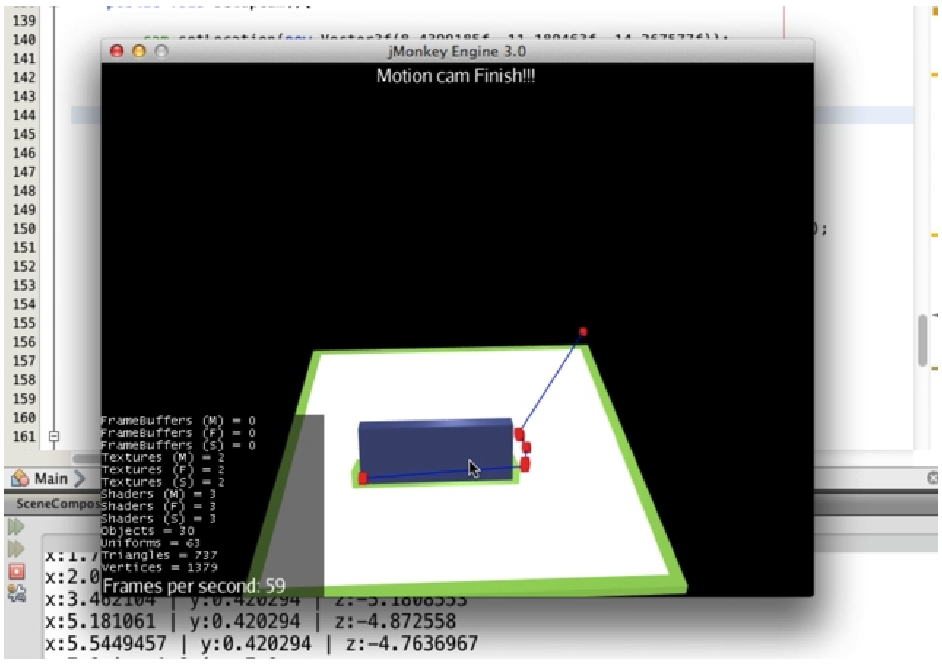
After getting the geometry we need the start position and destination. This produces a path that contains a series of waypoints that show the route around a building.

```
Path pathl = new Path();
    boolean buildNavigationPath =
    navmesh.buildNavigationPath(pathl,navmesh.findClosestCell(init), init,
    navmesh.findClosestCell(end), end, 0.4f);

    if (buildNavigationPath) {
        MotionPath path = new MotionPath();
        path.setCycle(false);

    for (Path.Waypoint p : pathl.getWaypoints()) {

        Vector3f vf = new Vector3f(p.getPosition().x,1.2f,p.getPosition().z);
        path.addWayPoint(vf);
    }

    paths.put(name, path);
```

Now that we have a shortest path from the initial positions to the destination. To use this path we have to use a different camera setup than the default one JMonkeyengine gives you. We define a camera node that will contain a chaser camera and a motion event. These two objects allows for the camera to follow a path and detect when the camera has finished the path.

```
public void setupCam(String key){
       path = paths.get(key);
       final Vector3f last = path.getWayPoint(path.getNbWayPoints()-1);
       camNode = new CameraNode("Motion cam", rndr.getCamera());
       camNode.setControlDir(CameraControl.ControlDirection.SpatialToCamera);
       camNode.setEnabled(false);
       path.setCurveTension(0.15f);
       cameraMotionControl = new MotionEvent(camNode, path, 10f);
       cameraMotionControl.setLoopMode(LoopMode.DontLoop);

       cameraMotionControl.setLookAt(rndr.getScene().getWorldTranslation(),
   Vector3f.UNIT_Y);
       cameraMotionControl.setDirectionType(MotionEvent.Direction.Path);

       rndr.getRootNode().attachChild(camNode);

       path.addListener(new MotionPathListener() {

           public void onWayPointReach(MotionEvent control, int wayPointIndex) {
               Vector3f vl = last;
               if (path.getNbWayPoints() == wayPointIndex + 1) {
                       rndr.getRootNode().detachChild(camNode);
                       cameraMotionControl.stop();
                       rndr.getFlyByCamera().setDragToRotate(true);
                       rndr.getModels().updatelod();

               }
           }
       });
       chaser = new ChaseCamera(rndr.getCamera(), rndr.getScene());
       chaser.setLookAtOffset(Vector3f.UNIT_Y.add(0, 10, 0));

       chaser.registerWithInput(rndr.getInputManager());
       chaser.setSmoothMotion(true);
       chaser.setMaxDistance(50);
       chaser.setDefaultDistance(50);
       chaser.setDragToRotate(true);
   }
```

Once the path is complete an event can be triggered which can display information to the user.



```
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 2
Textures (F) = 2
Textures (S) = 2
Shaders (M) = 3
Shaders (F) = 3
Shaders (S) = 3
Objects = 30
Uniforms = 63
Triangles = 737
Vertices = 1379
Frames per second: 59
```

```
x:5.181061 | y:0.420294 | z:-4.872558
x:5.5449457 | y:0.420294 | z:-4.7636967
```

Figure 4.4: Small Demo of the navigation

4.2.4 User Interaction

To get the navigation to work the user needs to be able to select a destination, which means a UI has to be in place. There are 2 main options for UI implementation, the first is using Java swing UI library. This is what I did for the basic demo where a user could select a building they wanted to view. The main problem of using this is that you have to separate the UI from the OpenGL renderer. The alternative method is to use the Nifty GUI library. This loads inside the renderer so it feels more fluid to the user to interact with the program. It also offers features such as heads up display (HUD) where information can be shown to the user as part of the scene.

I designed the various views in a program called wireframe sketcher studio [18], I used it to quickly mock up designs that I wanted to have. I designed the main menu, the heads up display and map overview.

The Nifty GUI library defines user interfaces as an xml file. In the XML file the UI is split up as a <screen >that defines the different views. Inside the screen tag you can define various panels that can be nested and positioned in order to build up the user interface. Inside the panel tag you can define the various user interface controls for example buttons, text, list-boxes etc. All these tags build up the user interface. To load the Nifty GUI file is a simple process of

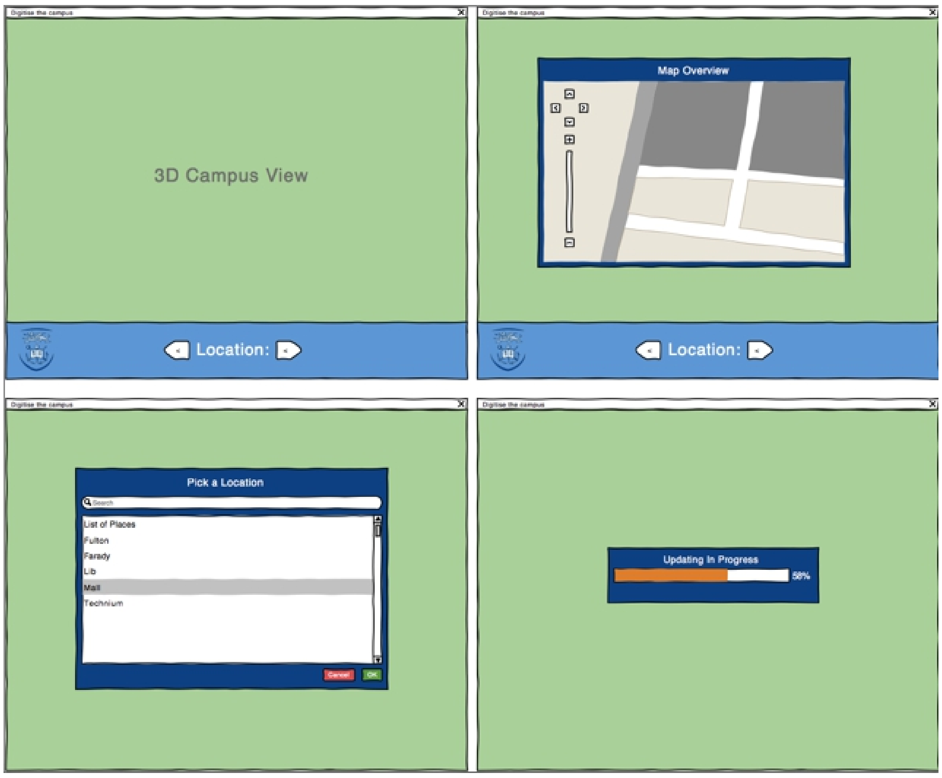


FIGURE 4.5: Design of the User Interface

initialising the display and loading into the file from the asset manager.

```
   NiftyJmeDisplay niftyDisplay = new NiftyJmeDisplay(rndr.getAssetManager(),
                                                         rndr.getInputManager(),
                                                         rndr.getAudioRenderer(),
                                                         rndr.getGuiViewPort());
        nifty = niftyDisplay.getNifty();
        nifty.fromXml("Interface/newbutton.xml","run", this);

         // attach the nifty display to the gui view port as a processor
        rndr.getGuiViewPort().addProcessor(niftyDisplay);
```

To detect whether a user has interacted with the interface we need a screen controller. Creating a separate java class and implementing the screen controller interface do this. In the Nifty Xml file you tell it which class is going to be its screen controller, as you can have multiple screen controllers per program.

```
<screen id="start"controller="mygame.UiController">
```

Now that the screen knows which class is going to be receiving the user inputs. All buttons need some code that gets triggered when the user clicks on the button. The code is stored as a method in the controller class but we need to tell the button which method to use. To do this you add an extra tag in the XML file. This tag tells the button which method to run when the button is clicked.

```
<control id="buttonOk" name="button" label="OK">
                        <interact onClick="select()"/>
                    </control>
```

To detect action events for example when the list box has been changed we have to add Nifty subscriber events to the program. These events are triggered when a user does something with the program, for example click on something in a list box or types text.

```
@NiftyEventSubscriber(id="search") //subscriber event for the control with an id of
    search


    public void onTextChanged(final String id, final TextFieldChangedEvent event)
    {
        String searchText = event.getText();
        System.out.println("text:"+searchText );
        Screen screen = nifty.getScreen("start");
        ListBox listBox = screen.findNiftyControl("myListBox", ListBox.class);
        listBox.clear();
        List<PlaceData> Searchlist = placeData.Search(searchText);
        if(searchText.equals("")){
            listBox.addAllItems(placeData.getList());
        }else if(Searchlist.size()>0){
            listBox.addAllItems(Searchlist);
        }

    }
```

4.2.5 Program data.

The program needs some extra data that the models don't give. This sort of information is about the building name. The information about a building can include: description of the building, the departments that the building contains and the position that building is in the model. This information is defined as a separate xml file. This information will be displayed in the programs list box and will be used in the program for the navigation.

```
<root>
    <place>
        <placeName>Library</placeName>
        <placeDesc>Place to do research</placeDesc>
        <xcordinate>-144.7875</xcordinate>
        <ycordinate> 2.011348</ycordinate>
        <zcordinate>22.18915</zcordinate>
    </place>
<\root>
```

When the program loads, this xml is also loaded into a custom database. This is used when a user searches for a department; it looks through the database and returns the building that department is in. When a user selects a building from the list, the position that the building is in the scene is given to the navigation. The file is stored in the asset folder and can be retrieved by the asset manager. The problem is that the asset manager does not know how to read an xml file. Creating a separate class and defining our own asset loader can resolve this. The asset loaded using the DOM (document object model) will read the xml into a simple custom database that will be used in the rest of the application.

4.2.6 Admin Section

For the xml to be made, a small admin section has been built that can be activated through a command line argument. This section is used to build the database and export the xml file. The admin ”flies around” adding a new building and filling out a form about the building. Once all the buildings in the scene have been added the database can be exported as an xml file, to be used in the main program.

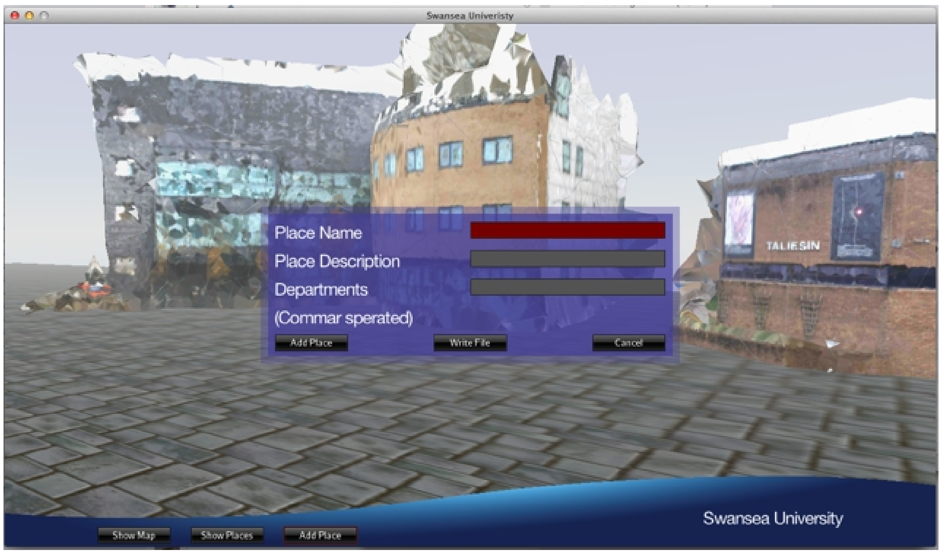
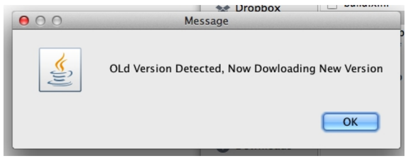


FIGURE 4.6: Admin section running

4.2.7 Auto Update.

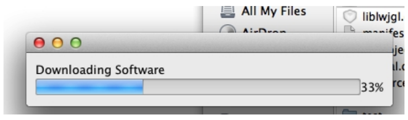
Consideration needs to be made for new buildings or improved buildings that could be added to the scene in future. So that the user gets the most accurate representation of the campus every time they load the program a way of auto updating is needed.

This has been achieved by making a small program that the user will initially get and run each time. This small program will only check to see if there is a new version available each time it runs if appropriate, it will then download the new version and run it. This has been used by several companies in order for their users to get the latest version of the program, for example Mojang for their Minecraft game. The program works first by checking to see if there


Message

OLd Version Detected, Now Dowloading New Version

FIGURE 4.7: Detecting Old version


Downloading Software

33%

FIGURE 4.8: Dowloading newer version

is Internet. If so it checks the server for the latest version code. If the code does not match the version on the computer it downloads the zip file, unzips it and runs the code. If there is no internet or the version is not newer then it will run the local version.

4.3 Optimisations and improvements

4.3.1 Mesh Optimisation

As the scene was built up in JMonkeyengine and was being tested a noticeable problem was appearing. As more buildings were being added the performance was dropping. After all the buildings had been added to the scene the program performance had reduced, at some parts, to a frame rate of 11 fps (frames per second). At this low frame rate the user will notice very jerky video.

To improve this performance we need to reduce the mesh complexity. There are too many faces per model being rendered. In Meshlab, there is a useful algorithm quadratic Edge Collapse. This algorithm will reduce the number of faces in the mesh. In the field of graphics it has been shown that you can compensate for a lack of detail in the mesh by increasing the detail in the texture. This makes is easier for the GPU to render while the user sees no noticeable difference.

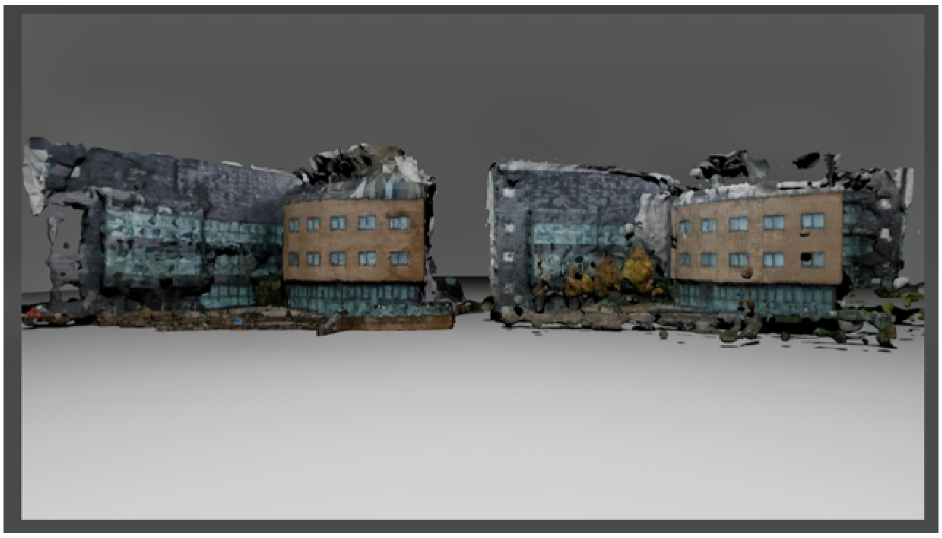


FIGURE 4.9: Mesh Comparision

The figure above (figure 4.9) shows my work. The building on the right is the old model of 16mb (1/2 million lines) and a 1024,1024 texture, the building on the left is the new one 3.5MB(100,000 lines) but with a 4096, 4096 texture. As you can see the quality looks the same in both images

By doing this dramatic reduction in mesh complexity the performance of the software improved so the user does not notice a large fps drop. Also by reducing the mesh size and by compressing the image from a high quality png to a compressed jpg image I have been able to reduce the memory consumption of the program. It reduced from 3GB of memory usage to only 1.2GB of memory.

4.3.2 Dynamic Culling

Even with the mesh reduced in size the application was still not performing as smoothly as I would like. Most renders have a function called culling. Culling is the process of not drawing faces when they are not viewable from the camera.

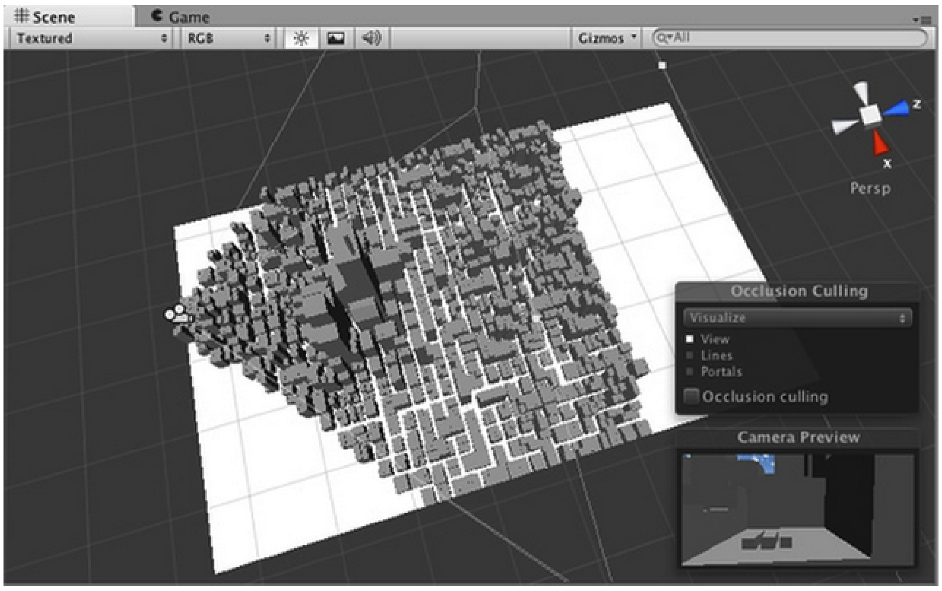


Figure 4.10: Culling diagram

JMonkeyengine does this, it will only render one side of a face cutting the number of faces per model down by half. The main problem is that it will still render the model even though it will never be seen by the user i.e. the model is behind the camera. Also some models will dynamically change the number of faces depending on how near the camera is, for example if the camera is far away (but still visible) from the camera, it does not need all the detail as if the camera is nearby. Due to how the models are created and the limitations in the SDK this dynamic change cannot be implemented

A solution is to build a custom culling function. My solution is a bit drastic but simpler to implement. Depending on where the user is in the scene, certain buildings are set not to be displayed and will only be displayed when the camera is in a position that the user could see the building.

```java
public void updatelod(){
       nav = root.getNav();
       Vector3f pos = root.getCamera().getLocation();
       String build = "";
        for(int i = 0; i< nav.getPlaceData().size();i++){
            if(pos.distance(nav.getPlaceData().get(i).getCo_ord())<=6){
                build = nav.getPlaceData().get(i).getName();
            }
        }
       if(build.equals("Libuary")){
           setLOD("technium",false);
           setLOD("faraday",false);
           setLOD("liblaw",false);
           setLOD("lib",false);
           setLOD("kehir",false);
           setLOD("tal",false);
           setLOD("fulton",true);
       }
  // set data for other models in the scene
}
public void setLOD(String key, boolean cull){
      Spatial obj = data.get(key);
      if(cull){
          obj.setCullHint(Spatial.CullHint.Always);
      }else{
          obj.setCullHint(Spatial.CullHint.Dynamic);
      }
      data.put(key, obj);
   }
```

This dramatically improves performance because the render has fewer buildings to render at any one time. The costly operation is redrawing the building back in that can take at most a second to do. This does cause a small artefact that the user may see a building disappear or reappear as they move close to the building.

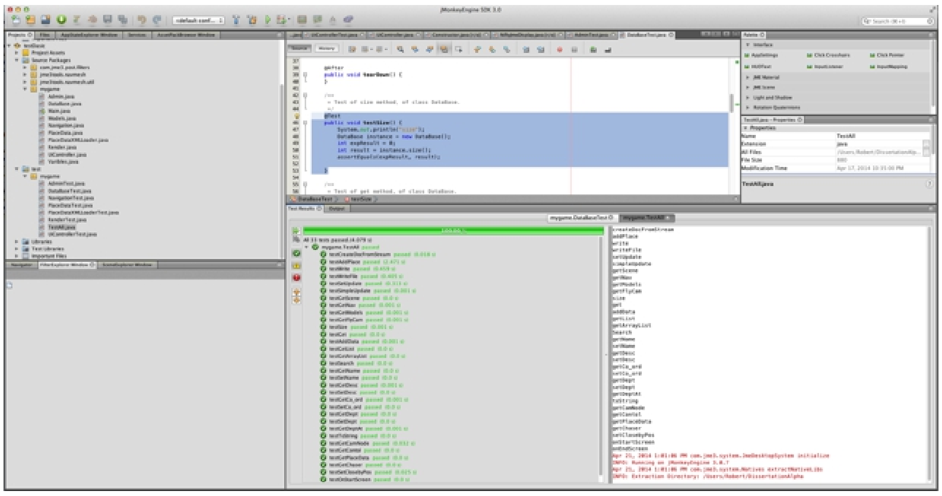
4.4 Testing

Testing the program was a two-fold method. The first test was the programs code. This was to check that the code I wrote did what I expected it to do. The approach I employed was to use simple black box testing and use boundary value analysis form testing. This method just checks the maximum and minimum results for each function. Since the program is written in Java, the IDE supports jUnit.

JUnit is a library that is used to test programs. To use it in a separate class you encode your test cases into jUnit test with functions like assert Equals (expected result) where you check whether your expected result matches up with the actual result of the program.

```
@Test
    public void testSize() {
        System.out.println("size");
        DataBase instance = new DataBase();
        int expResult = 0;
        int result = instance.size();
        assertEquals(expResult, result);

    }
```

The benefit of using jUnit is that you can encode all your test cases and run the same ones each time you change the function.



```
createDocFromStream
addPlace
write
writeFile
setUpdate
simpleUpdate
getScene
getNav
getModels
getFlyCam
size
get
addData
getList
getArrayList
Search
getName
setName
getDesc
setDesc
getCo_ord
setCo_ord
getDept
setDept
getDeptAt
toString
getCamNode
getControl
getPlaceData
getChaser
setClosebyPos
onStartScreen
onEndScreen
Apr 21, 2014 1:01:06 PM com.jme3.system.JmeDesktopSystem initialize
INFO: Running on jMonkeyEngine 3.0.?
Apr 21, 2014 1:01:06 PM com.jme3.system.Natives extractNativeLibs
INFO: Extraction Directory: /Users/Robert/DissertationAlpha
```

FIGURE 4.11: Full Pass of all tests run on the program

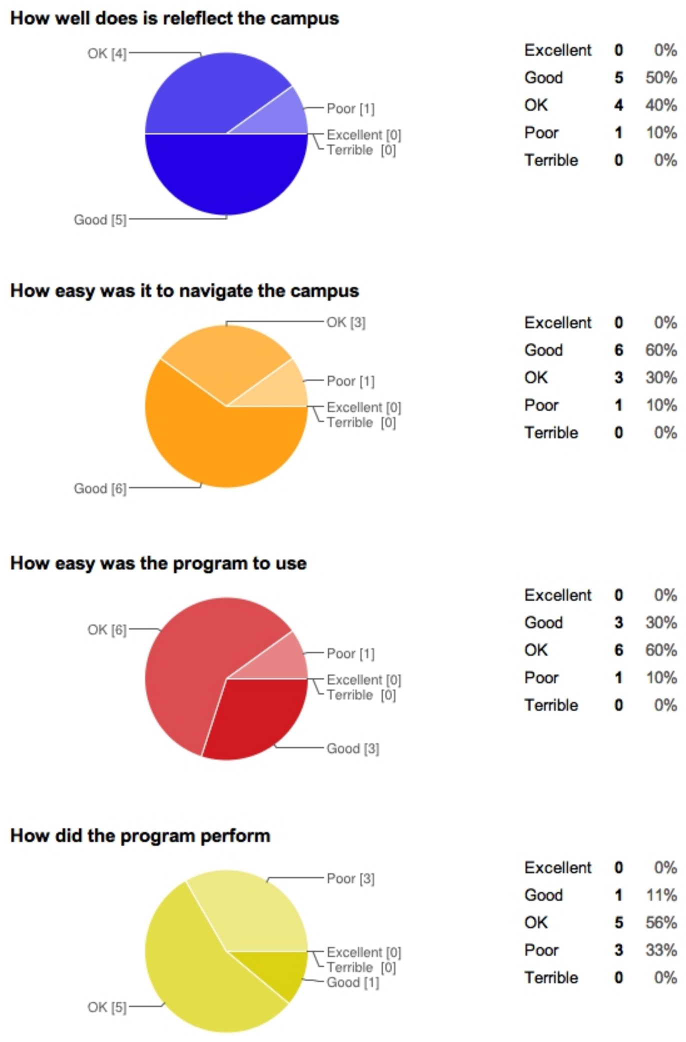
The second approach was to test the running code as jUnit cannot test the performance of OPenGL. By running the program on various hardware; including Desktop PC, 2013 Laptop and 2009 Laptop. I could measure the performance on older hardware.

4.4.1 User feedback

Along with testing the program I also carried out a user feedback. This was beneficial because I noticed how different users interacted with the program. As the developer of the software you can overlook small bugs in the program.

The feedback was conducted by giving the user access to the program for 5 minutes and observe them interacting with the program. After the 5 minutes were up they filled out a small feedback form.

The feedback form consisted of 5 simple questions about the program. These were: How well does it reflect the campus? How easy was it to navigate the campus? How easy was the program to use? How did the program perform?



Excellent	0	0%
Good	5	50%
OK	4	40%
Poor	1	10%
Terrible	0	0%

Excellent	0	0%
Good	6	60%
OK	3	30%
Poor	1	10%
Terrible	0	0%

Excellent	0	0%
Good	3	30%
OK	6	60%
Poor	1	10%
Terrible	0	0%

Excellent	0	0%
Good	1	11%
OK	5	56%
Poor	3	33%
Terrible	0	0%

FIGURE 4.12: User Feedback results

The results were interesting both from observing the participant and the results from the survey. 90% [19] said that the application did reflect what the campus looked like and was easy to navigate and use the program. What people commented on was the large load time. The program takes around 1-2mins to load.

4.5 Unity3D

Unity3D is an alternative IDE similar to jMonkeyEngine. Unity3D has been in development for a lot longer the JMonkeyEngine. This means that the environment is a lot more stable. Another difference is that the renderer that Unity3D used is written in C++. By using the C++ language they are able to get the renderer highly optimised for every platform they support. This means that the simpler applications don't need to use large amounts of memory or CPU usage. Unity3D is based upon using components; each component can be a script, a mesh, or prefab (a set of other components).

After the user review and during testing I knew that I was compromising the quality of the scene to try and improve the performance of the program. While continuing to develop the application I started doing small tests in Unity3D to see how difficult it would it be to move all developing to this platform. The tests were small separate applications that tested model imports, a navigation test and a UI test. The entire test came back very encouraging. The main problem is that the language that Unity3D uses is C# which would mean a complete re-write of all the code. The advantage is that since Unity3D has been in development for a long time a lot of functions have been developed for example, it has a very easier to use navigation system along with many functions used to optimise the mesh.

4.5.1 Developing in Unity3D

Importing the scene is even simpler than jMonkeyengine. Unity3D is able to read the Blender scene file so no export and conversion is needed. You tell unity3D where the Blender scene file is and imports it.

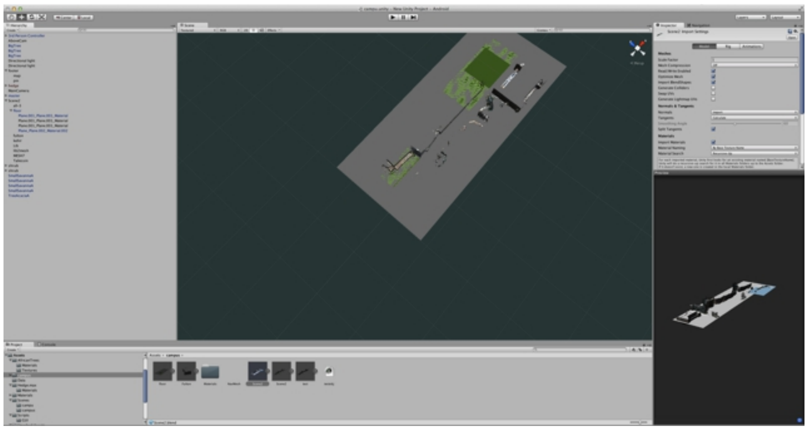


FIGURE 4.13: Unity with the imported blender scene

Navigation is done in Unity3D. In the navigation settings of Unity3D it can generate a navmesh. This will be better optimised for features inside of Unity3D than using the navmesh from Blender. It spends a couple of minutes analysing the scene and generating a highly optimised navmesh. To navigate on this new mesh we need a navmesh-agent. Navmesh-agent is a component that has in-built functions that perform path finding and navigation. On a prefab that contains a camera you add the navmesh agent. On the same component you attach a script that will control the navigation.

```
if (GameObject.Find ("3rd Person Controller").GetComponent <NavMeshAgent>().enabled
    ==true) {
                                                GetComponent<NavMeshAgent>
    ().destination = newLocation;
                        }
```

The code gets the navmesh-Agent on the prefab and then it tells it the destination you want to go. The destination can be dynamically changed at any time. When the destination has changed the prefab moves to that destination and will stop when it arrives at the destination.

The only complication in re-writing the code is re-doing the user-interface. Unlike jMonkeyEngine Unity3D does not define layouts in XML but instead it defines them in code. To redo the user interface I created a new C# script and attached it to the prefab that contains the camera. In the new script I designed three new screens with dynamic size and positions depending on the size of screen. This means that a window will be in the centre of the screen no matter which device it is on. The other complication is that unity (or Unity3D) has not got List box view, therefore a custom view had to be designed that will work similar to a list view.

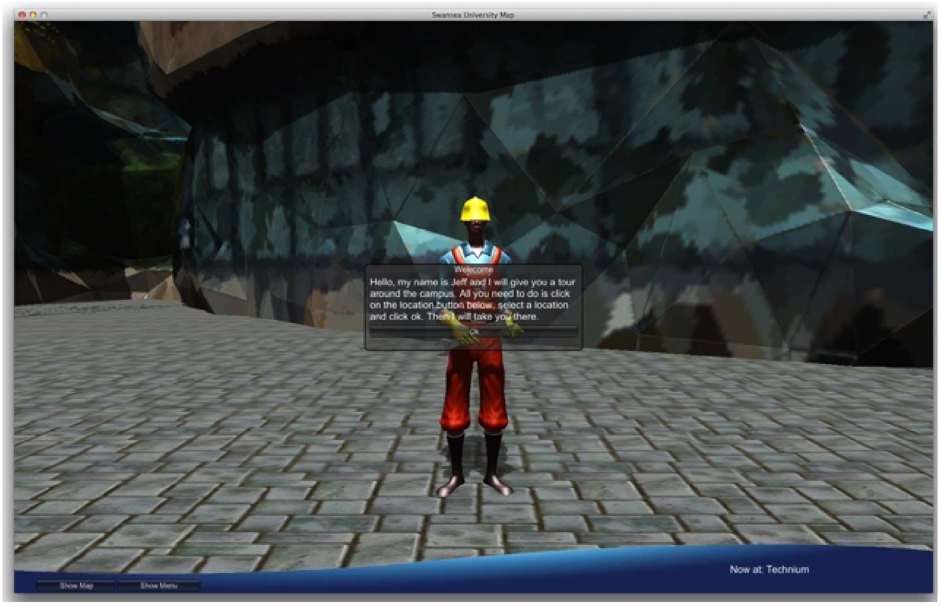
It consists of a series of buttons and a scroll view. When the button is pressed the scroll view collapses and the first button is the last selected item

```
public void Start(){
        windowRectangle = new Rect(Screen.width/2-200,Screen.height/2-80,400,160);
    //set up GUI window size
}

private void OnGUI(){
                GUI.skin = mySkin;
                if (message) {
                        messageRectangle = GUI.Window (0, windowRectangle,
    beginMessage, title);


        }

        private void beginMessage(int id ){
                GUILayout.Label (messages);
                if (GUILayout.Button ("Ok")) {
                        GameObject thisCam = GameObject.Find("MainCamera");
                        GameObject followCam = GameObject.Find("FollowCam");
                        GameObject aboveCam = GameObject.Find("AboveCam");

                        Debug.Log(aboveCam);
                        if(aboveCam !=null){aboveCam.SetActive(false);}
                        if(thisCam !=null){thisCam.SetActive(false);}

                        followCam.SetActive(true);
                        message = false;
                        messageRectangle.Set(0,0,0,0);

                }

        }
```

In developing the new unity application I came across one complication that is the way the scripts communicated with each other. With JMonkeyEngine all the different classes are instantiated and various pointers to the other objects are passed to it. Whenever a class needed to do something, for example, set destination, it would simply call the method in the object. In Unity3D there is no way to pass pointers to the other scripts the program holds. To communicate, the function has to be static this tells unity that other scripts can call it.

```
\\Main Gui
Follow.updateLocation(new Vector3(x,y,z), element.placeDesc,slectedItem);

\\ in the Follow Class

\\note Static means accessible from other seperate classes
public static void updateLocation(Vector3 newPos, string desc, string placename){
                        newLocation = newPos;
                        GameObject followCam = GameObject.Find ("FollowCam");
                        GameObject jeff = GameObject.Find ("3rd Person Controller");
                followCam.transform.rotation = jeff.transform.rotation;
                title = placename;
                description = desc;

                }
```

Unity3D offers many features, one of which is 3D character prefab. This is a 3D mesh with all animations for walking, jumping etc. already done. By using this prefab I was able to change the program slightly from a 3D navigation program into a customised tour application. In the XML that stores place locations I added extra information that describes what each building is, for example Fulton house is the home to the student union catering and transport facilities. This small change to the program makes it easier to use by the user and gives more information about the campus than simply the location of where the building is.



Welcome
Hello, my name is Jeff and I will give you a tour around the campus. All you need to do is click on the location button below, select a location and click ok. Then I will take you there.

Now at: Technium

FIGURE 4.14: Unity3D Working

4.5.2 Making the Scene more realistic

One problem with the scene is that it will be always incomplete because it is not feasible to create a 3D model of the surrounding area. To make the scene more realistic to the real campus, the first thing to do is to add models of trees and plants to the campus. Importing the default tree in the unity engine is a good start. The tree can distract the user from any imperfection in the models.

Another thing to improve how realistic the campus is, is to add borders to the scene. These borders will first stop the user walking beyond the scene. Also these borders will have a texture that represents the background to a campus. For example the south border will have a texture of Swansea bay so that when the user stands in front of Fulton house they will see the same view as if they were standing in front in real life.

4.5.3 Android

Since Unity3D has been in development for many years it has been able to write its renderer on many different platforms including Android. This means that developing a mobile version of the application has been made very simple. When the main application of the program has been done in the build settings you can create the application package for android. In the build settings you can change certain aspects, for example what the loading screen will look like and the icons. There is another format which is to create an Android project that can be developed in Android studio if there is a case that you need to add some custom code for the Android application.

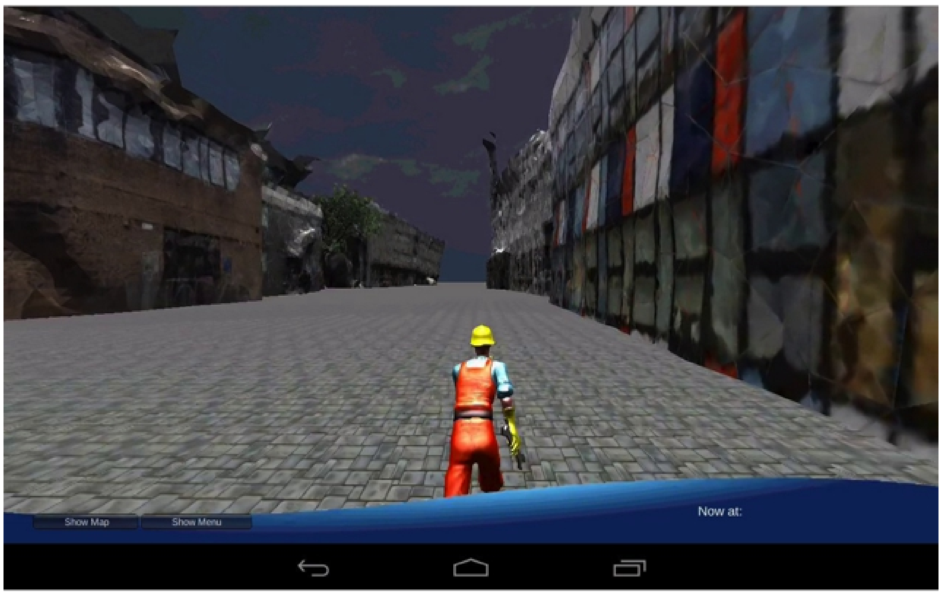


FIGURE 4.15: Unity3D Android

4.5.4 Performance improvements

After finishing developing the new application in Unity3D I tested the program performance.

	Start-up time (Seconds)	Memory usage (MB)	CPU usage
JMonkeyEngine	21	1800	30 50%(23 Threads)
Unity3D	10	126	1.2% (17 Threads)

As you can see from the test the Unity3D application is 50% faster to load the scene and a massive 93% reduction in memory utilisation. Due to the efficiency in the performance the Unity3D program can handle the more complex mesh that was first developed. This results in a scene that has much higher detail and therefore looks closer to the true view of the campus

Chapter 5

Conclusion

5.1 Limitations

This application is currently limited because not all of the buildings have been photographed, so only a portion of the Singleton Park campus is included in the current scene.

Another limitation is in the meshes that make up the scene. The way Structure from Motion works is very good at getting the geometry of a building but falls short at applying the texture. The texture gets made up from the point cloud, this makes it look fine when the building is far from the camera but when the camera gets close to a building it ends up looking like an artists watercolour of a building, all the features are blurred out. From viewing how other people interact with the program it is very interesting how certain actions in other programs have been ingrained in a person.

The use of Google Street View and the User Interface that that application presents is so familiar that users will try the same action in a different program. It occurred when a user used the map feature that shows a 2D representation of the campus and a pin marking their location. The user instinctively tried to click on a chosen building and expected to be automatically transferred there.

5.2 Future Work

The dissertation can take many different directions. One direction is to improve the models captured. The best approach would be to use Structure from motion (SfM) to get the basic geometry of the building. The geometry would be simplified to give large faces to apply the texture. The texture would be made from the photos rather than the point cloud. This would remove the blurry aspect of the modules when the model is closer to the camera.

To be able to get the most accurate models of the campus we need a full capture of a building. Due to the layout of the campus the only way of achieving this is to attach a camera to a helicopter. The best solution would be to use a quad-copter(aircraft with 4 blades) which can lift a heavier object. This means we can use a stabilised camera resulting in the capture of very high quality photos. Also with modern quad-copters we can use the on board controller to fly a pre-programed route around a building.

Other work that can be done with the application is a change in the user interface. The biggest improvement is to improve the map overlay. At the moment it is only used for an overview of the campus and giving the user a reference of where they are. Combining the map and selecting the destination would be beneficial because it makes the user interface more natural to a user who has been using applications like Google maps.

As the scene gets larger with more buildings and integrating the second campus a way of managing the scene would be beneficial. Depending on the performance, splitting the scene once with one scene being the Singleton Park campus and a second scene being the new Bay View campus. This would be be the simplest to implement. If the performance drops then there may be a need to split one campus into separate scenes and have seamless transition between the various scenes.

5.3 Final thoughts

Digitising the campus has been an exciting project with many challenges and learning various skills.

The research of many different people has explained the challenge of taking a structure and recreating it inside a computer. From the research the most interesting technique of doing this was the paper from Microsoft research of using a Kinect and doing the digitisation process in real time.

Although this project has had a few setbacks, for example trying to get Structure from motion software working, overcoming them and seeing a photograph of a building in 2D turned into a 3D building that can be manipulated inside a computer has been exciting to see.

The application has had many revisions during its development. Firstly the application was limited and would only render one building at a time. The second version slowly built up the scene that a user could walk around. The third version added in the navigation, which went from a straight-line motion through the buildings to using path finding algorithm A* to plot a route around buildings. In JMonkeyEngine I found the problems created of having large complex meshes on performance. I applied the knowledge learnt from computer graphics lectures, that a model can have a simple mesh but detailed texture and will look the same as a model with a complex mesh but be much faster to render.

By using the Agile development model to develop application, has been successful to prioritise and delivered on all the aims of the project. The aims of the project were:

To generate 3D buildings from 2D photos of the buildings, this was successfully accomplished by using a combination of VSFM to generate the point cloud and MeshLab to make the 3D mesh.

Display Models to the user. Using a Graphics program enables the resulting model to be viewed by the user. Using the modelling software Blender and Unity3D I was able to digitise the campus and used Unity3D to make an application to view the campus.

The project will also have navigation so that the user does not aimlessly walk around the campus but is directed to their selected location. Using a navigation mesh and path finding over the mesh a simple navigation system was successfully built up.

With the campus changing so frequently and better models being produced, a way to keep the map updated without the user having to check is required. Hosting the application on a server and using a simple tool on the device (PC, tablet, mobile telephone) to check the server for the version code and, if required download the newer version.

Create mobile apps for the user so that the program is available on Smartphones or tablets so that users can look at the project on the go. Using Unity3D multiple build platforms the same version of the program can be compiled for Mac, Windows and Android platforms.

The future of the project is to increase the campus size by adding more buildings. Additionally if there was more time use a slightly different approach to capture the buildings, it would be possible to achieve a higher quality representation.. Also add a more intuitive user interface (user interface that the user is already familiar to). In this case a user interface similar to Google Maps would mean that the user would not have to learn another new interface.

Chapter 6

Demo Videos

Demo	Description	Link
Mesh Demo	Showing the final result of the mesh	http://vimeo.com/84209398
Program V1	First Demo of the program, loads one model at a time	http://sugaming.co.uk/uni/ScreenFlow-all.mp4
Program V2	Partial scene of the campus you can walk	http://vimeo.com/87960753
Navigation	Demo Application showing navigation working around a simple object	http://sugaming.co.uk/uni/path-finding.mp4
Program V3	Full application of navigation, user interface working	http://vimeo.com/90014847
Program V4	Full Application on the Unity Engine	http://vimeo.com/90378847
Program Android	Full Application working on Android	http://vimeo.com/90378679

List of Figures

Appendix

I like to thank the following people:

- Dr Gary Tam for helpful instructions on how to improve the project
- Changchang Wu and his structure from motion software
- jMonkeyengine Forums for helpful support for small problems encountered while writing the java version of the project
- Martin Glaude (quill18) For some useful Unity Tutorials for how to get started with unity and building the User interface.

Bibliography

[1] Ucas lifestyle survey summary, April 2014. URL http://www.ucasmedia.com/sites/default/files/Lifestyle-Survey-2013-Summary.pdf.

[2] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[3] M. Koide, S Kato. 3-d human navigation system with consideration of neighboring space information. *Systems, Man and Cybernetics*, 2006.

[4] Murat Arikan, Michael Schwärzler, Simon Flöry, Michael Wimmer, and Stefan Maierhofer. O-snap: Optimization-based snapping for modeling architecture. *ACM Transactions on Graphics*, 32:6:1–6:15, January 2013. URL http://www.cg.tuwien.ac.at/research/publications/2013/arikan-2013-osn/.

[5] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. Processing and interactive editing of huge point clouds from 3d scanners. *Computers and Graphics*, 32(2):204 – 220, 2008. ISSN 0097-8493. doi: http://dx.doi.org/10.1016/j.cag.2008.01.010. URL http://www.sciencedirect.com/science/article/pii/S0097849308000253.

[6] Changil Kim, Henning Zimmer, Yael Pritch, Alexander Sorkine-Hornung, and Markus Gross. Scene reconstruction from high spatio-angular resolution light fields. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 32(4):73:1–73:12, 2013.

[7] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4):113:1–113:16, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461940. URL http://doi.acm.org/10.1145/2461912.2461940.

[8] S. D. Goodwin, S. Menon, and R. G. Price. Pathfinding in open terrain.

[9] Ruth Kusterer. *jMonkeyEngine 3.0 beginner's guide*. Packt Pub, Birmingham, U.K, 2013. ISBN 9781849516464.

[10] Flickr press release. URL http://web-archive-net.com/page/521114/2012-10-25/http://pressroom.yahoo.net/pr/ycorp/423991.aspx.

[11] Amazon compute cluster pricing. URL https://aws.amazon.com/ec2/pricing/.

[12] Autodesk 123d - free 3d modeling software, 3d models, diy projects, personal fabrication tools.

[13] Bundler - structure from motion (sfm). URL https://www.cs.cornell.edu/~snavely/bundler/.

[14] xplan. URL http://www.xplanapp.com/english/index.php.

[15] Peter Krogh. Backup overview. URL http://www.dpbestflow.org/backup/backup-overview.

[16] Changchang Wu. Visualsfm : A visual structure from motion system. URL http://ccwu.me/vsfm/.

[17] Yasutaka Furukawa. Clustering views for multi-view stereo. URL http://grail.cs.washington.edu/software/cmvs/.

[18] Wireframing tool for professionals - wireframesketcher. URL http://wireframesketcher.com/.

[19] Robert Fletcher. Digitisng the campus feddback. URL https://docs.google.com/forms/d/1xJwPbiMiBCoKsdYHXgHQ7RYG69r1Xtvim4NOFkurUwk/viewanalytics.