

# AR Driving Using Mobile Phones

**Timothy Zammit**

Supervisor: Dr Clyde Meli

June 2024

*Submitted in partial fulfilment of the requirements  
for the degree of Bachelor of Science in Information Technology (Honours)  
(Software Development)*



**L-Università ta' Malta**  
Faculty of Information &  
Communication Technology

# Abstract

With the release of new and exciting augmented reality solutions such as the Apple Vision Pro, the one drawback is always the price of such incredible technologies that may benefit the lives of people everywhere. One such area that augmented reality can significantly improve is driving. Augmented reality can significantly improve drivers' experiences by keeping drivers' eyes off the road and therefore reducing accidents and general stress. Augmented reality provides the opportunity to condense all driving information into one space that is constantly accessible to the driver, much like the heads-up display found in some planes and jet fighters.

This research explores a new technique that enables the use of mobile phones to achieve stereoscopic vision that will allow the use of augmented reality solutions on mobile phones. Achieving this allows users to gain access to the wonders of augmented reality using a device most people already own. The research revolved around achieving stereoscopic vision with minimal lag using a single camera. This solution breaks the mold of all modern augmented reality solutions by using a single camera as current solutions use two or more cameras to achieve stereoscopic vision.

The research achieved stereoscopic vision by cropping the video feed from a single camera to match the view of each eye. This proved highly effective, however limitations were tested and identified. These limitations were then clearly defined so that future researchers may have a clear view of what is needed to make augmented reality in mobile phones a viable and trustworthy solution.

# Acknowledgements

I would firstly like to express my gratitude to my parents and family for their constant support and guidance throughout my academic and personal life. A special thanks also goes to Martina Muscat for her unwavering support and guidance during this research. I would also like to thank Dr. Clyde Meli for his guidance during this research. Moreover, I would like to thank Cesca and Luca Giudice for providing the mobile phone used throughout this research. Finally, I would like to thank my friends, and peers in both my academic and personal life for their constant moral support.

# Contents

Abstract .....	ii
Acknowledgements .....	iii
Contents .....	iv
List of Figures .....	vi
List of Abbreviations.....	vii
Chapter 1 - Introduction.....	1
1.1 Problem Definition and Solutions.....	1
1.1.1 The Problem with Driving without AR .....	1
1.1.2 The Benefits to Driving with AR.....	2
1.1.3 The Problem with Current AR Solutions.....	3
1.1.4 The Solution to Make AR Accessible .....	3
1.2 Project Overview .....	4
1.2.1 Project Scope .....	4
1.2.2 Project Methodology .....	4
Chapter 2 - Background and Literature Review.....	5
2.1 Stereoscopic Vision and Binocular Vision .....	5
2.2 React Native vs Kotlin. ....	5
2.3 Pupillary Distance .....	6
2.4 Optimal fps.....	7
2.5 Capture to Display Latency.....	8
2.6 Summary.....	9
Chapter 3 - Specification and Design.....	10
3.1 Stereoscopic Vision Concept .....	10
3.1.1 Obtaining the Images for Each Eye.....	10

3.1.2	Headset Functionality.....	13
3.2	Code Framework.....	14
3.2.1	Setting Defaults .....	14
3.2.2	Setting the Camera .....	14
3.2.3	Outputting Two Video Feeds Simultaneously .....	14
3.2.4	Manipulation Of The Video Feeds .....	14
3.2.5	Artefact Evaluation .....	15
3.3	Calculation of Crop Factor .....	15
3.4	Evaluation of Screen Tearing.....	16
Chapter 4	- Implementation.....	18
4.1	Implementing Stereoscopic Vision .....	18
4.1.1	CameraX.....	18
4.1.2	Camera2 API.....	19
4.2	App Efficiency Evaluation.....	21
4.2.1	Camera To Display Latency .....	21
4.2.2	Average Frames per second .....	21
Chapter 5	- Evaluation .....	26
5.1	Stereoscopic Vision and Image Sharpness Results .....	26
5.2	Capture to Display Latency Results.....	28
5.3	Optimal fps Results and Video Tearing .....	28
Chapter 6	- Future Work .....	30
Chapter 7	- Conclusions .....	31
7.1	Final Remarks.....	32
References	.....	33

# List of Figures

Figure 1 : Forza Horizon heads-up navigation [7].....	2
Figure 2: Jet Fighter HUD [8]. ....	3
Figure 3 : Stroboscopic flash photographs of a moving ball under different frame rates [27] .....	8
Figure 4: Visualization of FOV blocking.....	10
Figure 5 : Full Example Scene .....	11
Figure 6 : Scene from the perspective of the left eye.....	11
Figure 7 : Scene from perspective of the left eye.....	12
Figure 8 : Stitched view of left and right eyes. ....	12
Figure 9 : Resized stitched views. ....	13
Figure 10 : Headset front view and side view respectively. ....	13
Figure 11 : Screen Tearing Test Setup .....	16
Figure 12 : Example of screen tearing [32] .....	17
Figure 13 : Log stating time of lowest fps capture. ....	23
Figure 14 : Graph of CPU, RAM, and Energy usage over one minute. The blue vertical line indicates the 12 second mark.....	23
Figure 15 : Log stating time of lowest fps capture. ....	23
Figure 16 : Graph of CPU, RAM, and Energy usage over one minute. The blue vertical line indicates the 58 second mark.....	24
Figure 17 : Android GPU Profiler .....	25

# List of Abbreviations

2D image	Two-dimensional Image
3D image	Three-dimensional Image
AR	Augmented Reality
CDC	Centers for Disease Control and Prevention
DPCM	Dots Per Centimeter
DPI	Dots Per Inch
FOV	Field of View
fps	frames per second
HUD	heads up display
rpm	revolutions per minute
VSYNC	Virtual Synchronization

# Chapter 1 - Introduction

Augmented Reality (AR) is the process of overlaying a computer-generated image on top of the user's vision of the real world. This provides the user with a different view of reality and can be used to display useful information to a user without the need for physical real estate.

See-through real-time AR headsets are required to replicate binocular vision for users wearing a headset. Pateras E details in [1] that "Binocular vision, developed and practiced during the first years of life, gives us an important advantage, the ability to see and observe space in three dimensions, that is, it gives us a sense of depth. This ability is called 'stereoscopic vision'." Essentially stereoscopic vision works as detailed by OCVT in [2] as follows. A person's eyes each create a two-dimensional image (2D image) of what it sees from the outside world. These 2D images are then merged by our brain to create a three-dimensional image (3D image), allowing us to perceive depth. See-through real-time AR devices need to display these two images relative to each eye as each eye sees objects from a different angle. From this point onwards any reference made to AR devices should be seen as a reference to see-through real-time AR devices unless otherwise stated.

## 1.1 Problem Definition and Solutions

### 1.1.1 The Problem with Driving without AR

A study conducted by the NHTSA in the United States in [3] shows that 3,000 people lose their lives each year in car accidents due to distracted drivers. Study [3] states that this accounts for 8% to 9% of all fatal motor vehicle collisions and that 12% of all accidents occur due to mobile phone usage. Another study done by unitedtires in [4] surveyed the top 20 US cities that contain the most cars per capita and surveyed 1,200 residents from these 20 cities. This study found that 93% of respondents were dependent on their GPS and that 1 in 5 use it every day.

The CDC (Centers for Disease Control and Prevention) [5] states that a driver that is distracted for 5 seconds and at a speed of 55mph(88kph) travels the entire length of a football field (about 100m). All these statistics point towards a causal relationship in which drivers are getting more distracted while driving due to the increased reliance on GPS, namely GPS apps such as Google Maps or Apple Maps which account for 63% and 14% of GPS users respectively as stated by unitedtires in [4]. Since drivers need to check their phone constantly to check for

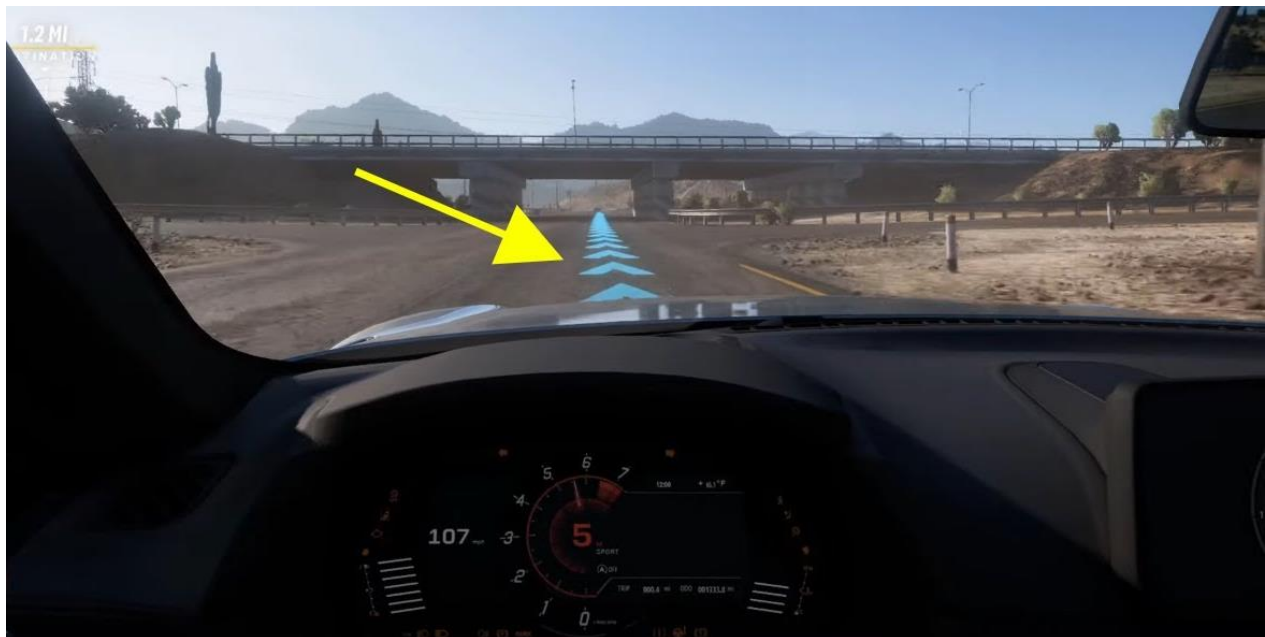


directions their phone needs to be displayed clearly to them. This means that when a notification is received drivers tend to notice it more than if their phone is placed somewhere else out of sight in the car.

From this it can be shown that the core problem is that drivers are required to look away from the road to obtain directions, check their speed, their fuel and even road signage. Also, the increased use of mobile GPS apps such as Google Maps means that users are more likely to get distracted by a phone notification.

### 1.1.2 The Benefits to Driving with AR

AR would help to increase driver focus by reducing the amount drivers would have to avert their gaze from the road. One example of the vision that this study is trying to inspire can be seen in the video game Forza Horizon 5 [6] shown in Figure 1 [7]. This game contains a heads-up style navigation display where the directions the player is required to take are displayed on the road itself. Such an implementation in AR would significantly reduce the time a user spends shifting their attention from the road to their GPS.



*Figure 1 : Forza Horizon heads-up navigation [7].*

Another aspect of AR that could help reduce driver distractions is to include information such as speed, revolutions per minute (rpm) and possibly even road signage. A similar system is already used in most jet fighters and even in some commercial aircraft in the form of a Heads Up

Display (HUD) as shown in Figure 2 [8]. This method of displaying data could significantly help drivers keep their eyes on the road as they would not be required to physically look down to observe important telemetry.



Figure 2: Jet Fighter HUD [8].

### 1.1.3 The Problem with Current AR Solutions

The problem with current AR solutions is primarily that there are no applications that tackle this problem. Such a problem stems from a larger root problem. Current AR hardware devices are extremely expensive, Apple's Vision Pro costs \$3499 [9] and Meta's Quest Pro costs \$999.99 [10]. This means that developing apps for these devices isn't very profitable since not a lot of these devices are being sold as can be seen in IDC's study [11] that amounted the total shipment of all AR devices to just 8.5 million units in 2023. Therefore, an AR driving solution needs to use hardware that is both available to the general public and fast, to ensure drivers have minimal visual delay while driving that could hamper their reaction time.

### 1.1.4 The Solution to Make AR Accessible

A study done by Statista [12] shows that in 2023 the smartphone industry shipped 1.17 billion units worldwide and another study by Exploding Topics [13] estimates that there are about 6.94 billion smartphones worldwide which accounts for approximately 85% of the global population.

Since the vast majority of the population has access to a smartphone, being able to create see through AR mobile applications would give many drivers access to AR Driving. The only add-on that would be required to a user's mobile phone is a headset case to hold the phone to the user's face. One example of this is the Vizor Pro listed at around \$18.98 on GGG [14]. Such a device will have to be modified for the purpose of a see-through AR application for this study.

## 1.2 Project Overview

### 1.2.1 Project Scope

The scope of this project is to attempt to achieve stereoscopic vision using a single camera with minimal lag. This can be expressed as two scopes:

1. Currently every AR solution that displays 3D content to users uses at minimum two cameras; one camera is assigned to one eye as achieved by X. Kang et al. in [15]. This is not possible with a mobile phone and therefore the main aim is to find a way to achieve stereoscopic vision using a single camera lens.
2. Once a proof of concept is built to show that the methodology of creating stereoscopic vision using a single camera works, then the code is to be streamlined to ensure that there is minimal lag.

### 1.2.2 Project Methodology

The research will aim to solve problems to the above two scopes. The methodology for each scope is defined below.

1. To achieve stereoscopic vision with one camera, the video feed must be duplicated and slightly cropped depending on which eye the feed is being shown to. This will be attempted to be achieved using multiple viable technologies. A sharp image must be produced for this to be achieved and the field of view must be as close to reality as possible.
2. A few considerations must be looked at. These are frames per second (fps), screen refresh rate, CPU usage, RAM usage and finally time from frame capture until frame display. These must all be compared to what levels are acceptable.

## Chapter 2 - Background and Literature Review

In this chapter, an overview of the knowledge required for this research is provided. This knowledge was mainly used to test the results of the research and provide a benchmark that was required to be reached during the testing of the final artefact.

### 2.1 Stereoscopic Vision and Binocular Vision

Stereoscopic vision is identified as “the ability to classify objects we see in the 3D world” as detailed by Paternas E [1]. Essentially, stereoscopic vision refers to the ability to see and recognize depth. Achieving stereoscopic vision is extremely important in VR and AR solutions today as for users to be able to be fully immersed, 3D vision is required as the absence of this can lead the user to feel disconnected and confused. This is even more prominent for this research as drivers on the road need to be able to judge the distance to other cars.

Humans achieve stereoscopic vision through what is known as binocular vision which is described as Paternas E [1] as “the combined use of the two eyes to create a single and unique brain impression”. In simple terms, each eye has a slightly different perspective of the world, once the eyes receive light from the real world, the brain interprets that light into two 2D images, one for each eye, and blends those images together to create a 3D image and allowing humans to perceive depth. VR and AR take advantage of this by showing a different image to each eye, emulating what each eye would see, allowing the brain to blend the virtual world being created into a 3D image. Stereoscopic vision is achieved either through the use of multiple cameras as done in the Apple Vision Pro [9] the Meta Quest Pro [10] and the VSII Vision Sense as used in [15] and detailed in [16].

This literature highlighted how important it was to achieve stereoscopic vision in order for AR to be relevant in mobile phones. It revealed that stereoscopic vision must be achieved using a single camera, which is what this research revolves around.

### 2.2 React Native vs Kotlin.

Choosing the correct technologies for this research was of the utmost importance as complex video manipulation processes were required to be made while keeping the application as fast and streamlined as possible. To achieve this balance two technologies were most prominent: React Native and Kotlin.

Kotlin [17] is compiled to bytecode meaning that it has a fast execution time for computation-heavy applications such as those involving real-time camera processing. It also has many technologies and APIs that may be used with it and eliminates null pointer exceptions which adds an extra layer of robustness to any application as any potential errors are discovered at compile time instead of at runtime. The downside of using Kotlin is that it is predominantly used for Android development, meaning another language such as Swift must be used for iOS development. Kotlin has two main camera APIs, CameraX [18] and the native Camera2 API [19]. Both of these APIs have extensive real-time video manipulation features.

React Native [20] has is extremely easy to develop for both iOS and Android as well as having many technologies and APIs that may be used. The use of the JavaScript engine, however, can introduce performance overhead when running real-time processing tasks. While using React Native, there are two main camera APIs that can be used with React Native. These are React Native Vision Camera, and Expo Camera. As can be seen from their developer guides in [21] and [22] respectively, these technologies do not support any real-time video feed manipulation. This means that these features would require native modules to be coded and integrated into the React Native code.

After reviewing the above pros and cons it was decided that Kotlin would be used. This is due to two main reasons. Firstly, Kotlin's bytecode compilation is faster than the JavaScript engine which uses an interpreter which means that real-time processes will experience less lag using Kotlin. Secondly, React Native has no camera technologies that enable real-time video feed manipulation, meaning that such methods would have to be coded in their native modules anyway. Since this is still preliminary research it was decided to narrow the minimal viable product and that the app would be developed in just one mobile operating system. With all this in mind an Android Kotlin app was chosen to be developed. Both CameraX and Camera2 were chosen to be used and compared.

## 2.3 Pupillary Distance

Pupillary distance is extremely important to this research as it is what is used to calculate how much the video feed must be cropped in order to produce a clear and sharp image. Pupillary distance is the distance between pupils and can vary from user to user. Having pupils at different distances means that people have different points of view and would require different images to be shown to them to achieve stereoscopic vision. To cater for this the average pupillary distance

needed to be found. From here the median distance of the average could be used as a default to calculate the image crop and then the built-in lenses could be used to fine tune for different people. There are multiple sources that have similar adult average distances. Glasses.com [23] says distance range from 54mm - 74mm, Zenni [24] 54mm - 74mm, All About Vision [25] 63mm on average with a range of 51mm - 74.5mm in women and 53mm - 77mm in men. Finally, an official survey by the US Army Personnel [26] found the average in women to be 6.17cm and 6.4 cm in men. For the purpose of this research the survey done by the US Army will be used to keep to a single standard since the variance is negligible. Therefore, the average pupillary distance being taken into account is the average between men and women which is 6.29cm rounded to two decimal places. This is the distance that will be used for calculations using pupillary distance for the remainder of this research.

## 2.4 Optimal fps

When measuring the efficiency of video output refresh rate and fps (fps) are extremely important. Refresh rate is the frequency at which the image on a screen is refreshed while fps refers to the number of frames the program is displaying to the screen per second. This distinction is important as refresh rate is determined by the hardware capabilities of the screen while fps is dictated by the software and in some cases like this, the amount of image the camera is able to capture.

One issue that arises between fps and refresh rate is that having these out of sync can lead to tearing as mentioned in the limitation of [27]. Tearing is when the computer screen renders multiple frames simultaneously due to a disparity in fps and refresh rate. Due to this the ideal scenario is to implement Virtual Synchronization (VSYNC), which is the process of matching the fps and the refresh rate by throttling one or the other. From this point, when referring to fps we will automatically assume that the refresh rate matches the fps unless otherwise stated.

While VSYNC helps to solve the issue of tearing it is important that the fps is as high as possible. This is because higher fps means that the movement of an object appears smoother and more cohesive. This can be explained in Figure 3 : Stroboscopic flash photographs of a moving ball under different frame rates [27]. At 60 fps, the moving ball appears at larger intervals compared to at 180fps, resulting in a less smooth video. To conduct appropriate benchmark tests two questions needed to be answered. What is the maximum possible frame rate perceived by

the human eye and what is the minimum frame rate needed for a passable prototype?

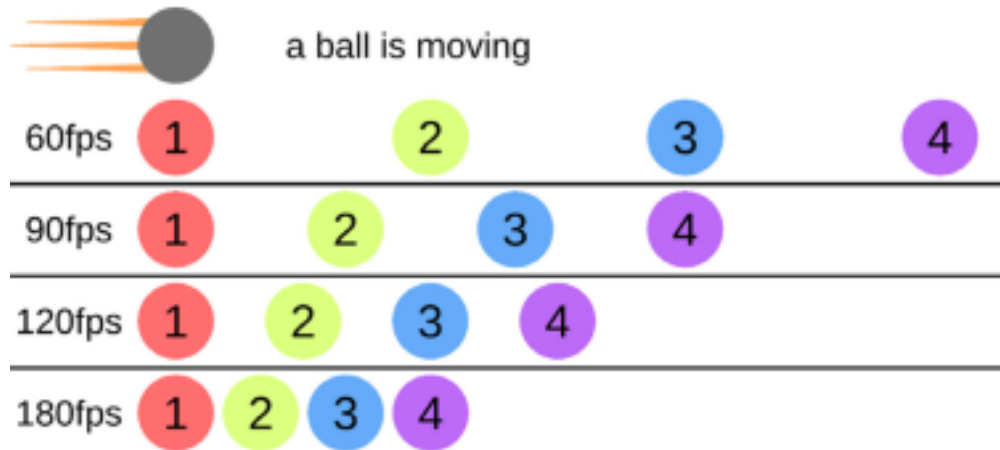


Figure 3 : Stroboscopic flash photographs of a moving ball under different frame rates [27]

J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang [27] tested VR users' responses and sickness at varying fps to find out the impact higher fps has on users. It was found that 120 fps is an important lower threshold for VR devices, as this is where users tended to feel less simulator sickness and exhibited the highest visual accuracy. Their general conclusion was that 120 fps should be the minimum in VR applications that deal with fast moving objects. Their study tested the ability of users to track a piece of fruit flying at up to 40m/s or 144km/hr over a distance of 50m. This provides a good benchmark study to use for this research as the vast majority of countries do not have roads that exceed 140km/hr [28],[29]. The conclusion that higher frame rates improve tracking and depth perception is corroborated by Kuroki Y [30] as he concluded that viewing 3D clips in 240 fps provided smoother motion and better natural depth impression.

J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang in paper [27] also noted that at 60 fps users resorted to rely on prediction-based methods to calculate where an object is going to be, especially with objects at higher speeds. This is therefore less than ideal for the purpose of driving and therefore 120 fps is set as the minimum target framerate to be reached in this research.

## 2.5 Capture to Display Latency

The final area of performance that was looked at was the latency from the time of camera capture until the display onto the user's screen. M. Walter, T. Wendisch, and K. Bengler in [31] created a practical experiment in which they tested the reactions and car positioning of multiple

participants at varying levels of latency. The tested latencies were 17ms, 50ms and 100ms and found that any latency of 50ms and above had a significant impact on drivers. Therefore, for the purpose of this research a latency of less than 50ms is desired.

## 2.6 Summary

Firstly, the definitions of stereoscopic vision and binocular vision were explained and their importance to this research stated. After this a cost-benefit analysis was conducted between using React Native or Kotlin while also introducing technologies that will be used during this research. Following this, the concept of pupillary distance was introduced, its importance stated and the pupillary distance to be used for the purpose of this research was decided. Subsequently, the topic of the optimal fps was discussed as well as the relationship between refresh rate and fps and their influence on one another. This resulted in the decision to set the benchmark fps to 120 fps. Finally, a benchmark was reached on the maximum latency allowed between the time of camera frame capture and the display of said frame to the user.



## Chapter 3 - Specification and Design

This chapter explains the proposed solution's concepts and design as well as the testing concepts that were used in order to evaluate the app's effectiveness.

### 3.1 Stereoscopic Vision Concept

As mentioned in Section 2.1, the issue with current AR solutions is that they use two cameras. Each camera is assigned to an eye and serves as the viewpoint of that eye, which is then displayed to their respective screen, achieving stereoscopic vision. In a mobile phone, however, this must be achieved using a single camera. For this a new concept was developed in order to achieve stereoscopic vision.

#### 3.1.1 Obtaining the Images for Each Eye

The new concept is centered around the view of each eye. The left eye is able to see more on the left of a person than on the right due to the nose blocking the visual of the left eye as shown in Figure 4. In this figure using average proportions, it is clear to see how the Field of View (FOV) lines stretch out further on their respective side but are obstructed by the nose. The same is true for the right eye.

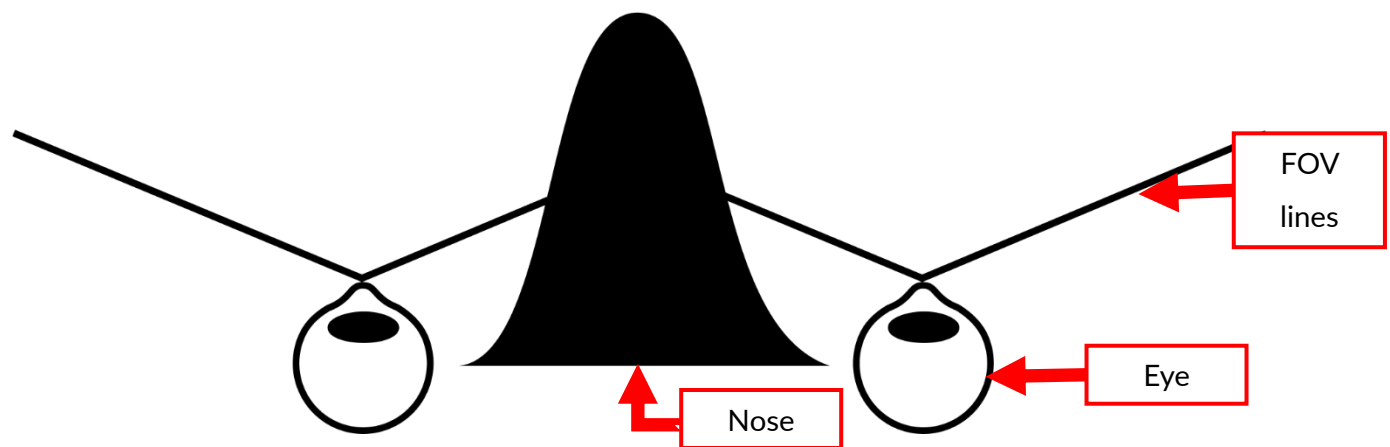
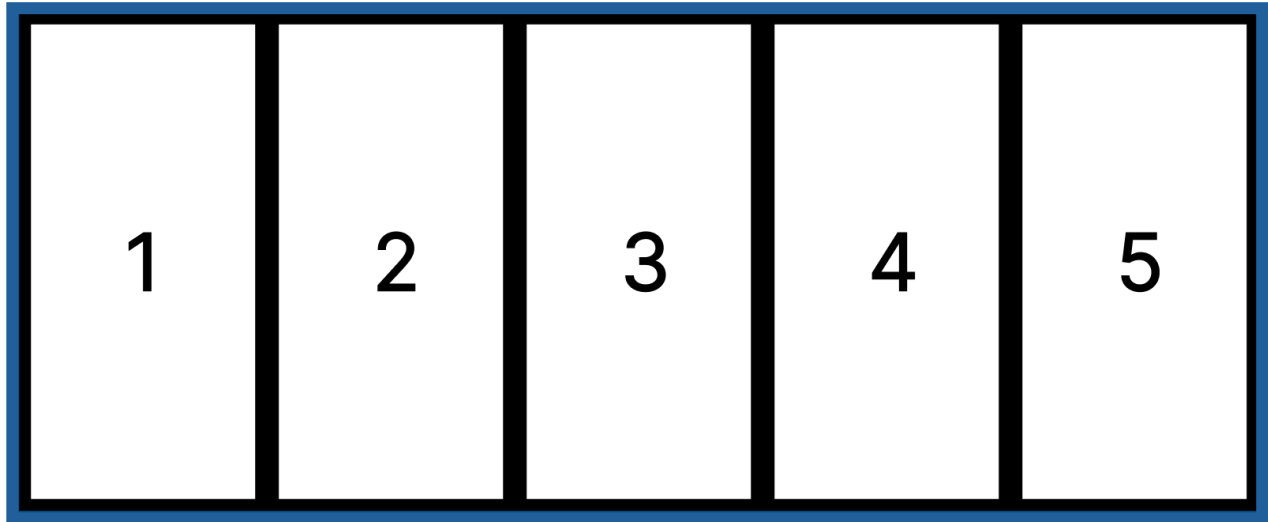


Figure 4: Visualization of FOV blocking.

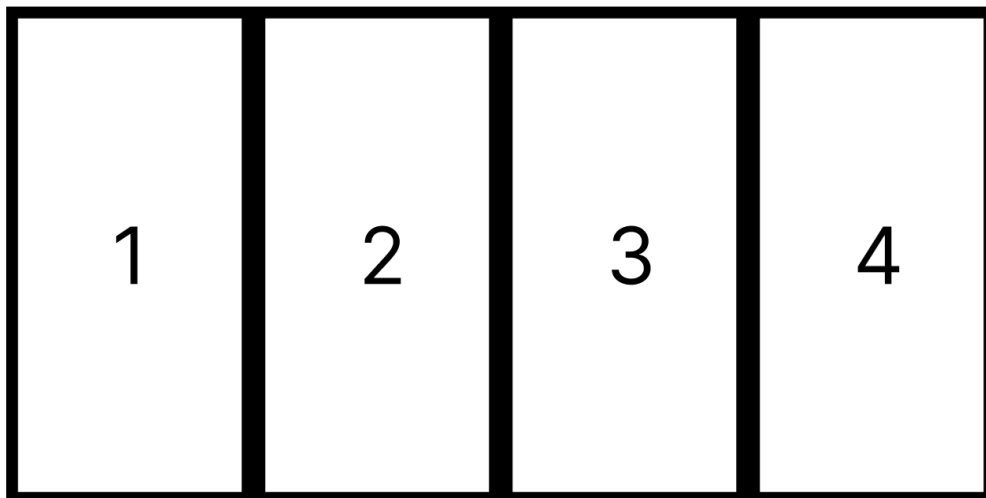
Due to this, when looking at a scene, each user's eye is only able to see part of the whole scene. In the case of the left eye, it is able to see more of the left side of the scene while seeing less of the right side of the scene. An example scene would be the image given in Figure 5. To clarify this is a representative image that is captured by the camera on the mobile phone. In this scene each box translates to 3.145cm, half the pupillary distance. This is done to visualize how the pupillary distance interacts with the scene. Also, for this case a blue line is used to denote the

size of the screen. If no blue line is present, as in Figure 6 and Figure 7, this means that the visuals are theoretical and not yet displayed to the screen. Finally, a red line is used to denote a difference between views for the left and right eye.

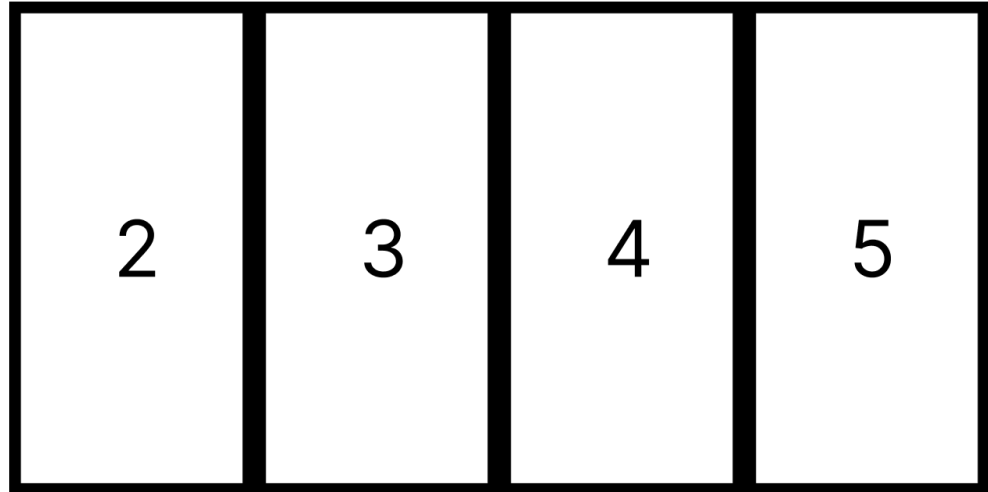


*Figure 5 : Full Example Scene*

Since half the pupillary distance is 3.145cm, the scene must be adjusted by 3.145cm according to the eye. Therefore, in the left eye, box 5 must be removed as is shown in Figure 6. This is the part of the scene constantly being shown to the left eye. The same is done for the right eye except in reverse. In this case box 1 is removed instead, as shown in Figure 7.

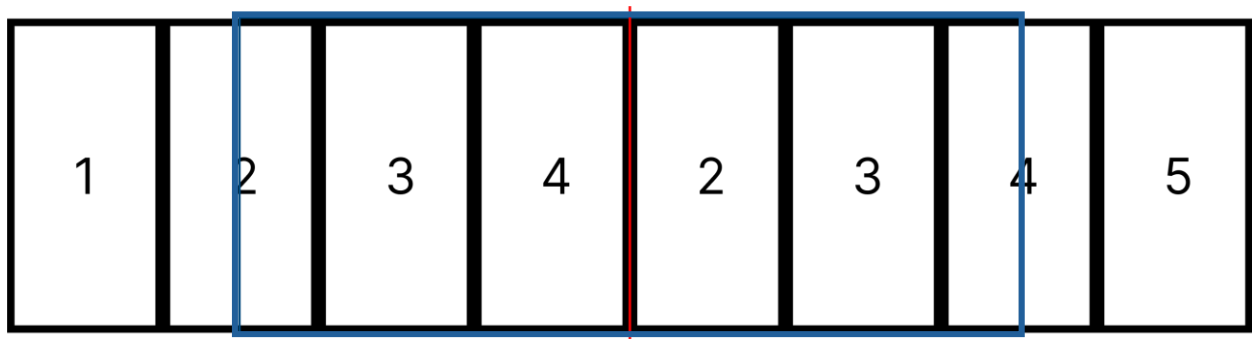


*Figure 6 : Scene from the perspective of the left eye.*



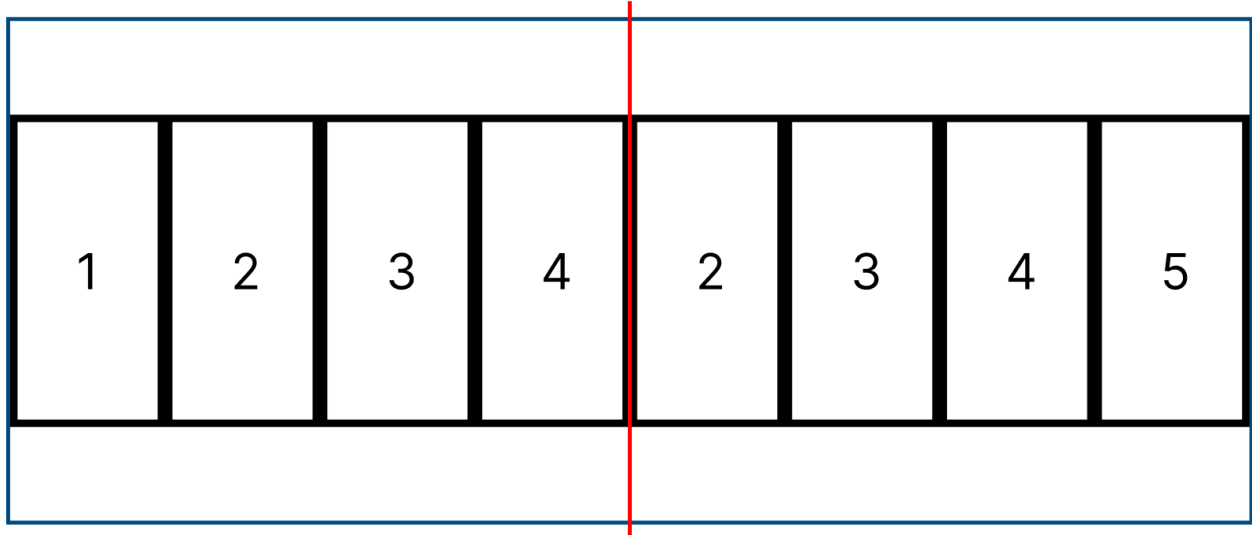
*Figure 7 : Scene from perspective of the left eye.*

These views are then stitched together and displayed to the screen as shown in Figure 8. However, as is clear to see from Figure 8, without any resizing the view extends past the screen. This results in a very blurry image. Stereoscopic vision is still somewhat achieved as depth can still be perceived; however, this is due to the lens adjusters on the headset and the image is extremely blurry. On top of this, the view appears extremely zoomed in making objects appear closer than they are in reality.



*Figure 8 : Stitched view of left and right eyes.*

To fix this issue both views are resized to fit the width of the screen while keeping the aspect ratio the same as shown in Figure 9. This provides sharp stereoscopic vision with adequate depth perception.



*Figure 9 : Resized stitched views.*

### 3.1.2 Headset Functionality

The headset being used for this research is a modified version of the Vizor Pro [14]. The front of the headset was removed and replaced with two elastic bands to allow a phone to be positioned without blocking the view of the camera as seen in Figure 10. This headset is fairly simple and cheap; however, it still contains two adjustable lenses which are extremely important. These lenses focus the images from the screen into the user's eyes. This allows the image to be adjusted to suit the needs of anyone using the headset. It means that this device can cater for users who have vastly different pupillary distances or even for people who wear glasses.



*Figure 10 : Headset front view and side view respectively.*

## 3.2 Code Framework

A general framework was developed to ensure that the code for both CameraX and Camera2 had set directives to follow and therefore provide a fair comparison between the technologies.

### 3.2.1 Setting Defaults

Firstly, the app must set certain defaults. These defaults include locking the app rotation to landscape regardless of device settings, disabling the sleep feature while the app is active and removing any default device navigation bars such as on-screen home buttons etc. The app rotation must be locked into landscape in order to prevent the app from randomly rotating by accident, resulting in a loss of vision for the user. Furthermore, the sleep feature must be disabled as users won't be touching the screen in any way while using the headset. Therefore, the lack of input would trigger most phones to turn off automatically. The app must ensure that it flags the app to the phone to ensure it stays on. Finally, navigation bars must be removed in order for the stereoscopic vision to remain smooth and for users to have a fully immersive experience.

### 3.2.2 Setting the Camera

Secondly, the app must access the rear facing camera and must instruct the camera hardware to zoom out as much as is possible. The zoom feature is used to ensure the camera does not zoom in by default limiting the users FOV and also so that if the camera is capable of supporting 0.5x zoom the camera does so increasing the user's FOV.

### 3.2.3 Outputting Two Video Feeds Simultaneously

Thirdly, the app must be able to output the same video feed to two separate views. This is done before attempting to crop the video feed as manipulating a video feed in real time is generally an incredibly complex task. This means that it is better to first ensure that two feeds can be displayed and eliminate one of the technologies should it not have the capacity to do so. Once both feeds are being displayed, the resolution of both video feeds is resized such that the entire width of the video feed fits inside the designated area for each feed.

### 3.2.4 Manipulation Of The Video Feeds

The fourth and most complex task is the manipulation of the video feeds in real-time. For this there are two methods that appeared to be the most viable and both would be tested. The first method is to simply crop a part off of each video feed and subsequently fit the resolution appropriately and display them on screen. It should be noted that the resolution will always be

scaled in such a way that the native aspect ratio is maintained to prevent stretching and distortion. The second method was to increase the resolution of each feed while locking their edges to the respective edges of the screen. In practical terms, the left video feed which is being shown to the left eye would have its left edge locked to the left edge of the screen. The same would be done for the right video feed in reverse. The views that would be holding the video feed would then be set to not allow overflow, meaning that any content outside this view is not shown. The increase in resolution coupled with the locked edges means that the opposing edges of the content are pushed to the center of the screen and cut off by the view since overflow is not allowed.

### 3.2.5 Artefact Evaluation

The final task is to implement the evaluation code. The tests that can be evaluated programmatically are the average fps and the average time taken from camera capture until that frame is displayed (referred to as capture to display latency). In both cases the average will be calculated over one minute of runtime and five minutes of runtime. To calculate both averages with minimal strain on the device's memory and CPU usage an iterative average formula is used, given by (3.1).

$$New\ Average = Old\ Average + \frac{New\ Value - Old\ Average}{Total\ Number\ of\ Values} \quad (3.1)$$

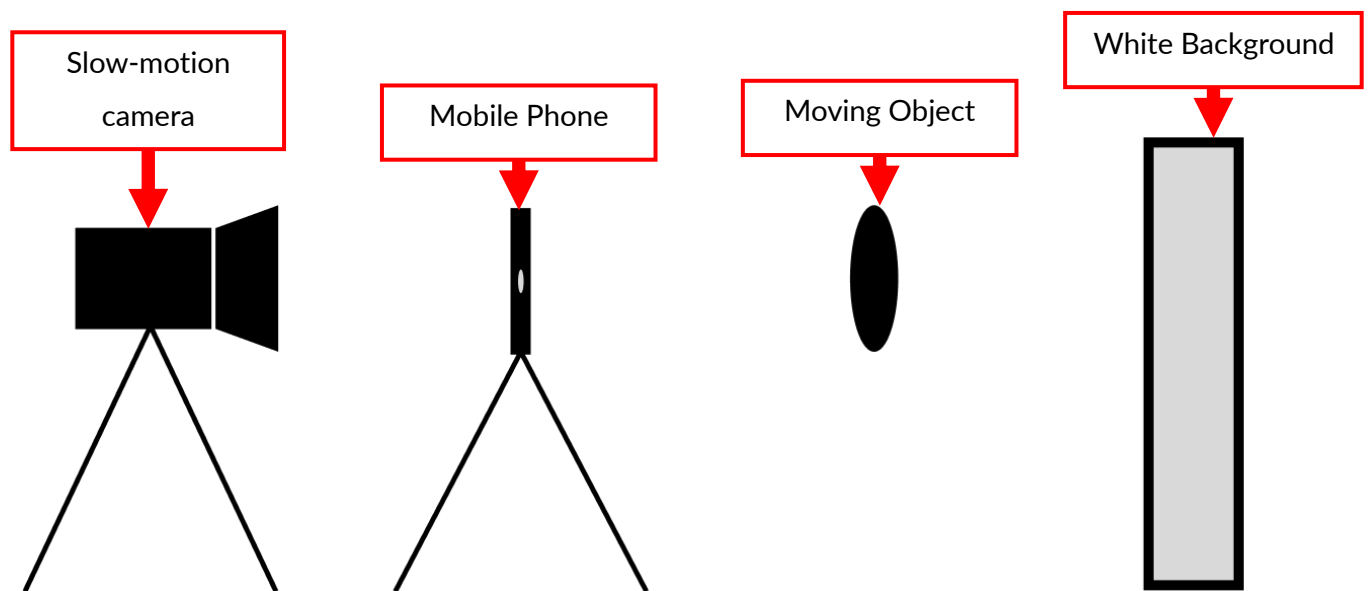
## 3.3 Calculation of Crop Factor

In Section 3.1.1 it was covered that the edges of the screen must be cropped by 3.145cm for each eye. However, in practice centimeters cannot be used and using pixels is unreliable as the image might require manipulation such as rotation or scaling, which may interfere with the pixel number obtained. For this reason, a cropFactor is used which is a percentage of the image to be cropped. This means that to calculate the amount of the image required to be cropped the width of the image may be obtained and multiplied by the cropFactor. The cropFactor is calculated using equation (3.2) where DPCM is dots per centimeter. DPCM can be calculated by obtaining the DPI (dots per inch) and dividing that number by 2.54 to convert from inches to cm.

$$cropFactor = \frac{cropWidth + DPCM}{viewWidth} \quad (3.2)$$

### 3.4 Evaluation of Screen Tearing

Since screen tearing has no defined metric or programmatic test to detect when screen tearing occurs, a more practical test must be implemented. This is done by setting up a stationary slow-motion camera to record the screen of the testing phone running the stereoscopic solution, which is also set up in a stationary position. The slow-motion camera will then record the stereoscopic phone as an object such as a pen is passed in the view of the testing phone at varying speeds against a white background. The white background will ensure that the pen can be clearly seen. This setup is drawn in Figure 11.



*Figure 11 : Screen Tearing Test Setup*

The slow-motion video that was recorded with the slow-motion camera will then be reviewed frame by frame to check for screen tearing. As can be seen in Figure 12, screen tearing is when same object appears to be offset by a small amount. It is important to note that the screen is refreshed from the top of the phone screen to the bottom when the phone is in portrait mode. Therefore, when the screen is in landscape mode with its camera on the left from the point of view of the user, the screen refreshes from left to right. This means that as opposed to Figure 12 the difference in image will be evident on a vertical line as opposed to a horizontal line.



*Figure 12 : Example of screen tearing [32]*



## Chapter 4 - Implementation

### 4.1 Implementing Stereoscopic Vision

#### 4.1.1 CameraX

The first step in implementing CameraX was creating the `activity_camera.xml` which dictated the visual layout that would be used to render the on-screen content. A constraint layout was used, and the orientation was set to horizontal to ensure app remained in landscape mode and did not rotate by accident. This is the first needed default to be set as defined in Section 3.2.1. From this point forward referring to the height and width of the screen is taken from the perspective of a landscape orientation. This means that the height refers to the short edge of the phone while the width refers to the long edge.

CameraX uses a `PreviewView` that allows users to see what they are about to capture using their camera. For the purpose of this research the `PreviewView` is used to display the camera capture. Two `PreviewViews` were initialized nested inside of two `FrameLayouts`. This was done to achieve clipping. The `FrameLayouts` were defined such that they took up the entire height of the screen and half the width of the screen, barring islands and insets as these may block certain items from being seen. The size of the `FrameLayout` is not changed and is used to clip any overflowing content. The `PreviewViews` are then attached to the respective side of each `FrameLayout` such that when their width is increased, it increases towards the center. This means that the left edge of the left `PreviewView` was bound to the left side of the `FrameLayout` allowing the image to be widened towards the right edge of the left `FrameLayout`. Once the width exceeded the `FrameLayout` it would be clipped.

The next step was to create the class `CameraActivity.kt`. This class is responsible for holding the code that captures and displays the camera content to `activity_camera.xml`. The `onCreate` method is run on app startup and therefore was used to initialize the left and right video feed view, add a flag to ensure the screen does not fall asleep while running the app, grant permissions to use the camera and to set the app to full screen mode. The full screen mode removes all extra on-screen bars such the default Android battery bar and home bar. These can be accessed by swiping the phone and are removed to make full use of the screen and ensure that the app is as immersive as possible and to minimize any variables that may influence the achievement of stereoscopic vision. The `onCreate` method achieves all remaining defaults mentioned in Section 3.2.1.

Once permissions are granted, the `startCamera()` function was run. This function created two separate previews for the left and right previews allowing the same video feed to be streamed to both eyes simultaneously. The next step was to resize the video such that the video's width fit completely inside the `FrameLayout`. The first way this was attempted to be achieved was by changing the height and width using a self-made method. The method obtained the height and width of the `FrameLayout` and matched the width of the `PreviewView`. The height was then scaled down using the aspect ratio. This did not work as the image refused to be scaled down. Many variations of this method were tried, the `PreviewViews` were set to `Odp` and also set to match parent, the height and width were also swapped to ensure that there was no confusion with the landscape orientation. And these solutions were all tried with each other. Therefore, a new method was tried.

The new method used was setting the `PreviewViews` scale type to `FIT_CENTER`. This scaled the videos to the center of the screen thus completing requirement 3.2.3. The next step was to crop the image. For this an `adjust preview size` method was used to scale up each image, widening each one which would then be clipped by the `FrameLayout` due to overflow. This method however did not work as the images were not clipped by the `FrameLayout` and were instead left to overflow. Since requirement 3.2.4 was not achieved and no other methods like live cropping exists `CameraX` was deemed to be insufficient for the purpose of this research.

In conclusion `CameraX` does not have sufficient tools to be able to scale and manipulate the live video feeds as is needed. This meant that the next step was to attempt using the `Camera2` API.

#### 4.1.2 Camera2 API

The `Camera2` API is a lower-level API compared to `CameraX` and therefore offers more control over the camera hardware and is more flexible. The `activity_camera.xml` file used the same constraint layout code as the `CameraX` code to satisfy the requirement for a landscape layout. However, instead of using a nested `PreviewView` a `TextureView` was used instead to cater for `Camera2`.

The `onCreate()` method used in `CameraActivity.kt` remained the same as the code used for `CameraX`, satisfying all required defaults from 3.2.1. Once all defaults were set and the program was resumed the camera was opened using the `openCamera()` method. This method obtains the required camera ID and opens the obtained camera. The `stateCallback` is then used to handle

camera events such as error disconnect and what happens when the camera opens. Once the camera is opened the `onOpened()` method is run and `createCameraPreview()` is run.

`createCameraPreview` sets up the necessary settings for the `TextureViews` and initializes the capture session. In the `captureRequestBuilder` both left and right `TextureViews` are added and initialized in the session creation. This enabled the application to display two video feeds simultaneously, satisfying the requirement in 3.2.3.

With the dual video feed working the next requirement of manipulating the video feeds as required was undertaken. To achieve this a new method was created called `configureTransform()`. This method was inserted twice (one for each view) in the `createCaptureSession` once the `onConfigured()` method was triggered. The `configureTransform()` method required the width and height of the `TextureView` to be passed as well as a true or false depending on whether the `TextureView` was the left texture view and the `cropFactor`.

The method first ran a check to ensure each variable was passed. Next the required variables were initialized. `totalRotation` is used to rotate the image to match the screen orientation, the `Matrix` object is used to apply transformations to the image, `viewRect` refers to the rectangle representing the area of the `TextureView` and `bufferRect` is the rectangle representing the camera output dimensions. `centerX` and `Y` are obtained from `viewRect` and are therefore the center of the `TextureView` on their respective axis. These are used as a pivot point for rotations and scaling. Next the `pixelCropFactor` is initialized. Note that an extra 0.5cm is added to the `cropWidth` to cater for insets.

The `bufferRect` is then centered inside the `viewRect` and subsequently made to scale to fit inside the `viewRect` using `Matrix.ScaleToFit.FILL`. This ensures that the `bufferRect` fits inside of the `viewRect` and that any overflow is cropped. It also scales the height of the `viewRect` as the width of the `viewRect` is scaled up. The matrix is then rotated 90 degrees anti-clockwise so that the image matches the orientation of the screen.

Next, the width of the matrix is increased by the `pixelCropFactor`, and the new image is translated left or right by the value of the `pixelCropFactor`. This is done to re-center the scaled image, meaning that the left feed is translated right while the right feed is translated left. The matrix is then assigned to its respective view.

Once the translation is complete `updatePreview()` is called which is used to constantly capture each frame and display it to the user.

This method proved highly effective, providing a sharp, stable image that achieved stereoscopic vision. Therefore, this code was used to carry out performance tests and look for any improvements that may need to be made.

## 4.2 App Efficiency Evaluation

The solution had two aspects that were possible to test programmatically. These were the camera to display latency and the fps. These were both tested individually to ensure that each test has as little impact on app performance as possible.

### 4.2.1 Camera To Display Latency

To test the camera to display latency the `System.nanoTime()` function was used. The first time is taken in the `captureCallback` and is captured after every frame is received. Then once the surface updates in `onSurfaceTextureUpdated()`, the time of surface update is also captured. From this point the time elapsed from camera capture until display is calculated by simply subtracting the capture time from the display time and converting this from nano seconds to milliseconds. Once this is calculated the average may be calculated.

The variable containing the average is labeled `oldAverage`. If `oldAverage` is zero then this is the first iteration of the accumulative average and therefore the `startTime` is set to the `captureTime`. This is done so that 1 minute may be timed from the first capture until the last capture. For every frame that is captured the accumulative average is calculated using the formula defined in 3.2.5. Once the one minute is finished the average is output and the start time is reset.

### 4.2.2 Average Fps

To capture the average fps a similar method to the camera to display latency was implemented. The difference in this code is that the starting time is the time from when the last frame was captured and also that `SENSOR_TIMESTAMP` was used as although `System.nanoTime()` is a high-resolution timer, it served as a general timer whereas `SENSOR_TIMESTAMP` is more accurate when referring to frame capture time. Therefore `SENSOR_TIMESTAMP` was used as only frames were being tracked.

Upon running the test, it was noticed that the average fps hovered at around 29fps after one minute of running. Since the expected fps was supposed to be 60fps as this is the default capture speed of the camera some extra tests were required to be run. A new method was created to calculate and log the live fps in real-time. This revealed that the vast majority of frames were being run at 30 fps with occasional dips to 15 fps. This presented two problems. The first being that 60 fps was not being reached and the second that the device was not able to run at 30 fps consistently.

In an attempt to remedy this, a new method was created called `configureCameraSettings` which attempted to set the target fps to 60. However this did not work, as running at 60 fps or higher required the use of a high-speed capture session as opposed to the regular capture session being used. The high-speed capture session was implemented however did not support the rendering of multiple feeds. For this reason, the aim of achieving 60 fps was abandoned.

Since the use of 60fps was abandoned, the focus turned to achieving a consistent 30 fps. First, logs were added to check for the amount of fps drops that occurred, as such a drop might have occurred on app startup while heavy processing was required. However, the app dropped below 30fps a total of 28 times in one minute, which although is a small amount compared to the 1,767 frames at 30fps in the same minute, it is still a significant enough number to have been happening outside of the startup period.

The next area to be checked was the strain on the CPU and RAM. The Android studio profiler was used to provide live tracking of the mobile's hardware strain. Code was also added to the average fps method that would track the lowest frame rate and log the amount of time this drop was detected since the start of the app. This could then be correlated with the Android studio profiler to check for spikes in CPU or RAM usage. This test was again run for the duration of one minute. It was noted that there was a steady rise in RAM usage as time went on with no reset, meaning that a function in the code required more memory the longer the app ran. The RAM was considered to be the problem since there was no obvious correlation with CPU strain as the CPU usage remained between 2-3%. This can be seen in Figure 13 and Figure 14. Figure 14 clearly shows a decrease in fps once the RAM increases. Also, it is clear that the RAM steadily increases as time goes on.

```

app x
W/System: A resource failed to call dispose.
D/ContentValues: Average FPS over the last minute: 29.62093655629862
D/ContentValues: Lowest FPS over the last minute: 15.003422505724894 at 12s since first capture

```

Figure 13 : Log stating time of lowest fps capture.

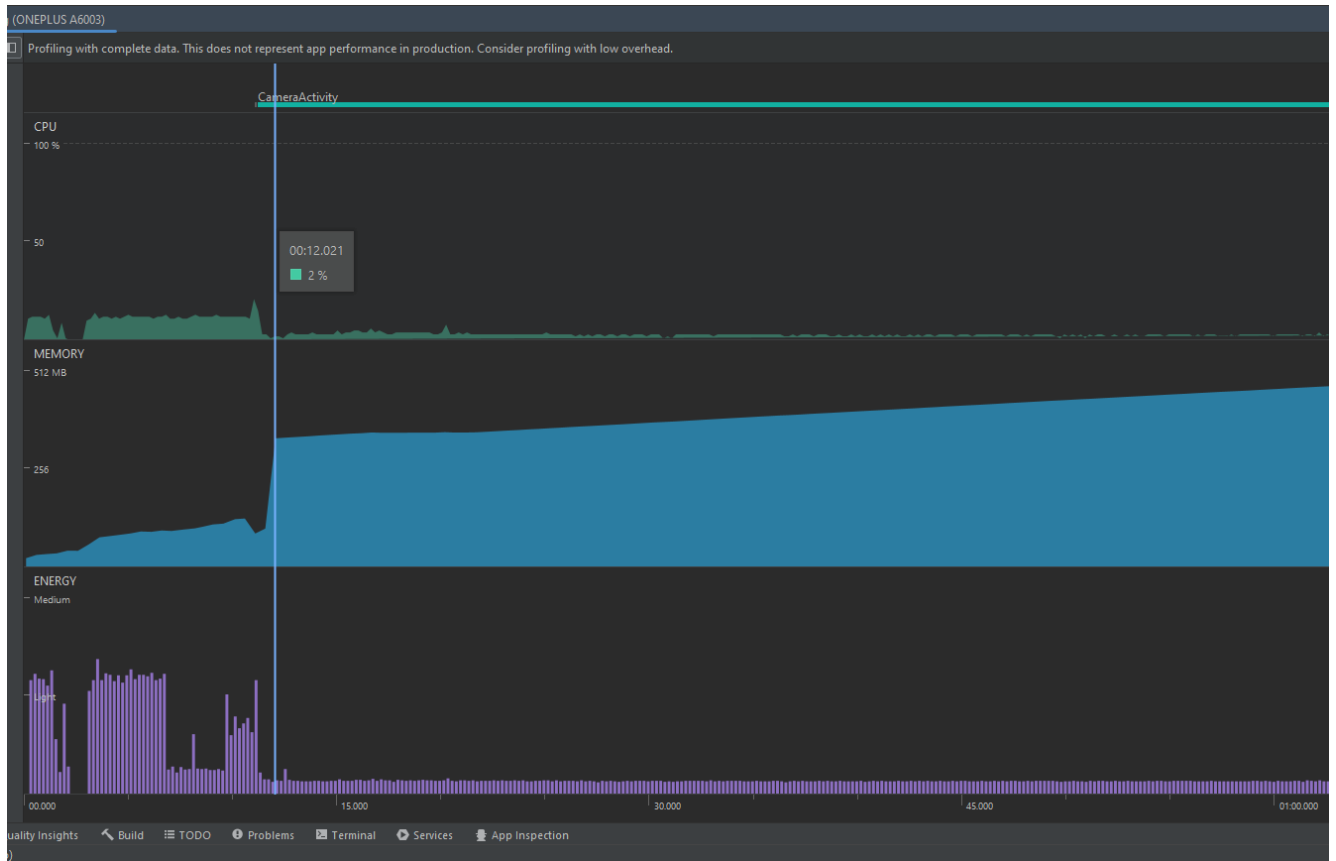


Figure 14 : Graph of CPU, RAM, and Energy usage over one minute. The blue vertical line indicates the 12 second mark.

To combat the RAM usage, the `configureTransform()` method was changed. Instead of initializing the matrix every call of `configureTransform()`, the matrix is defined only once. This helped to stabilize the RAM usage significantly, however fps drops were still prevalent. As can be seen in Figure 15 and Figure 16.

```

File Edit View Navigate Code Refactor Build Run Tools Git Window Help ARDriving - CameraActivity.kt [ARDriving.app.main]
src main java com example ardriving CameraActivity captureCallback <no name provided> onCaptureCompleted app ONEPLUS A6003
Run: app x
W/System: A resource failed to call dispose.
W/System: A resource failed to call release.
D/ContentValues: Average FPS over the last minute: 29.603378774908514
D/ContentValues: Lowest FPS over the last minute: 15.003422730827584 at 58s since first capture

```

Figure 15 : Log stating time of lowest fps capture.

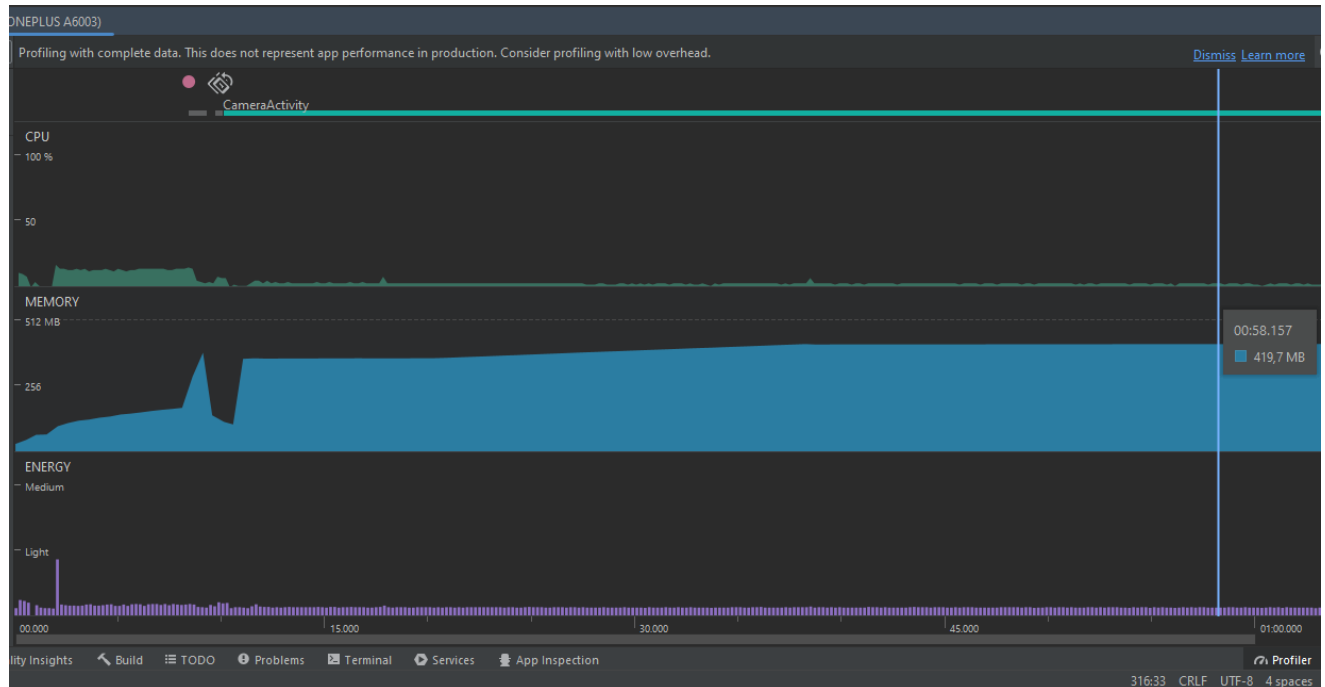


Figure 16 : Graph of CPU, RAM, and Energy usage over one minute. The blue vertical line indicates the 58 second mark.

Since the RAM usage and CPU usage did not seem to be causing the fps drop and the code is as streamlined as possible without modifying the native code, the last aspect to be checked is the GPU usage. This was done by using the Profile GPU rendering in Android's developer mode. This setting on the Android device displayed a real-time graph representing the amount of time taken to render a frame. This graph is displayed in Figure 17. As Android studio details in their developer documentation, the horizontal green line represents 16.67ms. To achieve 60fps the bar must stay below this line. Also, the bars themselves are colour coded. Since red and orange are the most prominent colours these are what the focus was on. Orange represents the time the CPU spent waiting for the GPU to finish work and the red represents the time spent by Android's 2D renderer to issue commands to OpenGL to draw and redraw lists.

With this in mind, we are able to see that a lot of time is spent for the CPU to wait for the GPU. In the below image the last spike exceeding the yellow line is ignored as this is the time at which the screenshot was taken, which resulted in an extreme use of resources. Since manipulation of the GPU is available only at the native level and therefore no more streamlining was possible, it was determined that, for the scope of this research, achieving a steady framerate of 30fps was not possible due to hardware limitations.

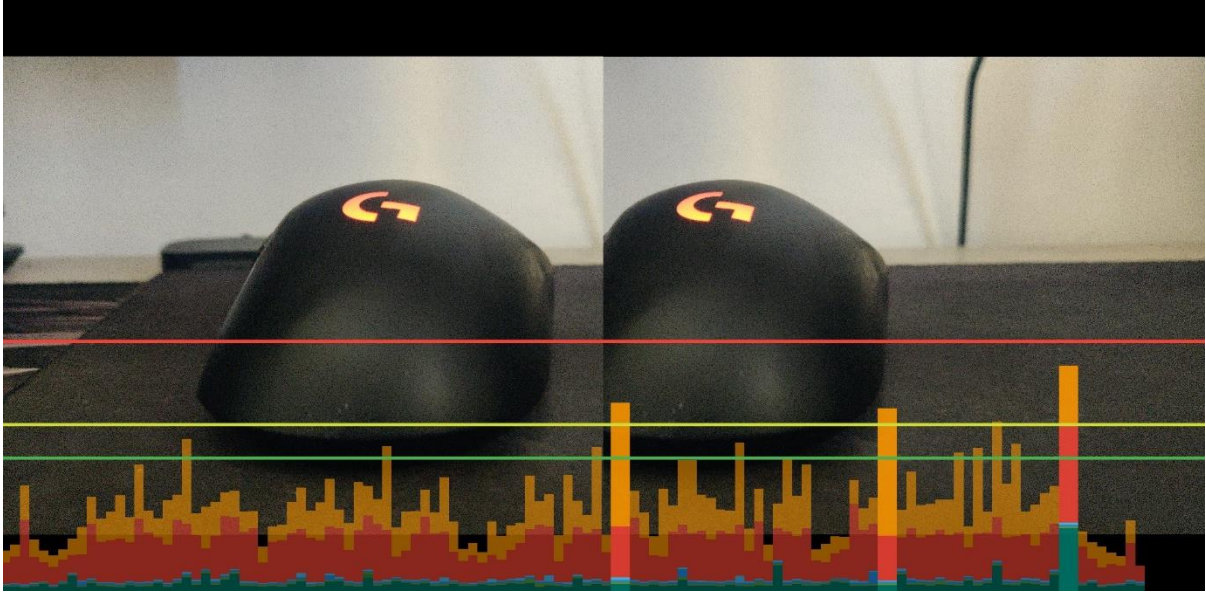


Figure 17 : Android GPU Profiler



## Chapter 5 - Evaluation

The following chapter discusses what was achieved in the current prototype and what areas were successful and what areas require further work. Suggestions for the issues encountered are provided in Chapter 6 -.

### 5.1 Stereoscopic Vision and Image Sharpness Results

Stereoscopic vision was achieved with a high level of sharpness. However, there are some considerations that were not taken into account that may affect user vision. Apart from this some aspects that were predicted to cause issues were not.

The mobile phone used contained a crack in the screen as shown in Figure 18. The initial assumption was that this crack would hinder the image quality completely, however this was not the case, although the crack was clearly visible since it was a hairline crack this had no significant impact on stereoscopic performance.



*Figure 18 : Showcase of crack in the prototype phone.*

One issue that was noted was that since the mobile phone camera is not centered with the user's face and is instead offset to the left, this caused an offset in the perception of where objects are. This caused a slight misjudgment in the positioning of objects and could easily be seen with the following test. A user would wear the headset and look straight ahead and adjust

only their head to look at a specific object in the distance. Once the headset was taken off it was noted that the user would be looking slightly away from the object. This can be easily visualized in Figure 19. In this diagram the blue line denotes the line of sight of the camera and the red light denotes the line of sight of the user as can be seen the perspective shift is enough to offset user perception.

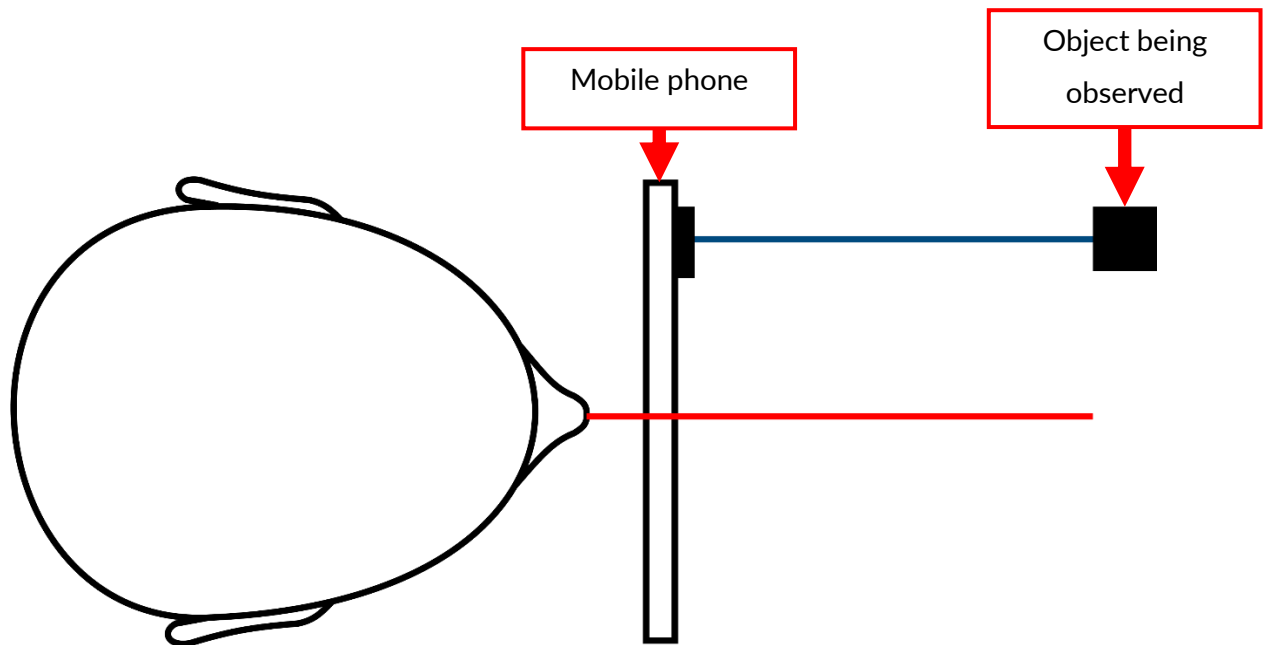


Figure 19 : Diagram showing perception offset.

Another issue that was encountered was static in the image once it was resized. This is due to absence of resizing algorithms that elegantly handle changes in resolution. These were purposefully omitted during this study in order to attempt to obtain faster execution times and therefore higher fps. However, for this technology to be viable, proper resizing algorithms must be implemented as the static resulting from the lack of these algorithms significantly impacts the image sharpness and introduces strain on the user's eyes.

The final issue was the FOV provided to the user. Since the mobile phone being used had no zoom out feature, this severely limited the FOV able to be provided and therefore certain objects seemed to be slightly closer than anticipated. However, this problem can be easily solved using more recent phones with a 0.5x zoom option.

## 5.2 Capture to Display Latency Results

In 2.5 it was detailed that this research aims to attempt to achieve less than 50ms of capture to display latency. As can be seen in the red box in Figure 20, after running the application for one minute the average display latency was 27ms. This is well under the required threshold and shows that the live processing of two video feeds is minimal.

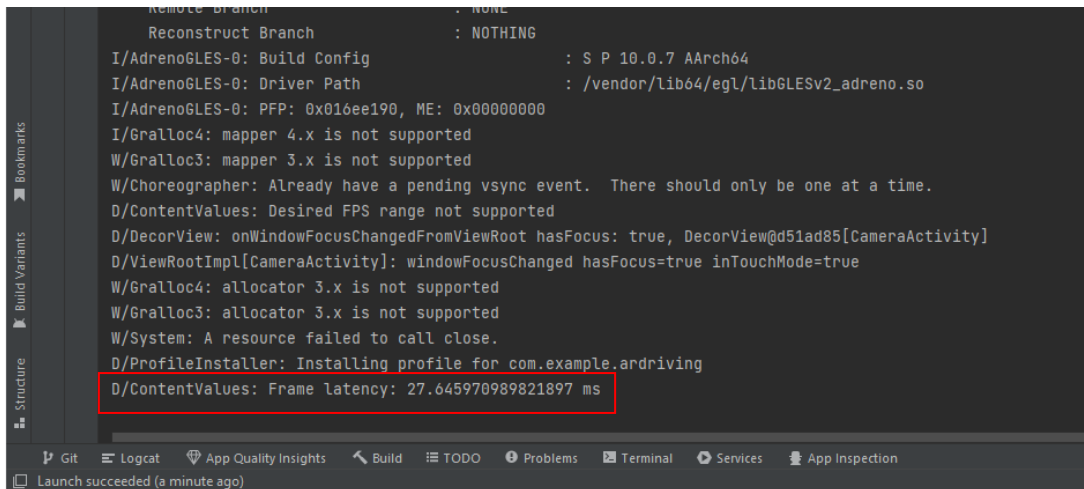
A screenshot of the Android Studio interface showing a log of system messages. The log is displayed in a dark-themed window with a sidebar on the left containing 'Bookmarks', 'Build Variants', and 'Structure'. The log text includes various system messages such as 'I/AdrenoGLES-0: Build Config', 'I/AdrenoGLES-0: Driver Path', 'I/AdrenoGLES-0: PFP: 0x016ee190, ME: 0x00000000', 'I/Gralloc4: mapper 4.x is not supported', 'W/Gralloc3: mapper 3.x is not supported', 'W/Choreographer: Already have a pending vsync event. There should only be one at a time.', 'D/ContentValues: Desired FPS range not supported', 'D/DecorView: onWindowFocusChangedFromViewRoot hasFocus: true, DecorView@d51ad85[CameraActivity]', 'D/ViewRootImpl[CameraActivity]: windowFocusChanged hasFocus=true inTouchMode=true', 'W/Gralloc4: allocator 3.x is not supported', 'W/Gralloc3: allocator 3.x is not supported', 'W/System: A resource failed to call close.', 'D/ProfileInstaller: Installing profile for com.example.arDriving', and 'D/ContentValues: Frame latency: 27.645970989821897 ms'. The last line is highlighted with a red rectangular box. At the bottom of the window, a status bar shows 'Launch succeeded (a minute ago)'.

Figure 20 : Frame latency log.

## 5.3 Optimal fps Results and Video Tearing

Due to the issues detailed in 4.2.2 the optimal fps was not able to be reached. The maximum fps reached was 30 fps with dips to 15 fps as shown in the red box in Figure 21. The failure to obtain higher fps effected two aspects of the app.

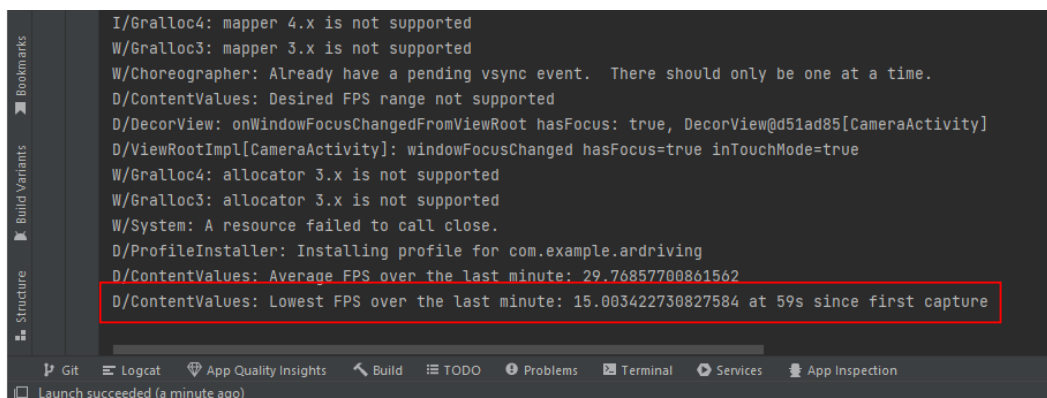
A screenshot of the Android Studio interface showing a log of system messages. The log is displayed in a dark-themed window with a sidebar on the left containing 'Bookmarks', 'Build Variants', and 'Structure'. The log text includes various system messages such as 'I/Gralloc4: mapper 4.x is not supported', 'W/Gralloc3: mapper 3.x is not supported', 'W/Choreographer: Already have a pending vsync event. There should only be one at a time.', 'D/ContentValues: Desired FPS range not supported', 'D/DecorView: onWindowFocusChangedFromViewRoot hasFocus: true, DecorView@d51ad85[CameraActivity]', 'D/ViewRootImpl[CameraActivity]: windowFocusChanged hasFocus=true inTouchMode=true', 'W/Gralloc4: allocator 3.x is not supported', 'W/Gralloc3: allocator 3.x is not supported', 'W/System: A resource failed to call close.', 'D/ProfileInstaller: Installing profile for com.example.arDriving', 'D/ContentValues: Average FPS over the last minute: 29.76857700861562', and 'D/ContentValues: Lowest FPS over the last minute: 15.003422730827584 at 59s since first capture'. The last two lines are highlighted with a red rectangular box. At the bottom of the window, a status bar shows 'Launch succeeded (a minute ago)'.

Figure 21 : Average fps log.

The first is that the minimum frame rate of 120 fps defined in 2.4 was not reached. Since the framerate was capped at a maximum of 30 fps, this caused the image to be choppy while moving and would make this prototype impossible to use while driving.

Secondly, since the fps was not able to be matched to the screen's refresh rate of 60Hz, screen tearing was evident as can be seen in Figure 22 and testing of screen tearing with a higher fps than refresh rate was no able to be tested.



*Figure 22 : Screenshot of screen tearing.*

Overall, this area requires significant improvement and possibly even better hardware as a big limitation was the overall power of the GPU. As can be seen in Figure 14 and Figure 17, while only 2% of the CPU was being used, the CPU was still waiting for the GPU to process and display the live images.

## Chapter 6 - Future Work

This section includes a few potential areas that may be undertaken to improve this concept and build on the research conducted.

- Further research could be done on the effects of slight screen inconsistencies such as cracks, dust clumps etc. Such research could establish a threshold needed for a device to be capable of convey stereoscopic images.
- The use of further cropping and video manipulation could be explored to solve the issue of the camera offset, ensuring the image appears centered to users and provides more precise perception of where objects are located.
- Different resizing methods and algorithms may be attempted to find out which algorithm would work best to avoid image static and provide clear resolution to resized images while keeping the processing costs as minimal as possible.
- This research may also be conducted using more modern phones to test the impact more powerful hardware has on the results of this research. Also, the use of 0.5x zoom cameras may test and explore if the optimal FOV may be reached to mimic a human's FOV as closely as possible.
- Although the capture to display latency was extremely low, research may still be needed ensure that such a latency is minimized as much as possible to provide as smooth an experience as possible to users.
- The largest area for further research is to apply this method while achieving higher frame rates of up to 120fps and above without applying too much strain to the hardware to allow for extra features to be added on top of the stereoscopic display.

## Chapter 7 - Conclusions

This project aimed to achieve a stereoscopic vision with minimum lag and delay. This was partly achieved. Stereoscopic vision was achieved with incredible results, however the video feed suffered from lackluster fps and suffered due to image static.

Stereoscopic vision was achieved by cropping the video feed from a single camera by half the length of the average pupillary distance and displaying the two feeds side by side through the use of a cheap headset. The use of a single camera enabled this solution to be used on a mobile device that could be used with the addition of only a cheap headset. This solution also enabled users with glasses to use AR as the lenses inside the headset could be directed into the pupil as the user required.

The research also covered various tests to quantify the efficiency of the solution. These tests helped to facilitate problem identification which provides a path for future researchers to understand the problems that are required to be overcome for AR to be viable using mobile phones.

The visual tests revealed that the solution requires appropriate resizing algorithms, and that camera positioning and human FOV must be taken into account. Appropriate resizing algorithms are required to scale images efficiently and elegantly without introducing image static that can cause strain to a user's eyes. Camera positioning must be looked at as an offset camera interferes with the user's perception of where an object is. If the camera is not centered on the user's face, then both video feeds must be shifted accordingly to correct a user's perception. Finally, the amount of the image being shown must match the user's FOV as a higher FOV than normal results in objects seeming further than they are and a lower FOV results in objects appearing closer than in reality.

The hardware tests revealed that the solution must be either improved at the native level or that faster hardware is required to enable the solution to run smoothly. This is highlighted by the fact that the capture to display latency was relatively low, meaning that the rendering of the image is what caused the most strain on the device.

## 7.1 Final Remarks

Despite being able to achieve high frame rates, stereoscopic vision was achieved. This means that this research may serve as a proof of concept that the idea of the solution works. With further research into this field, there is a possibility of making AR accessible to everyone with the addition of minimal costs to users.

# References

- [1] E. Pateras, 'Stereoscopic Vision', p. 2020, Apr. 2020.
- [2] OCVT, 'How Does Stereoscopic (3D) Vision Work?' Accessed: Mar. 04, 2024. [Online]. Available: <https://ocvt.info/how-does-stereoscopic-3d-vision-work/>
- [3] 'U Drive. U Text. U Pay.', NHTSA. Accessed: Feb. 27, 2024. [Online]. Available: <https://www.nhtsa.gov/campaign/distracted-driving>
- [4] 'Study Reveals Where Drivers Are Most Reliant on Their GPS', unitedtires. Accessed: Feb. 27, 2024. [Online]. Available: <https://www.utires.com/articles/where-drivers-need-gps-the-most/>
- [5] 'Distracted Driving', CDC. Accessed: Feb. 27, 2024. [Online]. Available: [https://www.cdc.gov/transportationsafety/distracted\\_driving/index.html#factsheet](https://www.cdc.gov/transportationsafety/distracted_driving/index.html#factsheet)
- [6] 'Forza Horizon'. Accessed: Mar. 08, 2024. [Online]. Available: <https://forza.net/horizon>
- [7] Sim Racing Corner, 'How To Enable/Disable Driving Line - Forza Horizon 5.', Youtube. Accessed: Feb. 27, 2024. [Online]. Available: <https://www.youtube.com/watch?v=UZ7-WPZkApc>
- [8] 'Pilot Report: 737-Next Gen Heads Up Display.', JetHead. Accessed: Feb. 27, 2024. [Online]. Available: <https://jethead.wordpress.com/2013/03/05/pilot-report-737-next-gen-heads-up-display/>
- [9] Apple, 'Buy Apple Vision Pro'. Accessed: Feb. 29, 2024. [Online]. Available: <https://www.apple.com/shop/buy-vision/apple-vision-pro>
- [10] 'This is Meta Quest Pro', Meta. Accessed: Feb. 29, 2024. [Online]. Available: <https://www.meta.com/quest/quest-pro/>
- [11] 'Another Slow Year Expected for AR/VR Headsets Before 2024 Rebound, According to IDC', IDC. Accessed: Feb. 29, 2024. [Online]. Available: <https://idc.com/getdoc.jsp?containerId=prUS51252823>



- [12] 'Global smartphone shipments from 2009 to 2023', statista. Accessed: Feb. 29, 2024. [Online]. Available: <https://www.statista.com/statistics/271491/worldwide-shipments-of-smartphones-since-2009/#:~:text=In%202023%2C%20the%20total%20amount,reached%201.17%20billion%20units%20worldwide.&text=The%20global%20smartphone%20market%20saw,shipments%20amounted%20to%201.47%20billion>.
- [13] Josh Howarth, 'How Many People Own Smartphones? (2024-2029)', Exploding Topics. Accessed: Feb. 29, 2024. [Online]. Available: <https://explodingtopics.com/blog/smartphone-stats>
- [14] 'Vizor Pro', GGG. Accessed: Mar. 04, 2024. [Online]. Available: <https://gadgetsiftgames.com/product/vizor-pro-virtual-reality-vr-3d-headset/>
- [15] X. Kang et al., 'Stereoscopic augmented reality for laparoscopic surgery', *Surg Endosc*, vol. 28, no. 7, pp. 2227–2235, Jul. 2014, doi: 10.1007/s00464-014-3433-x.
- [16] VISIONSENSE LTD, 'VSU - VISIONSENSE STEREOSCOPIC VISION SYSTEM', 2008
- [17] 'Kotlin'. Accessed: Apr. 21, 2024. [Online]. Available: <https://kotlinlang.org/docs/home.html>
- [18] 'CameraX', Accessed: Apr. 21, 2024. [Online]. Available: <https://developer.Android.com/media/camera/camerax>
- [19] 'Camera2'. Accessed: Apr. 21, 2024. [Online]. Available: <https://developer.Android.com/media/camera/camera2>
- [20] 'React Native'. Accessed: Apr. 21, 2024. [Online]. Available: <https://reactnative.dev/docs/getting-started>
- [21] 'VisionCamera'. Accessed: Apr. 21, 2024. [Online]. Available: <https://react-native-vision-camera.com/docs/guides>
- [22] 'Expo'. Accessed: Apr. 21, 2024. [Online]. Available: <https://docs.expo.dev/versions/latest/sdk/camera/>
- [23] 'Glasses.com'. Accessed: Apr. 21, 2024. [Online]. Available: <https://www.glasses.com/gl-us/blog/pupillary-distance-how-to-measure>

- [24] 'Zenni', Accessed: Apr. 21, 2024. [Online]. Available: <https://www.glasses.com/gl-us/blog/pupillary-distance-how-to-measure>
- [25] 'All About Vision'. Accessed: Apr. 21, 2024. [Online]. Available: <https://www.allaboutvision.com/eye-care/measure-pupillary-distance/#:~:text=PD%20is%20measured%20in%20millimeters,and%2077%20mm%20for%20men.>
- [26] C. C. ; B. C. L. ; B. B. P. J. L. ; B. P. P. S. P. ; C. B. D. ; C. J. M. ; V. J. C. ; R. B. M. ; M. M. K. S. Gordon, '2012 Anthropometric Survey of U.S. Army Personnel: Methods and Summary Statistics', 2012.
- [27] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, and H.-N. Liang, 'Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality', *IEEE Trans Vis Comput Graph*, vol. 29, no. 5, pp. 2478–2488, May 2023, doi: 10.1109/TVCG.2023.3247057.
- [28] WorldData.info, 'Speed limits worldwide'. Accessed: Apr. 23, 2024. [Online]. Available: <https://www.worlddata.info/speed-limit.php>
- [29] 'Wondering Maps'. Accessed: Apr. 23, 2024. [Online]. Available: <https://wonderingmaps.com/worlds-speed-limits/>
- [30] Y. Kuroki, 'Improvement of 3D visual image quality by using high frame rate', *J Soc Inf Disp*, vol. 20, no. 10, pp. 566–574, Oct. 2012, doi: 10.1002/jsid.107.
- [31] M. Walter, T. Wendisch, and K. Bengler, 'In the Right Place at the Right Time? A View at Latency and Its Implications for Automotive Augmented Reality Head-Up Displays', 2019, pp. 353–358. doi: 10.1007/978-3-319-96074-6\_38.
- [32] Vanessa Ezekowitz, 'File:Tearing (Simulated).jpg'. Accessed: Apr. 29, 2024. [Online]. Available: [https://commons.wikimedia.org/wiki/File:Tearing\\_\(simulated\).jpg](https://commons.wikimedia.org/wiki/File:Tearing_(simulated).jpg)