EXPLORATION OF THE 3D WORLD ON THE INTERNET USING COMMODITY VIRTUAL REALITY DEVICES

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER AND INFORMATION SCIENCES

Supervisor

Dr. Minh Nguyen

Dr. Wei Qi Yan

February 2018

By

Huy Tran

School of Engineering, Computer and Mathematical Sciences

Abstract

This thesis describes the development of a graphically interactive and online Virtual Reality (VR) application. Based on keywords provided by users, it automatically retrieves and display stereoscopic contents from the Internet. The system also includes a state-of-the-art feature matching algorithm which can filter stereoscopic contents from "normal" 2D contents.

With the main goal of delivering an affordable way of viewing 3D VR contents, the application is designed to be specifically compatible with the low-cost smartphone VR platforms such as Google Cardboard, Google Daydream, and Samsung Gear VR. The experiment results show that the current application prototype is portable, easy-to-use, and effective in retrieving and displaying stereoscopic contents. With this application, users all over the world could easily experience millions of stereoscopic contents on the Internet. It also has a huge potential of becoming a great tool for both VR testing and learning purposes.

Contents

List of Tables

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

[illegible]

Signature of student

Publications

Exploration of the 3D World on the Internet Using Commodity Virtual Reality Devices	(Minh Nguyen, Huy Tran, Huy Le), In Multimodal Technologies and Interaction, Multidisciplinary Digital Publishing Institute, volume 1, 2017.
A Personalised Stereoscopic 3D Gallery with Virtual Reality Technology on Smartphone	(Huy Tran, Minh Nguyen, Huy Le, Wei Qi Yan), In Int. Conf. Image Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4 Dec - 6 Dec, 2017.
A Tile Based Colour Picture with Hidden QR Code for Augmented Reality and Beyond	(Minh Nguyen, Huy Tran, Huy Le, Wei Qi Yan), In 23rd ACM Symposium on Virtual Reality Software and Technology (VRST), Gothenburg, Sweden, 2017.
Adaptive Stereo Vision System using Portable Low-cost 3D Mini Camera Lens	(Minh Nguyen, Huy Le, Huy Tran, Wai Yeap), In IEEE Int. Conf. Mechatronics Machine Vision Practice (M2VIP), Auckland, New Zealand, 21-23 Nov, 2017.
Digital Map using Augmented Reality on Smart Devices: Motivation, Design, and Implementation	(Lei Qiu, Minh Nguyen, Huy Le, Huy Tran, Wei Qi Yan), In Int. Conf. Image Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4 Dec - 6 Dec, 2017.
Pictorial AR Tag with Hidden Multi-Level Bar-Code and Its Potential Applications	(Huy Le, Minh Nguyen, Huy Tran, Wai Yeap), In Multimodal Technologies and Interaction, Multidisciplinary Digital Publishing Institute, volume 1, 2017.

Acknowledgements

The completion of this project could not have been possible without the assistance of many people whose names may not all be accounted for. Their contributions to mankind's library of knowledge are greatly appreciated. However, the researcher would like to send his deep appreciation to these particulars:

- Supervisor Dr. Minh Nguyen for always willing to provide great guidance to the researcher in developing the project, performing experiments, and conducting the report;
- Supervisor Dr. Wei Qi Yan and Mr. Huy Le for giving the researcher great feedbacks on the research;
- The researcher's family and relatives for always being there for the researcher, and giving him support to overcome multiple physical and emotional challenges;
- The researcher's good friends for giving him countless of memorable and enjoyable moments.

Chapter 1

Introduction

Since the information revolution started in the 1990s, information technology has been utilised in several different areas of human lives. With the advanced technologies, especially the World Wide Web and computing hardware, people can achieve many things that were only the dreams of their ancestors in the past such as: searching for books without going to the library, experiencing through the rain forest while staying at home, or checking in at the hotel with the bar-code scanning technique.

1.1 The power of smartphones

Besides the World Wide Web, mobile devices have also developed significantly in the last two decades. They not only become smaller, but also contain many "smart" features such as Wi-Fi, hidden cellular antenna, touchable screen, high-resolution display. Most of the smartphones today can perform many advanced tasks like efficient localisation data acquisition, outstanding visualisation, or object tracking. Because of this, they have become "could not live without" devices for many people. There were almost 2 billion mobile device users worldwide compared to less than 1.8 billion of desktop computer users in 2015 (Chaffey, 2016). Several different operating systems have been designed for smartphones; such as Android, iOS, and Windows. Android, an open-source Linux-based operating system(Nimodia & Deshmukh, 2012), is the most popular one, claiming 81.5% of the global smartphone operating system market in 2014 (Hahn, 2015). There were 1.8 million applications available on Google Play (Dogtiev, 2016) with millions of downloads. At the current time, thousands of new applications are being published online every month.

1.2 The potential of Virtual Reality

In the recent decades, Virtual Reality (VR) technology has developed and reached to a remarkably level in ICT industry. Today, there are many commodity VR systems. They employ the depth perception and binocular vision of human to generate various interesting applications. Underneath a general VR system, there is simply one flat LCD screen (e.g. of a smartphone) that projects two stereo images to viewer's left and right eyes synchronously and simultaneously.

These devices allow people to visualise the 3D scenes without getting out of the offices; for instance, people can safely and comfortably explore the tropical forests, swim in the deep oceans, or fly in the sky. This technology is not only ideal for entertainment, but also for education and training. While there are many VR applications available on the market, most of them only focus on the entertainment aspect. Moreover, their datasets are limited to just few pictures, videos, and virtual scenes in 2D. This is due to the limitation of data acquisition, storages, and copyrights.

There are billions of images publicly available on the Internet (e.g., the Google Image Search database). Just a tiny percentage of them are 3D visible by VR (e.g., images that contain left and right stereo parts); however, these small bits still count millions. These publicly available datasets are significantly larger than any others provided by existing tools.

1.3 Research goal

With this background in mind, the main goal of this research is to create an application that utilise both smartphones and VR to provide a quality 3D viewing experience. The application is responsible for two main roles: (1) it can retrieve stereoscopic contents from the Internet, and (2) it can display stereoscopic contents.

For the first role, the researcher will create an image classification algorithm that filter stereoscopic contents from the “messy” data of the Internet. Then the filtered contents will be converted into a VR viewable format. While the preferable approach is to add these features into the local system of the application, it is still unclear if they can work properly with the limited hardware of the smartphones.

The next role is to create a viewer that can display stereoscopic contents correctly. The application is required to include an easy-to-use interface in a 3D virtual environment. By simply wearing the virtual reality headset, users can select and view the 3D effects of all stereoscopic contents retrieved from the Internet.

Chapter 2

Project Approach

Since this is an application project, a suitable software development methodology is used to ensure that the development could keep up with the project schedule. With only one researcher working on the project, it is impossible to develop all features of the application simultaneously. Moreover, there are close connections between these features. One broken feature can cause the others fail to work properly. With this context, a customised Waterfall Model is used for the development methodology, which can be seen in Figure 2.1.

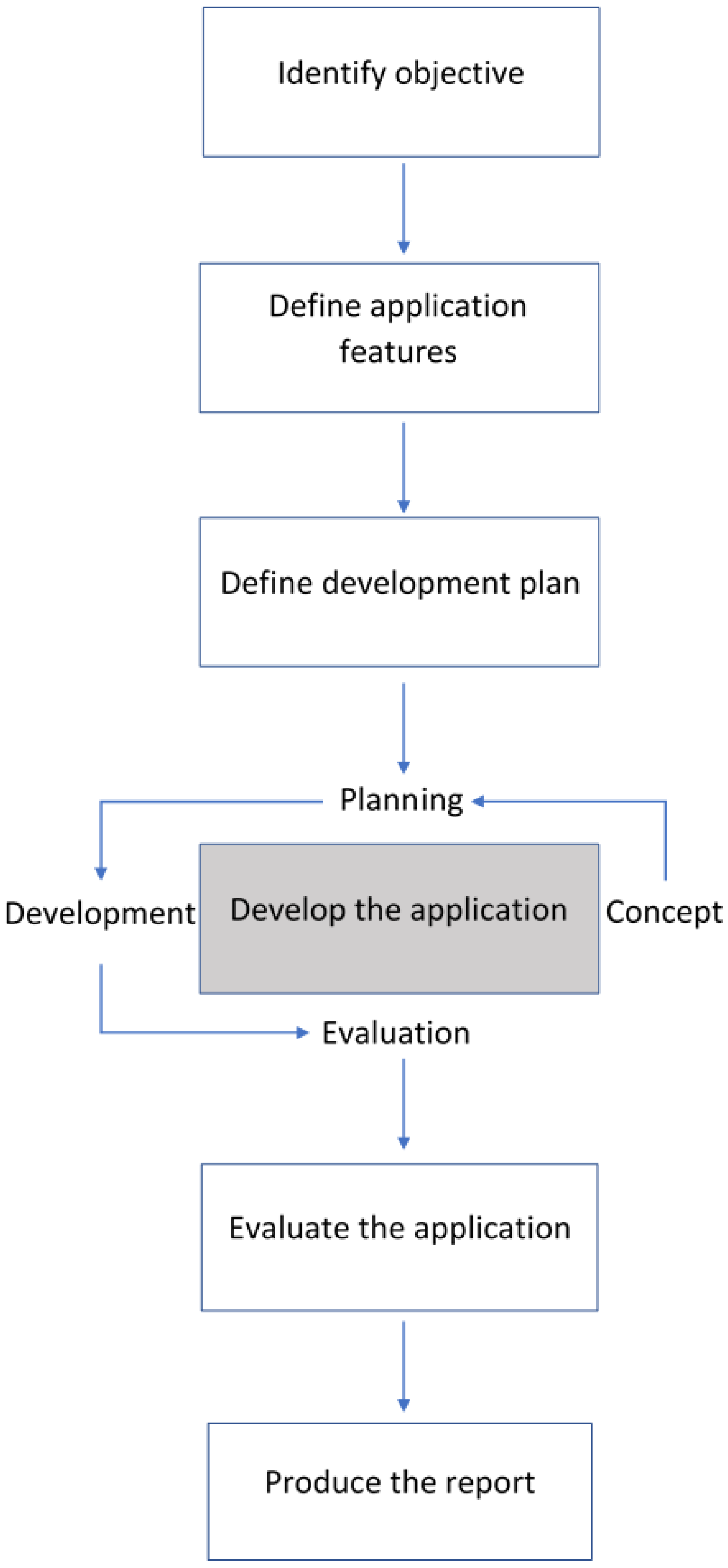


Figure 2.1: A diagram that describes the project approach

2.1 Define application features

In this initiation stage, all application features are highlighted. These features are necessary for the project goal fulfillment. Each of them also has a small-scale function analysis and previous work research. These previous works are related to the main features of the application, which are 3D stereoscopy and Virtual Reality (VR).

The purpose of this analysis is to make sure that the application will be developed through all of the optimised developing techniques that the previous work researchers use. These techniques also help the application avoid common problems in 3D stereoscopy and VR; such as virtual reality sickness, limited future contents. If a feature deems difficult to be developed in the time frame, an alternative solution will be needed, which can be either removing it or postponing it to be the last feature that is developed.

Although this application has been limited to be a mobile application, it is still important to decide which mobile platform, graphic engine and Integrated development environment (IDE) would be the most suitable for the development. Because of this, popular engines and IDEs will be compared so that the best development solution can be chosen. This is also the time when cost, resource, and development time are identified. Based on the highlighted features, these three factors can be finalised to be used for the next stage.

2.2 Define development plan

After the end of the previous stage, all gathered information such as previous works, resource, development time are used for the creation of the project schedule. Because all of the application features have been analysed, they can be sorted in the development order of mandatory and difficulty.

The chosen software development tool is also analysed in this stage. This is to decide if this tool needs any add-on and which version will be used throughout the development.

Moreover, the researcher also conducts a list of required hardware to be used for the development and evaluation stage. Each hardware has its purpose explained and cost highlighted to avoid the waste of resource and cost of the project.

2.3 Develop the application

In contrast with other stages, this is the most time and resource consuming stage. The stage also has a different structure which is designed to make sure all features would be developed successfully. Even though these features are not developed simultaneously, the core structure of Agile Development Model is still the main design of this stage. Based on the order of features to be developed in the project schedule, each of them starts with the concept, planning, development, and evaluation stage. Figure 2.2 shows how agile methodology works in this stage.

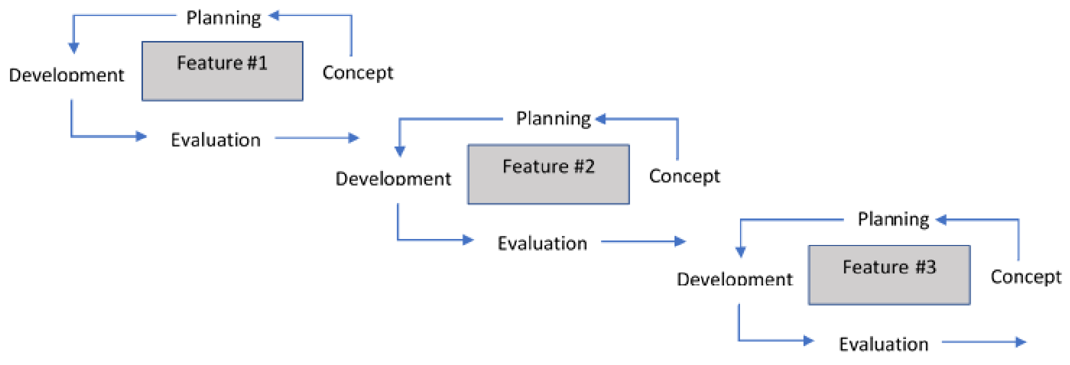


Figure 2.2: A diagram that describes the agile methodology usage in the development stage

After the function objective is analysed, an optimised approach will be used to develop the feature. When a feature development is finished, a small-scale evaluation will be conducted with the help of the researcher's colleagues as test users. If there are still some issues with the current feature, solving them will be prioritised. Only when the users are satisfied with the current stage of the feature, the development will move to the next feature with four agile methodology stages started again.

The development stage only completes when all feature developments are finished with good satisfaction from the users. They will be combined into a whole project.

2.4 Evaluate the application

Although all features have satisfied the users in the previous stage, they may have issues when implemented into a single application. The final stage of the application is needed to go through a simple performance test. The test results will be used to help determine if the application works properly.

In case there are issues found during the test, these issues will be analysed for their criticality, and documented in the evaluation checklist.

2.5 Produce the report

When the application can run without any critical issue, a report is conducted to highlight all research activities the researcher has performed. Any issue occurring during the development stage is also documented with a solution. For the application evaluation, the performance test methodology and feedback result are recorded. Feedbacks gathered throughout the application evaluation are reviewed to identify problems that are not resolved yet, as well as feature suggestions; this information is used for section **8.1 Future work**. Depend on the usefulness of the application in education and entertainment, the application source code will be either submitted to the public or privately secured for further development.

Chapter 3

Literature Review

3.1 What is Virtual Reality?

Virtual reality (VR) is a set of technology which is used to create a computer-generated virtual environment where users can experience and interact just as if they would do in real life (Emspak, 2016). It consists of four main components: Stereoscopic Displays, Motion Tracking Hardware, Input Devices, and Software Platforms (Parisi, 2015). The stereoscopic displays are generally designed as head-mounted displays for the users. However, in some VR sets such as Cave Automatic Virtual Environment, these displays are in combination with physical areas to create multi-displays environments surrounding the users (Ng, Chan & Lau, 2016).

3.1.1 Background of Virtual Reality

VR has been around since the early 20th century, when a science fiction short story Pygmalion's Spectacles, written by Stanley G. Weinbaum, described a pair of goggles which allowed users to interact with characters, objects in a movie (Sural, 2017). The concept sparked many research and development to adapt VR into entertainment, flight simulation, medical training, military training (Jerald, 2015).

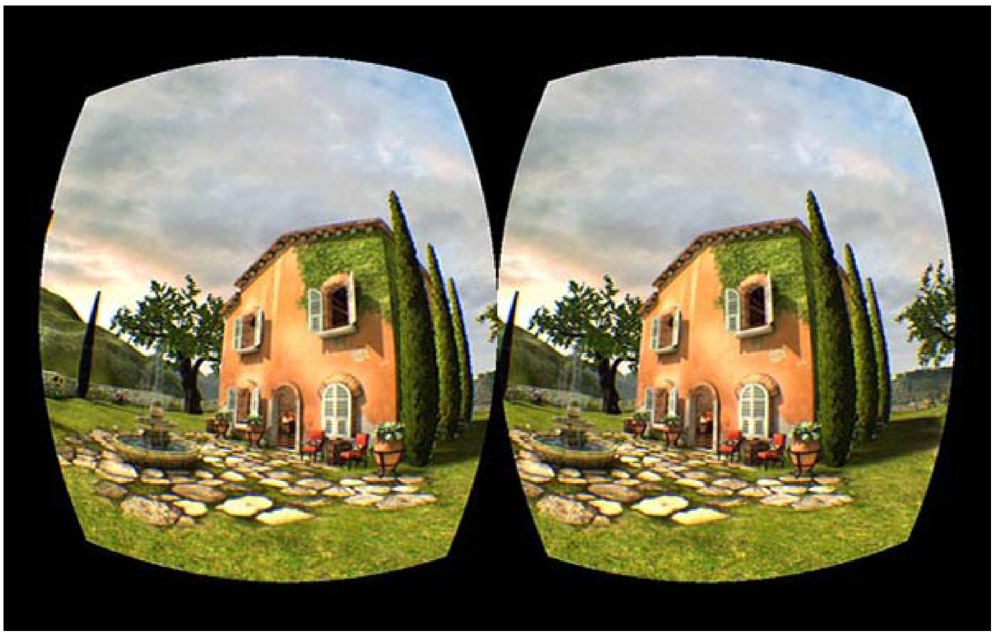


Figure 3.1: A typical VR Screen

While there were many successful applications for VR, the limitation of technology prevented it from becoming widely available for consumers. It was not until the 2010s that VR has become a system that people can set up in their home, which focuses on the use of VR head-mounted displays.

Currently, there are two main types of VR headsets which are available on the market: (1) the dedicated-hardware type and (2) the smartphone-based type. On the first hand, the dedicated-hardware headset type, such as Oculus Rift, HTC Vive, delivers superior VR experience with minimal compromise in image quality. This is thanks to the customised hardware built into the head-mounted display, and controllers with custom designs. However, the premium price-tag makes this type of VR only attract VR enthusiasts.

On the other hand, the smartphone-based type has the advantage of making use of most of the smartphone hardware on the market. The display, processor, and sensor in the smartphone are used as components for the platform's head-mounted display. With this characteristic, smartphone-based VR minimises the price-tag to only a couple of dollars. One of the most popular examples is Google Cardboard introduced in 2014, which has been shipped over 10 million units in March 2017 (Jonnalagadda, 2017). By using a cardboard box as a head-mounted holder for the smartphone display, users can experience VR through their smartphones.

3.2 Available smartphone-based Virtual Reality devices

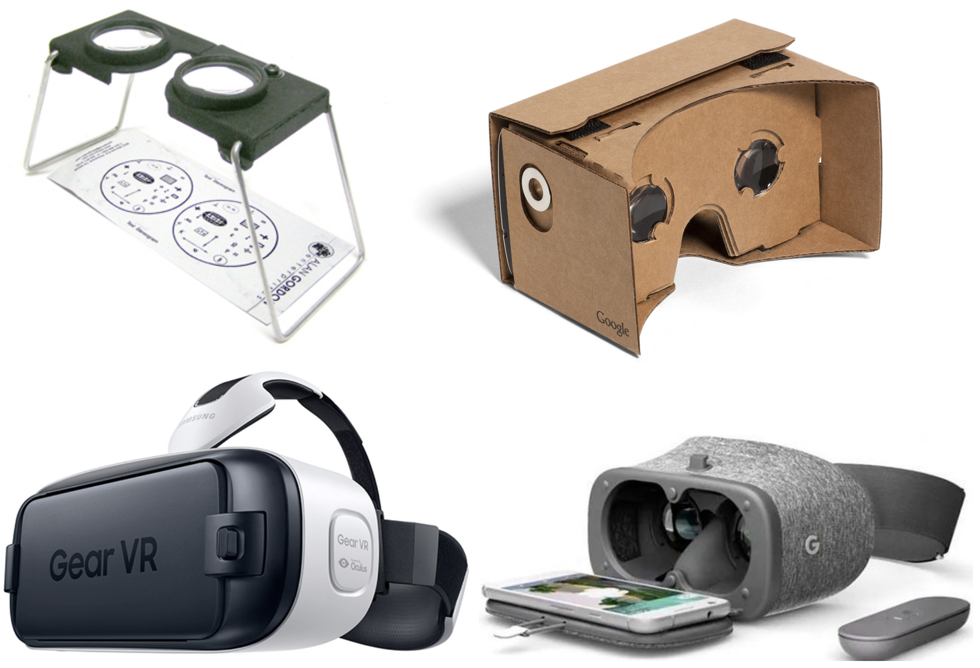


Figure 3.2: A portable pocket stereoscope, a low-cost Google Cardboard, a Samsung Gear VR, and a premium Google Day Dream
Acquired from (Bavor, 2016) (Samsung Electronics, n.d.) (Wee, 2016)

3.2.1 Google Cardboard

According to (Google, 2015a), Google Cardboard headset (Figure 3.2–top-right) consists of three main parts. The first part is a pair of lenses which have a diameter of 34 mm. These lenses are designed to achieve 80-degrees field of view, which is narrower than the lenses from dedicated hardware VR such as HTC Vive (110-degrees field of view) (Oscillada, 2015). Because of this, it is expected that users may experience "goggle effect", the effect that makes the Google Cardboard headset case still visible to the users' visual during the VR experience (Machkovech, 2016).

The second part is a mechanical body made of corrugated cardboard. This body is a combination of three separated cardboards, namely "chassis", "t-shirt", and "button". To make the headset lighter and easy to handle, Google requires the thickness of the cardboard material to be approximately 1.7mm.

Moving on to the final part, a button for Google Cardboard. Google has replaced the "magnetic switch" button introduced in 2014 to a capacitive button, which consists of two conductive parts named as "pillow" and "strip"; both parts are made from metalised fabric (polyester Ni/Cu), a type of material that can trigger the capacitive sensing system when it touches the smartphone screen. Moreover, Google also uses Velcro and rubber band to assemble Google Cardboard headset securely.

One of the main limitation of Google Cardboard is its control mechanism; this smartphone-based VR platform only has one single button to navigate through VR contents. In the first generation of Google Cardboard, the button is designed as a magnetic switch attached to the side of the cardboard. The newest second generation of the platform still keeps the design of one single navigating button. However, the magnetic switch is replaced with a capacitive button, which is more durable.

Wendy Powell (Wendy Powell, 2016) conducted a performance test to determine if the Google Cardboard control is efficient to navigate smoothly through the 3D environment. They created a virtual environment through Unity game engine and asked the testers to perform some straightforward and complex navigation through it. There were three control methods in this test, which were continuous motion, magnetic switch, and Bluetooth controller. The continuous motion was the most disliked method, due to the lack of control and the difficulty of turning while moving forward. The magnetic switch had better feedbacks from the testers, but it did not respond accurately. The

Bluetooth controller was the most favoured method, thanks to its direct control of forwarding and backwards travel. However, this is not a part of Google Cardboard and may have some compatibility issues with some smartphones. Since the headset material is low-cost and widely available on the market, Google Cardboard is the most affordable platform for consumers.

3.2.2 Google Daydream

In 2016, Google has introduced a successor of Google Cardboard named Google Daydream (Figure 3.2–bottom-right). This new platform contains many improvements from the predecessor such as a new controller, a better build material, and a new requirement for the smartphone hardware (Faulkner & phones, 2017).

To be able to experience VR through Google Daydream, users are required to have three main components: (1) A Daydream View headset, (2) a Daydream controller, and (3) a Daydream-ready phone. The headset is made mainly of thermos-bonded cloth, which is an improvement to Google Cardboard. With this new material, the headset is lighter and more comfortable than other similar headsets on the market. Its phone compartment contains six capacitive nubs which are used as directives for the phone automatic image alignment system (Amadeo, 2016).

Google Daydream also addresses the lack of a controller present in Google Cardboard. In this new VR headset, the company also develops a small controller consisting of a clickable touchpad, two physical buttons, a gyroscope, an accelerometer, and a magnetometer (Novet, 2016).

Based on the review of (Orland, 2016), unlike Google Cardboard, the control system of Google Daydream is versatile. Its functionality varies from a virtual laser pointer, virtual steering wheels, to a motion detector. The device is very similar to Nintendo Wii Remote, a controller from Wii game console. Google Daydream also requires smartphones to meet certain specifications to operate. With the addition of a new controller and a requirement of certain smartphones with a powerful processor, Google Daydream can deliver better VR experience when compared to Google Cardboard. However, the platform still has several issues like the predecessor, which are narrow field of views, glare on display due to light from the outside of the VR headset case.

Currently, there are only four compatible phones: (1) Google Pixel, (2) Motorola Moto Z, (3) Huawei Mate 9 Pro, and (4) ZTE Axon 7. They are pricing from 400 USD to 869 USD. The headset itself also costs 79 USD (Google, 2017b). With this fee barrier, it is tough for this device to be available as widely as Google Cardboard.

3.2.3 Samsung Gear VR

Samsung Gear VR (Figure 3.2–bottom-left) is a smartphone-based VR platform designed exclusively for Samsung Galaxy S and Samsung Galaxy Note product lines from 2015 onward. The product is a collaboration between Samsung and Oculus VR; the current generation, Gear VR 2017, is made mainly of plastic. Unlike Google VR platform, the Gear VR headset is equipped with a gyroscope, an accelerometer, a proximity sensor, and a head tracking system, which ensures that users can achieve virtual reality with low latency (Samsung, 2017).

In the current generation, besides four physical buttons and a touchpad for navigation, Samsung also introduces a Bluetooth controller similar to the Google Daydream controller. One advantage of Gear VR when compared with Google Daydream is that the device has wider field of view, which is 101 degree. With this, the VR experience can be more immersive and more natural. However, the headset still requires users to adjust centre alignment and focus whenever they put their phone into the headset (Smith, 2017).

While the compatible phone line-up is bigger than the Google Daydream one; the cheapest one, Galaxy S6, is still priced at 299 USD. Moreover, the users still have to spend around 45 USD for the Gear VR headset, and 39 USD for the controller (Amazon, 2016) (Peckham, 2017).

3.3 Virtual Reality application in Museum & Gallery

With the release of consumer VR headsets such as Oculus Rift and Google Cardboard, many museum organisations have found this kind of technology as an effective way for users to enjoy their museum without setting foot on the real location. While it is impossible to identify the total number of museums supporting VR around the world, they can be filtered into three main groups.

3.3.1 Experience through a first-party Virtual Reality application

One outstanding example is Renwick Gallery located in the USA. The museum has created a smart-phone application that allows users to visit the "WONDER" exhibition through their phone. The users can not only view several 360-degree photos of art installations, but also listen to, or watch art description throughout the virtual tour (Renwick Gallery, 2016).

Since the application is designed for only one museum, it can showcase all art photos, videos as intended by the creator. However, this approach usually takes more time and more cost than other approaches. It is also not popular due to the lengthy download time and complex installation process.

3.3.2 Experience through Web VR

Web VR is an open-source VR concept that allows the VR content to be displayed through the web browser. One of the best advantages of this concept is that it is compatible with most of VR platforms available for consumers, that includes Google Cardboard, Samsung Gear VR, Oculus Rift. Instead of requiring a custom-made smart-phone application, users can use any compatible web browser to enjoy the VR content (WebVR, 2017).

Recognising this peak, the National Museum of Natural History and the British Museum have implemented Web VR to their website to showcase the content of their museum (Smithsonian National Museum of Natural History, 2016) (British Museum, 2017). Due to the open-platform nature, this approach may cause a different type of issues when using through various web browsers or different smart-phones.

Indeed, the National Museum of Natural History virtual museum tour does not provide an error-free experience. While there are several 360-degree photos on the website, switching to VR mode does not display them in stereoscopic format, but instead, show them in a monoscopic format that can only be viewed without the VR headset.

3.3.3 Experience through third-party Virtual Reality application

Not all museums have a development team that is capable of developing a custom-made VR experience. Instead, they offer a collaboration with other development companies that are experts in this field. Some notable application examples are Jaunt VR, YouVisit, and Samsung Milk VR (Korolov, 2016).

Some museums even have their virtual tour designed by Google (AMNH, 2015), which can be experienced through a smartphone application called Google Arts & Culture (Google, 2016a).

While this approach is faster and easier for the museums, this type of virtual museum tour either either requires payment (View-Master, 2017) or does not provide a good museum experience (Vincent, 2016).

3.4 Challenges for smartphone-based Virtual Reality

3.4.1 Virtual reality sickness

When exposed to VR content, users may experience some negative side effects called virtual reality sickness. Similar to motion sickness, virtual reality sickness symptoms are seen as discomfort, headache, nausea, pallor, disorientation, stomach awareness, and apathy (Kolasinski, 2014).

Since the early 1990s, negative side effect in VR has been examined by researchers such as Cobb (Cobb, 1998), Sharples (Sharples, 2008). Based on the work of other researchers, Stanney (Stanney, 2009) states that after viewing virtual environment through a head-mounted display, 80-95% of the testers experienced some negative side effects. Another research from Sharples (Sharples, 2008) concludes that 60-70% of the testers experienced sickness symptoms after experiencing various presentations of the virtual environment.

While they have similar symptoms, the cause of virtual reality sickness is different from motion sickness. It is because of the disruption in the vestibular system. When the head moves, this system sends the motion signals to the brain, which helps it identify where the body is in the space. When viewing a fast-moving VR content, the eyes send fake motion signals to the brain. On the contrary, the vestibular system still tells the brain that the body is not moving. The mismatch between these two systems is the reason for virtual reality sickness (Pappas, 2016).

In the 2014 Google I/O developer conference, Alex Faaborg, a designer of Google

VR technology, stated that it was essential for the VR system to have a connection with the user body. For more detail, a VR mobile application needs to have a stable camera system with responsive head tracking sensor, which helps users navigate smoothly; the transition between scenes in VR content is also required to be seamless without any flashing image or sudden moving object. These characteristics will help avoid nausea problem in the users (O'Connor, 2015).

3.4.2 Limited Virtual Reality contents

Currently, VR contents for smartphone-based VR are limited by the capability of smartphone hardware. Most VR applications have a very simple graphic and a short playtime, making the platform only a gimmick. Moreover, current smartphone hardware has a very high latency when compared with the dedicated-hardware VR type (Petrovan, 2016) (Google, 2017a), which causes discomfort and nausea to users after a period of usage. While Google has addressed this issue with the introduction of a new VR platform named Google Daydream, there are only 4 compatible phones available on the market (Google, 2017b).

Another reason for the lack of VR applications is that most consumers are not familiar with the VR technology like Google Cardboard; content creators do not find the market large enough to develop a quality experience (Simonite, 2015).

3.5 3D contents in Virtual Reality

3.5.1 How does 3D effect work?

Colour and depth signals deliver visual information of the World to human. The human brain continuously receives visual cues (from the two eyes) to rebuild a spatial 3D structure of the surrounding effortlessly. In other words, it is natural for people to see things in 3D. This enables humans to discover the appearance, shape, and distance of distinct objects on their surroundings.

In recent years, the increasing popularity of 3D cinemas and 3D TVs have attracted attention from the public to the science of this two-eyed depth perception. There has been a dramatic expansion in the number of 3D display devices, movies, and games with 3D content in the consumer market. In principle, they attempt to deliver two different views, one to each eye; and force the brain to recover the desired 3D scene. For instance, 3D movies are simply made by two side-by-side video cameras. They are placed mimicking the arrangement of human eyes when observing scenes through two perspectives, which are horizontally separated by a small distance.

The stereo vision system of animals including humans has been evolving for millions of years and is evidenced to be an essential factor for survival. There are many advantages to the human vision; the ability to perceive distance is considered as an important factor. Even the human depth estimation can be made from analysing the perspectives of objects and their shadows. From the researcher experience, the best and fastest way is to solve the correspondence problem of stereo vision. It determines the patterns viewed from the left eye corresponding to which viewed from the right eye, to allocate the same points or regions (Churchland & Sejnowski, 1994). For instance, the separation between two correspondences of the same point from the two views defines how close or how far from us that point is in space.

3.5.2 Common stereoscopic 3D techniques

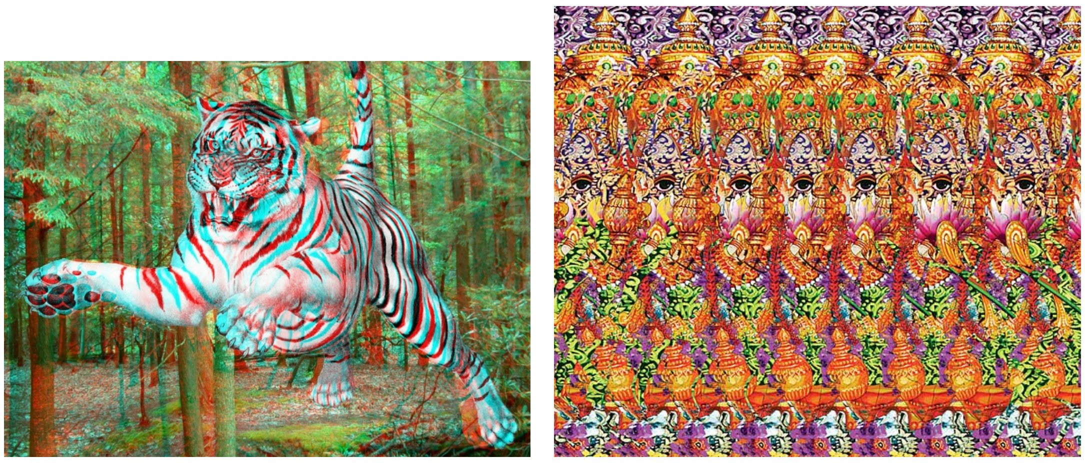

(a) An anaglyphic image (b) A stereogram image

Figure 3.3: Examples of anaglyphic and stereogram images

Anaglyph

In 1852, W. Rollmann first introduced a stereoscopy technique called "Farbenstereoscope", which manipulated the colour of the drawing to produce 3D effect (Rollmann, 1853). The technique requires the use of glasses colour coded with red/blue, red/green or more frequently red/cyan channels. These colours are merged with the same colour type in an image to create 3D effect.

For an example, when viewing with red/cyan filtering glasses, most of the left (red) and right (cyan) features in images are filtered to be perceived by the corresponding eye. After the golden age of 3D in the 1950s, anaglyph 3D has become popular once again in the 2000s thanks to the increasing trend of low-cost 3D movies in cinema and home video.

Stereogram

Another special type of stereoscopic image is the single-image stereogram or autostereogram, which creates depth illusions by using image patterns consisting of many random dots or repeating images. Stereogram has been experimented since the early 1990s. In 1970, Japanese graphic designer Masayuki Ito introduced a random-dot stereogram method based on the work of Bela Julesz (Howard & Rogers, 2012). The method was used again when Christopher Tyler combined it with single-image stereogram method to create the first autostereogram, which allowed human brain to see 3D effect without the need of glasses (McAllister, 2006). It is not until 1990 that autostereogram has gained huge interest from the public (Weibel, 2005).

The entire image is achieved by encoding three-dimensional images into distortions of symmetrical patterns which can then be decoded by misleading the eye convergences. To see the depth illusion, viewers must focus on one point in front of or behind the image surface. While doing that, the repeating patterns are moved sideways, and the brain attempts to fuse them together by focusing on certain pairs at different parallax angles, causing these patterns to appear floating at different depths correspondingly.

Side by side parallel-eyed and cross-eyed views

These are two popular methods for viewing 3D scenes with the naked eyes. Correspondingly, parallel-eyed and cross-eyed side-by-side stereo pairs are made for each method. They are single images stored in general image formats (JPEG, PNG, BMP, GIF, etc.).

The two images have simple internal arrangements: they can be placed horizontally side-by-side. Depend on these image positions, which are either left/right or right/left, viewers either perform cross-eyed technique or parallel-eyed technique to see the depth illusion of the images.

In the parallel-eyed technique case, their left eye must look at the left image and their right eye looks at the right image. The viewers can also perform an alternative technique by looking “through” the photo as if they focus on a far point behind the image surface for the same effect. For the cross-eyed images, viewers must converge their views in a way that the left eye looks at the right image and the right eye looks at the left image.

When using a VR headset, each eye can only see the image that the computer create specifically for it. Because of this, the viewers can easily see the 3D effect of parallel- and cross-eyed image without actively converging their eyes to focus on the correct image. This is the reason that the researcher chooses this type of stereoscopy as an approach to create 3D effect for the project.

3.5.3 3D visualisation with Virtual Reality devices

Before the time of digital photography, 3D cinemas, and 3DTVs; stereoscopy had been noticed and investigated by C. Wheatstone since the 1840s (Wheatstone, 1838), who had produced some early explanations of binocular vision. Motivated by his research, O. Holmes invented a stereoscope (Holmes, 1859) a few years later (as shown in Figure 3.2–top-left). Back in those days, this was a relatively popular kit which had a lens for each eye to make the image appear larger. It also horizontally translated the images to obtain a similar real-life stereoscopic view. The same idea still applies in today’s state-of-the-art VR devices such as the low-cost Google Cardboard (Bavor, 2016), Samsung Gear VR (Samsung Electronics, n.d.), and the premium Google Day Dream (Wee, 2016) (shown in Figure 3.2). In short, the LCD screen on each device displays a side by side image similar to what is seen in Fig. 3.1.1; here, each of the human eyes sees a different view, and the human brain does the hard work to merge these views and reconstruct a corresponding 3D scene.

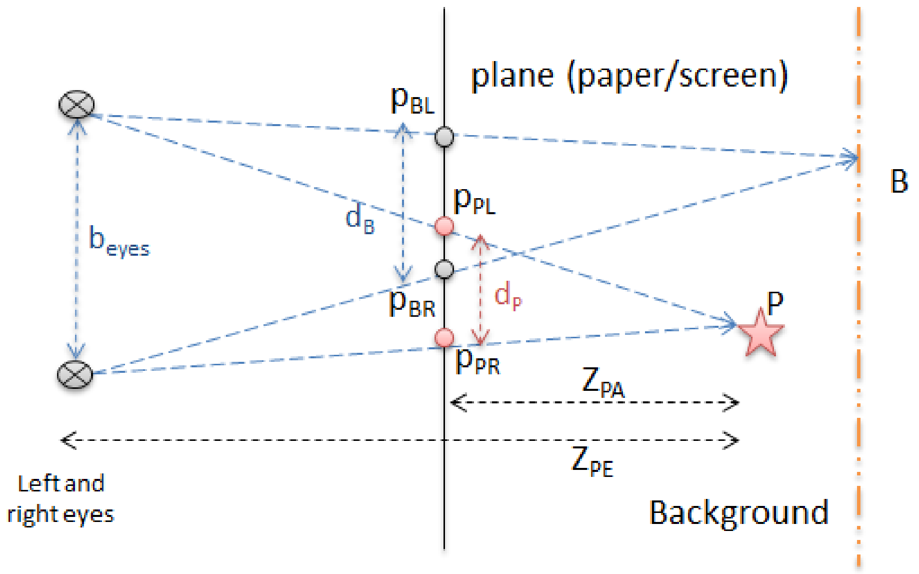


Figure 3.4: Principle of perceiving 3D from a VR screen

Figure 3.4 demonstrates the background theory of stereo vision that makes VR works. Shown in the figure are two points: B is a point virtually lying on the background, and P is a point lying in front of the background (they may look different in shape or colour). Human eyes have a base distance b_{eyes}. These two points are projected on to the screen at different points: p_{BL} and p_{BR} for B, and p_{PL} and p_{PR} for P. Many 3D points will appear in human brain when the eyes see a set of two projected correspondence points. Theoretically, the distance $d_P = p_{PR} - p_{PL}$ between two correspondence points p_L, p_R will determine how close/far the point is, in space:

$$\frac{Z_{PA}}{Z_{PE}} = \frac{d_P}{b_{eyes}} => d_P = \frac{Z_{PA} \times b_{eyes}}{Z_{PE}} \tag{3.1}$$

If $d_P issettobeequaltob_{eyes}$, the point P will appear at infinity. When d_P gets smaller, the point P gets closer to us. This establishes the optical principle of today's VR.

3.5.4 Binocular problems with traditional Virtual Reality kits

While traditional stereoscopic kits and VR devices have the same principal in binocular vision, the recent VR technology still has two advantages over the former. The first advantage is the constant distance and angle between stereoscopic images and human eyes. Even though 3D cinemas and 3DTVs also have digital processing computers to display multiple stereoscopic images and videos to users, their binocular vision can be affected by the binocular disparity.

On the first hand, with 3D cinemas and 3DTVs, The closer viewers to the stereoscopic images, the greater the binocular disparity is (Leroy, 2016). The overly wide binocular disparity can cause double vision and strain human eyes. The same issue can also happen if the viewers are on the far left or the far right of the stereoscopic images. To reduce this binocular disparity, the viewers can sit further from the stereoscopic images. However, sitting too far from the display is prone to create a conflict between accommodation (a process which the eyes are keeping the sharpness picture a particular object while focusing on it) and vergence (the movement of two eyes toward the same point of vision) (Patterson, 2015). To allow the viewers perceive the depth of image, 3D cinemas and 3DTVs trick the eyes to converge to a virtual object on the display while still maintain the accommodation on the picture background. When the distance between the viewers and the display is large, the object will appear very small; thus makes it difficult for the eyes to focus. Moreover, other objects that either move around the screen or have high luminance are prone to distract the viewers. They can create accommodation-vergence conflicts which can cause significant eye fatigue and visual discomfort (David M. Hoffman, Ahna R. Girshick, Kurt Akeley & Martin S. Banks, 2008).

On the other hand, VR devices allow the viewers to adjust the distance and angle of the image suitable for their preferences. Since there is no other object in the viewers' vision besides the stereoscopic image when using the devices, accommodation-vergence conflict is avoidable.

It is true that the old stereoscopic kit created by O. Holmes can get rid of these issues as well. However, the lack of digital processing power makes it impossible for the viewers to enjoy different stereoscopic images seamlessly. On the contrary, VR devices are capable of displaying not only static stereoscopic images but also stereoscopic videos, which is its second advantage.

Chapter 4

Project Objective

4.1 Research rationale

Currently there are many Virtual Reality (VR) applications that are compatible with smartphones. However, most of them only focus on the entertainment aspect. Moreover, their contents are very limited, making these applications become unproductive just after a few minutes of usage. Because of these issues, VR has not become an effective tool for education in general. Besides VR, stereoscopy is also a popular topic that attracts the public today. There are many creators around the world who are sharing their stereoscopic contents on the Internet every day. If used correctly, these contents can be great assets that help students become more interested in school sessions.

The purpose of this research is to combine these two trending topics into a new and exciting application that not only provides a new way to experience 3D effect, but also allows teachers to deliver their lectures more effectively. The application allows users to view 3D content without the need of 3D glasses, and improves the immersion by putting them in a virtual room covered with 3D contents. Besides some custom-made contents, most of the stereoscopic contents will be retrieved from the Internet. Moreover, the application is required to be low-cost and usable at any time and at any location.

4.2 Research ethic

Most contents will be retrieved from the Internet based on the topic that users choose. There are also some custom-made contents, but they are related to educational subjects such as animal, landmark, and museum. Therefore, it is unnecessary to apply for ethical approval for the research.

4.3 Application features

In order to meet this project objective, a list of core features for the application have been identified.

4.3.1 Low-cost and easy-to-use

Because the application is designed to be ubiquitous, it cannot be either costly nor complicated. The hardware required for the application operation are also required to be minimal.

4.3.2 VR menu

In contrast with traditional menu, it is very difficult to navigate VR menu through common hand controls such as keyboard typing, mouse clicking, or touch gesture. This is because users' vision and sense of direction are already used to navigate their "virtual-self" in the virtual world. Instead of requiring the users to "press" or "touch", VR menu should allow them to navigate through their "virtual-self". To be more precise, the menu should appear as a 3D object in the virtual world and can be navigated by head-movement camera or voice command.

4.3.3 Stereoscopic content classification

Since most of the stereoscopic contents will be retrieved from the Internet source, it is important to have a data retriever tool that can distinguish between stereoscopic and other types of content on the Internet. Based on the keyword chosen by users, the tool will perform a search for stereoscopic content related to it.

4.3.4 Stereoscopic content display

Instead of creating specific 3D contents, the application makes use of the huge library of stereoscopic contents available on the Internet. However, the problem of this approach is that these contents are not designed to be viewed in VR environment. Therefore, a special display mechanism is created to convert these contents into virtual reality objects.

4.3.5 Stereoscopic content storage

Application size is also an important factor that determines whether the application can be ubiquitous or not. Adding too many 3D contents can increase the size dramatically. Moreover, the 3D contents conversion process can also enlarge a kilobyte-size content into a megabyte-size content if the conversion algorithm is not optimised. At the current stage, an online content storage for the application is preferable.

4.3.6 Virtual gallery room

The virtual environment should have a friendly design that mimic the design of a real-life gallery room. This is to help users feel comfortable when using the application. Besides, the users should be able to navigate in the environment easily since they already had experience in moving around small rooms in real life.

Chapter 5

Development Plan

5.1 Smartphone-based Virtual Reality development environment

The main goal is to deliver an ubiquitous approach to 3D vision through Virtual Reality (VR). Thanks to the very low-cost headset and the vast list of compatible smartphones on the market, Google Cardboard is the most potential VR platform in the current time.

Even though Google Cardboard supports both Android and iOS, the project will just focus on Android development. This is because iOS is available exclusively on iPhone, a line of smartphones developed solely by Apple Inc (Apple Inc., 2017).

According to Google, an Android Google Cardboard application can be developed through three development environments: (1) Android IDE, (2) Unity Engine, (3) and Unreal Engine (Navarro, Pradilla & Rios, 2012).

5.1.1 Android IDE

With Android IDE, developers can choose either Android SDK or Android NDK to develop the application; each option has its pros and cons. The main benefit of Android SDK is its use of Java, one of the most popular programming language (Stack Overflow, 2017).

While using Android SDK may cause the application demand more powerful hardware power than it truly needs, the development time is shorter when compared with developing an application using native code (Horton & Portales, 2016).

The second option, Android NDK, is a toolset that helps developers access directly to smartphone hardware and native functions. It also allows the application to utilise the hardware power as well as achieving low latency in smartphone sensors. However, to use this set of tools, developers are required to have sound knowledge of C or C++ programming languages. They also need to understand how to find or create a suitable native code library of necessary functions for their application development (Google, 2017c).

There is no doubt that both options provide excellent development environments for Android applications. However, developing a VR application through these options requires more knowledge in API and a new set of function codes designed for VR (Google, 2016b) (Mullis, 2016).

5.1.2 Unity Engine

Unity is a game engine developed by Unity Technologies. The engine is utilised to work in over 20 popular platforms, such as iOS, Android, Windows; but its focus is the mobile platform. 34% of top 1000 free mobile games are based on the engine (Unity Technologies, 2016a).

There are several features that make Unity as a top choice for the game engine. Firstly, it supports many modern game development tools, including multi-threaded job system, State Machine hierarchies and transitions, NVIDIA® PhysX®, and Real-time Global Illumination. These tools not only help create games that are suitable for current users but also dramatically reduce development time.

Secondly, there are four versions of Unity: (1) Personal (Free), (2) Plus ($35 monthly), (3) Pro ($125 monthly), and (4) Enterprise (custom-made for enterprise); which are suitable for all types of game developers. The free Personal version is good enough for students and beginners to develop their games as well as releasing them on the market.

Finally, Unity has a huge community that includes a library of tutorial suitable for all types of developers. It also has an asset store that houses over 1700 free & paid extension tools. In addition, there is an online forum where developers across the world discuss and help each other in game development (Unity Technologies, 2017a).

5.1.3 Unreal Engine

Unreal Engine is a game engine created by Epic Games, one of the leading gaming technology company. In 2006, Unreal Engine 3 was released; from there, it became one of the most popular engines that was used for developing blockbuster games such as Gears of War, XCOM: Enemy Unknown. Moreover, it was also used for a low-cost but high-quality development platform for students and independent developers (Busby, Parrish & Wilson, 2009) (Tavakkoli, 2015).

When Unity, a game engine competitor, announced a free model for independent developers, Epic Games also decided that it was the time for its game engine to become ubiquitous. In 2015, all of the features from the latest version, Unreal Engine 4, has become free of charge for personal usage. The commercial developers only have to pay

5% of their profit when their games are on sale on the market. The company also states that they expect VR will be the highlight of the gaming industry; therefore, it is a vital part of the future of Unreal Engine (Crecente, 2016).

Unreal Engine 4 has many advanced features that are suitable for developing high budget games, including DirectX 12 Rendering, Cascade Visual Effects, Artificial Intelligence, and Post-Process Effects. Moreover, Unreal Engine 4 has introduced Blueprints Visual Scripting, a tool that simplifies the game scripting process (Epic Games, 2017).

5.1.4 Unity Engine & Unreal Engine comparison

Since Android IDE demands much more knowledge and development time to achieve the project goal, only Unity Engine and Unreal Engine have a potential to deliver the project in the time frame.

Each of the engines has their advantages and disadvantages; the researcher compares features of both the engines to find the most suitable development platform. The researcher groups them into three aspects: (1) Development feature, (2) Development fee, and (3) Community. The comparison is displayed in Table 5.1.

In the comparison table, it is noticeable that Unity Engine provides better performance for smartphone applications. It also has a much bigger asset and tutorial library for novice developers. Moreover, the engine ability to export the project to a wide range of platforms, including Android, iOS, and Web application, can help make the application ubiquitous.

Unity 5	Unreal Engine 4
Development feature	
Mainly based on C# & JavaScript	Mainly based on C++ & UnrealScript
Requires knowledge of programming (at least C# or JavaScript) for project development	Include Blueprint feature, which allows developing a project with minimum knowledge of Coding
Provides wider tweak & settings for the environment and object	Environment & object settings in Blueprint are limited. Requires custom-made model & platform for more settings
Graphic is good enough for mobile users, but limited compared with Unreal Engine	Graphic is generally better, including better shadow, physic, terrain
Better performance for mobile use	Better performance for PC & Console
Supports a wide range of platforms, including mobile & web	Supports mainly PC and Console
Development fee	
• Free for personal use or commercial use with annual venue less than \$100k. • The Free edition lacks of Profiler feature, which is used for performance benchmark for the project	• Free for personal use. Includes all features, it is no difference between Free edition & Premium edition. • Requires 5% profit for commercial use
Community	
Huge user base	Moderate user base
Extensive asset library, including objects, scripts, animation, tools required for a complete game with low fee	Small asset library. High fee of use
Huge database of the tutorial, including videos, demo & scripts. Requires less time for training	Limited database of tutorial
Community & Tutorials suitable for all types of developers, including beginners and hobbyist	Tutorials largely are designed for designers rather than programmers

Table 5.1: VR IDE comparison table
Acquired from (VR Status, 2016) (Pluralsight, 2014) (Eisenberg, 2016) (Unity Technologies, 2017a) (Epic Games, 2017)

5.2 Hardware resource for development

1. **Xiaomi Mi 4C**

   Released in late 2015, the smartphone from Xiaomi has a powerful processor with an IPS display. The resolution of the smartphone is 1920x1080, which is considered to be very clear for a 5-inches display. While the smartphone is not officially available in New Zealand, it can be bought from China with a price tag of 175 NZD.

2. **Windows Personal Computer**

   According to the system requirements from Unity Technologies, Unity development tool needs a GPU that supports at least DirectX 9 with shader model 3.0. The CPU is also required to be capable of executing SSE2 instruction set. For the required operating system, Windows and MacOS X version that are older than Windows 7 SP1 and Mac OS X 10.9 respectively are not supported. Since the researcher's computer contains a 2015 Skylake CPU and a 2016 AMD GPU, the development tool is expected to operate with great performance.

3. **Google Cardboard VR Headset**

   Of all smartphone VR headsets that are available on the market, Google Cardboard VR headset is the cheapest and the most popular one. By simply following the cardboard design guide provided by Google, users can modify any cardboard box to use with their smartphone. Moreover, this type of headset is also available for sale in many gadget stores.

4. **Xiaomi VR Play Headset**

   Xiaomi released one of their first VR headset in 2016 with the name as Mi VR Play. The headset follows the Cardboard design pattern provided by Google.

Therefore, it has a very similar structure, and can be considered as a Google Cardboard headset. While the material is changed from cardboard to more premium feel lycra, the headset still has a capacitive touch at the top.

5. **RITECH VMAX VR Headset**

   Instead of following Google Cardboard design template loosely, RITECH has introduced several new features for their headset. The most important one is an ability to alter headset lenses in axis position and dioptre. With this, the headset is suitable for users with different facial structure as well as the ones who have myopia or hypermetropia. Moreover, there are some improvements in the lenses and the capacitive button.

   For the lenses, they are clearer with wider field-of-view when compared with Mi VR Play. For the capacitive button, even though it is more tactile and easier to press, the button does not always work with Xiaomi Mi4C. The only disadvantage is the material of the headset, which is made of cheap plastic.

   RITECH VMAX can be bought from China for 15 USD. However, it is not officially available in New Zealand currently.

5.3 Software resource for development

1. **Unity 5.6**

   Unity is one of the most popular 3D engine in the market. The tool has a user-friendly graphical interface and a huge library of coding data for inexperience developers. Based on the comparison between Unity and another user-friend engine named Unreal Engine, which shows in section **5.1 Smartphone-based Virtual Reality development environment**, application created with Unity usually has better performance in the smartphone environment.

The development tool also utilises C# programming language and possesses a library of built-in 3D models. These two factors can help reduce the amount of resource and time required for the development. Unlike the previous versions, version 5.6 of Unity introduces a native integration for Google VR. This can minimise the amount of conflicts occurs when implement Google VR SDK to this development tool.

2. **Unity VR package**

   Besides Goole VR, Unity also supports other VR platforms such as Oculus Rift and Samsung Gear VR. This support allows Unity Technologies company to create a VR package for these platforms, which contains a lot of built-in assets and C# scripts. While this project focuses on Google Cardboard VR platform, this package still has benefits to the development. By analysing the package content, the researcher can create custom C# scripts and assets based on it.

3. **Google VR SDK for Unity**

   While developers can create their own script to help their applications run properly in Google VR platform, the process could take a lot of time and resource. Google has introduced a software development kit (SDK) that converts any Unity application to be operational in Google VR platform. The SDK is not only easy-to-use, but also frequently maintained by Google development team.

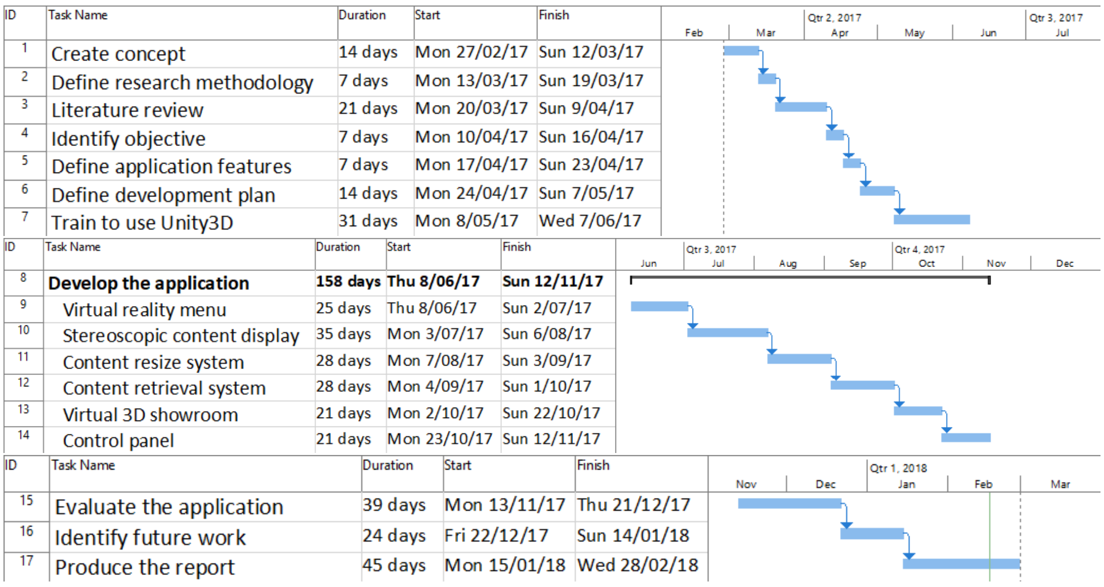
4. **AUT University Webserver**

   Most 3D contents are hosted outside of the application to avoid the massive increase of the application size. Since an online storage is currently the best option, the researcher has contacted the supervisor to reserve a small space of AUT University Webserver for the 3D contents.

5. **GIMP 2.8**

   GIMP is a free and open-source graphical editor that contains a lot of robust tools for image editing. This software helps fix any image issue of the 3D contents retrieved from the Internet. Moreover, it also allows the researcher to create notification panels and 2D background for the application.

5.4 Project schedule



ID	Task Name	Duration	Start	Finish
1	Create concept	14 days	Mon 27/02/17	Sun 12/03/17
2	Define research methodology	7 days	Mon 13/03/17	Sun 19/03/17
3	Literature review	21 days	Mon 20/03/17	Sun 9/04/17
4	Identify objective	7 days	Mon 10/04/17	Sun 16/04/17
5	Define application features	7 days	Mon 17/04/17	Sun 23/04/17
6	Define development plan	14 days	Mon 24/04/17	Sun 7/05/17
7	Train to use Unity3D	31 days	Mon 8/05/17	Wed 7/06/17
8	**Develop the application**	**158 days**	**Thu 8/06/17**	**Sun 12/11/17**
9	Virtual reality menu	25 days	Thu 8/06/17	Sun 2/07/17
10	Stereoscopic content display	35 days	Mon 3/07/17	Sun 6/08/17
11	Content resize system	28 days	Mon 7/08/17	Sun 3/09/17
12	Content retrieval system	28 days	Mon 4/09/17	Sun 1/10/17
13	Virtual 3D showroom	21 days	Mon 2/10/17	Sun 22/10/17
14	Control panel	21 days	Mon 23/10/17	Sun 12/11/17
15	Evaluate the application	39 days	Mon 13/11/17	Thu 21/12/17
16	Identify future work	24 days	Fri 22/12/17	Sun 14/01/18
17	Produce the report	45 days	Mon 15/01/18	Wed 28/02/18

Figure 5.1: Schedule estimation of the research

Figure 5.1 shows the schedule estimation of the research. After all of the necessary resource, including development tools and related documents, have been collected, the official development begins in June 2017. While six the main features have been organised clearly, their development durations are not completely accurate. There are certain factors, such as unexpected bugs and delayed evaluation process, that can make the development stage unable to meet the time frame. However, it is important to conclude the development stage in November 2017. This is to make sure that the research can complete at the deadline (February 2018).

5.5 System design

5.5.1 System architecture

The system combines human-computer interaction with local and global system collaboration to deliver stereoscopic contents to users. There are three main modules in this system: (1)Client Side, (2)Data Acquisition, and (3)Preprocessing. Each module has various functions that not only depend on each other, but also connect with functions in other modules. More detail can be seen in the Figure 5.2

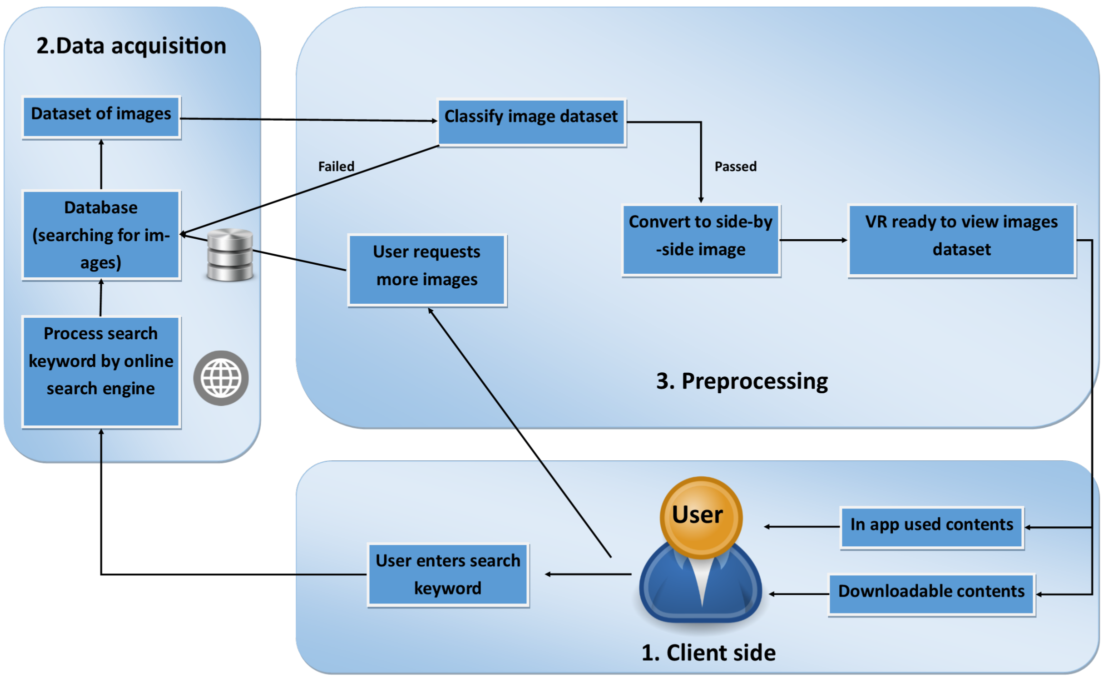

2.Data acquisition
Dataset of images
Database (searching for im-ages)
Process search keyword by online search engine
Classify image dataset
Failed
Passed
Convert to side-by -side image
VR ready to view images dataset
User requests more images
3. Preprocessing
User
In app used contents
Downloadable contents
User enters search keyword
1. Client side

Figure 5.2: A diagram that describes the system architecture

Starting with the Client Side module, when users input a keyword, the application will retrieve the data and use it for the content searching process in the Data Acquisition module. If there are suitable contents in the local storage, they will be prioritised. Otherwise, the application will use an online search engine to find the content related to the keyword. After that, the search results will be transferred to the Preprocessing module so that they can go through a stereoscopic content classification process. This is to ensure that when the classified contents go back to the Client Side module, they will be in correct format.

5.5.2 Application flowchart

Before the development stage begins, it is important to create an operation flowchart describing what requests the Client Side module send when being interacted by users. When the application starts, the users are asked to whether input a keyword or view the local gallery, which is stored in the application data. After that, depend on the initial choice, the application will load the correct stereoscopic content gallery for viewing. If the users want to change the content topic, they can input a new keyword at any time.

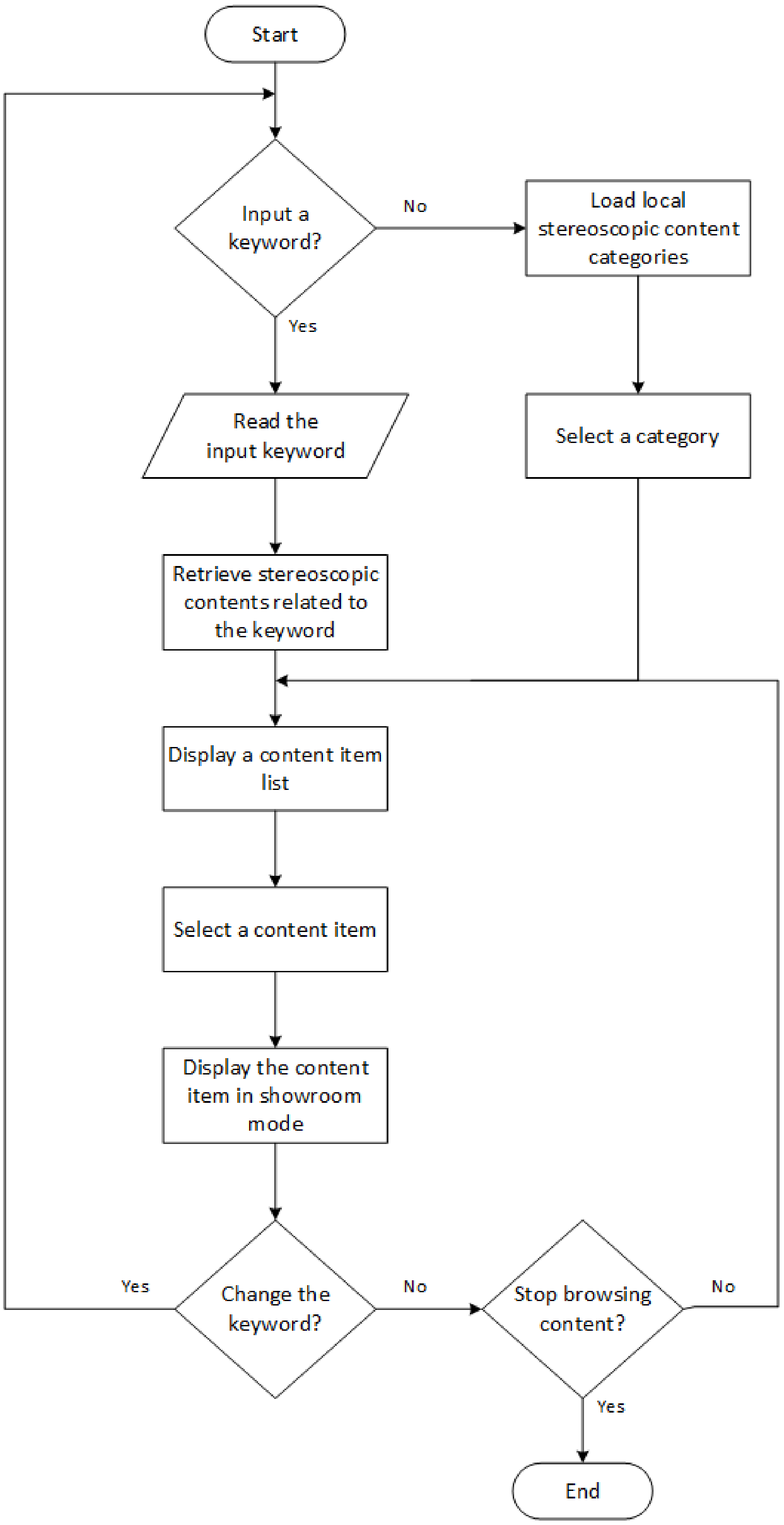

Start
Input a keyword?
No
Yes
Load local stereoscopic content categories
Read the input keyword
Select a category
Retrieve stereoscopic contents related to the keyword
Display a content item list
Select a content item
Display the content item in showroom mode
Change the keyword?
Yes
No
Stop browsing content?
No
Yes
End

Figure 5.3: Work flow of the Client Side module

Chapter 6

Development Stage

6.1 Getting familiar with Unity Engine

6.1.1 Roll-a-ball tutorial

This is the most basic video-tutorial that Unity provides for developers who are not familiar with Unity development tool. By creating a simple rolling ball game, the tutorial demonstrates how the engine works with the core features such as Objects, Physics, Prefabs, and Scripting (Unity Technologies, 2015b).

One of the main advantage of Unity engine is the ability of managing and controlling 3D objects and environments with just few simple steps. An example can be seen in Figure 6.1, the researcher could easily add a 3D sphere object into the environment.

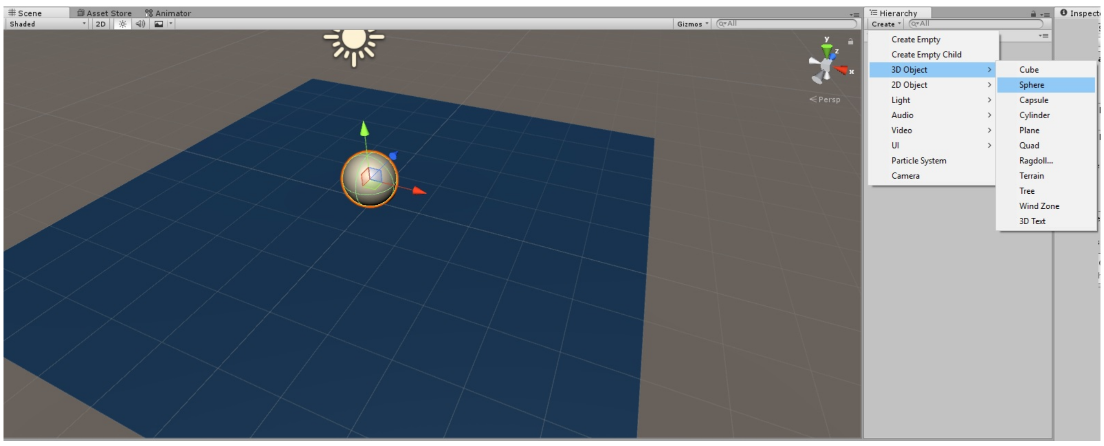


Figure 6.1: Steps to add a 3D sphere object into Unity environment

While Unity simplifies many of the initialisations of game development, developers are still required to manually program objects for more detail and complex behaviour, which is the characteristic of Scripting. This feature allows developers to input programming scripts to the objects through C# or Javascript. Even though the researcher was unfamiliar with these two programming languages, the past-experience with C++ helped him follow the tutorial with no difficulty, thus succeed in programming several object behaviours, which are player object tracking camera, physic collider, score system.

6.1.2 Survival Shooter tutorial

This is the tutorial for developers who are familiar with the UI and scripting feature of Unity Engine. In this tutorial, there are 10 video clips which were recorded during Unite 2014, a training event created for Unity developers. By following the instruction from these video clips, the developers are able to utilise the built-in AI feature as well as more advanced scripting technique and UI design. Figure 6.2 shows a part of the Unity AI system.

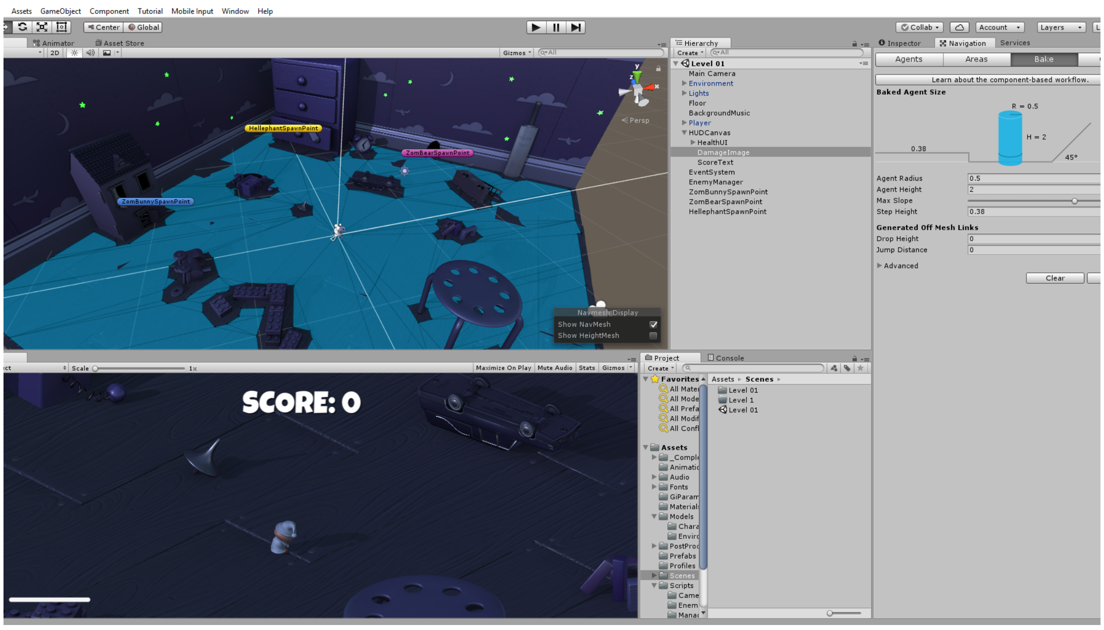
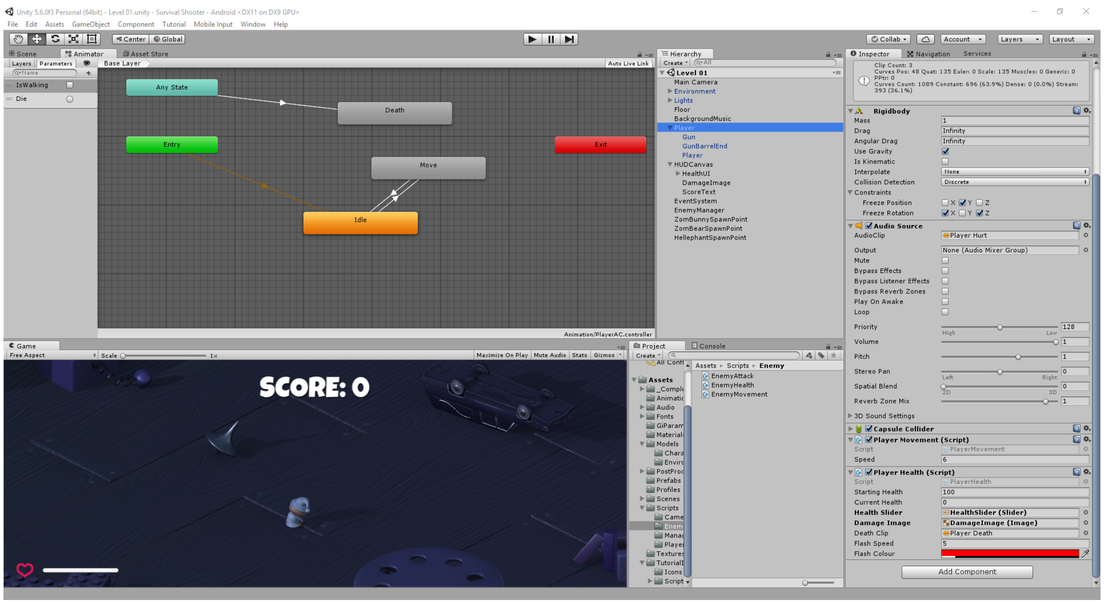


Figure 6.2: An image of the Unity AI system control layout

Although Unity supports both C# and JavaScript, the tutorial only focuses on one single programming language, which is C#. Throughout the video clips, the instructors also create several scripts to support the gameplay of this project. Each 3D object (a player, enemies) is required to be attached with scripts in order to perform basic actions such as moving, shooting. These scripts are also responsible for triggering Unity animation system. For an example, if the player health drops to zero, the script will trigger the player's dying animation, which is attached in Unity animator system.

Since this project is much more advanced than Roll-a-ball tutorial, it took quite a significant amount of time for the researcher to familiarise and finish all 10 video clips. This tutorial led to the conclusion of the first part of the training stage.

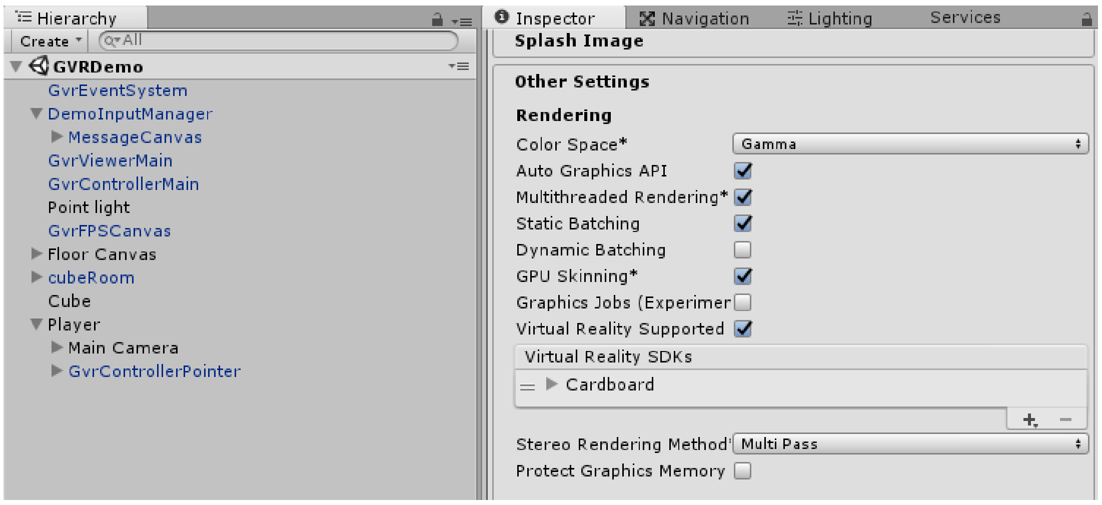


```
Clip Count: 3
Curves Pos: 48 Quat: 135 Euler: 0 Scale: 135 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 1089 Constant: 696 (63.9%) Dense: 0 (0.0%) Stream:
393 (36.1%)

Rigidbody
  Mass                     1
  Drag                     Infinity
  Angular Drag             Infinity
  Use Gravity              ✓
  Is Kinematic             ☐
  Interpolate              None
  Collision Detection      Discrete
  Constraints
    Freeze Position        ☐ X  ✓ Y  ☐ Z
    Freeze Rotation        ✓ X  ☐ Y  ✓ Z
Audio Source
  AudioClip                Player Hurt
  Output                   None (Audio Mixer Group)
  Mute                     ☐
  Bypass Effects           ☐
  Bypass Listener Effects  ☐
  Bypass Reverb Zones      ☐
  Play On Awake            ☐
  Loop                     ☐
  Priority                 128
  Volume                   1
  Pitch                    1
  Stereo Pan               0
  Spatial Blend            0
  Reverb Zone Mix          1
  3D Sound Settings
Capsule Collider
Player Movement (Script)
  Script                   PlayerMovement
  Speed                    6
Player Health (Script)
  Script                   PlayerHealth
  Starting Health          100
  Current Health           0
  Health Slider            HealthSlider (Slider)
  Damage Image             DamageImage (Image)
  Death Clip               Player Death
  Flash Speed              5
  Flash Colour             🟥
```

Figure 6.3: Animator system and some of the scripts attached to Player game object

6.1.3 Google VR SDK

Google VR SDK is a Unity project package developed by Google. It serves as a guideline for developers to implement Google Virtual Reality (VR) feature in their Unity projects (Google, 2017d). package includes several integrations, which are head tracking, user interaction system, side-by-side stereo rendering, VR distortion correction, automatic gyro drift correction, a centre alignment marker for assisting users to install the phone into the VR headset.


Hierarchy
Create All
GVRDemo
- GvrEventSystem
- DemoInputManager
  - MessageCanvas
- GvrViewerMain
- GvrControllerMain
- Point light
- GvrFPSCanvas
- Floor Canvas
- cubeRoom
- Cube
- Player
  - Main Camera
  - GvrControllerPointer

Inspector Navigation Lighting Services
Splash Image
Other Settings
Rendering

Color Space*	Gamma
Auto Graphics API	✓
Multithreaded Rendering*	✓
Static Batching	✓
Dynamic Batching	☐
GPU Skinning*	✓
Graphics Jobs (Experimer	☐
Virtual Reality Supported	✓

Virtual Reality SDKs
Cardboard

Stereo Rendering Method'	Multi Pass
Protect Graphics Memory	☐

Figure 6.4: Google implements custom game objects with Unity VR native system for Google VR.

While the package can dramatically reduce the VR implementation time, there are two issues with it. Firstly, scripts control most of the VR integrations, including stereo rendering and head tracking. In order to customise any of these integrations, developers are required to be familiar with all of the scripts included in the package.

Secondly, the package is designed as a Google VR template for developing multiplatform in Unity. Because of this, there are many assets and scripts that deem unnecessary for the current project.

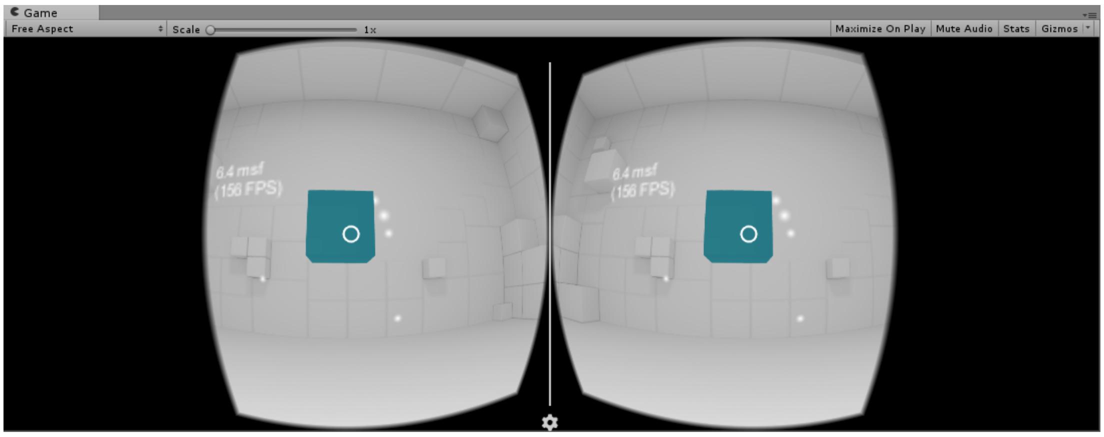


Figure 6.5: A demo scene from Google VR SDK

Since this was in the training stage, the researcher kept the whole package intact. All of the scripts and assets related to Android were to be identified at the end of the training stage.

6.1.4 Unity VR package

By version 5.6, Unity has included a native integration for Google VR (Unity Technologies, 2017e). Before that, only Oculus Rift and Samsung Gear VR were natively supported. Therefore, the VR package designed by Unity Technologies contains superior assets and more detail scripts when compared with Google's package.

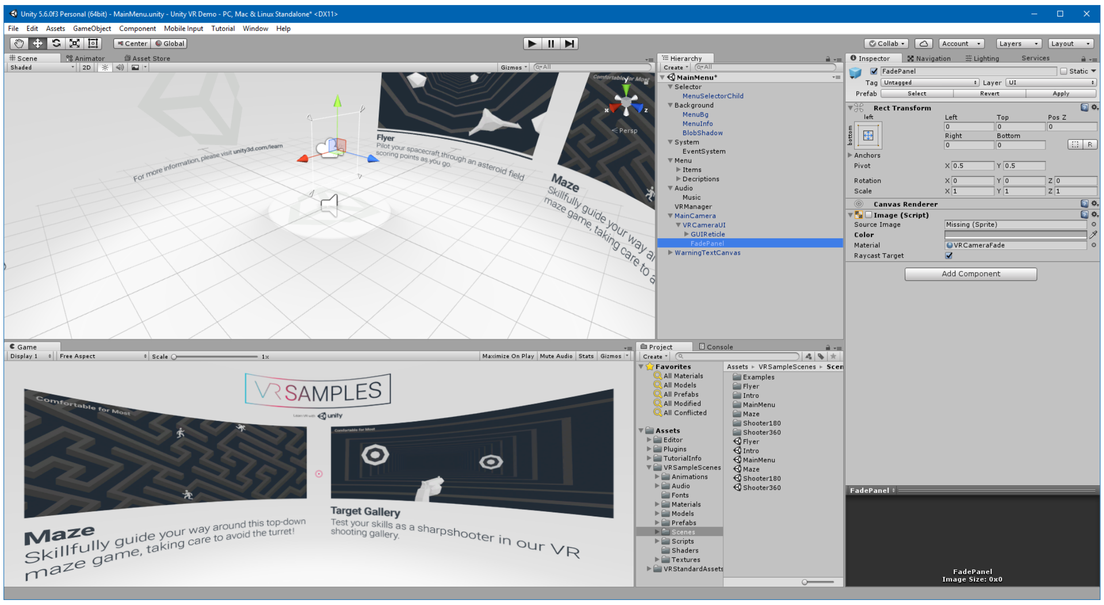


Figure 6.6: A demo scene and asset from Unity VR package

Because the package is only fully compatible with Oculus Rift and Samsung Gear VR, even if the package can be exported to an Android smartphone, it is impossible to either control or experience the application due to the lack of required hardware. Despite this issue, the assets and scripts still serve as templates for the researcher's project.

6.2 Application prototype development

6.2.1 First demo – July 2nd, 2017

The purpose of this demo is to demonstrate the ability of Google Cardboard platform in displaying stereoscopic photos and assisting users to experience 3D effect while looking at these photos. The content of this demo is short and simple. users are put into a virtual area where there are multiple stereoscopic photos surrounding them. By simply use head movement to put the camera pointer on one photo and press the trigger button, the triggered stereoscopic photo will be enlarged to cover all display area.

VR Menu

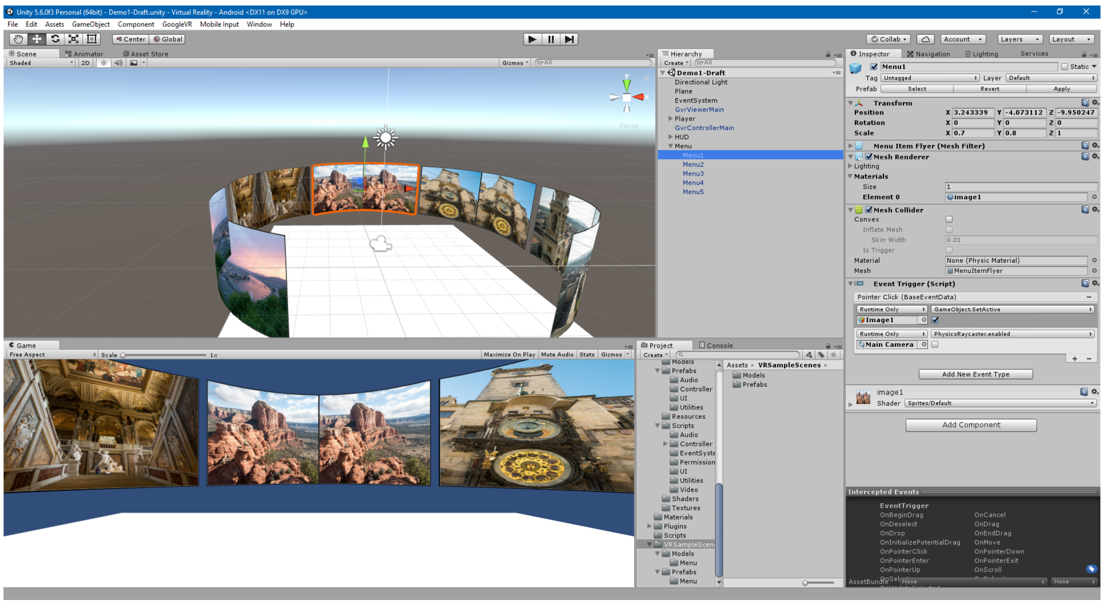
Cardboard VR control is different than traditional control. Instead of using a traditional control method such as a finger tap, a computer mouse, or a gamepad; users have to move their head to control the camera. A simple pointer on the centre of VR camera display acts as a virtual cursor for the users to point it at desired objects.

Because of this, VR menu needs to be simple, big, and easy to be navigated. The menu of this demo follows the Flat Menus Mapped on Geometry style, which is similar to the menu in Oculus Rift or Gear VR.

While Google VR SDK package includes a pointer and a VR camera template, Its 3D object is very limited. Even the 3D object library in Unity does not contain a 3D flat panel object that can be used as a VR menu. This gave the researcher two options: either (1) creating a custom 3D object or (2) borrowing some from other 3D library. The former option requires knowledge in 3D design, of which the researcher lacked, therefore it was not recommended.

Fortunately, Unity VR package has a demo of VR menu in Flat Menus Mapped on Geometry style, which includes a 3D flat panel object. Thanks to the templates from

Google VR SDK and Unity VR package, the VR menu of this demo was completed in couple of hours.


Unity 5.6.0f3 Personal (64bit) - Demo1-Draft.unity - Virtual Reality - Android <DX11 on DX9 GPU>
File Edit Assets GameObject Component GoogleVR Mobile Input Window Help
Center Global
Collab Account Layers Layout
Scene Animator Asset Store
Hierarchy
Demo1-Draft
Directional Light
Plane
EventSystem
GvrViewerMain
Player
GvrControllerMain
HUD
Menu
Menu1
Menu2
Menu3
Menu4
Menu5
Inspector Navigation Lighting Services
Game
Free Aspect
Maximize On Play Mute Audio Stats Gizmos
Project Console
Assets ▸ VRSampleScenes ▸
Models
Prefabs
Add Component
Intercepted Events

```
Menu1                                   Static
Tag Untagged            Layer Default
Prefab  Select  Revert  Apply
Transform
Position   X 3.243339  Y -4.073112  Z -9.950247
Rotation   X 0         Y 0          Z 0
Scale      X 0.7       Y 0.8        Z 1
Menu Item Flyer (Mesh Filter)
Mesh Renderer
Lighting
Materials
  Size        1
  Element 0   image1
Mesh Collider
Convex
  Inflate Mesh
    Skin Width   0.01
  Is Trigger
Material   None (Physic Material)
Mesh       MenuItemFlyer
Event Trigger (Script)
Pointer Click (BaseEventData)
  Runtime Only   GameObject.SetActive
  Image1         ✓
  Runtime Only   PhysicsRaycaster.enabled
  Main Camera
Add New Event Type
image1
Shader Sprites/Default
EventTrigger
OnBeginDrag              OnCancel
OnDeselect               OnDrag
OnDrop                   OnEndDrag
OnInitializePotentialDrag  OnMove
OnPointerClick           OnPointerDown
OnPointerEnter           OnPointerExit
OnPointerUp              OnScroll
```

Figure 6.7: The 3D menu is based on the 3D asset from Unity VR package.

Import stereoscopic photos

While there are many stereoscopic photos available on the Internet that can be easily found through Google, not all of them have the same quality and resolution. Moreover, copyright is also a concern when using Internet resource for the official application.

Since the purpose of this demo is to test the 3D photo demonstration ability of Google Cardboard platform, the researcher borrowed five photos from Nick Mann library (`stereoscopiclibrary.tumblr.com`) for the demo.

Importing image files to Unity is an easy process, users can follow the menu as “Assets/Import New Asset. . . ” or just simply drag the image files to Unity asset hierarchy. However, depends on whether the Unity object is 3D or 2D, image files are required to be converted to suitable formats.

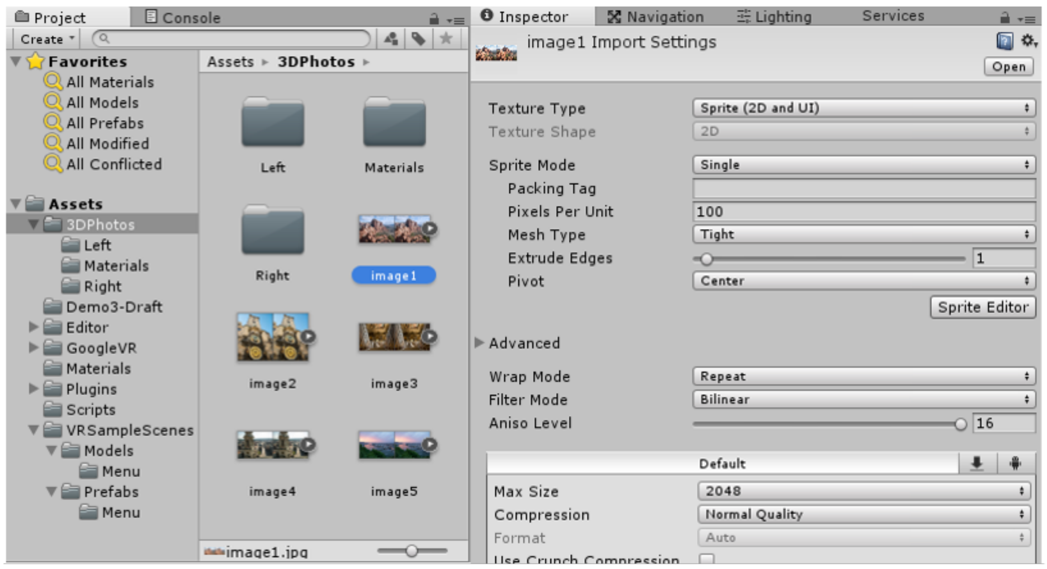


image1 Import Settings

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Single
Packing Tag:
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Pivot: Center
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 16
Max Size: 2048
Compression: Normal Quality
Format: Auto
Use Crunch Compression

(a) Image files as Unity sprite files

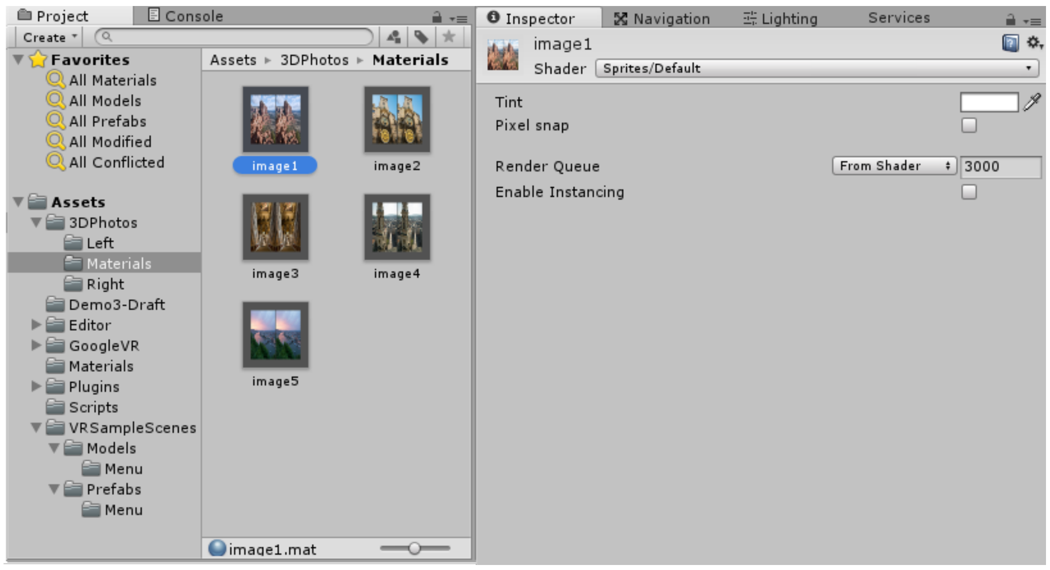


image1
Shader: Sprites/Default
Tint
Pixel snap
Render Queue: From Shader 3000
Enable Instancing

(b) Image files as Unity material files

Figure 6.8: A comparison image between Unity sprite and Unity material files

In order to have the imported photos displayed by the 3D flat panel objects, they are converted to Unity material files. Moreover, based on the Survival Shooter tutorial in the training stage, the photos have to be in the Unity UI component to fully cover the display area, thus being converted to 2D sprite texture are required.

Stereoscopic photo display mechanism

In this demo, there is a pointer that is used to target stereoscopic photos that users want to view. Developing this function requires two main parts. The first part is developing the pointer system. Google VR SDK package already has this feature. This pointer system also has a simple animation: whenever users hover the pointer on the menu, it will transform from a dot to a button. Therefore, the researcher easily completed this part.

The second part was creating a trigger mechanism. This was done by adding a Pointer Physic Raycaster script to the Unity camera, and an Event Trigger script to each menu. The purpose of the Pointer Physic Raycaster script was creating an invisible line from the camera (can be understood as “eyes” of the user) to the object. This script helped Unity understand which target the users were aiming.

For the Event Trigger script, in the Unity default settings, pressing Fire1 button or a finger tap on the touchscreen will serve as a pointer click (Unity Technologies, 2016b). Therefore, by setting up the Event Trigger script, the researcher could make Unity call a custom script or object behaviour whenever the object attached with the script had a pointer click triggered.

- **First try**

  According to the Survival Shooter tutorial, one of the easy way to set an image cover the whole display screen is using Unity UI Canvas. Because of this, the researcher tried to put all stereoscopic photos on the canvas object and set them to cover all display screen. Each photo was set inactive at default, and could only be activated when there was a pointer click trigger on the relevant menu. A custom script was also attached to each photo, which served as a trigger to deactivate the photo again if there was another pointer click trigger. While this approach worked well in Unity preview mode, the stereoscopic photos did not appear when the researcher tried the demo on an Android environment. More research was to be conducted on Google VR SDK and Unity UI Canvas to resolve this issue.

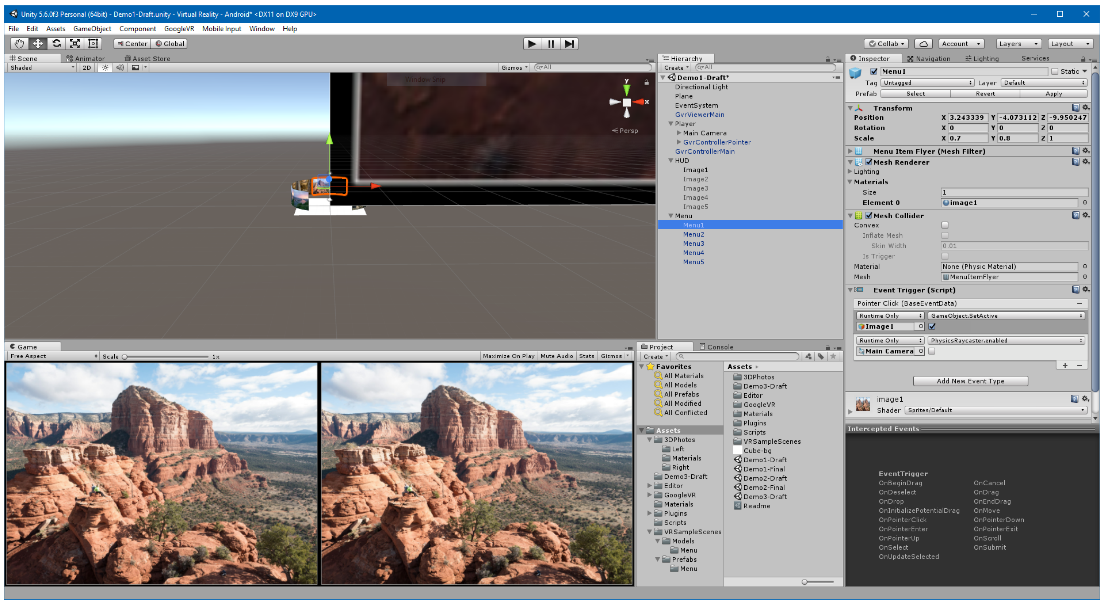


Figure 6.9: Using Unity UI Canvas feature for the stereoscopic photos display in the first approach

- **Second try**

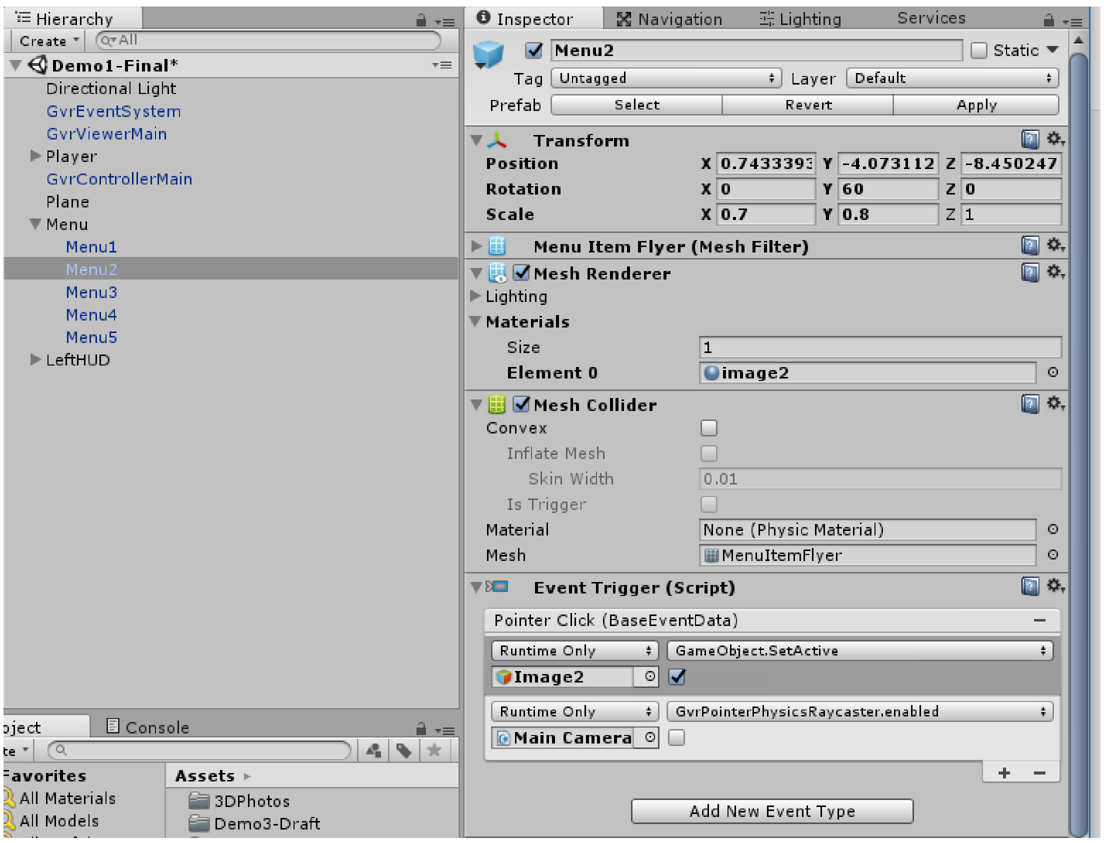
  Upon researching on Google VR SDK, the researcher realised that the VR camera display was controlled by GvrViewerMain object. Without it, the camera would go back to standard with only a single display screen. With its characteristic, another approach was proposed: by creating a custom script to automatically deactivate GvrViewerMain object, Unity UI Canvas as well as the stereoscopic photos would appear on the display screen.

  Once again, in Unity preview mode, the demo worked perfectly. But when the demo was tested in Android environment, the stereoscopic photos still not appeared on the display screen. Moreover, when GvrViewerMain object was deactivated, head-movement tracking stopped working, and display screen was still in VR mode with two separate cameras. While the issue was not solved, this approach helped the researcher understand that GvrViewerMain object contained a script that created two separate cameras at the start of the application. When the object was deactivated, these two cameras were still activated therefore it was impossible for the demo to switch to a single display with this approach.

- **The solution**

  With the introduction of VR native support, Unity has a toggle to switch the application from standard single camera view to VR camera view. This helps simplify the development of VR application through Unity. While developers can access this toggle with few simple steps, users are blocked from it.

  This led to the third approach of solving the issue of this demo: by creating a custom script to automatically switch between standard mode camera and VR mode camera, the stereoscopic photos could appear properly in Android environment as well as in Unity preview mode. Since there was a Unity library designed specifically for VR, the coding process was quite simple. The script was attached to every stereoscopic photo so that whenever these photos were activated, the script would trigger as well. Moreover, inside this script, there was also a function to recognise a touch input. This would help the script to switch back to both VR mode and the main menu whenever there was a single touch input from users.



```
Demo1-Final*
    Directional Light
    GvrEventSystem
    GvrViewerMain
  ▶ Player
    GvrControllerMain
    Plane
  ▼ Menu
      Menu1
      Menu2
      Menu3
      Menu4
      Menu5
  ▶ LeftHUD

All Materials
All Models
3DPhotos
Demo3-Draft
```

```
☑ Menu2                          ☐ Static
Tag  Untagged          Layer  Default
Prefab  Select  Revert  Apply

▼ Transform
Position   X 0.7433393   Y -4.073112   Z -8.450247
Rotation   X 0          Y 60          Z 0
Scale      X 0.7        Y 0.8         Z 1

▶ Menu Item Flyer (Mesh Filter)
▼ ☑ Mesh Renderer
▶ Lighting
▼ Materials
   Size          1
   Element 0     image2

▼ ☑ Mesh Collider
Convex          ☐
   Inflate Mesh  ☐
      Skin Width 0.01
   Is Trigger    ☐
Material        None (Physic Material)
Mesh            MenuItemFlyer

▼ Event Trigger (Script)
Pointer Click (BaseEventData)
Runtime Only    GameObject.SetActive
Image2          ☑
Runtime Only    GvrPointerPhysicsRaycaster.enabled
Main Camera     ☐
```

Figure 6.10: Each menu has an event trigger script which is automatically activated when the menu is clicked.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.VR;
using UnityEngine.UI;
public class ImageDisplay : MonoBehaviour
{
    public GameObject VRCam;
    void Update()
    {
        if (gameObject.activeInHierarchy == true)
```

```
            {
                VRSettings.enabled = false;
            }
            if ((Input.GetButton("Fire1") || Input.touchCount
                ↪ > 0) && gameObject.activeInHierarchy ==
                ↪ true)
            {
                gameObject.SetActive(false);
                VRSettings.enabled = true;
                VRCam.GetComponent<GvrPointerPhysicsRaycaster
                    ↪ >().enabled = true;
            }
        }
}
```

Listing 6.1: content of the custom script that is triggered when the menu is clicked

With this approach, the demo finally works in Android environment, which concludes the development of the first demo of this project.

Performance test of the first demo

- **Test methodology**

  Since this is the first demo of this project, the performance test was conducted in private with a small number of testers. The content of the test was also very simple. It was only about moving the camera to look at the stereoscopic photos, which lasted around 5 minutes. The testers consisted of the researcher, the supervisor and two other master students in AUT.

When the test ended, each tester was required to answer only one question: “Are you be able to see the 3D effect on those stereoscopic photos?”

- **Test result**

  After the test, four out of four testers confirmed that they could see the 3D effect. However, the supervisor and one master student pointed out an issue in this demo. Before this test, these two testers had experienced 3D effect on stereoscopic photos by tricking their eyes without the use of VR headset. They realised that the position of two images in each stereoscopic photo were placed in reserve, which resulted as “depth” 3D effect instead of the correct “pop-out” 3D effect. Because of this, the next demo has to address this issue before moving on to any new content of the project.

6.2.2 Second Demo – August 6th, 2017

The purpose of the second demo is to address an issue that is still present in the first demo, which is incorrect 3D effect on stereoscopic photos. Moreover, there is also a new feature introduced in this one, which serves as a solution for a new issue occurred during the development.

Display different image on two cameras

In the previous demo, the full-screen display of stereoscopic photos is set by switching from standard mode to VR mode through a custom script. With this approach, the first demo can make use of any stereoscopic photos available on the Internet. However, these online materials are not designed exclusively for VR headset. Therefore, it is possible that the two images on these stereoscopic photos are placed incorrectly: right image on the left side, left image on the right side. Due to this characteristic, a new display mechanism was introduced.

Instead of switching to a single camera, each camera of the VR mode represents each side of the stereoscopic photos. Because of this, each stereoscopic photo is required to be split into two photos acting as left side and right side of the original photo. According to the project goal, this process should be done automatically. However, due to the focus of fixing address the stereoscopic photo display problem in this demo, the researcher manually performed this process.

- **First try**

  The researcher used a simple method. Two new culling mask profiles were created with the name as LeftEye, and RightEye. Each culling mask profile was added to its related camera culling mask. After that, left image and right image of the stereoscopic photos were matched with the relevant culling mask, and placed in the same position in the application environment. Thanks to different culling mask profile, while each camera displayed the same image of the menu and the surrounding environment, they received different side of image of the stereoscopic photos.

  However, there was a problem. With the release of version 0.8, Google has added several changes to their VR SDK, including removing two cameras as standalone game objects. These cameras are incorporated in Unity native VR camera. Therefore, they only appear after the application starts. This update prevents these cameras culling mask from being altered in Unity standard UI.

  In order to change these values, steffkelsey (xdf103, 2016) has proposed an idea of creating a custom script to automatically do this process.

```
using System.Collections;
using UnityEngine;


public class EyeLayerCull : MonoBehaviour
```

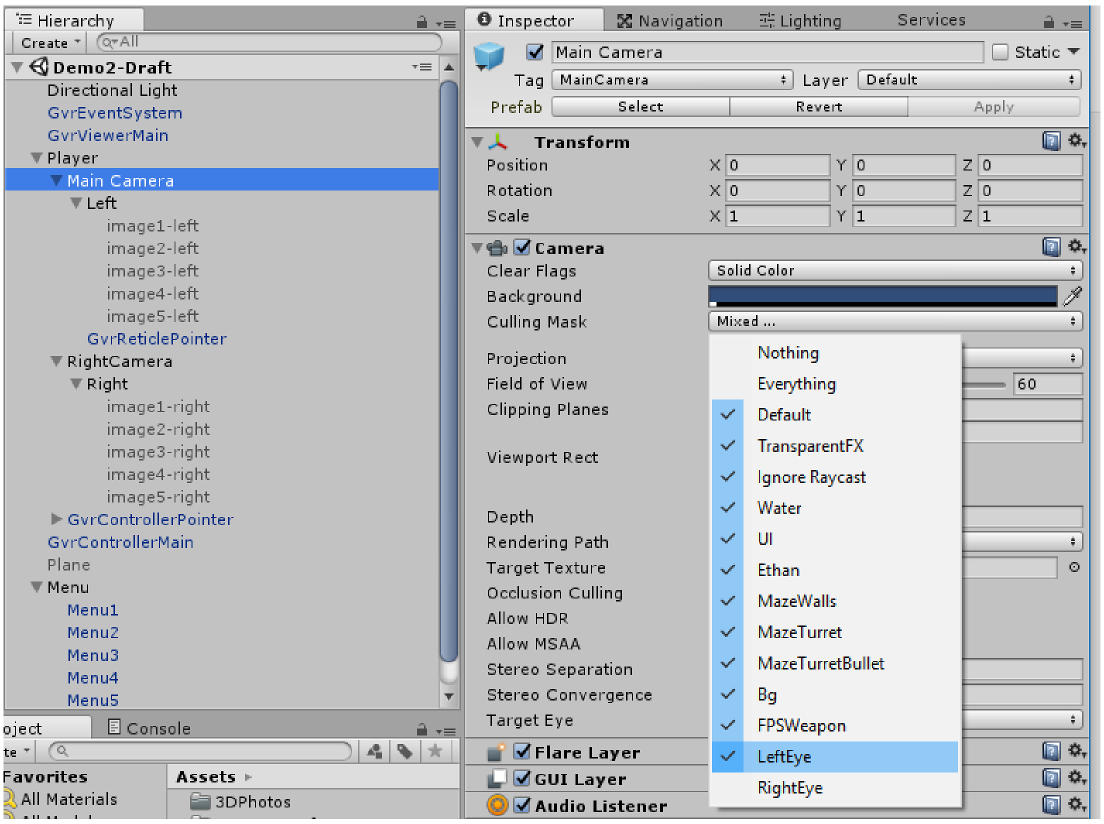
```
{
    public LayerMask LeftEyeMask;
    public LayerMask RightEyeMask;

    void Start()
    {
        StartCoroutine(LateStart());
    }

    protected IEnumerator LateStart()
    {
        yield return new WaitWhile(() => GvrViewer.
          ↪ Controller == null);
        yield return new WaitWhile(() => GvrViewer.
          ↪ Controller.Eyes.Length < 2);
        foreach (GvrEye gvrEye in GvrViewer.
          ↪ Controller.Eyes)
        {
            switch (gvrEye.eye)
            {
                case GvrViewer.Eye.Left:
                    gvrEye.toggleCullingMask =
                      ↪ LeftEyeMask;
                    break;
                case GvrViewer.Eye.Right:
                    gvrEye.toggleCullingMask =
                      ↪ RightEyeMask;
                    break;
                }
            }
            GvrViewer.Controller.UpdateStereoValues();
        }
}
```

Listing 6.2: Content of the custom script for changing culling mask value of the cameras

With the script attached to one game object that appeared when the demo started, the researcher could change the culling mask profile in each camera. The preview mode showed that each camera received a different side of the photos. Unfortunately, this custom script was created in September 2016. There has been several updates of Unity library, including a removal of UpdateStereoValues() function usage. Due to this change, while Unity preview mode accepted this function, exporting the demo to Android environment deemed the function as incorrect.

- **The solution**

  Since both left and right cameras from Google VR SDK are generated automatically by the start of the demo, it is impossible to change any value of them. Instead of doing that, a decision of adapting Unity native VR feature to the demo was made. This Unity feature allows developers to specify which user eye the camera is targeting when developing a VR application (Unity Technologies, 2017d).



```
Demo2-Draft
  Directional Light
  GvrEventSystem
  GvrViewerMain
  Player
    Main Camera
      Left
        image1-left
        image2-left
        image3-left
        image4-left
        image5-left
      GvrReticlePointer
    RightCamera
      Right
        image1-right
        image2-right
        image3-right
        image4-right
        image5-right
    GvrControllerPointer
  GvrControllerMain
  Plane
  Menu
    Menu1
    Menu2
    Menu3
    Menu4
    Menu5
```

```
Main Camera                         Static
Tag  MainCamera        Layer  Default
Prefab  Select  Revert  Apply
Transform
  Position   X 0  Y 0  Z 0
  Rotation   X 0  Y 0  Z 0
  Scale      X 1  Y 1  Z 1
Camera
  Clear Flags        Solid Color
  Background
  Culling Mask       Mixed ...
  Projection
  Field of View      60
  Clipping Planes
  Viewport Rect
  Depth
  Rendering Path
  Target Texture
  Occlusion Culling
  Allow HDR
  Allow MSAA
  Stereo Separation
  Stereo Convergence
  Target Eye
Flare Layer
GUI Layer
Audio Listener
```

```
  Nothing
  Everything
✓ Default
✓ TransparentFX
✓ Ignore Raycast
✓ Water
✓ UI
✓ Ethan
✓ MazeWalls
✓ MazeTurret
✓ MazeTurretBullet
✓ Bg
✓ FPSWeapon
✓ LeftEye
  RightEye
```

Figure 6.11: Two camera objects were added to the demo in this approach.

Based on this feature, two camera objects with the same characteristics were added to the demo. Each camera had different target eye setting and different culling mask profile, which matched with each image side of the stereoscopic photos. To be more precise, both cameras could receive the same image of all objects in the demo except the one from the stereoscopic photos. While the creation of these cameras caused the auto-generated cameras unusable, all of the Google custom scripts attached to the demo were still needed. This was because these scripts also gave users the control ability of the application such as camera movement, object interaction. With this approach, each camera can finally show different image based on the culling mask profile. The feature also works in Android environment.

Stereoscopic photos on dual camera

Due to the change of the camera system, the researcher realised that the old stereoscopic photos display mechanism approach in section **First demo – July 2nd, 2017** was not effective anymore. For Unity UI Canvas to display in VR mode, the canvas has to have render mode switched to World Space. In this mode, any object belongs to the canvas become a game object with its own position, rotation, and scaling setting. Because of this characteristic, the researcher had to attach the the canvas to the cameras. This canvas also needed to have a specific position, rotation, and scaling setting that could fully cover all display area of the cameras.

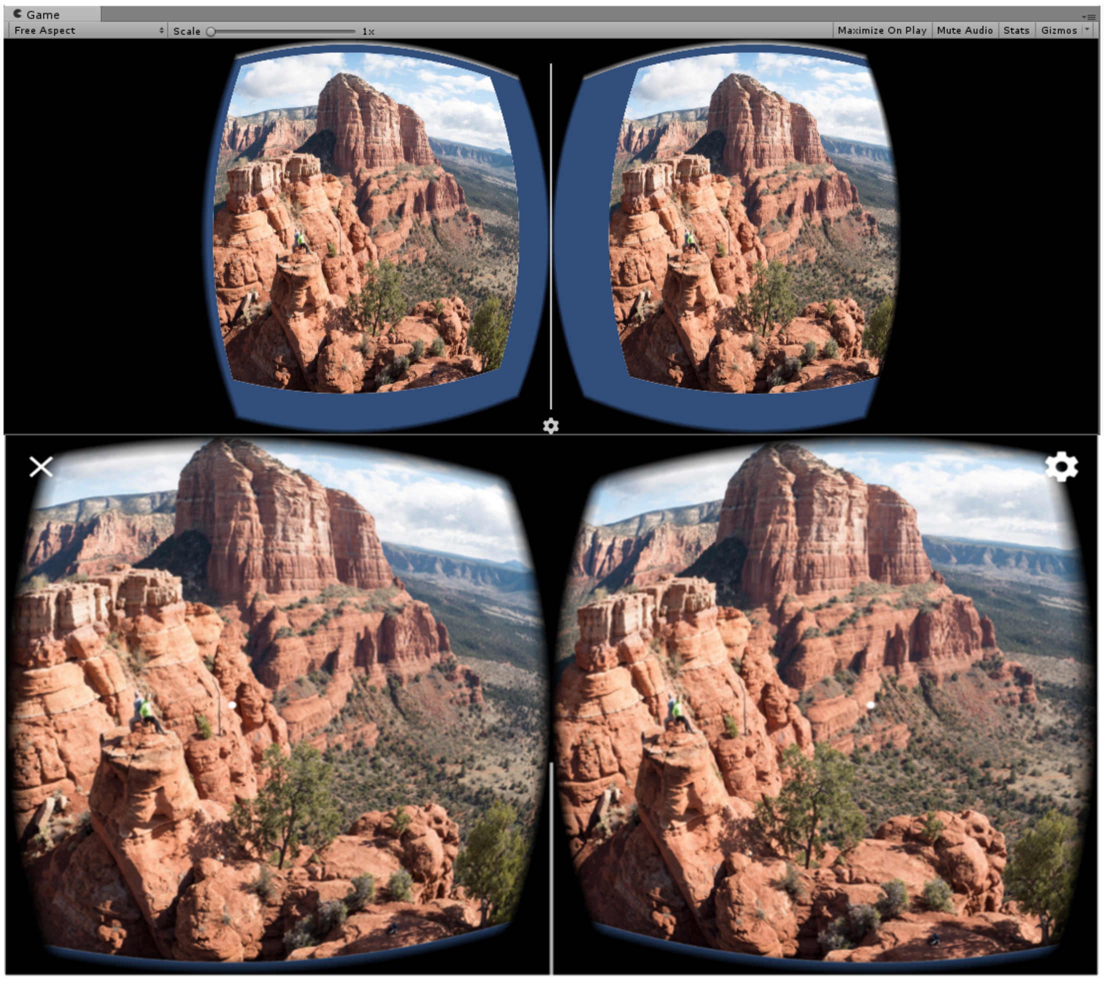


Figure 6.12: The camera display area in Unity preview mode(top) has different characteristics from the one in Android environment (below).

There was an issue in this process. The camera display area in Unity preview mode and in Android environment did not have the same characteristics. Therefore, a canvas that fully covered the camera display area in Unity preview mode would be out of place in Android environment and vice versa. This required the researcher to fix this issue through trial & error.

Performance test of the second demo

- **Test methodology**

  When compared with the first demo, there is no new feature or content added to this demo. The only difference is the back-end. Because of this, the performance test was the same as the previous one. The participants of this test were downsized to only 2 persons (the researcher and the supervisor) to ensure that the process could be completed as soon as possible.

  Besides the current resource of Mi VR Play and Xiaomi Mi4C, there was one addition equipment to this test: a VR headset named RITECH VMAX

- **Test result**

  When the test was performed in Xiaomi Mi VR Play headset, the 3D effect displayed correctly. However, switching the headset to RITECH VMAX caused both left, and right images appeared out of the camera display area. This was due to the mismatch of lens display area between Mi VR Play, and VMAX. While Google Cardboard design give several layouts for capacitive button location, shape of the outer-case, and material; It does not specific that the headset creators must follow closely on the template (Google, 2015c). This is why each headset has its own lens distance, field-of-view, which leads to the need of a viewer profile system (Google, 2015d).

This system allows the headset creators to create their own headset specification profile so that any Cardboard-compatible application can optimise its camera display area to match with the profile. Since the researcher optimised position, rotation, scaling settings of the stereoscopic photos for Mi VR Play headset, the images appeared out of place when the demo was tested when using VMAX headset. In addition, the change of stereoscopic photos mechanism caused the camera pointer (white dot) appeared during the test, which was distracting for the viewers.

6.2.3 Third Demo – September 3rd, 2017

With the image misplacement issue from the second demo in mind, the main goal of this demo is to fix it as well as to implement new feature if no new issue occur and halt the development.

New stereoscopic photos on dual camera approach

In the old approach, when a stereoscopic photo menu is clicked, left image and right image of that photo will appear and attach to the related camera object. Because the old display method required different position and rotation settings of these two images for different headset to cover whole camera display area, another effective approach was needed.

In the new approach, the researcher moved both cameras to a dimly closed 3D gallery room, where a size-enhanced version of the clicked stereoscopic photo were virtually hanged in front of the cameras. Users could freely move their head to explore the 3D photo as well as the area as if they were in a real museum.

Thanks to the simplicity of this approach, this feature was completed with just a simple script of automatically performing two steps: (1) move both cameras to the 3D gallery room, and (2) move the UI image object to the same room also.

Automatic camera rotation

While the side images could finally cover whole camera display area, a new issue happened. Since the rotation settings of the 3D gallery room and two side images were fixed, users had to frequently rotate their head to view the images. Moreover, when getting back to the menu screen, their head rotation settings were not changed, which required them to move their head again to see the menu centre. This issue not only annoyed the users, but also had a high chance of causing disorientation problem. Therefore, both cameras rotation settings were required to be constantly changed so that they were always in front of both the clicked menu and the side images hanged on the wall of the 3D gallery room.

- **First try: Camera rotation adjust**

  Since both cameras had their own rotation settings, creating a custom script to adjust them was the first thing to try. This script automatically changed the camera rotation settings to default settings if either any menu was clicked or the users got back to the menu screen.

  Unfortunately, Google VR SDK does not allow developers to change user camera rotation settings during runtime. According to Google, the reason is that it was essential for users to actively control their virtual head all time to enhance experience, and avoid discomforting (Google, 2015b).

- **The solution: Game objects rotation adjust**

  With the inability of changing the camera rotation settings, the researcher decided to use a more time-consuming approach. Instead of rotating the users' camera, all other game objects in the demo were rotated to match with the rotation settings of both cameras. This process was performed through a custom script, which divided into two parts:

  In the first part, when the menu was clicked; the rotation settings of the left camera would be copied, and pasted to the rotation settings of the side images as well as the 3D gallery room. Then the cameras and side images would be moved to this 3D gallery room.

  In the second part, when users got back to the menu screen, the difference between the centre menu rotation settings and the clicked menu rotation settings was calculated into a specific value. This specific rotation value was used to rotate whole menu screen so that the users' cameras would be in front the menu that was clicked recently.

  Since the menu and the UI had different rotation settings, performing these two parts would require many unnecessary steps and resource. A new menu structure that could fuse with the side images was required.

  Not only supporting 3D game development, Unity also allows developers to create two-dimension games. Therefore, the engine has many features and programming library for this development style. One of the most essential feature is an ability to import an image file to the project as a sprite game object (Unity Technologies, 2015a).

  Based on this technique, all of the old menu using 3D model from Unity VR package and the UI were replaced with the sprites of the left-sided image of all of the available stereoscopic photos. These new game objects also included their own custom-made Box Collider so that the cardboard VR pointer could interact with them. The sprite of the right-sided image was attached to the correspond left-sided image object as a child game object.

After this new menu object structure was assembled, a new custom script was attached to each menu. This script not only had the stereoscopic photo display mechanism in section **6.2.1 First demo – July 2nd, 2017**, but also included a new camera rotation function explained in section "Camera rotation mechanism".

Moreover, the script was also optimised to improve the demo performance. To be more detail, in the first and second demo, the photo display mechanism function was called at the start the demo, and remained constantly until the demo was closed. The new approach separated this function into two trigger part: image activation part and image deactivation part. The activation part only acted as a trigger when the script was enabled (a menu was clicked). The deactivation part only begun when the script was disabled (users got back to the menu screen).

With this approach, the camera was always in front of both the images hanged on the wall in the 3D gallery room and the clicked menu in the menu screen. There was also a new feature presented thanks to the new menu structure. All of the menu also displayed 3D effect for the users to enjoy. Moreover, the size of the demo was also reduced noticeably thanks to the removal of 3D model of the old menu.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Demo3_ImageDisplay : MonoBehaviour
{
```

```
GameObject VRCam, VRRoom, VRMenu, RightCam,
    ↪ Pointer;

private void Awake()
{
    VRCam = GameObject.Find("Player");
    VRRoom = GameObject.Find("Cube");
    VRMenu = GameObject.Find("Plane");
    RightCam = GameObject.Find("Main␣Camera");
    Pointer = GameObject.Find("GvrReticlePointer"
        ↪ );
}

private void OnEnable()
{
    Pointer.SetActive(false);
    VRCam.transform.position = new Vector3(VRCam.
        ↪ transform.position.x, -7, VRCam.
        ↪ transform.position.z);
    VRRoom.transform.rotation = Quaternion.Euler
        ↪ (0, gameObject.transform.rotation.
        ↪ eulerAngles.y, 0);
    gameObject.transform.localPosition = new
        ↪ Vector3(gameObject.transform.
        ↪ localPosition.x, -8, gameObject.
        ↪ transform.localPosition.z);
    gameObject.transform.localScale = gameObject.
        ↪ transform.localScale * 4;
}

void Update()
{
    if ((Input.GetButton("Fire1") || Input.
        ↪ touchCount > 0) && gameObject.transform.
        ↪ localPosition.y != 0)
    {
        gameObject.GetComponent<
            ↪ Demo3_ImageDisplay>().enabled =
            ↪ false;
    }
}

private void OnDisable()
{
    gameObject.transform.localPosition = new
        ↪ Vector3(gameObject.transform.
        ↪ localPosition.x, 0, gameObject.transform
        ↪ .localPosition.z);
    gameObject.transform.localScale = gameObject.
        ↪ transform.localScale / 4;
        VRMenu.transform.rotation = Quaternion.Euler
          ↪ (0, RightCam.transform.rotation.
          ↪ eulerAngles.y - gameObject.transform.
          ↪ localRotation.eulerAngles.y, 0);
        VRCam.transform.position = new Vector3(VRCam.
          ↪ transform.position.x, 1.4f, VRCam.
          ↪ transform.position.z);
        Pointer.SetActive(true);
    }
}
```

Listing 6.3: Content of the optimised script attached to each new menu object

Performance test of the third demo

While this demo only fixed the issue from the previous one, there was a whole new menu structure and custom script used to accomplish this goal. Because of this, the content of this test was the same as the second demo test. There was also no new headset device as well as no addition tester.

- **Test result**

  The first impression was that each menu could display 3D effect correctly, which improved the user immersion. When clicked on one menu, the users could see the enhanced-size stereoscopic photo right away without adjusting their head rotation. In order to get back to the menu screen, a simple capacitive button trigger would put the users' view right back to be in front of the menu they had clicked previously.

Yet again, the introduction to a new script and a new menu structure led to an occurrence of a new issue. While all image files could be imported to Unity, it was impossible for developers to input exact value for their resolution. The only option available in Unity front end UI was scaling, which had the value based on the image real resolution. Because of this, if the imported images had different resolution or aspect ratio, the menu objects in the demo would appear uneven. An image scaling mechanism was required for the next demo.

There was also a small issue with the 3D gallery room. The room was unable to contain the whole display area of the magnified images. During this performance test, the culprit was unknown to the researcher. Therefore, more research was planned to be performed in the next demo.

6.2.4 Fourth Demo - October 1st, 2017

Besides fixing any issue from the previous demo, this demo aims to implement two new features: image object scaling, and online image import. There are also some small tweaks to the 3D gallery room and the custom scripts that were developed in section **6.2.3 Third Demo – September 3rd, 2017**.

Image object scaling

When an image is imported to Unity as a sprite, its resolution can be changed based on the image compression technique and image size threshold that developers input. After that, resolution and aspect ratio of the sprite are converted to object scale value and object scale ratio respectively (Unity Technologies, 2015a). A uniform scale value of 1 always represent as the default resolution and aspect ratio of the sprite. Due too this, if both sprites have the same scale value, the difference in either resolution or aspect ratio will make them appear uneven in Unity (Unity Technologies, 2017b).

- **First try: Create multiple Unity UI**

  In this approach, the researcher utilised the stereoscopic photo display mechanism shown in section **6.2.1 First demo – July 2nd, 2017**. Any image object attached to a new Unity UI object had an option to stretch its resolution to fit with the UI resolution. By attaching and stretching each image object to a UI object with a fixed resolution, all image objects could appear evenly even if they had different resolution or aspect ratio.

  Although this technique could fulfill the objective, the researcher still considered it to be not an effective one. The first issue was the unnecessary use of too many UI objects. For an example, if there were 10 image objects appeared in the demo, 10 UI objects were required to be added and have an image attached to them. Moreover, since each stereoscopic photo had two side images attached to each other (as shown in section **6.2.3 Third Demo – September 3rd, 2017**), the addition of UI objects would require them to be also attached to these side images. This made the object structure complicated. The second issue was the unknown of specific resolution value. All UI objects had a relative scale value with a uniform default value of 1. Because of this, the size of these objects could only be changed based on the researcher observation, not calculation.

- **The solution: Scaling images by using bounding boxes**

  In Unity engine, each 3D object has its own axis-aligned bounding box that is used for object collision detection. This bounding box always matches with the object size and vice versa. In addition, instead of having relative object scale settings like the object, the axis-aligned bounding box has a specific scale value hidden from Unity development tool UI. Thanks to this feature, the bounding box can be used as a scaling value be retrieved and set by developers (Unity Technologies, 2017c).

The first step of this approach was to pick a standard Unity image object and ensured that all image objects related to this resizing feature had the same scale value. Since this image object type was also used as a menu object, the default scale was set to *0.25 : 0.25 : 0.25* in *x : y : z* axis. After that, an image object would be used as a standard image object with standard resolution and aspect ratio. All other image objects, including menu and stereoscopic photos, would be resized to be at the exact size as the standard image. While it was possible to pick any image object already available in the demo as a standard image, it could cause a scaling issue later when online image import feature was added. Therefore, a new image object was added as a standard image object. This object had the renderer component disabled so that only developers could only see it in Unity developer mode.

The second step was to get the bounding size value from both the standard image object and the image object which is needed resizing. These values were used to calculate a ratio that converted the image object's size to match with the standard image object's size. Due to the characteristic of axis-aligned bounding box of having its size dependent on the rotation, all image objects were required to be at default rotation value (*0 : 0 : 0* in *x : y : z* axis) before the calculation began.

After the size conversion completed, all image objects had their rotation settings reverted to their original value. Both these steps were performed by a custom script attached to every image object that was used as a menu and a side image of the stereoscopic photos.

With the use of this technique, not only was there no addition game object required, but also could any specific resolution and aspect ratio be converted into the same size as the standard image object.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Demo4_ImageScale : MonoBehaviour {

    Vector3 SizeRatio, StandardBound, CurrentBound;
    GameObject StandardPhoto;

    private void OnEnable()
    {
        Vector3 placeholder = gameObject.transform.
            ↪ rotation.eulerAngles;

        gameObject.transform.rotation = Quaternion.
            ↪ Euler(0, 0, 0);

        StandardPhoto = GameObject.Find("image-
            ↪ template");
        StandardBound = StandardPhoto.GetComponent<
            ↪ Renderer>().bounds.size;
        CurrentBound = gameObject.GetComponent<
            ↪ Renderer>().bounds.size;
        SizeRatio = new Vector3(StandardBound.x /
            ↪ CurrentBound.x, StandardBound.y /
            ↪ CurrentBound.y, StandardBound.z /
            ↪ CurrentBound.z);

        gameObject.transform.localScale = new Vector3
            ↪ (gameObject.transform.localScale.x *
            ↪ SizeRatio.x, gameObject.transform.
            ↪ localScale.y * SizeRatio.y, gameObject.
            ↪ transform.localScale.z * SizeRatio.z);

        gameObject.GetComponent<BoxCollider>().size =
            ↪  new Vector3(StandardPhoto.GetComponent<
            ↪ BoxCollider>().size.x / SizeRatio.x,
            ↪ StandardPhoto.GetComponent<BoxCollider
            ↪ >().size.y / SizeRatio.y, 0.1f);

        gameObject.transform.rotation = Quaternion.
            ↪ Euler(placeholder.x, placeholder.y,
            ↪ placeholder.z);
    }
}
```

Listing 6.4: Content of the attached script that automatically scaled the image object size to match with the standard image object size

Online image import process

Before this demo, all of the image objects had been manually added. These objects had been served as placeholders for the researcher to implement several features such as 3D image display, image scaling. These placeholders also met a requirement to appear as any stereoscopic photo for users to view. Therefore, it was possible for the researcher to add more image objects with the same characteristic of these placeholders to fulfill the project goal. However, frequent addition of these objects required the researcher to painstakingly searched the image source, checked if the object included the necessary components and scripts. Furthermore, the stereoscopic photo library had to be big enough for the users to enjoy viewing them in several times, thus could cause a dramatic increase of the application size if the image object was added manually.

One solution for this problem was to create a connection between the application and the online database. With the help of the supervisor in setting up an online database, the researcher only needed to create the online connection through a custom Unity script. This script contained two processing parts: download image from the online database and convert the downloaded image to a Unity game object.

The first part began with the importation of an index file from the online database. This file contained all the name of image files in the database that matched the characteristic of a stereoscopic photo. The content of this file would then be converted into an array of text serving as a guideline for the script to download the requested image files. Since Unity allows parallel runtime of functions, it was important that all other functions were halted until the download of this index file completed.

In the second part, the script created image objects using Instantiate, a Unity function that creates a game object based on a prefab. The prefab was based on the placeholder object structure so new game objects did not need any addition of custom scripts or components. After the new game objects were created, their texture was replaced with the texture of the corresponding downloaded image. Finally, the position settings and rotation settings of these objects were set to match with the intended position, and rotation. All attached custom scripts were also enabled to perform their intended functionality. Only one placeholder object was to be kept for this script operation, the others was removed out of the demo.

While the script could be attached to any game object available in the demo environment for it to work, the best location was an empty game object that have all other image objects as its child objects. This would help the **Camera rotation mechanism** to work correctly without further action.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Demo4_ImageImport : MonoBehaviour {
    public List<string> lines;
    public GameObject image;
    private string urll, urlr;
    private Vector3[] imagepos =
    {
        new Vector3(-4.8f,0,-4.7f),
        new Vector3(-3.5f,0,-1.3f),
        new Vector3(0,0,0),
        new Vector3(3.5f,0,-1.3f),
        new Vector3(4.8f,0,-4.7f)
    };
```

```
IEnumerator Start()
    {
        yield return StartCoroutine(GetIndex("http://www.
          ↪ ivs.auckland.ac.nz/stereogallery/w3/list.txt
          ↪ "));

        int x = 0;

        GameObject photo;

        for(int i = 0; i<=4; i++)
        {
            urll = new System.Uri("http://www.ivs.
              ↪ auckland.ac.nz/stereogallery/w3/" +
              ↪ lines[x]).AbsoluteUri;
            urlr = new System.Uri("http://www.ivs.
              ↪ auckland.ac.nz/stereogallery/w3/" +
              ↪ lines[x + 1]).AbsoluteUri;

            photo = CreateImage(i);

            yield return StartCoroutine(GetImage(urll,
              ↪ urlr, photo));

            x = x + 2;
        }
    }

IEnumerator GetIndex (string url)
    {
        WWW path = new WWW(url);
        yield return path;
        lines = new List<string>(path.text.Split(new
            ↪ string[] { "\n" }, System.StringSplitOptions
            ↪ .RemoveEmptyEntries));
    }

    GameObject CreateImage(int i)
    {
        var temp = Instantiate(image, transform.
            ↪ localPosition + imagepos[i], Quaternion.
            ↪ Euler(0, -90f+45*i, 0), transform);
        temp.name = "Image"+i+"-l";
        temp.transform.GetChild(0).name = "Image" + i + "
            ↪ -r";
        return temp;
    }

    IEnumerator GetImage(string urll, string urlr,
        ↪ GameObject photo)
    {
        // Start a download of the given URL
```

```
var left = new WWW(url1);
// wait until the download is done
yield return left;
// Create a texture in DXT1 format
Texture2D texture = new Texture2D(left.texture.
    ↪ width, left.texture.height, TextureFormat.
    ↪ DXT1, false);
// assign the downloaded image to sprite
left.LoadImageIntoTexture(texture);
Rect rec = new Rect(0, 0, left.texture.width,
    ↪ left.texture.height);
Sprite spriteToUse = Sprite.Create(texture, rec,
    ↪ new Vector2(0.5f, 0.5f), 100);

photo.GetComponent<SpriteRenderer>().sprite =
    ↪ spriteToUse;

var right = new WWW(urlr);
yield return right;
Texture2D texture2 = new Texture2D(right.texture.
    ↪ width, right.texture.height, TextureFormat.
    ↪ DXT1, false);
right.LoadImageIntoTexture(texture2);
rec = new Rect(0, 0, right.texture.width, right.
    ↪ texture.height);
```

```
        spriteToUse = Sprite.Create(texture2, rec, new
            ↪ Vector2(0.5f, 0.5f), 100);

        photo.transform.GetChild(0).GetComponent<
            ↪ SpriteRenderer>().sprite = spriteToUse;

        photo.GetComponent<Demo4_ImageScale>().enabled =
            ↪ true;

        left.Dispose();
        left = null;

        right.Dispose();
        right = null;
    }
}
```

Listing 6.5: Content of the script used to import image files from the online database

New 3D gallery room structure

In the previous demo, the 3D gallery room was created by importing a 3D room model from Google VR SDK. Since this model was not designed for a photo viewing purpose, there were some textures that blocked some parts of the stereoscopic photo objects. Moreover, the researcher did not prefer to borrow any model or texture from third-party provider. Thanks to the native 3D model library of Unity, the transition to a new 3D room structure was simple and fast.

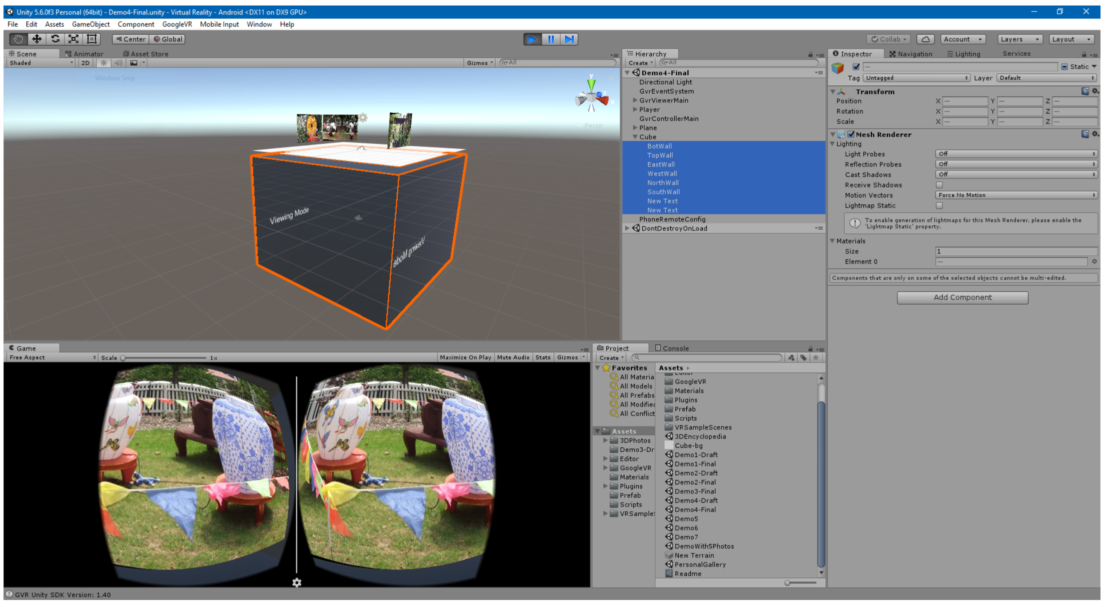


Figure 6.13: The new 3D room was created with the composition of multiple 3D Unity-default objects.

There were 3 steps in this creation. Firstly, six 3D cube objects were imported from Unity library. Secondly, these objects received a specific scale settings that transformed them into 3D flat panel objects. After that, they were used to assemble a virtual 3D room. The vital advantage of this new room structure was that there was no obstacle inside the room that could block users' view.

Performance test of the fourth demo

With the introduction of a new feature related to Internet connectivity, there was an addition content for this test. Besides using the same format as the previous test, this test required the participants to use the demo in two scenarios: (1) with Internet connected and (2) without Internet.

The test also used the same resource and participants as in the previous test.

- **Test result**

  Thanks to the image scaling script, even if the imported image objects had different original resolution or aspect ratio, all of them appeared equally to the testers. In the environment that had Internet connected, the demo was freeze in around 2 seconds at the beginning. This was due to the image downloading process of the application. However, it was still unclear if other smartphones would have the same symptom when using this demo. In the environment that had no Internet, there was no image or 3D object appeared in the demo except a white surface. This was normal due to the absent of an error panel.

  There were also two small issues occurred during this test. In the 3D gallery room, the images were not fully inside the room, a small part of them was left outside. This was because of the way the automatic camera rotation function handled. The camera pointer was also not in the centre of the display area.

  With these issues, the next demo would focus on resolving them and adding more features if possible.

6.2.5 Fifth Demo - October 22nd, 2017

New 3D images placement algorithm

Upon investigation of the misplacement issue, the culprit was identified inside the current images placement script. At the time, when users clicked on an image, only the rotation settings of the image and the 3D gallery room were changed to match with the camera rotation value. The problem with this approach was that by changing the rotation settings, the position value of the object was required to be changed also in order to maintain the object correct position. While it was possible to change the position value of every object through the script, completing this task would take a lot of time since every image on the demo had different rotation settings, thus different position value.

The best solution for this problem was creating one single image placeholder object for all of the available images on the demo. When the users clicked on an image object, this image's position and rotation settings would be matched with the placeholder object's settings. After that, the rotation and position settings of the image object would be changed again to match with the 3D room and camera settings. When the users got back to the menu screen, the clicked image's position and rotation value would be reverted to its original value. With this approach, the script was only needed to change the position and rotation settings of a single image object to match with the 3D room and camera settings. Therefore, only a small addition content was needed for the current script.

```
    using UnityEngine;

public class Demo5_ImageDisplay : MonoBehaviour
{
    GameObject VRCam, VRRoom, VRMenu, RightCam, Pointer;
    Vector3 ObjectPos;
    Quaternion ObjectRot;

    private void Awake()
    {
                    ...
        ObjectPos = gameObject.transform.localPosition;
        ObjectRot = gameObject.transform.localRotation;
    }
    private void OnEnable()
    {
        GameObject ImageTemplate = GameObject.Find("image
            ↪ -template");
        gameObject.transform.position = ImageTemplate.
            ↪ transform.position;
        gameObject.transform.rotation = ImageTemplate.
            ↪ transform.rotation;
                ...
    }

    void Update()
    {
                ...
    }

    private void OnDisable()
    {
        gameObject.transform.localPosition = ObjectPos;
        gameObject.transform.localRotation = ObjectRot;
                ...
    }
}
```

Listing 6.6: Addition content that was added to ImageDisplay script, fixing the misplacement issue of the images in the 3D room

New menu screen – category selection script

Since there were multiple different image categories, the current menu screen was required to be changed. The new menu screen only showed categories for users to choose. Each category would be demonstrated by showing an image related to it. For examples, an image of ring-tailed lemurs represented the “Animal” category, while an image of an elephant statue represented the “Statue” category. The researcher added a new script to each category so that by clicking on these categories, the system would automatically download five images belonging to these categories. Moreover, the script also hid all of the category images and brought up images related to the clicked category.

- **First try: setting unique number for each category**

  It was important that each category had a unique code so that the system could identify which category users picked. Initially, the researcher tried to set a unique number for each category. A new public variable was created that served as a code storage for these categories. The system would base on this variable to download suitable images from the database.

  While there was no performance issue with this approach, it was not suitable for the future development. The reason for this was that each unique number had to be assigned to a category manually; thus, a category code database was needed for the system to look up. If there was a change of the category name in the image database, the unique number was required to be changed to match with the new update. Moreover, it was not optimised for the script to have too many public variables; therefore, an approach that did not need any new variable was preferred.

- **The solution: identify a category through an image name**

  When an image is downloaded to the application, its properties (name, size, resolution) is stored in the system temporary. It is possible to change the name of the downloaded image object to match with the original name. This characteristic helped the researcher achieve the most optimal approach for this function: identify categories through the image object name.

  In the event of starting the demo, the system checked the database index file and downloaded the category images automatically. These images contained category names that they were related to. When users clicked on any category image object, the system would look up to the object name to identify the category that the users were interested in. With this approach, any new update of the image database would not require any change of the script. It was because as long as these category images have the name matching with the category name, the system always found the correct category that the users chose.

```
    using UnityEngine;

public class Demo5_MenuScript : MonoBehaviour {
    Demo5_ImageImport ImageImportScript;

    void Awake ()
    {
        ImageImportScript = GameObject.Find("Image").
            ↪ GetComponent<Demo5_ImageImport>();
    }

    private void OnEnable()
```

```
        {
            GameObject.Find("Image").GetComponent<
                ↪ Demo5_ImageImport>().MenuNum = "https://
                ↪ cerv.aut.ac.nz/vr/" + gameObject.
                ↪ GetComponent<SpriteRenderer>().sprite.
                ↪ name + "/";

            GameObject.Find("Image").transform.localScale
                ↪  = new Vector3(0.7f, 1, 0.7f);
            GameObject.Find("Menu").transform.localScale
                ↪ = new Vector3(0,0,0);

            ImageImportScript.InMenu = true;
            ImageImportScript.enabled = true;

            gameObject.GetComponent<Demo5_MenuScript>().
                ↪ enabled = false;
        }
}
```

Listing 6.7: Content of the script attached to each category image, helping the system to identify the clicked category

New control panel

In the previous demo, users could only see five images that were set by the developers. Thus, the content was not large enough for a good 3D experience. Since every image had to go through many processes in order to appear on the screen, adding too many images on the same screen would be not suitable, especially for the smartphone environment. Instead a control panel was added for the users to browse multiple images. By clicking a simple button, the users could move to the next five images or move back to the previous ones. The control panel was divided into three buttons: (1) Backward, (2) Forward, and (3) Return.

The Return button contained a very simple script, which automatically got the users back to the menu screen when being clicked. This was done by bringing up all category images while hiding the previous five images.

To allow the Backward and Forward button to browse the image database, a new public variable named "ImageNum" was created. This variable contained the ordinal number of the first image that was downloaded to the demo. If the users clicked on the Forward button, the value of "ImageNum" would be added to the ordinal number of the last downloaded image + 1. For an example, when the first five stereoscopic photos were downloaded, the value would be "10"; when the next five photos were downloaded, the value would be "20". Similarly, a click on the Backward button would subtract the "ImageNum" value to the ordinal number of the last downloaded image + 1.

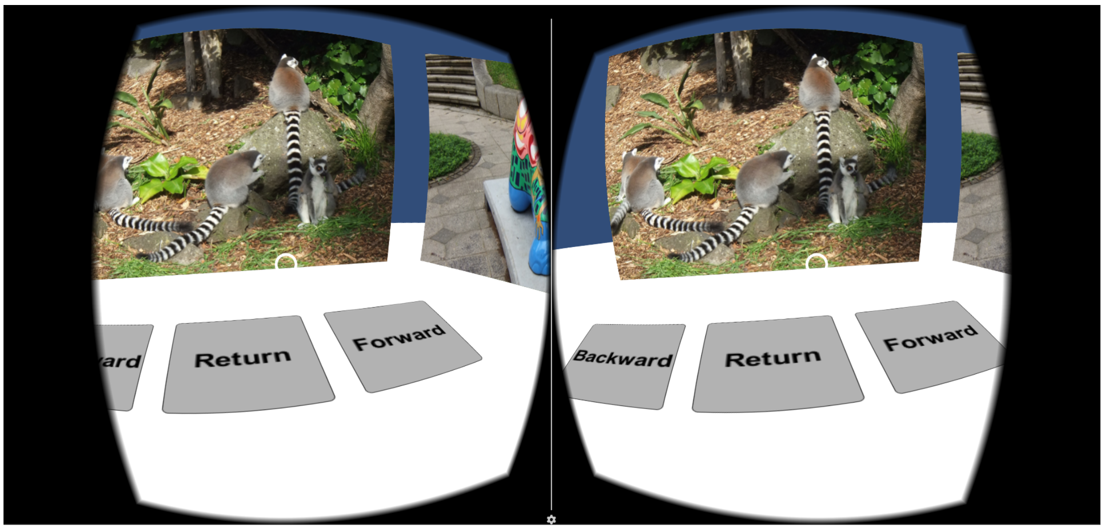


Figure 6.14: The control panel is located at the bottom of the screen, providing users an ability to browse images.

```
    using UnityEngine;
        using UnityEngine.UI;

public class Demo5_ButtonScript : MonoBehaviour {
    Button BackwardBtn, ForwardBtn, ReturnBtn;
    Demo5_ImageImport ImageImportScript;
    private void Awake()
    {
        BackwardBtn = GameObject.Find("Backward").
          ↪ GetComponent<Button>();
        ReturnBtn = GameObject.Find("Return").
          ↪ GetComponent<Button>();
        ForwardBtn = GameObject.Find("Forward").
          ↪ GetComponent<Button>();
        ImageImportScript = GameObject.Find("Image").
          ↪ GetComponent<Demo5_ImageImport>();
    }

        void Update () {
            ForwardBtn.onClick.RemoveAllListeners();
            ForwardBtn.onClick.AddListener(() =>
            {
                if (ImageImportScript.ImageNum <= 400 &&
                  ↪ ImageImportScript.InMenu == true)
                {
                    ImageImportScript.ImageNum += 10;
                    ImageImportScript.enabled = false;
                    ImageImportScript.enabled = true;
                }
            });

            BackwardBtn.onClick.RemoveAllListeners();
            BackwardBtn.onClick.AddListener(() =>
            {
                if (ImageImportScript.ImageNum >= 10 &&
                  ↪ ImageImportScript.InMenu == true)
                {
                    ImageImportScript.ImageNum -= 10;

                    ImageImportScript.enabled = false;
                    ImageImportScript.enabled = true;
                }
            });

            ReturnBtn.onClick.RemoveAllListeners();
            ReturnBtn.onClick.AddListener(() =>
            {
                if (ImageImportScript.InMenu == true)
                {
                    ImageImportScript.InMenu = false;
```

```
                    GameObject.Find("Image").transform.
                      ↪ localScale = new Vector3(0, 0,
                      ↪ 0);
                    GameObject.Find("Menu").transform.
                      ↪ localScale = new Vector3(0.7f,1f
                      ↪ ,0.7f);

                    ImageImportScript.ImageNum = 0;
                    ImageImportScript.enabled = false;
                }

            });
    }
}
```

Listing 6.8: Content of the script attached to the control panel object, allowing users to navigate the image gallery

New images import mechanism

With the addition of a control panel and a new change of the menu, the researcher needed to alter the images import script so that it could function correctly when either the Backward, Forward button, or a menu object was clicked. Firstly, the updated script created three new public variables named “ImageNum”, “MenuNum”, and “InMenu”. These three had a variable type in order of string, integer, and boolean. As described in the previous section, the “ImageNum” variable was used to allow users move to the next five images or move back to the previous ones. The “MenuNum” variable was utilised in a new category selection script, which contained the category name that the users had picked. The final variable, “InMenu”, served as a trigger for the system to identify if the users were currently in the menu screen or in the image selection screen.

In the old version of the script, the code structure only allowed the script to process once time, which was by the start of the demo. If any button in the control panel was clicked, the system could not process the image import feature again. In order to solve this problem, the content of the script was moved to the Unity “OnEnable” function; this function triggered the feature whenever this script was enabled even if it was enabled many times before. To be more detail, if the users clicked on any menu, the script was enabled; if the users got back to the menu screen from the gallery screen, the script was set to disabled waiting for the next enablement.

Besides this alteration, the image import script at this time could also check if there was any issue in the downloading process. In case of the downloaded image was corrupted, an error image would appear in the gallery screen.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Demo5_ImageImport : MonoBehaviour {
...
    public string MenuNum;
    public int ImageNum = 0;
    public bool InMenu = false;

    private void OnEnable()
    {
        InMenu = true;
        StartCoroutine(RunScript());
    }

    IEnumerator RunScript()
    {
        Debug.Log(MenuNum);
        yield return StartCoroutine(GetIndex(MenuNum + "
            ↪ list.txt"));

        int x = ImageNum;
        GameObject photo;

        for (int i = 0; i <= 4; i++)
            {
            urll = new System.Uri(MenuNum + lines[x]).
                ↪ AbsoluteUri;
            urlr = new System.Uri(MenuNum + lines[x + 1])
                ↪ .AbsoluteUri;

            photo = GameObject.Find("Image" + (i) + "-1")
                ↪ ;

            if (photo == null)
                photo = CreateImage(i);
                yield return StartCoroutine(GetImage(url1,
                    ↪ urlr, photo));
                photo.GetComponent<SpriteRenderer>().sprite.
                    ↪ name = lines[x];
                photo.transform.GetChild(0).GetComponent<
                    ↪ SpriteRenderer>().sprite.name = lines[x
                    ↪ + 1];
                x = x + 2;
            }

        }

        IEnumerator GetIndex(string url)
        {
...
        }

        GameObject CreateImage(int i)
        {
...
        }

        IEnumerator GetImage(string url1, string urlr,
            ↪ GameObject photo)
        {
            var left = new WWW(url1);
            var right = new WWW(urlr);
            yield return left;
            yield return right;

            Texture2D texture = new Texture2D(736, 547,
              ↪ TextureFormat.DXT1, false);
            Rect rec = new Rect(0, 0, 3648, 2736);

            Sprite spriteToUse;



            if (left.error == null)
            {
                spriteToUse = Sprite.Create(left.
                    ↪ textureNonReadable, rec, new Vector2(0.5
                    ↪ f, 0.5f), 100);
                photo.GetComponent<SpriteRenderer>().sprite =
                    ↪  spriteToUse;
               }

            else photo.GetComponent<SpriteRenderer>().sprite
              ↪ = GameObject.Find("error-panel").
              ↪ GetComponent<SpriteRenderer>().sprite;

            if (right.error == null)
            {
```

```
                spriteToUse = Sprite.Create(right.
                    ↪ textureNonReadable, rec, new Vector2(0.5
                    ↪ f, 0.5f), 100);
                photo.transform.GetChild(0).GetComponent<
                    ↪ SpriteRenderer>().sprite = spriteToUse;
            }
            else photo.transform.GetChild(0).GetComponent<
                ↪ SpriteRenderer>().sprite = GameObject.Find("
                ↪ error-panel").GetComponent<SpriteRenderer>()
                ↪ .sprite;

            photo.GetComponent<Demo4_ImageScale>().enabled =
                ↪ true;

            left.Dispose();
            left = null;

            right.Dispose();
            right = null;
        }
}
```

Listing 6.9: Content of the updated image import script, adding an ability of navigating the image gallery and error detection for the downloading process

Performance test of the fifth demo

While there were one new feature and multiple tweaks for the current scripts, the researcher was still unable to fix the performance hiccup stated in the previous demo. This performance test not only evaluated the reliability of the new feature, but also helped the researcher identify the culprit of the performance issue.

This test used the same resource and participants as the previous one.

- **Test result**

  For the control panel, the feature worked as intended. The testers could easily browse through several images just by pointing the pointer on the button and pressing the capacitive button of the VR headset. In the 3D gallery room, all images displayed correctly in front of the users' camera.

  For the performance hiccup issue, the culprit was identified through the trial & error method. Whenever a new sprite texture was added to the image object on the demo, all other tasks was halted; thus caused the frame freeze. Since the current version of Unity does not support parallel work for any function and class in its code library, it was impossible to fix the issue. Moreover, there was a memory leak issue with the current image import script. This issue did not occur when the demo was tested on a PC with high capacity memory. When being tested on the smartphone, an environment with limited memory, the demo crashed. A new approach would be researched for the next demo to address these issues. Besides fixing the memory leak issue, the frame freeze duration could be either removed or reduced with this new approach.

6.2.6 Final Demo – November 11th, 2017

New photo object – switching sprite object type to 3D object type

After several tasks of trial and error, an issue related to the performance issue in the last demo was identified. When a photo was retrieved from the server, its texture was stored and used in a sprite conversion process. This process required the smartphone to temporarily halted other task and focus on it, which caused the application frame to be freeze until the process completed. This process was impossible to be simplified because it was mandatory to the sprite object system in Unity3D. Because of this, the researcher decided to switch the photo object type to a more flexible 3D object.

The advantage of 3D object type in this scenario was that it did not need the sprite conversion process. Moreover, it automatically resized the photo to match with the developer's size settings without the use of the scaling image script stated in section **6.2.4 Fourth Demo – October 1st, 2017**. There were two steps of switching to the 3D object type.

Firstly, the prefab was changed from a sprite object to a Quad 3D object in Unity3D library. This new prefab was also attached with the image display script, the even trigger script, and box collider just as the previous prefab. Only the image scaling script was excluded.

Secondly, all of the code related to the sprite conversion process was removed from the online image import that was used in the previous demos (first stated in section **6.2.4 Fourth Demo – October 1st, 2017**). This code was replaced with a simple function that replaced the current texture of the photo object with the new texture retrieved from the online server.

The memory leak issue was also fixed by simply adding a built-in unused memory cleaning from Unity3D engine, which is "Resources.UnloadUnusedAssets()".

New loading notification

With the switch to the 3D object type, the freeze duration of the application decreased. However, it was still possible for users to misunderstand that the application was broken during the freeze frames. Since it was impossible to completely get rid of the freeze issue at the time, an approach of showing a loading notification to cover the freezing frames was used. Whenever a process of retrieving photos from the server was performed, a notification would appear to help the users understand that the application was loading. In addition, a loading bar also appeared to indicate when the process completed.

The notification had the same display mechanism as the Stereoscopic photos on dual camera mechanism stated in section **6.2.2 Second Demo – August 6th, 2017**. It was simply an image UI object that was activated when the photo retrieve process began, and deactivated when the process completed.

For the loading bar, it was also a part of the UI system from Unity3D engine. The visual part of the bar depended on an attached variable named "Value". By changing the value of this variable when the process completed, the loading bar would automatically fill up to indicate that the process had completed.

To allow the loading notification gradually display and cover the freeze frames, the loading notification function had to be running throughout the application runtime, which was unnecessary. Instead of that, the researcher put this function into a closed loop that only ended when the loading notification was displayed completely to the users.

```
...
public class Demo6_ImageImport : MonoBehaviour {
...
    private void OnEnable()
```

```
{
    ...
}

IEnumerator RunScript()
{
    float starttime = Time.time;
    while (Time.time < starttime + 1)
    {
        yield return LoadingBg.color = Color.Lerp(
          ↪ LoadingBg.color, new Color(255, 255,
          ↪ 255), (Time.time - starttime) / 1);
    }

    ...

    for (int i = 0; i <= 4; i++)
    {
        ...

        LoadingBar.value += 0.2f;

    }
    LoadingBar.gameObject.SetActive(false);
    starttime = Time.time;
    while (Time.time < starttime + 1)
```

```
        {
            yield return LoadingBg.color = Color.Lerp(
              ↪ LoadingBg.color, Color.clear, (Time.time
              ↪  - starttime) / 1);
            Pointer.SetActive(true);
        }

    }

    IEnumerator GetIndex(string url)
    {
                ...
    }

    GameObject CreateImage(int i)
    {
                ...
    }

    IEnumerator GetImage(string urll, string urlr,
      ↪ GameObject photo)
    {
        photo.GetComponent<Renderer>().material.name =
          ↪ urll;
        urll = new System.Uri(MenuNum + urll).AbsoluteUri
          ↪ ;
```

```
var left = new WWW(url1);
yield return left;

photo.GetComponent<Renderer>().material.
    ↪ mainTexture = null;
if (left.error == null)
{
    photo.GetComponent<Renderer>().material.
        ↪ mainTexture = left.textureNonReadable;
}
else photo.GetComponent<Renderer>().material.
    ↪ mainTexture = GameObject.Find("error-panel")
    ↪ .GetComponent<Renderer>().material.
    ↪ mainTexture;

left.Dispose();
left = null;

photo.transform.GetChild(0).GetComponent<Renderer
    ↪ >().material.name = urlr;
urlr = new System.Uri(MenuNum + urlr).AbsoluteUri
    ↪ ;

left = new WWW(urlr);
yield return left;
```

```
            photo.transform.GetChild(0).GetComponent<Renderer
                ↪ >().material.mainTexture = null;
            if (left.error == null)
            {
                photo.transform.GetChild(0).GetComponent<
                    ↪ Renderer>().material.mainTexture = left.
                    ↪ textureNonReadable;
            }
            else photo.transform.GetChild(0).GetComponent<
                ↪ Renderer>().material.mainTexture =
                ↪ GameObject.Find("error-panel").GetComponent<
                ↪ Renderer>().material.mainTexture;

            left.Dispose();
            left = null;

            Resources.UnloadUnusedAssets();
        }
}
```

Listing 6.10: Addition code to the ImageImport script file: (1) displaying the loading notification to indicate when the image import process completed; (2) replace sprite object with 3D object which had its texture replaced with the photo texture retrieved from the online server.

Performance test of the final demo

This concluded the development stage of the research, the final demo was used for the whole application evaluation. This was to ensure that all of the components would work properly when implemented together.

6.3 Content searching process

This process is used to identify any stereoscopic content that is available on the internet. Since there have been multiple searching engines with robust algorithm on the market, it was not necessary to develop a custom searching engine for the process.

Instead the researcher chose to utilise Google Image Search to perform the searching process. When compared with other engines such as Bing, Baidu, and Google Search is the most popular search engine. It not only provides more accurate results, but also includes many tools and addon for developers.

After users provide their searching keywords, the application will start loading Google Image Search API, an addon that has already been developed by Google.

```
<script type="text/javascript" src="www.google.com/jsapi">
</script>
```

The user inputs will be used to replace *SEARCHING_TEXT* to perform the image search:

```
<script type="text/javascript">
  google.load("search", "SEARCHING_TEXT");
</script>
```

In order to filter non-stereoscopic contents, these user's searching keywords are attached with multiple tags that define stereoscopic format.

```
SEARCHING_TEXT = ("side by side stereo 3D" OR "cross eye 3D" OR
"parallel eye 3D" OR “anaglyph 3D” OR “stereogram 3D”)
SEARCHING_TEXT = SEARCHING_TEXT + AND + "EXTRA_KEYWORDS"
```

After the Google Image Searching server receives the requests, search results will be transferred to the application storage. While these results are related to the stereoscopic tags, it is still possible that some non-stereoscopic or incompatible contents have been mistakenly retrieved. Therefore, these downloaded contents will go through a classification process.

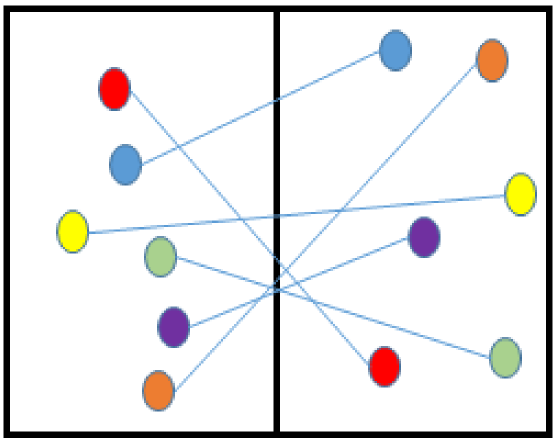
6.4 Content classification process

The classification process is the cooperation between the researcher and the supervisor.

The first step of this process is to remove non-stereoscopic contents from the content storage. All retrieved contents are classified into four types (3 stereoscopic types and 1 normal type)

- side-by-side photos (type SBS)
- anaglyphs (type ANAG)
- stereograms (type STE)
- "normal" (type NOR)

Each stereoscopic type has a distinguished method to make them viewable in the application. For the SBS type content, the process cut it into two parts: left stereo and right stereo image. These images are then send to the corresponding screen of the VR headset so that the 3D effect can be viewed properly. If the content format is in ANAG, all three colours are separated from the input content. Only Red and Green channels of the image are transferred to the left and right screen of the VR headset. Lastly is the STE type content. 90% of the left side and 90% right side of the content image are sent to the corresponding screen side of the VR headset. This will make the STE type content viewable. While these methods are straight-forward, the classification of these three stereoscopic types are complicated, requiring several utilisation and comparison of image processing algorithm.


(a) Matches are not horizontally aligned on “normal" image

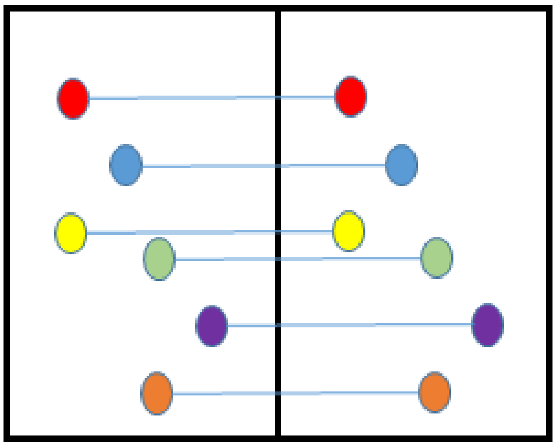

(b) Matches are horizontally aligned on side by side image

Figure 6.15: Feature Matching between two halves could determine the types of 3D images

The content classification was designed based on the characteristic of stereopsis, the system that allows human to see depth effect. All three stereoscopic types (SBS, ANAG, and STE) also make use of this system. Therefore, it was possible to classify these types by analysing the stereopsis feature in them (demonstrated in Figure 6.15). Identifying the stereopsis feature required the use of Feature Matching, a technique related to Computer Vision. This technique selects informative pixels from two halves of an image, then compare them together and determine if they are the corresponding pairs. There had been many matching algorithms from other research that could be used for this process. Because of this, the only necessary task was to find the most suitable algorithm which could be used to achieve the best result in the fastest time.

6.4.1 KLT, SIFT, and SURF Feature Extraction for Correspondence Matching

Feature Matching is a computer vision technique that is usually used for object recognition (Bicego, Lagorio, Grosso & Tistarelli, 2006), tracking (Tomasi & Kanade, 1991), as well as stereo matching (Hoff & Ahuja, 1989). It finds a notable set of pixels from a reference image, then compares them with a set of pixels of another image to determine if their similarity.

Currently there are three popular algorithms that are used for this process. The earliest one is Kanade-Lucas-Tomasi (KLT), which is designed based on Lucas-Kanade optical flow in a pyramid (Lucas & Kanade, 1981) of Lucas et al and Tomasi's good feature to track (GFT) (Shi & Tomasi, 1993). While this algorithm provides relatively accurate result in a short period of time, it can have a problem of returning too many outliers under certain circumstances.

To get rid of this problem and provide a more accurate result, more recent methods have been designed, which are Scale-Invariant Feature transform (SIFT) by Lowe et al. (Lowe, 1999) and the Speeded Up Robust Features (SURF) by Bay et al. (Bay, Ess, Tuytelaars & Gool, 2008). These two algorithms utilise the modern computing hardware to bring a good result even when there are large scale changes and affine transformation of features between the two images. Thus, they are widely used in various feature trackers (Tuytelaars & Mikolajczyk, 2008) nowadays. However, when implemented with low performance hardware, SIFT and SURF can be time consuming (Juan & Gwun, 2009).

Since the application is designed for smartphone environment, which is known to have limited processing power, it was unclear if the modern methods like SIFT and SURF was the better choices than the classic KLT algorithm. A comparison between these three was needed.

6.4.2 Evaluation of KLT, SIFT, and SURF trackers on 2005 and 2006 Middlebury Stereo Datasets

All algorithms were used to detect correspondences in a large library of image pairs. After the process, the result accuracy and total processing time were collected and compared to determine their overall performance. There were two image libraries that were used in this evaluation:

1. Middlebury 2005 and 2006 dataset. (Figure 6.16) the correspondences between images are known (Section 6.4.2).
2. Real-life near stereo images (Figure 6.17) with ground truth correspondences are not known (Section 6.4.3).

There were 30 pairs of indoor images in the Middlebury datasets. All of them had a resolution of 430 by 370 pixels. These images were captured by the Middlebury Vision Lab, using Structured Light techniques (Scharstein & Szeliski, 2003). In theory, all image pairs were aligned horizontally in a relative manner. Each of the three matching methods was applied on these datasets to acquire 30 sets of correspondences. The first result shows that all methods collected up to 1000 correspondence points, with the best 500 pairs were selected. For the average result: • KLT acquired 811 correspondences in 1.15 seconds • SIFT collected 714 correspondences in the longest time – 2.04 seconds. • SURF resulted with 810 correspondences in 1.72 seconds The overall measurements are shown in Table 6.1.

Figure 6.16: 30 image pairs from Middlebury 2005 and 2006 datasets (only left images are shown)

Method	AVG matches	STD matches	AVG time (seconds)	STD time (seconds)
KLT	811	60	**0.95**	0.29
SURF	810	167	1.72	0.36
SIFT	714	235	2.04	0.67

Table 6.1: Statistic details of correspondence matching using KLT, SURF and SIFT on 2005 and 2006 Middlebury stereo datasets.

Since the images was aligned, the matched points oughted to be aligned. By identifying the misalignment in y-direction $\varepsilon = |y_L - y_R|$, the researcher could analyse the accuracy of each matching method. The average values and standard deviation for each set of correspondences (displayed in Table 6.2) were calculated. With all ε being taken into account, the averages contain huge values (21 to 31 pixel misalignment). Therefore, it is determined that all three matching algorithms result with a large number of outliers.

However, In the situation where outliers are discarded by three thresholds $\varepsilon < 0.5$, $\varepsilon < 1.0$, and $\varepsilon < 2.0$, the average misalignments are relatively *close to zero*. While SIFT has the best result of having the smallest average values, the standard deviation values from KLT and SURF are usually better.

With the 2005 and 2006 Middlebury datasets, the evaluation shows that all three matching algorithms are efficient. The mismatches from the correspondence sets are not significantly different in the correspondences' vertical misalignment. Overall, SIFT and SURF take longer processing time (see Table 6.1) but do not return better correspondence sets than KLT. However, the Middlebury datasets are used as "extreme" cases. These images are sharp and horizontally aligned, with no noticeable changes and affine transformation of features between images.

	$\varepsilon < 0.5$		$\varepsilon < 1.0$		$\varepsilon < 2.0$		$\varepsilon < 1000$	
Method	AVG	STD	AVG	STD	AVG	STD	AVG	STD
KLT	0.16	0.12	0.27	0.24	0.47	0.50	24	46
SURF	0.16	0.12	0.26	0.23	0.43	0.47	31	53
SIFT	**0.13**	0.11	**0.17**	0.18	**0.21**	0.29	27	60

Table 6.2: Statistic details of vertical misalignment $\varepsilon = |y_L - y_R|$ in pixels between correspondences found by KLT, SURF and SIFT on 2005 and 2006 Middlebury stereo datasets

6.4.3 Evaluation of KLT, SIFT, and SURF on Real-life Images

The second evaluation was to determine the method efficiency in a real-life scenario. All three matching algorithms were tested on 200 real-life images, which were captured by the researcher and the supervisor. They were hosted on the shared image database[1]. These images were resized to resolution of 1024×768 pixels, which was larger than the images tested in the Section 6.4.2). Rather than having only indoor photos, this dataset contained images captured under four different conditions:

1. Outdoor under a bright daylight condition.

2. Outdoor under an overcast light condition.

[1]http://www.ivs.auckland.ac.nz/web/scene_gallery.php

3. Indoor under a daylight condition.

4. Indoor under a fluorescent light condition.

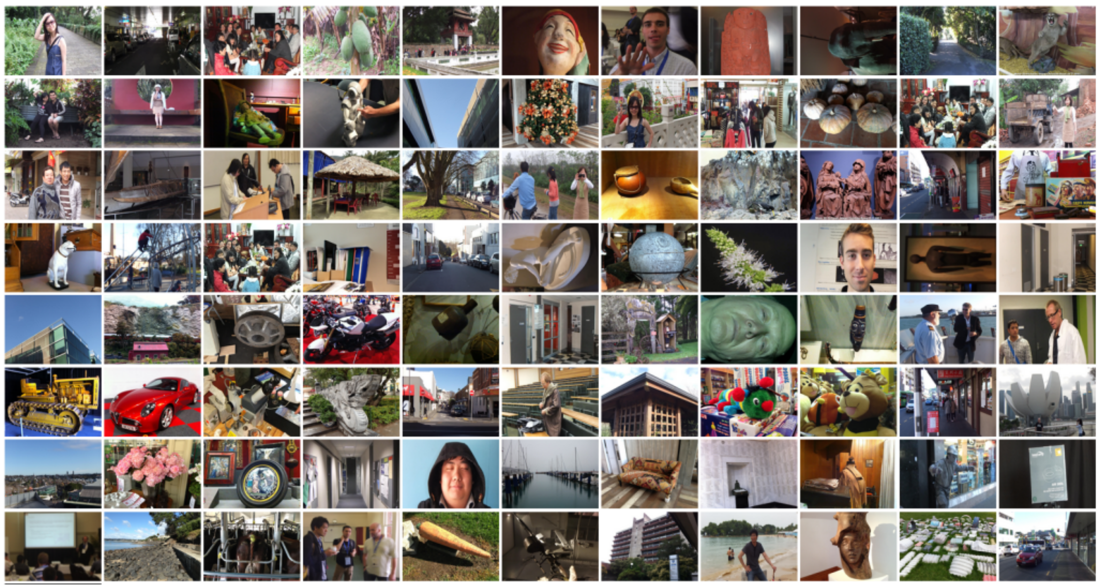


Figure 6.17: A portion of 200 pairs of real-life images are randomly selected from different categories for image feature point matching (only left images are shown).

KLT, SURF, and SIFT method were applied on these images to track for correspondences. Similarly, the matched points and processing times from each matching algorithm were also collected. On average, to obtain up to 500 strongest correspondences: • KLT ran in 2.1 ± 0.7 seconds. • SURF ran in 5.7 ± 2.9 seconds. • SIFT took 7.5 ± 2.0 seconds. With the standard deviation of 0.70 seconds, KLT is found to be significantly faster than both SURF (~ 2 times) and SIFT (~ 3 times). In conclusion, the evaluation shows that KLT is sufficient enough for the image type classification.

6.4.4 KLT Features for Content Classification

KLT feature matching returned two sets of corresponding points $P_1(x, y)$ on the first image and $P_2(x, y)$ on the second picture. To match these two sets together, the researcher used brute-force method. To be more detail, there were N pairs of points:

$P_1(x_i, y_i)$ was corresponding to $P_2(x_i, y_i)$, each of them was a sub-pixel coordinate (x, y) on the pair of images.

The absolute horizontal differences $h = |x_L - x_R|$, and vertical differences $v = |y_L - y_R|$ between corresponding points were also calculated. These values were used to identify the number of matched points which horizontally aligned and vertically aligned. $\P_h$ was the number of pairs that have $h < T$ and $\P_v$ was the number of pairs that have $v < T$. T was a threshold, which was set to be 5.0 pixels; $\P_h \le N$ and $\P_h \le N$. The percentage of horizontally and vertically aligned matches were calculated as:

$$P_h = \frac{\P_h}{N} \quad \text{and} \quad P_v = \frac{\P_v}{N} \tag{6.1}$$

Initially, different types of images returned different values of P_h and P_v when being tracked on regions or colour channels of images. There were three types of comparison that were applied with the feature matching (I_o is the original image):

1. **left/right half:** I_o was vertically cut into two and features were searched between those.

2. **1st/2nd quarters:** I_o was vertically cut into four, and first two quarters were chosen.

3. **red/green channels:** Red, Green, Blue channels were separated from I_o, Red and Green ones were selected for feature matching.

For the classification purpose, the images were resized to the same resolution of 1024×1024 pixels. Then the distance of correspondences between different types of images was calculated and categorised in two classes:

1. **Small**: Average distance is less than 50 pixels

2. **Large**: Average distance is more than 200 pixels

Image Types	Matched between	P_h	P_v
“Normal” & Anaglyphic	left/right half	**Small**	Small
Stereogram & Side by Side	left/right half	**Large**	Small
“Normal”	red/green channels	Large	**Large**
Anaglyphic	red/green channels	Large	**Small**
Stereogram	1st/2nd quarters	**Large**	Small
Side by Side	1st/2nd quarters	**Small**	Small

Table 6.3: Ranges of mean $\mu_{h/v}$ and standard deviation $\sigma_{h/v}$ found on four image types

The ranges of $\mu_{h/v}$ and $\sigma_{h/v}$ collected from the four different image types are shown in Table 6.3. The table shows a significant difference in μ_v between “Normal” & Anaglyphic and Stereogram & Side by Side images. Therefore, μ_v can be used to separate these two classes.

In the case of side by side pictures, the two halves of the pictures have a good correlation. Despite this, there are not many matched points between the two-quarters of the picture. In contrast, two quarters of the stereogram images have a strong correlation with a good number of matched points between them. Making use of this characteristic can help distinguish Stereogram and Side by Side images.

Finally, μ_h can be used to separate between “Normal” and Anaglyphic images. By analysing red and green channels of an image, Anaglyphic images can be easily detected. With these classification methods, a process is created to help identify all image types on the Internet based on thresholding the values of $\mu_{h/v}$ and $\sigma_{h/v}$. The process is simplified in Flowchart displayed in Figure 6.18.

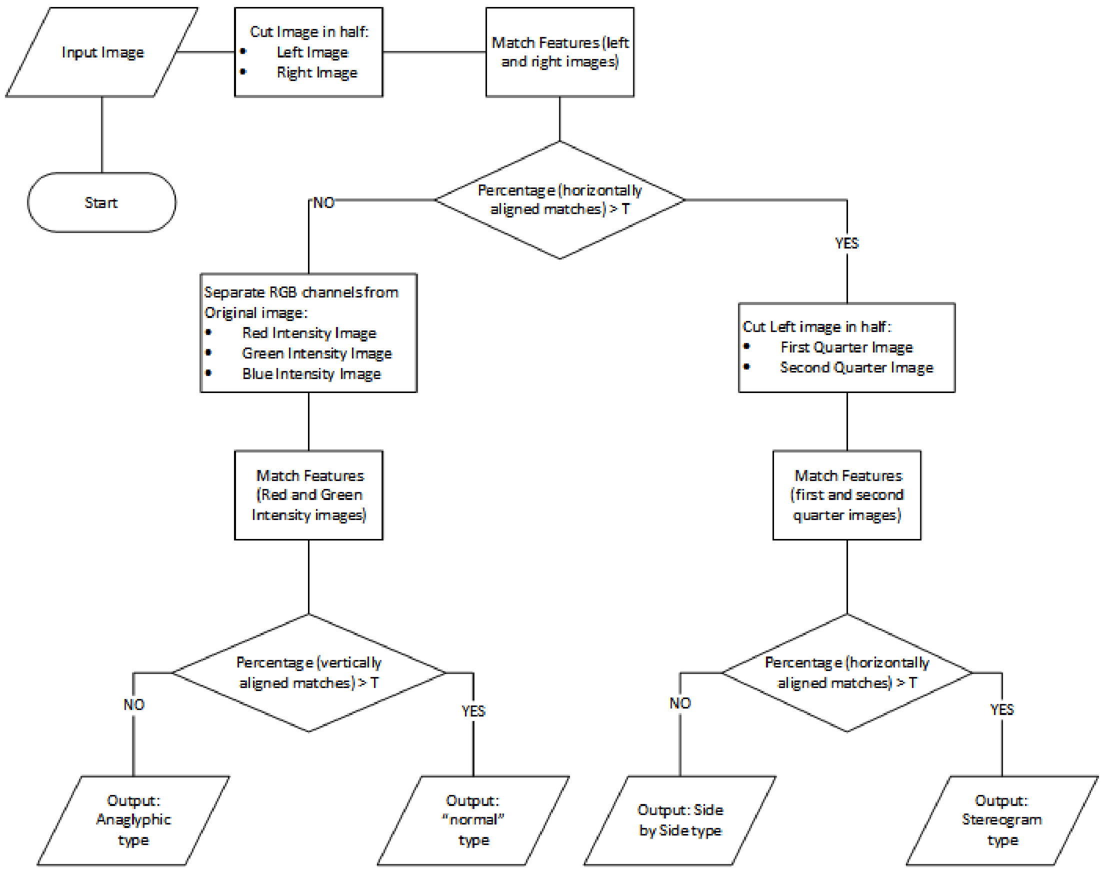

Input Image
Cut Image in half:
• Left Image
• Right Image
Match Features (left and right images)
Start
Percentage (horizontally aligned matches) > T
NO
YES
Separate RGB channels from Original image:
• Red Intensity Image
• Green Intensity Image
• Blue Intensity Image
Cut Left image in half:
• First Quarter Image
• Second Quarter Image
Match Features (Red and Green Intensity images)
Match Features (first and second quarter images)
Percentage (vertically aligned matches) > T
Percentage (horizontally aligned matches) > T
NO
YES
NO
YES
Output: Anaglyphic type
Output: "normal" type
Output: Side by Side type
Output: Stereogram type

Figure 6.18: Flow chart of the classification base on KLT matched features

6.4.5 Final Results

For the final evaluation, the described Content Classification was tested in a custom content dataset hosted by University of Auckland (`http://www.ivs.auckland.ac.nz/web/scene_gallery.php`). The database contained all highlighted stereoscopic types (SBS, ANAG, STE, and "normal" left or right image). Since the dataset had been used in a previous stereoscopic related project, all contents were verified with correct format. Due to time constraint, instead of using the whole dataset, only 50 contents in each stereoscopic format went through the classification process.

Image Type	Correct Detection	Incorrect Detection	Percentage
“normal” images (type NOR)	48	2	96%
anaglyphs (type ANAG)	50	0	100%
side-by-side photos (type SBS)	50	0	100%
stereograms (type STE)	50	0	100%

Table 6.4: Detection rate of the classification

Table 6.4 shows the number of correct detection and the percentage of accuracy of the Content Classification process. On the first hand, the process identifies 2 “normal” images incorrectly, making the type NOR detection accuracy being only 96%. On the otherhand, with other stereoscopic types, the process achieves 100% in detecting these types correctly.

Chapter 7

Project Evaluation

7.1 Evaluation scope

Even though every application feature has gone through a small performance test during the development stage, it is still unclear if the whole application can provide a satisfied experience. Therefore, it is important to conduct an application evaluation to identify persistent issues as well as features that meet the objective.

In order to meet the time frame of the project, the application test was simplified with only three users participating in a few tasks. While the amount of test users were limited, all of them had good knowledge in technical field and were trusted to provide critical feedback to the researcher.

7.2 Test Users

7.2.1 Entertainment Purpose - User A

User A is a university student, he finds Virtual Reality (VR) fascinating and would love to have one kind of system for entertainment purpose. He also loves to use his Galaxy S6 everyday thanks to the power the phone has.

While there are lots of VR devices available on the market, the high prices have prevented him from getting one. That is the reason why Google Cardboard catches his attention.

- Young university student.
- Easily adapt to the new technology.
- A smartphone guru, especially Android.
- Has good eye vision.
- Has experience with VR, which can benefit from giving the feedback on navigation feature of application.

7.2.2 Education Purpose - User B

User B is a professional specialised on education. During his teaching, he finds that most students are not interested in the class due to the high amount of words and the lack of field trips. He thinks virtual reality is a great idea for students to enjoy a virtual field trip to the museum, or even virtually live in the past.

The portability of smartphones VR, especially Google Cardboard, could help him easily carries the system to the class without any difficulty. It also sparks the student's enthusiasm when they understand that the smartphones they have can achieve such a wonderful experience.

- Lecturing professional.
- Want to enhance the study in classes.
- A smartphone casual user.
- Has myopia.
- Has experience with viewing 3D contents in many types, which can benefit from giving the 3D content's quality.

7.2.3 Marketing Purpose - User C

User C is a university student with a major in marketing. Even though he has no experience with VR, its popularity has caught his attention. In his opinion, VR could be a great tool for product promotion. Instead of viewing the catalogue in classic ways like brochure reading or web browsing, customers could view a great variety of products like they are in a brick-and-mortar store.

- Young university student.
- Interested in VR usage for marketing, viewing 3D contents.
- Has a good knowledge of smartphones.
- Has myopia.
- Has a bit of experience in viewing 3D movies. Never try VR before.

7.3 Evaluation methodology

7.3.1 Test prerequisite

This test was mainly based on the user's experience when wearing the VR headset. Therefore, the pressure could be eliminated if the test was performed in tradition method. However, the employment of co-operative inquiry method in full form could help the users express their feedbacks more openly.

Typically, a performance test consists of three persons: a logger, a facilitator, and the user (Heron, 1996). However, due to the small scale of this test, the researcher was responsible for both a logger and a facilitator role. Before it started, the testing location was decided based on the nature of the test. Thanks to the portability of smartphone VR, this test could be conducted at anywhere that users may seem fit. Since all of the users participating in this project were either students or lecturer from AUT, a small room located inside the AUT campus was the best choice for the testing location.

For the test result to be accurate, each user had to be comfortable during the test. Good relationship between the user and the facilitator was also an important factor. In this case, both of the AUT students were friends of the researcher; the university lecturer was actually the supervisor for the researcher's master program. Therefore, the test generally went well with a friendly and relaxing atmosphere.

In the core nature of co-operative inquiry, the facilitator & the user share the device. The facilitator needs to see how the users perform the test directly so that he can identify which issue the application is having (Kuniavsky, Goodman & Moed, 2012). However, each headset could only be used by only one person. With this scenario, besides the demo that was used for the test, there were some software additions used to assist the facilitator monitor the user's performance.

To monitor the smartphone screen during the test, the researcher used the Android Cast Screen feature, which was created a mirror image of the smartphone screen on a computer monitor or TV display; the smartphone was “cast” to a Windows 10 PC through a private Wi-fi network.

Next is the test session recording, the researcher used GameDVR, a built-in Windows 10 display recorder tool, to capture all images of the smartphone screen into a video file; while another smartphone was used to record all of the users’ behaviours during the test. These two recoding videos would be used for more detail analysis after the test.

7.3.2 Preliminary questionnaire

Instead of starting the test right away, each user had to take part in a preliminary questionnaire. This small survey was used to identify user characteristics such as age, health condition. These characteristics were important because they could help the researcher determine whether the issues occurring during the test truly related to the application, or due to the users’ characteristic.

One notable example is motion sickness. According to the observation study from (Solimini, 2013), after watching 3D contents, 54.8% of users had Simulator Sickness Questionnaire Scoring of over 15, which is considered to moderate-to-severe grade of motion sickness. Solimini recognised that all of the testers who had history for car sickness, vertigo disturbances also experienced the motion sickness issue.

1. What is your gender?

   Male   Female

2. What is your range of age?

   <18   18-25   26-40   >40

3. Do you have myopia or hyperopia?

   Yes   No   Not sure

4. Do you experience motion sickness while traveling by car for a long period of time?

   Yes   No   Not sure

5. How often do you play any 3D games?

   Never   Sometimes   A lot

6. How often do you watch 3D movies?

   Never   Sometimes   A lot

7. What functionality you use in your phone?

   Web browsing   Communication   Video   Gaming

   Social network   Literature reading (e-magazine, e-book)

8. What is the budget that you are willing to spend for your phone?

   100-299$ 300-599$ >600$

9. Do you know what virtual reality is about?

Figure 7.1: The content of the preliminary questionnaire

7.3.3 Test content

At the beginning of the test, the facilitator and the users had a friendly conversation, sharing their hobby and the reason why they love to try VR. Moreover, during the test, the users received a brief description of the tasks they had to, the feedback of which was used to analyse the demo controllability and performance.

Based on the preliminary questionnaire result, each user was assigned to a different task. These tasks were used to evaluate three aspects of the application: (1) application navigation, (2) 3D effect, and (3) virtual reality sickness. While the tasks were different, the length of the tests for all users were similar, lasting around 5 minutes.

After the test completed, the users were required to fill-out a feedback form, which consisted of several simple Yes-No questions. The answers for these questions would be used for three purposes: (1) determine if the demo was suitable for all types of user, (2) identify the current issues the demo was having, and (3) identify the potential improvements that could be added in the fast time frame.

Due to the fully immersive experience, any issue related to the smartphone display could invoke some significant emotions such confusion, horror on the users; this could affect the validity of users' feedback. Therefore, if these emotions happened, the test would be halted until the users' emotion state became stable again.

7.4 User activity log

7.4.1 User A

Adaptive task

Since this user was very familiar VR, the task was mostly on how to navigate the application.

1. Point the camera to any category image.
2. Use the headset button to choose the category.
3. After the loading completes, point the camera to the "forward button" and use the headset button to select it.

4. After the loading completes, point the camera to the “backward button” and use the headset button to select it.
5. Point the camera to the “return button” and use the headset button to select it.

Testing log

- The user used the camera to look around.
- The user chose category “animal” with the image of 5 lemurs.
- A loading screen showed up, the user complained that the loading time is a bit long.
- The user tried to look behind to see if there was any other change besides the five new images.

Technical issue

Initially the VR screen was a bit blurry. Adjusting the headset lens fixed the issue.

7.4.2 User B

Adaptive task

With this characteristic, the test focused on viewing multiple stereoscopic contents.

1. Point the camera to any category image.
2. Use the headset button to choose the category.
3. After the loading completes, point the camera to the centre image and use the headset button to select it.
4. In the 3D gallery room, view the image express whether the 3D quality is better, the same, or worse than viewing with 3D glasses.

5. Use the headset button again to get back to the menu screen.
6. Follow the first five steps again but on the image on the right side.
7. Follow the first five steps again but on the image on the left side.

Testing log

- The user used the camera to look around.
- The user chose category "statue" with the image of a blue statue of an elephant.
- When viewing the first stereoscopic image, the user expressed that the sudden enlargement of the image causes a bit dizziness.
- The user tried to look around in the 3D gallery room.

Technical issue

There was no technical issue in this test.

7.4.3 User C

Adaptive task

While the user's main interest was in viewing the 3D content, he had no experience in VR. Therefore, his test focused on viewing 3D contents in only one category.

1. Point the camera to centre image.
2. Use the headset button to select it.
3. After viewing the image in 10 seconds, use the headset button again to get back to the menu screen.
4. Follow the first three steps again but on the image on the right side.

5. Follow the first three steps again but on the image on the left side.

Testing log

- The user observed the environment around. He expressed that he felt a bit strange.
- When viewing the first stereoscopic image, the user expressed joyful since it was so easy to see the 3D effect.
- The user looked around the 3D gallery room and noticed there were two notifications on the left and right side of the room.
- The user continued with the next two images with no new noteworthy activity.

Technical issue

The headset button was not sensitive, the user frequently pressed two or three times for it to register the action.

7.5 Test result

Based on the activity log of three users, positive and negative feedback on the application are identified.

7.5.1 Positive feedback

User A

Each category did not have any text but instead an image describing what it was about. This made the category selection screen was straightforward. Users (especially children) could easily pick any topic they were interested in even if they did not know English.

The control panel was also very easy-to-use. With three simple buttons displayed at the floor of the virtual room, User A could navigate through any 3D contents of the application.

User B

User B complimented that the 3D effects was very good. Compared with traditional methods such as 3D glasses and eyes crossing, viewing with this application was easier. The darkness of the 3D gallery room also made the 3D content stand out, improving its depth. Moreover, it was also easy to get in and get out of the 3D gallery room.

User C

When viewing the first content, User C was surprised that he could see the 3D effect right away with no adjustment of his glasses. With three buttons located at the virtual floor, browsing the 3D contents was simple and quick. The user was also impressed with the impressiveness of the 3D gallery room. When being put in a completely dark room, the user could easily focus on the 3D content appeared in front of him.

7.5.2 Negative feedback

About the application navigation, User C had an issue with the button of the headset being not accurate. There were multiple times that he had to press two or three times for the action to be registered.

For the 3D gallery room and 3D quality, while the 3D effect was generally good. The sudden transition from the bright menu screen to the dark virtual room caused eye strain, especially with User C.

All three users agreed that the loading time was still a bit long, especially when the application only loaded five 3D images. Moreover, the menu screen also appeared “too basic”, with only contained some images and three buttons.

7.5.3 Evaluation checklist

Criteria	Result	Priority
Does the application use low-cost equipment?	Yes	
Is it easy to navigate through the application	Yes	
Is the language of the application understandable?	Yes	
Is the user interface easy to use?	Yes	
Does the 3D effect display correctly?	Yes	
Is the application responsive?	No Slow loading time Headset button not register accurately	High
Is the virtual environment attractive and well-designed?	No Basic 3D models Only contain few 3D objects	Normal
Does the 3D content satisfy the users?	No Limited category and content	Low
Does the application suitable for education use?	Yes	

Table 7.1: Evaluation checklist based on the test results

In **Chapter 4. Project Objective**, there are multiple features which are set to be the main objectives for this project. The criteria in the evaluation checklist are created based on them. The test feedback from the three users are used to find the results for this checklist. A criterion receives a result as yes when there is no negative feedback related to it. If a criterion has any issue identified in the test result, it will fail to pass the checklist. Depend on the criticalness of the issue, the criterion will receive a work priority of whether “High”, “Normal”, or “Low”.

7.6 Current challenges

Currently, the biggest issue that this application is facing is the long loading time. The culprit is the conversion process from 2D images to 3D objects, which is too demanding on the smartphone hardware. On the development computer, this conversion process can finish in just about 5 seconds. On the contrary, when testing the process on the development smartphone, there are multiple frame freezing which slow down the process dramatically. Since this issue is very noticeable, the researcher will focus on fixing it before moving to other improvement tasks.

The insensitive of the headset button do not actually related to the application. However, there is a workaround that can improve the navigation experience. Besides using the physical button press, the gaze-contingent feature will be added. With this feature, users can simply point the camera to the object and wait for around 2 seconds to have it selected.

Due to the limited time and resource, the application is still in a basic form of content as well as functionality. After fixing the issues recorded in this evaluation report, more functions will be added to improve the application experience. More detail is shown in section **8.1 Future work**.

Chapter 8

Conclusion

8.1 Future work

8.1.1 Online stereoscopic 3D encyclopedia

Background

Currently, there are many Augmented Reality (AR) Book products available on the market; some notable examples are The Dragon Defenders (Ford, 2017), The Mardles Storybook (Parsons, 2017), Wonderbook: Book of Spells (Sony Interactive Entertainment, 2012).

Pictorial markers are often required to allow the computer system to identify correct information on these AR Books. While these markers can contain hidden information as well as having a meaningful look to viewers, the system can only identify secret information if the marker image patterns are registered in its database. The more hidden information a book contains, the more demand on the system database and processing power. Moreover, when these markers have similar image patterns, the recognition process will become inaccurate (Tikanmäki & Röning, 2011). To resolve these issues, developers can use the bar-code marker type instead. This marker type is designed with black and white data cells that contain a set of binary number, which can be decoded into meaningful data accurately by any modern computer system (Hamming, 1950) (Reed & Solomon, 1960). However, this marker type has a disadvantage of having a meaningless appearance to the viewers.

To solve these problems, Minh Nguyen has introduced a new type of QR marker that can keep the vivid appearance of the picture (Nguyen, Tran, Le & Yan, 2017). By combining both Virtual Reality (VR) and AR technology, the application can utilise this new QR marker to create a new experience to explore the vast 3D stereoscopic datasets on the internet. The procedural steps of the application are listed below:

1. The users first use their mobile devices such as smartphones or tablets as display apparatus.

2. The system uses the devices' camera to find the target marker in the real world such as tags in encyclopedia books or magazines.

3. The system then processes and decodes the hidden information (searching keywords) from the detected tile-based marker and switches entirely to VR for visualisation.

4. The system automatically extracts and displays left/right stereo images and related text to VR devices from the Internet search using the extracted keyword.

5. The users then can easily switch back to the AR environment to detect different target.

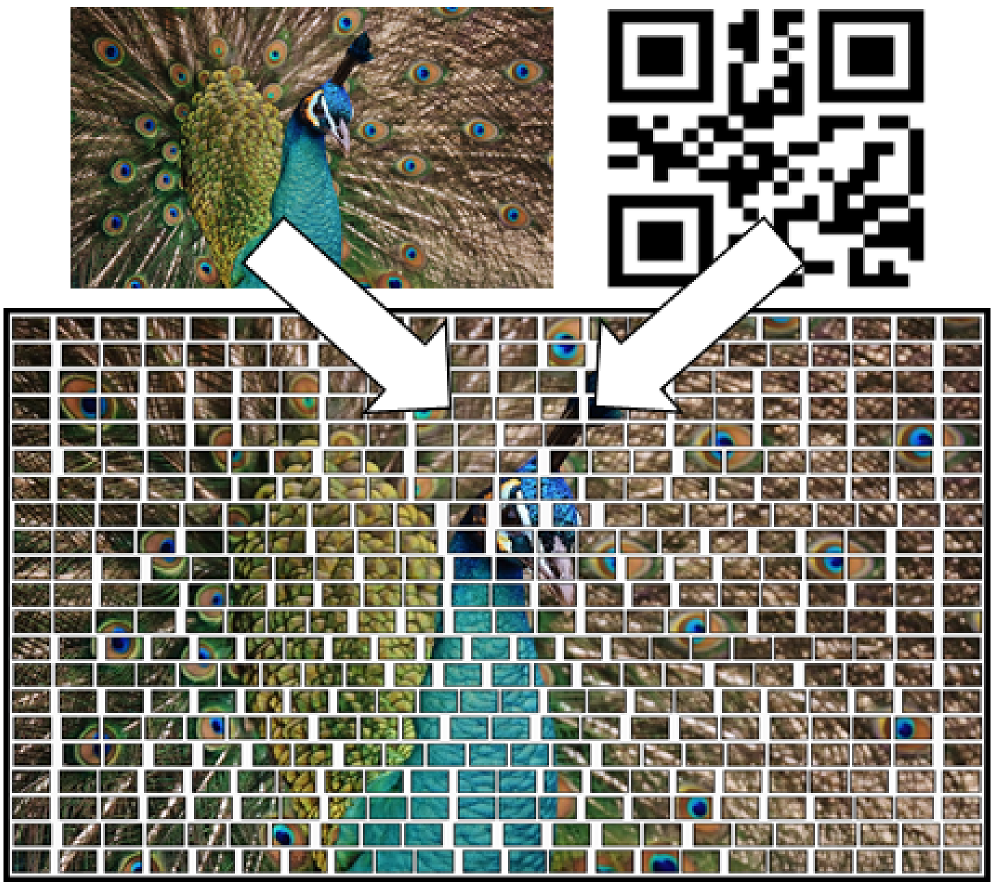


Figure 8.1: Design of the proposed AR marker

Application design

The marker detection feature is a core layer that is responsible for identifying secret data from the custom markers. For the detection process to operate efficiently, a high-resolution camera is required, which is usually the back-camera of most of the smartphones. Due to this requirement, this layer is not active throughout the application usage, but is only called when the users request instead.

Besides the other three buttons (“Backward”, “Forward”, and “Return”), a new button named “Marker Detection” is added. The button is a trigger for the marker detection layer, which switches the application to camera mode. In this mode, users can see all image that the smartphone back camera is capturing. If the users point the camera to any of the custom markers, the system will receive the secret data from it.

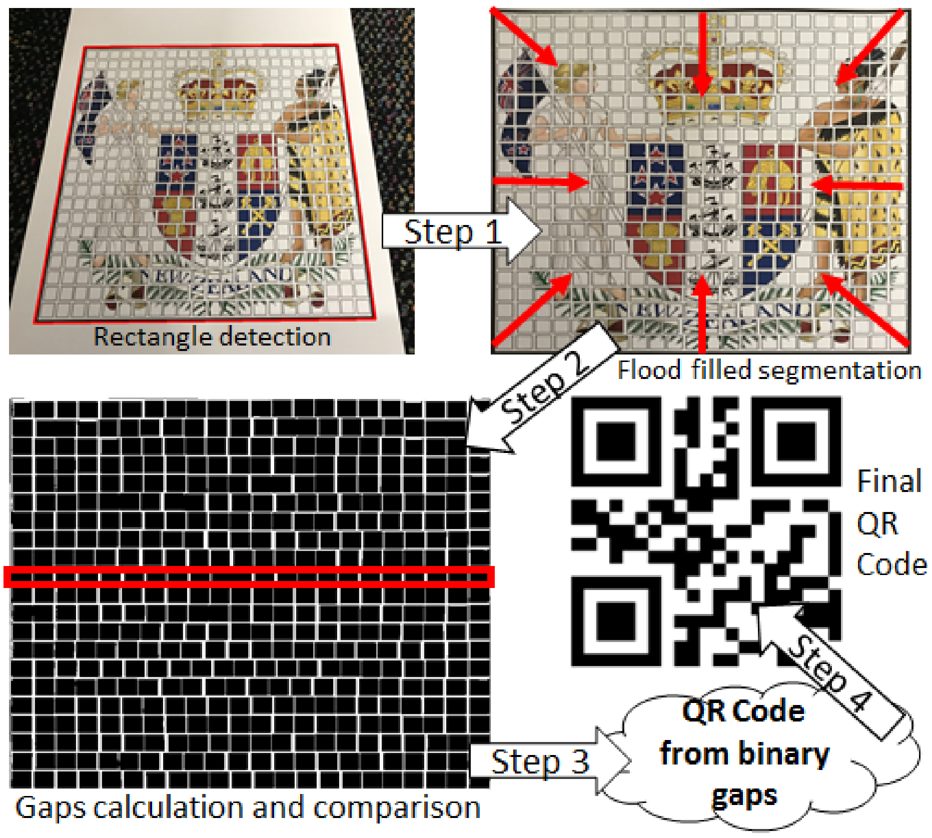
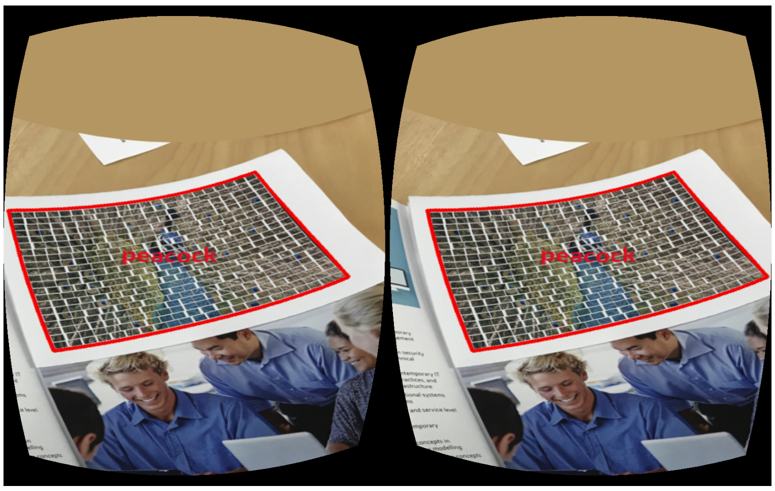


Figure 8.2: Steps of detection and decryption process

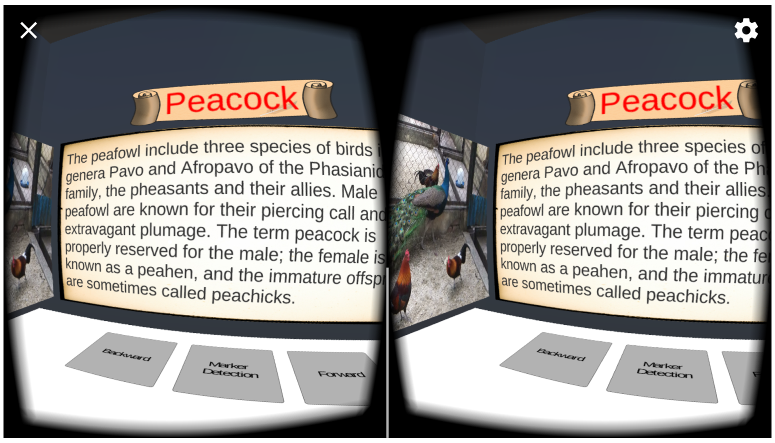
All information related to the data will then be displayed to the users.

In Figure 8.3, the camera captures a marker that has a picture of a peacock (previously shown in Figure 8.1), which contain a tag named “peacock”. The application then automatically displays text and images related to this tag. All of the displayed images are in stereoscopic format, which are retrieved using the same 3D contents classification approach of this project. The related text is retrieved from the first paragraph of the Wikipedia page that has the tag as the article name.

The “Backward” and “Forward” serve as a basic navigation function, which helps the users to browse related stereoscopic pictures that the system has found.



(a) Marker detection mode


Peacock

The peafowl include three species of birds i
genera Pavo and Afropavo of the Phasianid
family, the pheasants and their allies. Male
peafowl are known for their piercing call an
extravagant plumage. The term peacock is
properly reserved for the male; the female is
known as a peahen, and the immature offsp
are sometimes called peachicks.

(b) Presentation mode

Figure 8.3: Screenshot from the system client demo

8.1.2 Personalised stereoscopic 3D gallery

This feature has been published in Int. Conf. Image Vision Computing New Zealand (IVCNZ) 2017 by the researcher (Tran, Nguyen, Le & Yan, 2017).

Background

With the rapid development of mobile hardware in the last ten years, most of the current smartphones are equipped with many modern features such as Wi-Fi, 3G, GPS, high-resolution screens, and multiple cameras. The addition of the camera has made smartphones become a popular way of capturing pictures. While 2D pictures can be taken easily, acquiring 3D photos requires two side-by-side cameras, a camera setup which is uncommon on the market. Moreover, viewing these 3D photos also needs the use of specialised gadgets, which is either 3D glasses or parallax barriers attached with the phone display.

The purpose of this feature is to create an interactive public system that allows the users to view and share their 3D gallery. These 3D contents can be captured with any smart-phone on the market, as long as they are equipped with a camera. The description of this feature can be highlighted in four bullet points:

- It is an online system which the pictures from phones can be uploaded to.
- The system contains stereo image rectification (horizontal alignment) that utilises uncalibrated methods.
- To convert the captured pictures to visualised VR image pairs. The system uses the stereoscopic content classification algorithm.
- Viewing these captured 3D contents are the same as viewing the stereoscopic contents retrieved by the application

Application design

This feature make use of both the local system and the online storage system of the application. The interaction between these two main modules are described in Figure 8.4.

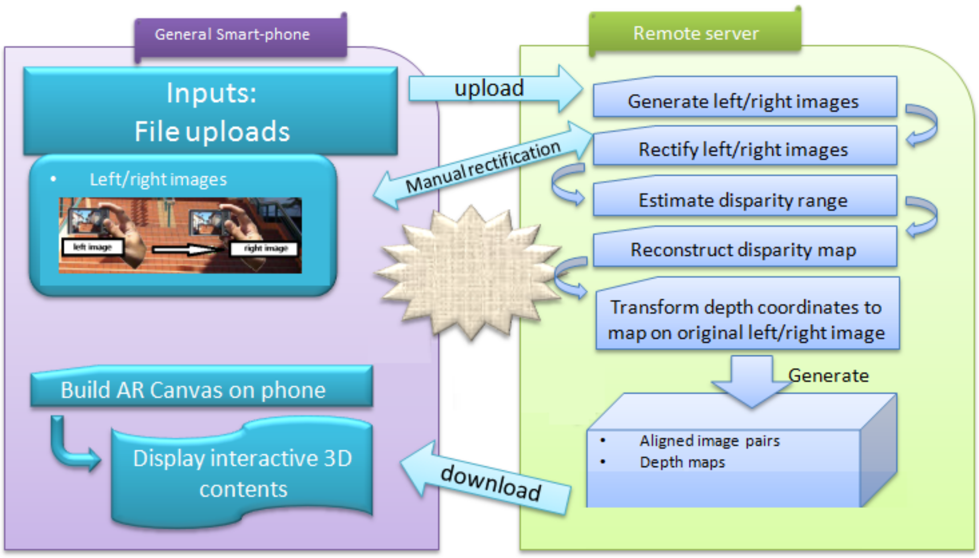


Figure 8.4: Basic processing steps: users upload smart-phone images to a remote server and receive a 3D description of the depicted scene.

When the application is installed on the smartphone, a unique private key is generated. This key is used as way to distinguish users' 3D gallery.

In order to capture 3D photos with the application, a new functionality called 3D photo capturing is added. This function requires the users to take two angle photo shots of the same object. After that, these two images are converted into a side-by-side stereoscopic photo that can be viewed through the application. These photos will be then tagged with the private key and uploaded to the online storage.

Viewing these captured 3D photos is similar to the stereoscopic content viewing feature of the application. The users can use the same navigation buttons to browse through their photos. For gallery sharing, inputting the private key from other smartphones will redirect the users to the 3D gallery that these smartphones contain.

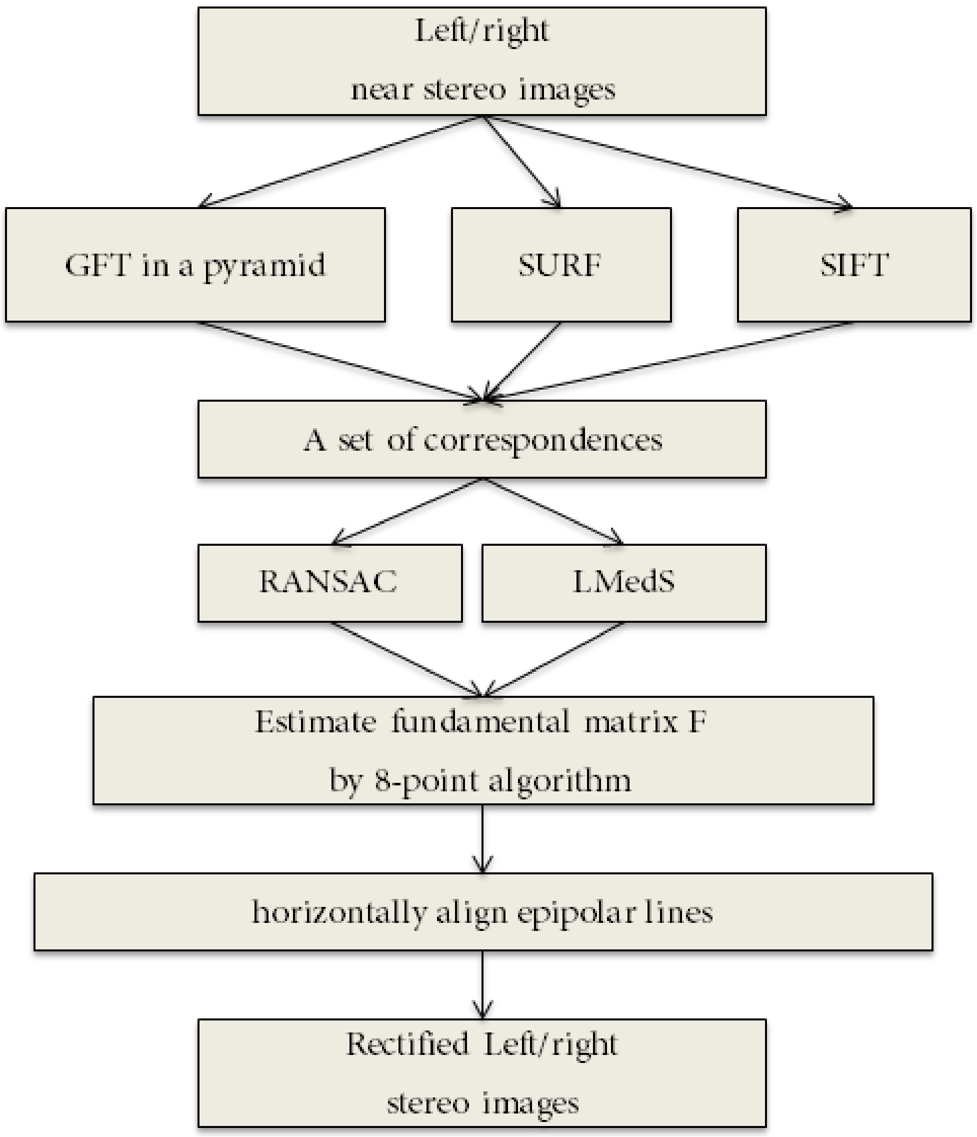


Figure 8.5: Possible processes of the uncalibrated stereo image rectification

8.1.3 Virtual rollercoaster and environment decoration

Initially the project aimed to create a virtual rollercoaster that allow users to travel through variety of virtual environments. For example, the users could go to the jungle to see more photos about animals. However, upon discussing with the supervisor, the constraint of resource does not allow this feature to complete in the timeframe.

After all of the critical issues are fixed, free 3D models from Unity3D library will be used to create multiple placeholder environments for the application. These areas will also be used as testing grounds for the virtual rollercoaster feature.

Depend on the resource and developers, these two features will be developed continually or stopped as proof of concepts.

8.1.4 Porting to iOS

One of the main reason that the application development started with Android environment is the high cost of iOS development. In order to run the demo on the iPhone and iPad systems, the researcher has to either pay a developer fee or try to get a jailbroken model. Moreover, Google VR platform is also optimised for the Android, which allows the compatibility issue to be minimised.

After the application is finalised and published to Google Play Store or AUT University server, the researcher will gather more feedbacks from the public users. If there are great interest in the application, it can be ported to the iOS and published to Apple App Store.

8.2 Conclusion

This thesis describes an online VR application that automatically retrieve stereoscopic contents on the Internet based on user requests. The front-end side of the application is developed with a combination of Unity3D engine and Google VR SDK. In order to retrieve the correct stereoscopic format, multiple feature matching algorithms are utilised to create a state-of-the-art content classification process for the application.

Based on the small-scale evaluation performed at the end of the project, the final prototype of the application proves to be successful in delivering a quality stereoscopic content viewing experience. However, besides some graphical and performance issues, the content classification process is still not implemented properly. This is due to the time and resource limitation of the project. After fixing these problems, the application will have a huge potential of becoming a great VR tool for both entertainment and education. With it, users around the world could easily enjoy millions of stereoscopic contents that are available on the Internet.

References

Amadeo, R. (2016, October). *Daydream VR hands-on: Google's "dumb" VR headset is actually very clever.* Retrieved from https://arstechnica.com/gadgets/2016/10/daydream-vr-hands-on-googles-dumb-vr-headset-is-actually-very-clever/

Amazon. (2016, December). *Samsung SM-R323nbkaxar Gear Virtual Reality 2016 for Galaxy S7, Galaxy S7 edge, Galaxy Note5, Galaxy S6, Galaxy S6 edge, Galaxy S6 edge+ (International Version, No Warranty) - Black.* Retrieved from https://www.amazon.com/Samsung-SM-R323NBKAXAR-Virtual-International-Warranty/dp/B01M0YDS2A/

AMNH. (2015, May). *Museum Joins With Google to Launch Virtual Reality Visits.* Retrieved from http://www.amnh.org/explore/news-blogs/news-posts/museum-joins-with-google-to-launch-virtual-reality-visits

Apple Inc. (2017, March). *Welcome to iOS.* Retrieved from https://support.apple.com/ios

Bavor, C. (2016). folding a virtual journey with google cardboard. *Google. Zugriff am*, *29*, 2016.

Bay, H., Ess, A., Tuytelaars, T. & Gool, L. V. (2008). Speeded-up robust features (SURF). *Computer vision and image understanding*, *110*(3), 346–359.

Bicego, M., Lagorio, A., Grosso, E. & Tistarelli, M. (2006, Jun). On the use of sift features for face authentication. In *Proceedings of ieee computer vision and pattern recognition workshop* (pp. 35–35). New York, USA.

British Museum. (2017, April). *New Virtual Reality tour of the Museum with Oculus.* Retrieved from http://blog.britishmuseum.org/new-virtual-reality-tour-with-oculus/

Busby, J., Parrish, Z. & Wilson, J. (2009). *Mastering unreal technology, volume i: Introduction to level design with unreal engine 3.* Pearson Education.

Chaffey, D. (2016). *Mobile marketing statistics compilation.* Retrieved from http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

Churchland, P. S. & Sejnowski, T. J. (1994). *The computational brain* (1st ed.). Cambridge, MA, USA: MIT Press.

Cobb, . N. S. C., S. V. G. (1998). Static posture tests for the assessment of postural instability after virtual environment use. *Brain Research Bulletin*, 459–464.

Crecente, B. (2016, May). *The future is Unreal (Engine).* Retrieved from https://www.polygon.com/a/epic-4-0/the-future-is-unreal-engine

David M. Hoffman, Ahna R. Girshick, Kurt Akeley & Martin S. Banks. (2008, March). Vergence–accommodation conflicts hinder visual performance and cause visual fatigue. *Journal of Vision, 8, 1–30.*

Dogtiev, A. (2016). *App store statistics roundup.* Retrieved from http://www.businessofapps.com/app-store-statistics-roundup/

Eisenberg, A. (2016, June). *Unity vs. Unreal Engine - Best VR Gaming Platforms.* Retrieved from https://appreal-vr.com/blog/unity-or-unreal-best-vr-gaming-platforms/

Emspak, J. (2016, March). *What Is Virtual Reality?* Retrieved from https://www.livescience.com/54116-virtual-reality.html

Epic Games. (2017, February). *Features.* Retrieved from https://www.unrealengine.com/features

Faulkner, C. & phones, . M. (2017, October). *Google Daydream release date, news and features.* Retrieved from http://www.techradar.com/news/phone-and-communications/mobile-phones/android-vr-release-date-news-features-1321245

Ford, E. (2017, June). *Children's author brings books to life through augmented reality.* Retrieved from http://www.stuff.co.nz/entertainment/books/93967200/childrens-author-brings-books-to-life-through-augmented-reality

Google. (2015a, April). *Google cardboard - Technical specification sersion 2.0.* Retrieved from https://vr.google.com/cardboard/downloads/wwgc_manufacturers_kit.zip

Google. (2015b, April). *Physiological considerations - Designing for Google Cardboard.* Retrieved from https://www.google.com/design/spec-vr/designing-for-google-cardboard/physiological-considerations.html#physiological-considerations-head-tracking

Google. (2015c, April). *QR code specifications.* Retrieved 2017-06-14, from https://support.google.com/cardboard/manufacturers/answer/6321902

Google. (2015d, April). *QR code specifications.* Retrieved 2017-06-14, from https://support.google.com/cardboard/manufacturers/answer/6321902

Google. (2016a, July). *10 Virtual Reality Tours You'll Love - Google Arts & Culture.* Retrieved from https://www.google.com/culturalinstitute/beta/theme/mwJiZHf_Y7FfLg

Google. (2016b, September). *Getting Started with the NDK | Android Developers.* Retrieved 2017-04-27, from https://developer.android.com/ndk/guides/index.html

Google. (2017a, May). *Android 7.1 Compatibility Definition.* Retrieved from `https://source.android.com/compatibility/android-cdd`

Google. (2017b, May). *Daydream – Phones.* Retrieved from `https://vr.google.com/daydream/phones`

Google. (2017c, March). *Google VR NDK Overview | Google VR.* Retrieved from `https://developers.google.com/vr/android/ndk/gvr-ndk-overview`

Google. (2017d, April). *Google VR SDK for Unity.* Retrieved from `https://developers.google.com/vr/unity/`

Hahn, J. (2015). *Android claims 81.5% of the global smartphone os market in 2014.* Retrieved from `https://www.digitaltrends.com/mobile`

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*(2), 147–160.

Heron, J. (1996, September). 2. Research Method and Participation. In *Co-Operative Inquiry: Research into the Human Condition.* Thousand Oaks, California: SAGE. (Google-Books-ID: gMJbubgmzjgC)

Hoff, W. & Ahuja, N. (1989). Surfaces from stereo: Integrating feature matching, disparity estimation, and contour detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(2), 121–136.

Holmes, O. W. (1859). The stereoscope and the stereograph. *The Atlantic Monthly*, *3*, 738–748.

Horton, J. & Portales, R. (2016, September). 1. Why Java, Android, and Games? In *Android: Game Programming* (pp. 8–10). Packt Publishing Ltd. (Google-Books-ID: EoNcDgAAQBAJ)

Howard, I. P. & Rogers, B. J. (2012, January). 24.1.5 Random-dot stereograms. In *Perceiving in Depth, Volume 2: Stereoscopic Vision.* Oxford University Press. (Google-Books-ID: AG2JAgAAQBAJ)

Jerald, J. (2015). 1.3 What is vr good for?, 2. A history of vr. In J. Jerald (Ed.), *The vr book: Human-centered design for virtual reality.* San Rafael: Morgan & Claypool.

Jonnalagadda, H. (2017, March). *Google has shipped 10 million cardboard vr headsets since 2014.* Retrieved 2017-04-13, from `http://www.androidcentral.com/google-has-shipped-10-million-cardboard-vr-headsets-2014`

Juan, L. & Gwun, O. (2009). A Comparison of SIFT, PCA-SIFT and SURF. *International Journal of Image Processing*, *3*(4), 143–152.

Kolasinski, E. (2014, July). *Simulator sickness in virtual environments (ARI 1027).* Retrieved from `http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA295861`

Korolov, M. (2016, April). *VR Travel: 10 Ways To See the World From Your Living Room.* Retrieved from `https://www.gearbrain.com/vr-travel-samsung-gear-vr-google-cardboard-1737536488.html`

Kuniavsky, M., Goodman, E. & Moed, A. (2012). Chapter 11. Usability Tests - How to Do It. In *Observing the User Experience: A Practitioner's Guide to*

User Research (pp. Amsterdam, Netherlands). Elsevier. (Google-Books-ID: jmj5LCAG6kwC)

Leroy, L. (2016, June). 3.2 Too much depth. In *Eyestrain Reduction in Stereoscopy*. John Wiley & Sons. (Google-Books-ID: 2E9gDAAAQBAJ)

Lowe, D. G. (1999, Sep). Object recognition from local scale-invariant features. In *Proceedings of the ieee international conference on computer vision* (Vol. 2, pp. 1150–1157). Kerkyra, Greece.

Lucas, B. D. & Kanade, T. (1981, Aug). An iterative image registration technique with an application to stereo vision. In *Proceedings of the international joint conference on artificial intelligence* (pp. 674–679). Vancouver, Canada.

Machkovech, S. (2016, May). *How side-mounted LEDs can help fix VR's "tunnel vision" and nausea problems.* Retrieved from `https://arstechnica.com/gaming/2016/05/how-side-mounted-leds-can-help-fix-vrs-tunnel-vision-and-nausea-problems/`

McAllister, D. F. (2006). Stereo and 3D Display Technologies. *Encyclopedia of Imaging Science and Technology*, 1327–1344.

Mullis, A. (2016, June). *Developing with the Google VR SDK and NDK.* Retrieved from `http://www.androidauthority.com/developing-with-the-google-vr-sdk-and-ndk-699472/`

Navarro, A., Pradilla, J. V. & Rios, O. (2012). Open source 3d game engines for serious games modeling. In *Modeling and simulation in engineering.* InTech.

Ng, A. K., Chan, L. K. & Lau, H. Y. (2016, July). Depth Perception in Virtual Environment: The Effects of Immersive System and Freedom of Movement. In *Virtual, Augmented and Mixed Reality: 8th International Conference, VAMR 2016, Held as Part of HCI International 2016, Toronto, Canada, July 17-22, 2016. Proceedings.* Springer. (Google-Books-ID: kBafDAAAQBAJ)

Nguyen, M., Tran, H., Le, H. & Yan, W. Q. (2017). A Tile Based Colour Picture with Hidden QR Code for Augmented Reality and Beyond. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology* (pp. 8:1–8:4). New York, NY, USA: ACM. Retrieved 2018-02-12, from `http://doi.acm.org/10.1145/3139131.3139164` doi: 10.1145/3139131.3139164

Nimodia, C. & Deshmukh, H. (2012). Android operating system. *Software Engineering*, *3*(1), 10.

Novet, J. (2016, May). *How Google's Daydream mobile VR system will work.* Retrieved from `https://venturebeat.com/2016/05/21/google-daydream-standards/`

O'Connor, F. (2015). Google to cardboard developers: Keep it short and simple, and watch out for nausea. *Good Gear Guide*, 2.

Orland, K. (2016, November). *Google Daydream review: The fast-casual restaurant of the VR world.* Retrieved from `https://arstechnica.com/gaming/2016/11/google-daydream-review-the-fast-casual-restaurant-of-the-vr-world/`

Oscillada, J. (2015, May). *Comparison chart of fov (field of view) of vr headsets.* Retrieved from `http://www.virtualrealitytimes.com/2015/05/`

24/chart-fov-field-of-view-vr-headsets/

Pappas, S. (2016, April). *Why Does Virtual Reality Make Some People Sick?* Retrieved from http://www.livescience.com/54478-why-vr-makes-you-sick.html

Parisi, T. (2015). *Learning virtual reality: Developing immersive experiences and applications for desktop, web, and mobile.* Sebastopol: O'Reilly Media, Inc.

Parsons, J. (2017, January). *Augmented reality children's books bring bedtime stories to life in 3d.* Retrieved from http://www.mirror.co.uk/tech/augmented-reality-childrens-book-brings-9684383

Patterson, R. E. (2015, February). 4.2 Accommodation-Vergence Conflict. In *Human Factors of Stereoscopic 3d Displays* (pp. 33–41). Springer. (Google-Books-ID: MMq3BgAAQBAJ)

Peckham, J. (2017, March). *Samsung Gear VR review.* Retrieved from http://www.techradar.com/reviews/samsung-gear-vr-2017

Petrovan, B. (2016, November). *Google clarifies requirements for Daydream VR-ready phones.* Retrieved from http://www.androidauthority.com/daydream-vr-ready-phones-specs-727780/

Pluralsight. (2014, November). *Unreal Engine 4 vs. Unity: Which Game Engine Is Best for You?* Retrieved from https://www.pluralsight.com/blog/film-games/unreal-engine-4-vs-unity-game-engine-best

Reed, I. S. & Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, *8*(2), 300–304.

Renwick Gallery. (2016, May). *WONDER 360 Virtual Reality App |.* Retrieved 2017-07-04, from http://americanart.si.edu/multimedia/wonder360/

Rollmann, W. (1853). Zwei neue stereoskopische Methoden (Two new stereoscopic methods). *Annalen der Physik (Annals of Physics)*, *166*(9), 186–187.

Samsung. (2017, March). *Specifications | Samsung Gear VR with Controller.* Retrieved from http://www.samsung.com/global/galaxy/gear-vr/specs/

Samsung Electronics. (n.d.). *The Samsung Gear VR - From the website.* (http://www.samsung.com/global/galaxy/gear-vr, visited on Aug 10, 2017)

Scharstein, D. & Szeliski, R. (2003, Jun). High-accuracy stereo depth maps using structured light. In *Proceedings of the ieee computer society conference on computer vision and pattern recognition* (Vol. 1, pp. 195–202). Wisconsin, USA.

Sharples, C. S. M. A. . W. J. R., S. (2008). Virtual reality induced symptoms and effects (vrise): Comparison of head mounted display (hmd), desktop and projection display systems. *Displays - Volume 29, Issue 2*, 58–69.

Shi, J. & Tomasi, C. (1993, Jun). Good features to track. In *Proceedings of the computer society conference on computer vision and pattern recognition* (pp. 593–600). New York, USA.

Simonite, T. (2015, November). *Inside Google's Plan to Make Virtual Reality Mainstream Before Facebook Can.* Retrieved from https://

www.technologyreview.com/s/542991/google-aims-to-make-vr-hardware-irrelevant-before-it-even-gets-going/

Smith, S. L. (2017, March). *Samsung Gear VR Guide: Everything You Need to Know.* Retrieved from http://www.tomsguide.com/us/samsung-gear-vr-guide,review-3267.html

Smithsonian National Museum of Natural History. (2016, May). *About These Tours: NMNH Virtual Tour.* Retrieved 2017-07-04, from http://naturalhistory.si.edu/vt3/about.htm

Sony Interactive Entertainment. (2012, November). *Wonderbook™: Book of Spells.* Retrieved from https://www.playstation.com/en-nz/games/wonderbook-book-of-spells-ps3/

Stack Overflow. (2017, March). *Stack Overflow Developer Survey 2017.* Retrieved from https://stackoverflow.com/insights/survey/2017/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2017&utm_content=social-share

Stanney, K. R. S., K. M. (2009). Simulation sickness. In J. A. W. M. M. Peter A. Hancock Dennis A. Vincenzi (Ed.), *Human factors in simulation and training.* Boca Raton: CRC Press.

Sural, I. (2017). Chapter 10 mobile augmented reality applications in education. In A. H. Kurubacak Gulsun (Ed.), *Mobile technologies and augmented reality in open education* (pp. 200–201). Hershey: IGI Global.

Tavakkoli, A. (2015, August). A history of unreal technology. In *Game Development and Simulation with Unreal Technology.* CRC Press. (Google-Books-ID: F4BjCgAAQBAJ)

Tikanmäki, A. & Röning, J. (2011). Markers–toward general purpose information representation. In *IROS2011 workshop: knowledge representation for autonomous robots.*

Tomasi, C. & Kanade, T. (1991). *Detection and tracking of point features.* School of Computer Science, Carnegie Mellon Univ.

Tran, H., Nguyen, M., Le, H. & Yan, W. Q. (2017). A Personalised Stereoscopic 3d Gallery with Virtual Reality Technology on Smartphone. In *Int. Conf. Image Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4 Dec - 6 Dec.*

Tuytelaars, T. & Mikolajczyk, K. (2008). Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, *3*(3), 177–280.

Unity Technologies. (2015a, November). *Manual: Sprites.* Retrieved 2017-06-14, from https://docs.unity3d.com/Manual/Sprites.html

Unity Technologies. (2015b, July). *Roll-a-ball tutorial.* Retrieved from https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial

Unity Technologies. (2016a, August). *Fast Facts.* Retrieved from https://unity3d.com/public-relations

Unity Technologies. (2016b, May). *UI Events and Event Triggers.* Retrieved from https://unity3d.com/learn/tutorials/topics/

`user-interface-ui/ui-events-and-event-triggers`

Unity Technologies. (2017a, March). *Editor.* Retrieved from `https://unity3d.com/unity/editor`

Unity Technologies. (2017b, June). *Manual: Sprite Editor.* Retrieved 2018-01-09, from `https://docs.unity3d.com/Manual/SpriteEditor.html`

Unity Technologies. (2017c, June). *Scripting API: Bounds.* Retrieved 2018-01-09, from `https://docs.unity3d.com/ScriptReference/Bounds.html`

Unity Technologies. (2017d, June). *Scripting API: Camera.* Retrieved 2017-06-14, from `https://docs.unity3d.com/ScriptReference/Camera.html`

Unity Technologies. (2017e, March). *Unity 5.6.* Retrieved from `https://unity3d.com/unity/whats-new/unity-5.6.0`

View-Master. (2017, May). *View What's Possible.* Retrieved 2017-07-04, from `https://www.view-master.com/en-us`

Vincent, J. (2016, July). *Google's app for touring virtual art galleries now supports Cardboard VR.* Retrieved from `https://www.theverge.com/2016/7/20/12234578/google-arts-culture-app-cardboard`

VR Status. (2016, May). *Unreal Engine VS Unity.* Retrieved from `https://www.vrstatus.com/news/unreal-engine-vs-unity.html`

WebVR. (2017, March). *Bringing Virtual Reality to the Web.* Retrieved 2017-07-04, from `https://webvr.info/`

Wee, A. (2016). Huawei digs into google daydream: Vr gadgets will be available by this year.

Weibel, P. (2005, May). VI. Vision Machines: Perception of Illusory Bodies and Illusory Movement. In *Beyond Art: A Third Culture: A Comparative Study in Cultures, Art and Science in 20th Century Austria and Hungary.* Springer Science & Business Media. (Google-Books-ID: xkk6U42Zl_sC)

Wendy Powell, P. B. M. C. J. U., Vaughan Powell. (2016). Getting around in google cardboard - exploring navigation preferences with low cost mobile vr. In *2016 IEEE 2nd Workshop on Everyday Virtual Reality (WEVR 2016).* Greenville: IEEE.

Wheatstone, C. (1838). Contributions to the physiology of vision on some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical transactions of the Royal Society of London*, *128*, 371–394.

xdf103. (2016, July). *How to target different contents for each eye?* Retrieved from `https://github.com/googlevr/gvr-unity-sdk/issues/263`