Dissertation in Augmented Reality

Emmanuel Adegboyega Obayomi

2109589

A thesis submitted for the degree of Master of Science in Artificial intelligence

Supervisor: Dr. Adrian Clark

School of Computer Science and Electronic Engineering

University of Essex

August 2022

Table of Contents

Acknowledgment

First and foremost, I would like to thank the universe for bringing me into existence and enabling me to provide this thesis. In addition, I would like to thank Dr. Adrian Clark for the invaluable guidance he provided to ensure the completion of this thesis. The completion of this thesis would not have been possible without him.

Furthermore, I would like to thank the members of Povray's forum for their guidance and insights, without which this project would not have progressed. Last but not least, I would like to thank my parents and the University of Essex for providing me with this opportunity and supporting me throughout my master's program.

Abstract

A growing number of smartphones and other devices equipped with Augmented Reality capability are becoming more accessible around the world, making augmented reality one of the biggest technology trends right now. An essential component of developing a Realistic augmented system is ensuring Computer generated object and the virtual cameras used to generate it match those of the actual camera. This thesis proposes a novel method for matching computer-generated objects with objects in the real world(Augmented Reality). The proposed method involves taking a series of images with a real camera that contains a checkerboard pattern. By using the checkerboard pattern as a reference and a computer vision library called OpenCV, this thesis proposes the calculation of the intrinsic and extrinsic parameters of a camera. Furthermore, this thesis proposes using this parameter to define a virtual camera and the geometry of the computer-generated object in pov ray. Based on the results of the experiments, it has been demonstrated that it is possible to integrate computer-generated objects with the natural world by using the physical properties of a camera in conjunction with computer graphics software such as pov ray.

1.1 INTRODUCTION

It has been documented throughout history that humans have made attempts to modify and improve their interactions with the natural world. Stone gathering and manufacturing weapons were early examples of human activity that altered and improved the physical world. These improvements and alterations were, however, limited to real-world objects. The invention of the Computer has, however, not only made it possible to improve and alter our environment, but it has also allowed us to integrate objects into the physical world enabling a more immersive experience. This sort of improvement is known as augmenting Reality or Augmented reality. Augmented reality is a subset of what is known as immersive technologies. These technologies, including virtual, augmented, and mixed reality enhances the user's experience by immersing them in a digital environment. For example, in virtual reality, a user is immersed in a completely different environment, whereas augmented reality displays computer-generated data in conjunction with real-life objects. In the figure below, one may see an example of what an Augmented Environment entails. Inside a room are two synthetic chairs and a lamp placed on top of a real table. As one looks closely, one will notice that the artificial objects are composed in such a manner that they appear to be a part of the physical environment.

Figure 1: Displays a real desk super composed of synthetic chairs and a lamp adapted from [1].

Another example is showcased by ESPN in 1998 when they used a yellow line on its screen called the first and ten lines to demonstrate how far an offensive team needed to move for a first down. The goal was achieved by combining data from cameras that observe the field during play with a 3D mathematical model of the field. Figure 2 shows what the "first and then line" looked like when ESPN showcased it in 1998.

Figure 2: "The First and ten-line." Adapted from [2]

In developing augmented reality systems such as the ones described above, it is essential to ensure that the virtual imagery and the virtual cameras used to generate it match those of the actual camera. This feature is essential for creating a realistic-looking augmented reality system. in theory, it is possible to achieve this match by combining computer vision techniques with computer graphics. The purpose of this thesis is to investigate this exact scenario and to determine whether it is feasible. This paper presents an approach to computing the physical properties of a camera used to capture a scene and simulating these properties with a virtual camera found within a rendering program called POV ray. Furthermore, this paper proposes the generation of synthetic objects and their matching with corresponding objects in the camera-captured scene. In Section 2, previous related works are reviewed in order to provide a perspective. Section 3 provides a brief background to the study. Additionally, a description of the experimental procedure and method used to experiment is provided. The fourth section presents the results obtained and discusses the methodology used. Finally, section 5 concludes with a brief conclusion and describes future research

2.0 REVIEW OF RELEVANT WORK

A great deal of research has been conducted over the last few decades to create realistic augmented reality systems that match virtual imagery with the natural world with a high degree of accuracy. It was proposed in 1999 by researchers at the Balearic island university of Spain that by obtaining important information such as a camera's location and orientation and combining this information with computer graphics applications such as Open GL it is possible to match a computer-generated object with that of the real world. According to their methodology, a camera was first calibrated to determine its physical properties. Furthermore, they developed a 3D generated scene and screen using Open Inventor, a graphic library based on OpenGL. Secondly, a virtual camera with the same characteristics as the calibrated real camera was developed and a series of images were mapped into a "3D generated screen" that

was in the virtual camera's field of view[2]. Afterward, the 3D generated scenes were placed between the virtual camera and the 3D-generated screen. The result was a novel system that gave an illusion of a virtual imagery object occupying a space in a natural scene.

Concurrently researchers at kangwon national university Korea developed a "snowman" character that moved around a dancer in a series of videos captured by multiple cameras[3]. It was observed that the snowman graphics were synthesized as if they were an integral part of the video scene recorded[3]. this was achieved by using multiple cameras: two of which were base cameras that were responsible for video capture, the other a virtual camera that generated new views between the base cameras, and the third, a graphic camera that generated the snowman[3]. All cameras were synthesized by computing their exact positions and orientations [3]. They also ensured they shared the same 3D world coordinate system[3]. By doing so, they could accurately place the snowman between the video scenes.

Furthermore, a team of scientists at Carnegie Mellon University developed a method in 1995 that allowed an actual human to walk in a computer-generated environment. The proposed method called z-keying takes four image inputs: an actual image, its depths map, a synthetic image, and its corresponding depth map, and compares each pixel located in them[4]. This separates each pixel into foreground and background. The one close to the camera is chosen as the foreground. This allowed for the integration of a natural and virtual scene.

Another research proposed matching virtual models with the natural world where a robot resides in other to cut pieces of bones[5]. The prosed method placed some reference markers(reference system) on the prosed bones and generated a 3d model of them. A match between the 3d models and the robot workspace was possible because the reference marker acted as a bridge between the virtual (3D model) and the natural world[5].

Researchers at the university of north Carolina developed a virtual environment ultrasound system that enables real-time augmentation of 3D echography data on a pregnant subject[6]. The method applied this by obtaining geometric information (positions and orientation) of sequences of 2D echography data and a camera connected to a head-worn display in a given 3d space with 6 degrees of freedom[6]. The geometric information is then used to transform the echography concerning the observer's position[6].

The literature review does not express explicitly, however, that one common theme among these methods is a comprehensive analysis of how pixels in a 2D image are related to pixels in a 3D world or having a robust camera model to register the virtual object accurately. Although this thesis illustrates the relationship between pixels in an image and the 3D world in a similar manner to some previously mentioned methods, this paper offers detailed information on how the information can be combined with a rendering software called pov ray to merge computer-generated objects with real-world images.

2.2 The Goal of this thesis

In this thesis, the primary goal is to simulate the properties of a real camera and super-compose two parallel lines generated by computer graphics with their corresponding lines located in a scene captured by a camera. Figure 2.1 illustrates what this thesis aims to accomplish. An image of a scene has been captured using a camera at the University of Essex. by Utilizing computer vision techniques known as camera calibration, this thesis intends to compute the physical properties of the camera and mimic them with a virtual camera within a rendering software known as POV-Ray. Afterward, this thesis intends to generate two pairs of parallel lines and match them with corresponding lines located in the real-life scene captured by the camera.

Once the above goal has been achieved, what should be seen is an augmented object created with computer graphics and superimposed over the actual scene captured by the camera.

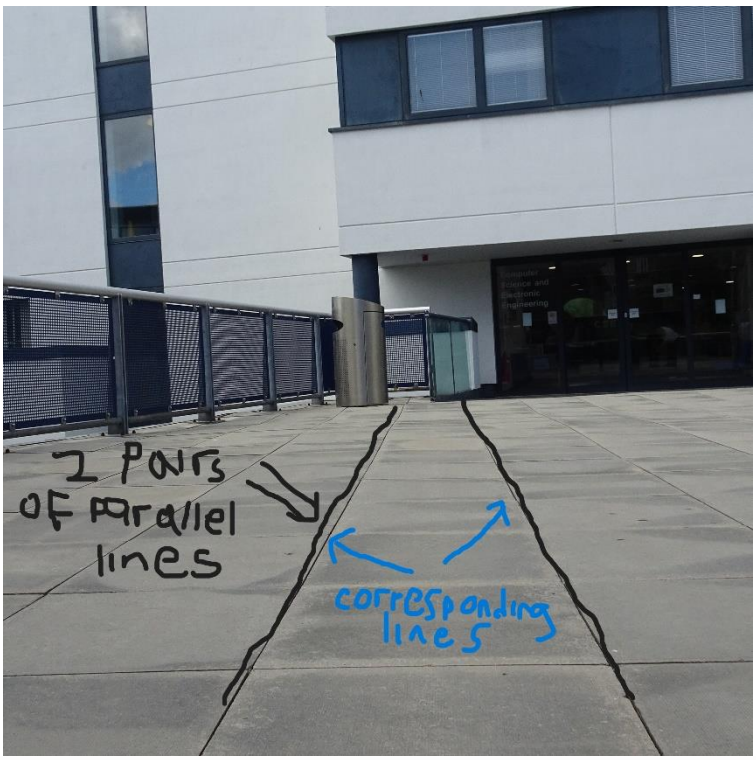


figure 2: Demonstrates the thesis' primary objective of simulating the properties of a real camera and composing synthetic parallel lines with the real ones thereby producing a realistic augmented reality system

2.3 Proposed experimental procedure

This thesis has set out some experimental procedures with the aim of achieving the primary objective of the thesis. This section explains a brief schema of the procedures needed to complete the proposed Goal. These steps are discussed extensively in Chapter 3. the steps consist of

1. Printing a nine-by-six chessboard calibration target and measuring at least one square on it.
2. Setting up a camera horizontally at the height of 5mm and taking a sequence of images with a chessboard calibration target.

3. Capturing images of square 1 at the University of Essex
4. Computing the physical properties of the proposed camera, which includes the intrinsic and extrinsic parameters using a computer vision tool called on OpenCV
5. Mimic real-world properties of the camera in a computer graphics software called PovRay and render a scene with two pairs of parallel lines placed the same distance apart as that of the paving slabs in square 1.
6. Superimpose the parallel lines and check if they match. If they don't, camera refinement will be done until the best match is found.

3.0 Methodology

In this chapter, we will discuss the perspective camera or pinhole camera model, the type of camera used to capture a scene of square one at the University of Essex. A perspective or pinhole camera is a mathematical model that establishes geometric relationships between points in the 3D world and their equivalent points in a 2D image[7]. We also discuss the digital perspective camera, which is a perspective camera with lenses used by many modern cameras, including the one used to capture images of square one. Furthermore, we discuss how the pinhole camera uses a projective transformation to map points located in a 3D world to points located in a 2D image. Additionally, we discuss how a homogenous coordinate system can be used to better represent projective transformation. In addition, we discuss how the physical properties of a pinhole camera, the intrinsic and extrinsic parameters, are encompassed in a homogeneous coordinate system and provide a complete projection from a 3D world to a 2D image. These parameters indicate the internal properties and position/orientation of the camera. Next, we examine how camera calibration can be used to estimate these parameters. After that, we describe how these parameters were estimated through an experimental calibration procedure. Furthermore, we discuss how these parameters are used in POV-ray to merge

computer-generated content with the real world. Our final contribution is to describe the experimental procedure used in POV-ray to integrate computer graphics with the real world.

3.1 The Perspective Camera

Over the past decades, the camera has been one of the essential tools used in many research areas, such as computer vision. It is a device that helps us map the world around us and produce an output called an image[14]. It is vital to understand how a camera is modelled to use in various applications, including augmenting computer-generated objects with the real world.

2 Pinhole cameras

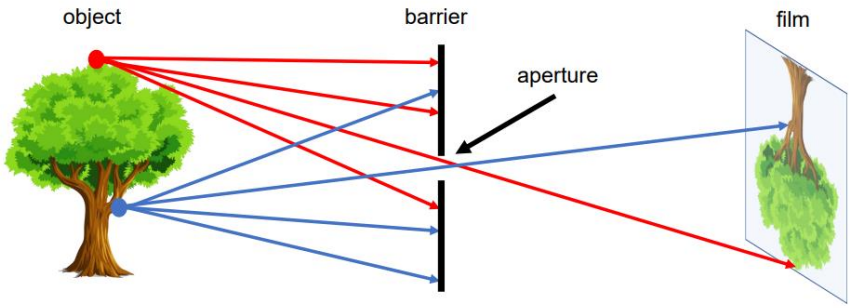


Figure 3: shows a simple pinhole camera adapted from[14]

The perspective camera or the pinhole camera model defines the geometric relationship between a point or object located in a 3D world and its equivalent point in a 2D film[7]. The figure above illustrates a simplified perspective (or pinhole) model in which the 3D object is separated from the 2D film by a barrier and small pinhole or aperture. The Pinhole and barrier serve as a pathway for the transmission of some light from 3D objects to the film. Theoretically, without them, each point in the film would be affected by light rays from the real world[14]. Since only some light rays pass through the Pinhole, it is possible to establish

a direct correspondence between the real-world object and the film[14]. As a result, what is seen is an image mapped onto a film.

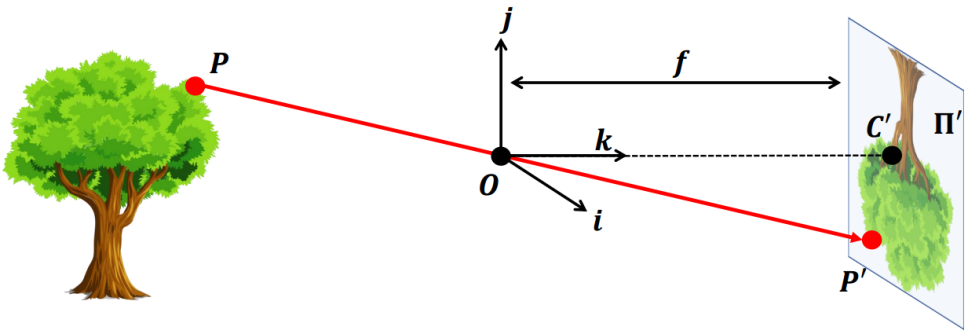


Figure 4: shows the mathematical model of a pinhole camera adapted from[14]

In Figure 4, we can observe a mathematical representation of a simple pinhole camera, characterized by a ray of light passing through the point **o** and onto the image plane 𝕀𝕀'. the point **o** where the ray of light passes through is called the optical center. The focal length can be defined as the distance between the optical center and the image plane. j, k, i refers to the camera coordinate system placed at the optical center so that the k axis is perpendicular to the image plane[14]. The optical axis is the line between C' and the optical center. Suppose that we have a point in 3D space p[x, y, z], the perspective camera projects this point into the image plane at the point P'[u,v], using a law called the law of similar triangles. Per [14], since P', C,' and **o** form a triangle that is similar to P, O,(0,0,z), the equation that projects these points can be written as follows:

$$P' = f[u/z] \;\; f[v/z] \qquad (1)$$

With this projective equation, it is possible to determine the relationship between the 3D point P and the image point P', thus understanding how images are mapped using pinhole cameras.

3.2 The Digital projective camera

To formulate a Morden pinhole camera, it is essential to consider the size of the pinhole. It is of utmost importance as it impacts the overall quality of the final image that is projected onto the image plane. By increasing the pinhole size, more light rays can reach the image plane from 3D space. Thus, the image appears blurred. In contrast, if the pinhole size is reduced, the image will become darker as little light passes through the hole. There is a good chance that this will be a significant problem in most real-life scenarios.

For this reason, a digital perspective camera with lenses was developed to capture images with varying brightness and darkness. By replacing the pinhole with a lens that bends and focuses light, you can project all light rays to the same P' [14]. However, projection to a single point is not valid for all 3D points[14]. In figure 5, for instance, the top of the object is projected to one point on the image plane, while the other point is scattered slightly. Furthermore, the camera lens also exhibits the property of focusing and blurring some points too close or too far from the image plane[14]. This can be seen when some parts of an image appear clear, and others don't.

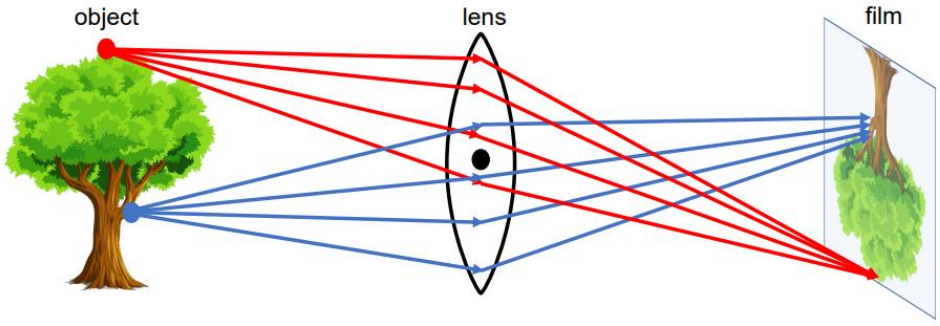


Figure 5 shows how lenses affect the image of a pinhole camera. There is a noticeable difference between some points in the 3D space projecting into a single point on the image plane and others that do not. adapted from [14]

The digital pinhole also has a focal point and focal length. When the light rays parallel to the optical axis intersect at a single point, this is called the focal point [14]. The distance between the focal point and the lens is called the focal length. Figure 6 shows an example of what this looks like.

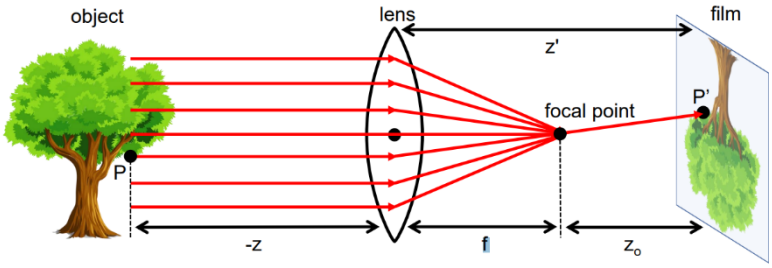


Figure 6: shows the focal point and the focal length of a camera

Observe that light rays passing through the lens do not deviate. This allows us to form a mathematical model that maps 3D locations into images.

$$P'[u,v] = [z'u/v] \quad (3)$$

Furthermore, digital pinhole cameras use a thin lens to approximate 3D points in space to 2D points in images, unlike pinhole cameras. As a result, images are captured with some issues. One of the most prevalent of these issues is radical distortion. In this case, straight lines in a picture appear curved due to the difference in focal lengths of different parts of the lens. Figure 7 shows different types of radical distortion.

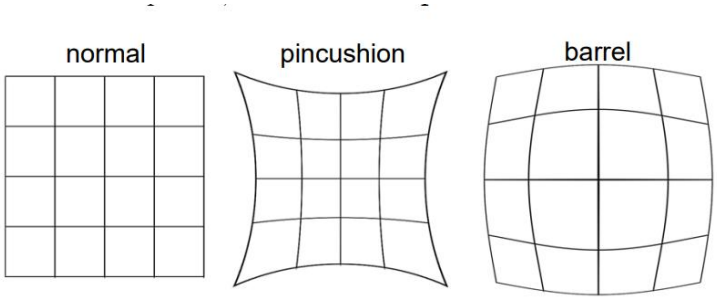


Figure 7: shows the different types of distortion and their effect on an image. Adapted from[14]

3.3 How perspective projection from 3D image to the image plane works

In the previous section, we discussed how a projective camera projects 3D points in space into 2D points positioned within the image. The mapping from a 3D point P[x, y, z] to a 2D point P'[u,v] is known as a projective transformation. Further discussion was conducted regarding the mathematical equation that maps a point P[x, y, z] to a point P'[u,v,] (see equation 1). However, due to several factors, the discussed equation does not give a complete mapping from 3D space to 2D space. First, the spatial coordinates of points in the image plane differ from those of digital images. For instance, the center of an image in the image plane is located at C', where the k axis meets the image plane, while the center of a digital image is located at the lower left corner. As a means of offsetting this different point, a translation vector [px, py] is used. These vectors are called the principal point. Thus, using these principal points, the new mathematical equation that maps or projects 3D points P[x, y, z] to image point P'[u,v] is

$$P'[u,v] = f[u/z + px]\ f[v/z + py] \quad (4)$$

Secondly, points on an image plane differ from points on a digital image. For example, in a digital image, points are represented by pixels, while in an image plane, points are represented by centimeters [14]. For these differences to be accounted for, new variables will need to be

added. Assuming the new variables are x and y, the new projection between 3D point P[x,y,z] and a point located in the image P'[u,v] will be described as follows:

$$P'[u,v] = fx[u/z + px]\ fy[v/z + py]$$

fx and fy are the camera's focal lengths in the x and y axes, respectively.

3.4 Perspective projection from 3D image to the image plane using a homogenous coordinate system

It is possible to represent a projection from a point in 3D space to a point in an image plane more effectively. An effective way of representing projection is by using a homogeneous coordinate system. It works by appending one to each points in an image plane and 3D space. As an example, suppose we have a point P'[u,v] and P[x,y,z]; using a homogenous coordinate system, P'[u,v] will be equivalent to P'[u, v,1] and P[x, y, z] will be equal to P[x, y,z,1]. [14] suggests that representing this projection as a homogenous coordinate system will facilitate future calculations. As a consequence, by introducing a homogenous coordinate system, we can rewrite the relationship between the two points P'[u,v] and P[x,y,z] as

$$P'[u, v, 1] = \begin{matrix} fx & 0 & 1 \\ 0 & fy & 0 \\ px & py & 1 \end{matrix} P[X, Y , Z,1] \quad (5)$$

In the formula, fx, and fy are the focal lengths corresponding to the x and y axes, and px and py are the principal points located at the x and y axes, respectively. This equation can be rewritten further into a simple equation

$$P'[u,v,1] = I\ P'[x, y, z,1] \quad (6)$$

It is here that I is referred to as the intrinsic parameters or the matrix of the camera. This parameter, which includes the focal length and the principal point, is imperative if one were to characterize a camera. It is important to note that other intrinsic parameters, such as skew and

distortion, are not included in this mathematical equation since they are outside the scope of the study.

For the project to be completed from points in 3D space to points in the 2D image, other parameters must be considered in addition to intrinsic parameters. This type of parameter is referred to as an extrinsic parameter. It is these parameters that provide helpful information about the camera's position in 3D space as well as its orientation.

A 3x3 rotation matrix R and 3x1 vector t define the position and orientation of the camera mathematically [7]. A model of the extrinsic parameters is shown in Figure 8. One must first translate the camera to the world coordinate origin before rotating it to locate the point P [7].

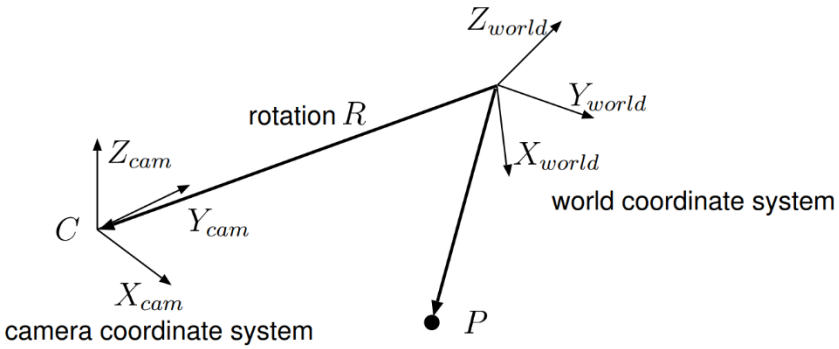


Figure 8: shows the relationship between a camera and the 3D space or world coordinate system. This relationship is characterized by a camera center, rotational vector R and a translation vector. adapted from [7]

Using mathematics, we can express rotation and translation as follows:

$$R \quad t \quad = \quad \begin{matrix} r1 & r2 & r3 & t1 \\ r4 & r5 & r6 & t2 \\ r7 & r8 & r9 & t3 \end{matrix} \tag{7}$$

Through the combination of intrinsic parameters and extrinsic parameters, we can derive a mathematical equation that fully projects points in 3D space onto an 2D image as

$$p'[u, v, 1] = \begin{matrix} fx & 0 & 1 & r1 & r2 & r3 & t1 \\ 0 & fy & 0 & r4 & r5 & r6 & t2 \\ px & py & 1 & r7 & r8 & r9 & t3 \end{matrix} P'(X, Y, Z, 1) \quad (8)$$

This complex formula can be reformulated into a simpler equation

$$P'[u, v, 1] = I\,[r\ t]\ P[X, Y, Z, 1] \quad (9)$$

It is evident from the above projection equation that it is divided into two groups: the intrinsic parameters which is describing a camera's internal characteristics and extrinsic parameters describing a camera's relationship with the 3D world through rotation and translation. We can shorten the mathematical equation by rewriting the equation with a product matrix of M. Upon completion, the final project from 3D space into 2D image is written as follows:

$$P'[u,v, 1] = M\ P'[x, y, z, 1] \quad (10)$$

3.5 Estimating the intrinsic and extrinsic parameters with Camera calibration

Camera calibration is the process of estimating the intrinsic and extrinsic parameters of a camera. Over the years, many methods have been proposed to tackle this problem because it's a vital part of computer vision applications. One of the most popular of these methods is the

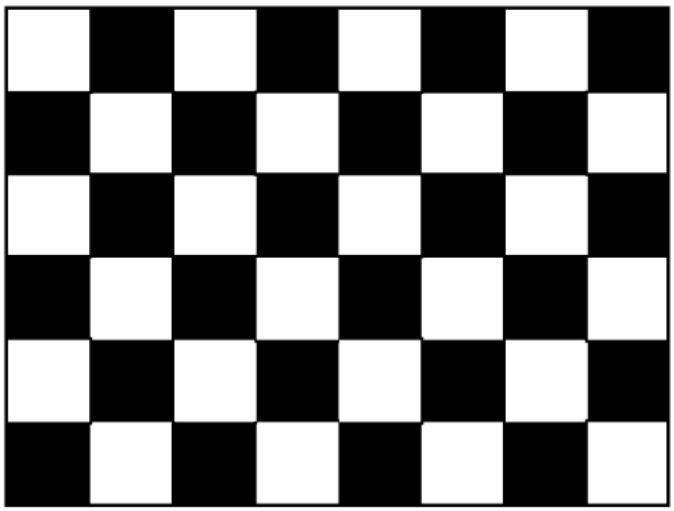


Figure 9: An image of a calibration pattern with a square topological structure from [9]

checkerboard calibration method proposed by Zhengyou Zhang. The technique requires a camera to capture a checkerboard pattern from multiple perspectives(see figure 9). The checkboard can be printed and pasted on a flat surface like a wooden board or book. The proposed method is quite robust, inexpensive, and easy to use. Because of this, it allows anyone with little to no experience to perform vision-related tasks tools.

By utilizing the topological structure of the checkboard, the proposed calibration method first uses a corner detector algorithm to extract "feature points," which relate the 2D points in an image plane to the 3D points on the checkerboard[7]. Theoretically, the Algorithm assumes the z-direction of the checkerboard is perpendicular to the plane of the checkerboard. This indicates that the 3D feature point belongs to the board plane z = 0(see figure 10)[7].

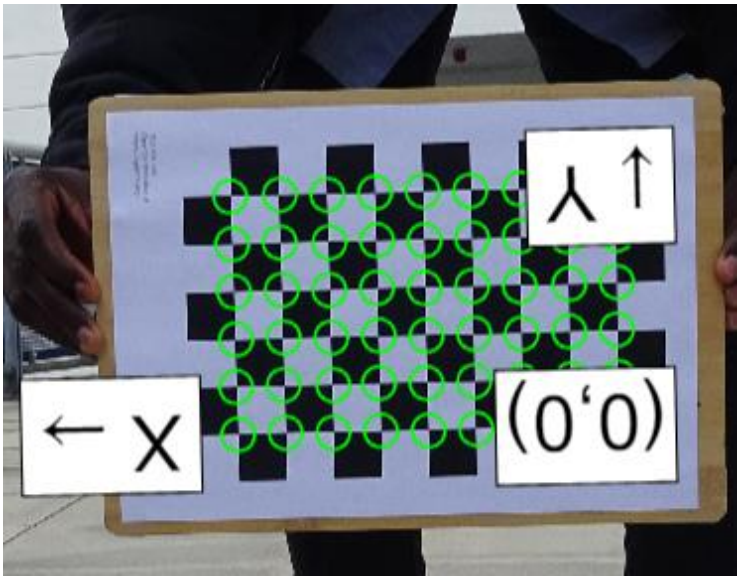


Figure 10: shows the 3D coordinate system of the checkboard where the z-axis is at the origin (0,0).

With regard to the above assumption, we can write the projection of 2D points on the checkboard onto the image plane as follows:

$$P'[u, v, 1] = \begin{matrix} fx & 0 & 1 & r1 & r2 & r3 & t1 \\ 0 & fy & 0 & r4 & r5 & r6 & t2 \\ px & py & 1 & r7 & r8 & r9 & t3 \end{matrix} P[X, Y, Z = 0,1] \quad (11)$$

With this assumption, the Z axis and the last column of the rotation vector can be discarded, allowing us to reformulate the above formula as follows:

$$P'[u, v, 1] = \begin{matrix} fx & 0 & 1 & r1 & r2 & t1 \\ 0 & fy & 0 & r4 & r5 & t2 \\ px & py & 1 & r7 & r8 & t3 \end{matrix} P[X, Y, 1] \quad (12)$$

To simplify, we can write the equation as:

$$P'[u, v, 1] = I\,[r1\ r2\ t]\ P[X, Y, 1] \quad (13)$$

[7] proposes that I [r1 r2 t] represents a 3 * 3 rotational metric product, which is known as a planar homography transform. The term planar homography refers to a homography that specifies the linear mapping between two planes[7]. In our case, the two planes are the checkboard and the image plane of the camera. The method calculates the homography transform, which will be donated as **H**, between points in the checkboard P(X, Y, 1) and those of the image plane P'(u,v,1). We can donate this with the following equation

$$P[u, v, 1) = \mathbf{H}\,(X, Y, 1) \quad (14)$$

The calibration method then uses the homography transform to estimate the intrinsic and extrinsic parameters.

3.6 summary of the calibration method

1. capture multiple images from different perspectives and use a corner detector to detect feature points

2. for each image, compute the **H** homography transform between feature points in the checkerboard with that of the image plane

3. use the homography transform to compute the intrinsic and extrinsic parameters.

3.7 Image Capturing

During the experiment, a tripod-mounted camera was placed on square 1, which is next to the network building at the University of Essex. A series of images were taken using the camera after measuring its height, which was about 5 mm. The images taken can be separated into two classes. The first class was of a checkboard pattern which was used to calibrate the camera, while the second was pictures of square 1, which acts like the real world where we want to super compose a virtual object. Figure 11 shows an example of the images taken during the image capture. About 55 images were taken with the checkerboard from different perspectives. While nearly 3 images of square 1 were taken consecutively.

Figure 11: shows an example of the checkerboard pattern target(left) and an image of square 1(right) that was taken with a camera placed at the height of 5mm.

3.8 Camera Calibration using OpenCV

Open Cv, also known as an open source computer vision library, is a powerful computer vision library that offers powerful algorithms for performing various computer vision tasks such as detecting faces, identifying objects, classifying human actions, calibrating cameras, tracking cameras, and tracking objects, and establishing markers for augmented reality[12]. Additionally, OpenCV provides an algorithm that allows users to calibrate cameras to determine intrinsic and extrinsic parameters. The algorithm first takes the size of the corners located in the width and height of a checkerboard in an image. You can obtain this information by counting the inner corners of each square in the width and height directions. In our case, the size of the squares on the checkerboard was 9 in width and 6 in height. Secondly, a termination criterion finds sub pixels that give the exact location of the chessboard corners(see figure 12). The algorithm then stores the pixel's corner points in the 3D world and their corresponding points in the 2D image plane. Thirdly a for loop is run through all the images, which are first converted into grey scale.

Afterward, a *findchessboardconners* function is passed to find chessboard corners in the images. For our case, the algorithm tries to find the 9 by 6 chessboard corners we specified at the beginning(see figure 12). If the chessboard is located, the algorithm then draws and displays the corners(see figure 12). The algorithms uses the information from finding the concerns and pass them through a *calibratecamera* function. This function finds the camera matrix, distortion coefficient, rotational and translation vectors for each image. Lastly, the algorithm calculates the reprojection error, which is the difference between the detected points from the calibration image and the reprojected points using the estimated camera parameters. Ideally, lower reprojection error is better as it produces a more accurate result.

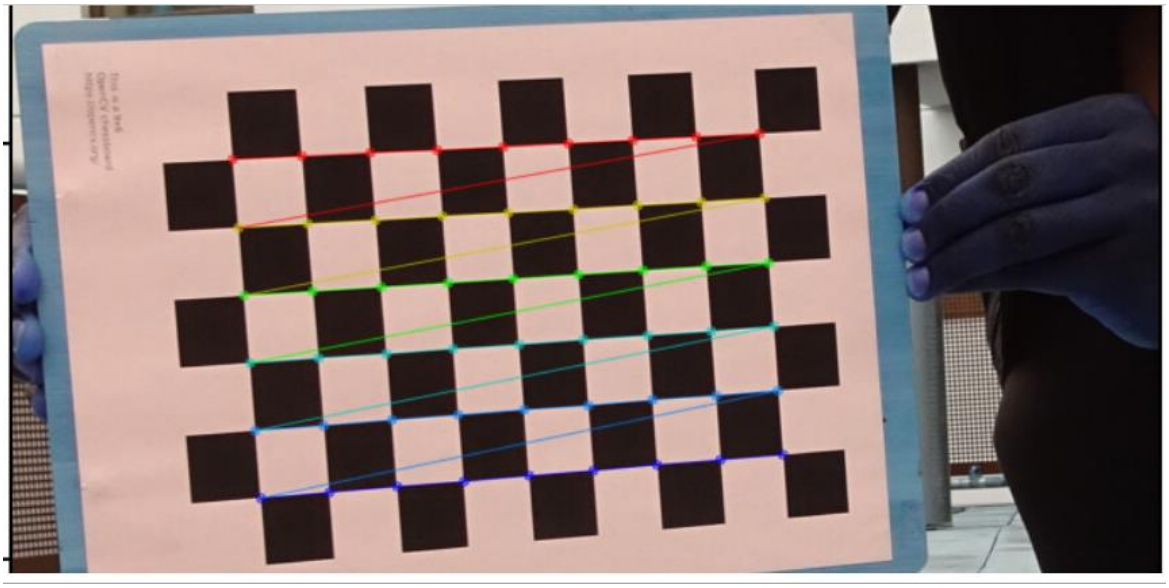


Figure 12 shows the detected feature point, which will be used to estimate our camera's intrinsic and extrinsic parameters.

3.9 Using the estimated parameters

In this section, we provide how the estimated parameters are used in POV-ray and how they crucial in the matching of computer generated object with the real world. As stated, this thesis aims to match a virtual camera with graphical content with a real camera used to capture a scene(square 1). To do this, we needed to understand how the real camera worked and what properties it had. This guided us to discuss the perspective camera and its projective properties(intrinsic and extrinsic parameters). It is the first parameter that will aid in defining the virtual camera in pov ray. It is the first parameter that will aid in defining the virtual camera in pov ray. In contrast, extrinsic parameters will dictate the geometry of the virtual object, which will be blended with the natural environment. However, these parameters wouldn’t give us a complete integration, which is why some experimental refining will have to be undergone in order to get the full integration.

3.10 PovRay

The persistence of vision Raytracer, or PovRay, creates high-quality scenes using text-based editors and ray tracing procedures. Ray tracing is a rendering technique that simulates how light behaves in computer graphics. For Pov Ray to produce high-quality scenes, it requires a file that contains information about the objects, lighting, and camera in the scene[11].

3.11. POV Co-ordinate system

Prior to defining our virtual camera and the computer-generated object, we must understand how povRay's 3D coordinate system operates. This coordinate system functions as a navigational aid in the Pov Ray universe. PovRay uses what is called a left-hand coordinate system. This system is illustrated by looking at a computer screen, in which the x-axis is moving along the bottom, the y-axis is moving up the left edge, and the z-axis is pointing into the screen [11]. Figure 13 shows an example of what this looks like.

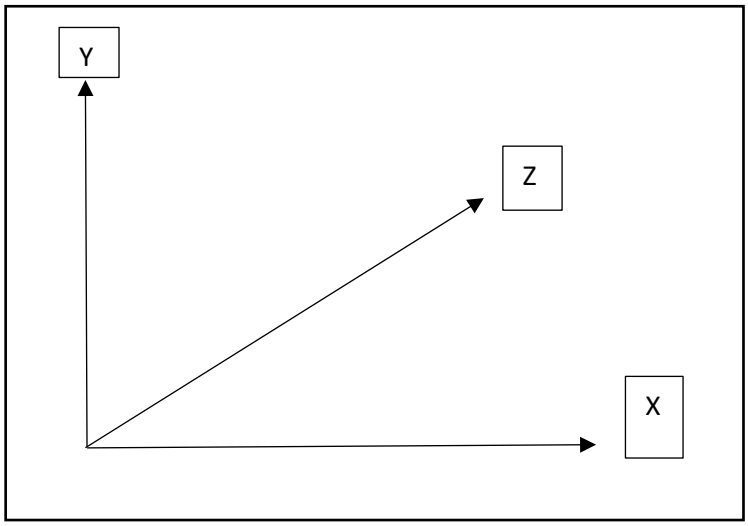


Figure 13: shows the left-hand co-ordinate system used by pov ray

To give another example, hold your left hand in the form of a fist. The first step is to extend your thumb to the right. Finally, point your middle finger straight ahead, then point your index finger straight up. Your thumb facing to the right connotes the positive x directions.

Your index finger facing straight up connotes the positive y direction. Finally, your middle finger points straight ahead connoting the positive z direction.

3.12 Importing Captured image to pov ray

As per the goal, it was necessary to import the image taken with the camera into the pov ray universe in order to augment two parallel lines. in Pov Ray, an image map statement allows users to import images or pigments into the Pov Ray universe. A simple way to describe Image Map is that it allows users to wrap 2D images around 3D objects. For instance, it is possible to generate a 3D box using an image as a pigment. This is how the image of square 1 was imported into POV Ray using the above example. In Figure 14, you can see the code that was used to accomplish this task:

```
box{<0,0,0>,<1,1,1>

pigment  {image_map  {"C:\Users\eobay\Downloads\square-1-pix\2022-06-08-
003.jpeg" once

map_type 0 } }

finish {ambient 1 diffuse 0}

scale <Image_W, Image_H, 1>

rotate<90,0,0> // Rotate into the x,z plane
```

figure 14: shows the image map statement for importing an image to pov ray. adapted from [15]

Pov Ray allows the definition of a simple 3D box using two corners. In pov ray coordinate universe, each corner is represented by the following vectors: x, y, and z. It is generally recommended to ensure that the vectors for each connector are opposite. In other words, one corner should contain the minimum value of x,y, and z, and the other should contain the maximum value of x,y and z. According to our design, we ensured that the box or minimum value originated at the lower left corner of the screen, which was then drawn upwards to the

upper right corner of the screen, at the coordinates 1,1,1. Figure 15 gives an example of what this looks like.

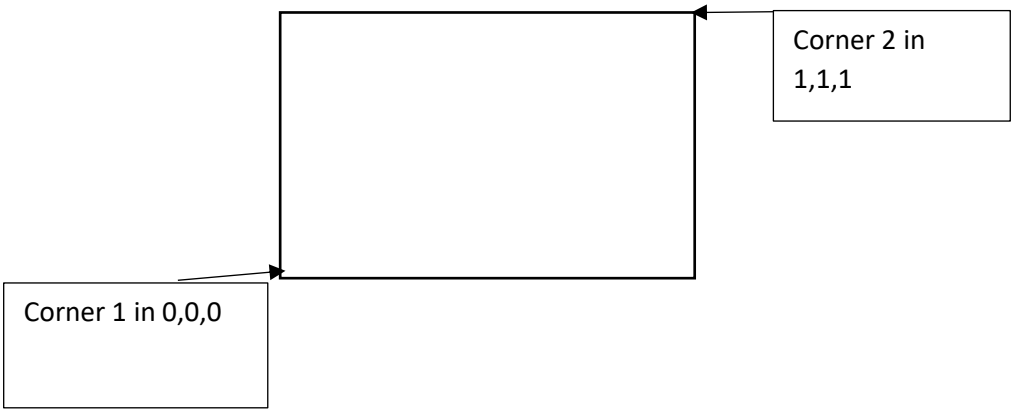


Figure 15: box statement for boxes in pov ray universe. Drawn from the left lower conner at <0,0,0> to the upper right corner at <1,1,1>.adapted from[15].

The pigment {image map} statement tells pov to use the image located at "C:\Users\eobay\Downloads\square-1-pix\2022-06-08-003.jpeg" as a pigment for the 3D box and Render it not more than once. Using the "once" method was necessary because pov ray renders more than one copy of the same image on the 3D box if it is not done. The map type 0 tells pov ray its mapping unto a planar surface. The finish {ambient 1 and 0} instructs pov ray to remove shadows from the imported image. In the absence of this method, the image appears darker than it actually is. By using the scale command, pov ray will resize the image and box to a predefined width and height. With the rotate (90,0,0) command, the image is rotated into the x and z planes of the pov ray universe. This method was used because all image map pigments in the pov ray universe are by default placed in the x and y plane. This causes the image to be unviewable as it is in the wrong plane. By rotating the x-axis, the image becomes visible in the x and z planes.

Initially, one of the challenges encountered when attempting to use the above method to import the proposed image into the pov ray universe was that the aspect ratio was not quite

appropriate. The image was rendered by PovRay, but it was quite small and did not fit on the screen. To resolve this issue, height and width variables were declared with the code above in order to inform pov ray of the output aspect ratio. Figure 16 shows the command that was used to define a suitable aspect ratio:

```
#declare Image_W = 2752;

#declare Image_H = 2752;
```

Figure 16: predefined width and hight. Adapted from[15]

One major drawback of using the proposed method is that changing these variables will change the aspect ratio of the image thereby affecting the registration of any augmented object.

3.13 Defining a camera in Pov ray

A camera statement in POV ray allows users to specify what type of camera they are using, where the camera should be placed, and how the camera is to view a specified scene. This statement requires multiple parameters: the type of camera to be used, and three numeric vectors x, y, and z separated by commas in angle brackets, which describe where the camera is located and where it is pointing. Figure 17 shows the syntax used to define a camera in pov ray

```
Camera{

        Location<0,4,-2>

        Look at <0, 2,1>

      }
```

Figure 17: the building blocks of a virtual camera. This statement tells POV-ray where the camera is located and where it looking at. Adapted from[15]

A camera location is a vector statement that indicates where the camera is situated within the coordinate system of the pov ray. Using the above example, POW Ray will place the camera 4 units up in the y-axis and two units back from the z-axis or center of the scene, defined as 0,0,0 by default. The + and - in the z-axis instruct pov ray to move the camera into the scene or out of it respectively[11]. The look-at statement tells pov ray to where the camera should look at. In most cases, this should be the center of the image or scene being worked on.

The same statement above was used to define the location and look at command of the camera that was used to augment the two parallel lines; however, some modifications were made. Figure 18 shows the command statement that was used to set the type of camera, its location, and look-at function.

```
camera {
perspective
location  <Image_W/2, 10000, Image_H/2,>
look_at   <Image_W/2, 0, Image_H/2>
```

Figure 18: defining the type of camera, where its located, and where it's looking at. Adapted from[15]

With the perspective command, we tell pov ray what kind of projection and properties we desire. Many cameras can be used in pov rays, each with distinct projection characteristics. Perspective cameras attempt to simulate the physical properties of pinhole cameras (see 3.1). Since the actual camera that captured images of square 1 emulates the pinhole cameras, the virtual camera must exhibit similar characteristics. The perspective syntax is used to specify the type of projection used by a pinhole camera. After defining the type of projection, the virtual camera was ready to be placed. As mentioned above, pov-ray's location syntax consists

of three vector statements x, y, and z, which determine where the camera is located. There was a predefined width and height of 2752 assigned to the x-axis and the z-axis respectively. It instructs pov ray to place the camera 2752 unit along the x and z axes. In contrast, a vector number of 10000 was assigned on the y axis. It instructs POV-ray to place the camera 10000 units up on the y axis. Two-fold division was done on the width and height in the location and look at the statement to instruct Pov Ray that the camera should be cantered in the image. This was necessary to keep the camera from being positioned at the top right corner of the screen.

For the look-at statement, the x and z axis were given a predefined width and height of 2752, respectively. This instructs pov ray to take into account the specified width and height. For the y axis however, a vector value of 0 was specified. It instructs POV ray to focus on the center of the image.

In the next parameter, we defined the right vector for the virtual camera. The purpose of this command is to inform POV ray what the final aspect ratio of the image will be. Figure 19 shows the command that was used to achieve this objective:

```
right     x*Image_W/2752   // aspect
```

Figure 19: defining the final aspect ratio of the image using the right command. adapted from [15]

3.14 Defining the focal length and principal point

In this section, we will describe how the intrinsic parameters, namely focal length and principal point, were defined in pov ray. Remember that in 3.4, we defined *intrinsic parameters* as those within a camera. To ensure that the virtual camera is as accurate as the real one, it must also have the same focal length and principal point. POV-Ray does not explicitly provide primitive syntax to define this parameters. however, it is possible to describe these parameters directly through other means.

3.14.1 replacing direction vector with the focal length

According to Pov Ray, the direction vector describes the distance from the camera and the image is being viewed. in some sense, it is assumed to be the focal length. This thesis made the assumption because the focal length and direction vector of pov ray share a common theme. They both describe the distance between the camera center and the image plane. To describe the direction vector a direction statement was used followed by 3 vectors in angle brackets which corresponds to the x, y, and z-axis. Figure 20 shows what this looks like:

```
direction<2.87590333,2.79384814 ,0>
```

figure 20: defining the direction vector in pov ray. it is assumed that the direction vector is the same as the focal length because it describes the distance between the camera and the image plane. adapted from[15]

The vectors that were used in the case were the vector gotten after using OpenCV to calibrate the real camera. Where `2.87590333,2.79384814` are the focal point in the x and y axis respectively.

3.14.2 replacing the focal point with the principal point

The focal point in POV ray is simply a point on the image plane where the camera should focus more attention. In a pinhole camera, the principal point is the projective center or the point of focus where light passes through, so it is assumed that the focal point in pov ray has similar characteristics. Pov ray allows us to define this point using a focal point syntax which takes 3 vector values in the x, y, and z-axis Just like the location and look-at statement. Figure 21 shows the command that was used to accomplish this :

```
focal_point < 5.37471722, 1.20777920 ,0>
```

Figure 21: defining the focal point in the x and y axis using the vectors gotten from Open Cv. adapted from [15]

The x and y axis vectors were adapted from the calibrated parameters done by open cv. A major drawback of the above method was that the image appeared to be inverted. The following figure 22 illustrates what this would look like:

Figure 22: inverted image when using the above method

The main Cause of this couldn't be determined. However, pov ray provided a solution. By defining a second direction syntax, this time in the z-direction, the image view was corrected.

3.15 other Camera settings

Other camera settings were also defined to ensure the maximum configuration is reached. An aperture command was defined which tells pov ray the area of focus we are interested in. ideally having a smaller aperture increases the area of focus and having a larger aperture decreases the area of focus. in pov ray an aperture value of 3. 3.6171875 was assigned to the position of our focal point to adjust how far or close we want the area of focus. the aperture values were adapted from the properties of the images captured.

Another camera setting that was assigned was something called blur samples. Depending on the blur sample setting, each pixel can be sampled with multiple rays. Higher sampling rays result to higher quality output image but slower rendering time. Lastly an angle vector was defined to determine the direction and zoom of the camera as it views the captured image of square 1. Increasing the angle will move the camera away from the image thereby making impossible to see. Decreasing it, however, does the opposite and zooms into the image. In our case, a viewing angle of 15 was defined which puts the camera at a suitable zooming point thereby making the image clear and visible.

3.16 Developing Virtual parallel lines

After developing a virtual camera that is similar to the real one, two pairs of parallel lines were developed with the translation and rotation vector from the calibrated camera. They were developed in such a way that they appeared between the camera and the image imported into pov ray. figure 23 shows an example of what this look like

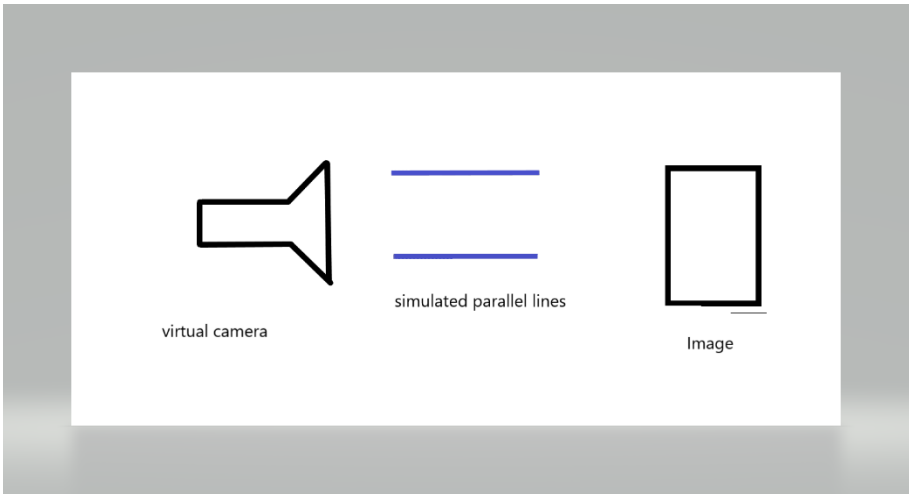


Figure 23: schema of the parallel lines developed in pov ray

The main purpose of developing the simulated lines is to match them with the real lines located in square 1. Unfortunately, they didn't which is why some experimental procedures were undertaken to ensure a match is found. Below details of how the augmented parallel lines were developed and matched with the real world are explained.

Pov ray allows users to place primitive shapes after a camera has been defined to view a proposed scene. These primitive shapes include 3D spheres,3D boxes, 3D cylinders, 3D cones, etc. as you may have noticed pov ray is a 3D rendering software and is not able to render 2D objects like parallel lines. However, there is an alternative way to fix this dilemma. In theory, it is possible to render a very long tin 3D box and scale it to a really small size which gives an illusion of a line. This could also been done with many primitive shapes such as 3D spheres and cylinders. However for this thesis we used a box instead of other primitive shape. The proposed hypothesis was tested and it worked as expected. Figure 24 shows the commands that were used to develop the augmented lines:

```
box{<0,0,0>,<1330,2,-7>        // Size
  finish {ambient 1 diffuse 0} // Make it visible
     rotate<0.70539152,-71.5,2.98700146>
     translate<1015,6.16673366,33.9743772>          // Move it about (the
y*1 puts it above //the image box)
}
box{<0,0,0>,<1330,2,-7>        // Size
     finish {ambient 1 diffuse 0} // Make it visible
     rotate<0.61271346,-108,0>
     translate<1950,2.71319379,29.76189959>  // Move it about (the y*1 puts
it above //the image box)
}
```

Figure 24: shows the commands that were used to develop the augmented lines

both boxes were drawn from the lower left of the screen <0,0,0> up into the screen at the co-ordinate <1330,2,-7>. a Finish ambient command was passed to ensure the boxes are visible for sight. The 3D boxes were then rotated and translated to a suitable position that matched those of the real world. As mentioned above it was vital to undergo some experimental procedures to ensure the parallel lines match that of the real world. The rotation and translation vector played a huge role to ensure this match was possible because they help in defining the geometry of the simulated object. Using the rotation and translation vector gotten from the real calibrated camera didn't really give a good geometric match. The simulated objects only appeared on the left side of the scene. To fix this the rotational y-axis vectors for

both boxes were changed to give us the correct rotation and the translation x-axis was changed to move the simulated lines to the correct position.

4.0 Results

In this section, we discuss the result gotten from the methods discussed above. It was discussed that the overall goal was to achieve the correct integration of the simulated parallel line with that of the real lines located in an image captured by a real camera. In order to render the scene, it was discussed that the virtual camera used to render the scene has to match the real one. This section presents the results of both the camera calibration and the correct integration of the computer-generated object with the real world.

We begin by providing the projection matrix for the camera. In contrast to the projection matrix provided in (5), Open CV provides a different format. In open CV, the projection matrix is expressed as follows:

$$P'[u, v, 1] = \begin{matrix} fx & 0 & px \\ 0 & fy & py \\ 0 & 0 & 1 \end{matrix} P[X, Y , Z,1] \quad (15)$$

The second parameter is the distortion parameter of the camera. In 3.3, distortion was discussed as occurring when straight lines appear curved. The distortion parameters can be used to fix this distortion, but this is outside the scope of study and will not be discussed here. Last but not least, the rotational and translational vectors are provided for each image.

Camera matrix:		
[[2.87590333e+03	0.00000000e+00	5.37471722e+02]
[0.00000000e+00	2.79384814e+03	1.20777920e+03]

[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion coefficient:
[[2.53631494e-01 -4.15638058e+00 -4.09496819e-03 -1.73704534e-02
1.62090749e+01]]

Rotation vector :
(array([[0.70539152], [-0.03214883], [2.98700146]]),
array([[-0.41210736], [-0.02263471], [3.00240164]]),
array([[-0.06106555], [0.03076827], [3.11999784]]),
array([[0.21912576], [0.17171452], [3.04537167]]),
array([[-0.45927613], [-0.53672869], [-3.0028573]]),
array([[0.58858482], [-0.36301724], [2.9428102]]),
array([[0.11654713], [0.17825752], [3.08681573]]),
array([[-0.15846253], [0.64771343], [2.9120293]]),
array([[-0.42066364], [-0.39603311], [3.0956315]]),
array([[-0.24941224], [0.43860015], [3.01096514]]),
array([[-0.20412042], [0.19248511], [3.11637753]]),
array([[-0.24336607], [-0.5045509], [3.04557737]]),
array([[0.61271346], [0.30822708], [3.04836008]]),
array([[0.36372836], [-0.10716652], [3.07382184]]),
array([[0.11396109], [0.14390488], [3.00927946]]),
array([[-0.21654534], [-0.73597176], [3.02380176]]),
array([[-0.3553219], [0.05024183], [3.04948686]]),

array([[0.31386242], [0.58803865], [-3.04430249]]),
array([[0.04092159], [0.56449214], [3.04687486]]),
array([[0.13815014], [0.54393731], [-3.05864635]]),
array([[-0.62606041], [-0.4786956], [-3.02374559]]),
array([[0.45008352], [-0.60544681], [2.92540969]]),
array([[-0.30199585], [0.54419748], [2.9742916]]),
array([[0.02432051], [-0.68139636], [3.00553576]]),
array([[0.04732468], [0.19744902], [3.05180377]]),
array([[-0.46343627], [-0.27435296], [-3.07853386]]),
array([[0.54089189], [-0.69385654], [2.86086298]]),
array([[0.01465555], [-0.69571713], [3.01627291]]),
array([[0.12208797], [0.64640443], [2.99475987]]),
array([[-0.20291608], [0.70166861], [3.00209185]]),
array([[0.28064043], [0.93783532], [-2.96495829]]),
array([[-0.14718257], [0.7226459], [3.00588956]]),
array([[-0.04297177], [-0.81386214], [2.9905683]]),
array([[0.53228619], [0.56513942], [3.0322054]]),
array([[0.45373518], [-0.79334027], [2.85861961]]),
array([[0.02444445], [-0.89283528], [2.96169429]]),
array([[0.04265904], [0.4458858], [3.03261545]]),
array([[-0.06546902], [0.60841583], [3.04746939]]),
array([[0.21151061], [0.76848278], [-2.99023863]]),
array([[-0.01272299], [0.41164223], [3.07181544]]),
array([[0.12418191], [-0.24610253], [-3.11532029]]),

array([[-0.15873293], [0.07049613], [2.92159437]]),
array([[-0.14675447], [0.04707395], [2.92189928]]),
array([[-0.32879934], [-0.06777758], [-3.06889339]]),
array([[-0.29113354], [-0.02602072], [-3.08566005]]),
array([[-0.42051418], [0.71417604], [-2.95917088]]),
array([[-0.26848577], [0.76522443], [2.80501513]]),
array([[-0.47470676], [-0.61998033], [3.01481525]]),
array([[-0.04914913], [0.71130533], [2.93967024]]),
array([[0.3300807], [0.65659434], [-3.02685693]]),
array([[0.08898946], [0.5149078], [3.0482395]]))
Translation vector :
(array([[5.70682217], [5.86145679], [29.41371438]]),
array([[5.68070597], [6.16673366], [33.9743772]]),
array([[7.13645509], [4.43917129], [33.69047925]]),
array([[6.86237339], [5.83785581], [31.85338292]]),
array([[5.40922492], [4.38833515], [31.16419256]]),
array([[6.48231405], [2.86442842], [31.84642992]]),
array([[5.56293375], [3.52339625], [33.1656564]]),
array([[5.67303211], [1.6005829], [34.4574324]]),
array([[5.40055956], [-0.19820382], [35.42317334]]),
array([[5.10972623], [2.96837272], [34.23287377]]),
array([[4.67707665], [3.80282722], [35.0108697]]),
array([[4.81519873], [-0.87044106], [35.58518967]]),

array([[5.06739791], [2.71319379], [29.76189959]]),
array([[6.00884862], [0.57813853], [32.80013818]]),
array([[7.79653949], [2.95620267], [36.14549044]]),
array([[5.35887206], [-1.23980766], [38.36238588]]),
array([[4.90027909], [2.16542051], [36.27209456]]),
array([[4.85694315], [-0.14335278], [37.73284189]]),
array([[5.19971782], [2.6365555], [34.89251786]]),
array([[4.3888673], [-1.75871614], [36.82334401]]),
array([[6.27664016], [1.63639691], [31.64106853]]),
array([[6.2645951], [-2.66649378], [32.8578558]]),
array([[5.03211724], [1.17129595], [36.23315459]]),
array([[6.8384191], [-3.11481266], [36.98519256]]),
array([[7.30975086], [2.75463118], [37.05904185]]),
array([[6.07952641], [0.17867322], [34.17042639]]),
array([[6.33165905], [-2.97051714], [34.96301864]]),
array([[6.94056139], [-3.30475523], [36.92176881]]),
array([[6.58813779], [2.64338429], [34.47440601]]),
array([[5.58141347], [1.54706498], [35.56411384]]),
array([[4.98548461], [-2.67684575], [38.531356]]),
array([[4.8198078], [0.4558878], [36.80304117]]),
array([[4.86746842], [-2.94039461], [38.406318]]),
array([[5.40072587], [0.1648664], [33.60538508]]),
array([[7.05845978], [-3.69153939], [35.9637912]]),
array([[5.90771334], [-3.84016443], [36.93434447]]),

array([[5.83360606], [0.78799326], [34.63428138]]),
array([[6.11561715], [-0.97175097], [34.15158035]]),
array([[5.84607832], [-4.89843305], [36.75816239]]),
array([[6.92167594], [10.26097671], [39.42270459]]),
array([[6.21602879], [7.93034104], [38.78081078]]),
array([[5.39135543], [10.86042513], [42.20306718]]),
array([[4.49493731], [9.18368706], [42.18408618]]),
array([[6.54977781], [10.06889005], [37.96553923]]),
array([[7.78156054], [8.11078696], [41.56070939]]),
array([[5.16784207], [4.30745448], [33.58525573]]),
array([[6.40294317], [10.14111674], [36.81515136]]),
array([[5.6450456], [4.83505336], [38.10037119]]),
array([[5.39624987], [7.90058547], [40.06370499]]),
array([[3.80886327], [6.51472411], [43.37860696]]),
array([[9.13884611], [12.28697278], [41.29896075]]))

Total error:
0.22542404913730743

Figure 25: shows the estimated camera matrix, distortion coefficient, rotational and translational vectors for 51 images where the checkerboard was detected.

Secondly, we present the experimental result that proves the proposed method works in integrating simulated parallel lines with that of the lines located in a scene captured with a camera. To validate this method we imported 5 images and render each scenes. 3 of the test images had the width of 4896 and the height of 2752. Since the 3 images are of the same

width and height they produce the same scene(see figure 26). the fourth validation image was however cropped to a width and height of 2752 and a scene was render. Figure 27 shows the resulting image after the translation x axis vector was refined to move the parallel line to the correct position.

Figure 26: Shows the simulated line correctly integrated with the real world

Figure 27: shows the resulting image after the translation x axis vector was refined to move the parallel line to the

The fifth image used was of the same width and height as the first 3 images. However to validate if the proposed method worked the declared image width and height in pov ray were changed. Figure 28 shows the resulting image after the translation x axis was refined

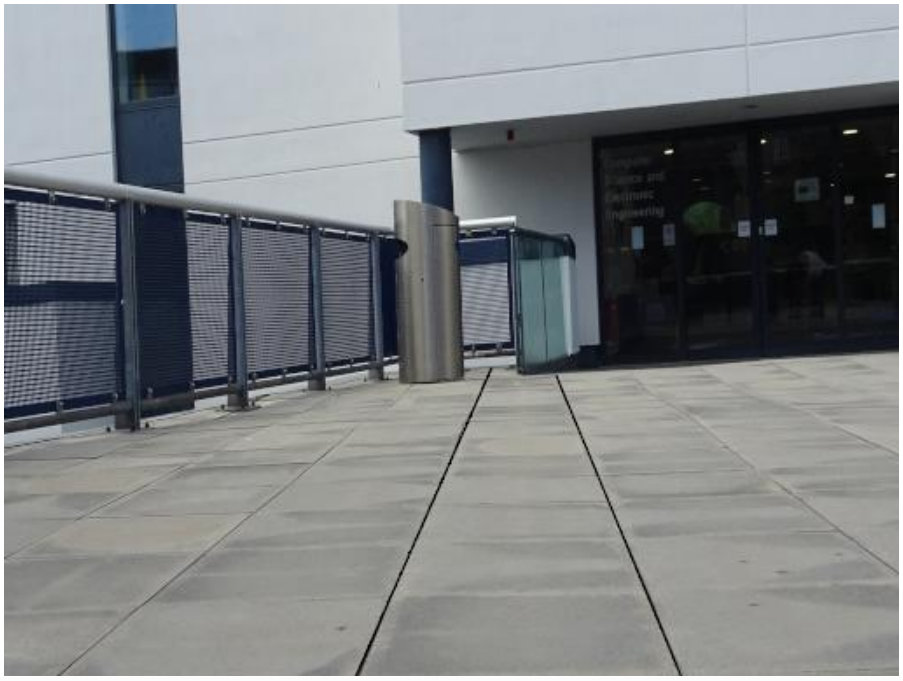


Figure 28: shows the resulting image when the width ,height of the image and the translation x axis within pov ray were change.

4.1 Discussion of the proposed methods

Research conducted in this study has provided insights into the development of augmented reality systems that match computer-generated objects with their real-world counterparts. Even though the research has achieved its intended purpose, a number of concepts still need to be addressed. This section examines some of the significant findings and discusses the implication of this study. Finally, this section discusses some of the limitations and potential drawbacks of the method and suggests further recommendations for future research.

During this study, some significant findings were identified, which should be considered when this study is replicated in the future. First of all, there is a discrepancy between the focal length estimated by OpenCV and the actual focal length stated on the JPEG header of the

images. A software program called Exif tool can be used to view the metadata of any image. Based on the information provided by that software, it can be determined that the focal length of the camera used to capture the image is approximately 4.1mm. This inconsistency may be because focal lengths have been made to be "35 equivalent," as 35 mm was the lens of choice for the vast majority of still cameras.

Furthermore, we discovered another way to define focal length by using the angle keyword. Theoretically, if the focal length could be converted to an angle of view, this could substitute for the directional statement. A camera's angle of view describes how wide the frame of an image will be after it is captured. The angle is typically measured in degrees and can be measured in three directions: horizontally, vertically, and diagonally. An angle of view of a small magnitude produces a narrower field of view, while an angle of view of a large magnitude produces a broader field of view. Generally, the viewing angle of a camera can be determined based on its image dimension and focal length. An equation for determining the viewing angle of a camera can be written as follows:

$$\text{Viewing angle} = 2 \arctan d/2f \tag{16}$$

Where arctan connotates arctangent, which is the inverse of the trigonometry function called tangent, d detonates the dimension of the image, and f is the camera's focal length. Since an image's focal length and dimension determine its angle of view, it is theoretically possible to convert the focal length to the angle of view and input that instead of the direction vector.

Earlier versions of pov ray used the directional vector to adjust the view of a camera. However, this can now be accomplished by using the angle vector. This hypothesis was tested by using the camera's 1/2" sensor, which has dimensions of 6.17 by 4.55 mm. In addition to the dimension, the estimated focal point provided by OpenCV was used to calculate the angle

of view. It was determined that the angle of view in a degree was too large, and when inputted into Povray, the camera was moved too far away from the image. By using the horizontal focal length (2.87590338 e) with the sensor size of 4.55 mm, the angle of view obtained was 76.68. when this was used with the angle keyword on pov ray, the camera moved far away from the image. In contrast, when decimal points are moved, the opposite occurs. It was discovered that the camera zooms into the screen when the decimal point is moved from 76.68 to 7.668.Due to this reason, we were unable to use this method in our project. Moreover, it was discovered that if this was used to represent the focal length, there would be no need to describe the two directional vectors.

A third factor that was discovered during the research was the effect of translation and rotation vectors on a computer-generated object. It was observed that these factors only had a little effect on the geometry of the computer-generated object. However, real changes were observed only when the y-axis of the rotational vector and the x-axis of the translation vectors were experimented on.

Furthermore, we observed a significant effect when the image's aspect ratio was changed within and outside Povray. When testing, it was discovered that if the aspect ratio of the image is changed, the image will not appear as originally captured. If, for example, one examines the original image captured (figure 26) and compares it with the images where the aspect ratio has changed (figures 27 and 28). There is a clear indication that most of the image has been cropped. As a result, when you render the image, the augmented content will appear at a different location and will not be aligned with the Real lines. In order to achieve a good match, the rotational and translation vectors will need to be manually adjusted.

4.2 The implication of this study

To date, this study has been the first to integrate computer-generated objects with the real world using OpenCV as a calibration method and POV ray as graphic library method. According to these results, the same results can be achieved with OpenCV and POV ray, whereas previous studies[2,3] have emphasized the use of other calibration methods and graphic libraries such as Open GL.

4.3Limitation of the Proposed Method

One of the major limitations of the proposed method is that it depends entirely on how the user manually defines each parameter. For instance, a constant manual experiment will be required to define a virtual camera and a computer-generated object until a match is found. Due to this limitation, this method is not recommended for dynamic situations. In the future, if this method is to be replicated, it should be automated. A suggestion is to build a program in Pov ray that automatically sets the location and look-at statement of the camera according to the aspect ratio of the image. Second, the program should be able to calculate the angle of view automatically based on the dimensions of the image and a defined focal length. Additionally, it would be ideal if this program could detect real-world objects ("e.g., lines that separate paving slides on square 1") and automatically generate computer-generated objects before integrating them.

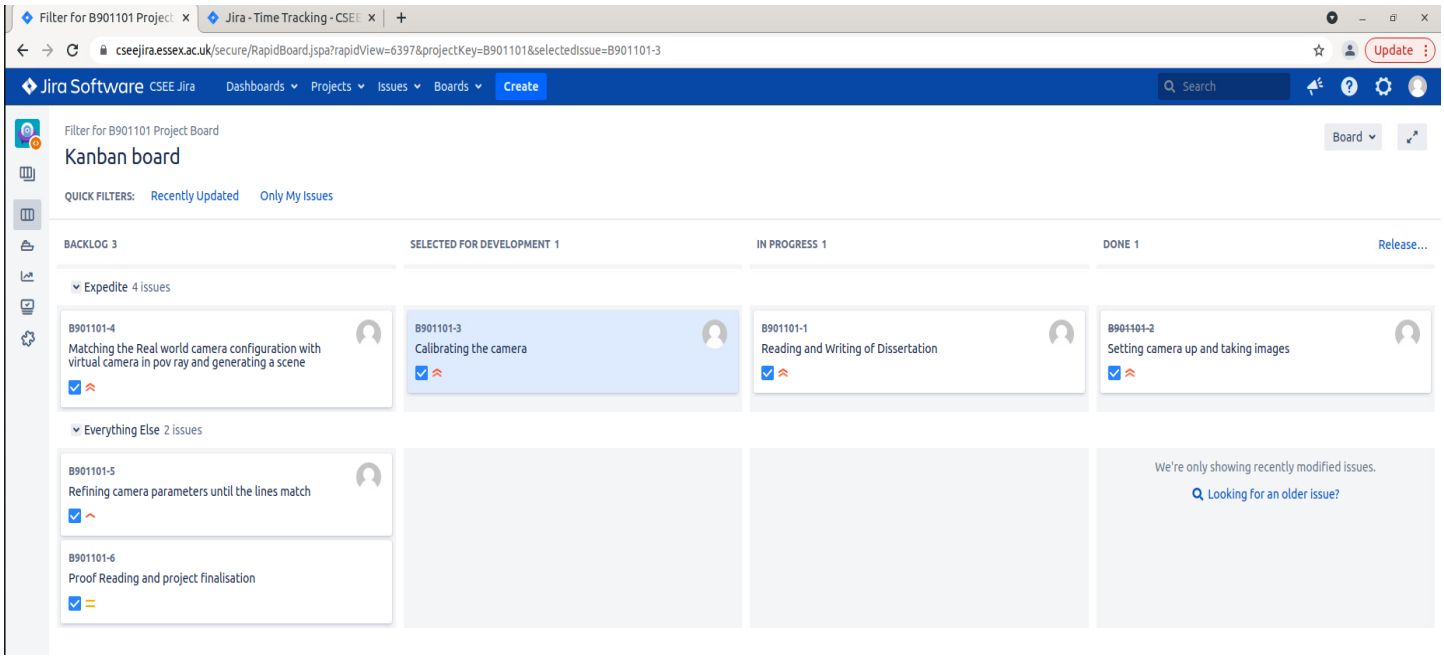
4.5 Project Management

During the development of this thesis, a waterfall approach was used as a methodology. Due to the relatively short duration of the project and the clear definition of the requirements, this approach has been chosen (see 2.3). In addition, we have used two project management software to ensure that the project will run smoothly and efficiently from start to finish.

The first software we used to manage this project was Gitlab, a project management tool. It is an online repository that facilitates the development, security, and management of a software

solution all in one place. As part of this project, we used Gitlab to store all the resources used to complete the project. The repository includes images captured on square 1, open CV camera calibration codes, pov ray codes, the thesis report, and a read-me note.

Secondly, a Kanban board based on Jira software was used as part of the project management process. It is an easy-to-learn and flexible tool that allows users to manage all aspects of software development. Because of this, it was used to manage all aspects of the project. It was discussed in 2.3 that some experimental procedures or objectives would be required to accomplish this thesis's primary goal. Using the Kanban board, each task was created for each objective or step to be completed(see 2.3 and figure 29). Every task was designed in such a way that it would progress according to a specific timeline. Additionally, each task has been given the highest priority based on how important it is to the success of this project



.

Figure 29: This figure illustrates the experimental procedures that were developed in order to achieve the main goal of this thesis.

Kanban boards provide an easy work flow that permits each task to be moved through four distinct phases. This phases includes the backlog, selection for development, work in progress, and work completed. As an example, one of the tasks involves setting up a camera and taking photographs. First, this task was created and moved from the backlog to the selected for the development stage. Then, upon the author's arrival at the University of Essex, the task was moved to the in-progress stage to facilitate the capture of images. Once the camera had captured the calibration checkboard and square 1, the task was moved from the in-progress phase to the Done phase. A similar step was undertaken for subsequent tasks as they moved from the backlog to the done phase. To illustrate this better, a cumulative flow diagram has been provided below to show how each issues progress over time.

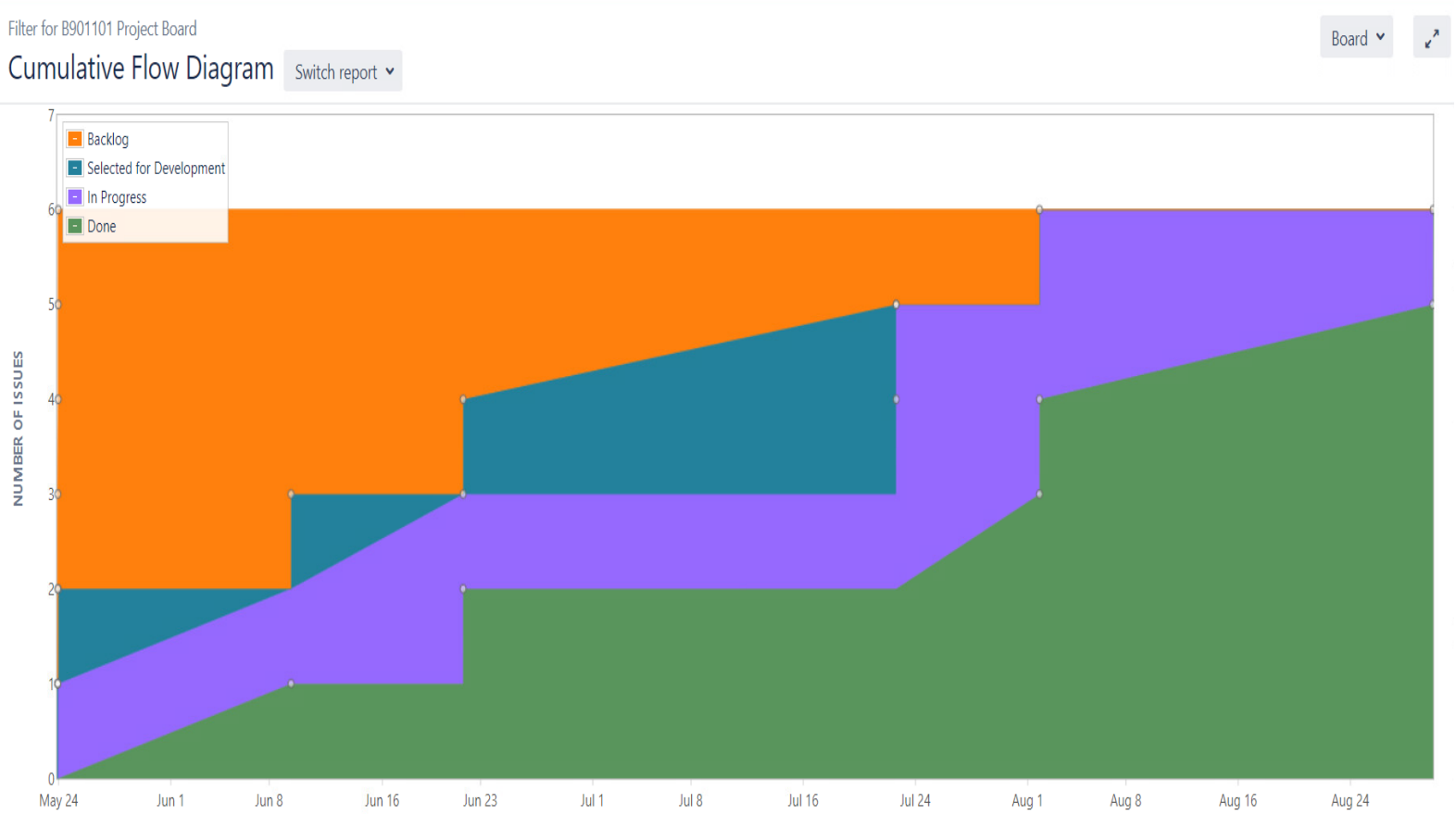


Figure 31: illustrates the cumulative flow diagram as each task progresses through its various phases. It shows the progress of tasks over time and the status of each task over time. Green indicates tasks that have been completed, purple indicates tasks that are in progress, blue

indicates tasks that have been selected for development, and orange indicates tasks that have been added to the backlog.

5.0 Conclusion

In this thesis, a novel method of matching computer-generated objects with the real world is proposed. The method entails capturing a series of images containing a checkerboard pattern with a real camera. Using the checkerboard pattern as a reference and a computer vision library called OpenCV the method computes the intrinsic and extrinsic parameters of a camera. The proposed method then uses the intrinsic parameters to define a virtual camera in pov ray while the extrinsic parameter is used to define the geometry of the computer-generated object. The experimental results show that one can use the physical properties of a camera with conjunction with computer graphics software such as pov ray to integrate computer-generated objects with the real world.

The proposed method is unsuitable for professional use since its results depend heavily on how the user manually defines the parameters. Therefore, a greater effort needs to be made in the future to ensure that this method is as automated as possible. Additionally, this project proposes the integration of numerous kinds of computer-generated objects other than only lines in the future.

Appendices

Camera calibration code based on Open Cv

```
#The following code was adapted from: OpenCV Open Source
Computer Vision, accssed on:Fri Jul 22 2022 01:33:15.
#avaliable
at:https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.
html

import numpy as np
import cv2 as cv
```

```
import glob
# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
30, 0.001)
# prepare object points, like (0,0,0), (1,0,0), (2,0,0)
....,(6,5,0)
objp = np.zeros((9*6,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
# Arrays to store object points and image points from all the
images.

objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob('*.jpeg')
for fname in images:
    print(fname)
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (9,6), None)
    # If found, add object points, image points (after
refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray,corners, (9, 6), (-1,-
1), criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        cv.drawChessboardCorners(img, (9,6), corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)
cv.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1], None, None)
print("\n camera Calibrated", ret)
print("\nCamera matrix:\n", mtx)
print("\ndistortion:\n", dist)
print("\nrotation vector : \n", rvecs)
print("\n translation vector : \n", tvecs)

mean_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i],
tvecs[i], mtx, dist)
    error = cv.norm(imgpoints[i], imgpoints2,
cv.NORM_L2)/len(imgpoints2)
    mean_error += error
print( "total error: {}".format(mean_error/len(objpoints)) )
```

Pov Ray code

The following code was adapted from:

//news.povray.org. (n.d.). POV-Ray: Newsgroups: povray.general. [online] Available at: http://news.povray.org/povray.general/thread/%3Cweb.62dbea13c23a6d94eb0f47cfb08dd02%40news.povray.org%3E/ [Accessed 23 Jul. 2022].

```
#version 3.7;



#declare Image_W = 2752;// declear a magic width number

#declare Image_H = 2752;// declear a magic height number

// perspective (default, not required) camera

camera {

   perspective      // describe the type of virtual camera

   location  <Image_W/2, 10000, Image_H/2,>  //set the location of the camera

   look_at   <Image_W/2, 5, Image_H/2>     // set where the camera is looking at

   right     x*Image_W/2752   //set the aspect ratio of the output screen

   direction<2.87590333,2.79384814 ,0>

   focal_point < 5.37471722, 1.20777920 ,0> // define the focal length of the camera

   aperture 3.6171875                        // define the apperture

   blur_samples 1000                      //define the sampling rate

   direction z                          // direction and zoom
```

```
   angle  15  // field (overides direction zoom) change this to zoom in and
out of the image
}
box{<0,0,0>,<1,1,1> // defines a 3D box
     pigment {image_map {"C:\Users\eobay\Downloads\square-1-pix\2022-06-08-
002.jpeg" once    // use image at this file part as a pigment once
map_type 0  } }  //map unto plannar surface
     finish {ambient 1 diffuse 0} //remove shadow and make visible
     scale <Image_W, Image_H, 0>  // scale to the magic height and width
     rotate<90,0,0> // Rotate into the x,z plane
}

box{<0,0,0>,<1330,2,-7>        // define a long thin 3D box
     finish {ambient 1 diffuse 0} // Make it visible
     rotate<0.70539152,-71.5,2.98700146>  //rotate the 3D box
     translate<1015,6.16673366,33.9743772>          // translate the box to
this co-ordinate
}
box{<0,0,0>,<1330,2,-7>        //  define a long thin 3D box
     finish {ambient 1 diffuse 0} // Make it visible
     rotate<0.61271346,-108,0>  //rotate the 3D box
     translate<1950,2.71319379,29.76189959>  //translate the box to this
co-ordinate  )
}
```

References

[1]Azuma, R.T., 1997. A survey of augmented reality. Presence: teleoperators & virtual environments, 6(4), pp.355-385.

[2]Simek,K.(2012). The Perspective Camera - An Interactive Tour. [Online image] Accessed on 27th June 2022. http://ksimek.github.io/img/1st_and_ten.jpg

[2] F. J. Perales. 1999. Generating Synthetic Image Integrated With Real Images In Open Inventor. In Proceedings of the 1999 International Conference on Information Visualisation (IV '99). IEEE Computer Society, USA, 434.

[3] Ignatov, A., Kim, M. (2006). Multi-view Video Composition with Computer Graphics. In: Pan, Z., Cheok, A., Haller, M., Lau, R.W.H., Saito, H., Liang, R. (eds) Advances in Artificial Reality and Tele-Existence. ACT 2006. Lecture Notes in Computer Science, vol 4282. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11941354_30

[4] Kanade, T., Oda, K., Yoshida, A., Tanaka, M. and Kano, H., 1995. Video-Rate Z Keying: A New Method for Merging Images. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST.

[5] Fadda, M., Wang, T., Marcacci, M., Martelli, S., Visani, A., Nanetti, M. and Dario, P., 1994. Matching between the virtual image and real robot world in robot-assisted surgery. IFAC Proceedings Volumes, 27(14), pp.877-882

[6] Bajura, M., Fuchs, H. and Ohbuchi, R., 1992. Merging virtual objects with the real world: Seeing ultrasound imagery within the patient. ACM SIGGRAPH Computer Graphics, 26(2), pp.203-210.

[7] Morvan, Y 2009, 'Acquisition, compression and rendering of depth and texture for multi-view video', Doctor of Philosophy, Electrical Engineering, Eindhoven. https://doi.org/10.6100/IR641964

[8] Z. Zhang, "A flexible new technique for camera calibration," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, pp. 1330-1334, Nov. 2000, doi: 10.1109/34.888718.

[9] Y. M. Wang, Y. Li and J. B. Zheng, "A camera calibration technique based on OpenCV," The 3rd International Conference on Information Sciences and Interaction Sciences, 2010, pp. 403-406, doi: 10.1109/ICICIS.2010.5534797.

[10] The MathWorks, Inc. n.d. Evaluating the Accuracy of Single Camera Calibration. [ONLINE] Available at: https://uk.mathworks.com/help/vision/ug/evaluating-the-accuracy-of-single-camera-calibration.html. [Accessed 11 July 2022].

[11] Clark, A. Nd. EE222: Interactive Computer Graphics Further POV-ray available at: http://vase.essex.ac.uk/3d-graphics/notes-pov-1.pdf, accessed on: 14th June 2022.

[12] Anon., 2022. *About - OpenCV*. [online] Available at: <https://opencv.org/about/> [Accessed 24 July 2022].

[14] Hata, K. and Savarese, S., 2017. Cs231a course notes 1: Camera models.

[15] news.povray.org. (n.d.). POV-Ray: Newsgroups: povray.general. [online] Available at: http://news.povray.org/povray.general/thread/%3Cweb.62dbea13c23a6d94eb0f47cfb08dd02%40news.povray.org%3E/ [Accessed 23 Jul. 2022].