University of Dublin

TRINITY COLLEGE

The Visualisation of Internal Components through Diminished Reality Techniques

Davy Nolan
B.A.(Mod.) Computer Science
Final Year Project April 2021
Supervisor: Dr. John Dingliana

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

DECLARATION

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

_____ _____

Name Date

Acknowledgements

I would like to extend a sincere thank you to my project supervisor, Dr. John Dingliana, for the time and knowledge he provided me throughout the course of this project. He gave me indispensable advice and guidance during the development of this project. His expertise and patience allowed me to have successful results and an overall very enjoyable experience.

I am also grateful for my family and close friends for supporting me through this challenging process and encouraging me to power forward through the current unprecedented times.

Table of Contents

Table of Figures

Abstract

Augmented reality (AR) typically consists of adding virtual graphical objects to the real world. In some cases, there is a need for the opposite also known as diminished reality (DR). This is the process of visually removing a real object from the real world. This project aims to utilise both Augmented and Diminished Reality to help visualise the internal structures of real-world objects through two techniques: the cutaway method and the transparency method. This project outlines the creation of an application run by Unity Engine, which will be used to explore these methodologies.

1 Introduction

1.1 Overview

Augmented Reality (AR) is the process of overlaying virtual graphical objects onto the real world in order to enhance or improve one's experience. The topic of AR is more relevant than ever and is growing at an alarming rate with new technology such as Microsoft's HoloLens AR headset. With AR becoming more accessible and widely accepted, new uses of the technology are being discovered every day whether it be from the gaming industry or the construction industry.

The existence of AR has opened up a new branch of Virtual Reality known as Diminished Reality (DR). DR can be seen as the opposite of AR, in the sense that AR is adding graphical objects to the world and DR is virtually removing objects from the real world. DR has many applications mainly in the interior design industry to visualise virtual furniture and in the construction industry to visualise underground structures and the internals of buildings. Furthermore, DR is also recognised in engineering to analyse internal parts of machinery. This project makes use of the cutaway [28] DR technique to visualise the internal parts of different objects. This technique entails removing a hole from the targeted object and then displaying the desired internal part, giving the illusion of a cutaway. The transparency method of internal visualisation is also explored in this project to compare and contrast this approach with the cutaway technique.

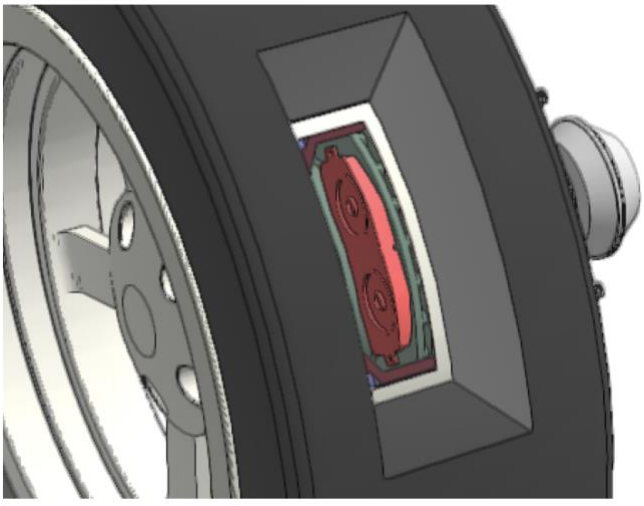

Figure 1.1: Cutaway of a car tyre. Source: [28]

1.2 Structure

1.2.1 Chapter 2

This chapter discusses the background of the project, outlining the main motivations behind it. Specific terminology used throughout this project is researched in this chapter and existing solutions are explored through a literature review.

1.2.2 Chapter 3

This chapter explains how the project application was implemented. The various technologies used throughout the project are listed, with each of their purposes outlined. The creation of the Unity application is fully explained, describing how the 3D models were constructed and how the lighting and shaders work. This chapter goes into depth with the cutaway and transparency techniques and how they were implemented in AR. The different approaches to computer vision and tracking are also demonstrated. This chapter also justifies how and why the project concept and application have changed over the course of the last year.

1.2.3 Chapter 4

This chapter discusses the various results of the project, outlining what went well and what did not. Future work for this project is also suggested, mentioning what could be improved given more time.

2 Background and Research

2.1 Diminished Reality

2.1.1 What is Diminished Reality?

Diminished Reality (DR) is a term used to envelop the methodologies and techniques used for concealing, eliminating, and looking through objects in a perceived environment in real time to diminish the reality [20]. It is practically the opposite of Augmented Reality (AR) as AR involves adding a virtual object to the real world and DR entails virtually removing an object from the real world.

2.1.2 Current Uses of Diminished Reality

At the moment, DR is primarily being used to enhance an AR experience in the world of interior design. AR is being used in this industry to allow users to place virtual furniture and decorations in their homes to visualise how it would look before making the purchase. Using only AR for this only works well if the room is already cleared of furniture to make way for the virtual impression. This is where DR is used to hide the real-world obstacles before displaying the virtual object with AR. This combination of AR and DR is widely known as Mediated Reality [3], a term coined by MIT researcher Steven Mann in 1994.

Figure 2.1: Using Mediated Reality to display virtual furniture. Source: [3]

This application of DR is carried out by a process known as “in-painting” [1]. This method works to mask the real-world object by taking the surroundings of the object into account, replacing the object with the colours and textures around it, rendering the object invisible.

The concept of Mediated Reality is also beginning to be used in the construction industry for builders to virtually clear lots to overlay the visualised building plans.

2.2 Visualisation Techniques

2.2.1 Transparency

The transparency technique of internal visualisation is a widely popular method to display the inner components of an object. Designers utilise this technique to show occluded internal information by making the shell of the component transparent, allowing the user to peer into the object [25]. In the example in Figure 2.2 below, a car can be seen where the outer body is transparent, exposing the internal mechanical systems.

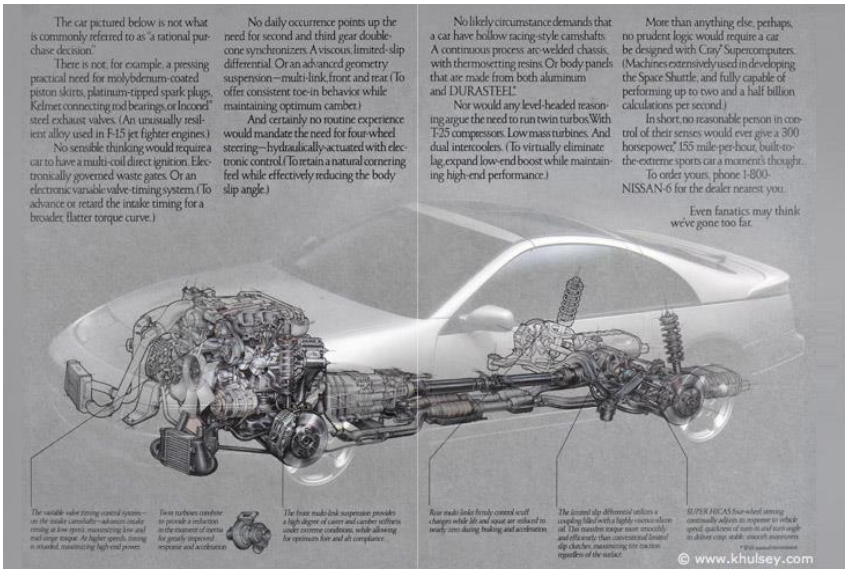

Figure 2.2: Transparency visualisation of the inner mechanics of a car. Source: [2]

This poses a challenge when trying to implement the transparency technique in AR as the transparency of the shell of a real-world object (parent object) cannot be changed. However, the transparency of the components to be displayed within the parent object can be changed, since they are implemented virtually in the first place. As seen in Figure 2.3 below, the parent object represents the real-world object in the AR. The internal component is the object to be virtually displayed within the parent object. If the internal object is opaque at the time of rendering, it will render over the parent object, appearing as though it is outside the parent object. However, making the internal object transparent creates the illusion that the

internal component is actually inside the parent object [29]. Rendering an object transparent in AR is known as alpha blending.

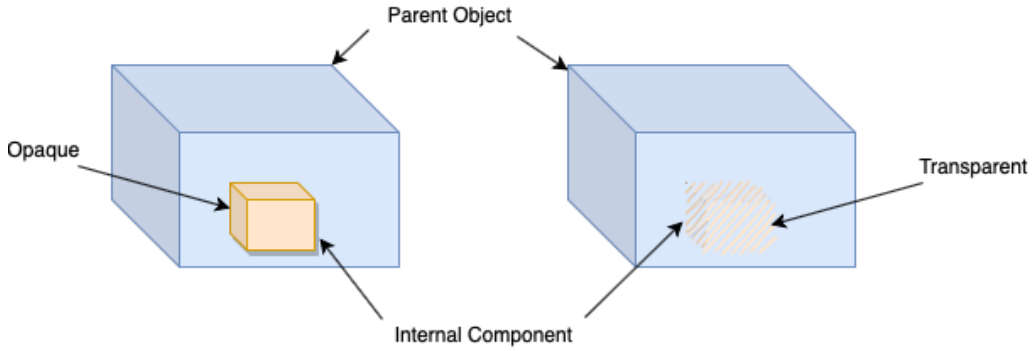

Figure 2.3: Opaque internal part appears as though it is outside the parent object (left). Transparent internal part appears as though it is inside the parent component (right).

Creating this illusion successfully in AR, the internal component's position in relation to the parent object must be taken into account. Inaccurate rendering of occluded objects over real-world objects can lead to a complete misunderstanding of the visualisation and poor perception of the depth and dimensions of the object's components [19].

2.2.1.1 Alpha Blending

Alpha blending is a technique used in computer graphics. where the alpha channel and other layers in an image are combined to show translucency [21]. The alpha channel represents 256 levels of translucency, 0 being fully transparent and 255 being opaque. This information is input within the RGBA value in the shader file, deciding the colour and translucency of an object.

2.2.2 Cutaway

The cutaway method of internal visualisation is a more precise illustration technique. The designer must define and mark out an area or section cut plane to slice objects into parts [25]. This method can be used to focus on specific internal parts of an object, removing occluding parts blocking the view of the internal parts.

This technique is usually done by cutting a planar cross-section through an object, similar to in Figure 2.4. Implementing this form of cutaway would involve the complete removal of the real-world object and replacement with a virtual copy of that object with the cross-section cutaway. This project aims to perform the cutaway visualisation on a real-world object

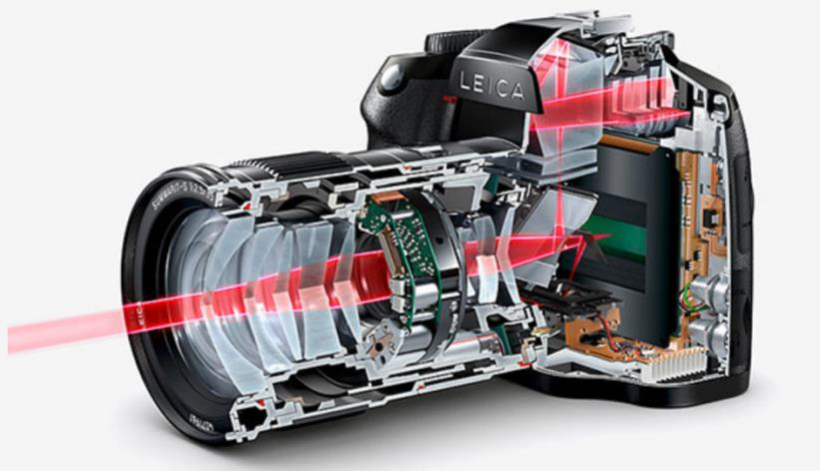

Figure 2.4: Camera diagram using the cross-section cutaway technique for internal visualisation. Source: [13]

without having to virtually remove the entire object through DR. This can be fulfilled by cutting smaller “holes” away from the object, leaving the rest of the object untouched. This operation is outlined in Figure 2.5 below, with the diagram on the left depicting the real-world parent object with the virtual internal part being displayed without a cutaway hole. This

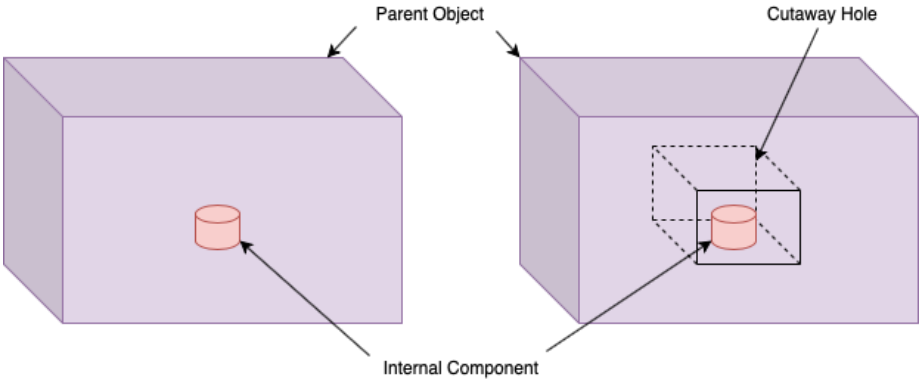

Figure 2.5: Diagram of the cutaway hole technique in AR.

creates no illusion of depth and appears as though the internal component is floating outside the parent object. The diagram on the right shows the same objects except this time with a cutaway hole. Using DR techniques to cut this hole from the object and then displaying the part within the hole, creates an illusion of depth and truly looks like the internal part is inside of the parent object.

2.2.3 Pros and Cons

Although both techniques of internal visualisation are effective in illustrating the inner components of an object, both methods are different and come with their own positive and negative aspects.

The transparency technique does not require the methodology of DR, since nothing needs to be virtually removed from the real world. All that is required is the editing of the alpha value of the internal component to be displayed. With the cutaway hole technique, a cavity has to be virtually removed from the real-world object, meaning there is a need for DR. Therefore, the cutaway method is arguably more difficult and time-consuming to implement.

Another positive feature of the transparency approach is that it reveals the true complexity of the internal structure of an object, showing all of the components and how they interact with one another. However, this has a downside of the parts being burdensome to distinguish from one another as they all occlude each other. On the other hand, the cutaway hole technique has the ability to focus on a specific internal element by removing the unnecessary occluding parts, making it effortless to differentiate the specified part and surrounding parts.

With the cutaway approach, the internal components are more detailed since they can be fully opaque with their own textures and materials. However, with the transparency technique, since the internal parts are translucent, it is quite difficult to perceive accurate details.

Lastly, the transparency technique grants the user the ability to view the internals in a 360° view, but the cutaway hole method allows the user to view the internals from only the opening of the hole, limiting the possible points of view.

2.3 Real-World Applications

2.3.1 Engineering

The cutaway technique of visualisation is frequently used in the engineering industry. In design apparatus, it is imperative that engineers are able to see each internal part and their

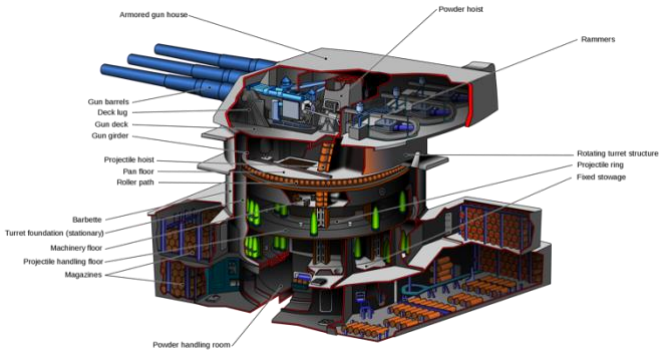

Figure 2.6: Cutaway visualisation in engineering. Source: [8]

correct proximity to neighbouring parts. The cutaway technique is powerful in this field as it allows the user to focus on certain components without occluding the view of others. In Figure 2.6 above, you can see a wedge-shaped cutaway in a piece of machinery, clearly revealing the internal parts, allowing the engineer to label each part accordingly.

2.3.2 Medical

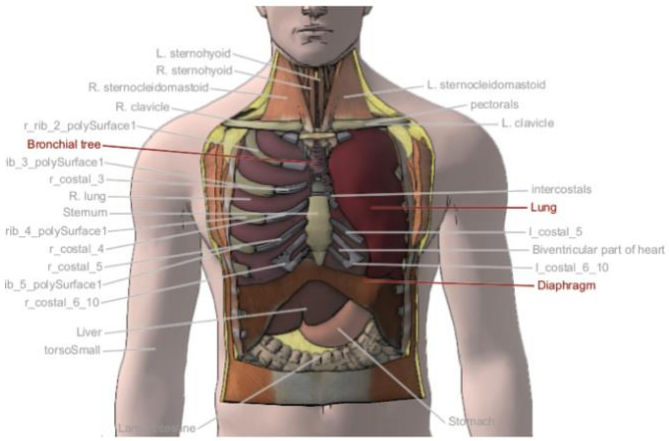

Figure 2.7: Cutaway visualisation of human thorax. Source: [28]

The cutaway method is also seen to be used in medicine. Figure 2.7 above depicts anatomical visualisation of the male torso, with each organ displayed clearly and labelled correctly. This is a perfect example of how the cutaway technique can be used to help teach students of biology-related subjects.

2.3.3 Construction

The cutaway technique is also very prevalent in the world of construction, proving to be advantageous when used along with DR as seen below in Figure 2.9.

Figure 2.9: Cutaway visualisation of road piping. Source: [6]

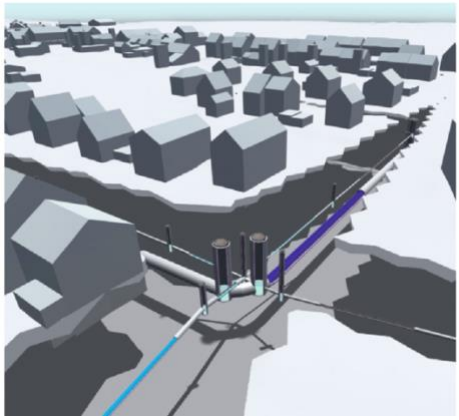

Figure 2.8: Cutaway visualisation of sub-terrain piping. Source: [12]

A camera is being used in this example to track the road. The application makes use of DR to cut a hole from the road surface and then display the appropriate piping inside the hole. This gives the user the illusion of an actual hole, providing them with a sense of depth and accuracy in terms of the pipe's location. This is paramount in this area of work to reduce/completely eliminate the risk of bursting a pipe during construction.

Figure 2.8 depicts how this technique of internal visualisation can be used on a large-scale road junction rather than focusing on one small area. This is helpful for planning for construction.

2.4 Literature Review

There were various pieces of literature used throughout the course of this project to aid in the implementation and understanding of both the cutaway and transparency techniques of internal visualisation. Extracts from these papers are used during this review for both research and explanatory purposes.

2.4.1 Spatial Understanding

Allowing viewers to see the internal visualisation of objects drastically improves their spatial understanding of that object. Not only seeing the internal components is important, but actually being able to make out the relationships between the different parts interacting with one another is crucial for understanding the dimensions of an object (Li et al, 2007)[28]. Since this project is concerned with displaying a virtual projection of the internal structure of

an object onto a real-world object through Augmented and Diminished Reality techniques, understanding the space between these structures is imperative. This requires the full knowledge of the object being visualised including its measurements and dimensions. Zollmann et al (2014) overcame this obstacle by interpreting information from an image taken by a camera. Since their studies purely explored the field of visualising underground piping through outdoor Augmented Reality, their project was at a larger scale. This meant that they could employ the method of using “depth cues” (Zollmann et al, 2007)[29] based on different environmental factors from the image. For this project, the concept used by Zollmann et al cannot be applied as this project works on a much smaller scale of visualisation. Furthermore, exact measurements are required for this project in order to have as accurate internal visualisation as possible.

Li et al (2007) believe that the cutaway method of internal visualisation does not always show the context of the surrounding components in an object. Since the cutaway method only portrays a certain area of the internal structure, the inter-connections are sometimes severed, negatively affecting the viewer’s overall spatial understanding of the object. The cutaway technique is most effective when it focuses on the desired internal part as well as the surrounding connected components to aid spatial understanding (Li et al, 2007). Burns et al (2008)[4] possess a similar ideology, believing that the cutaway obtrusions should expose “objects of interest” at the same time as preserving the context of the infrastructure.

Burns et al (2008) outline the struggle of determining the area of 3D space around the specified internal component. This can be predetermined manually or perhaps it could be done automatically at run time. In fact, Li et al (2007) introduced a feature into their visualisation system where the user can specify the internal part they wish to see, and then a cutaway hole is created automatically, exposing the component for viewing. This implementation of automation poses a challenge in the world of Augmented Reality without having a predetermined set of possible cutaway holes due to the unpredictability of the subject surface at runtime. This concept of automation will be explored throughout the course of this project. Burns et al (2008) believe that the result of automation is that it reduces the problems involved with camera placement and it also enhances the user experience, both very important aspects when it comes to an Augmented Reality application.

On the other hand, with the method of transparent internal visualisation, the focus is on the entire internal infrastructure, and not just a particular region of the object. Zollmann et al (2014), through their X-ray visualisation studies, disregard the importance of areas with the use of alpha blending. This is a method used in computer graphics to adjust the alpha value

of an object's material, making it either more or less translucent. This approach will be used greatly in this project to deliver the transparent view of visualisation.

2.4.2 Occluding Components

When it comes to internal visualisation, components occluding one another are a given. The cutaway technique can be used to reduce the number of occlusions by only revealing the important components within intertwined structures (Li et al, 2007). Li et al (2007) believe that the cutaway holes should be shaped around the geometry of occluding parts so that the viewer has the chance to mentally reconstruct the missing parts of the geometry, aiding their overall understanding. Burns et al (2008) agree with this idea, stating that internal structures can be ranked based on their importance and occlusion should follow suit. Less important components that are occluding the more important parts should be removed in order to expose the desired part. This concept of removing something from a scene to make way for another object greatly concerns Diminished Reality. In order to deliver the cutaway technique in an augmented experience, the surface of a real-world object must be virtually cut away, removing the less important occluding parts in order to reveal to more important component the user wishes to focus on. Doing this without the methodology of Diminished Reality introduces a visual problem of virtual internal objects appearing as though they are floating above the real-world surface (Zollmann et al, 2014). Zollmann et al (2014) found a solution to this problem whilst performing transparent visualisation in Augmented Reality. While trying to visualise subterranean piping, it would sometimes appear as though the virtual pipes were not actually underground. To combat this issue, they thought of the process of "ghost mapping". Using information provided by a camera, this process involved indicating the importance of each pixel in the image and deciding if they should be rendered opaque or transparent based on some criteria. "Edge Mapping" (Zollmann et al, 2014) was used to detect pixels that made up structures with sharp edges and then making them fully opaque. This was used on an image of a zebra crossing with a virtual pipe running underneath. The road markings were detected as edges and therefore made opaque, giving the illusion of the virtual pipe going beneath it. This concept may be explored to create a better transparent visualisation experience in this project.

This project is greatly inspired by the research of the papers discussed in this section, taking into account the techniques used by all the researchers in carrying out the cutaway and transparency techniques of visualisation. This project will take on board these concepts and adapt them in the hopes of introducing them to the world of Augmented and Diminished reality.

3 Method and Implementation

3.1 Technologies Used

3.1.1 Unity

Unity is the engine used to create this project's DR application. This was chosen as it is free to use and due to its ease of implementing different models with different shaders and materials. There is also an abundance of resources on Unity Engine, making it easy to pick up, even for beginners in computer graphics. Unity has a variety of different AR libraries which work fluidly with any project. This graphics engine also runs fantastically on MacOS which is the operating system being used for this project.

3.1.2 Vuforia

Vuforia is an AR software development kit for mobile devices that enables the creation of AR applications that can be easily imported to a Unity project. It uses computer vision technology to recognise and track images and 3D objects in real-time using the device camera. This library was chosen instead of other AR libraries such as AR Kit and AR Foundation due to its ability to track real-time 3D models which is a crucial dependency for this project. Vuforia is also free to use as a developer, making it very accessible.

3.1.3 Blender

Blender is an open-source 3D content creation program. This was used in this project to edit 3D models and to create new models from scratch. Blender was used during the Computer Graphics module last semester, as well as there being a wide variety of online Blender tutorials, so it was very easy to utilise in this project. This program also allows the export of models in FBX format which is required to use the models in Unity.

3.1.4 Open Cascade

Open Cascade CAD Assistant is a 3D CAD viewer, editor, and converter. This was used to view and edit downloaded IGS models and then convert them to OBJ files to be edited in Blender and inevitably imported to Unity. This application's simple and intuitive user interface made it smooth and straightforward to learn to use.

3.1.5 Visual Studio C#

Visual Studio is a development environment that is free to use. It integrates nicely with Unity, allowing the editing of scripts and shaders. These scripts and shaders are the main driving forces behind this project.

3.1.6 Xcode

Xcode is Apple's integrated development environment for MacOS which is required to develop applications for Apple devices. This was used in conjugation with Unity to deploy the main DR application onto an iPhone XR device.

3.2 3D Modelling

3.2.1 Cutaway Hole Creation

3.2.1.1 Hole Interior

The interior of the hole was created using a cube object. In order to give the illusion of looking inside a cube, the normals of the mesh were inverted. Normals are perpendicular reference points to surfaces on 3D models [11]. They represent which direction the polygon

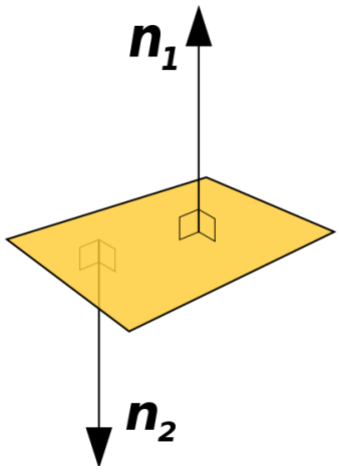

Figure 3.1: A polygon and its two normal vectors. Source: [9]

of an object is facing, therefore defining whether the surface is an inside or outside face. Inverting a normal makes the normal point in the wrong direction, making the outside face the inside face. Inverting all of the normals of a cube makes it appear to the user as if they are viewing the inside of the cube as seen in Figure 3.2 below. To do this in Blender, "back face culling" must be enabled first and then the normals can be flipped in the mesh menu.

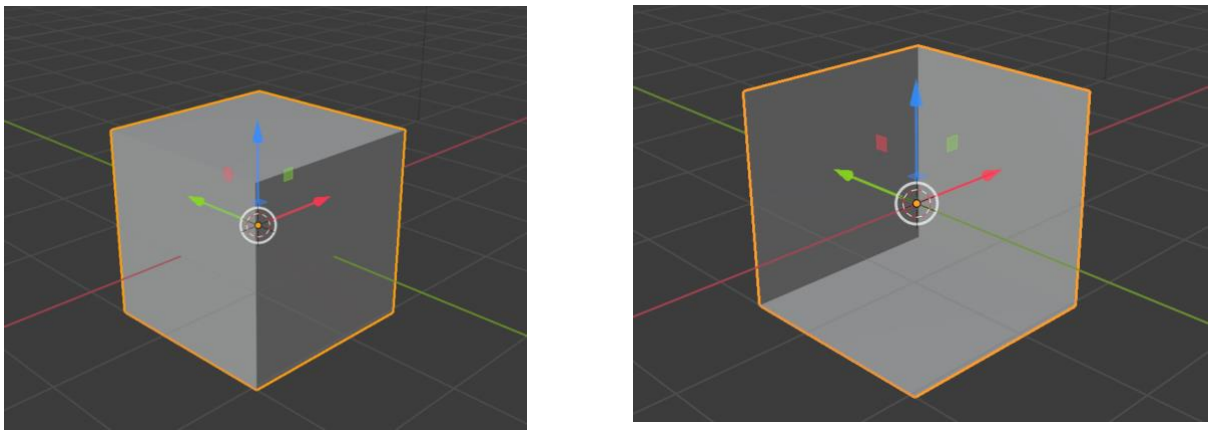

Figure 3.2: Inverting the normals of a cube.

This can also be done with other shapes such as hemispheres, cylinders, and cones as seen in Figure 3.3.

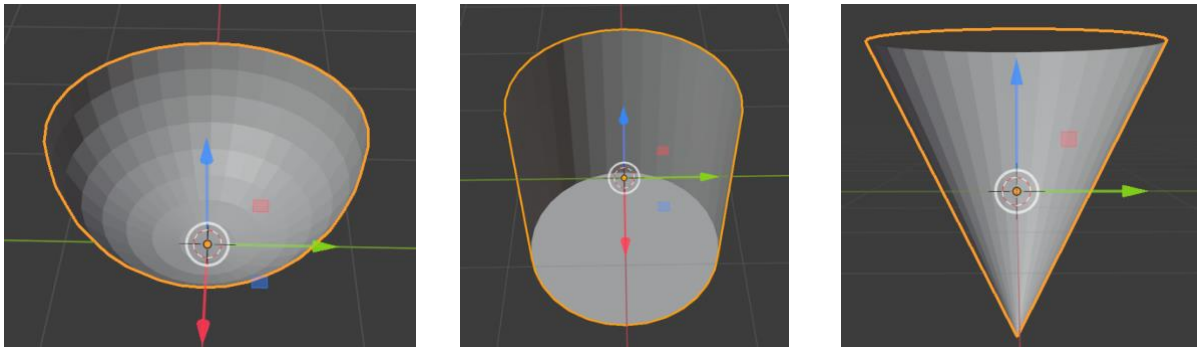

Figure 3.3: Hemisphere, cylinder and cone with inverted normal.

3.2.1.2 Hole Opening

In order to create the illusion of peering into a hole, a separate object had to be created representing the opening of the hole in the surface. This opening should have as little height as possible and cover the whole opening of the hole without hanging over it. For a cube hole on a flat surface, the opening should only be the same dimensions as the face on top of the cube.

Creating an opening on a curved surface is more challenging as the opening must match the curve. This can be done using Boolean modifiers in Blender. Boolean modifiers can be applied to two objects (the target mesh and the modified mesh) to change their mesh as opposed to editing the mesh manually. The three Boolean operations are “intersect”, “union” and “difference”. “Difference” is where the target mesh is subtracted from the modified mesh.

“Union” is the opposite where the target mesh is added to the modified mesh. Intersect is where everything inside both the target mesh and the modified mesh is kept.

Creating a hole opening on a curved surface involves the use of only the “intersect” Boolean modifier [8]. This can be done by creating a copy of the hole and then setting it to the modified mesh for the intersect Boolean modifier, then setting the surface as the target mesh. This process can be seen in Figure 3.4 below.

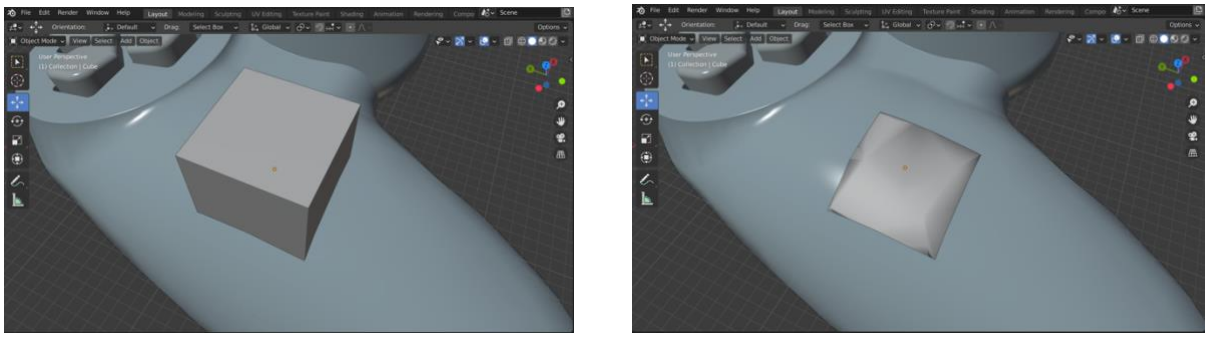

Figure 3.4: Creating a hole opening on a curved surface in Blender using Boolean modifiers.

3.2.2 Models Used

This project made use of two device models to showcase internal visualisation through the cutaway technique. The main reasons these models were chosen are that the devices consist of many different internal parts and both of them were readily available to test upon during the course of this project.

3.2.2.1 DualShock 4 Model

Model source: https://grabcad.com/library/playstation-4-controller-dualshock-4-1

The “DualShock 4” (DS4) is the primary game controller used to operate Sony’s PlayStation 4. The model used was in IGES format and was downloaded from the link above. Since it was in this format, it was edited and then converted to OBJ using Open Cascade CAD Assistant.

The model used also included the internal components of the controller. In Open Cascade, the model was loaded to view all of the different parts making up the DS4. Open Cascade has the ability to remove specified components in the hierarchy of a model so this was how the internal and external parts were separated. They had to be separated as their own distinctive objects because they will later require their own unique materials and shaders in

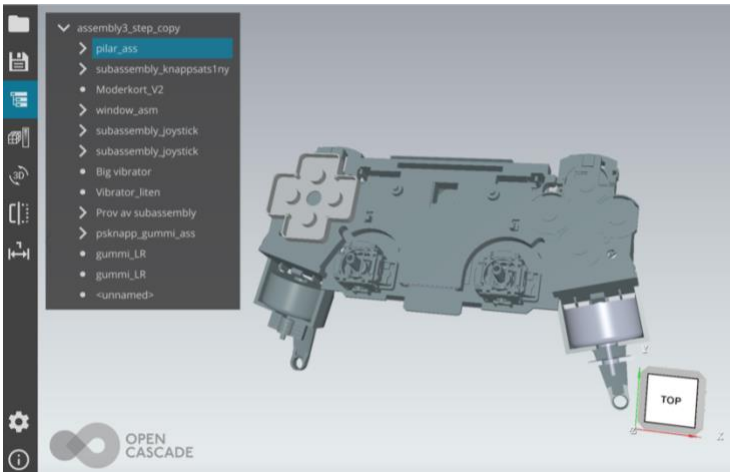

Figure 3.5: Editing the DS4 model with Open Cascade.

the Unity application. Once the internal and external parts of the controller were separated and exported as unique OBJ files, they were imported to Blender to edit further and resize.

3.2.2.1.1 DualShock 4 Holes

The shell of the DS4 controller was imported to Blender so that the holes could be shaped by the shell. The holes created were made using cube and cylindrical meshes. The meshes were first placed around the controller shell in the desired locations. These locations were chosen based on what internal parts were to be shown with the application.

Using the method of creating the hole opening mask as outlined in section 3.2.1.2, Boolean modifiers were applied to the meshes to get the intersection mesh between the cube and the DS4 mesh. In Figure 3.6 below, you can see the hole opening masks being created for the various components of the DS4 controller.

Figure 3.6: Using the intersect Boolean modifier on the target meshes to create the hole opening masks for the DS4.

Since the shell model consists of some unnecessary internal parts, they had to be removed from the newly created mask. This was done by separating the hole opening by “loose parts” in Blender and then simply deleting the unnecessary components, leaving the mask flush with the DS4 shell as seen below.

Figure 3.7: Using the intersect Boolean modifier on the target meshes to create the hole opening masks for the DS4.

Using the same meshes from before and the method described in section 3.2.1.1, the normal of the meshes were flipped to give the illusion of the inside of a hole.

Figure 3.8: Flipping the normals of the meshes to create the hole walls.

This model was now ready to save and export to Unity.

3.3 Lighting and Shaders

3.3.1 Shaders

3.3.1.1 What are Shaders in Unity?

Shaders are widely used in the area of Computer Graphics. They are small scripts that contain the mathematical calculations and algorithms for calculating the colour of each pixel rendered, based on the lighting input and the material configuration [27]. These programs are known as shaders because they are mainly used to control lighting and shading effects. Some shaders even have the ability to allow you to see through objects in a scene (also known as a transparency shader).

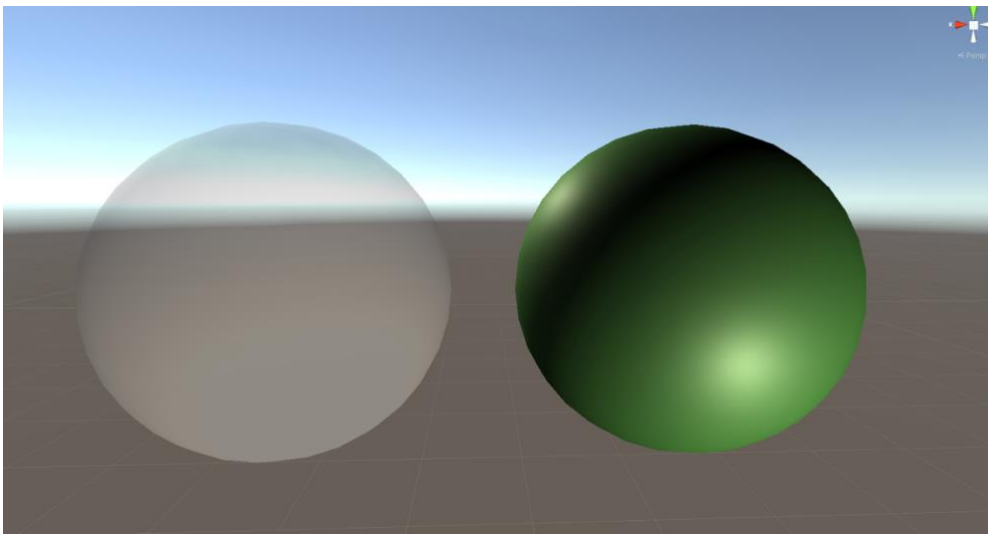

Figure 3.9: Two identical spheres with different shaders.

There are different types of shaders. Fragment shaders output the colour of a singular pixel which makes up a polygon. The Fragment shader program is run repeatedly to generate every pixel making up the polygon being processed by that shader. Another type of shader is the Vertex shader. This is used to calculate the position of the vertices in the image being processed. Both of these shaders are used together in a GPU pipeline to display the final image as seen below in Figure 3.10.

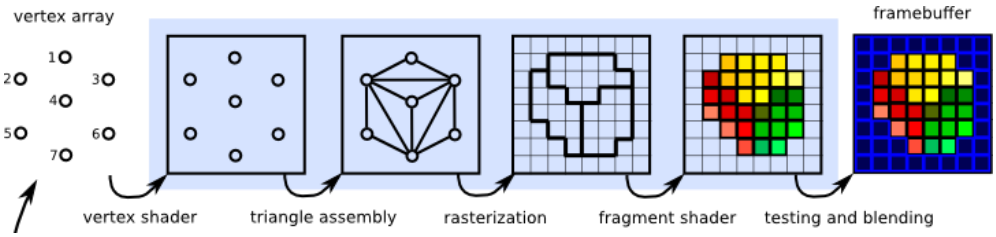

Figure 3.10: GPU pipeline showing the function of the vertex and fragment shaders. Source: [10]

In Unity, the use of both of these separate shaders is not required. Instead, Unity makes use of a Surface shader, a written program that controls surface properties such as texture and colour. The best part of using Surface shaders is that Unity automatically generates code to handle the lighting [22].

Surface shader programs are split up into 3 code blocks. The `Properties` block at the top of the shader file declares different user-defined variables which can be edited in the inspector menu of the material. Examples of these values include the `Color` RGBA value (controlling the colour of the material) and the `Glossiness` value (controlling how glossy the material is). The `SubShader` block hosts the majority of the shader code. This is where you can set the render type of the material, whether it be transparent or opaque. Lastly, the `CGPROGRAM` block is purely written in the CG programming language whilst the other two blocks were written in Unity's ShaderLab language. Using the `#pragma` directive, the type of shader can be defined (i.e., surface or fragment or vertex). Within this block, different property variables are set using the various input variables with the main goal of outputting an RGBA value.

3.3.1.2 Depth Buffer

The depth buffer is a crucial feature of shaders for this project. Their purpose is to compare the depths of objects in a scene to ensure that they occlude each other properly [23]. A stencil buffer is a part of the depth buffer which is used for stencil operations. These stencil operations mainly consist of rendering part of an object whilst discarding other parts. Stencilling is the act of masking out a region to single it out for special treatment. With this

technique, two shader files can be created, the first creating a stencil mask to represent the opening of the hole and the second representing the hole walls and internals which the user can see through the stencil mask. The stencil mask is defined within the `SubShader` block of the shader file.

3.3.1.3 Hole Cutaway Shaders

The Hole Internal shader is the shader applied to the walls of the cutaway (hole) and the object to be displayed within the cutaway. The Hole Opening shader is the shader applied to the mask covering the opening of the hole. The idea behind this is to make use of a stencil mask at the opening of the hole so that the contents of the hole can only be seen when the user is looking through the opening. In Figure 3.11 below, the blue internal walls of the hole can only be seen when looking through the opening (stencil mask). This is a very important

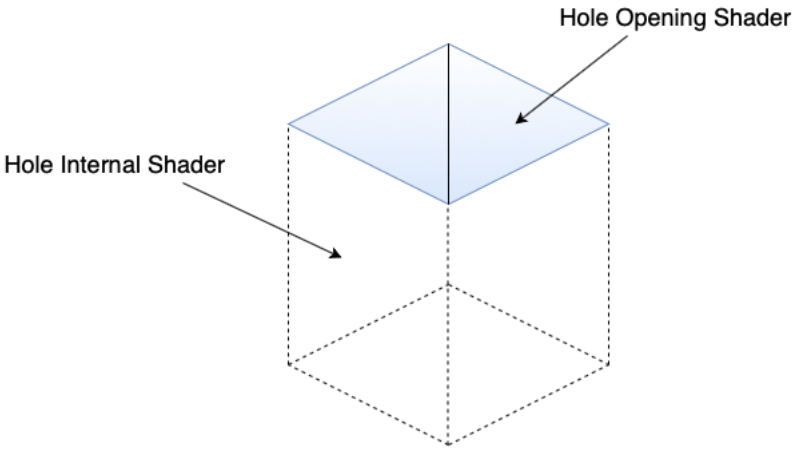

Figure 3.11: Diagram of cutaway using the Hole Internal shader and the Hole Opening shader.

requirement of the cutaway holes as it adds to the realism of it. If the user can see the internal objects without peering into the opening, it will appear as though the internals are just floating above the hole, completely ruining the illusion, thus making the cutaway redundant.

In Figure 3.12 below, the first image on the left contains the hole internals and the hole opening mask, however, neither of the shaders have been applied, so the user can still see the non-occluded internals when looking at the hole from the side. In the middle image, the Hole Opening shader has been applied to the opening mask and the Hole Internal shader has only been applied to the hole walls, not the internal object. Therefore, the internal object does not even appear as though it is inside the cutaway hole, floating above it. In the last image on the right, the Hole Internal shader has now also been applied to the internal object,

completing the illusion of the cutaway hole. The hole walls and internal object are now successfully occluded by the opening depending on the user's viewing angle.

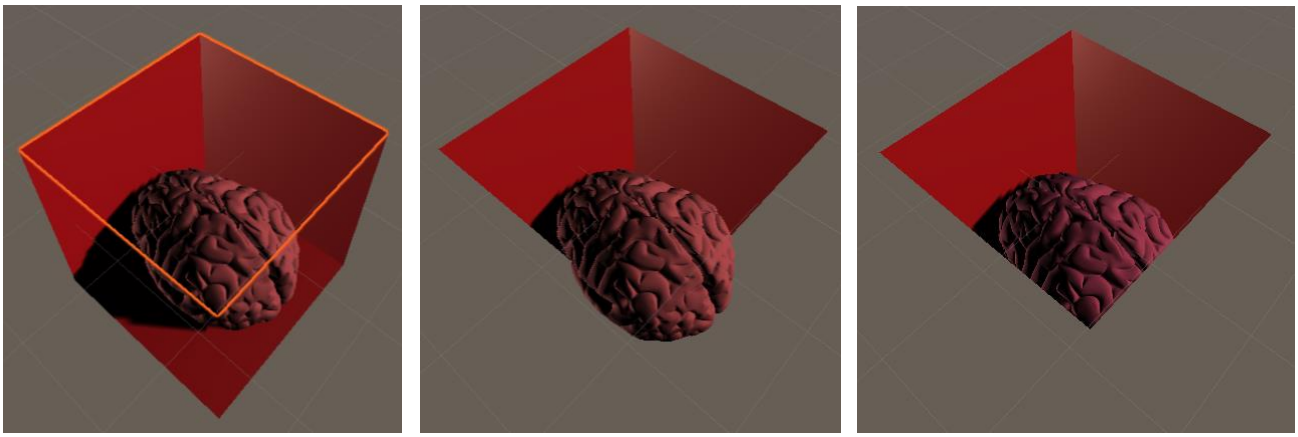

Figure 3.12: Cutaway hole before and after applying the shaders.

3.3.1.4 Shader Code

3.3.1.4.1 Hole Opening Shader

In the `Properties` block of the Hole Opening shader, the `Stencil Reference Value` is defined as an integer equal to 1. This represents the value of the stencil buffer, which is used in the rest of the shader code.

```
Properties {
    _StencilRef("Stencil Reference Value", Int) = 1
}
```

Figure 3.13: Hole Opening shader Properties block.

In the `SubShader` block, `Queue` is set to `Geometry-100`. This is so the stencil mask gets rendered first, therefore affecting normal geometry. The `ColorMask` is set to 0 as the stencil

```
SubShader {
    Tags {
        "Queue" = "Geometry-100"
    }
    ColorMask 0
    ZWrite off
    Stencil {
        Ref[_StencilRef]
        Comp always
        Pass replace
    }
```

Figure 3.14: Hole Opening shader SubShader block.

mask must be invisible to allow the user to look through it. `ZWrite` is set to “off” to prevent the shader from writing to the Z-Buffer. If `ZWrite` was set to “on”, the stencil mask would occlude the objects being displayed behind it, therefore the user would be unable to see the internals of the hole.

The predefined mask value to the stencil buffer is also defined here. The stencil is set to use the value of the `_StencilRef` property. The `Comp` value of the stencil operation defines the case in which the stencil operation passes. Since it is set to “always”, no matter what the reference value is, the object will always be drawn [23]. `Pass` being set to “replace” makes the stencil operation replace what is already in the stencil buffer, if the contents of the stencil test pass.

The `CGPROGRAM` block of the shader just returns an RGBA value so that the object is drawn.

3.3.1.4.2 Hole Internal Shader

In the `Properties` block of the Hole Internal shader, the `Stencil Reference Value` is written the same way as in the Hole Opening shader. Other properties relating to the appearance of the hole such as the colour and glossiness are also defined here. All of these properties are variables with values that can be changed within the inspector menu in Unity.

```
Properties {
    _StencilRef("Stencil Reference Value", Int) = 1
    _Color ("Color", Color) = (1,1,1,1)
    _MainTex ("Albedo (RGB)", 2D) = "white" {}
    _Glossiness ("Smoothness", Range(0,1)) = 0.5
    _Metallic ("Metallic", Range(0,1)) = 0.0
}
```

Figure 3.15: Hole Internal shader Properties block.

In the `SubShader` block, the `RenderType` is set to “Opaque”. This tag categorises the shader into the Opaque render type so that the object is displayed normally.

The stencil operation references the `Stencil Reference Value`. `Comp` is set to “equal”, meaning the object will only be rendered when the stencil buffer value is equivalent to the reference value. Both `Pass` and `Fail` are set to “keep”, meaning the current stencil buffer value will stay the same whether or not the test passes.

```
SubShader {
    Tags { "RenderType"="Opaque" }
    Stencil {
        Ref[_StencilRef]
        Comp equal
        Pass keep
        Fail keep
    }
```

Figure 3.16: Hole Internal shader SubShader block.

3.3.1.5 Materials

In Unity, materials are used to tell the program how to render the surface of an object. Materials can be assigned a shader file and then that material can be applied to an object in a scene, therefore applying the aspects of that shader to that object.

In this project, there were two different types of cutaway holes used: the “DualShock 4” (DS4) controller holes and the wall holes. Both of these holes have their own respective

colours and visuals. Since the DS4 that was used for model tracking is red, the DS4 holes had to also be red. The material “DS4 Hole” was created and it was assigned the Hole Internal shader since it will be used on the internal walls of the DS4 holes. The colour was also set to bright red, and the smoothness was set to 0.5 to match the appearance of the DS4 controller.

The “Wall Hole” material was given the Hole Internal shader, the colour was set to cream and the smoothness was set to 0.25 to blend in with the colour of the real-world sample walls used for tracking.

A material was also created for the internal components to be displayed within the holes. The internal parts to be displayed within the DS4 controller are grey and metallic in appearance so the “Parts” material was created with the colour set to grey and the metallic property set to 0.5. This material was also recycled and reused on the pipe objects displayed within the wall holes. To allow for the Hole Opening shader to be applied to the hole opening masks, the “Mask” material was created with the shader assigned to it.

Lastly, the “Transparent” material was created so the transparent visualisation technique could be used to display the internal parts of the DS4 controller. This material was assigned the Standard shader and was given an alpha value of 0.5 to make the material “see-through”.

3.3.2 Lighting

Lighting in Unity can be seen as either “real-time” or “precomputed” in some way, shape, or form [26]. The main real-time light sources that Unity offers are directional, spot, and point lighting. These light sources contribute light to the scene and update every frame with the movement of objects around the scene.

The only light source that is used in this project's Unity application is the directional light source. Since this application is an AR application, the main goal of the lighting is to look as realistic as possible. Directional lights are widely used to create the effect of sunlight in a scene. The light rays that are emitted from this light source run parallel to one another and never diverge. The position of a directional light is irrelevant as the rays travel infinitely along their path through the scene so only the rotation of the light source affects the direction of the light's rays [14].

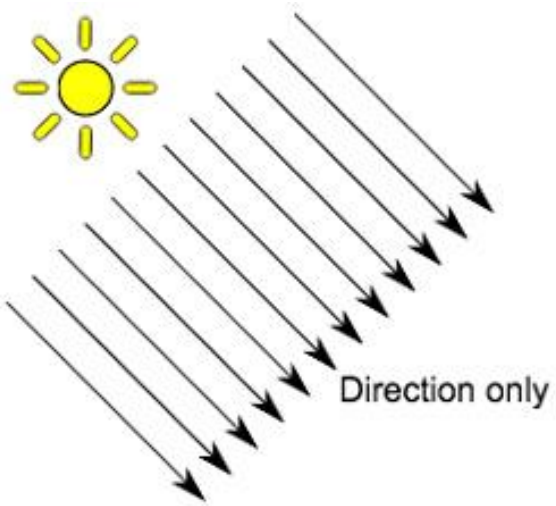

Figure 3.17: Directional light source. Source: [26]

To make the holes and internal objects in the scene appear more realistic, all objects were set to cast shadows. This was done by simply selecting "Cast Shadows" in the object's Inspector menu. The inclusion of shadows in the cutaway holes helped with the understanding of the depth and the positioning of the internal components.

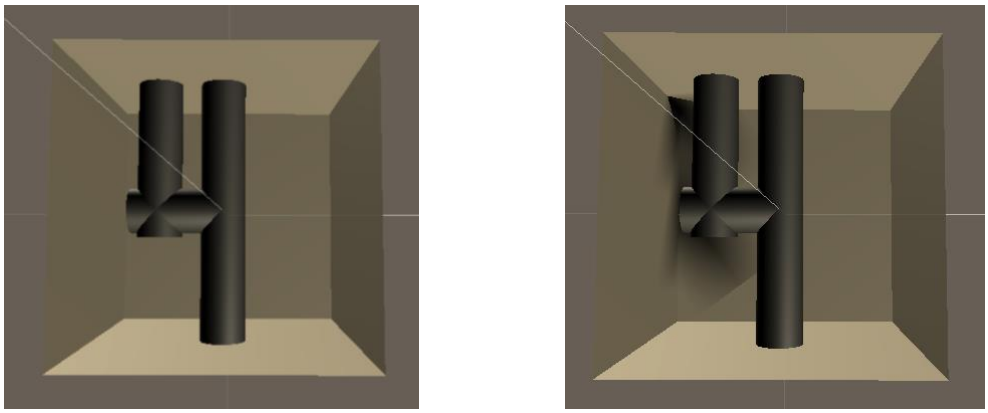

Figure 3.18: Before and after enabling shadows on cutaway holes.

3.4 Vision and Tracking

3.4.1 Vuforia AR

Vuforia Engine Library is an AR toolkit that is free to install and can be used with Unity by importing the relevant plugins. The library offers the ability to track 2D Image Targets or 3D Model Targets. Both of these capabilities were explored throughout this project.

3.4.2 AR Tag Tracking

Vuforia has the ability to track Image Targets at application run time. Image Targets represent images that the AR engine can detect and track. The engine does this by comparing the “extracted natural features” from the image seen through the camera lens against the image data stored in a database [16]. Once Vuforia detects the Image Target, it will track the image and augment the content of the Unity scene with the real-world as seen through the camera.

3.4.2.1 Creating an Image Target

The Vuforia Developer Portal has a Target Manager interface where an image database can be created. Once the database was successfully created for this project, images could be uploaded to the database to train the Vuforia AI to recognise them as Image Targets. After uploading the image file, Vuforia analyses the image as an Image Target and rates it on how suitable it is to detect and track. Various attributes of the image are evaluated to decide this rating. To enable the best detection and tracking performance from the Vuforia Engine, Image Targets should [15]:

- Be rich in detail.
- Have good contrast; images with bright and dark regions and well-lit areas work well.
- Not have repetitive patterns, employing unique features and distinct graphics covering as much of the target as possible to avoid symmetry, repeated patterns, and feature-less areas.
- Be 8-bit or 24-bit PNG and JPG formats; less than 2MB in size.

Images containing as many sharp details as possible are best for tracking and detection. For example, an image containing just one square is better than an image containing just one circle. The square has four features, being each of its corners, however, the circle has no features as it contains no sharp features. In Figure 3.19, two Image Targets are being compared. The image on the left has been awarded a low rating due to its lack of feature points as it has more rounded shapes than sharp shapes. The image on the right has been

Figure 3.19: Vuforia Engine rating Image Targets, showing the different feature points. Source: [15]

awarded a high rating as it has an abundance of feature points due to its sharp details.

Figure 3.20: Vuforia Engine rating Image Targets, showing the importance of contrast. Source: [15]

Image Targets are also rated by contrast. The more contrast of colours there is in an image, the more feature points will stand out allowing for better tracking. In Figure 3.20, the same image is being used except one has a coloured background and the other has a fully white background. The image with the coloured background has been granted a low rating by Vuforia Engine due to there being low contrast in colours. The opposite can be said for the image on the right, having a white background with high contrast in colours, therefore being given a high rating.

3.4.2.2 Image Targets Used

The first set of Image Targets that were used for tracking in this project were generated from ArUco [5]. The images were downloaded, printed, and cut out as seen in Figure 3.21. These images did not follow the recommended guidelines as outlined in section 3.4.2.1 and therefore received an extremely low rating of 0-stars from the Vuforia Engine. They do not contain many sharp details as there are so few corners on the polygons, therefore they performed badly in terms of detection and tracking.

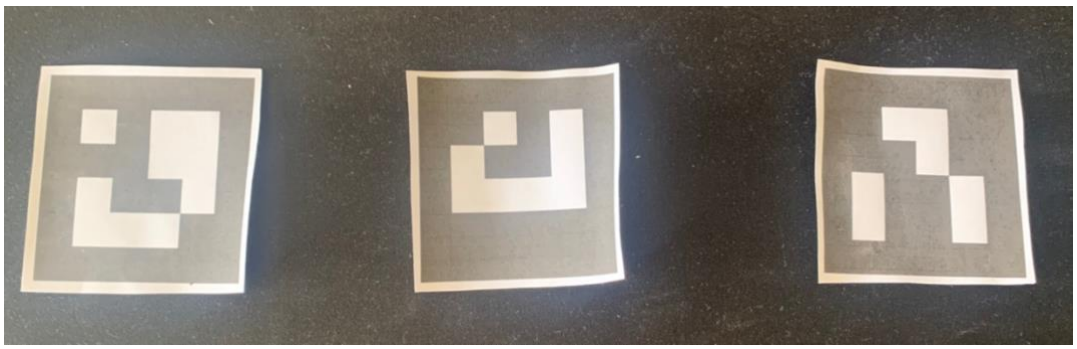

Figure 3.21: First set of Image Targets created – unsuccessful at being detected and tracked.

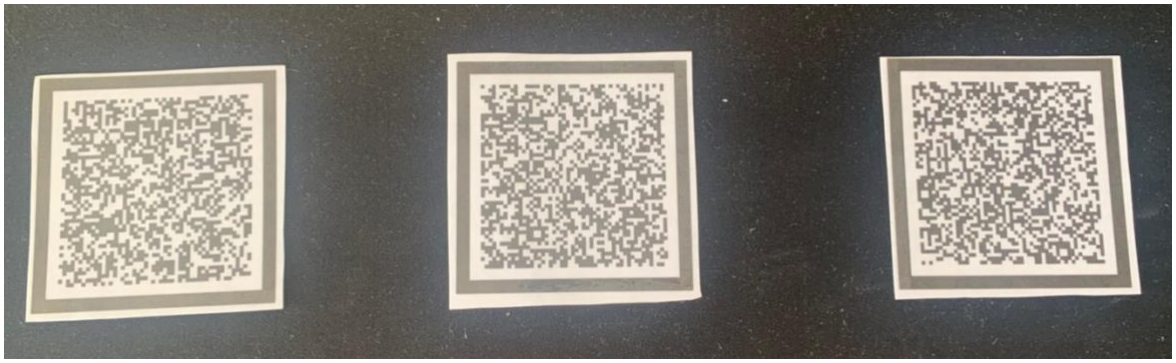

Figure 3.22: Final set of Image Targets - successful at being detected and tracked.

Due to the failure of the first set of Image Targets, a new batch that was more suitable had to be generated. They were created using ARMaker [24]. These Targets resemble QR codes and consist of many sharp details and no repeated patterns as seen in Figure 3.22. Therefore, all of these images scored a 5-star rating from Vuforia and worked perfectly for detection and tracking. After printing the images, they were also laminated so they would remain as flat as possible. As you can see in Figure 3.21, the first set of images were not laminated and therefore the edges began curving inwards, making it even more difficult for the Vuforia Engine to perform a consistent track.

In Figure 3.23 below, the results of the Vuforia Engine's rating process can be seen, showing the recognised feature points of each image.

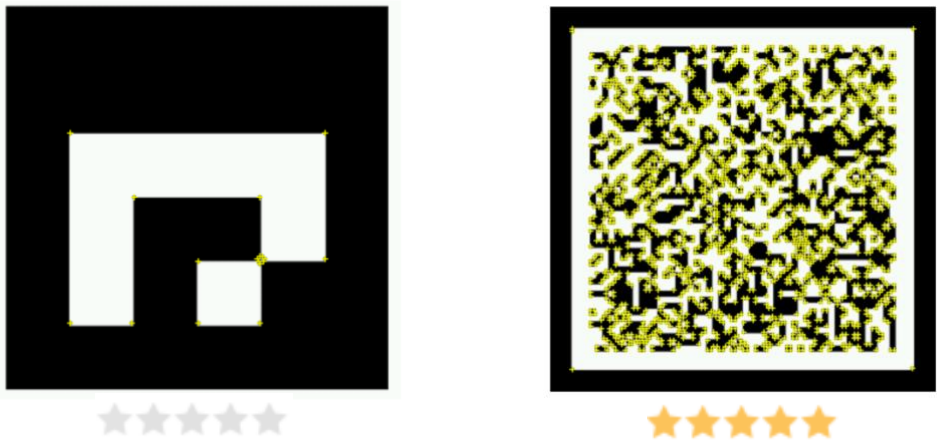

Figure 3.23: Images from both sets and their respective feature points.

3.4.2.3 Image Targets in Unity

The Vuforia Library integrates well with Unity, allowing for an easy process of including Image Targets in a scene. An Image Target can be added to a scene by simply selecting it under the "Create Object" menu in Unity. Once it has been added to the scene, the Target's properties can be edited from within the inspector menu as seen in Figure 3.25. From here, the image can be selected from the database that was created on the Vuforia Developer Portal. After selecting the image, the Image Target object in the scene is updated to have the appearance of the image as seen in Figure 3.24.

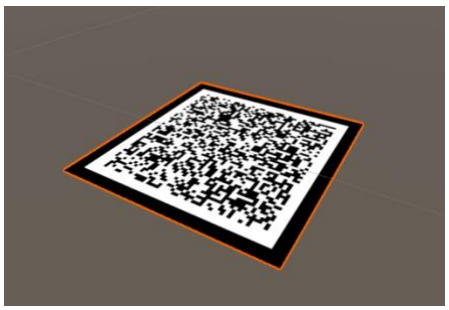

Figure 3.24: Image Target object.

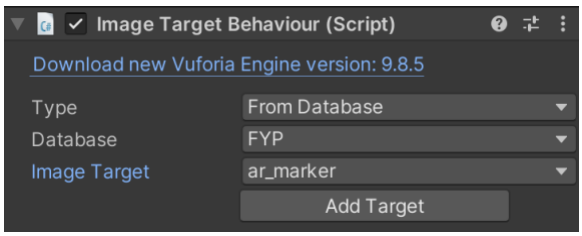

Figure 3.25: Selecting Image Target from database using the Inspector interface.

Adding a display object to the Image Target is done by moving the object into the hierarchy of the Image Target object and then repositioning the object on top of the Image Target in the scene. For the cutaway hole objects, they have to be placed so that the Image Target is located at the opening of the hole. Mistakenly placing the entire hole on top of the Image Target causes the hole to appear as though it is floating in mid-air above the Image Target in

AR as seen in Figure 3.26. Placing the hole object so that the Image Target is flush with the hole opening allows for the hole opening to be flush with the real-world surface in AR, appearing more realistic.

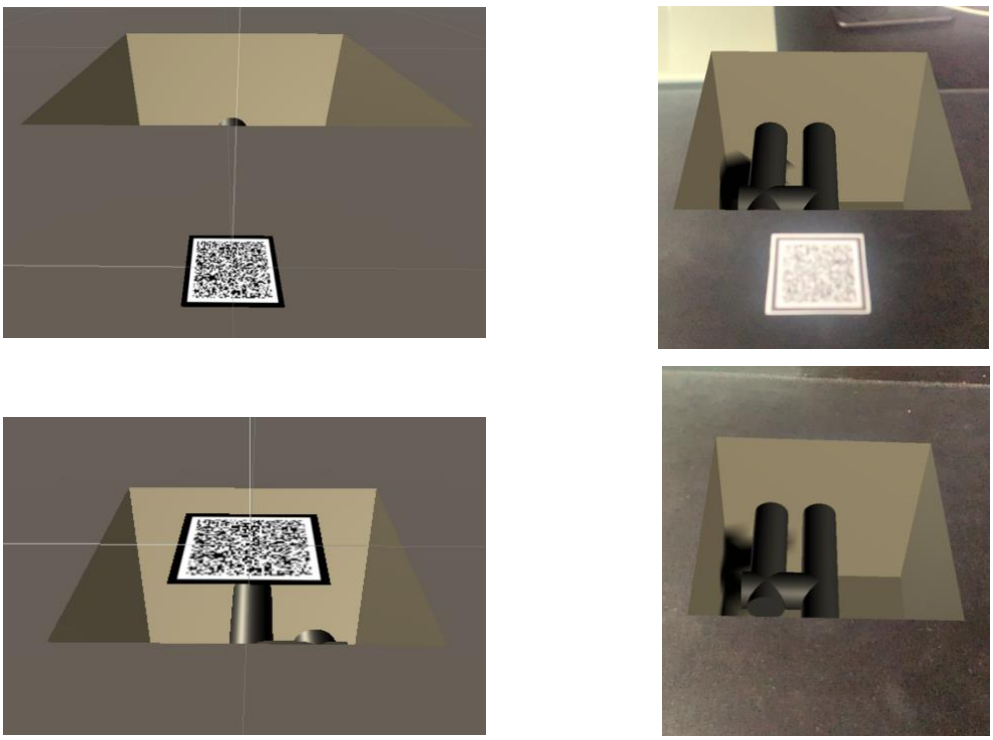

Figure 3.26: Placing the cutaway hole on the Image Target in Unity.

3.4.3 Model Tracking

Vuforia Engine not only has the ability to track Image Targets, but it can also detect and track 3D Model Targets. This enables the app to recognise and track particular objects in the real world based on the shape of the object [17]. This was used in this project to track a real-world “DualShock 4” game controller so that the application could place cutaway holes around it.

3.4.3.1 Creating a Model Target

3.4.3.1.1 Prerequisites

In order to create a Model Target for a real-world object, you must also have access to the 3D model data for that object, such as a 3D CAD model or a 3D scan of the object. This is for the Vuforia Engine to compare the real-world object against it so it can recognise and track the object successfully.

There are various attributes the Model Target should have to provide the best possible performance from Vuforia Engine. To allow for the best tracking results, the Model Target should ideally remain static after having been detected by the application. The user can move around the object but should not move the object itself [18].

The colour and pattern on the Model Target also affect tracking performance. Typically, objects that have multiple colours or patterned surfaces work better. Although objects with a single colour can be reliably detected by the Vuforia Engine, they are difficult to track as the camera pans around it as some variation in the surface appearance is required. This means that attempting to track objects that are of similar colour to the background is almost impossible. This also applies to transparent or highly reflective objects. That being said, the colour of the physical model does not need to match the colour of the CAD model used to generate the Model Target. One Model Target can be used to track the same object with different colour forms and combinations [18].

Similar to Image Targets, Model Targets should also provide sufficient geometric detail to differentiate themselves from other objects in the environment. It is recommended to ensure the Model Target has enough sharp edges and extrusions to help with recognition. One should also avoid symmetrical objects as Model Targets as it can be challenging for the Vuforia Engine to distinguish the sides of the object [18].

Model Target tracking favours objects that are non-flexible and mainly rigid. The Vuforia Engine assumes that the object to be tracked has no moving parts since the comparable 3D model is always static. Therefore, as a best practice, parts that are freely movable or not always present on the physical object should be removed from the CAD model.

The physical Target Model object should have identical dimensions as the 3D CAD model used for target generation. The Model Target should be set to the correct size of the physical object in metres as accurately as possible [18].

Due to common problems that have been encountered with CAD models, the Vuforia Library website recommends that the CAD model used for Model Target generation should:

- Have a maximum of 400,000 polygons or triangles.
- Contain a maximum of 20 parts.
- Contain a maximum of 5 textures.

3.4.3.1.2 Model Target Generator

In order to train the Vuforia Engine with a 3D Model Target for it to recognise, the Model Target Generator application must be used. This is free downloadable software provided by Vuforia. The Model Target Generator takes as an input, a 3D CAD Model representing the model to be tracked. This application checks the model for suitability and allows the user to configure it for the desired tracking experience.

After uploading the CAD model to the Model Target Generator application, various steps are taken before it is ready for tracking. First, the “Model-up vector” must be set properly. The “Up vector” which aligns the plane with the model should be selected. As seen in Figure 3.27 below, “Up vector: Y” was chosen when generating the Model Target for the “DualShock 4” controller as this plane is aligned correctly with the model.

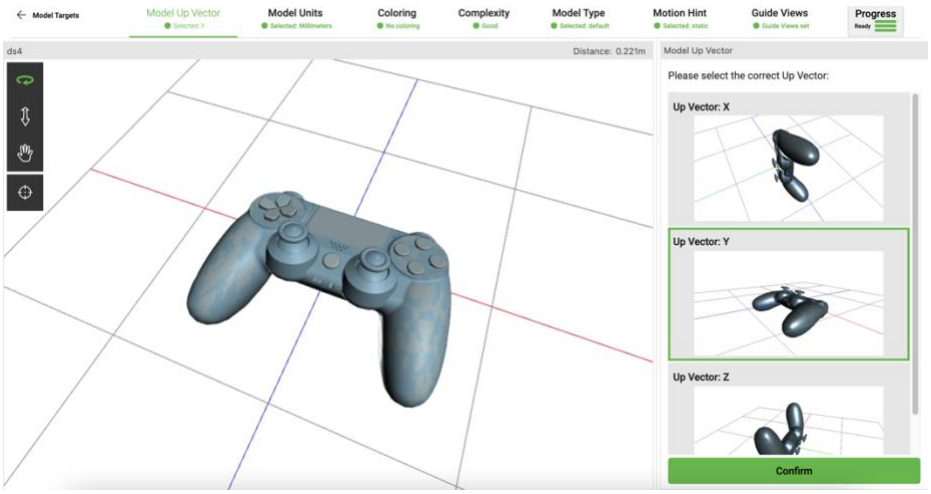

Figure 3.27: Model Target Generator - setting the model-up vector.

The next step is to set the “Model Units”. The Model Target should be the same size as the object in the real world. The correct units of the dimensions of the model should

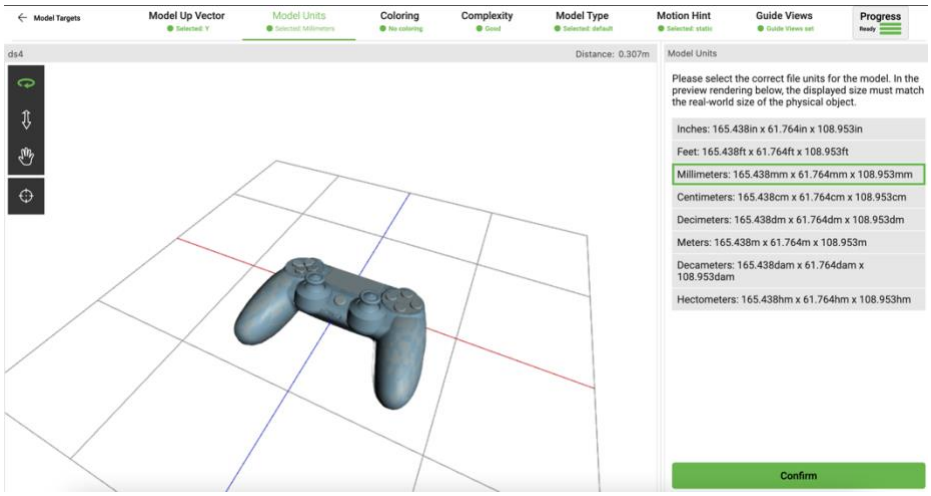

Figure 3.28: Model Target Generator - setting the model units.

be selected. In Figure 3.28 below, millimetres were selected for the DS4 model.
The model should also be randomly coloured to improve tracking capabilities by distinguishing different parts and edges. In Figure 3.29, the DS4 model was randomly assigned a purple colour, with the buttons given a dark black colour to improve recognition and tracking.

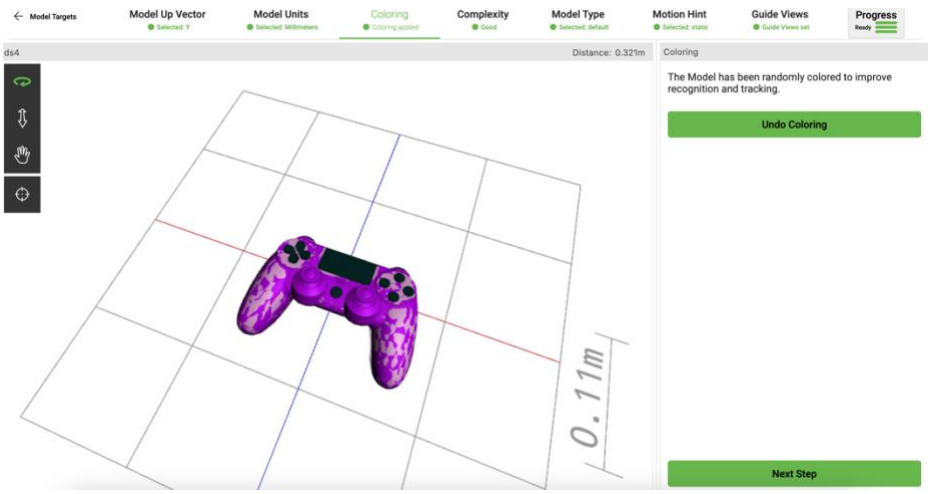

Figure 3.29: Model Target Generator - colouring the model.

In the next step, Vuforia analyses the “Complexity” of the Model Target, checking how many parts and triangles it is made up of. The DS4 model was given a “Good” complexity, meaning it is suitable to be used as a Model Target. The “Model Type” must also be selected, being either a 3D scan, a car or the default option which represents any other object type. Therefore, the DS4 model was given the “Default” model type.

It also must be stated whether the object to be tracked will be “Static” or “Adaptive”. Static implies that the object will be stationary during the AR experience. Since this “Motion Hint” is less demanding of processing power, it also saves power during tracking. Adaptive objects may move during the AR experience. Static was chosen for the DS4 model as the object itself will not be moving during tracking.

The final step is the “Guide View” creation. Guide views are essential to initiate the tracking of a Model Target. They are presented as a rendered outline of the Model Target. During the AR session, the tracking will initiate when this rendered outline is lined up correctly with the

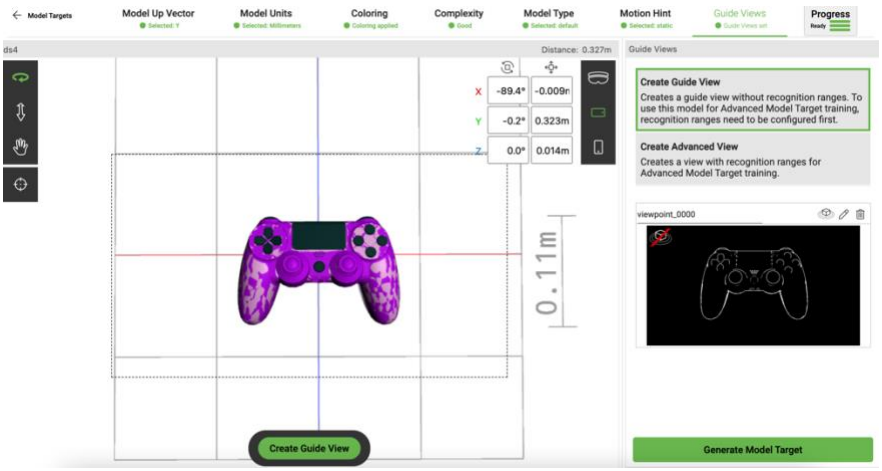

Figure 3.30: Model Target Generator - guide view creation.

object [17]. When setting up the guide view, the orientation of the device camera can be selected, with it being set as landscape for the DS4 guide view. For this Model Target, the guide view was taken from above the controller, as all of the distinguished features of the model are clearly visible as seen in Figure 3.30. Now the Model target was ready to be generated and used for tracking.

3.4.3.2 Model Targets in Unity

3.4.3.2.1 Adding Model Target to Scene

Similar to the Image Target implementation in Unity, the Model Target can be added to a scene by selecting it under the “Create Object” menu. After doing so, via the inspector menu, the database of the Model Target can be selected and various options can be changed such as the dimensions, motion hint, mode of tracking and guide view.

Figure 3.32: DualShock 4 Model Target in Unity.

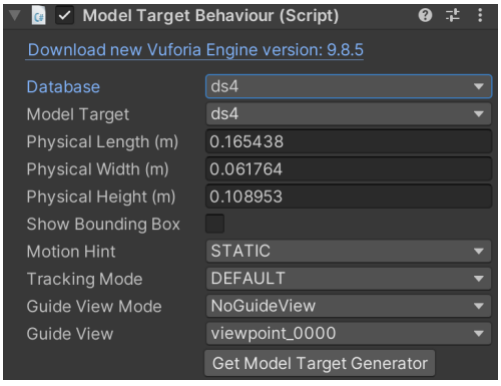

Figure 3.31: Model Target inspector menu in Unity.

3.4.3.2.2 Displaying Objects on Model Target

To display objects around the Model Target when it has been recognised in AR, the objects must be added to the hierarchy of the Model Target in Unity and then placed in their respective positions around the Model Target in the scene. For the DS4 model, the holes and internal part objects were imported from Blender (as outlined in section 3.2.2) to be displayed with the Model Target. The appropriate shaders were applied to the holes and

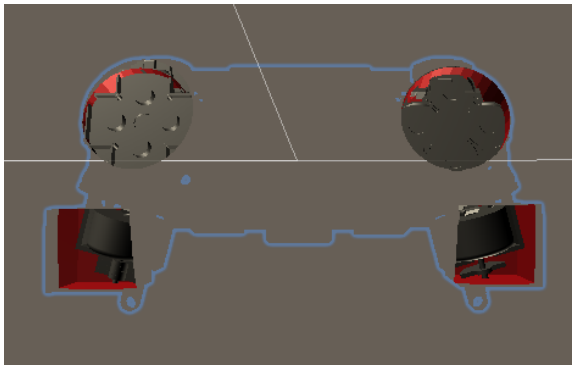

Figure 3.33: Holes and internal parts before placement on Model Target in Unity.

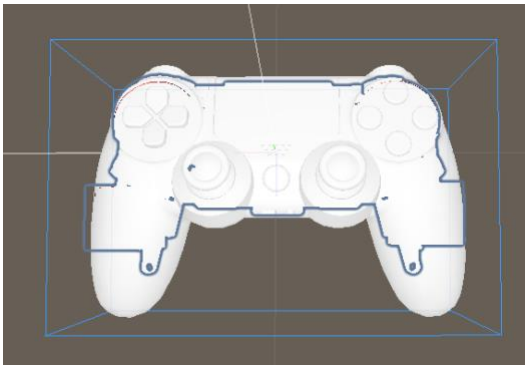

Figure 3.34: Aligning the holes and internal parts with the Model Target in Unity.

internal parts as described in section 3.3.1. The display model also included the shell of the controller to help with aligning the objects with the Model Target in Unity. Once everything was correctly positioned, the shell of the controller was removed from the scene, leaving the holes and internal parts placed in the perfect position within the Model Target.

After positioning the objects to be displayed, the Unity application successfully detected the DS4 object and placed the holes and internal parts at their correct positions as seen in Figure 3.35 below.

Figure 3.35: Holes and internal objects being successfully placed on the real-world DualShock 4 object.

3.5 Extra Features

Some additional features were added to the Unity AR application to aid the demonstration of the project.

3.5.1 Mode Selection

Unity applications work by having different scenes containing different objects. In this project's application, there are two scenes:

1. The Image Target tracking scene.
2. The Model Target tracking scene.

The Image Target scene consists of the different Image Targets and their respective cutaway holes/internal parts to be displayed as outlined in section 3.4.2.3. When this scene is running as the application, the Vuforia AR camera opens and begins searching for only the Image Targets. The Model Target scene contains the DS4 Model Target and the respective holes/internal parts to be displayed. When this scene is running as the application, the AR camera only looks for the Model Target.

The reason why these two scenes could not just be all within one scene is due to performance. Having the Vuforia Engine searching for both Image Targets and a Model Target at the same time is strenuous on the engine's capabilities and renders the application almost unusable. Separating both of these scenes into two different modes which can be selected by the user through UI buttons, allows for the Vuforia Engine to focus on either Image Targets or Model Targets but not at the same time, thus improving performance.

Two UI buttons were added to the scene under the hierarchy of the AR Camera object. This means that whenever the AR Camera (the device camera) is moved, the UI buttons will move with it, so they remain at the same position on the screen in the top left corner. These

Figure 3.36: The mode selection UI buttons as seen in Unity scene.

buttons had to be added to both the Image tracking scene and the Model tracking scene. If they were only added to one scene, the buttons would disappear when the user switches to the other scene, making it impossible to return to the first scene.

Each of the buttons corresponds to a function within the `sceneSwitcher.cs` script. This is a simple script containing a single function that tells the application to load a specified scene. The `SwitchScene()` function takes the name of the scene as its only argument and then

```
public class sceneSwitcher : MonoBehaviour
{
    public void SwitchScene(string sceneName)
    {
        Application.LoadLevel(sceneName);
    }
}
```

Figure 3.37: The sceneSwitcher.cs script.

loads the scene. Within the inspector menu of the buttons, the function is selected to run when the button is clicked. The scene name is also specified in this menu as seen in Figure 3.38 below.

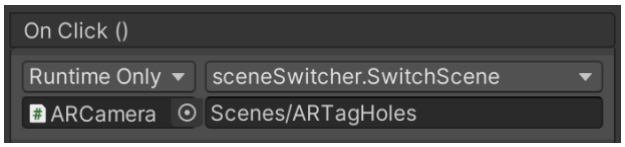

Figure 3.38: Button On Click() function selection in the Inspector menu.

3.5.2 Visualisation Selection

This application contains two different types of visualisation techniques for the Model Target scene: the cutaway technique and the transparency technique. Both of these methods are showcased within the app to highlight the differences between both of these approaches. This is why it was decided to include a feature where the user can easily switch between both of the visualisation techniques with the press of a button.

Two UI buttons were created, one for activating the cutaway technique and the other for the transparency technique. The button objects were created within the Model Target hierarchy so that they only display when the model has been recognised. Within the scene, the buttons were positioned to the right side of the DS4 Model Target, so they clearly get displayed near the real-world controller object.

Figure 3.39: The visualisation selection buttons placed beside the DS4 Model Target.

In order to switch between the visualisation techniques, the internal parts of the controller had to be able to change material with the press of a button. Thus, the `changeMaterial.cs` script was written. This script contains three functions:

1. The `Start()` function: This function runs every time the script is run. The necessary objects are defined here such as the internal parts of the DS4 controller and the cutaway holes.

2. The `MakeTransparent(Material)` function: This function takes the "Transparent" material as its only argument and applies it to the internal parts of the DS4 controller. This material uses the "Standard" shader with an alpha value of 0.5 to make it transparent. This function has another role which is the deactivating of each of the cutaway holes, preventing the application from rendering them.

3. The `MakeCutaway(Material)` function: This function takes the "Parts" material as its only argument and applies it to the internal parts of the DS4 controller. This material uses the "HoleInternal" shader which is defined in section 3.3.1.3.

Within the Inspector menu of the buttons, this script is run when the buttons are pressed. The `MakeTransparent()` function is assigned to the "Transparent" button and the `MakeCutaway()` button is assigned to the "Cutaway" button.

3.5.3 Interactive Cutaway Holes

To establish a more immersive AR model-tracking experience, the ability to make cutaway holes appear and reappear was added. The user of the application can make the holes appear/disappear by pressing on the screen at the location of the cutaway hole. This was done by utilising invisible UI buttons. Four invisible buttons were created within the Model

Target hierarchy and they were positioned over each of the four cutaway holes. The buttons were scaled to ensure that they cover the entire hole so the “press” will register as long as the user presses anywhere around the hole location.

In order to make the holes appear and disappear, the `toggleInvisible.cs` script was created. This script contains the function `HideObject()` which takes the hole object as its only argument. If the hole is already active, the hole becomes deactivated. If the hole is already inactive, the hole is activated. Within the inspector menu, each of the four buttons are assigned the `toggleInvisible.cs` script so it runs when the buttons are pressed. Each of the button’s respective holes is also passed as their arguments.

Figure 3.41: The cutaway visualisation of the DS4 controller.

Figure 3.40: The transparent visualisation of the DS4 controller.

3.6 Deploying the Application

Building an application and deploying it to iOS is extremely easy with Unity. The only requirements are having a computer running Mac OS with Xcode installed and a device running iOS. Once both of these are fulfilled, Unity builds and deploys the application automatically when “Build and Run” is selected within the “Build Settings” menu. After doing so, Xcode opens up and installs the application on the iOS device plugged in via USB. In this project, the final application was built and installed on an iPhone XR device.

3.7 Explored Applications

The initial plan for this project was for it to have a medical concept application rather than an engineering concept. The main goal was to display different cutaway holes on a real-world human body dummy, highlighting the different internal organs and body parts. The thought process behind this was that it would aid students studying biology-related subjects to learn the human anatomy by having an immersive visual experience with DR and cutaway holes. The idea was to make use of the Image Targets and place them around the dummy body at the positions of the internal parts to be displayed, and then display the holes with the corresponding Image Targets. An example of cutaway holes displaying human organs can be seen below in Figure 3.42.

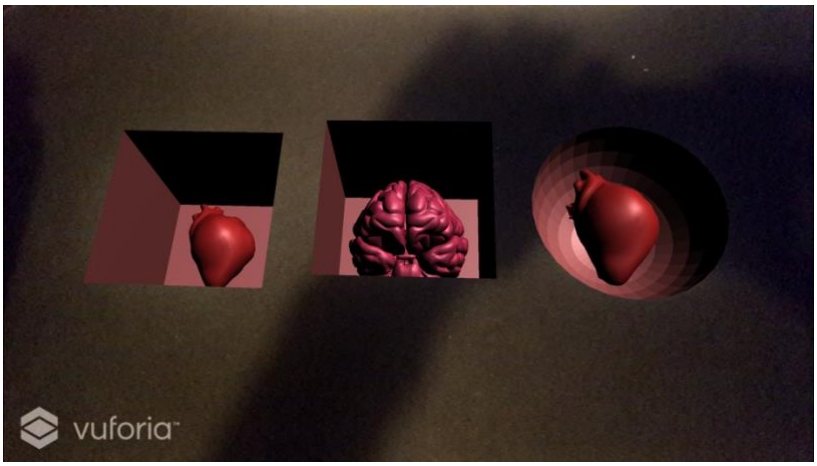

Figure 3.42: Initial plan for the application - visualising human organs.

However, due to the unforeseen circumstances of the Covid-19 pandemic, travel restrictions were put in place preventing the retrieval of the human body dummy. This transformed the entire concept of the project to take on an engineering application.
The engineering concept was chosen based on three main reasons.

1. The cutaway visualisation technique is widely used in the engineering industry to visualise internal parts of machinery.

2. Model Target tracking could be used to track the object as 3D models for popular devices (i.e., DualShock 4 controller) can be easily found online for free.

3. Given the travel restrictions, it was favourable to take on a concept that could be done with the resources at home.

4 Results

4.1 Internal Visualisation Results

4.1.1 Cutaway Results

The cutaway technique of internal visualisation was successfully implemented during the course of this project. Using the ideology of DR, the cutaway holes were created to appear as though they were being virtually dug out of the real-world sample objects and surfaces.

4.1.1.1 Image Target Cutaway Holes

Selecting the “AR Tags” Mode, the Vuforia Engine begins searching for the specified Image Targets. Bringing the camera close enough to the Image Target, the target is recognised, and the respective cutaway hole is displayed on top of it. As seen in Figure 4.1, the model being displayed is a cubic hole in the wall, revealing cylindrical pipes, representing the plumbing beneath the wall. This showcases the usefulness of this method of visualisation in

Figure 4.1: Vuforia Engine detects the Image Target and displays the cutaway hole.

the construction industry for showing builders where subterranean structures are located, helping them understand their location in an immersive and accurate experience.

As shown in Figure 4.2, the user pans the device camera around the cutaway hole, demonstrating how the viewing angle of the hole changes with the camera position. Once

Figure 4.2: Different viewing angles of a cutaway hole displayed using Image Targets.

the application detects the Image Target, the displayed hole remains in that location, providing an accurate tracking experience.

Using another Image Target, the application also successfully displayed different-shaped cutaway holes. In Figure 4.3 below, a spherical hole can be seen from different angles. Although both of these examples are different-shaped holes, the hole opening remains flat as the surface is flat in both cases.

Figure 4.3: Spherical cutaway hole being displayed using Image Targets.

Applying this logic to a curved surface means that the hole opening has to also be curved. Figure 4.4 depicts a dog with an Image Target on its body, displaying a cubic hole with a curved opening, revealing the heart. The curve of the opening is faithful (however not exact)

to the curve of the surface, making the hole appear more believable and realistic, thus improving the user's understanding of the internals being visualised.

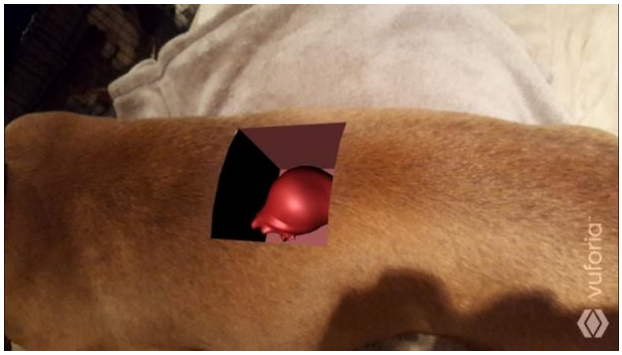

Figure 4.4: Cutaway hole with curved opening being displayed on a curved surface.

4.1.1.2 Model Target Cutaway Holes

Selecting the "DS4" mode initiates the model tracking feature of the application and the Vuforia Engine begins searching for the "DualShock 4" object. Once the object is discovered, the four cutaway holes are displayed on the controller, highlighting different features of the

Figure 4.5: DS4 controller with cutaway holes displayed on it in AR.

internal components, which can be seen in Figure 4.5. After the holes have been displayed, they remain in the same position, meaning the user can pan around the controller with the

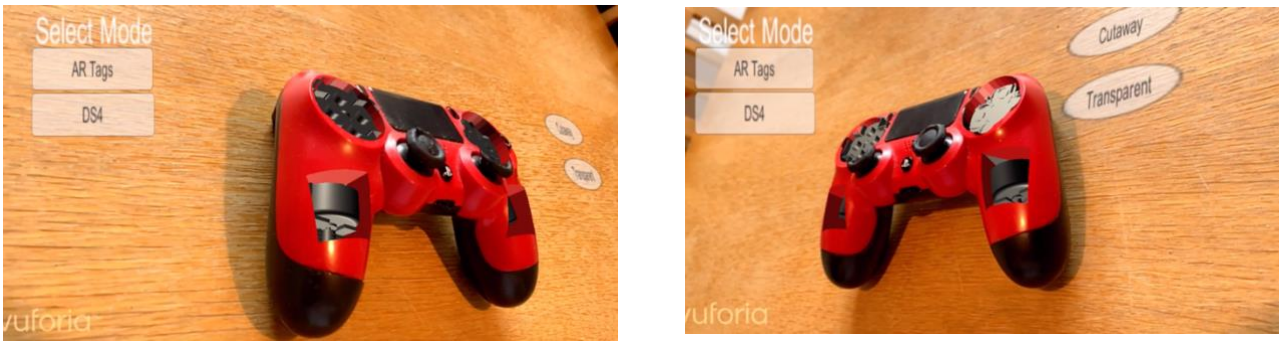

Figure 4.6: DS4 cutaway holes from different angles.

camera, viewing the cutaways from different angles. Two of the holes are cylindrical,

revealing the D-Pad and button internal mechanisms of the controller. The other two holes are cubic with curved openings, revealing the vibration motors of the controller. The curves of the openings are identical to the curves of the controller at that position since Boolean modifiers were used in Blender to create these holes (see section 3.2.2.1.1).

Figure 4.7: User presses on DS4 cutaway holes to make them appear and disappear.

The holes are also interactable in this application, meaning the user can press the hole location to make it either appear or disappear. This feature showcases how immersive the model tracking mode of cutaway visualisation can be. In Figure 4.7 above, the user zooms in on the lower left handle of the controller and presses the location, only revealing that cutaway hole. This ability allows the user to view the finer details of the internals by focusing in on a specific part.

4.1.1.3 Cutaway Holes – Problems Encountered

4.1.1.3.1 Tracking

Tracking with a camera introduces various issues introduced by environmental factors, whether it be image tracking or model tracking. The main complication that arose was related to environmental lighting. Lighting has a huge influence on the tracking performance. With both image and model tracking, if the area is too dark, the object cannot be detected and if the area is too bright, reflections off the object affect the tracking. The glossiness of the image or model also plays a part in this problem, reflecting more light the glossier the surface is.

An Image Target related issue is the flatness of the target. To carry out the best possible tracking, the image target should be completely flat. The initial images for this project were printed out on A4 white paper and cut out. Due to the lack of durability of the paper, the targets began to fold and curve at the corners. This made it almost impossible for the Vuforia Engine to detect the targets. To combat this issue, the targets were laminated after being printed, to help them remain as flat as possible. However, this introduced a lighting issue, since the surfaces were now more reflective. Similar to the “DualShock 4” controller’s glossy

finish, both of these targets were now causing light rays to reflect off the surfaces. As a result, the cutaway holes being displayed on the targets struggled to remain still and were constantly jumping around the scene. This obstacle was eventually overcome by providing the correct level of lighting. Natural lighting such as sunlight appeared to work better than artificial lighting as there were fewer reflections. Therefore, most of the demonstrations done in this project took place next to a window, with the artificial room lighting turned off.

Another tracking problem that happened only involved the model tracking portion of the project. There were two samples of “DualShock 4” controllers that were used to test the tracking abilities, one fully black and the other red/black. As mentioned in section 3.4.3.1.2, the Vuforia Engine struggles to track models that are entirely of one colour. After learning this information, the theory was tested on the fully black controller, as seen in Figure 4.8 below. This controller had no colours differentiating the different features of the object, therefore, Vuforia failed at reliably tracking the model. This problem was solved by simply using the red controller instead, as it had a variation of colours outlining its defining details.

Figure 4.8: Black variant of the DS4 controller.

4.1.1.3.2 Curved Surfaces

Curved surfaces posed an issue in both image and model tracking. In order to display a believable cutaway hole, the opening of the hole must match the curve of the surface. Displaying a curved hole on a curved surface with an Image Target involves estimating the curvature of the surface at the time of the creation of the hole model. Since there is no automation in the creation of the hole as it is all created manually in Blender, the curve will never be 100% accurate with image tracking. Since this also involves the tracking of a flat image target, it is difficult to stick the image target to a curved surface. The image will almost always tilt one way more than the other, resulting in an inaccurate depiction of the curved hole. This can be seen in Figure 4.9 below, where a curved hole is being displayed on a

cylindrical water bottle. The image target tilted slightly to the right resulting in the hole tilting respectively, causing an inaccurate placement of the curved hole.

Figure 4.9: Image Target placed on curved surface with a tilt resulting in an inaccurate cutaway hole.

Since model tracking concerns the tracking of a 3D model, this model could be used in Blender to create accurate curved holes. This resulted in extremely accurate cutaway holes. However, this was quite a tedious process to do manually and would be inefficient to replicate on an industrial level.

4.1.1.4 Cutaway Holes – Final Thoughts

4.1.1.4.1 Without Cutaway Holes

The cutaway technique of internal visualisation was very effective in AR. If the internal components were to be displayed in AR without the cutaway holes, it would look as though the part is just floating above the real-world object. In Figure 4.10 below, an Image Target is

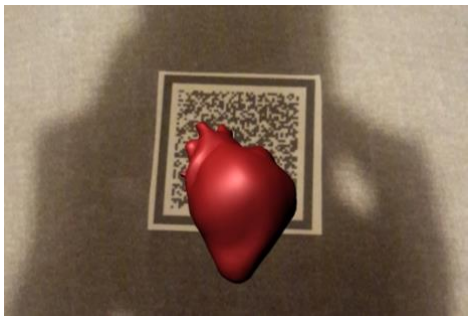

Figure 4.10: Internal component being displayed on an Image Target without the cutaway hole.

being used to display an internal part, but the cutaway hole is absent, so the component looks like it is sitting above the target.

In Figure 4.11, the “DualShock 4” Model Target is being utilised to visualise the internal mechanisms of the controller, however, there are no cutaway holes, so the parts do not

Figure 4.11: Internal components being displayed on the DS4 Model Target without the cutaway or transparency techniques.

appear as though they are actually inside the controller. This shows that creating cutaway holes through DR is imperative to demonstrate the illusion of an opaque internal component inside a real-world object.

4.1.1.4.2 Industrial Use

The cutaway technique being utilised with model tracking is an extremely accurate method of internal visualisation, since the portrayal of the components can follow a precalculated CAD model. This approach to showing the inner components of machinery and devices can be very useful in the engineering and construction industries. This could help users understand inner mechanisms and allow them to see sub-terranean objects in an x-ray vision-like manner.

This methodology can also be applied to a classroom scenario, such as a biology class, allowing students to better understand the anatomy of a human body by having a hands-on immersive experience through AR.

4.1.2 Transparency Results

The transparency technique of visualisation was implemented in the model tracking mode of this application purely to compare and contrast it with the cutaway method. Rendering the internal components slightly transparent was successful in creating an illusion of depth and making them look as though they are inside of the real-world object. Panning the camera around the Model Target allows the user to view the internals from different angles, as seen in Figure 4.13 below.

Figure 4.12: Internal components being displayed within the DS4 controller using the transparency technique.

The transparency technique of visualisation is extremely effective when the user wishes to view all of the internal parts at once, being able to see how the parts occlude one another and how they intertwine. The main downside is how the internal models are less detailed, since they employ a transparent material. It is much easier to make out the finer details when viewing the opaque internal parts through the cutaway technique, since they have their own texture and material.

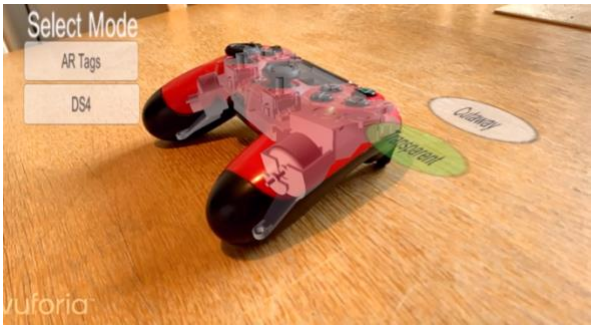

Figure 4.13: The transparency technique of visualisation being viewed from different angles on the DS4 controller.

4.1.2.1 Transparency – Problems Encountered

Since the transparency technique of internal visualisation is straightforward to implement, there were not many problems encountered. A minor issue occurred with the alpha value of the internal components. Setting the value too low made the parts barely visible and setting

the value too high occluded the “DualShock 4” controller. Through trial and error, the alpha value was finalised to give the best interpretation of an internal component.

4.2 Future Work

Although this project successfully fulfilled the main requirement of exploring different internal visualisation techniques with the use of DR, there are further advancements that could be worked on, given more time.

4.2.1 Cutaway Hole Automation

The cutaway holes for this project were created manually in Blender. This includes the creation of the hole opening and hole walls. The main issue with hole formation arises with the hole opening. The opening of the hole must match the surface it is being displayed on. If the surface is flat, the opening should be flat and if the surface is curved, the opening should match that exact curve. In Figure 4.14 below, the diagrams depict the cross-sections of a

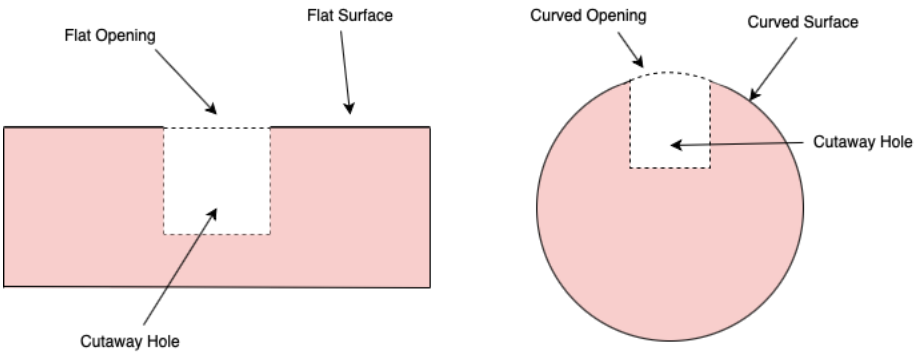

Figure 4.14: Cross-sections of a cuboid surface (left) and a cylindrical surface (right). The hole opening keeps the same shape as the curve of the surface.

cuboid and cylinder. Both of the surfaces have the same holes cut out from them, except the openings are different, given the different surfaces.

The hole openings were created manually in Blender, using the Boolean modifiers. The intersection Boolean modifier was utilised to get the intersection between the surface and the hole itself, leaving just the opening of the hole in the surface. The main idea for automating this process was by somehow getting the intersection between the cutaway hole and the surface at runtime in the Unity application and then applying the hole opening shader to the intersection area.

4.2.1.1 Intersection Shader Explained

This concept began with the creation of the “Intersect” shader file. This shader can be applied to an object and whenever this object is intersecting another object in the scene, the intersection is highlighted. The idea was to apply this shader to the hole itself, highlighting the hole opening (the intersection) in the surface.

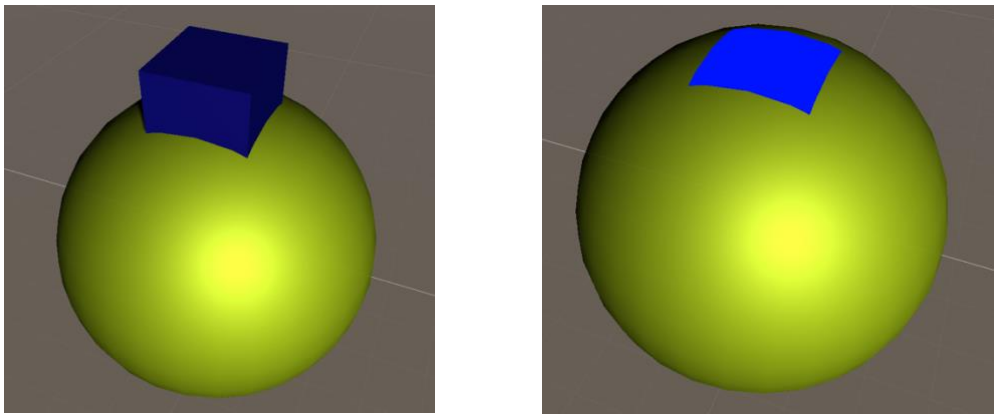

Figure 4.15: The intersection shader being applied to the blue cube, therefore highlighting the intersection between the cube and the sphere.

This shader works by using `ZTest Greater`. The default for all shaders is to use `ZTest LEqual` also known as “less than or equal”, which means if the object is closer to the camera than the closest previously drawn opaque object, it will render. Therefore, `ZTest Greater` means that it only draws if something else was drawn closer, hence only showing on “intersections”.

The main problem with this solution is that the intersection shader only works on opaque objects. In order for the user to be able to look through the hole opening created by the intersection shader; the opening must be transparent. However, since the intersection can only be made from an opaque object, the opening will always be occluded, meaning the internal parts will appear as though they’re levitating above the hole and not inside of the hole. The “intersection” effect is reliant on having opaque geometry that has rendered to the depth buffer before the intersection shader renders. Transparent geometry both renders after most things and more importantly does not render to the depth buffer at all.

If given more time to do this project, this method of hole opening automation would be explored deeper to make the entire hole creation process more seamless and less tedious.

5 Conclusion

In summary, during the course of this project, the cutaway and transparency techniques of internal visualisation were successfully implemented in Augmented Reality with the help of Diminished Reality methodologies. In order to display the virtual components, both image and model tracking were researched and tested, outlining each of their pros and cons. This project highlights the usefulness of virtually visualising the inner components of real-world objects and how it betters the viewer's understanding of the object's infrastructure.

Altogether, this project accomplished all of the main goals and objectives set out in the beginning. However, there is further work that can be done in this subject area. This project just began to explore the concept of automation in creating cutaway obtrusions but did not successfully implement a working sample. Hopefully, the ideas outlined in this project can spark the advancement of this technology in the years to come.

6 Bibliography

[1] ARPost. 2021. *The Amazing Power Of Diminished Reality | ARPost*. [online] Available at: <https://arpost.co/2017/12/13/the-amazing-power-of-diminished-reality/> [Accessed 27 April 2021].

[2] Automotiveillustrations.com. 2021. [online] Available at: <http://www.automotiveillustrations.com/carimages/nissan-300z-chassis-magazine.jpg> [Accessed 23 April 2021].

[3] Boyajian, L., 2021. *New opportunities for augmented reality*. [online] Network World. Available at: <https://www.networkworld.com/article/3184071/new-opportunities-for-augmented-reality.html> [Accessed 27 April 2021].

[4] Burns, M. and Finkelstein, A., 2008. Adaptive cutaways for comprehensible rendering of polygonal scenes. *ACM Transactions on Graphics*, 27(5), pp.1-7.

[5] Chev.me. 2021. *Online ArUco markers generator*. [online] Available at: <https://chev.me/arucogen/> [Accessed 18 April 2021].

[6] Côté, S., 2021. *Augmented reality for underground infrastructure: the problem of spatial perception*. [online] Communities.bentley.com. Available at: <https://communities.bentley.com/other/old_site_member_blogs/bentley_employees/b/stephanecotes_blog/posts/augmentation-of-subsurface-utilities-the-problem-of-spatial-perception> [Accessed 8 April 2021].

[7] Docs.blender.org. 2021. *Boolean Modifier — Blender Manual*. [online] Available at: <https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/booleans.html> [Accessed 8 April 2021].

[8] En.wikipedia.org. 2021. *Cutaway drawing - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Cutaway_drawing> [Accessed 29 April 2021].

[9] En.wikipedia.org. 2021. *Normal (geometry) - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Normal_(geometry)> [Accessed 8 April 2021].

[10] Enlightenment.org. 2021. *Evas GL Programming Guide*. [online] Available at: <https://www.enlightenment.org/playground/evas-gl.md> [Accessed 15 April 2021].

[11] Imaterialise.helpjuice.com. 2021. *Wat Are Inverted Normals? | Help Center | i.materialise*. [online] Available at: <https://imaterialise.helpjuice.com/design-printing/inverted-normals> [Accessed 8 April 2021].

[12] Konev, A., Matusich, M., Viola, I., Schulze, H., Cornel, D. and Waser, J., 2018. Fast cutaway visualization of sub-terrain tubular networks. *Computers & Graphics*, 75, pp.25-35.

[13] La Vida Leica. 2021. *Leica Updates S and S2/S2-P Firmware | La Vida Leica!*. [online] Available at: <https://lavidaleica.com/content/leica-updates-s-and-s2s2-p-firmware> [Accessed 29 April 2021].

[14] Learn.foundry.com. 2021. *Directional Light*. [online] Available at: <https://learn.foundry.com/modo/902/content/help/pages/shading_lighting/lights/directional_light.html> [Accessed 25 April 2021].

[15] Library.vuforia.com. 2021. *Best Practices for Designing and Developing Image-Based Targets | VuforiaLibrary*. [online] Available at: <https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html> [Accessed 17 April 2021].

[16] Library.vuforia.com. 2021. *Image Targets | VuforiaLibrary*. [online] Available at: <https://library.vuforia.com/features/images/image-targets.html> [Accessed 17 April 2021].

[17] Library.vuforia.com. 2021. *Model Targets | VuforiaLibrary*. [online] Available at: <https://library.vuforia.com/features/objects/model-targets.html> [Accessed 19 April 2021].

[18] Library.vuforia.com. 2021. *Model Targets Supported Objects & CAD Model Best Practices | VuforiaLibrary*. [online] Available at: <https://library.vuforia.com/articles/Solution/model-targets-supported-objects.html> [Accessed 19 April 2021].

[19] Mendez, E. and Schmalstieg, D., 2009. Importance masks for revealing occluded objects in augmented reality. *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology - VRST '09*,.

[20] Mori, S., Ikeda, S. & Saito, H. A survey of diminished reality: Techniques for visually concealing, eliminating, and seeing through real objects. IPSJ T Comput Vis Appl 9, 17 (2017).

[21] PCMAG. 2021. *Definition of alpha blending*. [online] Available at: <https://www.pcmag.com/encyclopedia/term/alpha-blending> [Accessed 25 April 2021].

[22] raywenderlich.com. 2021. *Introduction to Shaders in Unity*. [online] Available at: <https://www.raywenderlich.com/5671826-introduction-to-shaders-in-unity#toc-anchor-001> [Accessed 15 April 2021].

[23] Ronja-tutorials.com. 2021. *Stencil Buffers*. [online] Available at: <https://www.ronja-tutorials.com/post/022-stencil-buffers/> [Accessed 15 April 2021].

[24] Shawnlehner.github.io. 2021. *ARMaker - Augmented Reality Marker Generator*. [online] Available at: <https://shawnlehner.github.io/ARMaker/> [Accessed 18 April 2021].

[25] Tang, S., 2014. Simulating Transparency and Cutaway to Visualize 3D Internal Information for Tangible Uls. [online] pp.22-25. Available at: <https://dspace.mit.edu/handle/1721.1/95613> [Accessed 23 April 2021].

[26] Unity Learn. 2021. *Introduction to Lighting and Rendering - Unity Learn*. [online] Available at: <https://learn.unity.com/tutorial/introduction-to-lighting-and-rendering#5c7f8528edbc2a002053b529> [Accessed 25 April 2021].

[27] Unity Technologies, 2021. *Unity - Manual: Meshes, Materials, Shaders and Textures*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/Shaders.html> [Accessed 15 April 2021].

[28] Wilmot Li, Lincoln Ritter, Maneesh Agrawala, Brian Curless, and David Salesin. 2007. Interactive cutaway illustrations of complex 3D models. In ACM SIGGRAPH 2007.

[29] Zollmann, S., Grasset, R., Reitmayr, G. and Langlotz, T., 2014. Image-based X-ray visualization techniques for spatial understanding in outdoor augmented reality. *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design*,.

7 Appendix

7.1 Shader Code

7.1.1 HoleOpening.shader

```
Shader "Custom/HoleOpening" {
    Properties {
        _StencilRef("Stencil Reference Value", Int) = 1
    }
    SubShader {
        Tags {
            "Queue" = "Geometry-100"
        }
        ColorMask 0
        ZWrite off
        Stencil {
            Ref[_StencilRef]
            Comp always
            Pass replace
        }
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            struct appdata {
                float4 vertex : POSITION;
            };
            struct v2f {
                float4 pos : SV_POSITION;
            };
            v2f vert(appdata v) {
                v2f o;
                o.pos = UnityObjectToClipPos(v.vertex);
                return o;
            }
            half4 frag(v2f i) : COLOR {
                return half4(1, 1, 0, 1);
            }
            ENDCG
        }
    }
}
```

7.1.2 HoleInternal.shader

```
Shader "Custom/HoleInternal" {
    Properties {
        _StencilRef("Stencil Reference Value", Int) = 1
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        Stencil {
            Ref[_StencilRef]
            Comp equal
            Pass keep
            Fail keep
        }
        CGPROGRAM
        #pragma surface surf Standard fullforwardshadows
        #pragma target 3.0
        sampler2D _MainTex;
        struct Input {
            float2 uv_MainTex;
        };
        half _Glossiness;
        half _Metallic;
        fixed4 _Color;
        void surf (Input IN, inout SurfaceOutputStandard o) {
            fixed4 c = tex2D(_MainTex, IN.uv_MainTex) * _Color;
            o.Albedo = c.rgb;
            o.Metallic = _Metallic;
            o.Smoothness = _Glossiness;
            o.Alpha = c.a;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

7.1.3 Intersect.shader

```
Shader "Custom/Intersect"
{
    Properties
    {
        [HDR] _Color("Color", Color) = (1,1,1,1)
    }
    SubShader
    {
        Tags { "RenderType"="Transparent"
```

```
        "Queue" = "Overlay"
        }
        ZWrite Off
        Pass
        {
            Cull Back
            ZTest Greater
            ColorMask 0
            Stencil {
                Ref 1
                Comp Always
                Pass Replace
            }
        }

        Pass
        {
            Cull Off
            ZTest Greater
            Stencil {
                Ref 1
                Comp NotEqual
            }
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            struct appdata
            {
                float4 vertex : POSITION;
            };
            struct v2f
            {
                float4 vertex : SV_POSITION;
            };
            float4 _Color;
            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                return o;
            }
            fixed4 frag (v2f i) : SV_Target
            {
                return _Color;
            }
            ENDCG
        }
    }
}
```

7.2 C# Code

7.2.1 changeMaterial.cs

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

public class changeMaterial : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject DS4Internals;
    public GameObject Hole1;
    public GameObject Hole2;
    public GameObject Hole3;
    public GameObject Hole4;
    void Start()
    {
        DS4Internals =
GameObject.Find("/ModelTarget/ds4(holes_masks)/ds4_internals");
        Hole1 = GameObject.Find("/ModelTarget/ds4(holes_masks)/ds4_left_hole");
        Hole2 = GameObject.Find("/ModelTarget/ds4(holes_masks)/ds4_right_hole");
        Hole3 = GameObject.Find("/ModelTarget/ds4(holes_masks)/ds4_dpad_hole");
        Hole4 = GameObject.Find("/ModelTarget/ds4(holes_masks)/ds4_buttons_hole");
    }
    public void MakeTransparent(Material Material){
        DS4Internals.GetComponent<Renderer>().material = Material; //change
material of ds4 internals to transparent

        Hole1.SetActive(false);
        Hole2.SetActive(false);
        Hole3.SetActive(false);
        Hole4.SetActive(false);
    }

    public void MakeCutaway(Material Material){
        DS4Internals.GetComponent<Renderer>().material = Material; //change
material of ds4 internals to transparent

    }
    // Update is called once per frame
    void Update()
    {

    }
}
```

7.2.2 sceneSwitcher.cs

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

public class sceneSwitcher : MonoBehaviour
{
    public void SwitchScene(string sceneName)
    {
        Application.LoadLevel(sceneName);
    }
}
```

7.2.3 toggleInvisible.cs

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


public class toggleInvisible : MonoBehaviour
{
    public void HideObject(GameObject GameObjectToHide)
    {
        if (GameObjectToHide.activeInHierarchy)
        {
            GameObjectToHide.SetActive(false);
        }
        else
        {
            GameObjectToHide.SetActive(true);
        }
    }
}
```

8 Glossary

- **Augmented Reality (AR):** Virtually adding an object to the real-world to provide an interactive experience for the user.
- **Diminished Reality (DR):** Opposite of Augmented Reality. Virtually removing a real-world object from an augmented experience.
- **Mediated Reality:** The utilisation of both AR and DR together.
- **Shader:** A programmable file that is used to change the appearance of virtual models.