TRINITY COLLEGE

*TCD Virtual Tour for Android*

Brid Ailish Carey  
B.A.I. Engineering  
Final Year Project  April 2012  
Supervisor: Professor Carol O'Sullivan

School of Computer Science and Statistics  
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

______________________________________________        ________________________
Name            Date
Acknowledgements

I would like to thank Professor Carol O'Sullivan for agreeing to supervise this project and for all her help and guidance throughout.

For his constant help and support, without one complaint, I would like to give a very special thanks to Paul Shirley. He has truly been an inspiration throughout this project and over the last four years.

I would also like to thank my family and friends, especially Kristina Kenneally and William Hughes, for putting up with me and listening to me complain…I promise to stop now.

I would like to thank my Eibhlin Carey for being the self-proclaimed “idiot” tester for this application, without her the application would be far less user friendly.

And finally I would like to thank Ronan Murtagh for his constant supply of tea and banter over the past few months, I would have been lost without it!
## Contents

1. **INTRODUCTION** ............................................................................................................... 1  
   i. PROJECT AIM ........................................................................................................ 1  
   ii. READER’S GUIDE ................................................................................................. 1  

2. **BACKGROUND** ........................................................................................................... 3  
   i. MOTIVATIONS ......................................................................................................... 3  
   ii. RESEARCH ............................................................................................................ 4  
   iii. SOFTWARE ........................................................................................................... 6  
   iv. PROGRAMMING LANGUAGES ............................................................................. 7  

3. **DESIGN GOALS** ......................................................................................................... 8  
   i. VISUAL GOALS ....................................................................................................... 8  
   ii. INTERACTIVE GOALS .......................................................................................... 8  

4. **SCENE SETUP – ANDROID BASICS** ......................................................................... 10  
   i. SURFACE ............................................................................................................... 10  
   ii. RENDERER ............................................................................................................ 10  
   iii. LIGHTING ........................................................................................................... 10  
   iv. TEST – CREATING THE CUBE ........................................................................ 11  
   v. LAYOUT ................................................................................................................ 12  
   vi. OVERLAY ............................................................................................................. 12  

5. **MODEL LOADER** ....................................................................................................... 13  
   i. MIN3D .................................................................................................................... 13  

6. **CAMERA INTERACTION** .......................................................................................... 15  
   i. CAMERA FUNCTION .............................................................................................. 15  
   ii. DIRECTIONAL BUTTONS ..................................................................................... 15  
   iii. JOYSTICK ............................................................................................................ 16  
   iv. NAVIGATING THE SCENE ................................................................................... 17  

7. **THE MODELS** .......................................................................................................... 18  
   i. BUILDING THE MODELS ....................................................................................... 18  
   ii. TEXTURING THE MODELS ................................................................................... 19
8. INTERACTIVE MODELS ................................................................. 23
   i. WHAT WAS TOUCHED ............................................................ 23
   ii. INFORMATION DISPLAY ...................................................... 26
9. FINISHING TOUCHES ............................................................... 27
   i. ADDING THE SKY ................................................................. 27
   ii. CAMERA ANIMATION ......................................................... 30
   iii. WELCOME SCREEN ............................................................ 31
10. CONCLUSION ............................................................................. 33
    i. REVIEW ............................................................................ 33
    ii. EVALUATION .................................................................... 33
    iii. FUTURE WORK .................................................................. 34
11. BIBLIOGRAPHY ....................................................................... 36
12. APPENDIX 1 – ROTATING CUBE PROGRAM ............................... 38
13. APPENDIX 2 – CAMERA FUNCTION ........................................ 43
    i. PART A – DIRECTIONAL BUTTONS CODE .......................... 43
    ii. PART B – JOYSTICK CODE ................................................ 44
14. APPENDIX 3 – SCREEN TO WORLD COORDINATES ............... 45
Introduction

Project Aim

The aim of this project is to combine the Virtual Dublin project, more specifically the Trinity College Metropolis, with the Android platform to create a highly accessible, user-friendly, interactive virtual tour.

Reader’s Guide

- **Chapter 1** provides the reader with an introduction to the project and the report, giving a general overview of the aims of the project and how the report is structured.
- **Chapter 2** discusses the background of the project, including motivations behind the project and the technology and programming languages necessary to complete the project. It also gives an insight into the research carried out before beginning the project.
- **Chapter 3** discusses the goals set for the project.
- **Chapter 4** describes how the scene was created and set up for Android.
- **Chapter 5** reviews the complications around and solutions to the issue of loading a 3D model into our Android scene.
- **Chapter 6** examines the camera movement, with its main focus being user control of the camera.
- **Chapter 7** explains the creation of the models used to represent the buildings in the scene.
• **Chapter 8** discusses the research behind and attempts made to create the interactive models.

• **Chapter 9** explores the finishing touches added to the scene to create a more complete application.

• **Chapter 10** draws conclusions about the entire project and illustrates where the project will go in the future and how it can be built upon.
Background

Motivations

Due to the dramatic increase in the number of smartphone and tablet users in recent years there is an increasing demand for easily accessible, highly portable, user-friendly media.

Virtual tours are one area of this media in which there has been a huge adaptation for use on smart devices. The definition of a virtual tour is

_A panoramic view or video simulation of an existing place that can be viewed online, often for travel or vacation-related research. A virtual tour is usually a collection of panoramic images that are played in sequence to view like a moving video with added sound and text effects, or a virtual tour may use models in place of real-life video and images. A virtual tour is designed to give the viewer a more life-like 3D view of the location being presented in the tour._

The majority of virtual tours available use panoramic images and videos of the locations which provide the user with a comprehensive view of the area but do not allow for navigation through the scene in the same way that a tour using models does. Although these tours provide the user with a more life-like view of the location it does not give rise to a feeling of movement through the location.

---

Virtual Dublin is an urban simulation of Dublin city centre created by the Image Synthesis group in Trinity College Dublin. The simulation uses 3D models of the city to create a virtual metropolis through which the user can freely navigate.

At present the Virtual Dublin project lacks the aspect of portability which is in such high demand in recent years. This is the main motivation behind the project. The project aims to take the idea of Virtual Dublin and create a version compatible with smartphones and tablets.

Due to the size of the Virtual Dublin project the focus of this project became Trinity College Dublin. This seemed like a good place to focus project as it is a tourist attraction for which a virtual tour would be both extremely popular and highly useful.

Research

As with beginning any project, research needed to be done to see what is currently available with regard to virtual tours, and more specifically college virtual tours. It became clear very quickly that virtual tours of colleges have become extremely popular in recent years,
especially in America, allowing students and their families to see what a college has to offer from the comfort of their own home.

**Still Image Tours**

Many college websites provide visual tours which are very basic and consist of a set of still images taken from various locations around the campus saved to a picture gallery. These types of tours are very static and do not provide a user with a feeling of a realistic tour. One such example of this can be seen on the NUI Maynooth website.³

**Panoramic Virtual Tours**

As mentioned previously the majority of virtual tours available use panoramic images or videos to create a realistic walk through of an area. Research showed that most of the college tours currently available used such methods. The most popular of provider of such tours is YOUVISIT⁴ (previously YourCampus360). YOUVISIT provides campus tours for many of the major Universities in America. They provide virtual tours via their website, Facebook applications, and mobile device applications.

⁴ Tours available at YOUVISIT, [http://www.youvisit.com/](http://www.youvisit.com/)
⁵ Screenshot from the virtual tour of Yale University, [http://admissions.yale.edu/virtual-tour](http://admissions.yale.edu/virtual-tour)
Software

Platform

Choosing the platform to with which to develop this project was not a difficult task. There are many operating systems to choose from with regard to development, with the two main platforms being Google’s Android and Apple’s iOS. Since Android is a more commonly used, developer friendly platform it made sense to use it for this project.

Development Environment

Most Android applications are developed using the Java programming language along with the Android Software Development Kit (SDK). There are a variety of Java integrated development environments (IDE) available. The IDE chosen for this project was Eclipse. Eclipse is an open source Java development environment supported by Google developers. Eclipse is the officially supported IDE.

Software Development Kit

The Android SDK contains an exhaustive set of development tools necessary for programming in Android. Included in this set are libraries, a debugger, an Android emulator along with sample code and tutorials for learning to use Android.

Autodesk 3ds Max

Autodesk 3D Studio Max is just one of the many software options available for creating, animating and rendering 3D models. 3D Studio Max is the program used to create the models for the Virtual Dublin project so it seemed appropriate to continue using this software for this project.
Programming Languages

Java

As mentioned previously Java is the most commonly used programming language for developing Android applications. Java is a class based object-oriented programming language that was designed to have as few implementation dependencies as possible to allow for use across various platforms.

OpenGL ES

OpenGL is the language used to write applications using 2D and 3D computer graphics. OpenGL itself is too large so Android must implement a subset of OpenGL known as OpenGL ES. OpenGL ES is a version of OpenGL designed for use on embedded systems such as mobile phones.
**Design Goals**

Before beginning the project the goals needed to be clearly outlined. The goals for this project can be split into two categories, visual goals and interactivity goals.

**Visual Goals**

As this is a graphics project visuals play a major role. There are two main aspects to making this application visually pleasing, the 3D design of the models and the design of the interface.

The design of the models was not overly difficult as they are models of real buildings and so have to be designed as the actual buildings stand. The layout of the buildings must match how the campus itself is laid out. The design of how the user sees the created world is the important part of the 3D design. The campus feels like an enclosed space with most of the buildings facing inwards, making it a good option for a virtual tour.

Designing the UI is an interesting part of the application. The aim of the UI is to allow full functionality while still being aesthetically pleasing, a careful balance is needed. It is important that the screen does not become overly cluttered with buttons and instructions which can be complicated and distracting. However it is also important that the UI is not so simplistic that control of the application becomes compromised.

**Interactive Goals**

Interactivity is a very important aspect of a virtual tour. Without it the tour can become very mundane and boring and can cause users to lose interest extremely quickly. There are two main aspects of interactivity for this application, camera control and interactive models.
The design behind camera control is simple. The aim is that the user has full control of what they see and where they go in the scene. This control is linked with the UI with regard to how the camera will be controlled.

The majority of the models created should be interactive. The goal is to allow the models tell their own story by creating a function to display important information about a model when it is touched, e.g. the name of the building, when it was built etc.
Scene Setup – Android Basics

Surface

As a starting point the project needed a surface on which to draw. For this project a GLSurfaceView is used. GLSurfaceView is a type of Android view which is specifically used for OpenGL rendering. The surface size and format can be controlled for specific applications and GLSurfaceView ensures that the surface is at the correct location of the screen.

Renderer

The renderer used for this project implements GLSurfaceView.Renderer, meaning it contains three public methods, onSurfaceCreated, onSurfaceChanged, and onDrawFrame.

The onSurfaceCreated method is called when the surface being rendered is created or re-created. This method contains any code which needs to be implemented once the rendering begins, e.g. setting up lighting, or enabling textures.

The onSurfaceChanged method is called after a surface has been created or whenever the surface changes size. The viewport is set in this method.

The onDrawFrame method is continuously during the rendering process. This method is responsible for drawing the current frame. All other calls necessary for rendering the scene, e.g. calls to create models etc. are called in this method.

Lighting

Lighting is a very important part of any OpenGL scene. It is important not only to allow visibility of the contents of the scene but it can also make objects appear more realistic.
OpenGL allows up to 8 light sources to be created in a scene and there are two parts to these light sources, the light itself and also what the light is shining on.

The position of a light source determines what the light is shining on. The more complicated part of lighting is creating the light itself. There are three types of lighting, ambient, diffuse and specular. A single light can contribute to all three types of light so it is important to know the difference between them.

Ambient light contributes to the entire scene because it is a directionless light. Diffuse light is the main contributor of light to the scene. Diffuse lights are soft directional lights. Specular lights are shiny lights usually coming from bright point sources. Specular lights are those which can create a more realistic affect, e.g. if a specular light is combined with a shiny material.

**Test – Creating the Cube**

To test that the setup had been carried out correctly and to become familiar with programming for Android it seemed intuitive to create a simple program which would utilise all of the factors described above. The test program simply created a rotating textured cube\(^6\).

---

\(^6\) Code for this can be seen in Appendix 1.
Layout

When creating an Android application there are two ways to create a user interface (UI), either programmatically, i.e. building the UI using code, or else using XML based layouts. Creating a UI was unnecessary for the test program but is needed for the project. For this project using XML based layouts are sufficient for the UI. Each UI of an application is displayed through an Activity; therefore each Activity needs its own XML layout file.

The layout file refers to the container for all of the child objects and the position of the objects on the screen, e.g. the GLSurfaceView is one of the child objects of this project. There are a variety of different layouts available for Android depending on how the screen is to be laid out. The main layouts available for Android are:

- FrameLayout – this layout arranges the child objects so that they all begin on the top left corner of the screen.
- RelativeLayout – the objects are laid out in relation to each other or a parent.
- LinearLayout – all children are laid out in a single row or column.
- TableLayout – objects are arranged into rows and columns, like a table.

For this project the main layouts being used are FrameLayout and LinearLayout.

Overlay

Using an overlay allows UI widget, e.g. buttons, to be superimposed onto the GLSurfaceView. The widgets need to be created in the XML layout file and are then given functionality in the main activity’s onCreate function.

---

7 Overlay example can be found at http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/SurfaceViewOverlay.html
**Model Loader**

One of the most important aspects of the project was finding a way to load 3D models into an Android scene, which is not always an easy task. It became apparent quite quickly that models saved as OBJ files were the easiest to load into the scene. The OBJ file format stores the geometry of a 3D model. The materials associated with the model are stored in a separate external .mtl file.

Investigation showed that there are Android packages available for loading OBJ files to a scene, some better than others. In the interest of efficiency it seemed appropriate to make use of a package that was freely available as opposed to wasting time recreating the same thing. The process of choosing which package to use was very much an exercise of trial and error. After trying many different packages it became clear that the best package available for the purpose intended was min3d.\(^8\)

**min3D**

min3D is an open source Android framework for 3D models. Since min3d uses Java and OpenGL ES it was very compatible for use with this project.

The min3d framework comes with a variety of sample projects and helpful tutorials showing how to implement the framework as part of a project.

To test the functionality of min3d an arbitrary 3D model was found online\(^9\) since the models for the tour had not yet been created. Some of the main problems found when loading the model were missing textures and scaling issues.

---

\(^8\) Min3d package, [http://code.google.com/p/min3d/](http://code.google.com/p/min3d/)

\(^9\) Model downloaded from 3DVIA, [http://www.3dvia.com/models/66964B5C6E405264/face](http://www.3dvia.com/models/66964B5C6E405264/face)
The missing textures problem was mainly due to the textures being referenced badly, i.e. changing the file name or location without updating the .mtl file. Another problem with textures arose with any files beginning with capital letters. The Android platform doesn’t accept files with capital letters so it was important rename all files in lowercase.

Incorrect scaling caused models to appear invisible. Altering the scale solved this problem which is a simple procedure when using min3d.

After successfully loading the face model using min3d it was time to load one of the building models. As a test the Virtual Dublin model of the museum building was used.  

When this model was correctly loaded simple extra features were added to test the android functionality, such as making the object rotate when the screen was touched.

Fig 5 & 6: Screenshots of models loaded using min3d

---

Camera Interaction

Another important part of the project was to allow the user to be able to control the movements of the camera to navigate through the scene. This part of the project was done in two stages, the first was create a function allowing the user to navigate around the original scene created containing just the cubes, the second and more difficult stage was to combine the camera movement with the model loader to allow for navigation around the models which was essential to the project.

Camera Function

Rather than using the function available in OpenGL known as gluLookAt, creating a new, simpler camera function seemed appropriate.

The camera function contains 6 variables, an x, y, and z position and rotation value. The function then uses the OpenGL rotate and translate functions to move the camera depending on the value of the previously mentioned variables.

The values of these variables are altered using trigonometric functions. A different set of functions is used depending on the direction in which the camera is to be moved. 11

Directional Buttons

The original implementation of camera movement was created using 6 directional buttons, forward, back, left, right, up and down. Each button was linked to an onClickListener which performed one of the functions found in Appendix 2. Although this method worked correctly having six buttons on the screen was not appealing to the eye so other options were explored.

11 The code for the camera movement can be found in Appendix 2 Part A
Joystick

Since the directional buttons lacked visual appeal and made the screen look overly full it was decided that another approach would be needed for camera movement. A suggestion was made from one of the members of the GV2 research group that a joystick would look better for this application.

A short internet search showed that a joystick widget was available for Android from mobile-anarchy-widgets.¹² Like with the min3d framework, learning to implement the joystick widget took some time but it improved the look of the UI remarkably.

Implementing the joystick meant introducing slight variations in the camera movement code, more specifically the alteration of the variable values. There was no longer a need for the up, down, left or right buttons so these were removed and the code was altered to suit. Since the joystick would simply pan around the scene there was still a need for two buttons

to allow for forward and back movements, so the code for these buttons remained unchanged. To make the forward and back buttons look more appealing these were changed from Buttons to ImageButtons. ImageButtons allow a picture to appear on the button instead of text.

![Fig 10: Joystick and ImageButtons](image)

**Navigating the Scene**

With the camera movement working with the original scene it was important to get both the model loader and the camera working in unison. This involved altering the camera class and the renderer provided by min3d to include the function created for camera operation. Once this was complete the project contained a scene through which the user could manoeuvre easily, although at this point it did not contain very much!

![Fig 11: Linking camera movement with model loader](image)

---

13 The altered code can be seen in Appendix 2 Part B
**The Models**

As mentioned previously the majority of the models created for this project were based on the Trinity College Models created by the Virtual Dublin team. The original goal was to create a new set of models for the project however due to time constraints some of the original Virtual Dublin models were also used\(^\text{14}\).

As stated previously the models needed to be saved as OBJ files to be compatible with the min3d framework.

**Building the Models**

After examining the models from the Virtual Dublin project, it was time to get started on creating the new models. Having never previously used 3DS Max there was quite a significant amount of trial and error when creating the first model, the chapel. There were many attempts made to recreate the chapel. One effort saw the chapel being created in complete 3D as can be seen in the figure below. All aspects of the building were created individually.

![First attempt at building the chapel](image)

**Fig 12: First attempt at building the chapel**

\(^{14}\) The Postgraduate library and Campanile Virtual Dublin models were used.
After rethinking some ideas it seemed much better to create a far simpler object and add detail to it using textures.

![Fig 13: Simpler chapel model without texture](image)

The rest of the models created were then developed using this method of detailing with textures.

**Texturing the Models**

After creating the simple models it was then time to make them look more realistic by adding more detail. This was done by using the images from the Virtual Dublin models as textures for the new models. This was where the process started to get more difficult.

The first obstacle which arose was trying to apply different textures to different faces of the model. In 3DS Max only one material can be applied to an object at once. To apply multiple images to a model the material had to be created as a multi-sub object in 3DS Max. This allowed each face of the object to be textured with a different image rather than the same image being applied to all sides of the object.
Since the majority of the models were unusual shapes to allow for the building’s apex, texturing the buildings became slightly complicated. The main problem was that the textures did not sit on the models correctly. For example, on the first attempt at texturing the chapel model the texture added to the front face was distorted.

![Chapel model with distorted texture](image)

**Fig 14: Chapel model with distorted texture**

To overcome this problem a modifier called a UVW map was added to the model. UVW mapping is a method for applying textures to models in 3DS Max. This method describes the relationship between the areas of the texture and the geometry of the surface to which it is applied. The UVW map adds another modifier called a gizmo. This can be set to planar, spherical, cylindrical, shrink-wrap, box or face depending on the face selected for texturing. The setting of the gizmo gives a basic mapping to the selected object. The UVW Unwrap map modifier can provide a more accurate mapping, allowing manual modification of texture vertices and faces, but in the case of the models for this project the UVW map was sufficient.

Since the models created for this project were very simple the box setting was used for the gizmo as this most accurately described the face being textured. After some playing around with the UVW mapping the distorted texture problem was corrected.
When creating some of the more complicated models, e.g. the Graduate Memorial Building (GMB), applying the UVW map was not sufficient to solve the texturing issues. For some of the models the texture images had to be edited before they were used as a material. For example the column section of the GMB needed an image for the column face but also the column apex. Rather than adding these as separate textures they were combined to one image using Adobe Photoshop.

Combining the images in this manner meant that applying the UVW map to the model to fix the distortion was a far simpler process than if the two images were added separately.
On completion of the models the files were exported as OBJ files to be loaded to the scene.

**Adding Models to the Scene**

Having completed building and detailing the models of the buildings it was time to load them to the correct positions in the scene. Having gotten min3d working with the program previously this was a relatively simple, yet tedious task.

There were some minor scaling issues when combining the new models and the Virtual Dublin models in the same scene but with a few tweaks these problems were easily overcome.

The only other matter that arose when loading in the models was the lighting in the scene. When the models were initially added the scene looked quite dreary and dark. However adding another light to scene brightened it and removed the dull look.

![Fig 17, 18, 19, & 20: Screenshots of the scene with the finished models](image-url)
Interactive Models

The next aspect of the project to consider was making the models interactive. The main idea here was that the application could tell when a model has been touched, be able to tell which model has been touched and display some useful information about that model. The two main considerations here were how to determine exactly what had been touched and then deciding how to display the information about the model.

What Was Touched

Screen Coordinates to World Coordinates

To begin deciphering exactly what had been touched there needed to be a method implemented which would convert the 2D coordinates touched on the screen to 3D world coordinates. Research showed that there are two main ways to do this, using the OpenGL library’s gluUnProject method, or implementing a method of 3D picking.

To use gluUnProject a number of variables are needed to be input to the function. The function firstly needs the x, y, and z screen coordinates. There is a function available to get the x and y positions of a motion event, e.g. a touch, in Android. However the screen coordinates begin with the point (0, 0) in the top left whereas OpenGL coordinates start at the lower left so the screen coordinates have to be converted to OpenGL coordinates. This conversion is carried out by letting the screen y coordinate equal to the screen height minus itself. The screen z coordinate is still missing. This can be found by using the OpenGL function glReadPixels.

gluUnProject also takes in three variables in which to store the x, y, and z OpenGL coordinates. These are just declared as GLDouble values.
The next variable needed is the viewport origin and extent. An array containing the starting x and y position of the viewport along with the viewport width and height is needed to be input into the function. The contents of this array can be found using the glGetIntegerv function.

The final two variables needed are the modelview and projection matrices. The modelview matrix is responsible for determining how the vertices of OpenGL objects are transformed into eye coordinates. The projection matrix is responsible for transforming vertices in eye coordinates to clip coordinates. Both of these variables are found using the same method, using the glGetDoublev function to fill their respective arrays. This is where the gluUnProject method runs into a problem when being used for Android as the OpenGL ES library does not support a glGetDoublev function so these variables cannot be found too easily.

The gluUnProject function for the Android platform contains many errors so it cannot be used for this project.

The other option to convert screen coordinates to world coordinates is to use a technique called picking. When a point on the screen is touched picking simply takes that point and applies the inverse transforms applied by OpenGL in order to get the point in the world which the user wanted to choose.

A function was created to try to implement picking in the project. This function takes in two objects, a 2D vector which is the screen touch point and a camera object which contains the camera position along with the screen width and height.

---

15 This function can be found in Appendix 3. It is an adaptation of the code found at http://magicscrollsofcode.blogspot.com/2010/10/3d-picking-in-android.html
As with gluUnProject it is important to invert the Y touch coordinate to convert to the bottom left coordinate system used by OpenGL. After this is done the following steps of the picking algorithm are carried out:

- The screen coordinates are converted to clip coordinates.
- The transform matrix is obtained and inverted.
- The clip space coordinates are multiplied by the inverse transformation.
- The x, y, and z points are divided by w to obtain the world coordinates.

To obtain the projection and modelview matrices two further Android classes must be included in the project, MatrixTrackingGL and MatrixStack. The final addition is to add the GLWrapper call to the GLSurfaceView being used as the MatrixTrackingGL acts as a wrapper for the gl context.

Unfortunately this function did not work correctly with this particular project for reasons which have yet to be discovered. The function did output world coordinates when the screen was touched however these coordinates seemed not to make sense.

Although neither method worked further investigation was carried out with regard to creating the interactive models.

Model Intersection

Another aspect of discovering what was touched in the scene was to see if the world coordinates calculated actually intersect a model. The easiest solution to this seemed to be to create bounding boxes around each of the models which are to be interactive and check

---

16 2D, 3D and camera classes are also added to the project, the code for these are also contained in Appendix 3.
17 These classes are included in the Android samples.
where the coordinates lie in relation to these boxes. If the point intersected or lay inside one of the bounding boxes then it could be assumed that the model inside the box was touched.

This aspect of the project was merely an idea. It did not get to be tested since the previous step was inaccurate.

**Information Display**

There are many different ways of displaying information about the models. The two main ideas were to overlay pop-up information windows like those used in Google maps or else to create an activity for each model which would contain the information.

Assuming that the application can tell which model has been clicked the easiest way of displaying information about the models would be to create an activity for each model which is launched when the respective model is touched. These activities would use a combination of ScrollView and TextView to create the effect of text inside a large container. The text would be specified inside the strings.xml file, which is created when an android project is created in eclipse. A dialog theme would then be added to allow for the scene to still be seen behind the text.

The second option requires some research and adaptation however it would probably look much better for this application. InfoWindows are a type of overlay used with Android’s MapView. InfoWindows display information in a window floating above the map. This type of overlay would be perfect for presenting the information about the models in this application. At present this method is only used with MapView but with further research an adaptation could be created to work with the GLSurfaceView used for this project.
Finishing Touches

With the models loaded into the scene and the ground work complete it was time to add finishing touches to the project to make it complete.

Adding the Sky

Adding a sky element to the project was one of the most important finishing touches. Without the sky the scene did not look perfect.

There were many approaches taken to insert a sky to the scene. The first attempt was a very simple and basic idea, to set the colour of the GLSurfaceView to a shade of blue. This however was not the correct approach. Since the models are loaded onto the GLSurfaceView altering its colour meant also altering the colour of the models.

Fig 21 & 22: GLSurfaceView background colour set to blue

With an alpha value of 255 the models could not be seen at all, lowering the value to 100 just caused the entire scene to be tinted blue as can be seen from the above figures on the left and right respectively.

The next method was to add a skybox to the scene. A skybox is a technique used in computer graphics to create a background for an area. When a skybox is used the area is enclosed inside a cuboid and the sky, or other background e.g. distant mountains, is
projected onto the faces of the enclosure. This creates a look of being far in the distance, making the area appear bigger than it is.

The min3d framework used for loading in the models contains a skybox class so creating a skybox for the scene was a relatively simple task.

![Skybox attempt](image)

**Fig 23: Skybox attempt**

Although skyboxes are widely used it was decided that this technique was not the best way to create the sky for the scene and other procedures were to be explored. The skybox however remained an option on which to fall back if all else failed.

The next effort to add a sky to the scene was a relatively simple process. The idea was to use an ImageView. An ImageView is a very simple concept. It is a type of Android view which displays an arbitrary image. In order to use this method the main view, the GLSurfaceView, had to be set to have a transparent background so that the ImageView was visible behind it.

This method seemed to be the correct option for the project. The only downside of using this method is that when the project is initially loading, i.e. before the models appear, the ImageView is visible.
Fig 24: Project loading, ImageView visible

This however is a relatively minor price to pay for the finish it brings to scene.

Fig 25, 26, 27, & 28: Screenshots of scene with finished sky
Camera Animation

Although the scene was now finished there was something incomplete about the project.

Even though camera could be moved by the user the scene still felt very static. To overcome this problem it was decided to introduce an animated camera option which would navigate around the scene without the user having to control the camera, but still giving the user the option to do so.

To do this firstly an OnTouchListener was added to the GLSurfaceView. An OnTouchListener defines what happens when the view to which it is attached is touched. This listener then updates a Boolean value which tells the rest of the program whether the camera is being manually controlled or automatically controlled.

When the Boolean value is set to be true then the camera becomes animated. To animate the camera a number of key points were created around the scene, a point looking onto each model, and then spline interpolation is used to manoeuvre the camera between these points. Interpolation is used to approximate the values between the key points given.

Each of the key points was given an x, y, and z value but also a y-rotation value so that the camera would not only move through the scene but would also rotate to face the models correctly.

After testing this camera animation it was found that using the touch function was not the best idea as it was very sensitive and could easily be triggered by accident. The solution to this problem was to attach an OnLongClickListener to the GLSurfaceView instead of an OnTouchListener. An OnLongClickListener works similarly to an OnTouchListener but waits to ensure the touch was long enough to be intentional.
Welcome Screen

To further complete the project and make it look like a more professional application a welcome screen was added. This avoids launching the scene directly without giving the user any inclination about what it does, how it works and how to use it.

Creating a welcome screen means that rather than having a single activity the project now has a four, the “welcome” activity, the “controls” activity, the “about” activity and the main “tour” activity. These extra activities are created like the main activity but have far less functionality. Each activity must have its own XML layout file similar to that created for the main activity.

The first activity to create is the welcome activity. This is the activity which runs when the application is first opened. The layout for this is described as above with three buttons linking to the other activities.

![TCD Tour](image)

**Fig 29: Screenshot of welcome screen**

An OnClickListener has been added to each of the buttons so depending on which button is clicked a different activity is launched.

Clicking the enter button simply begins the main activity. When either the about or controls buttons are pressed a pop-up window is shown containing information about the project
and how to manoeuvre the scene respectively. These activities are created in the same way described earlier for displaying information about the models.

**Fig 30 & 31: Screenshot of control instructions and project information**
Conclusion

Review
The aim of the project was to create an interactive 3D virtual tour of the Trinity college campus for Android. After creating the necessary models and finding a way to load the models into an Android scene, Front Square was recreated so that some interactive functionality could be added.

The resulting application shows the potential for applications of this type, user controlled graphical tours, to be created for Android.

Evaluation
Evaluating the project can be done from two perspectives, that of the developer and that of the user. As the developer it makes sense to evaluate the project in terms of progress made versus progress that could have been made. Evaluation from the perspective of the user would be much different, more in terms of how useful the application actually is.

Evaluating the project as its developer can be a difficult task as much of the focus is put on what the application can become as opposed to what it has not achieved at this point. With regard to the aims set out at the beginning of the project it is clear at this stage that they were overly ambitious, some due to time constraints, others due to lack of knowledge. For example it became clear during the project that creating the entire Trinity campus was not feasible considering the time left so the project was focused to creating just Front Square. Some other goals also had to be put on the backburner, e.g. the interactive models. However the project as a whole does contain the basics needed for an interactive 3D virtual tour and provides a solid foundation for future work.
In hindsight it is easy to see how things could have been done differently. As a whole the approach taken to carry out the project was logical however poor time management was a serious problem. With regard to the models themselves, it is obvious that further practice using 3DS Max is needed but the models are acceptable for this stage of the project. In terms of the project code although it is working correctly there is room for improvement in terms of efficiency.

Since the application is not yet 100% ready for users, comprehensive testing was not possible. However slight changes were made based on the reviews of friends and family. With a few improvements made to the application it would be helpful to receive feedback from a group of users likely to use the application if it was ever to be made available, e.g. tourists or prospective students. The application could be released as a beta version on the condition that users complete a questionnaire after using the application specifying likes or dislikes and suggesting any further improvements which could be made to the application.

**Future Work**

This section of the report has the potential to be very long due to the large number of directions in which this application can be taken and also the number of additions which can be made to an application of this type.

In the short term the main focus for future work would be to complete all of the goals set at the beginning of the project before deciding to add further improvements.

- **Completing the campus** - The predominant goal would be to create the rest of the campus so that a complete virtual tour is possible, i.e. creating models for the rest of the buildings on campus and ensuring the campus layout is correct.
• **Interactive Models** - Improving the interactivity of the application is another important aspect which can be looked at in the near future. Finding a way to make the models interactive would drastically improve the user experience when using the application. The main to be dealt with is trying to determine exactly what has been touched, i.e. converting the touch coordinates to world coordinates.

Additional adjustments that could be made include:

• **Collision detection** – the application would be able to detect when a user’s view has collided with a building or other object in the scene. This would also be useful for ensuring that the user cannot go outside the bounds of the campus.

• **Interactive Tour Guide** – introducing an interactive tour guide to the application would be an excellent addition which would improve the virtual tour experience. The tour guide would be responsible for giving information about the current area of the campus which the user is in and also portraying information about the models when they are clicked, similar to the tour guides in the YOUVISIT tours mentioned earlier.

• **Sound Effects** – introducing sound to the application would make the tour more realistic.

• **Campus Map** – a campus map could be added to show the user where they are in relation to other areas of the campus.

• **Connectivity** – the application could contain links to the Trinity website where users could find additional information about the college. There could also be connectivity with useful services which may also be needed by users, e.g. travel information.
Bibliography


5 January 2012].


Appendix 1 – Rotating Cube Program

```java
package fyp.FYP;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.app.Activity;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;
import android.os.Bundle;
import android.util.AttributeSet;
import android.view.View;
import android.widget.Button;

public class FypActivity extends Activity {
    /** Called when the activity is first created. */
    GLSurfaceView view;
    View container;
    public static int a = 3;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = new GLSurfaceView(this);
        setContentView(R.layout.main);
        view = (GLSurfaceView) findViewById(R.id.view);
        view.setRenderer(new GLRenderer(this));
    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    //Rendering the scene
    class GLRenderer implements GLSurfaceView.Renderer {
        private static final String TAG = "GLRenderer";
        private final Context context;

        private final GLCube cube = new GLCube();
        GLRenderer(Context context) {
            this.context = context;
        }
```
}  
//Called over and over in rendering thread  
public void onDrawFrame(GL10 gl) {  
    //Clear screen to black  
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);  

    //positioning model1  
    gl.glMatrixMode(GL10.GL_MODELVIEW);  
    gl.glLoadIdentity();  
    gl.glTranslatef(-1.0f, 0.0f, -3.0f);  

    //draw model  
    cube.draw(gl);  
}  

//Called after surface is created and whenever surface size changes  
public void onSurfaceChanged(GL10 gl, int width, int height) {  
    //Define the view frustrum  
    gl.glViewport(0, 0, width, height);  
    gl.glMatrixMode(GL10.GL_PROJECTION);  
    gl.glLoadIdentity();  
    float ratio = (float) width/height;  
    GLU.gluPerspective(gl, 75.0f, ratio, 1, 100f);  
}  

//Called when OpenGL surface is created/re-created  
public void onSurfaceCreated(GL10 gl, EGLConfig config) {  
    //Setup OpenGL options needed  
    gl.glEnable(GL10.GL_DEPTH_TEST);  
    gl.glDepthFunc(GL10.GL_LEQUAL);  
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);  

    //Lighting  
    float lightAmbient[] = new float[] { 0.2f, 0.2f, 0.2f, 1 };  
    float lightDiffuse[] = new float[] { 1, 1, 1, 1 };  
    float[] lightPos = new float[] { 1, 1, 1, 1 };  
    gl.glEnable(GL10.GL_LIGHTING);  
    gl.glEnable(GL10.GL_LIGHT0);  
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, lightAmbient, 0);  
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, lightDiffuse, 0);  
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, lightPos, 0);
// Materials
float matAmbient[] = new float[] { 1, 1, 1, 1 };
float matDiffuse[] = new float[] { 1, 1, 1, 1 };
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, matAmbient, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, matDiffuse, 0);

// Enable textures
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glEnable(GL10.GL_TEXTURE_2D);
// Load the cube's texture from a bitmap
GLCube.loadTexture(gl, context, R.drawable.android1);
}

// GLCube class
package fyp.FYP;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;

public class GLCube {
    private final IntBuffer mVertexBuffer;
    private final IntBuffer mTextureBuffer;

    public GLCube() {
        int one = 65536;
        int half = one/2;
        int vertices[] = {
            // FRONT
            -half, -half, half, half, -half, half, half, half,
            -half, half, half, half, -half, half, half, half,
            // BACK
            -half, -half, -half, -half, half, -half, half, -half,
            -half, half, half, half, half, -half, half, -half,
            // LEFT
            -half, -half, half, -half, half, half, half, -half,
            -half, -half, -half, -half, half, half, half, -half,
            // RIGHT
            half, -half, -half, half, half, half, half, half,
            -half, half, half, half, half, half, half, half,
            // TOP
            -half, half, half, half, half, half, half, half,
ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
vbb.order(ByteOrder.nativeOrder());
mVertexBuffer = vbb.asIntBuffer();
mVertexBuffer.put(vertices);
mVertexBuffer.position(0);

int texCoords[] = {
    0, one, one, one, 0, 0, one, 0,
    one, one, one, 0, 0, one, 0, 0,
    one, one, one, 0, 0, one, 0, 0,
    one, one, one, 0, 0, one, 0, 0,
    one, 0, 0, 0, one, one, 0, one,
    0, 0, 0, one, one, 0, one, one,
};
ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4);
tbb.order(ByteOrder.nativeOrder());
mTextureBuffer = tbb.asIntBuffer();
mTextureBuffer.put(texCoords);
mTextureBuffer.position(0);

static void loadTexture(GL10 gl, Context context, int resource) {
    Bitmap bmp = BitmapFactory.decodeResource(context.getResources(), resource);
    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bmp, 0);
    gl.glTexParameteri(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
    gl.glTexParameteri(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
    bmp.recycle();
}

public void draw(GL10 gl) {
    //Tell OpenGL to use texture coordinates
    gl.glEnable(GL10.GL_TEXTURE_2D);
    gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, mTextureBuffer);
gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);

    gl.glColor4f(1, 1, 1, 1);
    gl.glNormal3f(0, 0, 1);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glNormal3f(0, 0, -1);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);

    gl.glColor4f(1, 1, 1, 1);
    gl.glNormal3f(-1, 0, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
    gl.glNormal3f(1, 0, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);

    gl.glColor4f(1, 1, 1, 1);
    gl.glNormal3f(0, 1, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
    gl.glNormal3f(0, -1, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
Appendix 2 – Camera Function

//Camera function called in the renderer
public static void camera (GL10 gl) {
    gl.glRotatef(_scene.camera().xrot, 1.0f, 0.0f, 0.0f);
    gl.glRotatef(_scene.camera().yrot, 0.0f, 1.0f, 0.0f);
    gl.glTranslatef(_scene.camera().xpos, -_scene.camera().ypos, -_scene.camera().zpos);
}

Part A – Directional Buttons Code

//Original alteration of camera position and rotation values (Buttons Method)
@Override
public void onClick(View v) {
    // TODO Auto-generated method stub
    if (v == findViewById(R.id.up)) {
        scene.camera().xrot -= 1;
        if (scene.camera().xrot < -360)
            scene.camera().xrot += 360;
    } else if (v == findViewById(R.id.down)) {
        scene.camera().xrot += 1;
        if (scene.camera().xrot > 360)
            scene.camera().xrot -= 360;
    } else if (v == findViewById(R.id.right)) {
        scene.camera().yrot += 1;
        if (scene.camera().yrot > 360)
            scene.camera().yrot -= 360;
    } else if (v == findViewById(R.id.left)) {
        scene.camera().yrot -= 1;
        if (scene.camera().yrot < -360)
            scene.camera().yrot += 360;
    } else if (v == findViewById(R.id.forward)) {
        double xrotrad, yrotrad;
        yrotrad = (scene.camera().yrot / 180 * 3.141592654f);
        xrotrad = (scene.camera().xrot / 180 * 3.141592654f);
        scene.camera().xpos += Math.sin(yrotrad);
        scene.camera().zpos -= Math.cos(yrotrad);
        scene.camera().ypos -= Math.sin(xrotrad);
    } else if (v == findViewById(R.id.back)) {
        double xrotrad, yrotrad;
        yrotrad = (scene.camera().yrot / 180 * 3.141592654f);
        xrotrad = (scene.camera().xrot / 180 * 3.141592654f);
        scene.camera().xpos -= Math.sin(yrotrad);
        scene.camera().zpos += Math.cos(yrotrad);
scene.camera().ypos += Math.sin(xrotrad);
}

Part B – Joystick Code

Note when the joystick was introduced the up and down movements were eliminated and the left and right alterations depend on the joystick movement.

//Variable alteration using the joystick method
@Override
public void OnMoved(int pan, int tilt) {
    // TODO Auto-generated method stub
    scene.camera().yrot += pan/2;
    scene.camera().zrot += tilt/2;
}
Appendix 3 – Screen to World Coordinates

// Picking Function
public Vec3 GetWorldCoords( Vec2 touch, Camera cam)
{
    // Initialize auxiliary variables.
    Vec3 worldPos = new Vec3();

    // SCREEN height & width (e.g.: 320 x 480)
    float screenW = cam.getScreenWidth();
    float screenH = cam.getScreenHeight();

    // Auxiliary matrix and vectors to deal with ogl.
    float[] invertedMatrix, transformMatrix, normalizedInPoint, outPoint;
    invertedMatrix = new float[16];
    transformMatrix = new float[16];
    normalizedInPoint = new float[4];
    outPoint = new float[4];

    // Invert y coordinate, as android uses top-left, and ogl bottom-left.
    int oglTouchY = (int) (screenH - touch.Y());

    /* Transform the screen point to clip space in ogl (-1,1) */
    normalizedInPoint[0] = (float) ((touch.X()) * 2.0f) / (screenW - 1.0));
    normalizedInPoint[1] = (float) ((oglTouchY) * 2.0f) / (screenH - 1.0));
    normalizedInPoint[2] = -1.0f;
    normalizedInPoint[3] = 1.0f;

    /* Obtain the transform matrix and then the inverse. */
    Matrix.multiplyMM(transformMatrix, 0, getCurrentProjection(r.gl()), 0, getCurrentModelView(r.gl()), 0);
    Matrix.invertM(invertedMatrix, 0, transformMatrix, 0);

    /* Apply the inverse to the point in clip space */
    Matrix.multiplyMV(outPoint, 0, invertedMatrix, 0, normalizedInPoint, 0);

    if (outPoint[3] == 0.0)
    {
        // Avoid /0 error.
        Log.e("World coords", "ERROR");
        return worldPos;
    }
}
// Divide by the 3rd component to find out the real
position.

return worldPos;
}

/* Record the current modelView matrix state. Has the side
effect of * setting the current matrix state to GL_MODELVIEW */
public float[] getCurrentModelView(GL10 gl)
{
    float[] mMModelView = new float[16];
    getMatrix(gl, GL10.GL_MODELVIEW, mMModelView);
    return mMModelView;
}

/*Record the current projection matrix state. Has the side
effect of * setting the current matrix state to GL_PROJECTION */
public float[] getCurrentProjection(GL10 gl)
{
    float[] mProjection = new float[16];
    getMatrix(gl, GL10.GL_PROJECTION, mProjection);
    return mProjection;
}

//Fetches a specific matrix from openGL
private void getMatrix(GL10 gl, int mode, float[] mat)
{
    MatrixTrackingGL gl2 = (MatrixTrackingGL) gl;
    gl2.glMatrixMode(mode);
    gl2.getMatrix(mat, 0);
}

//Camera Class
package com.modelTest;

public class Camera {

    Vec3 pos;
    float screenH, screenW;

    public Camera() {
        pos = new Vec3();
        screenH = screenW = 0;
    }
}
public void setScreenSize(float w, float h)
{
    this.screenH = h;
    this.screenW = w;
}

public float getScreenWidth() {
    return this.screenW;
}

public float getScreenHeight() {
    return this.screenH;
}

public float getX() {
    return this.pos.X();
}

public float getY() {
    return this.pos.Y();
}

public float getZ() {
    return this.pos.Z();
}

public Vec3 getPosition() {
    return this.pos;
}

//Vec2 class
package com.modelTest;

import android.util.Log;

//Two dimensional vector class.
public class Vec2
{
    public float x;
    public float y;

    //Empty constructor, initializes to 0
    public Vec2()
    {
        this.x = 0;
        this.y = 0;
    }

    //Creates an instance of the Vec2 class and initializes x and y
    public Vec2(float x, float y)
```java
public Boolean equals(Vec2 v)
{
    if (this.x == v.x && this.y == v.y) return true;
    else return false;
}

public float length()
{
    return (float)Math.sqrt((this.x*this.x + this.y*this.y));
}

public void normalize()
{
    float len = length();
    this.x /= len;
    this.y /= len;
}

public Vec2 getvectorTo(Vec2 point)
{
    Vec2 aux = new Vec2();
    aux.SetX(point.x - this.x);
    aux.SetY(point.y - this.y);
    return aux;
}

public Vec2 getvectorTo(int x, int y)
{
    Vec2 aux = new Vec2();
    aux.SetX(x - this.x);
    aux.SetY(y - this.y);
    return aux;
}

// Sets the x,y
public void set(float x, float y)
{
    this.x = x;
    this.y = y;
}
```
```csharp
this.x = x;
this.y = y;
}

//Sets the x
public void SetX(float x)
{
    this.x = x;
}

//Sets the y
public void SetY(float y)
{
    this.y = y;
}

//Adds the offset to the current position
public void Offset(float x, float y)
{
    this.x += x;
    this.y += y;
}

//Gets the x value
public float X() { return this.x; }

//Gets the y value
public float Y() { return this.y; }

// Prints the vector value to the log
public void Print(String tag, String msg)
{
    Log.i(tag, msg + ": " + this.x + ", " + this.y);
}

// Calculates the dot product of this Vec2 with another
public float Dot(Vec2 vec)
{
    return (float) (this.x * vec.X() + this.y * vec.Y());
}

//Adds to this Vec2 the values of another
public void Add(Vec2 vec)
{
    this.x += vec.X();
    this.y += vec.Y();
}

//Multiplies the x and y components by the value
public void Scale(float val)
{
```
```java
this.x *= val;
this.y *= val;
}

// Gets a Vec2 with the truncated values of the float coordinates
public Vec2 GetIntValue()
{
    Vec2 intVec = new Vec2();
    intVec.Set((int)this.x, (int)this.y);
    return intVec;
}

// Checks if the rounded coordinates of both vectors are equal
public boolean RoundEqual(Vec2 vec)
{
    return (Math.round(this.x) == Math.round(vec.X()))
    && (Math.round(this.y) == Math.round(vec.Y()));
}
}

// Vec3 Class
package com.modelTest;

import android.util.Log;

// Three dimensional vector class.
public class Vec3
{
    public float x;
    public float y;
    public float z;

    // Empty constructor, initializes to 0
    public Vec3()
    {
        this.x = 0;
        this.y = 0;
        this.z = 0;
    }

    // Creates an instance of the Vec3 class and initializes x and y
    public Vec3(float x, float y, float z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
public boolean Equals(Vec3 v) {
    if(this.x == v.x && this.y == v.y && this.z == v.z)
        return true;
    else return false;
}

public float Length() {
    return (float)Math.sqrt((this.x*this.x + this.y*this.y + this.z*this.z));
}

public void Normalize() {
    float len = Length();
    this.x /= len;
    this.y /= len;
    this.z /= len;
}

public Vec3 GetVectorTo(Vec3 point) {
    Vec3 aux = new Vec3();
    aux.SetX(point.x - this.x);
    aux.SetY(point.y - this.y);
    aux.SetZ(point.z - this.z);

    return aux;
}

public void Set(float x, float y, float z) {
    this.x = x;
    this.y = y;
    this.z = z;
}

public void SetX(float x) {
    this.x = x;
}
// Sets the y
public void SetY(float y)
{
    this.y = y;
}

//Sets the z
public void SetZ(float z)
{
    this.z = z;
}

//Adds the offset to the current position
public void Offset(float x, float y, float z)
{
    this.x += x;
    this.y += y;
    this.z += z;
}

// Gets the x value
public float X() { return this.x; }

// Gets the y value
public float Y() { return this.y; }

// Gets the z value
public float Z() { return this.z; }

// Prints the vector value to the log
public void Print(String tag, String msg)
{
    Log.i(tag, msg + ": " + this.x + ", " + this.y + ", " + this.z);
}
}