Technical University of Crete

School of Electrical and Computer Engineering

Visual Representation of Cultural Monuments in Augmented Reality using Marker-less Localization

Christos Kavallaris

Chania |Crete , May 2023

Περίληψη

Η παρούσα διπλωματική διερευνά το σχεδιασμό, την ανάπτυξη και την υλοποίηση μιας εφαρμογής επαυξημένης πραγματικότητας για κινητά τηλέφωνα για τους επισκέπτες του αρχαιολογικού χώρου των Δελφών, με στόχο την ενίσχυση της εμπειρίας τους και της κατανόησης της πολιτιστικής κληρονομιάς του χώρου. Η εφαρμογή αυτή επιτρέπει την επαυξημένη απεικόνιση επιλεγμένων μνημείων χωρίς την ανάγκη αναγνωριστικών εικόνων και παρέχει μια καθηλωτική και διαδραστική εμπειρία. Ως κύριο γνώμονα είχαμε την καλύτερη κατανόηση της πολιτιστικής σημασίας και αρχαιολογικής αβεβαιότητας των μνημείων μέσω της δυνατότητας προεπισκόπησης διαφορετικών πιθανών σεναρίων αυτών, σύμφωνα με σχετική έρευνα που έχει διεξαχθεί.

Η εφαρμογή επιτρέπει στους χρήστες να περιηγηθούν στον ιστορικό χώρο με προορισμό ένα επιλεγμένο μνημείο, παρέχοντας έναν διαδραστικό και ελκυστικό τρόπο εξερεύνησης της περιοχής. Η εφαρμογή χρησιμοποιεί τεχνολογίες Επαυξημένης Πραγματικότητας και τρισδιάστατης μοντελοποίησης για να επιτευχθεί η ακριβής απεικόνιση των μνημείων, μαζί με σχετικές πληροφορίες αυτών, συμπεριλαμβανομένου του ιστορικού και πολιτιστικού πλαισίου τους και των διαφόρων πιθανών σεναρίων. Η δυνατότητα προεπισκόπησης των πολλών διαφορετικών σεναρίων των διαθέσιμων μνημείων επιτρέπει στους επισκέπτες να κατανοήσουν καλύτερα και να παρατηρήσουν την αρχαιολογική τους αβεβαιότητα.

Η εφαρμογή έχει σχεδιαστεί ώστε να είναι διαισθητική και φιλική προς τον χρήστη με ένα εύχρηστο περιβάλλον που παρέχει πρόσβαση σε πληροφορίες και διαδραστικά χαρακτηριστικά. Η διπλωματική αυτή εργασία θα καλύψει τις τεχνικές λεπτομέρειες της υλοποίησης της εφαρμογής, συμπεριλαμβανομένης της χρήσης τεχνολογιών AP, τρισδιάστατης μοντελοποίησης, βάσεων δεδομένων σε πραγματικό χρόνο και σχεδιασμού περιβάλλοντος χρήστη. Συνολικά, αυτή η εφαρμογή στοχεύει να παρέχει έναν καινοτόμο και ελκυστικό τρόπο για τους επισκέπτες να βιώσουν και να μάθουν για την πολιτιστική σημασία των μνημείων στους Δελφούς, επιτρέποντάς τους παράλληλα να κατανοήσουν την πολιτιστική αβεβαιότητα των μνημείων και να την παρατηρήσουν σε πραγματικό χρόνο.

Abstract

This thesis explores the design, development, and implementation of an augmented reality mobile application for Delphi guests, with the aim of enhancing their experience and understanding of the cultural heritage of the Delphi Archaeological site. This application enables marker-less visualization of select monuments and provides an immersive and interactive experience. It is designed to provide a better understanding of the cultural significance of the monuments through the ability to preview different potential scenarios, based on the research carried out.

The application allows users to navigate through the historical site to a selected monument, providing an interactive and engaging way to explore the area. The application uses AR technologies and 3D modeling to allow users to visualize the monument and preview relative information about it, including historical and cultural context and different potential scenarios for the monument. The ability to preview the many different scientifically proven scenarios of the monument allows guests to better understand their cultural uncertainty.

The application is designed to be intuitive and user-friendly with an easy-to-use interface that provides access to information and interactive features. The thesis will cover the technical details of the development of the application, including the use of AR technologies, 3D modeling, real-time databases, and user interface design. In addition, user testing and feedback will be conducted to evaluate the effectiveness of the application in achieving its goals, including improved understanding and participation in the Delphi cultural heritage. Overall, this application aims to provide an innovative and engaging way for guests to experience and learn about the cultural significance of the monuments in Delphi, while also allowing them to understand the cultural uncertainty of the monuments and preview them in real time.

Acknowledgements

Initially, I am sincerely grateful to my supervisor professor Aikaterini Mania, for their guidance, expertise, and unwavering support throughout the entire research process. Their valuable insights, constructive feedback, and dedication have played a significant role in shaping this thesis.

I would like to extend my heartfelt appreciation to Giannis Kritikos, Andeas Polychronakis and all my colleagues from the SURREAL team for their valuable contributions and recommendations. Their expertise and encouragement have greatly enriched my understanding of this thesis' subject.

I would also like to express my gratitude to my family and friends for their unwavering support, patience, and understanding throughout this academic journey. Their encouragement and belief in me have been a constant source of motivation.

While it is impossible to mention everyone individually, please know that your contribution, no matter how big or small, has certainly not gone unnoticed. I am sincerely grateful for your support and encouragement, as this thesis wouldn't have been possible without it.

This research forms part of the project 3D4DEPLHI, co-financed by the European Union and Greek funds through the Operational Program Competitiveness, Entrepreneurship, and Innovation, under the call Specific Actions, Open Innovation for Culture

Publications

Psalti, A., Tsakoumaki, M., Mamaloukaki, C., Xinogalos, M., Bolanakis, N., Kavallaris, C., Polychronakis, A., Mania, K. and Maravelakis, E. (2023). Advanced Digitization Methods for the 3D Visualization and Interpretation of Cultural Heritage: The Sphinx of the Naxians at Delphi. 3rd International Conference Transdisciplinary Multispectral Modelling and Cooperation for the Preservation of Cultural Heritage.

Contents

List of Figures

List of Source Codes

Chapter 1

Introduction

1.1 Purpose of the Thesis

Augmented Reality (AR) has emerged as one of the fastest-growing technologies in recent years, revolutionizing the way we perceive and interact with our surroundings. AR allows for the integration of virtual objects into the real world, creating an enhanced experience that is both immersive and engaging. As a result, AR has gained popularity in various fields, including education, entertainment, and marketing. However, an area where AR has shown particular promise is in the visualization of cultural monuments.

Visualizing cultural monuments has always been a challenge for archaeologists and historians, as it requires a delicate balance between accuracy and accessibility. However, with the advent of AR technology, this challenge has become much easier to overcome. AR can be used to create digital reconstructions of cultural monuments, allowing for a more immersive and engaging experience. This technology can also be used to provide contextual information and educational resources, making the experience more informative and enriching.

The significance of AR in the visualization of cultural monuments is evident in its ability to bridge the gap between the past and the present. By providing a dynamic and interactive experience, AR allows individuals to engage with history in a way that was not previously possible. Furthermore, the advantages of AR in the visualization of cultural monuments extend beyond education and entertainment. AR can be used to preserve and protect cultural heritage sites, as it allows virtual reconstruction and documentation. As such, AR represents a powerful tool for the preservation and celebration of cultural heritage, as well as a means of engaging and inspiring future generations.

1.2 Brief Description

The Delphi AR app is a mobile AR application that has the potential to transform and enhance the tourist experience at the archaeological site of Delphi. The application will enable tour hosts to superimpose 3D reconstructed models of some of the site's monuments and visualize them through AR. Different 3D models of these monuments have been created, highlighting the different scenarios for each and pointing out their archaeological uncertainty. This will provide tourists with an immersive and engaging experience, allowing them to see and learn about these monuments through this new innovative medium.

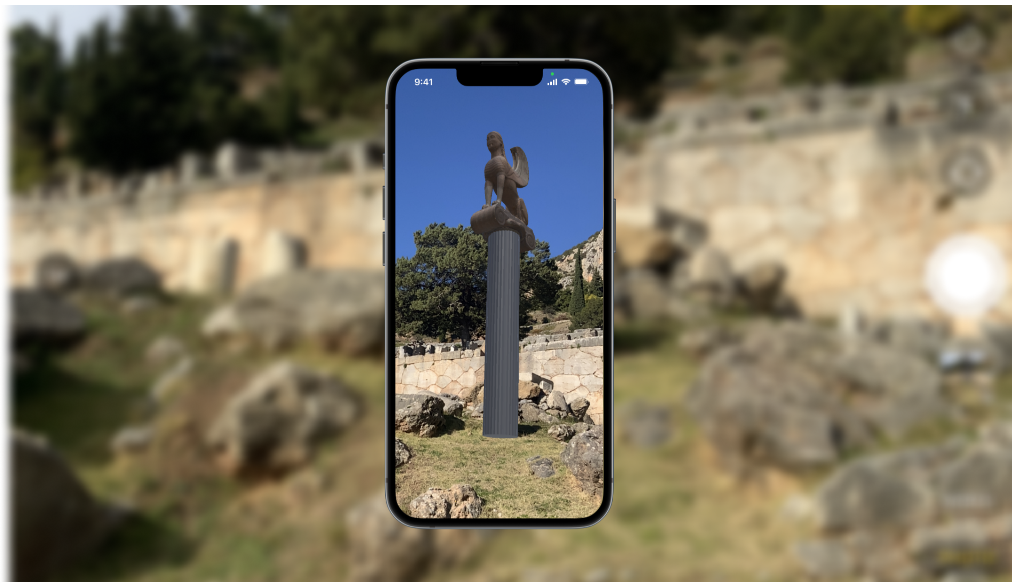


Figure 1.1: App Preview Visualization

In general, we have developed an outdoors, on-site Augmented Reality application for mobile devices that will be used for tour purposes at the Archaeological site of Delphi. Our implementation will support both Android and iOS devices that are compatible with the Hardware specifications that we have listed later in the Thesis. The user will be able to visit Delphi and either setup a new Augmented Experience for a specific on-site monument, along with its relevant info or be guided through the site and experience the different already registered augmented monuments.

More specifically, our AR application was developed in Unity3D's game engine using its AR Foundation packages. We have also implemented a Real-time database through Firebase that will handle and ultimately sync all of our data through the different users, providing us with a more seamless and hassle-free experience. In order to achieve a Marker-less AR experience will have also opted to use Google's Cloud Anchors and made it possible for us to spatially scan any desired location and place the corresponding cloud anchors that our superimposed monuments will utilize.

The overall aim of the Delphi AR App is to demonstrate the potential of AR technology in cultural heritage tourism. By developing and evaluating this mobile AR application, we hope to show that the Delphi AR app can be used to enhance the tourist experience at cultural heritage sites, making them more accessible, engaging, and informative. Moreover, by focusing on the archaeological site in Delphi, the Delphi AR App aims to demonstrate the applicability of this technology in real-world settings, where it can be used to preserve and promote cultural heritage, while also focusing on the different uncertainty scenarios.

The benefits of the Delphi AR App are twofold. First, it will provide a practical application of AR technology in cultural heritage tourism, making it accessible to both guides and tourists, to accurately superimpose the site's monuments and have a better understanding of them. Second, it will contribute to the growing body of literature on AR and cultural heritage, providing information on the challenges and opportunities associated with the use of an AR app in real-world settings. Ultimately, we hope that this will inspire further research and development in this area, leading to the creation of more innovative and impactful AR applications in this field.

1.3 Structure of the Thesis

Regarding the structure of our thesis, all of its aspects will be presented in full detail in the following chapters. The first chapters will focus on the research that has been done and the technologies that were used. Continuing form that, we will analyze the final result and how it has been implemented.

The second chapter will provide an overview of the current state-of-the-art regarding AR applications, the medium of AR, and how AR is used in cultural heritage. This chapter will include an in-depth review of the literature and will analyze previous research in the field.

In Chapter 3, we describe the technology and resources used to develop this AR app. Specifically, we will discuss the tools used such as Unity, Blender, and Firebase. This chapter will also describe the user experience of your app, including use cases and personas. There will be a rough overview of the database used and the different APIs that have been implemented.

In Chapter 4, we discuss the user experience of our app in more detail. Specifically, you will describe how a tourist would use the app and navigate it. We will go over the whole design of the user interface and explain the reason behind some choices.

Chapter five will analyze the technical aspects of your app. We will provide a detailed analysis of the code and libraries used in the development of your application. This chapter will also describe the various technologies and analyze their implementations and the required modifications / challenges that we encountered.

Finally, Chapter six summarizes the results of your app's field testing and user feedback. We will evaluate the results and draw conclusions about the effectiveness and the app's potential. This chapter will also include any future improvements that we think would benefit the app and improve it, based on the aforementioned results.

Chapter 2

Research Overview

2.1 Introduction

In this chapter, it is important to provide a concise representation of the current development of Augmented Reality (AR), which has grown exponentially in recent years. We will list various AR mediums, different tracking technologies, and analyze some existing AR applications that have been used around cultural heritage and/or have some collaborative aspect. We will then delve into the importance of archaeological uncertainty in cultural heritage and how it can be aided by Augmented Reality.

Through all this, one can gain a better understanding of the current state of AR, and more specifically MAR (Mobile Augmented Reality) and thus be able to better evaluate our developed application and grasp its potential and importance regarding monument augmentation and its cultural uncertainty.

2.2 Augmented Reality

In general, Augmented Reality (AR) is a technology that enables the user to superimpose 3D objects on the real world. It enhances the physical world with virtual elements that interact or blend into the given environment. [2] Contrary to virtual reality (VR), visualized 3D objects are tracked and incorporated into our actual surroundings and not in a computer-generated artificial environment.

This exact distinction is one of the bigger challenges regarding Augmented Reality and plays a vital role in the user's immersion and the overall experience. The technology has seen some major breakthroughs through the years that should be mentioned.

2.2.1 Brief History of AR

The concept of object augmentation dates all the way back to the early 1960s, with the development of the "Sword of Damocles", a head-mounted display (HMD), which used computer-generated graphics to augment the user's surroundings. [3]

However, it was not until the 1990s that AR technology saw significant progress with the development of "Virtual Fixtures" by Louis Rosenberg. It used two real physical robots, controlled by a full upper body exoskeleton worn by the user, and allowed interaction with the augmented virtual objects.[4] The benefits of Virtual Fixtures were primarily in its potential to enhance user performance and training in various fields, such as teleoperation, surgery, and manufacturing. The system allowed users to interact with virtual objects superimposed on the real world, providing them with real-time guidance and feedback to perform tasks with improved precision and accuracy. Louis Rosenberg's Virtual Fixtures system was a significant step in the early development of AR technology, demonstrating the potential for augmenting the real world with virtual information and enhancing human-computer interaction.

Following this, in 1992 the first AR system, known as KARMA-one, was developed by researchers at Columbia University. This system incorporated knowledge-based AR to automatically infer appropriate instruction sequences for repair and maintenance procedures.[5] KARMA-one aimed to provide maintenance workers with context-sensitive real-time information through an augmented reality interface. The system utilized a head-mounted display (HMD) to overlay digital information onto the real-world environment, assisting workers with performing complex maintenance tasks. Through computer vision and object recognition techniques, KARMA-one could recognize physical objects and provide relevant instructions and guidance to the user. The system used a knowledge-based approach to provide customized instructions, allowing workers to access and retrieve specific information based on the current task and context. The benefits of KARMA-one included improved efficiency and accuracy in maintenance tasks, reduced downtime, and improved worker training and knowledge transfer. By providing real-time guidance and access to relevant information, the system aimed to streamline maintenance processes and support workers in their decision-making.

Examples such as these make it apparent that, in order for AR to become more accessible to a wider audience, it's form factor will have to be reduced.

Around 1997 the first mobile AR system, called the "Touring Machine", was introduced by Steve Feiner. It used a see-through HMD with integral orientation tracking, a backpack that contained a computer, GPS, and a digital radio for web access.[6] The key contribution of the Touring Machine system was the ability to seamlessly integrate virtual content with the user's perception of the real world, providing a more immersive and intuitive experience to interact with complex data. It enabled users to gain a better understanding of intricate spatial relationships and visualize data in context, such as architectural models, medical imaging, or scientific simulations. Steve Feiner's work on the Touring Machine system and his broader contributions to augmented reality research have played a significant role in advancing the field and exploring the potential applications of AR in various domains, including education, design, and visualization.

Nowadays, with the advent of smartphones and tablets, which come pre-equipped with cameras and sensors, AR technology has become more accessible and its advantages can be better utilized. However, the accurate registration of virtual objects onto our real-world environment remains an issue, and multiple methods have been developed to facilitate this continuous tracking.

2.2.2 Registration problem & Tracking Methods

Registration in an Augmented Reality (AR) system is crucial for accurately presenting virtual information within the real environment. It involves aligning virtual objects with their corresponding real-world counterparts to create a seamless integration between the two. The quality of registration directly impacts the illusion of coexistence between the real and virtual worlds [1]. Unlike Virtual Reality (VR), where registration errors may result in visual-kinesthetic conflicts, AR conflicts are primarily visual and more easily detectable. Registration errors can be classified as static or dynamic. Static errors persist even when the user and environment are stationary. These errors can be due to factors such as poorly calibrated mechanical parts, incorrect tracker-to-eye ratios, field-of-view parameters, and optical lens distortions. While static errors can be minimized through proper calibration, dynamic errors arise when either the user's viewpoint or the annotated object is in motion. Dynamic errors are significant contributors to the registration problem in Mobile Augmented Reality (MAR) and vary depending on the implementation.

In early AR systems, end-to-end system delays were the primary cause of dynamic errors. The time between a user's movement and the system's update of digital artifacts was often substantial, leading to misalignment. However, advances in hardware have significantly reduced system delays, shifting the focus to pose estimation as the main source of registration errors. Pose estimation refers to accurately determining the position and orientation of the viewer in relation to a real-world anchor. Various technologies, such as magnetic trackers, paper image markers, GPS, and inertial tracking, have been employed as real-world anchors depending on the application and available technologies.

For an AR system to overlay digital information onto the real world, it needs to track its position with 6 degrees of freedom (6 DoF), encompassing three variables for position and three for orientation. Modern smartphones are equipped with inertial sensors, GPS, and optical sensors that can provide the necessary data for these computations. Although there are multiple approaches to pose estimation in AR systems, our focus will be on implementations for consumer-grade smartphones. Each tracking approach has its advantages, depending on the specific use case. Moving on, we will explore various tracking methods that address the registration problem, each with its unique approach. By understanding the strengths and limitations of these methods, we can gain insight into effective registration techniques for AR applications.

2.2.2.1 Marker-Based Tracking

While Rosenberg had already introduced the concept of Marker-Assisted Augmented Reality, it was not until the early 2000s with the development of ARToolKit by Hirokazu Kato and Mark Billinghurst that this field started to grow and became more widely utilized. [7] In general, this implementation of AR uses physical markers, usually in some form of a specific pattern or QR code, to track the position and orientation of the real-world coordinates. These markers are detected by your camera and are used to align any virtual content that the user may want to visualize. The general steps in a Marker-Assisted AR application are the following:

1. The user points the camera or a sensor at the provided physical marker, which is usually a printed image or a specific pattern, QR code.
2. The AR system detects the marker provided, using computer vision techniques, such as feature/image detection.
3. Calculates the marker's position and orientation relative to the camera or sensor.
4. After this calibration, the AR system can superimpose virtual content, such as 3D models, images, or text, on the camera view, aligned with this physical marker.
5. As the user moves the camera or sensor, the AR system continually updates the position and orientation of the virtual content relative to the physical marker, creating the illusion that the virtual content is part of the real world, and thus successfully superimposing it.

Marker-Assisted AR Systems have become increasingly popular in recent years, as they provide precise tracking and registration capabilities. They are usually easier to use and very user-friendly. Their main use case is in indoor spaces and usually for augmenting small 3D objects, which can prove to be quite useful in many use cases.

However, these systems come with their limitations and restrictions, which are worth noting, as they played an integral part in our decision not to go with this approach in our application. Marker-based AR systems tend to be limited in their flexibility because they are immediately dependent on the marker they are using. They are not ideal for outdoor applications, as they are affected by lighting variations and can prove to be unreliable as far as tracking is concerned. A study by Manresa Yee [10] evaluated the performance of such systems in a museum environment and concluded that variations in lighting, such as shadows or glares, can significantly interfere with the system's tracking accuracy and greatly decrease the user's experience. Furthermore, their field-of-view can also be limited, as the marker always has to be within the camera's view to initiate and keep track of the AR experience.

2.2.2.2 Feature-Based Tracking

Feature-Based tracking, is a method that relies on the identification and tracking of characteristics of the real world. These features include corners, edges, and other distinctive points in the environment. The system uses computer vision algorithms to track these features and determine the camera's pose and the virtual object's position. However, natural feature tracking does have its limitations. The system's tracking capability is heavily dependent on the number and quality of features present in the scene. If there are only a few features or if they are occluded or poorly illuminated, the tracking performance may be compromised. Variations in lighting conditions, changes in the environment, or dynamic objects can also pose challenges to the tracking of natural features. This method is flexible and works well in various environments, but can be affected by changes in lighting and occlusion.

2.2.2.3 Sensor-Based Tracking

Sensor-based tracking utilizes the sensors present in the device, such as the accelerometer, gyroscope, and magnetometer, to determine the position and orientation of the device in space. Accelerometers measure the acceleration forces acting on the device, allowing the detection of linear movements. Gyroscopes, on the other hand, measure the angular velocity of the device, enabling the tracking of rotational movements. By combining these sensor readings, it becomes possible to estimate the device's position and orientation changes in real-time. Magnetometers are used to detect the Earth's magnetic field, providing information about the device's orientation with respect to the Earth's magnetic north. This allows orientation tracking relative to the magnetic field, which can be useful for certain applications. This method is used in conjunction with other tracking methods to improve their accuracy and robustness.

2.2.2.4 Model-based Tracking

This method that involves using a 3D model of the environment to track the camera's pose and the virtual object's position. The system matches the features of the real environment with the features of the 3D model to determine the camera's pose. This method is useful for tracking in large environments and can be used to create realistic virtual environments. This approach typically involves the construction of 3D models using edge detection techniques. In some cases, a predefined model is utilized to track the resemblance of an object in the environment, such as tracking a moving car on a street. It should be noted that model-based tracking often requires more processing power compared to other tracking techniques.

2.2.2.5 Hybrid Tracking

Hybrid tracking combines different tracking methods to overcome the limitations of each method. For example, a system can use both sensor-based tracking and natural feature tracking to improve its accuracy and robustness.

2.2.3 Markerless AR Tracking

Due to these limitations mentioned in Marker-Based Tracking and the fact that the installation of various markers on a historical site would be a bit intrusive, we decided to use another technology to track and visualize our AR experience. Also in the early 2000s, the concept of markerless AR was introduced, but it was not until the development of robust computer vision algorithms that the technology became practical and received major advancements, especially as far as mobile hardware is concerned.

With the help of machine learning and computer vision algorithms, we are able to accurately superimpose 3D objects onto the real world without the need for markers or reference points. Although this might seem like a minor advance compared to the marker-based AR mentioned above, in reality it provided much needed flexibility. The user could now customize the AR experience to his location and preferences, without being greatly affected by changes in the environment.

The adoption of the SLAM (Simultaneous Localization and Mapping) concept in augmented reality systems has revolutionized the ability to extract structure information from the real world. SLAM allows for the creation and updating of a map of the environment while simultaneously localizing the system's position within it. This concept has significantly expanded the capabilities of AR systems, enabling them to track not only planar surfaces but also more complex geometries and 3D structures.

One notable development in AR is the emergence of PTAM (Parallel Tracking and Mapping), a process that separates the tracking of the camera from the mapping of the environment components. This separation has led to significant performance improvements in AR systems. By leveraging SLAM and implementing PTAM, we can create detailed and dynamic maps of the environment, enabling accurate registration and seamless integration of virtual objects into the real world.

According to a study by Liu [11] on this technology and the advances over its predecessor, it was concluded that while both technologies can enhance the visitor experience at cultural heritage sites, Markerless AR would be more suitable for outdoor environments that have a larger area to cover. The users found that the absence of initialization markers made the experience more immersive and engaging while giving them a better understanding and appreciation of the historical and cultural significance of the site.

In this thesis, having taken all of the above conclusions into consideration, we too decided to implement a Marker-less AR approach that enabled as to augmented our monuments without the need of any specific markings or codes that would distract the user from their experience.

2.3 AR Experiences Research

As a next step in our research, after having concluded on the technology we will use, it was considered crucial to present some of the articles and AR experiences that have already been developed that were an inspiration to us and guided us through the development of our application.

2.3.1 Cultural Heritage Applications

One of the early publications that caught our attention was the paper "Augmented Reality for Historical Storytelling: The INCIPICT Project for the Reconstruction of Tangible and Intangible Image of L'Aquila's Historical Center" [12] that describes a project aimed at reconstructing the tangible and intangible image of L'Aquila's historical center through augmented reality.

The authors highlighted the importance of using AR as a tool for historical storytelling and the preservation of cultural heritage, especially in areas affected by natural

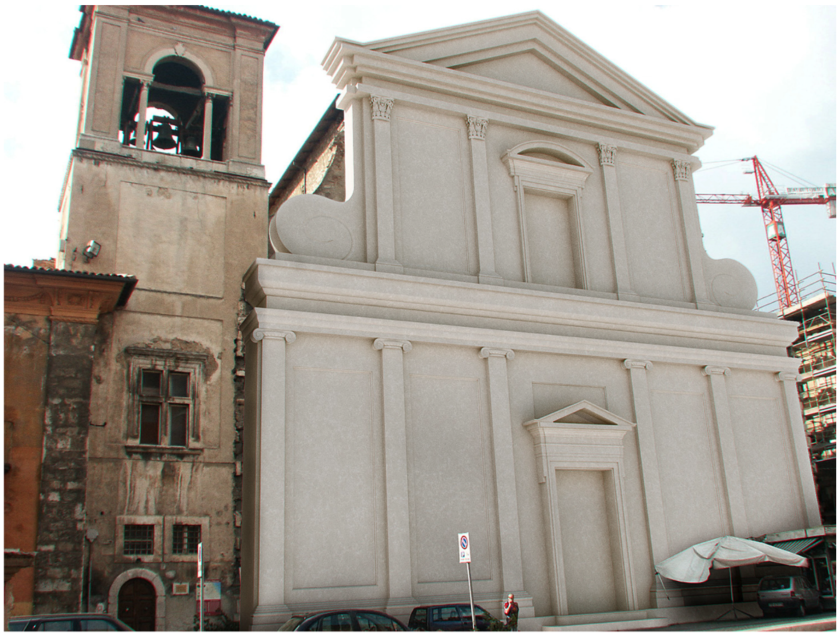


Figure 2.1: Augmented reconstruction of facade through the INCIPICT Project [12]

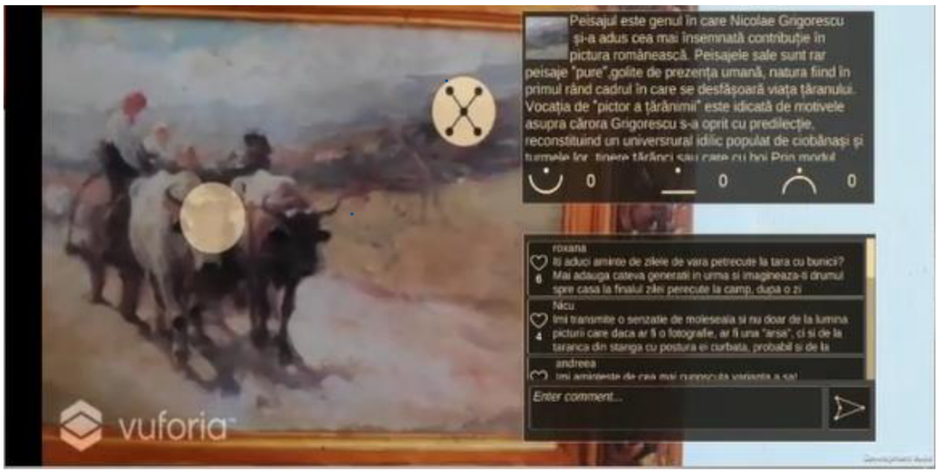
disasters. More specifically, they developed an AR system that made use of the aforementioned marker-based technology and allowed the user to visualize the historical centre of L'Aquila as it was before the 2009 earthquake and interact with it to gain additional information.

They concluded that AR can create a sense of presence and immersion that traditional methods cannot achieve, making it an effective tool for storytelling and interpretation, while enhancing the visitor's experience with additional information, context, and interactivity. However, they also pointed out some of the limitations, such as the need for specialized markers, the potential for distraction from the physical environment, and the possibility that technical issues interfere with the experience.

In this thesis we took those issues into account and tried to minimize them through the use of markerless technology, a minimal user interface (UI), and a robust real-time database, all of which will be explained in the following chapters.

Another interesting publication that greatly influenced the development of our application was AR-Things [13], which introduces a collaborative AR mobile application that allows users to interact with overlaid models on top of existing museum exhibits.

The user can scan a QR code on each of the museum exhibits and have additional information projected in front of him. Multiple users could view the same model at any given point and provide feedback on it, which would be available and could be rated by other guests. This collaborative aspect increased the viewer's interaction and their overall immersion. Their minimal UI helped to not "overcrowd" the experience so that the user was able to focus solely on the important details and better understand the exhibits.


Peisajul este genul în care Nicolae Grigorescu și-a adus cea mai însemnată contribuție în pictura românească. Peisajele sale sunt rar peisaje "pure",golite de prezența umană, natura fiind în primul rând cadrul în care se desfășoară viața țăranului. Vocația de "pictor a țărănimii" este idicată de motivele asupra cărora Grigorescu s-a oprit cu predilecție, reconstituind un universrural idilic populat de ciobănași și turmele lor, tinere țărănci sau care cu boi Prin modul

0 0 0

roxana
6
Iti aduci aminte de zilele de vara petrecute la tara cu bunicii? Mai adauga cateva generatii in urma si imagineaza-ti drumul spre casa la finalul zilei petrecute la camp, dupa o zi

Nicu
4
Imi transmite o senzatie de moleseala si nu doar de la lumina picturii care daca ar fi o fotografie, ar fi una "arsa", ci si de la taranca din stanga cu postura ei curbata, probabil si de la

andreea
Imi aminteste de cea mai cunoscuta varianta a sa!

Figure 2.2: ARThings User Interface [13]

Having seen that, we decided to also implement in this thesis an overall minimal user interface (UI) that will not feel overwhelming to the user and will help to better visualize the actual monuments of our application. To make our AR experience more interactive and fun, a sandbox mode has also been introduced in our app, allowing visitors to freely move the augmented monuments and view them from whichever viewpoint they prefer.

2.4 Historical Cultural Uncertainty

Finally, since our application is primarily based on a few monuments in Delphi and their historical cultural uncertainty, it is vital that we elaborate a bit on that subject. The study of cultural monuments presents unique challenges due to historical uncertainties that arise from various factors such as fragmented or incomplete information, changes over time, interpretive bias, and subjective perspectives.[14] These uncertainties can significantly affect the interpretation and understanding of cultural monuments and their historical significance. In this section, we will discuss the key points associated with historical uncertainty in cultural monuments.

The historical context is crucial to understanding cultural monuments, as it provides information on their historical significance, cultural background, and relevant historical events or periods. However, historical records may be incomplete, inconsistent, or biased, leading to uncertainties in the interpretation of cultural monuments. For example, conflicting accounts or changes in interpretations over time can result in divergent narratives about the purpose, meaning, or origin of a cultural monument. Furthermore, fragmented or incomplete data, missing artifacts or documentation, and changes in cultural, social, or environmental conditions over time can further compound the historical uncertainties associated with cultural monuments.

Interpretation and attribution are significant challenges in the study of cultural monuments. Multiple attributions, debates between scholars, and issues related to authenticity and provenance can create uncertainties in understanding cultural monuments. Interpreting historical information can also be subjective and influenced by personal perspectives, cultural biases, and theoretical frameworks. Such interpretive biases can affect the understanding of cultural monuments and their historical significance, leading to uncertainties and complexities in their interpretation.

The limitations of historical information and the challenges of reconstructing the past based on limited evidence can also introduce uncertainties in the study of cultural monuments.[15] For example, missing or fragmented data can hinder a complete understanding of the historical context, purpose, or meaning of a cultural monument. Changes in political regimes, religious beliefs, or environmental factors over time can further add to the historical uncertainties associated with cultural monuments.

The importance of critical analysis and a multidisciplinary approach cannot be understated when addressing historical uncertainty in cultural monuments. Interdisciplinary research, the use of diverse sources, the critical evaluation of evidence, and the consideration of alternative perspectives are vital to navigate historical uncertainties and arrive at nuanced interpretations.

We will mainly focus on a handful different scenarios regarding the Sphinx, the Athenian Treasury, and the Ancient Theater of Delphi. All of them have been extensively researched, each one based on a different theory, and with their unique and significant differences. The ultimate goal of our application is to highlight these differences and make them clearer and more apparent to the user.

Chapter 3

Technological Background and Project Structure

3.1 Introduction

Before we begin to describe the overall user experience of our application and its detailed implementation, we must first establish an accurate technological foundation. We will start by introducing the different applications/SDKs that were used and define their key technical terms, as well as any specific features that were implemented.

A detailed description of the project structure of our application will also be provided, including a general introduction to the major systems that were implemented.

In general, this chapter aims to serve as a technical guide and can be used as an index for the technological background of our application.

3.2 Blender 3D

Blender is a free open source 3D creation software widely used to create and edit 3D models, animated movies, visual effects, and many more. It was mainly used in our thesis for 3D model manipulation. We were provided with realistic 3D renders of the different monuments that our app would be augmenting, as well as the different scenarios for each for them. These models were remodeled with Blender, and their polygon complexity was reduced to be better superimposed on our application. It was crucial for the efficiency of our system to lower their resolution, so that Unity could better process them, and thus reduce any buffering during initialization. The shaders and materials used for each reconstructed monument were carefully selected to make them feel realistic and make the user's experience more immersive. This, accompanied by the applied lighting and shadows, made the models not seem out of place and blend in with the surrounding environment.

3.3 Unity

Unity was chosen as our programming environment, not only because we were very familiar with it, but also because it provided a variety of different APIs that we could use. Unity's game development workflow is object-oriented and provides an easy way to export your application for different operating systems, which would be useful to us. Before we begin analyzing our project's structure and its different components, we should take a quick look at Unity's technical terminology and briefly mention each one of them.

3.3.1 Assets

Assets in Unity 3D include files such as models, scripts, textures, and sounds. Properly classifying assets early on helps ensure the success of the project.

3.3.2 Scenes

Scenes in Unity function as individual levels or areas of the game, and some developers create entire games in a single scene. Building a game in multiple scenes allows developers to allocate loading and testing times separately. However, we chose to have ours built in a single scene and offload any unused assets dynamically. This greatly reduced buffering times between menus and provided some more flexibility in coding.

3.3.3 Game-Objects

Game-Objects are active objects in the current open scene, with each game object containing at least one component, the Transform Component, which provides Unity with information about the object's position, rotation, and scaling through the Cartesian coordinates X, Y, Z. Other components can be added to an object to provide additional functionality. These are the main "building blocks" of our application.

3.3.4 Components

Components come in various forms and are attached to game objects to create behavior, determine appearance, and affect other aspects of the object's operation in the app. Common components integrated into Unity include rigid body components, lights, cameras, and particle transmitters. Scripts, which are components that extend or modify Unity's functionality, can also be written to create further interactive elements. We will elaborate further in the Implementation chapter of our thesis about each component that was used and analyze their functionality.

3.3.5 Scripts

Scripts are created and placed as components in game objects to give them functionality without any restrictions on the number of scripts that can be used. Unity's full-fledged scripting code editor, MonoDevelop, allows developers to program in either C# or JavaScript. C# was utilized in our case, as we had previous experience with it and were comfortable with its syntax and functionality. Some of the key scripts will be further analyzed and their code will be explained step by step.

3.3.6 Prefabs

Prefabs are prefabricated and stored versions of an object that can be reused in various parts of a program. Using prefabs allows complex objects with different elements and settings to be used at any time and modified individually. Prefabs were mainly used in our application not only for the different pop-ups and menus, to provide consistency, but also for our 3D models for the different scenarios.

3.4 Project Structure

Continuing, we will go through the main systems of our application and how they were utilized. We will justify each of the choices made and why these systems were thought to be suitable for the needs of our application. The detailed technical implementation of them will be thoroughly analyzed in a subsequent chapter.

3.4.1 General Requirements

The application's primary objective is to deliver a comprehensive and engaging experience to the end-user throughout their visit in Delphi. It is essential that the user is physically present at the designated location to fully access the application's features. The device must be equipped with specific sensors necessary to support 3D visualization in the real world, and their availability should be verified to ensure an optimal user experience.

The application must offer a reliable navigation system to facilitate seamless exploration of the spatially distributed content. Since all information is location-based, an intuitive interface is required to assist users in locating and accessing the available information relevant to their current position. It is important to note that the application will only be functional within the confines of Delphi's archaeological site, and its functionalities are focused our the site's monuments.

The target audience for the application includes both visitors and tour guides of Delphi's archaeological site, with no specific age or educational requirements. Additional functional requirements for the overall system are outlined below:

- The application should have a user-friendly and minimalist interface, ensuring ease of use for all users.
- The system must deliver immersive 3D visualizations to users within the Archaeological site of Delphi, as well as a clear distinction between the different monument scenarios.
- The application should provide clear instructions and guidance on how to navigate and utilize its features effectively.
- The application should avoid presenting redundant information, ensuring a streamlined and concise user experience.
- The system requires access to the GPS sensor to accurately determine the user's location within the area.
- The application must have access to the phone's camera to enable augmented reality overlays.
- Internet access is necessary for the application to fetch and update relevant data in real-time.

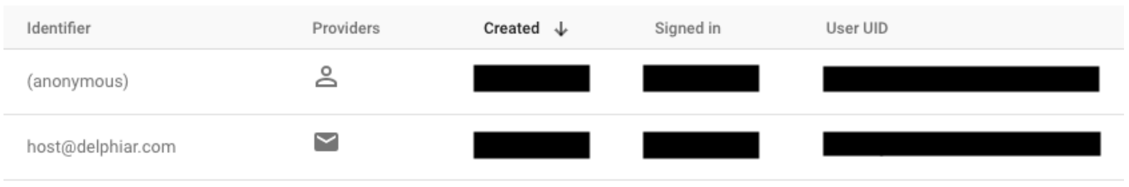
- The application should be designed with room for future upgrades and enhancements, allowing for continual improvement and expansion of its capabilities.

3.4.2 Real-time Database

Given the scale of our application and the fact that it aims to be used by a large number of tourists who visit the Delphi Archaeological Site daily, we needed a back-end solution to manage and store our data and authenticate the different users. To address this need, we have integrated Firebase's Real-time Database, a cloud-based NoSQL database, into our app.

3.4.2.1 User Authentication

A Real-Time Database like Firebase allows easy access for both the host and the various guests of the Archaeological Site. The host can easily log in using his specified credentials to make additions or changes to the database and import any new monuments that will be visualized through our app. All other users will access the database through an automatically generated anonymous account in order to simplify the authentication process and to make the user experience simpler and intuitive. There is no need to have a proper authentication for them, as they will only be accessing the stored database data and will not have any privileges to make changes to them.



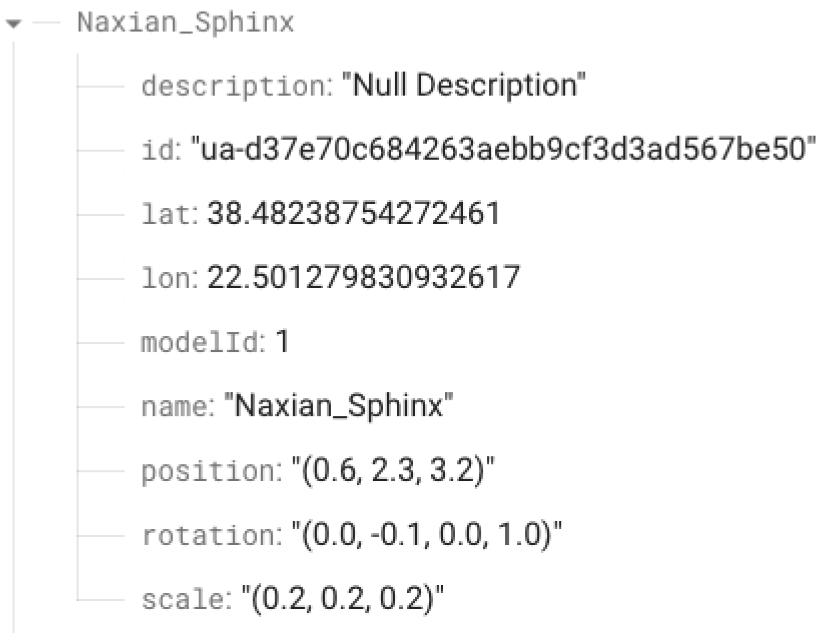
Identifier	Providers	Created ↓	Signed in	User UID
(anonymous)	👤	[redacted]	[redacted]	[redacted]
host@delphiar.com	✉	[redacted]	[redacted]	[redacted]

Figure 3.1: Firebase User Authentication

All users are assigned a specific and unique User UID that enables us to differentiate the host from the guests and to give them the corresponding credentials. When logging out of the app, the anonymous account to which the user was assigned will be deleted and no personal data will be stored at any point.

3.4.2.2 Monument Data Synchronization

Another vital feature of a Real-Time Database is that of data synchronization, allowing updates to be instantly propagated to all connected devices. This real-time capability ensures that our app's host and users always have up-to-date monument data, enabling a seamless and dynamic AR experience. Monument data can be added, updated, or deleted from the database, and these changes are immediately reflected on all connected devices, eliminating the need for manual data updates and ensuring that all users are viewing the most current and accurate version of the files. With that we were able to store all the necessary data that are needed in order to visualize and augment a specific monument making use of a flexible data structure in JSON format.



```
Naxian_Sphinx
    description: "Null Description"
    id: "ua-d37e70c684263aebb9cf3d3ad567be50"
    lat: 38.48238754272461
    lon: 22.501279830932617
    modelId: 1
    name: "Naxian_Sphinx"
    position: "(0.6, 2.3, 3.2)"
    rotation: "(0.0, -0.1, 0.0, 1.0)"
    scale: "(0.2, 0.2, 0.2)"
```

Figure 3.2: An example of a monument's stored JSON Data

JSON (JavaScript Object Notation) is a lightweight and widely used data-interchange format that is well suited for storing data and any needed characteristic values. JSON's simple format makes it easy to organize and store monument data in a structured manner. JSON allows us to represent complex data structures, such as monument properties like positioning, scale, rotation, name, description, and other relevant information, in a hierarchical tree-like structure. This makes it convenient to store and retrieve monument data in a format that is easy to understand and work with.

In order to better understand the structure and contents of this JSON format, let us take apart the example that is illustrated on Figure 3.2 regarding the monument of the Naxian's Sphinx. The various variables inside the JSON format are so:

- **Name:** Monument name for guest selection
- **Id:** Unique identifier for its cloud anchor
- **Description:** Information that will be presented to them for the selected monument
- **Lat:** Latitude real-world coordinate
- **Lon:** Latitude real-world coordinate
- **ModelId:** Unique model identifier
- **Position:** Local position of the model relevant to its corresponding anchor
- **Rotation:** Local rotation of the model relevant to its corresponding anchor
- **Scale:** Scale of the model

The importance and use of these values will become apparent in the Implementation section of our thesis.

3.4.3 AR Marker-less Anchors

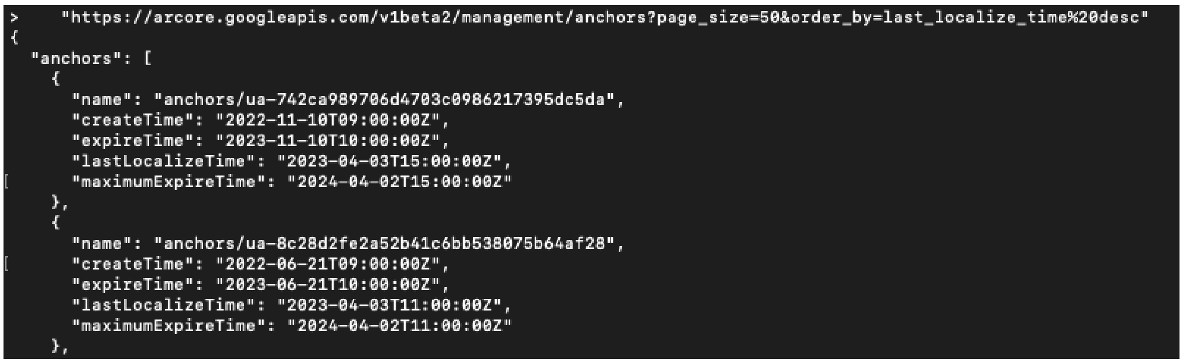
As we have previously discussed in the earlier sections of this thesis, a Markerless Anchor approach is best suited for our use case. In general, to correctly place a 3D object inside an AR scene, there needs to be an **anchor** point, which will be reference from the application and correctly place the desired 3D model on top of it, eliminating the need for any kind of predefined marker or image. This is achieved through the use of computer vision algorithms, which allow the device to scan and map the surrounding environment, recognize features and landmarks, and track the device's position and orientation in real-time.

At the beginning of our development, we experimented quite a bit with making our own **Markerless Anchor** system but it was considered too resource-heavy and very dependent on the hardware on which it was running. Since we want our application to run on the vast majority of mobile devices that any given tourist may have available, we decided to opt for a **Cloud Anchor** solution and, more specifically, Google's Cloud Anchor API. This provided us with a scalable and efficient solution to implement our anchors, which could easily be integrated into our Unity project.

The major advantage and one of the key reasons why we picked this particular solution is its resource load. It uses cloud computing to perform its calculations and offloads the processing needs to Google's cloud servers, reducing the resource requirements on the client devices. It utilizes a combination of computer vision and machine learning algorithms to map the physical environment and track the position and orientation of the AR models. This process involves analyzing large amounts of data and requires significant processing power, which is why its so beneficial to offload it to a cloud server. In addition to that, it also utilizes a technique called sparse point-cloud mapping, which further reduces the resource load, by analyzing only a subset of the data captured by the devices camera, rather than the entire frame.

In general, in order to maintain a consistent and accurate placement for our augmented monuments, we needed a way to store and retrieve each anchor data, so that they are identical throughout the uses of our application. Through the implementation and use of Google's Cloud Anchor API for Unity, we were able to have persistent Cloud Anchors that are able to sync to the different users and provide identical results in each iteration. After the correct mapping and registration of our anchors, each of them is assigned a unique id that is identifiable by the cloud servers and is forever linked to that specific anchor data. This token id is authenticated through a signed JWT using the RS256 algorithm, which will be further analyzed in the Implementation part of the thesis, and stored in the database so that it can later be retrieved.

All cloud anchors, in order to optimize efficiency and not clutter the database, have a maximum lifespan of 1 year, which is forced on us. This wasn't a setback for us, as re-registering an anchor is easily available through our application and it can be done by the host with not much complexity. The host is given regular annotations and help lines to help guide them through the process and successfully register an anchor. Moreover, since our use case is in an outdoor environment, re-registering an anchor annually would be important nonetheless. The surrounding environment can change over time, and it is vital that it is remapped to ensure the correct placement of our persistent cloud anchors.



```
>    "https://arcore.googleapis.com/v1beta2/management/anchors?page_size=50&order_by=last_localize_time%20desc"
{
  "anchors": [
    {
      "name": "anchors/ua-742ca989706d4703c0986217395dc5da",
      "createTime": "2022-11-10T09:00:00Z",
      "expireTime": "2023-11-10T10:00:00Z",
      "lastLocalizeTime": "2023-04-03T15:00:00Z",
[     "maximumExpireTime": "2024-04-02T15:00:00Z"
    },
    {
      "name": "anchors/ua-8c28d2fe2a52b41c6bb538075b64af28",
[     "createTime": "2022-06-21T09:00:00Z",
      "expireTime": "2023-06-21T10:00:00Z",
      "lastLocalizeTime": "2023-04-03T11:00:00Z",
      "maximumExpireTime": "2024-04-02T11:00:00Z"
    },
```

Figure 3.3: Example of two stored Cloud Anchors and their lifespan

3.4.4 AR Navigation

Having established a reliable way for us to not only store and load our available monuments, but also to have them accurately visualized, we also needed a way to guide the user towards their selected AR experience.

We initially experimented with the use of Mapbox's API, which provides us with guiding solutions and map visualizations. However, this solution is best suited for environments that have an established road network, in order to provide clear instructions and directions towards a selected destination. In our case, Delphi's Archaeological site is not sufficiently mapped, and thus we were not able to utilize their system. We also wanted to keep our application's interface as minimal as possible, so that the user would not be distracted and could focus solely on their AR experience.

With all of this in mind, we decided to implement a simple compass-like indicator that would assist the user and clearly indicate to them towards where they needed to go. This was accomplished by making use of the stored geographic coordinates that we have available in our database for each of our monuments. These coordinates are initialized and stored during the hosting of our anchors, making them as accurate as possible. After the selection of the desired monument by the user, a minimal round pointer indicates to the user in which direction they should look and go, in order for the camera to detect the already mapped environment and superimpose their monument. The pointer will always be pointing towards the destination coordinates, no matter the device's orientation, giving the user the ability to freely roam through the site, and explore their surroundings in their own pace.

This kind of solution eliminates the need for us to have specialized signs in the Archaeological site and prevents overcrowding, as visitors will not have to go to a specific spot in the area and can have a monument augmented from different angles.

3.5 Hardware Requirements

In order for our application to run successfully on the user's device and to have all of its features available, there are some specific hardware requirements that the device must meet based on the developing platform that we have chosen and the utilized SDKs and technologies that are included. These requirements are the following:

- The device should be equipped with GPS, compass and a gyroscope.
- The device must have a back camera.
- The device must be able to maintain a stable cellular internet connection.
- The device should run Android 7.0+/iOS 11+ respectively.
- The device must be supported by ARCore/ARKit SDKs (see ARCore/ARKit Supported Devices [16] for more information).

Chapter 4

Use Cases & Experience

4.1 Introduction

Continuing on, we will present the different use cases and personas of our application, providing a detailed look at the average user experience and a better understanding of the various UI elements and their uses. Both the use cases and the personas that will be presented will mainly focus on our two distinct user types, that of the **host** and the **guest**.

We will go through every screen of our application that is available to the user and elaborate on the different functionalities that they provide. Depending on the type of user, each has different capabilities and access to their own UI elements.

4.2 Login Screen

The user upon entering our application is greeted with the Login screen and has the following available interactions. The main purpose of this screen is to differentiate the user and establish whether they will be using the application as a **host** or a **guest** of the exhibition.

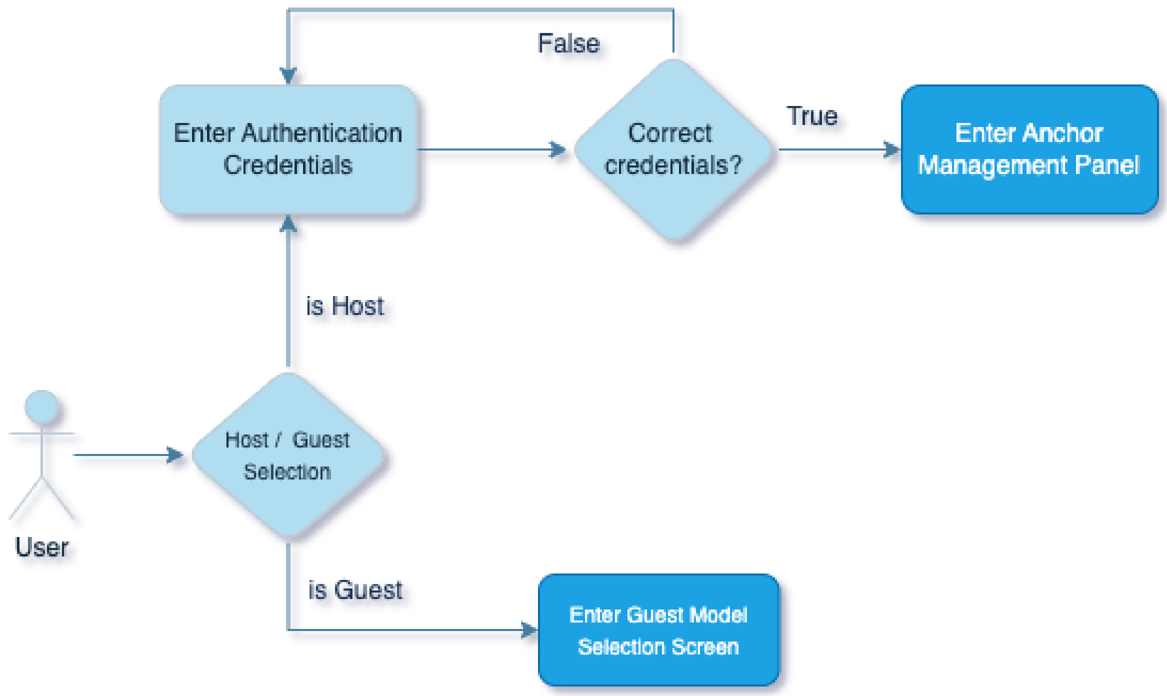

False
Enter Authentication Credentials
Correct credentials?
True
Enter Anchor Management Panel
is Host
Host / Guest Selection
User
is Guest
Enter Guest Model Selection Screen

Figure 4.1: Initial Use case for Login

- **Login as Host**

This option is selected by the host of the exhibition and requires knowledge of the unique authentication credentials. This was considered essential for the application, as the host user has access to the entire database of monuments. After authentication, the user is presented with the Anchor Management panel.

- **Login as Guest**

By selecting this option, an anonymous user account is automatically created and assigned to your device giving you access to the Guest Model Selection panel.

4.3 Host User Type

4.3.1 Anchor Management Panel

Upon signing into the application as a **host** the user is presented with the following available actions, which are depicted in the following use case.

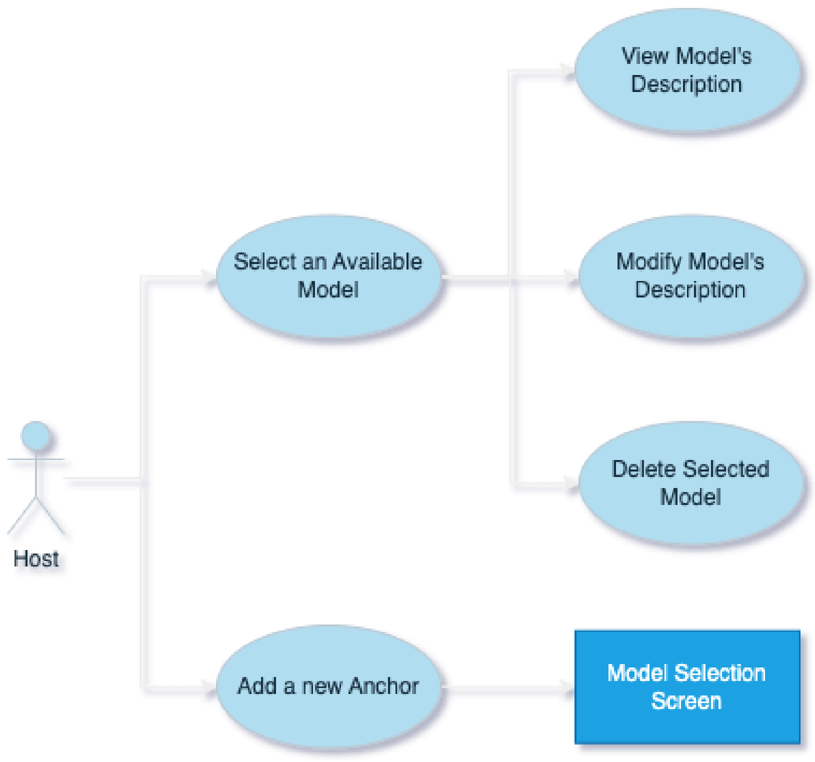


Figure 4.2: Host Use Case for Anchor Management Panel

- **Available Model Selection**

  The host can interact with the on-screen drop-down menu to preview the already submitted models that are available in our database. This menu is refreshed automatically when a change is made to ensure that the host always has access to the latest updates on the uploaded models.

- **View Model's Description**

  After selecting one of the available models, the host is presented with the submitted description that accompanies that exact model. This information is the same as the one presented to the guest during their AR experience, so it is important to keep it accurate and packed with useful information. For newly added models, there will be null placeholder description.

- **Modify Model's Description**

In order to ensure the quality and accuracy of each model's description, the host has the ability to make any needed modifications to the already submitted text and, by pressing the "Commit" button, have them be uploaded to our database and replace the previous version of the model's description. It is important to note that any changes made during this stage cannot be reversed.

- **Delete Selected Model**

By pressing the button "Remove Selected Anchor" the host can have the selected model deleted from our database along with any information accompanying it. This cannot be reversed and should be reserved as an option only when sure that the model is no longer needed or has been faulty submitted/corrupted.

- **Add a new Anchor**

Finally, the "Add new Anchor" button presents the host with the Model Selection screen, which is the next step in order for a new model to be added in the database.

- **Back Button**

The use of the "Back" button returns the host back to the Login screen and signs them out of the application.

4.3.2 Model Selection Screen

In this following screen the host is presented with the available 3D models that they can attach to their new anchor. Their use case for this section is quite simple and straightforward.

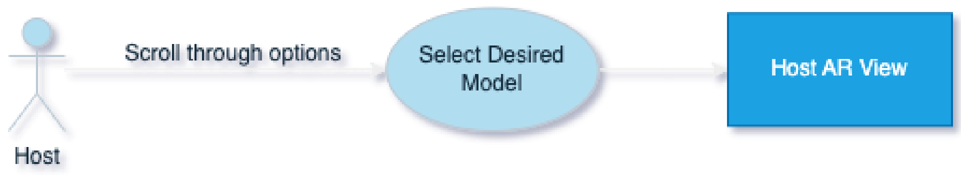


Figure 4.3: Host Use Case for Model Selection Screen

The host can scroll through the available 3D models and select the one that they would like to attach to the new anchor. The selected model will be permanently attached to that exact anchor and cannot be changed later. After selecting the desired model and confirming it with the button "Attach Selected Model", the host will be presented with the final screen of our application for this user.

4.3.3 Host AR View

The interactions depicted below are available to the host after they have entered their AR view and have selected the "Add New Anchor" option. This final screen has a variety of functionalities in terms of correctly positioning the augmented monument in the surrounding environment and making sure that it is on the correct scale and distance from the user. All changes made here will be permanent and will always be attached to the corresponding anchor.

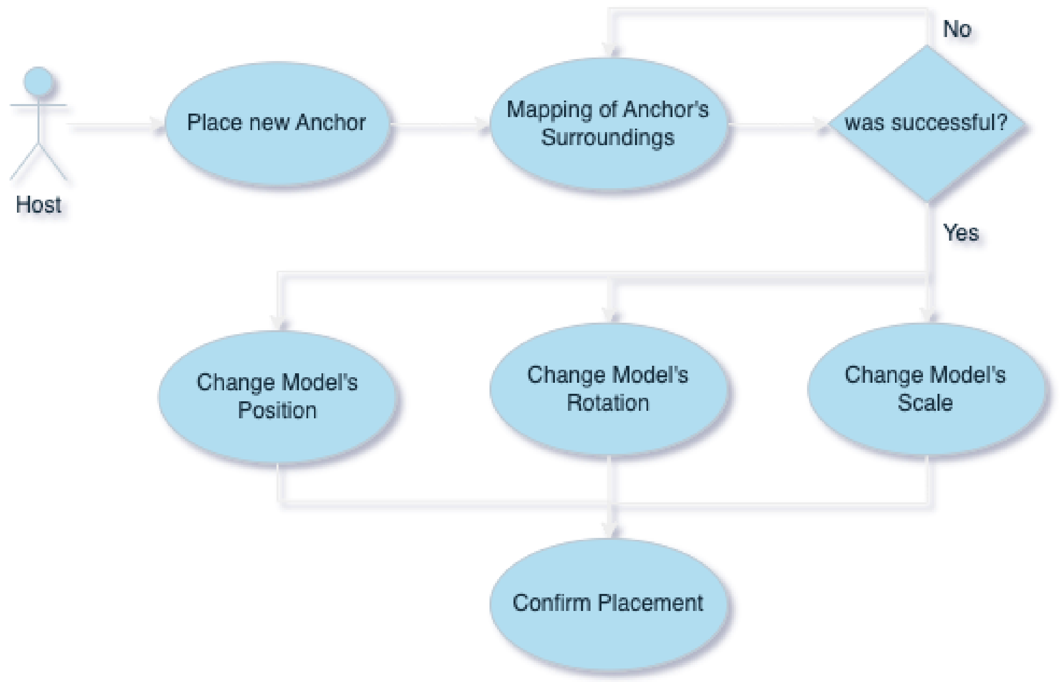


Figure 4.4: Use Case for Host's AR View

- **Placing a new Anchor**

Upon entering the Host AR view, the user is asked to point and scan their surroundings using the camera on their device. After pointing the camera towards the desired anchor placement, the host can tap the screen once, which will instantiate their new anchor point.

- **Mapping of Anchor's surroundings**

After placing the desired anchor point, the user will be asked to rotate around it and have his device map its surroundings. Visual indicators will help the user during this process and provide feedback on the mapping status. This step should be performed slowly in order to ensure that the device accurately captures the maximum amount of detail so that it can later correctly re-identify the location.

- **Model placement**

Having finished with the anchor initialization, the host will be presented with their selected 3D model on top of the previous anchor. The anchor's placeholder model will disappear and the host will be able to focus on correctly placing their augmented model at the desired spot. In order to aid them in this process, the application has a number of functionalities that are in their disposal and will be listed here.

1. Moving the model:
   The augmented model can be moved freely along all the axes by having the host tap on it and drag it to the desired placement. It is recommended for the host to move around the model through this process in order to ensure that the desired depth has been achieved.

2. Rotating the model:
   With 2-finger clockwise or anti-clockwise swipes, the host is able to rotate the augmented model around its own axis and have the model face to the correct spot.

3. Scaling the model:
   Again with 2-finger pinch actions, the on-screen model can easily be scaled, without it moving from its set position.

4. Locking Y axis:
   In order to ensure that the augmented model won't float in the air and will be flush with the ground, the button ”Lock Y axis” can be pressed that will lock this specific axis and any further change will no longer affect its placement height.

5. Hide Model's Base:
   On some of the models, a rudimentary base has been implemented, so that the model's facades can better be depicted. The host can have this base disappear by enabling the ”Hide Model's Base” toggle option.

6. Precise positional changes:
   With the aid of the on-screed D-pad the host can make precise changes in the model's position and accurately place the model on top of the desired location.

- **Confirm changes and Submit**

Finally, by pressing the on-screen button "Confirm" the host will finalize any changes made to the position, rotation and scale of the model and will have them uploaded to our database. These changes are final after this step and can no longer be edited.

4.4 Guest User Type

4.4.1 Guest Model Selection panel

Upon signing into the application as a **guest** the user is presented with a Model selection panel and the following available actions, which are depicted in the following use case.

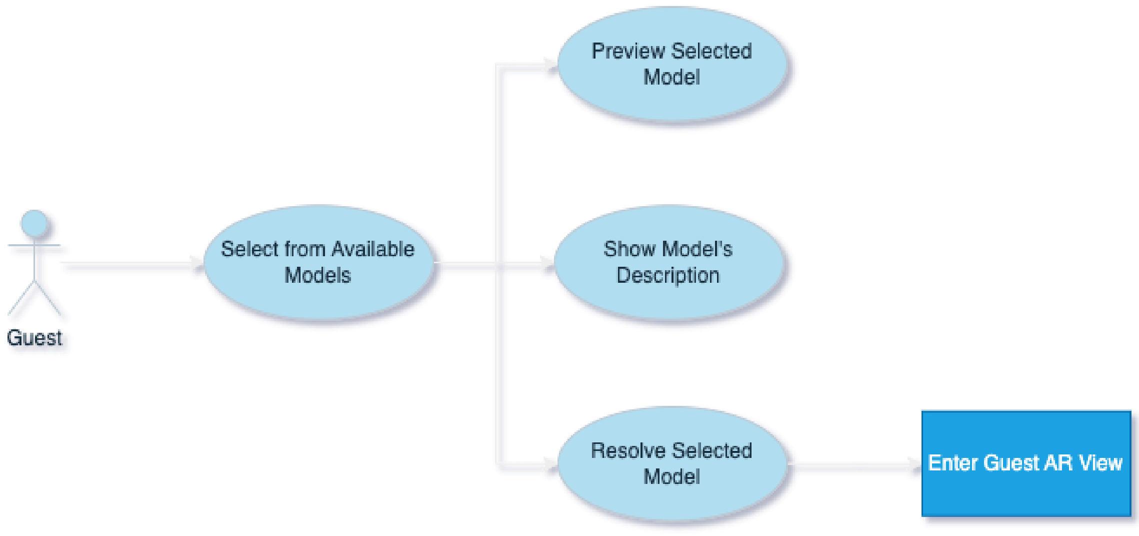


Figure 4.5: Use Case for Guest's Model Selection panel

- **Select from Available models**

  The guest is presented with an expandable drop-down menu, which will list all the available models that are in our database and have been registered by our hosts. This list is automatically updated each time any change is made to the database and always shows up-to-date data. The drop-down menu is also scroll-able, so that it does take as much screen space and keeps our UI minimal and user friendly.

- **Preview Selected Model**

  Once the user has selected their preferred choice, a preview of its 3D model will be shown to them on the scree. This helps in order for them to better understand which exact model we are referring to and give them a rough idea of what we will be presenting to them. This preview changes depending on the model they have selected and has a smooth animation for a better user experience.

- **Show Model's Description**

  Right by the side of the model's preview, the user can press the "i" button, which will instantiate a text window that will include relevant information regarding the selected model. This information is the one that the host has submitted in their initialization process and is kept always up-to-date with our database. This pop-up can easily be closed by an "x" button, so that the user can continue on.

- **Resolve Selected Model**

  Finally, once the guest has read the available information and is sure about his selection, the "Resolve Selected Anchor" button can be pressed and have the guest be redirected into the Guest AR view which will contain the rest of their Augmented experience.

4.4.2 Guest AR View

The interactions depicted below are available to the guest after they have entered their AR view and have selected to resolve one of the available models. This final screen has a variety of functionalities and options that will improve the guest's experience and help him through the process.

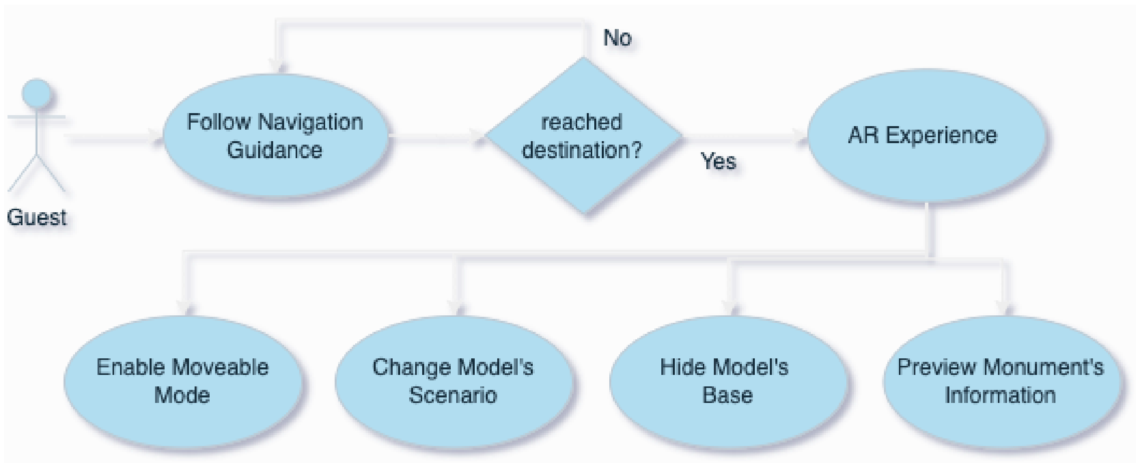


Figure 4.6: Use Case for Guest's AR View

- **Follow Navigation Guidance**

Upon entering the AR experience the guest is presented with a camera view and instructions on what to do next. There is a distinct on-screen indicator that points towards their selected monument and guides the guest through the archaeological site. The indicator keeps track of the device's orientation and rotates around the screen, working as a compass for the guest to follow.

Once the guest has reached the desired destination the guidance indicator disappears and the guest is prompted to scan the surrounding area to start the AR experience.

- **AR Experience Functionalities**

After successfully scanning the registered environment, the application s able to identify the placed anchor and have the 3D model augmented to the user. All other UI elements will be disabled at this moment, enabling the guest to focus on the augmented model and make us of any of the following functionalities:

1. Change Model's Scenario:
   The guest can press any of the on-screen arrows and change the augmented scenario of the model. This change will be immediate enabling the guest to scroll freely between them and observe the changes in real-time.

2. Hide Model's Base:
   On some of the models, a rudimentary base has been implemented, so that the model's facades can better be depicted. The guest can have this base disappear by enabling the "Hide Model's Base" toggle option, in order to preview the monuments facades, as if their are augmented on top of the actual monument.

3. Preview Information:
   The guest can also click the information "i" button and be presented with all the relevant monument info that the host has chosen for this particular model. This information will include a brief history about the monument, its significance as well as some of the key reasons regarding their archaeological uncertainty between the different scenarios.

4. Toggle Move-able Mode:
   A sandbox mode is also available to the guest upon enabling the "Movable mode" toggle. While in this mode, the guest can freely move the augmented model, bring it closer to him and scale or rotate it according to their needs.

   The "Reset" button, upon pressed, discards any changes made to the position of the model and returns it back to its initial placement.

4.5 End User Experience

In this section, we will go through a detailed overview of the end user experience and the overall flow of our application. We will present it from both the **host's** and the **guest's** point of view, emphasizing on their individual available functionalities.

4.5.1 Host's User Experience

4.5.1.1 Initial Login Screen

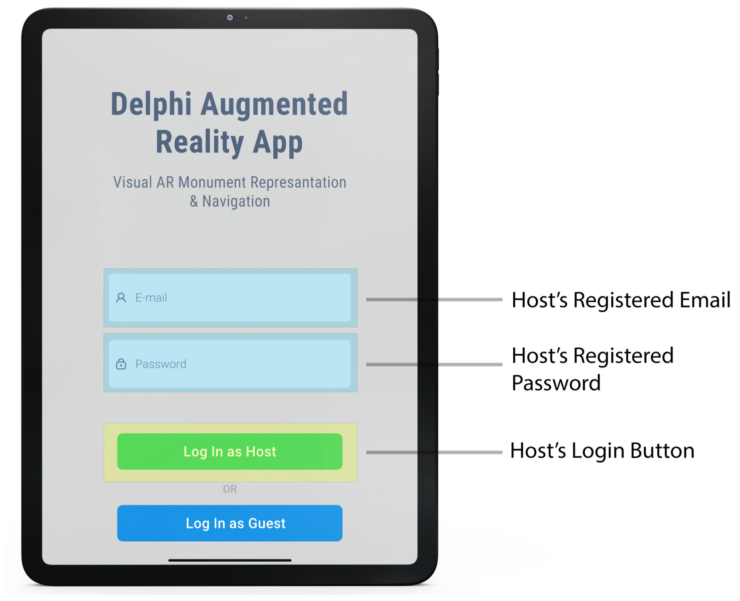

Delphi Augmented Reality App
Visual AR Monument Represantation & Navigation
E-mail
Password
Log In as Host
OR
Log In as Guest
Host's Registered Email
Host's Registered Password
Host's Login Button

Figure 4.7: Login Screen for Host

The host upon launching the Delphi AR application is met with the above Login screen. From here he is instructed to enter his unique credentials, which are bound to the host user in our database and enable him to have unrestricted access and commit any required changes. These credentials are only available to the different hosts and should not be shared with anyone as they pose a critical risk to the database and can lead to none reversible changes.

After filling in their unique email and password, the host must press the "Log In as Host" button. Their credentials will pass through an authentication process and give or deny access to the application based on their credibility.

4.5.1.2 Anchor Management

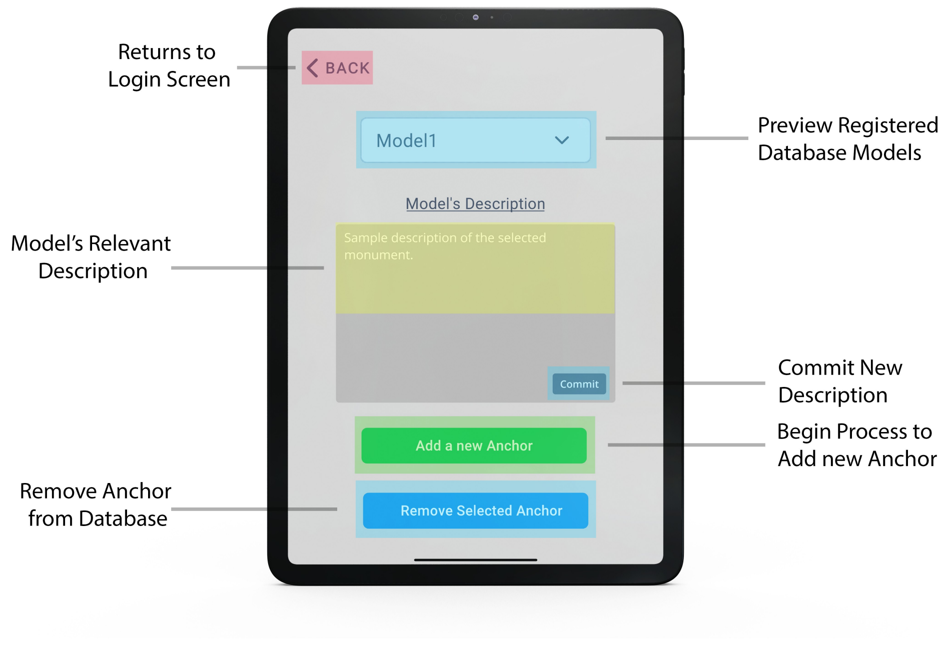

Returns to Login Screen
BACK
Model1
Preview Registered Database Models
Model's Description
Model's Relevant Description
Sample description of the selected monument.
Commit
Commit New Description
Add a new Anchor
Begin Process to Add new Anchor
Remove Anchor from Database
Remove Selected Anchor

Figure 4.8: Anchor Management Screen

While on this screen the host has unrestricted access to the database and can freely add or remove models for it. On the top of the screen, there is a drop-down menu and automatically refreshes and keep an up-to-date list of all the available registered anchors in our database.

When selecting one of them, the text box in the middle of the screen will auto-fill with the submitted description for that specific model and the host will be able to make any necessary changes to it, or replace it all together. When satisfied with the potential changes, the host can commit them to the database by pressing the "Commit" button right below the text-box.

Furthermore, the host has the ability to delete the selected Anchor from the database with the use of the "Remove Selected Anchor" button, but it is important to note, that this will be irreversible and should be done with caution. In order to prevent accidental deletions, we have implemented a pop-up window that will should up upon clicking the button and will ask for confirmation regarding the ongoing deletion.

Finally, by pressing the "Add a new Anchor" button, the host will begin the process of registering a new Anchor to the database and will be presented with the Model Selection Screen.

4.5.1.3 Model Selection

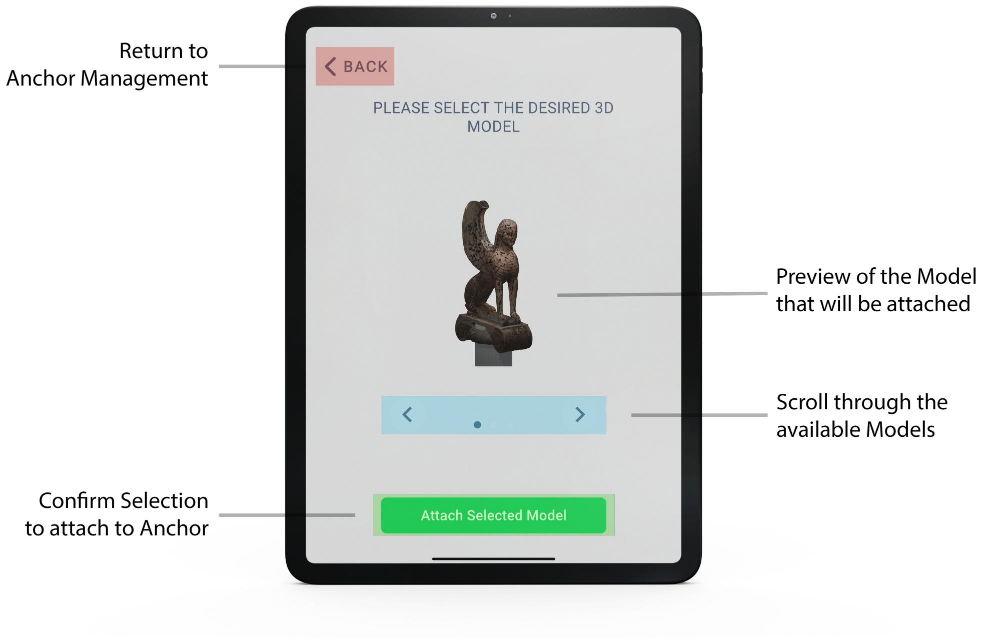

Return to Anchor Management
BACK
PLEASE SELECT THE DESIRED 3D MODEL
Preview of the Model that will be attached
Scroll through the available Models
Confirm Selection to attach to Anchor
Attach Selected Model

Figure 4.9: Model Selection Screen

The first step of creating a new Anchor is to select on a Model that will be attached to it and will eventually be augmented to the user. In this screen the host can preview through the few different available models that are included in our database and select the one that corresponds to their new anchor. While scrolling through the different option, a preview of the model will also be presented to him to ensure that the right one has been selected.

Once the selection has been made, the host can commit it by pressing the "Attach Selected Model" button and continue on with the placement process through its own AR view.

4.5.1.4 AR View

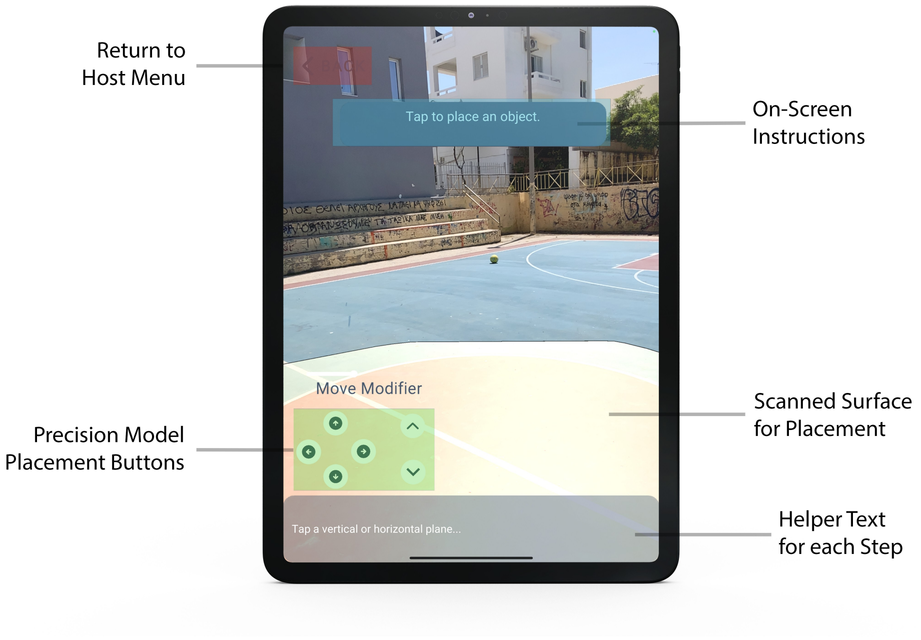

Return to Host Menu
On-Screen Instructions
Tap to place an object.
Move Modifier
Scanned Surface for Placement
Precision Model Placement Buttons
Tap a vertical or horizontal plane...
Helper Text for each Step

Figure 4.10: Host AR - Environment Scanning

Upon entering their AR view, the host is presented with the screen of Figure 4.10. On-screen instructions will guide the host through the placement process for each step, providing a fluent experience.

More specifically, the host will have to scan the surrounding environment around their desired location until it is adequately depicted on their screen. Once the environment has been scanned, the host will be asked to place their anchor by tapping at the exact location on screen.

It is important to note here that the host should try and find a surface with as many unique identifiers as possible. These will be depicted to him by yellow dots on their screen, indicating the best available placement spots.

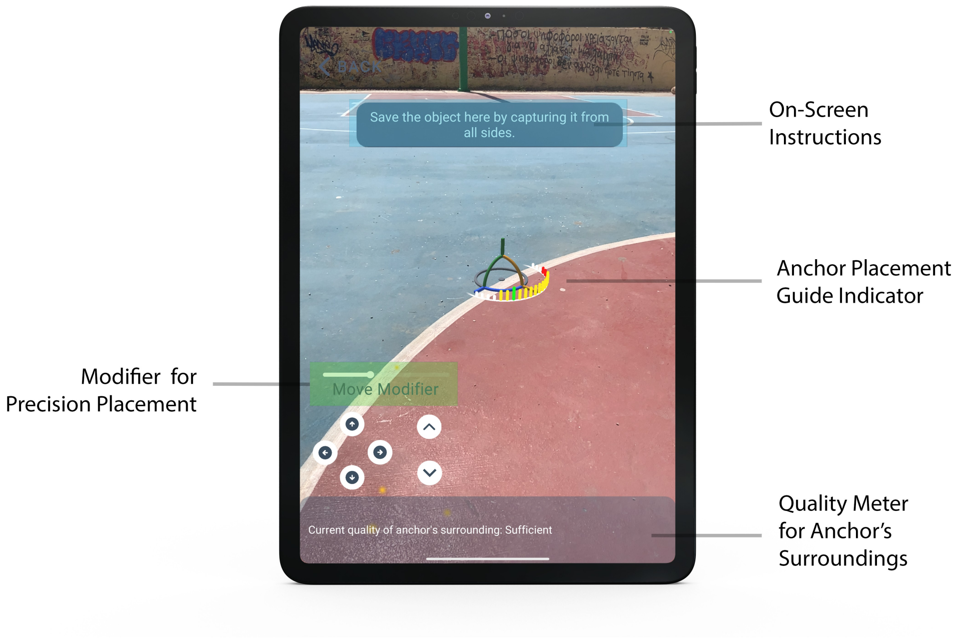

On-Screen Instructions
Anchor Placement Guide Indicator
Modifier for Precision Placement
Quality Meter for Anchor's Surroundings
Save the object here by capturing it from all sides.
Move Modifier
Current quality of anchor's surrounding: Sufficient

Figure 4.11: Host AR - Anchor Placement

After tapping on their screen, an anchor indicator will be instantiated, as shown in the above figure. This model has specific indicators surrounding it that indicate the anchors mapping.

In this step, the host is asked to rotate around the placed anchor and capture it from as many sides as possible. Through this process, the surrounding indicators will change colors from red to green, signaling to the host of the current mapping quality of the anchor. The host should repeat rotating around the anchor until all of its indicators are green and the mapping process has finished.

It is advised to perform this process slowly and methodically so that we can achieve the maximum possible effective mapping, that will greatly help our application to identify this location in the future and place the augmented model correctly.

Finally, once the mapping process has successfully finished, the anchor indicator will disappear from screen and will be replaced by the host's selected model. A pop-up win-

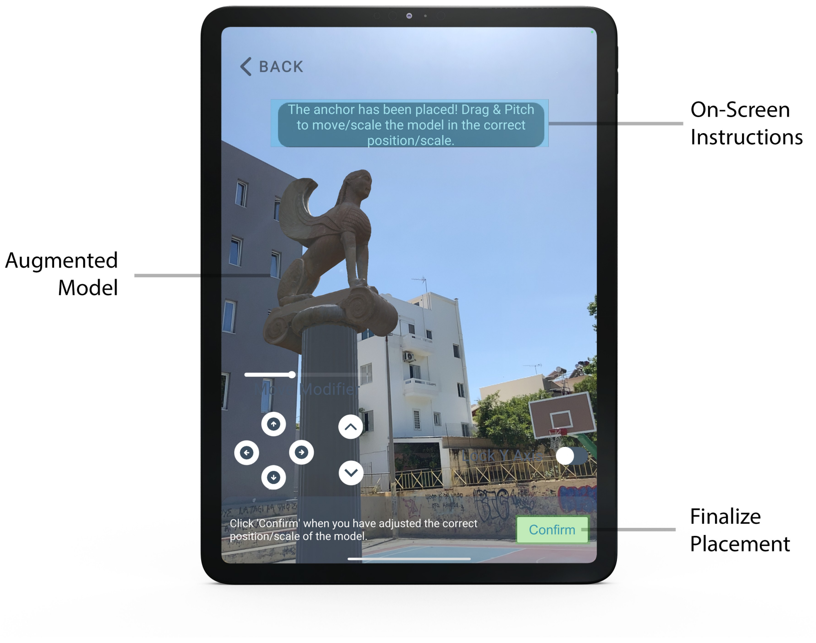

BACK
The anchor has been placed! Drag & Pitch to move/scale the model in the correct position/scale.
On-Screen Instructions
Augmented Model
Modifier
Click 'Confirm' when you have adjusted the correct position/scale of the model.
Confirm
Finalize Placement

Figure 4.12: Augmented Model Placement

dow will also appear, confirming the anchor placement and asking from the host to type the desired name for this specific model. This submitted name will also be the one displayed to any guests and should be accurate and representative of the selected model.

Continuing on, the host will be instructed to place their model in the exact desired location. Movement of the model can be achieved by dragging on the screen. Two finger swipes will enable scaling of the model while clockwise or counter-clockwise rotations will rotate it around each axis.

In order to achieve an accurate placement, we have also implemented on-screen buttons that can be utilized to fine tune any changes. By toggling the "Lock Y axis" the model will be locked to its current y height and will not be influenced from any further lateral changes. In addition, the "Move Modifier" scroll bar, changes how much the model will be moved by our placement controls, offering a wide range of movement.

Once finished, the host must press the "Confirm button" and finalize the changes. This will mark the end of this process and all changes will automatically be uploaded to our database.

4.5.2 Guest's User Experience

4.5.2.1 Initial Login Screen

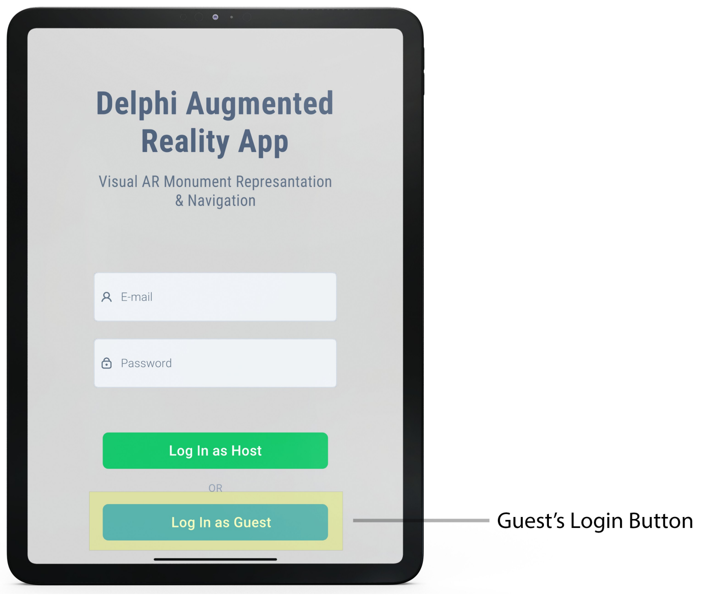

Delphi Augmented Reality App
Visual AR Monument Represantation & Navigation
E-mail
Password
Log In as Host
OR
Log In as Guest
Guest's Login Button

Figure 4.13: Login Screen for Guest User

The guest upon launching the Delphi AR application is met with the above Login screen. In order to simplify their experience the guest only has to press the "Log In as Guest" button. Upon clicked, an anonymous unique account will be created for him, with the required privileges to access our database.

This account is temporary and will be deleted once the application is closed. The only requirement from the guest is a stable Internet connection to ensure our database synchronisation.

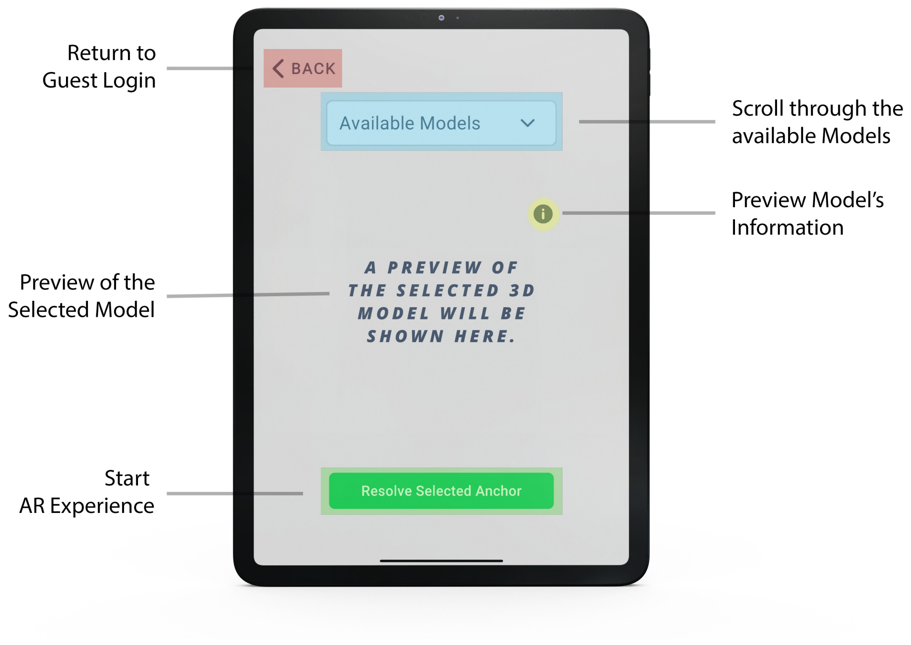


Figure 4.14: Guest's Monument Selection Screen

4.5.2.2 Monument Selection

After logging in, the guest will be met with the above Monument Selection screen. The top drop-down menu will have an always updated list of our available database models. Once the desired one has been selected, a preview of the 3D model will be presented to them in the area below to ensure a correct selection.

The guest can also click on the "i" button, which will make a text windows pop-up, including all the relevant information regarding the selected monument that the host has submitted in our database. This will give a detailed overview to the guest about their monument and inform them about its archaeological uncertainty and potential different scenarios.

By clicking the "Resolve Selected Anchor" button, the guest is confirming his selection and will initiate their AR experience.

4.5.2.3 AR Experience

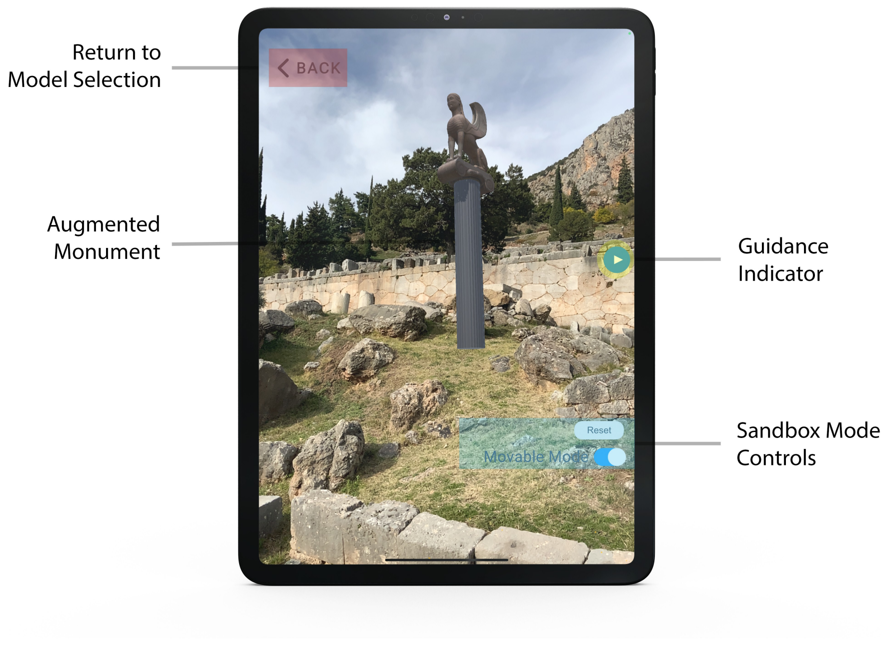


Figure 4.15: Guest's AR View

Inside their AR view, the guest is met with the above camera view, presented in Figure 4.15. As a starting point, the guest will be instructed to follow the blue guidance indicator that is located on the side of their screen. This indicator keeps track of the location of the desired augmented monument and always point towards it. The guest can utilize it as a compass and follow it until he has reached the required destination.

From there, the guest's camera will identify the scanned location and correctly position the augmented monument in its correct position. Once this identification has been completed the device's UI will for the most part disappear, leaving the guest with a minimal and simplistic camera view, enabling them to focus on the superimposed monument and aid in their immersion.

By clicking on the on-screen "i" button, the guest can view again the monument's information, alongside the augmented monument and observe the different scenarios. The arrow buttons on the bottom of their screen, will rotate through the monument's different scenarios in real time, making it easier for the guest to identify any changes or variations that they have between them.

Finally, a sandbox mode has also been implemented, which can be enabled by clicking the "Movable Mode" toggle. While enabled, the guest will be able to transform the augmented model, changing its placement, scale or rotation freely. This feature aims to help the guest better visualize the monument and better focus on any specific part that they might want. The "Reset" button, upon clicked, will revert any positional changes and return the monument to its original correct location and scale.

Chapter 5

Implementation

5.1 Introduction

In this section, we will be analysing specific key components of our application and how they were implemented. We will focus on all the different developing steps of our implementation and provide explanation of our thinking process and the different challenges that we faced during it.

In particular, code snippets will be provided and explained in order for the reader to get a better grasp of the technical side of our application and be able to understand the various aspects of it and our back-end development.

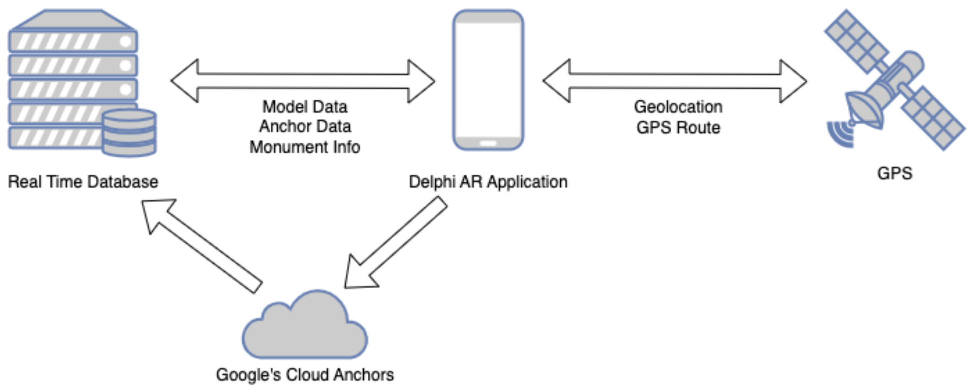


Figure 5.1: Application's Top Level Structure

5.2 Unity Scene Structure

We will start by presenting the basic structure of our application inside Unity, how we handled scene management and analyse the various components inside our main scene. In general, in order to avoid loading delays and optimize the user experience we have opted to avoid the use of different Scenes and instead have individual screens inside a single main Scene that will be toggled depending on the user's selections.

A detailed overview of our scene composition can be found in the Figure below.

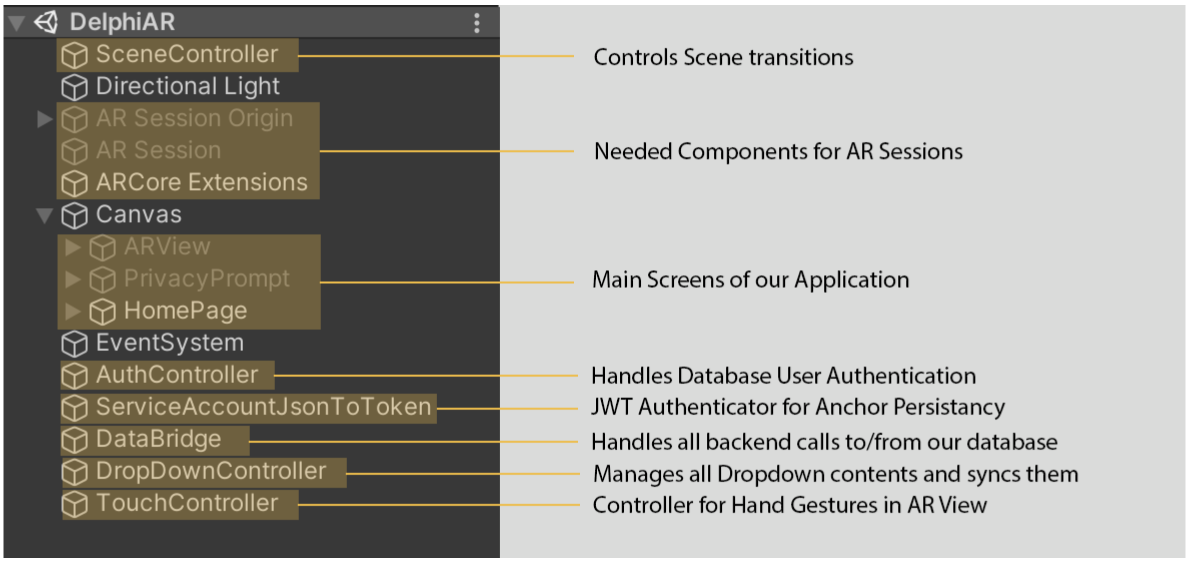

DelphiAR
SceneController
Directional Light
AR Session Origin
AR Session
ARCore Extensions
Canvas
ARView
PrivacyPrompt
HomePage
EventSystem
AuthController
ServiceAccountJsonToToken
DataBridge
DropDownController
TouchController
Controls Scene transitions
Needed Components for AR Sessions
Main Screens of our Application
Handles Database User Authentication
JWT Authenticator for Anchor Persistancy
Handles all backend calls to/from our database
Manages all Dropdown contents and syncs them
Controller for Hand Gestures in AR View

Figure 5.2: Application's Scene Structure

A list of the Scene's game-objects can also be found below, along with their uses:

1. SceneController:
   This controller is responsible for handling all of our Scene transitions. It keeps track of the current state of our application and enables the required components, depending on which screen the user chooses to go to.

2. ARSession/ARCore Objects:
   These components are required by ARKit and ARCore Foundation. These processes encompass retrieving data from the device's motion sensing hardware, managing the

device's integrated camera, and conducting image analysis on the camera images obtained. With their help the session establishes a coherent mapping between the physical environment occupied by the device and a virtual space in which AR content is modeled.

3. Canvas:
   Inside the main Canvas one can find all of our applications scenes. ARView will manage both the host's and the guest's AR Experiences, while HomePage include the view different UI screens that are analysed in the previous chapters of our thesis.

4. AuthController:
   This controller manages all of our authentication requirements for our database. Each time a user logs in or creates a guest account, this controller will authenticate them to the database and allow or deny access to our app.

5. ServiceAccountJsonToToken:
   In order to have persistent CloudAnchors, we need to pass them through a JWT authentication which involves the use of JSON Web Tokens, that are compact, URL-safe tokens with verifiable claims. These tokens are digitally signed using a secret key or a public/private key pair. This process will be greater analysed further in this Chapter.

6. DataBridge:
   All database back-end calls are managed by this component. It keeps all the updated information regarding a selected anchor and ensures that we are utilizing the latest version of our database's data.

7. DropDownController:
   Throughout our application scenes we are utilizing a number of drop-down menus that need to be kept up-to-date. With this component we can manage them all at the same time and ensure their consistency throughout the different scenes.

8. TouchController:
   This component is responsible for any touch-based gesture inside our ARView. It keeps track of the user's finger screen taps and translates them accordingly so that they can be mapped to a specific function.

The implementation of all of the above components will be further analysed in detail in the later parts of this Section.

5.3 Real-time Database Selection & Integration

In order to initialize a Real-time database for our application the Firebase configuration file, google-services.json, must be downloaded and added to the Unity project's Assets folder. This file contains crucial information that allows the Unity application to communicate with the Firebase services. After importing the Firebase SDK, a connection between our application and the database has to be established and is done by the following code, which is included in the DataBridge component of our scene.

```
DBReference = FirebaseDatabase.DefaultInstance.RootReference;
DBReference.ValueChanged += HandleValueChanged;

private void HandleValueChanged(object sender, ValueChangedEventArgs args)
    {
        if (args.DatabaseError != null)
        {
            Debug.LogError(args.DatabaseError.Message);
            return;
        }
        LoadAvailableAnchors();
    }
```

Listing 5.1: Real-Time Database Initialization

By assigning the **RootReference** to the **DBReference** variable we have access to the entire databse's structure and can freely perform operations on its contents.

In a real-time database, it is crucial to monitor and handle value changes as they occur. The Firebase SDK provides event-driven mechanisms to accomplish this. In the presented above code snippet, the **ValueChanged** event is subscribed to the **HandleValueChanged** method, enabling the application to respond to changes.

More specifically. we have implemented the function **LoadAvailableAnchors()** that automatically refreshes our list of available registered anchors asynchronously, ensuring real-time data synchronization. Within the same script, a number of different functions have also been implemented to handle the different variables of our anchors, save and load them on demand and modify them on request.

5.3.0.1 User Authentication & Guest Account Creation

The host authentication for our database is solely handled by the AuthController component through the following code snippet:

```
FirebaseAuth.DefaultInstance.SignInWithEmailAndPasswordAsync(emailInput.text,
passwordInput.text).ContinueWith((task => {
if (task.IsFaulted)
{
    Firebase.FirebaseException e =
    task.Exception.Flatten().InnerExceptions[0] as Firebase.FirebaseException;
    GetErrorMessage((AuthError)e.ErrorCode);
    return;
}
if (task.IsCompleted)
{
    hostLoggedIn = true;
    resolving = false;
}
}));
```

Listing 5.2: Database User Authentication

A specific email and password combination has been registered in our database and will be given to any potential host of our application. These credentials will be used through the host login process and are checked by the **SignInWithEmailAndPasswordAsync** method provided by FirebaseAuth.DefaultInstance. In case an error occurs during the authentication process, error handling logic has been implemented to flag the event and notify the user accordingly. Upon successful completion, the **hostLoggedIn** variable is set to true, indicating that the user has been successfully authenticated.

As far as the guest user is concerned a similar process takes place that instantly creates a unique anonymous account that is linked to the current user's session and will be deleted upon closing the application.

5.4 Cloud Anchor Implementation

The primary basis of our application revolves around Google's Cloud Anchors, which was implemented to provide us with a way to correctly register our augmented monuments. Our **ARViewManager** class is directly responsible for handling the hosting and resolving of our cloud anchors.

```
private void PerformHitTest(Vector2 touchPos){
List<ARRaycastHit> hitResults = new List<ARRaycastHit>();
Con.RaycastManager.Raycast(
    touchPos, hitResults, TrackableType.PlaneWithinPolygon);

// If there was an anchor placed, then instantiate the corresponding object.
var planeType = PlaneAlignment.HorizontalUp;
if (hitResults.Count > 0){
    ARPlane plane = Con.PlaneManager.GetPlane(hitResults[0].trackableId);

    planeType = plane.alignment;
    var hitPose = hitResults[0].pose;
    _anchor = Con.AnchorManager.AttachAnchor(plane, hitPose);
    }

if (_anchor != null){
    cloudAnchorModel = Instantiate(CloudAnchorPrefab, _anchor.transform);

    // Attach map quality indicator to this anchor.
    var indicatorGO =
        Instantiate(MapQualityIndicatorPrefab, _anchor.transform);
        _qualityIndicator = indicatorGO.GetComponent<MapQualityIndicator>();
        _qualityIndicator.DrawIndicator(planeType, Controller.MainCamera);
}
```

Listing 5.3: New Anchor Registration

The registration of a new anchor is handled by our **PerformHitTest()** function, which code snippet is presented above. When a user interacts with the screen, a hit test is performed to identify a suitable surface for placing the anchor. This is achieved through the ARRaycastManager, which searches for ARPlane surfaces within the camera's field of view. Once a suitable surface is found, an ARAnchor is attached to it, and a pawn object is instantiated to visually represent the anchor's location. Additionally, a map quality indicator is associated with the anchor, providing feedback on the mapping quality of it.

```
private void HostingCloudAnchorAsync(){
FeatureMapQuality quality =
Controller.AnchorManager.EstimateFeatureMapQualityForHosting(GetCameraPose());
_qualityIndicator.UpdateQualityState(qualityState);

authToken = jwtToken.authToken;
Controller.AnchorManager.SetAuthToken(authToken);
ARCloudAnchor cloudAnchor = Con.AnchorManager.HostCloudAnchor(_anchor, 365);
if (cloudAnchor == null){
    OnAnchorHostedFinished(false);
}else{
     _pendingCloudAnchors.Add(cloudAnchor);
    switch (modelSelect.index+1) {
         case 3:
           hostedModel = Instantiate(AnchorModel3, cloudAnchor.transform);
           break;
        case 2:
           hostedModel = Instantiate(AnchorModel2, cloudAnchor.transform);
           break;
        case 1:
           hostedModel = Instantiate(AnchorModel1, cloudAnchor.transform);
           break;
        default:
           hostedModel = Instantiate(CloudAnchorPrefab, cloudAnchor.transform);
           break;}
```

Listing 5.4: Saving a Registered Anchor

Saving the anchor involves the **HostingCloudAnchorAsync()** method. This method estimates the mapping quality of the anchor's surroundings using the **EstimateFeatureMapQualityForHosting()** function, which assesses the environment's suitability for cloud anchor hosting. Several checks are performed to ensure the anchor's position, distance, and mapping quality meet the requirements for hosting. If these conditions are satisfied, the ARCloudAnchor is created using **HostCloudAnchor()** and added to the pendingCloudAnchors list. Furthermore, the hosted cloud anchor is associated with a specific model based on the selected model ID, and the anchor's information, including its name, ID, position, rotation, scale, and description, is saved to a database using the **bridge.SaveData()** method.

```
private void ResolvingCloudAnchors(){
    string selectedCloudId = bridge.resolveId;
    Controller.ResolvingSet.Add(selectedCloudId);

    authToken = jwtToken.authToken;
    Controller.AnchorManager.SetAuthToken(authToken);

    Debug.LogFormat("Attempting to resolve the selected Cloud Anchor: {2}",
     Controller.ResolvingSet.Count,
        string.Join(",", new List<string>(Controller.ResolvingSet).ToArray()),
        selectedCloudId);

    foreach (string cloudId in Controller.ResolvingSet){
        ARCloudAnchor cloudAnchor =
             Controller.AnchorManager.ResolveCloudAnchorId(selectedCloudId);
        if (cloudAnchor == null){
            OnAnchorResolvedFinished(false, selectedCloudId);
        }else{
             _pendingCloudAnchors.Add(cloudAnchor);}
        }
    Controller.ResolvingSet.Clear();
}
```

Listing 5.5: Resolve a Registered Anchor

To resolve a registered cloud anchor the process begins with the user selecting a specific cloud anchor to resolve. The **ResolveCloudAnchorId()** method is employed within the **ResolvingCloudAnchors()** function. This function checks if there are any pending or finished resolving tasks and ensures that the ARCore session is ready for re-solving.

Once the resolving process is initiated, we attempt to resolve the selected cloud an-chor by calling **ResolveCloudAnchorId()** on the CloudAnchorManager. This method takes the cloud anchor's ID as a parameter and returns an ARCloudAnchor if successful. The resolved anchor is then added to the pendingCloudAnchors list.

Upon successful resolution, the resolved cloud anchor is associated with a specific model based on the provided model ID. The resolved model is instantiated and posi-tioned according to the anchor's saved position, rotation, and scale information retrieved from the database.

All of the above code snippets were striped down of any logic checks and initialization processes in order to have a more clear picture of how the many code logic works.

5.5 Persistent Cloud Anchors

Moving on, we needed a way to ensure persistence for Google's Cloud Anchors. By utilizing JWT authentication and RSA encryption, the Cloud Anchors service ensures secure and continuous access to spatial data. JWT authentication provides a reliable and standardized method for authenticating clients, granting them authorized access. RSA encryption safeguards data transmission, ensuring confidentiality and integrity. The technology enables long-term access to Cloud Anchors without frequent re-authentication, facilitating uninterrupted usage. It also offers scalability, interoperability, and simplifies integration with existing systems. This authentication process is mainly handled by the JwtAuthenticator script through the following processes.

```
void Start()
    {
        json = JsonUtility.FromJson<Json>(jsonFile.text);
        prvtKey = json.private_key;
        InvokeRepeating("RefreshAuthKey", 0.1f, 3000.0f);
    }
```

Listing 5.6: JWT Authentication Initialization Process

Starting on, the script initializes our json variable by deserializing the JSON data from the private key provided by Google's Cloud Anchor service and extract the private key that will be needed for our authentication. It then sets up a repeating invoke to call the **RefreshAuthKey()** function with a repeat interval of 3000 seconds (5 minutes) to refresh the authentication key.

The **RefreshAuthKey()** function generates a new JWT (JSON Web Token) authentication token. It calls the **GenerateJWTToken()** function, passing the serviceEmail and prvtKey as parameters. The resulting JWT token is stored in the authToken variable.

Before explaining how the JWT generation is implemented, we need to have a basic understanding about the 2 main helper function that it will be utilizing.

RSA (Rivest-Shamir-Adleman) is a widely used public-key encryption algorithm. It uses two main keys: a public key for encryption and a private key for decryption. RSA parameters refer to the mathematical values used to generate the RSA keys, including modulus (n), public exponent (e), and private exponent (d). These parameters define the behavior of the RSA algorithm and enable secure communication and encryption.

RS256 (RSA with SHA-256) is an asymmetric cryptographic algorithm used for JWT token signing and verification. It mainly combines the RSA parameters for key generation and management with SHA-256 for secure hashing. In RS256, a private key is used to generate the signature, while a corresponding public key is used to verify the signature. The RS256 algorithm provides strong security and is widely supported in JWT implementations.

```
private static IJwtEncoder GetRS256JWTEncoder(RSAParameters rsaParams){
        var csp = new RSACryptoServiceProvider();
        csp.ImportParameters(rsaParams);

        var algorithm = new RS256Algorithm(csp, csp);
        var serializer = new JsonNetSerializer();
        var urlEncoder = new JwtBase64UrlEncoder();
        var encoder = new JwtEncoder(algorithm, serializer, urlEncoder);
        return encoder;}

private static RSAParameters GetRsaParameters(string rsaPrivateKey){
        var byteArray = Encoding.ASCII.GetBytes(rsaPrivateKey);
        using (var ms = new MemoryStream(byteArray)){
            using (var sr = new StreamReader(ms)){
            var pemReader = new Org.BouncyCastle.Utilities.IO.Pem.PemReader(sr);
            var pem = pemReader.ReadPemObject();
            var privateKey = PrivateKeyFactory.CreateKey(pem.Content);

            return DotNetUtilities.ToRSAParameters(privateKey as
            RsaPrivateCrtKeyParameters);}}}
```

Listing 5.7: RS256 Algorithm for Encryption

The **GetRS256JWTEncoder()** function sets up the JWT encoder with the RS256 algorithm. It creates an RSACryptoServiceProvider and imports the RSA parameters. It then creates instances of various components required by the JWT library, such as the algorithm (RS256Algorithm), serializer (JsonNetSerializer), and URL encoder (Jwt-Base64UrlEncoder). It uses these components to construct and return a JWT encoder.

The **GetRsaParameters()** function retrieves the RSA parameters from the provided RSA private key string. It converts the string to a byte array, reads it as a PEM object, and extracts the private key. It then converts the private key to RsaPrivateCrtKeyParameters and finally converts it to RSAParameters required by the RSACryptoService-Provider.

Finally the JWT token generation is handled by the following function:

```
public string GenerateJWTToken(string email, string rsaKey){
       var rsaParams = GetRsaParameters(rsaKey);
       var encoder = GetRS256JWTEncoder(rsaParams);
       var time = DateTimeOffset.Now.ToUnixTimeSeconds();
       int expiresInSecond = 3600;

       var payload = new Dictionary<string, object>{
           { "iss", email},
           { "sub", email },
           { "iat", time },
           { "exp", time + expiresInSecond },
           { "aud", "https://arcorecloudanchor.googleapis.com/" }
       };
       var header = new Dictionary<string, object>{
           { "kid", "ae83070f38a3c020a69fe37a4430aa2fac568de8" }
       };
       var token = encoder.Encode(header, payload, new byte[0]);
       return token;
    }
```

Listing 5.8: JWT Token generation function

It begins by retrieving the RSA parameters from the RSA private key using the **GetRsaParameters()** function that we have already explained.

The function sets the current time (time) as the issued at (iat) value of the token, represented as the number of seconds. It also sets the expiration time (exp) to the current time plus a predefined number of seconds (expiresInSecond), which we have set to one hour.

Next, the function constructs the payload of the JWT token, which contains a set of claims or statements about the entity being authenticated. In this case, the payload includes the issuer (iss) and subject (sub), both set to the provided email. Additionally, it includes the issued at (iat) and expiration time (exp) mentioned earlier. The aud claim is set to the specific audience, which for us is ”https://arcorecloudanchor.googleapis.com/”, indicating that the token is intended for use with the Cloud Anchors service.

The function creates a header dictionary, which contains metadata about the token. In our case, it includes a key ID (kid) field, typically used to identify the key that was used to sign the token, which in our case is hardcoded into the script. An encoder (encoder) is obtained using the RSA parameters from the **GetRS256JWTEncoder()** function.

The encoder is responsible for encoding the header and payload dictionaries into a token string using the RS256 algorithm and other necessary components.The token is generated by calling the Encode() method on the encoder. The Encode() method takes the header and payload dictionaries, along with an empty byte array as the signature, and produces the final JWT token string.

Finally, the generated token is returned as the result of the function and is used from our application to maintain persistence for our Cloud Anchors.

5.6 AR Guest Navigation

Finally, the script regarding the On-Screen indicator of our application will be explained. As previously mentioned, we opted to have a minimal arrow indicator that will guide the guest to his selected monument by keeping track of the real-word coordinated of it and constantly compares them to the user ones in order to accurately point towards the direction of the destination.

Inside our script we have declared two enums: "TargetVisibilityState" and "ArrowDir". The "TargetVisibilityState" enum represents the different possible visibility states of the target, such as whether it is visible on screen or off to the left, right, up, or down. The "ArrowDir" enum represents the direction of the arrow indicator, which is either left or right.

The "OnEnable()" function is called when the script is enabled. It checks if the authentication process is resolving, and if so, initializes the indicator and sets the target position based on the anchor latitude and longitude coordinates.

The "Init()" function initializes the indicator if it hasn't been already. It creates a new canvas and game object for the indicator, sets up the necessary components (Canvas and Image), and configures its transform properties. It also assigns the camera transform reference.

The script also defines three helper functions:

1. "isLeftOfCamera": Determines whether the target position is to the left of the camera's forward vector.

2. "isBehindCamera": Determines whether the target position is behind the camera.

3. "isVisible": Determines whether the target position is visible on screen.

The most important function of this script is the "OnRouteUpdate()" function, that is responsible for updating the indicator based on the position and visibility of the target. It uses the helper functions and camera calculations to determine the target's visibility state. If the target is visible, the indicator is hidden. Otherwise, the indicator is shown and positioned accordingly. The indicator's rotation is also adjusted based on the target's visibility state and the neutral arrow direction. The code of this function will be presented below and explained.

```
var p = cam.WorldToScreenPoint(TargetPos);

if (p.x < 0){targetVisibility = TargetVisibilityState.OffLeft;}
else if (p.x >= Screen.width){
    targetVisibility = TargetVisibilityState.OffRight;}
else if (p.y < 0){
    targetVisibility = isBehind ? TargetVisibilityState.OffUp :
                                  TargetVisibilityState.OffDown;}
else{
    targetVisibility = isBehind ? TargetVisibilityState.OffDown :
                                   TargetVisibilityState.OffUp;}

p.x = Mathf.Clamp(p.x, Margin, Screen.width - Margin);
p.y = Mathf.Clamp(p.y, Margin, Screen.height - Margin);

if (isBehind){
p.y = Screen.height - p.y;
        if (isLeft){
            p.x = Margin;
            targetVisibility = TargetVisibilityState.OffLeft;}
        else{
            p.x = Screen.width - Margin;
            targetVisibility = TargetVisibilityState.OffRight;}
        }
        indicator.position = p;

        switch (targetVisibility){
            case TargetVisibilityState.OffLeft: {...}
            case TargetVisibilityState.OffRight: {...}
            case TargetVisibilityState.OffUp:   {...}
            case TargetVisibilityState.OffDown: {...}
            default:                            {...}
```

Listing 5.9: OnRouteUpdate() function for On-Screen Indicator

Firstly, we will be defining some boolean variables (isLeft, isBehind, isFront, isRight) based on the target's position relative to the camera. These variables will be used to determine the target's direction in relation to the camera.

Next, we check if the target is visible by calling the isVisible() function. If the target is visible, it means it is within the camera's view, and there is no need to display the indicator. In this case, we hide the indicator by setting its game object to inactive (gameObject.SetActive(false)) and set the targetVisibility. The function then returns, as no further updates are necessary.

If the target is not visible, it means it is off-screen, and the indicator needs to be displayed. We then use the camera's WorldToScreenPoint() function to convert the target's position from world coordinates to screen coordinates, storing the result in the p variable.

Continuing on, we check the screen coordinates to determine the targetVisibility state (OffLeft, OffRight, OffUp, or OffDown). By compering the x and y coordinates of the screen position with the screen width and height we can determine if the target is off the left, right, top, or bottom edges of the screen.

1. If the target is off the left or right edges of the screen (OffLeft or OffRight), the function adjusts the screen position (p) to keep the indicator within the screen boundaries while accounting for the Margin distance. It clamps the x coordinate between the Margin and Screen.width - Margin values.

2. If the target is behind the camera (isBehind is true), the function flips the y coordinate by subtracting it from the screen height (Screen.height - p.y) to ensure the indicator is shown in the correct direction

After positioning the indicator, the function sets its rotation (indicator.rotation) based on the targetVisibility state and the NeutralArrowDirection. It uses a switch statement to handle different cases of targetVisibility and sets the rotation accordingly.

Chapter 6

Evaluation, Limitations & Future Work

6.1 Introduction

In this final chapter of our thesis, we will be discussing the evaluation methods that were used during and at the end of our application's development, as well as, some of the limitations that were observed and any possible future work that could be implemented to further improve our user's experience and the application's functionality.

6.2 Evaluation Method

For all of our user testing we have opted to use a think-out-loud evaluation method, a well-established usability evaluation technique that involves participants verbalizing their thoughts, opinions, and decision-making processes while interacting with our application.

During the evaluation, users were encouraged to think aloud and express their impressions, challenges, and expectations in real-time regarding our app and its functionalities. This method provides us with valuable insights into the users' experience, by gaining a deeper understanding of the users' behavior and preferences. Unlike other evaluation methods that rely solely on observation, this method encourages participants to articulate their thoughts, providing a rich source of qualitative data. They provided us with direct and immediate feedback, that greatly helped to identify usability issues, comprehension difficulties, and areas for improvement.

Additionally, through this think-out-loud method our user testers were encouraged to engage actively with our application, in real-world conditions that were either in a designated campus area or directly at the archaeological site of Delphi. This enabled us to pinpoint errors and limitation areas in real time, that proved vital for the continuous development of our implementation.

Since our application is ultimately designed for an outdoor environment, it was dimmed essential that we tested it in the exact conditions that it will be used. We thankfully had the opportunity to visit the Archaeological site of Delphi during its working hours and were able to correctly setup everything there so that passing tourists were able to use and comment on our application and its features. This process was by far the most valuable one as we received genuine comments from people that were unknown to our development process and we got an accurate estimation about the average users' experience.

This technical evaluation was mainly conducted from an iOS device, more specifically an iPad Pro 4th Gen that was shared by the users. Unfortunately, we weren't able to also test the Android version on the Delphi site, as no Android device was available at the time and we hadn't finished the development of that APK. However, in later stages of our development and after successfully porting our application for an Android environment, we were able to test it locally and ensure that it is working as expected.

6.3 Evaluation Results & Limitations

In this section we will be listing all of the major observation and comments that were pointed out by our users on regard to their experience and usability of our application. The majority of those were taken into consideration and changes have been applied to improve upon them and provide a more fluent user experience. These are the following:

1. One of the earlier comments that we received was regarding the terminology used in our drop-down menus and buttons. We had opted to use more technical terms like "Anchors" and "Mapping points", that the users weren't familiar with and

caused a bit of confusion to them. To counter that, we implemented a more basic terminology like "Available Monuments" or "Area scanning" that were easier to understand. Also, in the same spirit, we changed the on-screen instructions too, so that they are more suitable and clear to a wider audience by simplifying them to smaller easier steps with no technical terms used.

2. Some of the users also had issues correctly interacting with the augmented monuments and were unable to accurately use the touch controls in order to move our models around and correctly place them. In particular, they were struggling to comprehend the depth of the superimposed monument and pointed out that it was difficult to make precise changes to it. To counter this, an on-screen button layout was also implemented that had a configurable sensitivity slider, which enabled them to choose how precise each of their movement would be aided them in handling the AR monuments correctly.

3. In regards to the User Interface, the majority of the comments were positive and they seemed to enjoy the minimal feel of it. As per their comments, the UI was simple and to the point with the only exception being that during the Monument selection process, some were unable to differentiate the different monuments solely by their registered names. We improve upon this by adding preview images to our interface that the user can see when selecting their desired monument and be certain that he has chosen the right one.

4. In earlier iterations of our application some users expressed the need to be able to freely move the superimposed monument so that they can bring it closer to their screen and observe it through all angles. We made sure to have that implemented by offering a "Sandbox" mode that can be toggle on/off in the Guest's AR view and provides unrestricted movement of the monument through all axis, as well as, scaling and rotating possibilities.

There were however a few points that were directly impacted by limitations of the implemented technology and there was nothing that we could do to fix them in the context of our thesis. These points are the following:

1. We received a few comments by users that our On-screen navigation arrow, which guided them to their selected monument, would lose accuracy when too close to their destination and point to a false location. This was a known issue to us and it basically has to do with the GPS accuracy that mobile devices can output. As of today, most mobile phones can only accurately pinpoint the users location to approximately a 5 meter radius, but that greatly depends on sky and weather conditions, as well as the technical specifications and signal strength of the users device.

2. Another issue that we had to compensate for was that of the maximum render distance available to AR mobile devices. Due to the nature of our application, the host that would place our monuments should take into consideration the distance that the anchor will have from the desired monument location. There were cases that if the anchor was placed too far from the final monument location then parts of the said monument would clip outside the device's render distance and would disappear from the viewers perspective. In order to mitigate this, we always advised the host to place his anchor as close to the desired monument as possible and always test it before commit anything to our database.

3. Finally, the last limitation that we were able to observe was regarding the weather and environment conditions of the site. Due to the fact that we are using a Markerless approach for localizing our monuments, it is greatly affected by environmental factors. For example, if a monument was placed during a sunny bright day, its mapping would be done with these conditions under consideration and it will be difficult for the device's sensor to identify the location on another day that might be raining or be foggy. It is always recommended for our users to prefer days with good weather conditions for a more fluent user experience.

6.4 Future Work

As we conclude our thesis and building upon the findings gained through our evaluation and the users' feedback, we would like to propose a few changes and additions that would improve our developed application. Below we will mention some key areas that should be worked upon and developed further in order for our application to be more complete and provide a better experience.

AR Navigation with Augmented Path
Since one of the major issues during our user testing was in regards to our on-screen navigation arrow we believe that it should change all together. Our original idea was to implement an Augmented path that would guide the visitor to his destination with clear arrows along his route. This could be implemented but requires a detailed mapping of the Archaeological site, so that distinct paths can be defined and loaded into our application. This would negate all of our present limitations that have to do with GPS accuracy and would provide the visitors with a more immersive experience.

Database storage for Augmented Models
In order to offload some of the processing times and size of our application it would be ideal for us to store all of our utilized 3D models directly into our database and instantiate them form there when needed. This will allow us to retrieve and render them dynamically, eliminating the need for large model files to be bundled within the application itself. This approach not only makes the application lighter, requiring less storage space on the user's device, but also enables more efficient updates and maintenance of the models. This will enhance our applications scalability, reduce bandwidth consumption, and enable faster loading time, contributing to an optimized and streamlined user experience within the application.

Combination of multiple Anchors for Model Positioning
One potential way to combat the issue regarding the maximum render distance that was observed during testing and caused model clipping problems is the implementation of multiple registration anchors for the placement of the model. In particular, we should have rather than a single anchor, multiple of them surrounding the placement of the final monument, that the users device will be able to identify and make on-the-sop correc-

tions to the augmented model. This will greatly increase our model positioning, it will eliminate any possible clipping issues and will overall provide a more unified and stable experience.

Potential Tour Guide Experience

Finally, one last part that could be improved upon is that of the tour experience. Although users were satisfied with the way that we display all the monument's relevant information, this could be further enhanced by having audio recordings or an augmented virtual guide that would play at a specific part of the tour route and introduce the user to the viewed model and its registered information. This would also help with people that are visually impaired as screen text can be at times hard to read.

It is our hope that the findings and insights shared in this thesis contribute to the broader landscape of AR applications, inspiring fellow researchers and developers to continue pushing the limits of this technology.

Bibliography

[1] Azuma, R. T. (1997). A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4), 355-385. 17

[2] Feiner, S., MacIntyre, B., & Seligmann, D. (2016). Knowledge-Based Augmented Reality. Communications of the ACM, 36(4), 53–62. 15

[3] Ivan E. Sutherland, A head-mounted three dimensional display, In AFIPS '68 (Fall, part I), 1968. 16

[4] Rosenberg, L. B., & K. R. B. Bachmann. "Virtual fixtures as tools to enhance operator performance in telemanipulation tasks." Proceedings of the ASME Winter Annual Meeting Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. Vol. 55. American Society of Mechanical Engineers, 1994. 16

[5] S. Feiner, B. Macintyre, and D. Seligmann, 'Knowledge-based augmented reality', Commun. ACM, vol. 36, no. 7, pp. 53–62, Jul. 1993, doi: 10.1145/159544.159587. 16

[6] Hollerer S. Feiner, B. MacIntyre and A. Webster, A touring machine: prototyping 3d mobile augmented reality systems for exploring the urban environment, IEEE International Symposium on Wearable Computers, 1997. 17

[7] H. Kato, M. Billinghurst, and I. Poupyrev, ARToolKit Hiroshima City University 2000 18

[8] Fuchs, P., & Boulanger, C. (2011). Past, present, and future of augmented reality. *Springer.*

[9] Seichter, H., & Tscheligi, M. (2012). Augmenting the museum experience: a literature review. *Journal of Multimedia*, 7(1), 1-11.

[10] C. Manresa-Yee, A. Dang, N. Bataille, D. Siret, and D. Coutellier, ”Evaluation of the Impact of Lighting Conditions on Marker-Based Augmented Reality in Museums,” in *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2019, pp. 205-215. 19

[11] Liu, J., Li, X., Zheng, Y., & Zhang, Y. (2021). A comparative study of marker-based and markerless augmented reality for cultural heritage. Journal of Cultural Heritage, 50, 36-44. doi: 10.1016/j.culher.2021.06.002 22

[12] S. Brusaporci, G. Ruggieri, F. Sicuranza, and P. Maiezza, ‘Augmented Reality for Historical Storytelling. The INCIPICT Project for the Reconstruction of Tangible and Intangible Image of L’Aquila Historical Centre’, Proceedings, vol. 1, p. 1083, Nov. 2017, doi: 10.3390/proceedings1091083. 9, 22, 23

[13] A. G. Lupascu, A. Ciupe, S. Meza, and B. Orza, ‘ARThings – enhancing the visitors’ experience in museums through collaborative AR’, in 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Mar. 2021, pp. 669–670. doi: 10.1109/VRW52623.2021.00217. 9, 24

[14] Carruthers, W. (2007). Uncertainty, archaeology, and the materiality of anxiety. In Carver, M., & Gaydarska, B. (Eds.), Archaeological Uncertainties: Proceedings of the 14th Annual Conference of the International Association for the Study of the Archaeology of the Caucasus (pp. 13-22). BAR International Series 1657. 25

[15] Wheatley, D., & Gillings, M. (2002). Spatial technology, time-depth, and social visibility: An example from Bronze Age Greece. European Journal of Archaeology, 5(3), 275-293. 25

[16] List of ARCore/ARKit Supported Devices:
https://developers.google.com/ar/discover/supported-devices

36