Real Time Augmentation of Video with 3D Graphics

Paul Shirley
B.A.I. Engineering
Final Year Project  April 2011
Supervisor: Fergal Shevlin

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

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

__________________________________________  _________________________
Paul Shirley                                    April 2012
I agree that the Library and other agents of the College may lend or copy this thesis upon request.

Paul Shirley

April 2012
I would like to thank my supervisor Fergal Shevlin for all his support during this project. His constant advice and guidance have been a great help. I would also like to thank Ivan Shirley for spending the time to proof read this report.

I would like to thank my friends and family for their support during this time.
This project looked at the use of augmented reality applications on smartphone devices. A process was developed using OpenGL graphics library and OpenCV vision library to map 3D models onto a live video stream. The aim was to augment furniture in a room so users could visualize what these items of furniture would look like in their homes. The project investigated this on smartphone devices and experimented with sensors in these devices to improve the results. This application worked on the test device and shows that there is a lot of potential for this type of development, however, more research is required to refine this process.
# TABLE OF CONTENTS

Declaration .......................................................................................................................... i

Permission To Lend .............................................................................................................. ii

Acknowledgements .......................................................................................................... iii

Abstract ............................................................................................................................ iv

Table of Contents .............................................................................................................. v

Table of Figures ................................................................................................................ vii

Introduction ....................................................................................................................... 1

  Background ...................................................................................................................... 1

  Aim .................................................................................................................................. 2

Similar Projects ................................................................................................................ 4

Graphics .............................................................................................................................. 7

  OpenGL ......................................................................................................................... 7

Using OpenGL for Augmentation on Android ................................................................. 9

  Virtual camera positioning .......................................................................................... 12

Min3D model Loader ....................................................................................................... 15

3DS Max ............................................................................................................................ 17

  Lighting ......................................................................................................................... 18

Vision ................................................................................................................................. 19

  OpenCV ......................................................................................................................... 19

Native Camera .................................................................................................................. 21
Pose Estimation ...........................................................................................................23
Camera Calibration .........................................................................................................25
Object Tracking ...............................................................................................................27
user Interface and Sensor Data ......................................................................................29
User Interface .................................................................................................................29
Sensors .............................................................................................................................30
Presentation .....................................................................................................................32
Conclusions .....................................................................................................................33
Evaluation .......................................................................................................................33
Future Work .....................................................................................................................35
Skills Gained ....................................................................................................................36
Bibliography ...................................................................................................................37
Appendix 1 – CD Contents .............................................................................................40
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Couch Model</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>AndAR</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>AugmenteDev</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Edibear</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>OpenGL in Android Architecture</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>View Hierarchy</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Model Overlayed</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>3D Coordinate Systems</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>Model view Matrix</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>Projection</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Min3D</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>3DS Max</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>3D Lighting</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>OpenCV logo</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>Native Camera in Android Architecture</td>
<td>21</td>
</tr>
<tr>
<td>16</td>
<td>Pose Estimation</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>Camera Calibration</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>Find Features</td>
<td>27</td>
</tr>
<tr>
<td>19</td>
<td>User Interface</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>Raw accelerometer sensor data</td>
<td>30</td>
</tr>
</tbody>
</table>
Figure 21 - Sensor Data after low pass filter .......................................................... 30

Figure 22 - Prezi and Android Assets Studio .......................................................... 32

Figure 23 - Screenshot .......................................................................................... 33

Figure 24 - Screenshot showing augmented model ................................................. 34
INTRODUCTION

BACKGROUND

Over the past few years the technology in mobile phones has advanced greatly. Many smartphones on the market today are as powerful as some small computers and laptops. These smartphones run their own optimized operating systems so they are capable of achieving very impressive results.

Modern smartphones also incorporate a wide range of technologies that have never been combined in devices before now. These technologies include high quality cameras, touch screens and a wide array of sensors\(^1\). The sensors can determine the location, direction, the light and sound levels, the orientation and the movements of the phone.

By combining all these features there is a lot of potential for augmented reality applications. Augmented reality is a live view of the world combined with computer generated content. The computer generated content is defined by the surroundings and interacts with them. Computers have been used to do image and vision processing for a long time. Everyday smartphone users generally do not carry out this type of operation. With these developments, powerful augmented reality applications can properly be put in the hands of everyday users.

\(^{1}\) (Apple)(Google)
The title of this project is “Real time augmentation of Video with 3D graphics”. It looks at using augmented reality and smartphones to bring furniture stores into your living room. This idea would allow a customer to browse a furniture store catalogue on their phone, and then view, live on the screen, what an item of furniture might look like in their room. This project looks at the augmented reality aspect of this. It is important to make the application accessible and user friendly. Some augmented reality applications take the approach where the user is required to print off a marker sheet for the application to detect, which is an inconvenient process. This project is going to attempt to tackle the problem of marker-less augmented reality.

For this project the Android platform was selected. Android\(^2\) is a mobile operating system which runs on many smartphones. It is an open source platform\(^3\) and has a strong support for developers. OpenCV\(^4\) is one of the industry’s standard frameworks for computer vision processing. OpenCV has been fully ported to Android. Along with this, Android also has 3D graphic libraries built into it. These features combined make Android an ideal testing environment for this project. OpenCV is also ported to other mobile phone operating systems.

\(^2\) (About Android)
\(^3\) (Android Philosophy and Goals)
\(^4\) (OpenCV, 2012)
systems such as iOS\(^5\), but this is not as well supported. iOS is the operating system which runs on Apple devices such as the iPhone and the iPad. This does however mean that any application developed on Android could be then ported to iOS.

This project aims to explore the true power of smartphone devices and to investigate the potential of augmented reality applications that can be used practically by everyday smartphone users.

The project has many different aspects to it and each presents their own unique set of challenges. It has elements of mobile app development, 3D graphics, computer vision, and processing sensor data. These technologies need to be studied and mastered as they need to eventually work together to create a robust and reliable product. For this project there are two major sections which are GRAPHICS and VISION. Other aspects include the User Interface and Sensors.

\(^{5}\) (eosgarden, 2012)
AndAR\textsuperscript{6} is an augmented reality project for the Android platform. It is an open source project and works by printing off a marker sheet. The camera then detects the marker sheet and overlays the model on top. When the model is loaded it makes the model look as if it is actually there. As the user moves the camera around, the model looks stationary on the marker. AndAR allows for some simple models to be loaded in, and also for the developer to create and design their own markers.
AugmenteDev⁷ is a company that has built augmented reality smartphone applications for Android and iOS. Similarly to AndAR, it requires the printing off of a marker sheet. 2D or 3D models are augmented on the marker.

The AugmenteDev app allows users to create their own simple models by adding pictures to the sides of a cube. These models can then be uploaded to the internet for others to use. As a result there is an increasingly growing list of models available. In Figure 3 we see that many different tablets devices are simulated side by side in the scene.

---

⁷ (AugmenteDev)
Edibear$^8$ is a marker-less augmented reality interactive pet. It generates a 3D model of the bear which will move around the screen and perform a few other tasks. As this application is marker-less, there is no need to print in order to be able to use the app. The application can also be used in many settings.

To use Edibear the user has to position the camera looking down from above onto the scene. The user then touches the screen to signify that the phone camera is in this starting position. The model of the bear will then load and the user can move the camera to wherever they want, keeping the bear in the picture. The bear can be moved around the scene and the original initialization area doesn’t even need to be kept in the screen.

This is a very advanced implementation of augmented reality. It uses a closed source augmented reality toolkit which is designed only for Samsung mobile devices.

---

$^8$ (Eduard, et al., 2011)(Oks, 2011)
The graphics aspect of this project involves creating and loading 3D models into Android. These models are overlayed on a live camera view. It also maps the location and the rotation of the model to match the results obtained in the vision section of the project. OpenGL is the graphics library used for this project.

OpenGL

OpenGL is a 2D and 3D graphics library. It is implemented on a wide variety of computer platforms and also, the fact that it is open source, OpenGL is widely used and greatly supported in the computer industry.

OpenGL is a big package with extensive functionality. This proves great for computers and gaming consoles but for smaller devices it is often too big. There is a reduced version of OpenGL which is designed for less powerful devices called OpenGL ES. The ES stands for

---

9 This image is reproduced from work created and shared by the Android Open Source Project and used according to the terms described in the Creative Commons 2.5 Attribution License. (Android Open Source Project, 2012)
10 (Khronos Group, 2012)
embedded systems. OpenGL is the driver behind 3D graphics and often 2D graphics on many smart phone devices including iPhone, Android and Blackberry. As seen in Figure 5 OpenGL is a major part of the Android Architecture. OpenGL ES 1.0, 1.1, and 2.0 are available on nearly all Android devices. On some very old devices, only OpenGL ES 1.0 is available. These old devices only account for 1.2% of all Android devices currently in use\textsuperscript{11}.

\textsuperscript{11} (Android Open Source Project, 2012)
Augmented reality is the merging of a live camera picture with computer generated content. The computer generated content usually interacts with the live camera picture in some form. This project uses augmented reality to overlay a 3D model on a live camera picture and position the model so it fits in the scene.

There are a few different ways of displaying a live camera behind a 3D model. With regard to augmented reality with OpenGL, a textured skybox is often used. Before OpenGL renders the models it draws a 2D texture in the background.

This method however was very difficult to implement on Android as it required a lot of conversions. The first conversion was to convert the camera image into the correct texture format for OpenGL. Android’s camera generally uses the YUV image format while the texture has to be in the RGB format. The second conversion was from the RGB image to a texture. These conversions need to take place every frame. After preliminary trials, it was decided not to use this method for the project because of the complexity and the processing required.
Android uses a view hierarchy\textsuperscript{12} to layout the screen and render it as seen in Figure 6. Each screen can have many different views within it and the user can interact with the each view in different ways. There are many different predefined views, and custom views can also easily be defined. Views can be created inside or alongside other views. If a view has other view within it these are referred to as children. For example a button in android is itself a predefined view. A LinearLayout is another view type. The views inside of a LinearLayout are laid out beside each other. If two buttons were children of a LinearLayout they would appear beside each other. A frame view is another view type which allows its children to be overlayed on top of each other.

OpenGL on Android is generally rendered in a specially defined view called a GLSurfaceView. This is a view class which is optimized for using OpenGL and runs on a separated thread, meaning it doesn’t prevent the other processes from running. Camera previews are often shown in a SurfaceView. A SurfaceView is a dedicated view class which is designed for drawing. By creating a custom surface view the camera image was able to be displayed on the screen.

\textsuperscript{12} (User Interface, 2012)
Using a frame view it is possible to overlay views on top of each other. When the GLSurfaceView and the camera view were overlayed on top of each other initially this didn’t work. It would only show one view at a time.

The problem here occurred because SurfaceViews and GLSurfaceViews are treated differently than other views. To overlay them it was necessary to change the z-ordering a different way than what is traditionally done. This means that they can be placed on top of each other in a defined order. By changing the background colour of the GLSurfaceView and by changing settings of the GLSurfaceView to translucent, it was then possible to have a correctly overlayed image.

Figure 7 - Model Overlayed

In Figure 7 the overlay is demonstrated clearly. The view seen in this matches the view hierarchy seen in Figure 6.
VIRTUAL CAMERA POSITIONING

Figure 8 - 3D Coordinate Systems

To get the model to position itself correctly the virtual camera needs to be set up so that its position and rotation matches that of the real camera. This is calculated in the vision section.
OpenGL maps points in 3D space by using a right hand coordinate system. It stores the current position, rotation and scale of the model in a matrix\(^\text{13}\). When the model is initially loaded it loads the identity matrix. The left, up, and forward vectors in the matrix (as seen in Figure 9) define the rotation, while m12-m14 define the position, and finally m3, m7 and m11 define the scale.

![Figure 9 – Model view Matrix](image)

OpenGL doesn’t have a virtual camera\(^\text{14}\). Instead the eye space coordinate always remains at (0,0,0). This means that instead of moving the camera the model has to be moved in the opposite direction. OpenGL also uses other matrices, one of which is the projection matrix. The projection matrix sets up the viewport by defining the angle of view that the camera can see and the range in which objects will be visible. The projection matrix also looks at the aspect ratio of the screen so it can scale the model appropriately. OpenGL provides functions to easily set this viewport up. The aspect

---

\(^\text{13}\) (Ahn, 2008), (FAQ of OpenGL Transformations)

\(^\text{14}\) (Using Viewing and Camera Transforms, and gluLookAt())
ratio is found simply by dividing the number of pixels horizontally by the number of pixels
vertically. The angle of view is normally around 45°. Android does provide accurate
information of the camera parameters so it is possible to improve the accuracy. Ultimately
OpenGL should simulate the real camera as close as possible.

The vision section of this project computed the pose of the scene. This produced a 3x3
rotation matrix and a translation vector, giving rotation and position. The results provided
by the vision aspect used a row ordered matrix while OpenGL used a column ordered
matrix. This meant the matrix had to be transposed. For this to work in OpenGL, some of the
signs needed to be changed in order to position the model correctly. It took some testing to
work out how the different matrices correspond with each other. The results are shown
below.

\[
\text{Rotation matrix: } \begin{bmatrix} r_0 & r_3 & r_6 \\ r_1 & r_4 & r_7 \\ r_2 & r_5 & r_8 \end{bmatrix} \quad \text{and translation vector: } [t_0 \quad t_1 \quad t_2] \quad \text{translates to}
\]

\[
\begin{bmatrix}
 r_0 & r_1 & r_2 & t_0 \\
 -r_3 & -r_4 & -r_5 & -t_1 \\
 -r_6 & -r_7 & -r_8 & -t_2 \\
 0 & 0 & 0 & 1
\end{bmatrix}
\]
Min3D\textsuperscript{15} is an open source framework for Android which allows you to import and load 3D models into your applications. It uses OpenGL ES 1.0 so it is supported by all Android devices. Min3D allows the importing of models and textures from many different formats. These include `.OBJ` files and `.3DS` files. Most 3D graphics modelling software, such as 3DS max and Blender, will export models to these formats. These are universal and widely used formats. This makes it straightforward enough to get models displaying on Android devices.

Min3D uses its own functions and classes to set up the scene, set up the lights, position the camera, and draw the models. The first task was to simply get a model loaded on Android. This proved straightforward enough as there are many tutorials\textsuperscript{16} posted online and also sample code provided with the framework itself.

The next step was to get the model loaded on top of the camera preview. The camera code was set up to work with OpenGL as shown previously. Even though Min3D has its own functions and classes, the method for making the background translucent was similar. This

\textsuperscript{15} (Min3D) \textsuperscript{16} (MatD, 2011)
allowed the model to be displayed over the camera preview as it involved reproducing the
steps used earlier to achieve the same goal.

The final step to loading the model was to manipulate the model to match the scene. The
camera calculations are all worked out by the use of sensor and camera data. This is then
used to manipulate the model thus matching the scene.

Min3D uses its own special functions and classes to manipulate the camera. Unfortunately
the way Min3D loads the camera wasn’t compatible with the information received from the
cameras and sensors. To overcome this difficulty it was necessary to study the Min3D code
and understand how it worked. After extensive study the Min3D code was changed to make
it compatible.

OpenGL stores all transformations, rotations, and scaling of the scene in a matrix as
discussed earlier. Min3D uses an OpenGL function called \textit{gluLookAt}^{17}. This takes in the
position you want the camera to be, the target the camera is looking at, and an up vector.
This function creates the corresponding matrix and loads it. This function is not of any use as
the data received is already in the matrix format. To load the camera it involved changing
the function from \textit{gluLookAt} to a function which loaded the proper matrix called \textit{glMultMat}.

The final step in the model loading aspect of the project was to load a model of some
furniture. It was possible to source models online. Many models didn’t come with texture
files so it was necessary to edit them using software such as 3DS max.

\footnote{Using Viewing and Camera Transforms, and gluLookAt()}
Autodesk’s 3DS Max\textsuperscript{18} is extremely powerful 3D modeling software. It is the industry standard for creating models. 3DS Max has a extensive features and a lot of very advanced rendering effects.

With 3DS Max it is also possible to edit models, such as models downloaded off the internet. This is important because it is necessary to scale models so that they are an accurate size when inserted into the scene. It is also sometimes necessary to apply textures because not all models come with the texture files.

When the model is finished it can be exported to a .OBJ file. This exports the model, a texture reference file, and the actual texture. Android has a very specific file system, so the model files have to be put in a separate folder to the texture image. To point the model to the texture image it is necessary to open the texture reference file with a text editor and change the location of the texture file to match the updated location.

\textsuperscript{18} (Autodesk, 2012)
Lighting is a key aspect of 3D graphics. It needs to be correct in order for the graphics to look real and legitimate. OpenGL allows you to define lights in your scene except, these don’t necessarily match up with the lighting in the actual room.

Specifying the exact location of the light is a difficult procedure. To make the models in the project look proper a few predefined lighting locations were specified. With these specified it was possible for the user to select approximately where the light was coming from. When the lighting is set up correctly, the model displays a lot better and the shadows on the model look correct.
VISION

The vision aspect of this project takes in a live video stream from the camera. It tracks points from the stream and uses them to calculate where the 3D model should be placed. OpenCV\textsuperscript{19} is a vision library which is used to assist with this task.

OPENCV

OpenCV is a computer programming library. It contains over 2500 optimized functions that deal with different aspects of computer vision. These functions deal with problems and tasks such as facial recognition, object tracking, human-computer interaction, 3D motion, detection of cameras, stereo vision and much more. OpenCV has C, C++ and python interfaces which run on Windows, Linux, Android and Mac.

OpenCV on Android originally started as a port of OpenCV and was a separate project. As this port evolved it was merged with the main branch of OpenCV. Android allows development in native code so a majority of OpenCV’s functionality can be run on Android. There are a couple of functions that rely on external dependencies (e.g. video codecs) which sometimes cause issues.

Android mainly runs Java applications in a virtual machine. It also allows for development of applications that access and run native C and C++ code\textsuperscript{20}. OpenCV requires native code

\textsuperscript{19} (OpenCV, 2012)

\textsuperscript{20} (OpenCV, 2012)
development to run on Android, but they have provided an SDK which allows the most common functions to be called from Java. This makes getting started on OpenCV a lot easier for people. Unfortunately the whole code base isn’t provided in this manner. Even though the SDK exists, to create fully optimized OpenCV code on Android, it is advised to write applications in native code. This project used a mixture of native code and the SDK.

(What is the NDK?, 2012)
OpenCV\textsuperscript{22} has a couple of ways to use the camera on Android devices. The first is to use the Android API calls and then to convert the image to a format which suits OpenCV. Another option is for OpenCV to use the native camera. To use the native camera OpenCV provides its own driver for Android. When the native camera is used OpenCV can call the camera as it traditionally would on a computer.

The latter option was used for this project. The reason being is it imported the video frames straight into an OpenCV matrix and in the correct colour format. The OpenCV matrix is required for the majority of the OpenCV functions. The video capture was set up in a surface view and set to read in the frames and process them on a separate thread. Choosing this

\textsuperscript{21} This image is modified from work created and shared by the Android Open Source Project and used according to the terms described in the Creative Commons 2.5 Attribution License. (What is Android?, 2012) \textsuperscript{22} (OpenCV on Android, 2012)
method meant that the camera would refresh at a rate it deemed possible and would take the input frames as it needed them.

This method has many advantages and proved more efficient for making OpenCV applications. It sometimes has disadvantages. Recently a new version of Android came out called Ice Cream Sandwich. It dramatically changed the way the camera was initialized, possibly for performance reasons. As a result, it took some time before OpenCV supported the native camera on it. This means applications might be slower to be updated for newer devices.

In Figure 15 the areas of the Android framework that are used by the OpenCV native camera can be seen highlighted. The majority of Android applications are run in the Dalvik virtual machine. This runs Java based applications. OpenCV however utilizes a lot more of the Android architecture than standard applications. It uses native code to interact with the camera at the kernel level. OpenCV is entirely written in native code but as discussed earlier there is an SDK that makes running common OpenCV functions much easier when only working in the Dalvik Virtual Machine.
To match the 3D model with the image, the pose of the camera needed to be calculated. When the two cameras are matching, and the scale of the model matches the size of the room, then the 3D model should fit in the room in the correct position. Much research has been done, and many algorithms and functions have been made that can estimate the pose in an image. These functions require four or more known, noncoplanar feature points.

OpenCV has a few functions which can calculate the pose based on known points. This is often referred to as finding the cameras extrinsic parameters. The extrinsic parameters are the position and orientation of the camera.

The OpenCV function used to find the camera extrinsic parameters was called solvePNP. This function is actually the same as a legacy function called findCameraExtrinsics. The solvePNP function parameters are listed below. To successfully calculate the extrinsic parameters the input points need to be known. The camera intrinsic parameters also need to be known. The intrinsic parameters give the focal length, the field of view and the centre point of the camera. How these parameters are determined are discussed in the Camera Calibration section.

- Object Points – A list of points in 3D space that correspond to the object
- Image Points – The points on the image matching the object points

---

23 (DeMenthon & Davis), (Schweighofer & Pinz, 2005)
24 (Calib3d, camera calibration, pose estimation and stereo), (Posit Pose Estimation)
Camera Matrix – The intrinsic parameters of the camera need to be known
Distortion – This is to remove any known distortion on the camera or can be set to null
Rotation Vector – The output vector of the rotation
Translation Vector – The output vector of the position
useExtrinsicGuess – This last parameter is a Boolean value. If it is set to true it uses the previous Rotation and Translation vector in the calculation which makes it much more accurate and smooth for augmented reality applications.

The solvePNP function gives a rotation and translation vector. The rotation vector needs to be translated into a rotation matrix. There are functions built into OpenCV to do this. For augmented reality the useExtrinsicGuess aspect of this function is very important as it reduces the shake and makes the results more accurate and steady over time. To use this function it is necessary to set up default values to begin with and that all variables are initialized. Many errors occur otherwise.

This function worked great by the end but provided many problems and required many tweaks along the way. The first achievement was that the function did interact with the model and changed the position of it, yet it was not in the correct position or the correct rotation. Initially only 4 points were used and the model didn’t match up very well. It wasn’t until 6 points were added that the results were reasonable. Even with the six points however the scale wasn’t correct. After calibrating the camera correctly the scale became a lot more accurate.
To calculate the extrinsic camera parameters successfully some information on the camera itself is needed. The information that is needed is the cameras intrinsic parameters and also a camera distortion matrix.

The camera intrinsic properties are stored in a 3x3 matrix which is shown below\textsuperscript{25}.

\[
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\]

Where \( c_x \) & \( c_y \) are the coordinates of the centre point of the lens and \( f_x \) & \( f_y \) are the focal length multiplied by a scaling factor based on the field of view and the image aspect ratio.

OpenCV has a function called \texttt{findChessboardCorners}. This function can be seen working in \textit{Figure 17}. To accurately find the camera intrinsic parameters multiple pictures of a chessboard can be taken. Bundled with OpenCV is a small program which will take in the images, detect the corners, and return the correct camera intrinsic parameters matrix along

\textsuperscript{25} (Calib3d, camera calibration, pose estimation and stereo), (Single Camera Calibration, 2011)
with a camera distortion matrix. The test device for this project is a *Samsung Galaxy S2*. The results for the calibration are shown below.

\[
\text{Intrinsic Parameters} = \begin{bmatrix}
5.6143 \times 10^2 & 0 & 3.238 \times 10^2 \\
0 & 5.6695 \times 10^2 & 2.4077 \times 10^2 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\text{Distortion Matrix} = \begin{bmatrix}
2.4887 \times 10^{-2} \\
1.077 \\
-6.10291 \times 10^{-6} \\
4.1592 \times 10^{-3} \\
-5.2316
\end{bmatrix}
\]

Calculating these values is not something the user would generally want to be doing. The distortion matrix isn’t actually necessary as the camera itself tends to fix distortion before the image reaches OpenCV. The results seen here are only minor changes and could also be a result of distortion in the chessboard image.

We can use the intrinsic parameters here to make a good guess at the camera matrix for other devices. To do this we can use the functions:

\[
f_x = f \times m_x \quad \text{&} \quad f_y = f \times m_y
\]

Where \( f \) is the focal length and \( m_x \) & \( m_y \) are scaling factors. Using Android API calls we determine that the focal length is 3.97mm for the *Samsung Galaxy S2*. We can use this to determine scaling factors which can be used across all devices. To estimate \( c_x \) & \( c_y \) we use:

\[
c_x = \frac{imageWidth - 1}{2} \quad \text{&} \quad c_y = \frac{imageHeight - 1}{2}
\]

This allows us to create an accurate estimation of the camera matrix. Built into the application the actual calibration matrix could be included for the most popular devices and then the estimation could be used for all the others.
OpenCV provides many different methods of tracking points, many of which are based around the find features algorithm. The find features algorithm (as seen in Figure 18) scans an image and picks out unique features. These features are generally features that can be rediscovered in similar photos.

The algorithm that was used to track and update the scene was a function that tracks the optical flow of the camera. This function takes an input set of points on one frame, and maps them to the corresponding points on the next frame. This function is very useful for 3D graphics as it can determine the movement of the camera and the direction in which it has moved.

This algorithm proved very useful as the points that are entered to position the model at the beginning, can be tracked frame by frame. This gives the correct points to recalculate the pose. This method proved very effective. When run on a Samsung Galaxy S2 it didn’t produce any lag when the feature tracking and pose calibration were run every frame. This

26 (Find Features)
method did cause problems when the area surrounding a point was plain. There weren’t enough features to accurately track the point from frame to frame.

This method proved efficient on the test device because the video remained smooth while doing calculations on every frame. However this is a high end and powerful device. On less powerful devices it would be interesting to see how efficient these algorithms are. To overcome the need to recalculate the pose each frame sensor data can be used.

The find features algorithm is implemented in native code while the calculating optical flow function is implemented using the SDK. This is a good example of using both the OpenCV Java SDK and native code.
The user interface for this project was very basic. It involved the camera view and 3 buttons. The first button was for taking a screenshot, the second button was for entering points on the display, and the final button was to specify the lighting.

The user could input points on the screen which corresponded with corners in their room. To make it easier for the user to see what was going on different colours were used to signify different planes. Once the points are inputted the configuration button has to be pressed to show the model. If the user wants to enter new points they can do so by toggling the configuration button again.

The user interface in its current form is designed for development, testing and demonstration. This user interface has many weaknesses and requires much development. The points have to be inputted in a specific order which isn’t shown to the user.
Figure 20 – Raw accelerometer sensor data

Figure 21 - Sensor Data after low pass filter
There are many different types of sensors in Android devices. The most common sensor is the accelerometer. This measures force along the X, Y, and Z planes. In Figure 20 sample data for the accelerometer is shown. While recording, the phone was in a horizontal position pointing slightly downwards. The forces that are seen are due to gravity.

As you can see from Figure 20 the data isn’t uniform and the results jump around the place. To make the result a lot more accurate and smoother a simple algorithmic low pass filter was used. This is a basic filter used to create clearer signals. The code used is shown below.

\[ x'(t) = x(t) \times f + x'(t - 1) \times (1 - f) \]

Where:  
- \( x(t) \) is the sensor data at time t.  
- \( x'(t) \) is the value at time t when passed through the low pass filter.  
- \( f \) is the filtering factor. 0.05 was used.

Figure 21 shows the same values as Figure 20 when passed through a low pass filter. The results are greatly improved. After many tests the filtering factor of 0.05 was determined to be the optimum value smoothened out the curve without having an adverse affect on the data. You will notice at one point in the graph that the X axis drops suddenly and the Z axis peaks simultaneously. This occurred during the recording of the data when a sharp movement was induced. This shows that the accelerometer is capable of detecting movements.

Vision tracking can be a heavy process for computers. For smartphones the same applies. By using the sensor data it is possible to reduce the need to track the points and recalculate the position every frame. The sensor data provides enough information to move the model slightly. When the accelerometer has detected changes above a certain threshold it requests a frame update. This method had some good results even though the model moves around in small amounts. This is due to the old position from the sensor and the new position from the camera not matching up perfectly. Determining a reasonable threshold, which reduces the movements but also reduces the processing requirements, is difficult. For low powered smartphone devices this will definitely improve the overall experience. With less power demand the video will be much smoother.

---

27 (Google), (Sensors, 2012)
For the presentation two different tools were used. The first was for the presentation itself. This tool was called Prezi\textsuperscript{28}. Prezi is a presentation tool with the aim to make presentations more aesthetically pleasing and more interesting. Prezi provides a simple user interface to design and layout presentations. It includes tools to determine the path that the presentation will take. It supports text, images and video. The Prezi presentation is included in the appendix.

The second tool was called Android Asset Studio. Android assets studio contains a set of tools for creating Android icons and other pictures. One such tool allows you to frame screenshots inside pictures of devices like seen in Figure 22.

\textsuperscript{28} (Prezi)
The aim of this project was to develop a marker-less augmented reality application for smartphones. The aim of the application was to augment furniture in a room, so users could visualize what these items of furniture would look like in their homes. There were also a few secondary aims of this project. These aims were to investigate the practicality and potential of augmented reality applications on smartphones, and to investigate the use of sensors in making the results more accurate.
Overall the project was a success. A model of a sofa was loaded and manipulated so it looked like it was sitting in the room. Feature tracking and using sensor data meant that the model stayed in position, within reason, as the phone was moved around. Although the project worked there is still a lot of refinement needed. The model loads into the correct position but often the tilt is slightly off. This incorrect tilt can be seen in Figure 24. Figure 23 shows a more idealistic representation. The feature tracking works well but if it loses the points it isn’t capable of finding them again. It also struggles if the scene is plain and lacks features that it can repeatedly detect.

Even with the errors, the project shows that modern smartphone devices have more than enough processing power to compute vision problems in real time. This means there is great potential for augmented reality applications.

Using the sensor data proved difficult yet it still has many advantages in being used. This project explored using the sensor data to move the model as opposed to recalculating all vision algorithms each frame. As discussed earlier the use of sensor data does have the merits of reducing the processing power but also induces extra undesired movements. With further research and the development of some tests to determine optimum calibration of the sensor data, this could be significantly improved. One such calibration test would be to record a video and sensor data simultaneously. This would then be used to create a relationship between the sensor movements and the screen movements.
FUTURE WORK

There are many tasks that need to be completed before this application is fully functional and is suitable for use. The next steps for this application is to build a well designed user interface, enable loading models from other sources such as the internet, and refining the vision aspects.

The user interface is currently optimized for testing and development but doesn’t provide information for a user to use the application without been giving prior instructions. The aim would be to create a simple to use, informative, and intuitive user interface to bring this application to the next level.

As the project was being developed the dynamic loading of models was always an end goal. Currently there is one model that can be used in the application. Ideally many models could be loaded from many sources. Because this end goal was always part of the design it means that implementing the functionality to load the models won’t be too difficult. Another aspect to this is creating a catalogue and a database to support it. This would allow one to download only the information and models that they require.

There will always be many vision problems that can be tackled to improve the overall user experience. To refine the vision aspect of the project the accuracy of the model positioning needs to be increased, and the image tracking needs to be improved and made more robust. When these tasks are completed it will then be possible to do further research into the vision aspect. This further research would aim to reduce and simplify the tasks a user needs to complete in order to successfully show the models. Some areas for future research would be to generate and map the surroundings, and to implement stereo tracking.
SKILLS GAINED

Many technologies were explored and learned for this project. These include:

- Android development including developing native code on Android
- 3D graphics using OpenGL
- Vision processing using OpenCV
- Processing sensor data in Android

Overall this project was very challenging yet it proved to be extremely interesting and enjoyable.


http://www.augmentedev.com/drupal7/

http://usa.autodesk.com/3ds-max/features/


http://www.samsungapps.com/topApps/topAppsDetail.as?productId=G00002052831

eosgarden. (2012). OpenCV for iOS. Retrieved March 2012, from eosgarden:

http://www.google.com/nexus/#/tech-specs

http://www.opengl.org/about/


http://www.opengl.org/archives/resources/faq/technical/viewing.htm

http://www.mat-d.com/site/tutorial-load-a-3d-obj-model-with-min3d-for-android/

http://code.google.com/p/min3d/


The contents of the CD and the main components are listed below.

<table>
<thead>
<tr>
<th>File/Folder</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• fyp_workspace</td>
<td>Contains the code and dependencies of the project</td>
</tr>
<tr>
<td>o OpenCV-2.3.1</td>
<td>Library of OpenCV for using with Android</td>
</tr>
<tr>
<td>o fyp</td>
<td>Code for final year project</td>
</tr>
<tr>
<td>▪ jni</td>
<td>Native Code</td>
</tr>
<tr>
<td>▪ res</td>
<td>Layout files, images, and models</td>
</tr>
<tr>
<td>▪ src</td>
<td>Source code for project and model loader</td>
</tr>
<tr>
<td>o calibrateCamera</td>
<td>An application to take camera calibration pictures</td>
</tr>
<tr>
<td>• Presentation</td>
<td>The Prezi presentation</td>
</tr>
</tbody>
</table>