Mobile Outdoors Augmented Reality for Urban Planning Visualization

Gerasimos-Dimitrios Danalis

2024

School of Electrical and Computer Engineering, Technical University of Crete

Thesis Committee
Associate Professor Mania Aikaterini (ECE), (Thesis Supervisor),
Associate Professor Dimelli Despina (ARCH),
Associate Professor Deligiannakis Antonios (ECE)

Abstract

This thesis explores the process of designing, developing and implementing a fully functional, mobile, marker-less, augmented reality (AR) application for the visualization of urban planning changes outdoors. This application focuses on the urban reconstruction of three separate locations in the district of Nea Chora, Chania. In each of these locations the user can place and interact in unique ways with a set of different 3D virtual objects, such as a building, trees, benches, street lamps and a bicycle lane. Additionally, the application offers map navigation in order to assist the user in reaching the target locations and to ensure that the experience remains localized.

Our application was developed using the Unity Game Engine in conjunction with the ARFoundation SDK, which enables the augmented reality capabilities of the application, such as plane detection, environmental and object tracking and virtual object anchoring. ARFoundation provided the necessary framework for developing a marker-less, feature and sensor-based AR application that runs on both Android and iOs devices, by utilizing their respective native AR SDKs. We have divided the AR functionality into three distinct modes: 1) Placement mode, which is responsible for ground detection, previewing the 3D models and placing them, 2) Edit Mode, which implements all the ways the user can interact with the virtual objects and 3) View Mode, in which the user can view the scene they created. In addition, we have also implemented a manager script that handles the transitions between modes. Furthermore, we use the Haversine formula to calculate the distance between the user and each target location based on latitude and longitude received from the mobile device's GPS sensor. The MapBox SDK was used to implement a map screen capable of displaying a map of the local area containing all the target locations, as well as providing navigational instructions to the user. For the creation of our 3D models we used Krita and Unity's ProBuilder. Lastly, we employed the "think-out-loud" method to evaluate our application and gather user feedback.

Περίληψη

Η παρούσα διπλωματική εργασία διερευνά τη διαδικασία σχεδιασμού, ανάπτυξης και υλοποίησης μιας πλήρως λειτουργικής, εφαρμογής επαυξημένης πραγματικότητας για φορητές συσκευές, που δε βασίζεται σε αναγνωριστικές εικόνες, με σκοπό την οπτικοποίηση αλλαγών πολεοδομικού σχεδιασμού σε εξωτερικούς χώρους. Αυτή η εφαρμογή εστιάζει στην αστική ανακατασκευή τριών τοποθεσιών στη συνοικία της Νέας Χώρας Χανίων. Σε καθεμία από αυτές τις τοποθεσίες, ο χρήστης μπορεί να τοποθετήσει και να αλληλεπιδράσει, με ξεχωριστό τρόπο, με ένα πλήθος από διαφορετικά τρισδιάστατα εικονικά αντικείμενα, όπως ένα κτίριο, δέντρα, παγκάκια, λάμπες φωτισμού δρόμου και έναν ποδηλατόδρομο. Επιπλέον, η εφαρμογή προσφέρει δυνατότητες πλοήγησης σε χάρτη ώστε να βοηθήσει τον χρήστη να φτάσει στις τοποθεσίες-στόχους.

Η εφαρμογή αναπτύχθηκε μέσω της Unity Game Engine σε συνδυασμό με το ARFoundation SDK, που υλοποιεί τη λειτουργικότητα επαυξημένης πραγματικότητας, όπως την ανίχνευση επιφανειών, το tracking του περιβάλλοντος και αντικειμένων και το anchoring εικονικών αντικειμένων. Το ARFoundation παρέχει τα απαραίτητα εργαλεία ανάπτυξης εφαρμογών AR, που δε βασίζονται σε αναγνωριστικές εικόνες, αλλά σε χαρακτηριστικά στοιχεία του περιβάλλοντος και μετρήσεις αισθητήρων, τόσο για συσκευές Android όσο και για iOS, χρησιμοποιώντας τα αντίστοιχα native AR SDK τους. Έχουμε χωρίσει τη λειτουργικότητα AR σε τρία διακριτά modes: 1) Placement Mode, υπεύθυνο για την ανίχνευση του εδάφους, την προεπισκόπηση των 3D μοντέλων και την τοποθέτησή τους, 2) Edit Mode, το οποίο υλοποιεί όλους τους τρόπους αλληλεπίδρασης του χρήστη με τα εικονικά αντικείμενα και 3) View Mode, στο οποίο ο χρήστης μπορεί να δει τη σκηνή που δημιούργησε. Επιπρόσθετα, έχουμε δημιουργήσει ένα manager script που είναι υπεύθυνο για τις μεταβάσεις μεταξύ των modes. Επίσης, χρησιμοποιούμε τον τύπο Haversine για να υπολογίσουμε την απόσταση μεταξύ του χρήστη και κάθε τοποθεσίας-στόχο με βάση το γεωγραφικό πλάτος και το μήκος που λαμβάνεται από τον αισθητήρα GPS της κινητής συσκευής. Το MapBox SDK χρησιμοποιήθηκε για την υλοποίηση μιας οθόνης ικανής να εμφανίζει έναν χάρτη της τοπικής περιοχής που περιέχει όλες τις τοποθεσίες-στόχους, καθώς και να παρέχει οδηγίες πλοήγησης στον

χρήστη. Για τη δημιουργία των 3D μοντέλων μας χρησιμοποιήσαμε το Krita και το ProBuilder της Unity. Τέλος, χρησιμοποιήσαμε τη μέθοδο "think-out-loud" για να αξιολογήσουμε την εφαρμογή μας και να συγκεντρώσουμε σχόλια από τους χρήστες.

Acknowledgements

I would like to express my gratitude for Professor Aikaterini Mania, my thesis supervisor, who provided me with the necessary guidance and support, without which it would have been impossible to complete this research process.

Additionally, i would like to thank Professor Despina Dimelli for providing her expertise and invaluable insights regarding urban planning and the reconstruction of Nea Chora.

I am also grateful for all my friends, old and new, that have supported, helped and encouraged me in all my academic endeavors.

Furthermore, i am thankful for my family who has provided me with continuous support and love and have made all of this possible.

Last, but certainly not least, i would like to dedicate this work to my grandpa Michalis who was always there for me throughout my entire life.

Contents

Abstract **1**

Περίληψη **2**

Acknowledgements **4**

List of Figures **9**

List of Code Snippets **13**

1 Introduction **15**
1.1 Purpose of the Thesis 15
1.2 Brief Description 16
1.3 Thesis Structure 17

2 Research Overview **19**
2.1 Introduction 19
2.2 Augmented Reality (AR) 19
2.2.1 A brief history of Augmented Reality 20
2.2.2 The Registration Problem and Tracking Techniques . . 25
2.2.3 Trends in Augmented Reality Research 29
2.2.4 Head Mounted Displays (HMDs) 32
2.2.5 Mobile Augmented Reality (MAR) 37
2.2.6 Currently available Augmented Reality Software Development Kits (SDKs) 37
2.2.7 Currently Available Game Engines 44
2.3 Modern Augmented Reality applications 46
2.3.1 Commercial Applications 46
2.3.2 Academic papers focused on Augmented Reality applications 47
2.4 Outdoors Augmented Reality and Urban Planning 50
2.4.1 What is Urban Planning 50

2.4.2 Academic papers on Outdoors AR and Urban Planning 50

3 Requirements and Technological Background 60
3.1 Introduction . . . 60
3.2 Unity . . . 60
3.2.1 Assets . . . 61
3.2.2 Prefabs . . . 61
3.2.3 Materials . . . 61
3.2.4 Scenes . . . 61
3.2.5 Scripts . . . 61
3.2.6 Packages . . . 62
3.3 AR Foundation . . . 62
3.3.1 ARCore . . . 62
3.3.2 ARKit . . . 63
3.4 ProBuilder . . . 63
3.5 Krita . . . 64
3.6 Visual Studio Community 2019 . . . 64
3.7 Map SDKs . . . 64
3.7.1 Mapbox . . . 65
3.7.2 Google Maps . . . 65
3.7.3 ArcGIS . . . 65
3.8 Hardware . . . 66
3.9 Case Analysis . . . 66
3.9.1 Urban reconstruction of the district of Nea Chora in the city of Chania . . . 66
3.9.2 Current State of Nea Chora . . . 66
3.9.3 Proposed Urban Reconstruction of Nea Chora . . . 69
3.9.4 The target areas . . . 71
3.10 Thesis Requirements . . . 74

4 End User Experience 76
4.1 Introduction . . . 76
4.2 Android Permissions . . . 76
4.2.1 Location Service Permissions . . . 76
4.2.2 Camera Permissions . . . 77
4.3 Home Screen . . . 77
4.4 Map Screen . . . 79
4.5 AR Locations Screens . . . 81
4.5.1 Location 1 Screen (Trees, Benches and Street Lamps) . . . 82
4.5.2 Location 2 Screen (Building) . . . 86
4.5.3 Location 3 Screen (Bicycle Lane and Bicycle Station) . . . 88

5 Implementation **91**
5.1 Introduction . . . 91
5.2 Unity . . . 91
5.2.1 Setting Up AR Foundation . . . 92
5.2.2 Setting UP Mapbox . . . 92
5.2.3 Player Settings . . . 93
5.3 Designing the User Interface (UI) . . . 94
5.3.1 UI Sprites . . . 94
5.4 Home Scene . . . 97
5.4.1 Home scene hierarchy and components . . . 97
5.4.2 Switching Scenes . . . 98
5.4.3 GPS and Distance from Locations . . . 99
5.5 Map Scene . . . 103
5.5.1 Map scene hierarchy and components . . . 103
5.5.2 Adding Mapbox to the scene . . . 103
5.5.3 Tracking and displaying the user's position . . . 106
5.5.4 Centering the map to the user's location or the destination . . . 107
5.5.5 Displaying directions . . . 108
5.5.6 Switching between Light and Dark Map Theme . . . 110
5.6 Location Screens Overview . . . 111
5.6.1 Adding AR Foundation to the scene . . . 111
5.6.2 AR Modes . . . 114
5.6.3 Selecting and Outlining Objects . . . 120
5.7 Location 1 Screen (Trees, Benches and Street Lamps) . . . 121
5.7.1 Location 1 scene hierarchy and components . . . 121
5.7.2 Tree Models . . . 122
5.7.3 Street Lamp Models . . . 123
5.7.4 Bench Models . . . 123
5.7.5 AR Modes . . . 124
5.8 Location 2 Screen (Building) . . . 129
5.8.1 Location 2 scene hierarchy and components . . . 129
5.8.2 Building Model . . . 129
5.8.3 AR Modes . . . 131
5.9 Location 3 Screen (Bicycle Lane and Bicycle Station) . . . 133
5.9.1 Location 3 scene hierarchy and components . . . 133
5.9.2 Bicycle Lane Model . . . 133
5.9.3 Bicycle Models . . . 134
5.9.4 Bicycle Station Model . . . 135
5.9.5 AR Modes . . . 135
5.10 Tutorial System . . . 138

5.10.1 Enabling the tutorial system . . . 139
5.10.2 Tutorial Scripts . . . 140

6 Evaluation &Testing **142**
6.1 Introduction . . . 142
6.2 Evaluation Method . . . 142
6.3 Evaluation Results &Limitations . . . 143
6.3.1 User Interface (UI) and User Interaction . . . 143
6.3.2 Map Navigation . . . 144
6.3.3 Floor Detection, Object Tracking and Other Technical Issues . . . 144

7 Conclusions &Future Work **146**
7.1 Introduction . . . 146
7.2 Conclusions . . . 146
7.3 Future Work . . . 147

Bibliography **149**

List of Figures

1.1 Preview of the AR application's visualization capabilities. . . . 16

2.1 Fighter plane HUD. . . . 21
2.2 Sword of Damocles, created by Ivan Sunderland in 1986. . . . 21
2.3 Louis Rosenberg testing "Virtual Fixtures". . . . 22
2.4 Reality-Virtuality Continuum. . . . 22
2.5 ARQuake equipment. . . . 23
2.6 AR Tennis and Wikitude. . . . 24
2.7 Classification of AR tracking [27]. . . . 25
2.8 Marker-Based Tracking. . . . 26
2.9 Feature-Based Tracking. . . . 27
2.10 Model-Based Tracking [30]. . . . 28
2.11 Taxonomy of model-based tracking methods for Augmented Reality [30]. . . . 28
2.12 Magic Leap One. . . . 33
2.13 Magic Leap One controller and lightpack. . . . 33
2.14 Magic Leap 2. . . . 34
2.15 HoloLens. . . . 34
2.16 HoloLens 2. . . . 35
2.17 Meta Quest 3. . . . 35
2.18 ThinkReality A3. . . . 36
2.19 Nreal Air AR Glasses. . . . 36
2.20 Block Diagram of EasyAR CRS. . . . 43
2.21 Pokemon GO. . . . 46
2.22 IKEA Place. . . . 47
2.23 Testing the three different forms of IVAs. . . . 48
2.24 Virtual piano lesson from the point of view of a student. . . . 49
2.25 Mentor and mentee subsystems for surgical telementoring. . . . 49
2.26 City 3D-AR physical structure. . . . 51
2.27 City 3D-AR detailed flow chart. . . . 52
2.28 Major components of the proposed technique. . . . 53

2.29 Superimposition of weeds over the panoramic video frames; (A) Combination of computer-generated weeds with the background. (B) Several frames of the weeds in different years. . . 53
2.30 Overview of the proposed image-based AR system. 55
2.31 Outline of the offline training stage. 55
2.32 Outline of the on-line processing stage. 56
2.33 Overlay on scale model and the Graphical User Interface. . . . 57
2.34 Renderings of the two projects. 58
2.35 Prototype AR application displaying different levels of detail (LOD). 58

3.1 The ProBuilder window, featuring various different editing options. 63
3.2 Current land usage of Nea Chora. 67
3.3 The 3 zones. Zone A: light blue, Zone B: black, Zone C: gray. 68
3.4 Current road arteries. 68
3.5 Proposed land usage. 70
3.6 Proposed road system. 71
3.7 Proposed green spaces. 71
3.8 Aerial view of Area 1. 72
3.9 Aerial view of Area 2. 73
3.10 Aerial and street view of Area 3. 74

4.1 Location service permissions. 77
4.2 Camera permissions. 77
4.3 Home Screen use-case diagram. 78
4.4 Home Screen. 78
4.5 Tutorial Message example and Enabling/Disabling the tutorial system. 79
4.6 The current distance from each AR location and the AR location buttons. 79
4.7 Map Screen use-case diagram. 80
4.8 Map Screen. 80
4.9 Panning back to the user's current location or to a destination and viewing all target-locations on map. 81
4.10 Navigational instructions and Switching Map Themes. 81
4.11 Location Screen Flow Diagram. 82
4.12 Location 1 View Mode use-case diagram. 83
4.13 Location 1 View Mode. 83
4.14 Placement Mode use-case diagram. 84
4.15 Detecting the floor. 84

4.16 Placing a virtual tree, bench and street lamp. 85
4.17 Location 1 Edit Mode use-case diagram. 85
4.18 Location 1 Edit Mode and selected object outline. 86
4.19 Location 2 Placement Mode. 86
4.20 Placing a virtual building. 87
4.21 Location 2 Edit Mode use-case diagram. 87
4.22 Location 2 Edit Mode and Floor number. 88
4.23 Location 3 View Mode. 88
4.24 Location 3 Placement Mode. 89
4.25 Placing a virtual bicycle lane and a virtual bicycle station. . . 89
4.26 Location 3 Edit Mode use-case diagram. 89
4.27 Location 3 Edit Mode. 90

5.1 Unity version and modules. 92
5.2 Required packages for AR Foundation. 92
5.3 Importing a package. 92
5.4 Mapbox public access token. 93
5.5 Mapbox sample scenes. 93
5.6 Graphics API. 93
5.7 Android API. 94
5.8 Scripting backend and target architectures. 94
5.9 Example of a UI button using sprite swapping. 95
5.10 Button sprites. 95
5.11 Miscellaneous sprites. 96
5.12 Home Scene Background. 96
5.13 Home Scene Canvas. 97
5.14 Home Scene hierarchy and components. 98
5.15 Buttons that switch scenes. 98
5.16 Scene Manager components. 99
5.17 Displaying the distance from a location. 99
5.18 Map Scene hierarchy and components. 103
5.19 Map components and the The three points of interest (POIs). 104
5.20 Waypoint (POI marker). 105
5.21 Location Provider components. 106
5.22 The PlayerTarget prefab and its components. 106
5.23 UI components for centering the map to a specific AR location or the user's position. 107
5.24 Map Scene canvas components. 107
5.25 Directions and Waypoints. 108
5.26 The components of the Directions object. 110
5.27 UI component for switching between Light and Dark Mode. . 110

5.28 Setting up AR Foundation in our scene. 112
5.29 AR Session components. 112
5.30 AR Session Origin components. 113
5.31 AR Camera components. 114
5.32 The View Mode component attached to each scene's AR Session Origin. 116
5.33 Occlusion toggle check box. 116
5.34 The Placement Mode component attached to each scene's AR Session Origin. 117
5.35 The placement marker. 117
5.36 The Edit Mode component attached to each scene's AR Session Origin. 119
5.37 Example of the components attached to one of our bench models. 121
5.38 Location 1 Scene hierarchy and components. 122
5.39 Trees 1, 2 and 3 (bush). 122
5.40 Trees 4, 5 and 6. 122
5.41 Street Lamps 1 and 2. 123
5.42 Street Lamps 3, 4 and 5. 123
5.43 Benches 1 and 2. 124
5.44 Benches 3 and 4. 124
5.45 Bench 5. 124
5.46 The ARTreeManager component, attached to the scene's AR Session Origin. 125
5.47 The slider used for changing the tree's scale. 128
5.48 Location 2 Scene hierarchy and components. 129
5.49 The building model, with 2 floors. 130
5.50 The building model, with 4 floors. 130
5.51 The hierarchy of the building model with 2 floors. 131
5.52 The hierarchy of the "Floors" prefab. 131
5.53 The ARBuildingManager component, attached to the scene's AR Session Origin. 132
5.54 Location 3 Scene hierarchy and components. 133
5.55 The Bicycle Lane model and a closer look at its texture. . . . 134
5.56 The two materials attached to the middle plane and the bicycle image. 134
5.57 Bicycle 1. 134
5.58 Bicycle 2. 135
5.59 The Bicycle Station model. 135
5.60 The ARBicycleLaneManager component, attached to the scene's AR Session Origin. 136
5.61 Examples of the tutorial messages' appearance. 139

5.62 Tutorial canvas example (Location 1). 139

List of Code Snippets

5.1 Loading scenes via the Scene Manager. 98
5.2 Example of a scene getting loaded. 98
5.3 Starting the location service based on the user's platform. . . 100
5.4 Updating the Distance to each AR location. 100
5.5 Calculating the Haversine Distance. 102
5.6 Moving and zooming the map using touch. 105
5.7 CenterMapToLocation script. 107
5.8 Placing a waypoint on the map using geocoordinates. 109
5.9 Beginning and ending the navigation process. 110
5.10 ToggleLightDarkMode script. 111
5.11 ARManager script. 115
5.12 Toggling occlusion on or off. 116
5.13 Raycasting to place virtual objects. 117
5.14 Creating a virtual object. 118
5.15 Rotating and Deleting a virtual object. 119
5.16 Raycasting in the Edit Modes Update function. 119
5.17 SelectARObject script. 120
5.18 Enabling each AR Mode. 125
5.19 Selecting the virtual object prefab from the dropdown list. . . 126
5.20 Function for scaling trees. 128
5.21 Enabling the scale slider and setting it to the correct value. . . 128
5.22 Replacing the placed building model with a diffrent one to change the number of floors. 132
5.23 Bicycle manager script. 137
5.24 Removing and adding bicycles to the bicycle lane. 138
5.25 Enabling or disabling the Tutorial System. 139
5.26 Showing the next tutorial message in location 1. 140
5.27 Resuming the tutorial from where the user last left off in location 1. 141

Chapter 1

Introduction

1.1 Purpose of the Thesis

Augmented Reality (AR) is a technology that has seen a significant surge in research and improvements the past 20 years. Its rapid rate of development and its wide range of application, has granted AR both commercial popularity (eg. Pokemon GO) and popularity in various research fields, including those of medicine, education and architecture. Superimposing virtual elements onto the real-world offers new ways to perceive our environment and adds a new layer of interactivity and freedom to previously mundane or unengaging processes.

A particularly interesting sector that could vastly benefit from the application of AR is that of architecture and more specifically urban planning. Urban planning is the process of guiding and directing the use and development of land to ensure its sustainability and efficient use [31]. Oftentimes, visualizing the proposed urban planning changes can be a challenging task, limited to architectural drawings, virtual 3D models or construction spans, none of which can accurately portray how the proposed change will actually look when implemented. This is where AR comes in.

AR provides the tools needed to visualize these urban planning changes in a way that is cost-effective, realistic and accurate. By utilizing this technology, we can create systems that allow us to visualize the changes in real-time, while also being able to have a level of interactivity that couldn't be possible otherwise. The potential of combining AR with the urban planning process is what lead to our decision to develop a mobile AR application focused on visualizing outdoors urban planning changes.

1.2 Brief Description

The mobile AR application that was developed for this thesis provides a system through which the user can visualize certain urban planning change scenarios in the district of Nea Chora, in the city of Chania. The application focuses on three separate locations in the district, aiming to provide a different and unique experience in each of them. Our goal is to showcase the potential of AR in creating interactive visualization experience's to aid the process of urban planning.

We decided to opt for mobile devices (Android and iOS) as our applications target, since those provide widespread and relatively cheap availability as opposed to more specialized gear such as head mounted displays which can cost up to thousands of euros. For this purpose, we selected the Unity Game Engine and its AR Foundation package which allows us to develop a single codebase for both platforms by utilizing either ARCore or ARKit depending on the user's mobile device.

Figure 1.1: Preview of the AR application's visualization capabilities.

The developed application features location based AR, by tracking the users location via GPS coordinates using the mobile devices location sensors. Additionally, a map screen has been implemented to act as a guidance system that gives directions on how to reach each AR location. For the AR locations a plethora of 3D models were selected or created to fulfill our visualization needs.

The three locations that were chosen for our application feature the following three AR scenarios. At the first location the user can place and interact with 3D models of benches, trees and street lamps, creating their own unique take of how a coastal sidewalk should look. The second location

allows the user to place a virtual building and then edit the number of floors that it has. The last location features the placement of a bicycle lane and a bicycle station at the intersection of two of the main road arteries of Nea Chora.

We hope that our mobile application will act as a practical showcase of the potential that AR has in the field of urban planning and promote the use of interactive, real-time visualization as an architectural tool.

1.3 Thesis Structure

Chapter 1 describes what the goal of this thesis is and how the thesis text is structured.

Chapter 2 provides an overview of augmented reality (AR) and urban planning and includes an include an in-depth review of the literature regarding AR. We start by examining the history and evolution of AR, the trends in research and the current state of the art. Then, we take a closer look at both commercial and research applications of AR. Lastly, we provide a definition for urban planning and go over some academic papers focused on outdoors AR applications or AR applications used for aiding the urban planning process.

Chapter 3 goes over the software used for the development of our application, including some that were considered but eventually rejected. It also provides an analysis of the specific case of the district of Nea Chora, Chania, and presents a proposed urban reconstruction of the area.

Chapter 4 goes over the end-user experience when using our AR application. It provides a detailed overview of how a user would interact with the application, how the user interface (UI) is designed and also provides use-case diagrams for each screen.

Chapter 5 describes the technical implementation of our AR application. It provides a detailed analysis of how the application was developed and how each of its features are implemented. In this chapter we will describe how AR Foundation is utilized in our application, how the Mapbox map scene was designed, how the UI elements were designed, how each virtual model was selected or created, how the interactions with the AR objects are implemented and how the tutorial system works.

Chapter 6 describes the evaluation method that was used, as well as its results and what those mean for our application.

Chapter 7 goes over our final conclusions regarding our application and the use of AR as an urban planning visualization tool and also presents what future research and work could entail.

Chapter 2

Research Overview

2.1 Introduction

In this chapter, we will define some core concepts related to this thesis, such as augmented reality (AR), head mounted displays (HMDs) and urban planning. We will also take a dive into the history of augmented reality, since its inception as an idea, the research trends of the last two decades and the current state of the art. Lastly, we will examine some more practical applications of augmented reality, both in the academic research and commercial sectors, and especially those that pertain to urban planning and outdoors use.

Our goal is that by the end of this chapter we will have attained a better understanding of the fundamental concepts of augmented reality and a clear view of the academic landscape. Additionally, it will become apparent that augmented reality can have a multitude of beneficial use-cases in urban planning and aid the process in a unique way.

2.2 Augmented Reality (AR)

Augmented Reality is the integration of virtual information with the user's environment in real time. It is an interactive experience during which virtual elements are superimposed onto the user's physical environment through the use of a virtual medium. This computer-generated content can be visual, auditory, haptic, somatosensory, or even olfactory. Ronald T. Azuma in his paper "A survey of Augmented Reality" [1] defines AR as systems that have the following three characteristics: 1) combines real and virtual, 2) interactive in real time and 3) registered in 3-D. This is a very important step towards firmly defining what augmented reality is.

Augmented Reality differs from Virtual Reality (VR) in the sense that the user still retains a direct connection to his immediate physical environment. In the latter the user experiences a totally virtual environment and all the information he receives is digitally constructed.

2.2.1 A brief history of Augmented Reality

Even though AR has only started to become widespread and more mainstream in recent years, through the development of powerful HMDs or commercial applications like "Pokemon GO" that can run on any mobile phone, the concept goes further back in time that one might initially guess. The first description of a concept that feels eerily similar to AR dates all the way back to 1584, though the first actual use of AR as we know it today would come far later.

AR has been used in the military, notably in the HUDs of fighter planes, since 1958 and the first HMD was created in 1986. The potential applications of AR are limitless, spanning from medical, military, educational to purely recreational. There has been a continuous effort led both by scientists and commercial companies to improve the technology and create a more robust and reliable AR experience.

The first iterations of HMDs were limited by the technology of their time. Inaccurate tracking and cumbersome contraptions are just a few of the multitude of problems that had to be overcome for AR to reach the state it is now. It is worth taking a look at the history and gradual advancement of AR technology before diving into it's modern counterparts [16].

1584: Giambattista della Porta, in his book "Magia Naturalis" [36] presents a concept of a room in which you have a glass pane set up in a way that reflects light bouncing off objects in a different position, creating the illusion that these objects are in a different position than they actually are. This description resembles the principle of reflection that head-mounted displays use.

1958: Head-up displays (HUDs) are used in fighter planes, overlaying information by projecting onto a glass plane in the pilots field of view.

Figure 2.1: Fighter plane HUD.

1986: The first head mounted display is created by Harvard Professor Ivan Sutherland, who was aided by his students Bob Sproull, Quintin Foster, and Danny Cohen [14]. "The sword of Damocles", as this device was named, is the first augmented reality display and tracking system ever created. The device was attached to a mechanical arm suspended from the ceiling. Of course, this prototype head mounted display was limited by the technology of it's time.

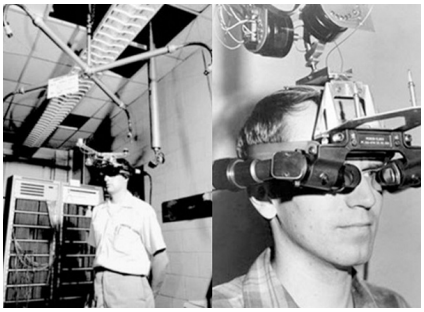


Figure 2.2: Sword of Damocles, created by Ivan Sunderland in 1986.

1990: Tom Caudell and David Michel coin the term "augmented reality". The pair was working on a project to assist electricians when assembling complicated wiring harnesses and realized that Heads-Up Displays (HUDs) could be used in order to provide the workers with the needed information in real time, thus creating a form of Augmented reality displays [15].

1992: Louis Rosenberg created the first fully functional augmented reality system, called "Virtual Fixtures" [25]. It was a virtual flying training tool for Air Force pilots.

Figure 2.3: Louis Rosenberg testing "Virtual Fixtures".

1993: J. M. Loomis, R. G. Golledge, and R. L. Klatzky combine a notebook with a differential GPS receiver and a head-worn electronic compass to develop an outdoor application for the visually impaired [2]. The application uses GIS data and provides navigational assistance using an acoustic virtual display.

1994: Julie Martin creates "Dancing in Cyberspace", the first augmented reality theatre performance. During this performance, acrobats danced around virtual human-sized objects.

1994: Paul Milgram and Fumio Kishino write the paper "A Taxonomy of Mixed Reality Visual Displays" [4] [5]. This paper defines the "Virtuality Continuum", a continuous scale that connects completely real environments to complete virtual ones and encompasses all possible compositions of virtual and real environments.

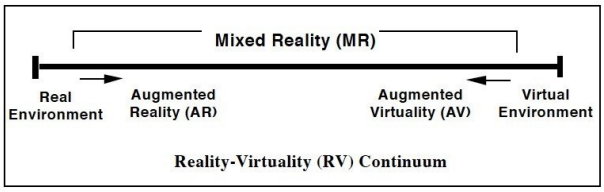


Figure 2.4: Reality-Virtuality Continuum.

1997: Ronald T. Azuma writes the paper "A survey of Augmented Reality" [1] and defines AR as systems that have the following three characteristics 1) Combines real and virtual 2) Interactive in real time 3) Registered in 3-D. This is a very important step towards firmly defining what augmented reality is.

1999: NASA's X-38 spacecraft uses AR to assist in providing navigation during test flights [3]. This virtual cockpit window provides an all-weather, day/night situation awareness display, enriched with a wide variety of flight-related information.

2000: Hirokazu Kato develops an open-source software library named "ARToolKit", a framework for creating real-time augmented reality applications. It tackles one of the most difficult parts of developing an AR application: calculating the user's viewpoint in real time, thus taking a step towards easier mainstream AR development. Through the use of computer vision algorithms this framework can calculate real time camera position and orientation relative to physical markers and allow the placement of virtual objects on these markers. ARToolKit (as well as ARToolKitX, a project that promises continuous support for the ARToolKit community) exists to this day and is still used.

2000: ARQuake [6], an AR version of the popular video game Quake, is developed by Bruce H. Thomas in Wearable Computer Lab at the University of South Australia. It uses an AR head mounted display, a portable unix based computer in the form of a backpack, head tracker, GPS system and a plastic prop gun to provide a first person shooter experience in the real world. It has a 6DOF tracking system based on GPS, a digital compass and fiducial vision-based tracking.

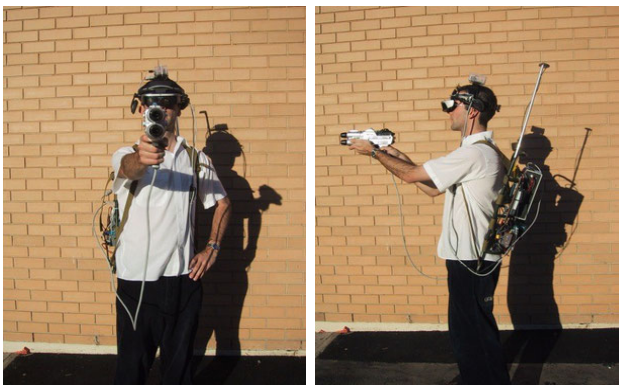


Figure 2.5: ARQuake equipment.

2001: J. Newman, D. Ingram, and A. Hopper develop the BatPortal [7], a PDA-based, wireless AR system. This implementation of AR allowed the user to roam freely within an entire building and view the augmented world

through a PDA and a worn “Bat”, a specially built device for the purpose of localization though the measurement of ultra-sonic pulses.

2005: The first collaborative AR application for mobile phones, “AR Tennis” is developed using a ported version of ARToolKit [26].

2008: Mobilizy launches Wikitude, an application that combines GPS and compass data with Wikipedia entries. The Wikitude World Browser overlays information on the real-time camera view of an Android smartphone.

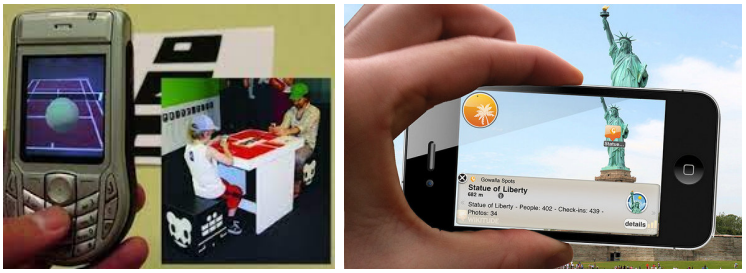


Figure 2.6: AR Tennis and Wikitude.

2012-2013: Google glass is announced in 2012 and released in 2013. Goggle Glass is an optical HMD that can be controlled with an integrated touch-sensitive sensor or natural language commands.

2016: Microsoft releases the HoloLens, the first iteration of their HoloLens Head Mounted Displays (HMDs) series.

2016: Pokemon GO, a mobile phone game using Augmented Reality, is released. Pokemon GO uses AR to project Pokemon onto the real world and ends up being a commercial success. This showcases that AR has use cases both in the scientific and the entertainment sectors.

2019: Magic Leap releases the Magic Leap One, the first iteration of their Magic Leap Head Mounted Displays (HMDs) series. The device became available on the United States after AT&T invested in Magic Leap and was initially only available through AT&T stores.

2019: Microsoft releases the HoloLens 2, the successor of HoloLens. This devices offers various improvements over it’s predecessor and aims to replace it.

2022: Magic Leap releases the Magic Leap 2, the successor of Magic Leap One, featuring upgraded hardware such as higher resolution, wider field of view (FOV) and dynamic dimming.

2.2.2 The Registration Problem and Tracking Techniques

One of the most important issues in Augmented Reality (AR) research is the registration problem, since correct registration is crucial to ensuring that the virtual information is being accurately displayed within the real environment. The objects in the real and virtual worlds must be properly aligned with respect to each other, or the illusion that the two worlds coexist will be compromised [1]. More importantly, in certain applications of augmented reality, eg. medical telementoring [13], accurate registration is imperative and the application can't be considered fully-functional without it.

Registration errors can be divided into two categories: static and dynamic. Static errors are the ones that cause registration errors even when the user's viewpoint and the objects in the environment remain completely still. Dynamic errors are the ones that have no effect until either the viewpoint or the objects begin moving [1]. Static errors can be minimized with correct calibrations since some of their main causes are incorrect viewing parameters, poorly calibrated mechanical parts and optical distortions. Dynamic errors on the other hand are more difficult to combat. Dynamic errors in modern augmented reality application are mainly cause by incorrect pose estimation, meaning an incorrect determination of the viewers position and orientation in relation to a real-world anchor point.

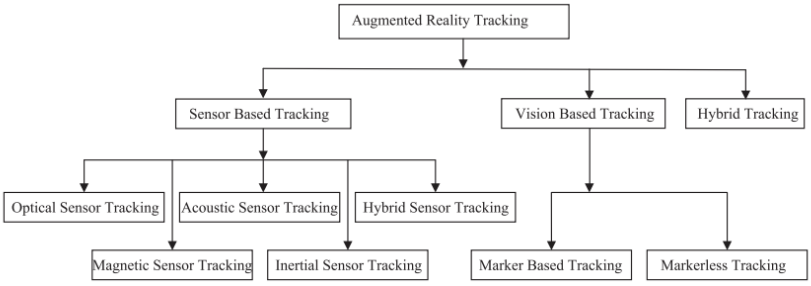


Figure 2.7: Classification of AR tracking [27].

Various tracking approaches exist, each of them with its own advantages and disadvantages, making them suitable for certain use-case scenarios but not for others. The most popular of these approaches will be explored in the next few sections.

2.2.2.1 Marker-Based Tracking

This method relies on physical markers, such as QR codes [28] or other distinct patterns, to calculate the viewers pose in relation to these markers. Once the marker is detected it's relatively easy to calculate its relative position and orientation and then project virtual content that is aligned with the marker. This approach offers high tracking precision making it a very reliable way to calculate pose, but is limited by the fact that the markers must always be in view of the camera. Secondly, marker-based tracking requires, by definition, an environment in which the markers have already been placed, which significantly limits the users freedom of movement. Therefore, this tracking technique is most suited for indoors spaces, for example museums or art, design and cultural heritage projects [29].

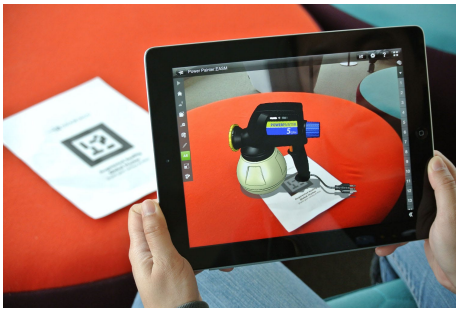


Figure 2.8: Marker-Based Tracking.

2.2.2.2 Feature-Based Tracking

Feature-Based tracking is the method of detecting and using distinct characteristics or features of real-world objects instead of markers. These features can be edges, corners, flat surfaces or any other distinct feature of the environment. Once those features are detected they are used to calculate the camera pose in relation to them. This methods viability and accuracy depends on the number of distinct features that are present in the use case environment, thus being more suited for feature-rich environments. Other factors that can negatively impact the performance of feature-based tracking include poor illumination and occlusion of the distinct features.

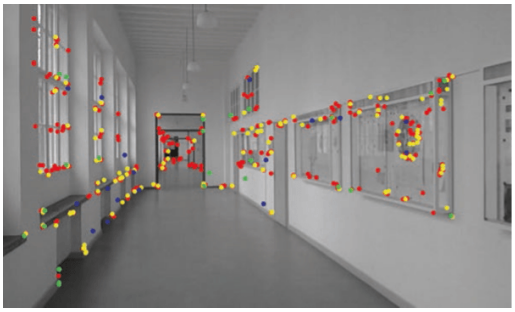


Figure 2.9: Feature-Based Tracking.

2.2.2.3 Sensor-Based Tracking

This approach utilizes the augmented reality devices sensors in order to calculate the devices orientation and position. Such sensors include accelerometers, magnetometers, gyroscopes or even GPS coordinates, though other sensors could also be utilized in more complex or specialized devices. The most commonly used approach is combining the readings from the device's accelerometer (linear movement calculation), the magnetometer (orientation calculation) and the gyroscope (rotational movement calculation) to estimate changes in the devices position and orientation. This method is prone to drift over time and is usually used in conjunction with other tracking techniques to improve their accuracy and performance. Mobile augmented reality (MAR) systems most often utilize such sensor-based techniques. For example both the ARCore and ARKit Software Development Kits (SDKs), which we will explore in future sections of this thesis, utilize some form of sensor-based tracking.

2.2.2.4 Model-based Tracking

Model-Based tracking uses 3D models of the environment or the objects therein to track orientation and position. Unlike marker-based tracking, this class of tracking methods does not rely on artificial patterns but rather on natural objects found in the scene [30]. This type of tracking is most often achieved through edge detection in order to create 3D models of the actual objects and subsequently detect and track them in the real-world.

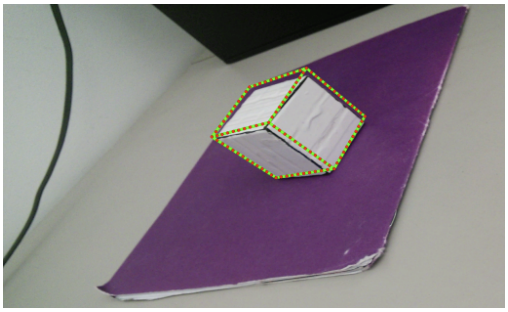


Figure 2.10: Model-Based Tracking [30].

Model-based tracking can be classified into two categories: recursive tracking and tracking by detection. In recursive tracking, the previous camera pose is used as an estimate to calculate the current camera pose. Tracking by detection, on the other hand, can calculate the pose without any prior state knowledge, but these methods are usually computationally expensive and require high processing power [30].

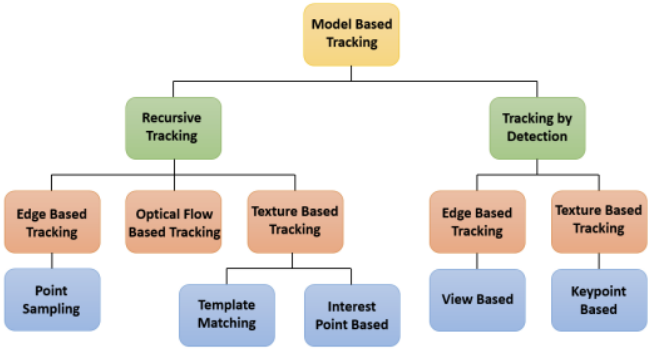


Figure 2.11: Taxonomy of model-based tracking methods for Augmented Reality [30].

2.2.2.5 Hybrid Tracking

Hybrid tracking is the combined usage of more than one tracking techniques. This approach aims to utilize each used tracking method's strengths, while trying to minimize it's limitations. For example a lot of mobile augmented reality (MAR) applications use a combination of sensor-based and feature-based tracking, or sensor-based and marker-based tracking.

In this thesis, we will employ a marker-less, hybrid tracking technique, via the use of the ARFoundation SDK. This SDK utilizes the ARCore and ARKit

SKDs, for Android and iOS devices respectfully, which use both sensor-based and feature-based tracking to detect and track objects. This is achieved by utilizing the mobile device's camera, GPS tracker, gyroscope and any other sensors a conventional smartphone possesses.

2.2.3 Trends in Augmented Reality Research

Before we dive into the currently available AR hardware we should first take a look at the trends in AR research spanning the last 20 years. A good starting point is the International Symposium on Mixed and Augmented Reality (ISMAR), a major academic conference focused in the fields of Augmented Reality and Mixed Reality.

2.2.3.1 A review of the first decade of ISMAR [8]

Feng Zhou, Henry Been-Lirn Duh and Mark Billinghurst in their review of the first decade of ISMAR [8], published in 2008, note that published research papers can be broken down into two groups. The first group contains the five main research areas of:

1. Tracking techniques (20.1%): methods for tracking a target object/environment via cameras and sensors, and estimating viewpoint poses.
2. Interaction techniques (14.7%): techniques and interfaces for interacting with virtual content.
3. Calibration and registration (14.1%): geometric, or photometric calibration methods, and methods to align multiple coordinate frames.
4. AR applications (14.4%): research on AR systems in application domains such as medicine, manufacturing, or military, among others.
5. Display techniques (11.8%): research on display hardware to present virtual content in AR, including headworn, handheld, and projected displays.

These are the five topics that papers are published on most frequently, which is interesting because they are also the core AR technology areas needed to deliver an AR application. The second group of topics reflects more emerging research interests, including:

1. Evaluation/Testing (5.8%): research focusing on human-subject studies evaluating AR techniques or systems.

2. Mobile/Wearable AR (6.1%): research on AR applications and techniques for wearable and mobile platforms, such as tablets and smartphones.

3. AR authoring (3.8%): research on methods, techniques, and systems for authoring virtual content in AR.

4. Visualization (4.8%): research into methods that use AR to make complex 2D/3D data easier to navigate through and understand.

5. Multimodal AR (2.6%): research combining different input (and output) modalities together, such as combined speech and gesture interfaces.

6. Rendering (1.9%): research into techniques for computer graphics rendering; and other sensory modalities, such as sound and haptics.

2.2.3.2 A review of the second decade of ISMAR [24]

Kangsoo Kim, Mark Billinghurst, Gerd Bruder, Henry Been-Lirn Duh, and Gregory F. Welch conducted a second review. as a follow-up to the previously published review of the first decade of ISMAR, covering its second decade [24] of conferences. Their goal was to answer three key questions:

1. Have the research trends changed from the first decade of ISMAR?

2. Are there any new research areas that have been introduced?

3. What are the key developments and challenges in the research areas?

Their results showed that several further topics relating to emerging research had appeared and there had been some changes in the most frequent research topics as well.

Regarding the research categories, the authors added 4 new ones to the original 11, bringing the total up to 15. The four new categories are:

1. Perception: research investigating human perception/cognition in AR.

2. Collaboration/Social: research on interactive collaborative systems for multiple remote or co-located users.

3. Reconstruction: research on methods that automatically generate 3D virtual environments/objects based on images or other forms of data collected from the real environment/objects.

4. Modeling: research on methods for creating virtual content via virtual primitives and tools with a human user's involvements.

The new trends in research papers showed a major increase in Evaluation research (16.4%), which signifies that AR technology has become more mature, advanced and accessible to more users, therefore increasing the need for systematic evaluation. Another category that saw a significant increase in research was Rendering (12.5%). Below we list all the categories sorted by number of papers they appear in:

1. Tracking techniques(16.2%)
2. Evaluation/Testing (13.7%)
3. AR Applications (10.5%)
4. Rendering (10.5%)
5. Interaction techniques(9.5%)
6. Mobile/Wearable AR (7.6%)
7. Perception (7.2%)
8. Reconstruction (5.5%)
9. Calibration and Registration (5.3%)
10. Visualization (4.0%)
11. Display techniques (2.7%)
12. Modeling (2.1%)
13. AR Authoring (1.9%)
14. Multimodal AR (1.7%)
15. Collaboration (1.7%)

2.2.3.3 Conclusions

After examining these two reviews it becomes clear that there is interest in the development of new AR applications, as is highlighted by the fact that AR applications have remained in the top five research categories (in terms of academic papers published) in the entirety of the last two decades. The sector of mobile AR applications is of especial interest, since it ranks high enough (7.6% of recent papers) without being over-saturated.

In this thesis, for the reasons mentioned above, we have chosen to focus on developing a new AR application for mobile devices.

Now, let's take a closer look at the Display hardware that is readily available for AR use today.

2.2.4 Head Mounted Displays (HMDs)

A head-mounted display (HMD) is a display device, worn on the head or as a built-in part of a helmet, that has a small display optic in front of one (monocular HMD) or each eye (binocular HMD). There are also optical head-mounted displays (OHMD), which are wearable displays that can reflect projected images and allow a user to see through them. Modern HMDs no longer resemble the cumbersome prototypes of the past ("Sword of Damocles" or the ARQuake kit) but instead can be worn comfortably and require minimal to none additional equipment.

HMDs can be utilized both for AR and VR applications. The deciding factor when choosing a HMD for AR is its ability to display computer-generated imagery onto the real world, as opposed to VR HMDs that can only display entirely virtual elements. Combining real-world view with CGI can be done by projecting the CGI through a partially reflective mirror and viewing the real world directly.

The techniques and technology used in order to achieve these "transparent" displays are explored in great detail on the paper written by G. A. Koulieris et al. [9]. In this paper the authors give an overview of the basics of light and image formation and proceed to present a detailed report of the state of the art concerning near-eye display and tracking technologies.

The rapid advancements and popularization of AR has pushed more and more companies to try and be competitive and create powerful AR HMDs.

Industry giants like Microsoft (HoloLens) and Google (Google glass) have been competing with each other and investing into creating accessible AR solutions.

Below we'll take a look at some of the currently available Augmented Reality (AR) Head Mounted Displays (HMDs).

2.2.4.1 Magic Leap One

Magic Leap One is a popular AR and Mixed Reality (MR) standalone device designed by Magic Leap. It offers a variety of different sensors (peripheral cameras, depth sensors, motion sensors, eye cameras, microphones, picture camera) and features to the user that can be used to develop advanced AR applications. Namely some of the capabilities of Magic Leap One include: building and refining spatial maps of your environment, identifying hand gestures, measuring light intensity, determining head position and speed, as well as issuing speech commands.

Figure 2.12: Magic Leap One.

In addition to the HMD the Magic Leap One comes with a tethered Lightpack (a portable mini computer that is the main source of processing power for the device) as well as a controller with 6DOF of movement.

Figure 2.13: Magic Leap One controller and lightpack.

2.2.4.2 Magic Leap 2

Magic Leap 2 features a similar design to its predecessor but comes with upgraded hardware and capabilities. It has a higher resolution, wider FOV and dynamic dimming which makes it easier to view the projected virtual content in bright areas. It still comes with a controller and a compute pack like Magic Leap One.

Figure 2.14: Magic Leap 2.

2.2.4.3 Microsoft HoloLens

Microsoft's first iteration of the HoloLens series is a fully untethered holographic computer. It comes with a controller but in contrast with Magic Leap it doesn't require an external processing unit. The HoloLens is an HMD unit connected to an adjustable, cushioned inner headband, which can tilt HoloLens up and down, as well as forward and backward. Some of its feature's lineage can be traced back to the Kinect. It has been replaced by HoloLens 2 but still remains a respectable option for developing powerful AR applications.

Figure 2.15: HoloLens.

2.2.4.4 Microsoft HoloLens 2

The successor of HoloLens boasts numerous improvements over the previous model. It has improved processing power, a wider field of view and a slightly

better battery life. Some of its features include: Head tracking, Eye tracking, Hand tracking, Voice commands and recognition, 6DOF tracking and spatial mapping.

Figure 2.16: HoloLens 2.

2.2.4.5 Meta Quest 3

Meta Quest 3 is the successor of Meta Quest 1 and 2, designed by Meta. It is primarily used as a Virtual Reality (VR) HMD but can also function as an Mixed Reality (MR) HMD. The key difference from the previously mentioned HMDs is that Meta Quest 3 doesn't feature a see-through lens but instead relies on camera feeds.

Figure 2.17: Meta Quest 3.

AR Glasses

It's worth noting that there's also a plethora of AR glasses available, which are lightweight and even though they're not as powerful as the aforementioned HMDs they can still provide an immersive AR experience, albeit more restricted or contained. Some of the available options are the following:

2.2.4.6 ThinkReality A3

ThinkReality A3 are smart AR glasses produced by Lenovo that can be used to create and customize your virtual workspace. The difference between these and standalone HMDs is that they need to be connected to a computer or mobile phone in order for the user to be able to fully use them. They're meant to enhance your work experience rather than create autonomous AR experiences.

Figure 2.18: ThinkReality A3.

2.2.4.7 Nreal Air AR Glasses

Similar to Lenovo's ThinkReality A3 these glasses also require a connection with a computer, mobile phone, tablet or gaming console. They're meant as a medium to experience movies, games or working on a computer in a different, "screen-free" way.

Figure 2.19: Nreal Air AR Glasses.

In this thesis, we decided to opt-out of using an HMD and instead focus on mobile devices as the target for our application and in particular mobile phones. The advantages of this approach will be examined in the section below.

2.2.5 Mobile Augmented Reality (MAR)

Mobile Augmented Reality systems refer to any portable device that can be used for AR or MR applications. In the current age that includes mobile phones, tablets, PDAs and many more similar devices. The two major advantages of this approach compared to more "traditional" HMDs is that it increases the availability and accessibility of the application, since the equipment required is readily available for all users (especially in the case of mobile phones, where the vast majority of the population already has access to one) and that the experience is no longer tied to a restricted specially equipped physical location. Easier widespread access to equipment and flexibility on the location equals a broader audience that can now experience AR.

Mobile phones in particular are the perfect specimens for AR applications. Modern smartphones possess powerful processing units and all the sensors required for AR. They can perform image recognition, object tracking, have a built-in camera that can be used as a replacement of the see-through lens of HMDs, can access GPS, gyroscope and motion sensor data and can also recognize voice commands. The only drawback in using mobile devices for AR is that the immersion of the user in somewhat lessened but the advantages far outweigh this disadvantage in most cases.

In this thesis, after careful consideration, we elected to utilize mobile devices over HMD's. The main appeal of mobile devices, and more specifically mobile phones, is their widespread availability which ensures that a variety of different user's would potentially be able to access our application. This means that our application could be used both as a participatory tool for the public and a planning tool for architects and engineers.

2.2.6 Currently available Augmented Reality Software Development Kits (SDKs)

There's a wide selection of available Software Development Kits (SDKs), both from smaller "indie" developers and from well-established industry companies like Google or Apple. Sadly, there's quite a few SDKs that have been discontinued or are not updated as often. Below we'll list some of SDKs available at the time of writing this thesis.

2.2.6.1 ARCore

Googles SDK for developing AR applications [39], initially released on March 2018 and with its first stable release on January 2021. It offers cross-platform APIs for building AR experiences on Android, iOS, Unity, Unreal and Web.

Some of the fundamental features that ARCore provides include:

Motion tracking: ARCore uses Simultaneous Localization and Mapping (SLAM) to understand where the mobile device is relative to its environment. Any changes in location are computed based on visually distinct features captured on camera, called feature points. The visual information is combined with inertial measurements from the device's IMU to estimate the pose (position and orientation) of the camera relative to the world over time. It's also worth noting that ARCore has 6DOF tracking.

Environmental understanding: ARCore detects clusters of feature points that appear on common horizontal or vertical surfaces in order to make those surfaces available as geometric planes for the application to interact or place items on.

Depth understanding: ARCore can create depth maps, images that contain data about the distance between surfaces from a given point, using the main RGB camera from a supported device.

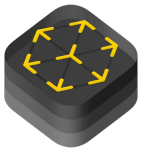
Light estimation: ARCore provides information about the average intensity and colour correction of the environment which allows you to light virtual objects correctly to appear more seamlessly interwoven into the real world.

Anchors and Trackables: Anchors ensure that ARCore tracks virtually placed objects positions over time. Trackables are environmental objects, like planes or feature points, thay ARCore keeps track off. By anchoring objects on trackables you ensure that the relative position of those objects will remain stable while the mobile device moves around.

ARCore is still actively supported and updated by Google and used for a multitude of AR applications.

2.2.6.2 ARKit

Apple's counterpart to Google's ARCore [40]. It's an AR development platform for iOS devices that has similar features to those of ARCore.



Some of these features are:

Environmental understanding: ARKit is able to analyse the camera feed video to extract information about the scene's geometry. It can create a topological map of the surrounding space with labels identifying objects like floors, walls or doors.

Depth understanding: ARKit uses the devices LiDAR scanner to collect per-pixel depth information about the surrounding environment which in turn are combined with scene generated 3D mesh data. This depth information makes virtual object occlusion more realistic by enabling instant placement of virtual objects and blending them seamlessly with their physical surroundings.

Plane detection: The LiDAR Scanner enables incredibly quick plane detection, allowing for the instant placement of AR objects in the real world without scanning.

Anchors: Just like ARCore, it is possible to create anchor relations between objects and surfaces.

Motion Capture: ARKit is able to process body motions and receive them as input to the AR experience.

People Occlusion: AR content realistically passes behind and in front of people in the real world, making AR experiences more immersive while also enabling green-screen-style effects in almost any environment.

Similarly to ARCore, ARKit is still actively supported and updated by Apple, with ARKit 6 being the latest version and bringing 4k support to the

SDK.

2.2.6.3 ARFoundation

AR Foundation is a Software Development Kit (SDK), designed by Unity Technologies, focused on creating multi-platform augmented reality (AR) applications using the Unity game engine. AR Foundation utilizes both the ARCore and ARKit SDK's by choosing which one to access based on which SDK is native to user's device's platform. This significantly reduces the development time needed to create an AR application for mobile devices, since it eliminates the need for different codebases for each of the available platforms.

2.2.6.4 Vuforia

An AR development kit, owned by PTC [41]. The Vuforia Engine can be used for free but there's also a paid option that includes more features and support.

The core functionality offered by the SDK, as described in their developer documentation is:

Image Targets: This represents the images that Vuforia can detect and track by extracting natural features from the camera feed and comparing them to a known resource database. Common uses of Image Targets include recognizing and augmenting printed media and product packaging for marketing campaigns, gaming, and visualizing products in the environment where the product was intended to be used.

Cloud Recognition Service: A larger scale version of Image Targets that is meant for enterprise use and can recognize millions of images. Part of a paid add-on service.

Multi Targets: A Multi Target is a collection of multiple Image Targets combined into a defined geometric arrangement such as boxes. This allows tracking and detection from all sides and can serve numerous use cases. Multi Targets functions in such a way that all of the faces of a Multi Target can

be tracked at the same time because they have a shared pre-defined pose relative to Multi Target's origin.

Cylinder Targets: Cylinder Targets enable you to detect and track images wrapped into cylindrical and conical shapes. Vuforia Engine can track the sides and the flat top and bottom of the Cylinder Target.

Model Targets: Model Targets enable apps built using Vuforia Engine to recognize and track particular objects in the real world based on the shape of the object. In order to create a Model Target, 3D model data of the object are required and the object must be both geometrically rigid and present stable surface features.

Area Targets: An environment tracking feature that enables tracking and augmenting areas and spaces.

Ground Plane: Supports the detection and tracking of horizontal surfaces and allows to place virtual content on them or even mid-air.

2.2.6.5 ARToolKit/ARToolKitX

As mentioned previously in this thesis, ARToolKit was developed in 2005 by Hirokazu Kato. It is software library for building Augmented Reality (AR) applications. Some of the features of ARToolKit include: single camera position/orientation tracking, tracking codes that use simple black squares, the ability to use any square marker patterns, easy camera calibration code and fast enough speed for real time AR applications to be viable. ARToolKitX is now onwed by ARToolworks and is an open source project meant to provide continuous support to the ARToolKit community [42].

2.2.6.6 Wikitude

As mentioned previously Wikitude was launched in 2008 by Mobilizy and focused on providing location-based AR experiences through its mobile ap-

plication. It has since then switched focus on providing a robust AR SDK. Wikitude was purchased by Qualcomm in 2021 [43]. It supports development for Android, iOS and Unity platforms.

The current features of the SDK include:

Image Tracking: Wikitude supports detection, tracking and augmentation of 2D images and can track more than one simultaneously.

Cloud Recognition: Cloud-based image recognition for large-scale AR projects. It can be done both online or offline with the latter allowing for up to 1000 images without a network connection.

Object and Scene Tracking: Object Tracking recognizes 3D structures and objects. Those objects have to be previously recorded and embedded into the AR application so that they can be detected based on their pre-recorded map. Scene Tracking advances in recognizing and tracking a wide span of objects and spaces and offers more precise localization for more comprehensive recognition demands.

Multiple Object Tracking: More than one objects can be tracked simultaneously.

Multiple Trackers: Wikitude allows mixing of different types of targets, be they physical objects, environments or images.

Instant Tracking: WIkitude utilizes SLAM technology in order to identify the user's precise location within an unknown environment by simultaneously mapping the area during the Instant Tracking AR experience. This allows for environmental mapping and marker-less AR experiences.

SMART: A Wikitude API that integrates ARKit, ARCore and Wikitude's SLAM engine in a single cross-platform AR SDK that dynamically identifies the users' devices and chooses which SDK will be used.

Cylinder Tracking: Cylinder Tracking allows AR appliations to detect, track

and augment cylinder targets.

2.2.6.7 EasyAR

EasyAR offers three different options EasyAR Sense, EasyAR Mega and EasyAR CRS [44]. EasyAR Sense is an SDK for creating mobile AR applications. It offers features like generation of special maps in real time, motion tracking, surface tracking, 3D object tracking and planar image tracking. Its features are similar to ones discussed previously in this thesis (ARKit, ARCore, etc) so there's no need to go into greater detail. EasyAR Mega is the large-scale counterpart of EasyAR. Meant for city-scale AR cloud solutions. Finally, EasyAR CRS is a web service that enables cloud computing and storage, to provide cloud-based image recognition for large scale databases.

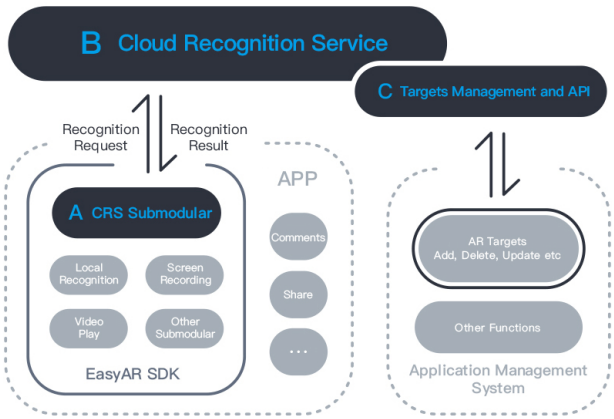


Figure 2.20: Block Diagram of EasyAR CRS.

2.2.6.8 Notable mentions

Lumin: Magic Leap's SDK used to develop AR applications for Magic Leap One and Magic Leap 2.

MixedReality Toolkit: Google's SDK used to develop AR applications for HoloLens and HoloLens 2.

Both these SDKs provide powerful tools to create immersive AR experiences for their respective HMDs. Their features include object and environment tracking, gesture recognition, eye tracking, special mapping, anchors and depth understanding.

In this thesis, ARFoundation was used as our SDK of choice, since the ability to develop a mobile AR application for both Android and iOS devices without the need of separate codebases, significantly cuts down the time needed for the development process and also increases our target audience. It's also worth noting that AR Foundation would be easy to integrate into a Unity (our game engine of choice) project, since both the Unity Game Engine and the AR Foundation SDK are developed by Unity Technologies.

2.2.7 Currently Available Game Engines

A game engine is a software framework designed for the development of video games. The core functionality that is typically provided by a game engine includes rendering (via a 2D or 3D rendering engine), physics (via a physics engine), collision detection, sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graphs, and video support for cinematics. It's apparent that game engines provide most of the required tools for developing a video game and make the process available even to smaller developers. The target platforms for the games include video game consoles, computers or even mobile devices. In addition to game development, game engines can also be used for the development of AR or VR applications.

2.2.7.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies, with its initial release in 2005. The game engine has received numerous updates since then and offers support for a multitude of platforms including video game consoles, mobile devices, computers and web. Unity provides a

physics engine, the Unity Scripting API, both 2D and 3D support and VR and AR capabilities. The Unity Asset Store offers a massive selection of free or paid assets that can be easily imported into any Unity project. Those assets include 2D or 3D models, audio assets, a variety of tools, templates and add-ons that speed-up development. Finally, Unity supports most of the previously mentioned AR SDKs and even provides its own SDK called ARFoundation that combines ARCore and ARKit into one SDK, allowing the developer to write one application that can be ported both to Android and iOS and automatically choose which SDK should be used in each case. This significantly speeds up the development process as there's no need to develop separate applications for those two platforms.

2.2.7.2 Unreal Engine

Unreal Engine is a 3D game engine developed by Epic Games, with its first showcase in the 1998 first-person shooter (FPS) game Unreal. Even though it initially focused on the development of FPS games it now supports a variety of game genres as well as AR and VR applications. Just like Unity it supports a wide variety of platforms including computers, video game consoles, mobile devices and web. The current generation of the engine is called Unreal Engine 5 and was formally released in 2022, replacing Unreal Engine 4. Besides C++ scripting, Unreal Engine features its own scripting language called Verse which was launched this year and is planned to become available to all Unreal Engine users by 2025. Just like Unity, Unreal Engine provides an asset marketplace (Unreal Asset Marketplace). Finally, Unreal Engine has its own AR framework and there's also plug-ins for ARCore, ARKit and other AR SDKs.

In this thesis, we chose to use the Unity Game Engine based on the fact that we already had some familiarity in using it and also because it is a very well established and powerful tool for developing digital applications for a multitude of different platforms. Lastly, Unity Technologies develop and support the ARFoundation SDK which would integrate well with the Unity Game Engine and be the basis of our application's AR capabilities.

2.3 Modern Augmented Reality applications

Now that we have a better understanding of augmented reality's evolution and current state, it's time to examine some actual, real-world applications. This will help illustrate how big the scope of augmented reality's possible use-cases is both as a research, educational, scientific and commercial tool.

2.3.1 Commercial Applications

The applications mentioned in this section showcase how commercially successful augmented reality can be if used creatively. First, we'll see how augmented reality can be integrated into a mobile game to increase the feeling of immersion and motivate the user's to engage in outdoors activity. Then, we'll inspect a practical application that allows the user to test how a product would look when placed in their personal space.

2.3.1.1 Pokemon GO

One of the most popular and well-known AR applications is Pokemon GO, an AR game for mobile phones based on the Pokemon franchise. Following it's release by Niantic in 2016 it quickly became a worldwide phenomenon with millions of players across the world. Even now, in 2023, the estimated active player count of Pokemon GO is 80 million players, with a peak daily player count of 5 million. The AR portion of the game involves a Pokemon being placed virtually into the user's surroundings as he attempts to catch it by throwing PokeBalls at it. The user has to navigate to a specific location through an in game map for this AR experience to trigger for a specific Pokemon.

Figure 2.21: Pokemon GO.

2.3.1.2 IKEA Place

This application, designed by IKEA, functions as an interactive room planner. It allows customers to place virtual versions of IKEA's furniture in their real-world environment. This allows the user to preview how a certain piece of furniture would look in their space before committing to a purchase. This application was developed using Apple's ARKit for iOS devices.

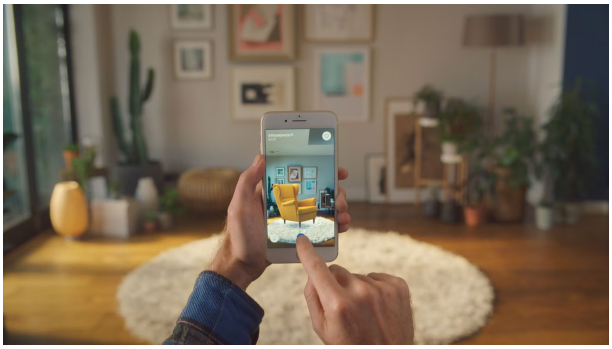


Figure 2.22: IKEA Place.

2.3.2 Academic papers focused on Augmented Reality applications

There's a plethora of different research sectors where augmented reality applications can have positive use cases. Below we'll take a closer look at some such applications.

2.3.2.1 Intelligent Virtual Assistants (IVA) [10] [11]

A common use case scenario for AR is creating a body for an IVA and animating gestures to reinforce its physical presence. One such experiment measured the level of trust that users had for Alexa and the results indicated a trend of positive influence of embodiment on the perceived trust in the IVA [10]. Another similar experiment [11] had users interact with three forms of IVAs: 1) IVA S (Speech Only), 2) IVA SG (Speech and Gesturing), 3) IVA SGL (Speech, Gesturing and Locomotion). They concluded that imbuing the IVA with a visual body and natural social behaviours increased the user's confidence in the agents ability to influence the real world and the level of trust that the agent would respect the users privacy. All in all it appears that IVAs with a natural (albeit virtual) "physical" presence are perceived more positively by users and provide a richer experience and a higher degree

of engagement.

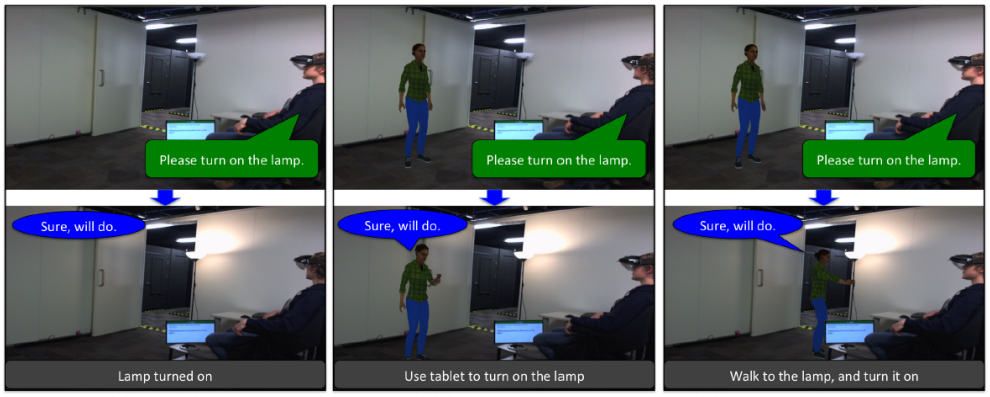


Figure 2.23: Testing the three different forms of IVAs.

2.3.2.2 Design of an AR-Based System for Group Piano Learning [12]

Another interesting sector in which AR can have a positive and major impact is education. AR can help enrich the learning process by increasing the interactivity, providing addition information "on the fly" or allowing the trainee to practice in a risk-free environment. One such application was centered around teaching piano to a group of students [12]. The app had two modes. In the first one the teacher could virtually show his hand movements to the students. The second mode aimed to solve the disadvantage of limited interaction during a normal piano lesson by allowing the students to compete in playing a song.

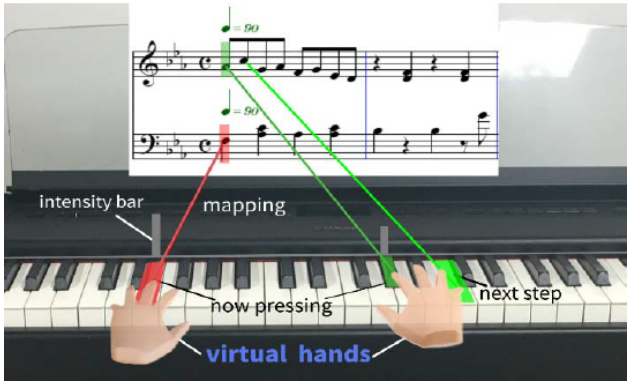


Figure 2.24: Virtual piano lesson from the point of view of a student.

2.3.2.3 A First-Person Mentee Second-Person Mentor AR Interface for Surgical Telementoring. [13]

A second example of an educational AR experience involved a teacher telementoring a trainee on a surgical operation [13]. The system consisted of an overhead camera that transmitted a live feed of the surgical field to the mentor, a touch-based interaction table on which the feed was displayed and an HMD that the mentee was equipped with. The mentor was able to annotate on the video feed, which in turn was displayed on the mentees HMD after being reprojected to match the mentees first person view of the surgical field.

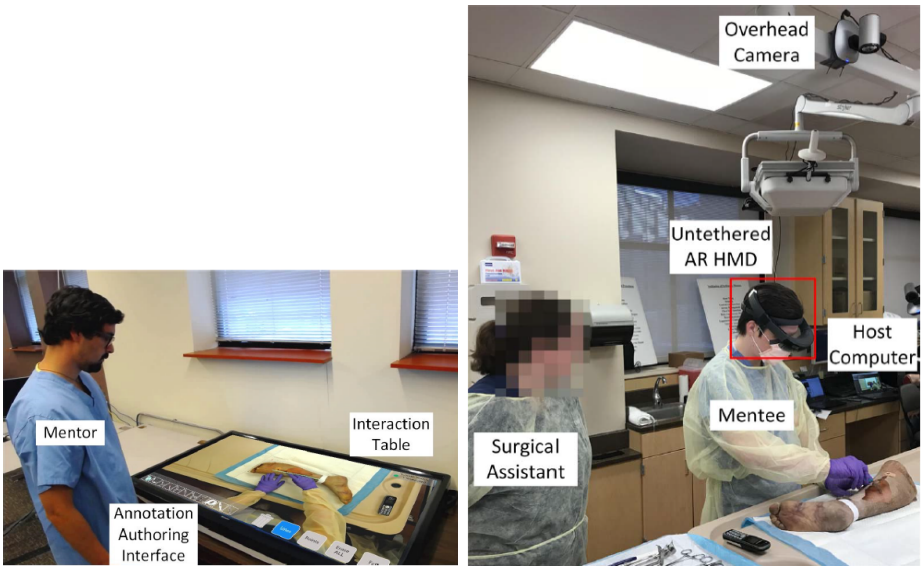


Figure 2.25: Mentor and mentee subsystems for surgical telementoring.

2.4 Outdoors Augmented Reality and Urban Planning

After reviewing the aforementioned real-world applications, it becomes clear that augmented reality can be a powerful tool for aiding and improving sectors such as education, emergency training, personal health and well-being, urban and environmental planning and safer living [32]. One especially interesting category of AR applications are those focused on outdoors usage and more specifically, the process of urban planning. In this section, we'll define what urban planning is and then examine a few outdoors applications of augmented reality.

2.4.1 What is Urban Planning

Urban planning is defined as the process of guiding and directing the use and development of land, urban environment, urban infrastructure, and related ecosystem and human services in ways that ensure the maximum level of economic development, high quality of life, wise management of natural resources, and efficient operation of infrastructures [31]. Its ultimate goal is to create cities that are sustainable, livable and attractive to the residents.

Urban planning has become even more crucial in the modern era of bustling metropolises and extended urban centers. It is imperative we take the time and effort to thoroughly plan the development process of these cities to ensure that they will continue to be viable living spaces for decades to come.

2.4.2 Academic papers on Outdoors AR and Urban Planning

Now that we've defined what urban planning is and what it aims to achieve, let's examine a couple academic papers that pertain to the use of augmented reality outdoors and to aid the urban planning process.

2.4.2.1 3D Outdoor Augmented Reality for Architecture and Urban Planning [17]

In this paper use of AR for urban planning and architecture is explored a more specifically the development of an app that allows the user to view digital

buildings in the real world via a tablet or an HMD. The researchers sum up the various technologies used in AR and divide them into two groups, marker-based and marker-less systems. The proposed system (named "City 3D-AR") falls in marker-less systems category and places virtual three-dimensional buildings in the real world using GPS coordinates. Due to the outdoors nature of the project and the bulkiness of the objects using a marker-based approach wasn't possible.

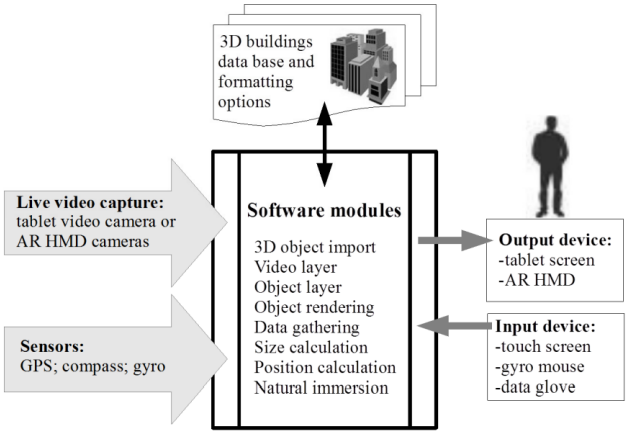


Figure 2.26: City 3D-AR physical structure.

The participants GPS coordinates had to be measured at all times to ensure the 3D objects were subjected to the correct transformations, rotations and scaling to accommodate the participants new point of view. The application utilized the cameras of the device (HMD or tablet) to gain a live video stream of the user's view, the GPS sensor to triangulate the users location in 3D space, the digital compass for sight direction and field of view and the gyroscopic sensor for head orientation. The user was able to interact with the application via a touchscreen or any other portable input device.

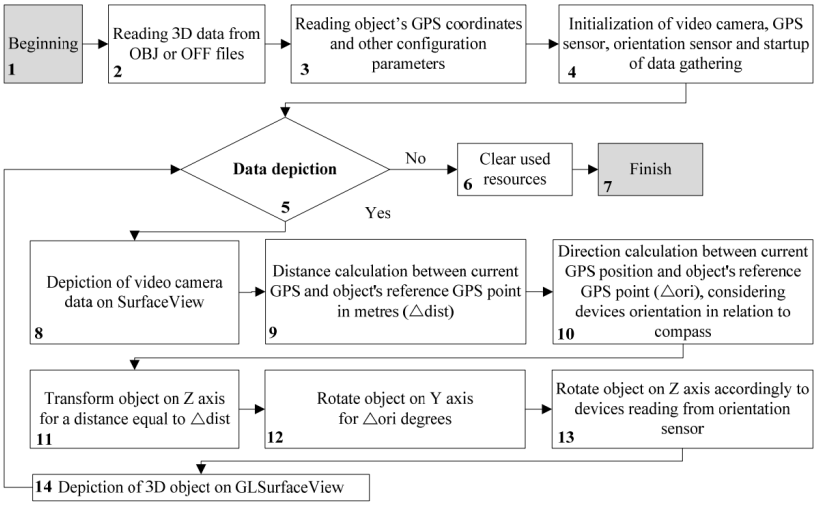


Figure 2.27: City 3D-AR detailed flow chart.

In conclusion, this urban planning AR project allowed the merging of real city and virtual 3D buildings. The authors mention that it's still far from a real use professional application and that validation under urban planning requirements is continuing. But even though there's still room for improvement it proves the possible value of AR in this field.

2.4.2.2 Integration of augmented reality and GIS: A new approach to realistic landscape visualisation [18]

This paper describes the development of an AR application using a geographic information system (GIS) for visualization of weed (blackberries) growth in Victoria, Australia. The system combines GIS with off-line AR to create a representation of the dynamic spread of weeds and their effects on landscape over a period of 14 years.

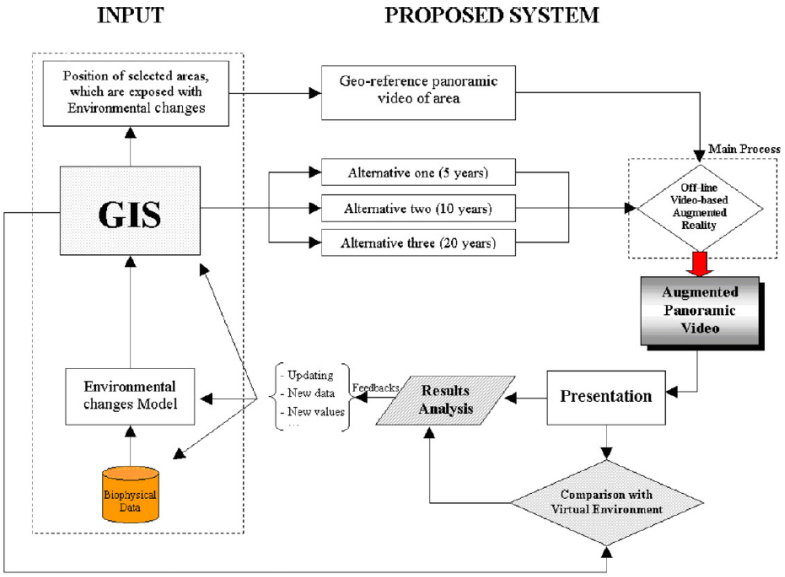


Figure 2.28: Major components of the proposed technique.

The system uses three overlapped video sequences to create a panoramic video which acts as the background of the AR experience when re-projected in a wide-screen theatre. Using a weed spread modelling the spread of the blackberries is computed. This model was run to predict the spread over a period of 14 years and the output is a series of map showing grid cells that have been infested, as well as the size of said infestation. Realistic images of the weeds, corresponding to different growth stages, are then superimposed on the video to match the results of the spread model.

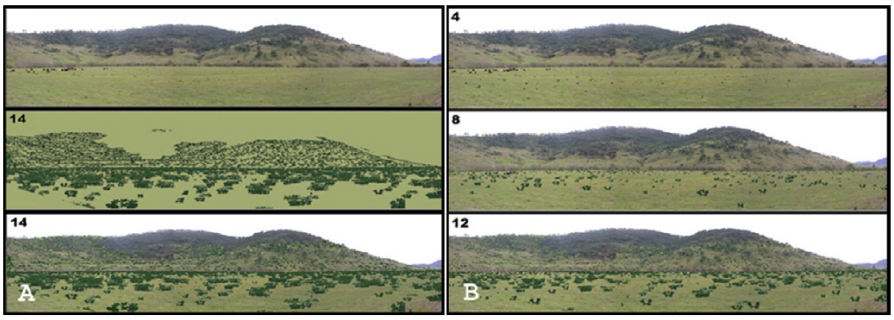


Figure 2.29: Superimposition of weeds over the panoramic video frames; (A) Combination of computer-generated weeds with the background. (B) Several frames of the weeds in different years.

It's important to mention that the proposed method is not real-time, but rather an off-line solution that inserts the virtual elements on prerecorded video, therefore eliminating a majority of tracking issues that a real-time AR application would have to tackle. The authors mention that one the major challenges they faced was the colour mismatch between the frames of each of the three cameras. That and a number of other related problems made the production of a seamless final product very difficult. Lastly, it's mentioned that the proposed technique received positive feedback from informal tests and was more preferable than even highly detailed renderings or emerging real-time (VR) options.

In conjunction to this paper, it's worth taking a look at "A 3D GIS-based interactive registration mechanism for outdoor augmented reality system" [19], where a method for more precise registration combining AR and 3D GIS that does not rely on vision tracking is proposed.

2.4.2.3 Markerless Vision-Based Augmented Reality for Urban Planning [20]

Another paper where an AR system for virtual building placement and view in the real world is developed. This system is markerless and additionally it doesn't rely on inertial sensors or beacon-based localization technologies (e.g., GPS).

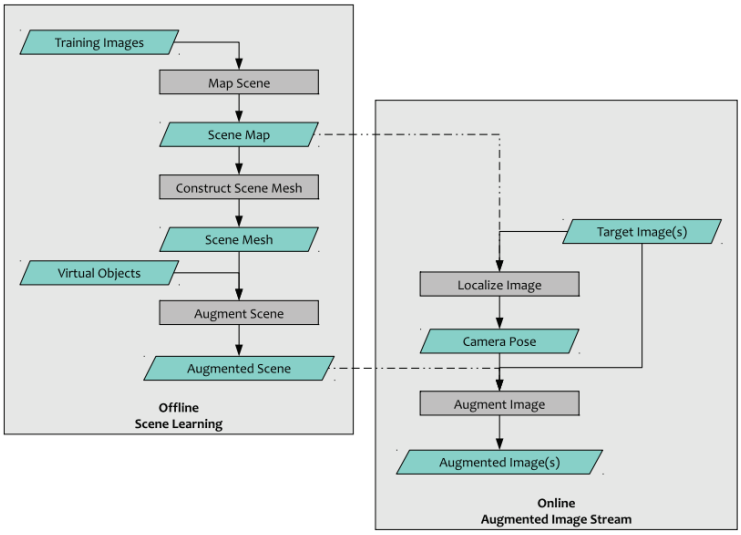


Figure 2.30: Overview of the proposed image-based AR system.

The system is comprised of an offline learning/training stage and an online localization and image augmentation stage. During the offline stage a set of images of the scene of interest are used to build a map of 3D referenced SURF features and a 3D reconstruction of the scene is created in which the virtual objects are inserted.

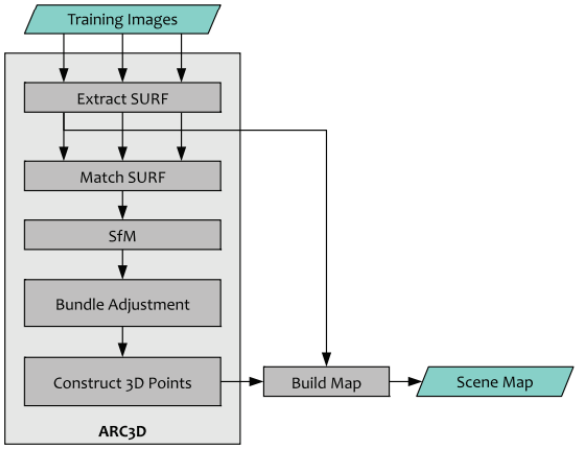


Figure 2.31: Outline of the offline training stage.

The online stage consists of a localization and an image augmenting stage. The system initializes the pose and then tracks or if deemed necessary resets

it. After the pose has been successfully calculated for a target image the image is augmented with a projection of the virtual objects.

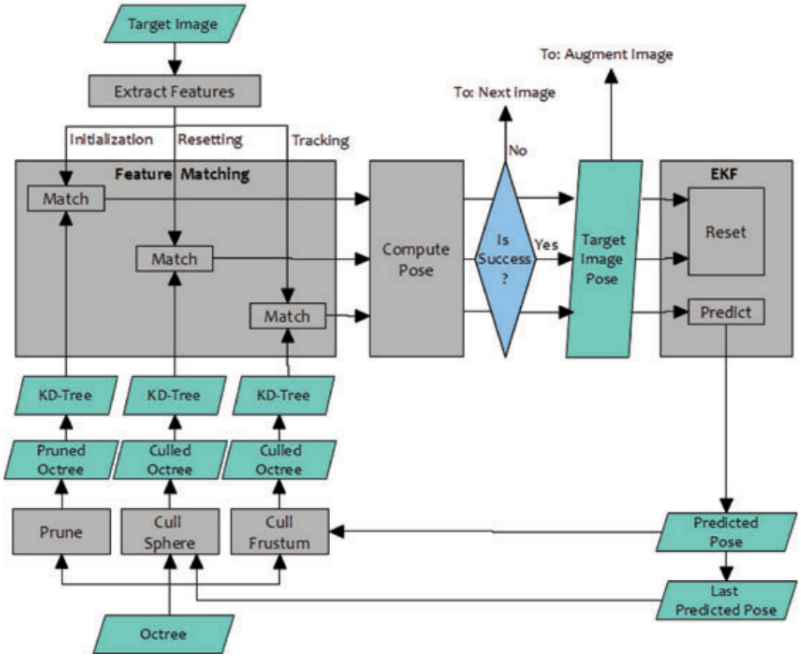


Figure 2.32: Outline of the on-line processing stage.

The experiments results indicate the potential of the proposed system. These experiments are comprised of three datasets of different level of complexity. The researchers also note that the system doesn't achieve a high enough framerate (up to 3fps) to be considered a real-time AR system.

2.4.2.4 Smart-Phone Augmented Reality for Public Participation in Urban Planning [33]

A smart-phone prototype system for visualizing proposed 3D architectural designs on top of existing architecture. The system that was developed consisted of a smart-phone and the software and content needed in order to overlay the designs. The content was 3D architectural models, in 3D studio Max format, which were then processed and converted into a form compatible with the software that was used for AR tracking.

The tracking method used for this application was based on a panorama tracker, which would create a panoramic image of the scene. The drawback of this technique is that the user has to remain stationary while using the system.

Figure 2.33: Overlay on scale model and the Graphical User Interface.

The end user experience was limited to being able to view the different available 3D models, switching between them, calibrating the scene and voting based on which model the user preferred. A user study was conducted to generate quantitative and qualitative data regarding the application which were split into four data analysis categories: mobile device familiarity, user experience, perceived participation and willingness to participate. The feedback the authors received was mostly positive, with users acknowledging that the prototype system had potential as a tool for visualizing architectural designs.

2.4.2.5 An augmented reality study for public participation in urban planning [34]

This study aims to evaluate the suitability of augmented reality as an urban planning visualization tool and also conduct a field study comparing construction spans against a prototype augmented reality system.

Two projects were selected for the field study, each of which was in a different planning stage. The first project used construction spans whereas the second one relied on the AR application to aid in visualizing the building that would be constructed.

Figure 2.34: Renderings of the two projects.

The augmented reality applications was designed for iOS, using ARKit, and featured three different levels of detail (LOD) for the AR building.

The study conducted consisted of groups of users randomly split into three groups, one for each different LOD. One surprising result is that the users from the lowest LOD group rated the system the highest in terms of realism. This may be caused by the lack of motion lag that was present in the other two LODs or by the small screen-size of the device used for the evaluation. Another noteworthy result is the fact that LOD 3 rated the highest in terms of ease of visualizing what the finished building would look like. Overall, the results showed that augmented reality was highly suitable for the task at hand and shows great potential for visualizing architectural projects.

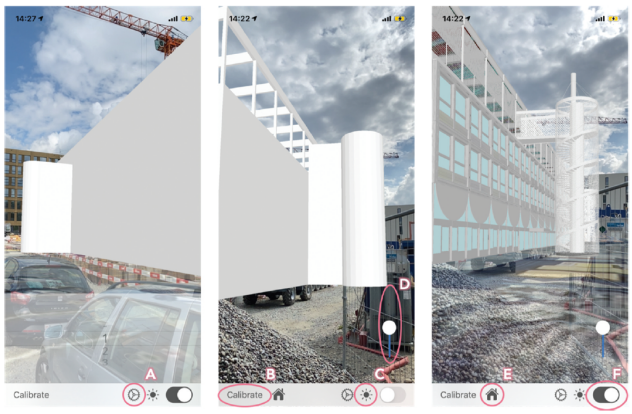


Figure 2.35: Prototype AR application displaying different levels of detail (LOD).

In this thesis, we will be combining augmented reality with urban plan-

ning to develop a mobile application for visualizing urban planning changes outdoors. This application aims to act as an supporting tool both for the planning process required by engineers and architects but also as a participatory tool for the citizens. Similar to the applications we examined in the sections above, our goal is to overlay various possible virtual urban planning changes onto the real-world and allow the user to interact with them, customize them and finally view the end-result of their experimentation.

Chapter 3

Requirements and Technological Background

3.1 Introduction

In this chapter, we will give an overview of all the development tools used in the creation of our application as well as an analysis of the case of Nea Chora from an urban planning perspective. First we will go over what the Unity Game Engine is and define some of its core concepts and terms. Next, we'll introduce the various SDKs that were selected and used in our implementation of the application, as well as any additional software that was utilized. Lastly, we'll provide a detailed analysis of the case of Nea Chora, Chania, how it could be reconstructed from an urban planning perspective and the areas that were chosen for our application.

Upon finishing this section the user should have attained a better understanding of the technical background of our application, which will significantly aid his understanding of our implementation. They should also have a clear image of the current state of Nea Chora and why it would significantly benefit from an urban planning reconstruction.

3.2 Unity

As mentioned previously, Unity is a cross-platform game engine developed

by Unity Technologies. It provides a physics engine and allows for the development of both 2D and 3D applications, as well as AR or VR applications.

Unity was chosen for this thesis since it's a very robust game engine that provides a plethora of useful features, add-ons, external package support, as well as it's own AR SDK called AR Foundation. These featured accompanied by an existing familiarity with the game engine made it the perfect candidate for this thesis.

3.2.1 Assets

The term asset describes any file used by Unity such as scripts, prefabs, materials, etc.

3.2.2 Prefabs

The term prefab originates from the word prefabricated and in essence represents a pre-constructed object or model that has been saved so that it's easy to re-use in multiple parts of the application. One major advantage of using prefabs is that modifications made to the original one apply for all used copies of it. Additionally, it is possible to create separate versions of a prefab using the original as a base.

3.2.3 Materials

Materials are definitions of how a surface should be rendered, including references to textures used, tiling information, color tints and more.

3.2.4 Scenes

Unity Scenes can be described as assets that contain all or a part of a game or application. For more complex application, scenes usually correspond to a level or a specific screen. A scene may contain a multitude of other assets, such as scripts, UI components or prefabs. In our AR application each scene is a different screen that the user may interact with while using the app.

3.2.5 Scripts

Scrips are how we apply functionality to our scenes and objects. They can be attached as components to other unity assets. The scripting language used

for our application was C# and the integrated development environment (IDE) used is Visual Studio Community Edition.

3.2.6 Packages

Packages contain various features that can be integrated into a Unity project. They can be official Unity releases (eg. AR Foundation) or external third party packages (eg. Mapbox).

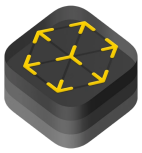
3.3 AR Foundation

AR Foundation is a Software Development Kit (SDK), designed by Unity Technologies, focused on creating multi-platform augmented reality (AR) applications using the Unity game engine. AR foundation utilizes the corresponding native AR SDK based on the device currently being used, thus significantly simplifying the process of developing applications that are compatible over multiple platforms, since it eliminates the need to have separate applications developed for each platform. The two SDKs that are in particular interest for this thesis are Google's ARCore, the native SDK for Android devices, and Apple's ARKit, the native SDK for iOS devices. Thus, by utilizing AR Foundations features we can develop and application that will function both on Android and iOS devices while having a single codebase for both of platforms. ARFoundation is the AR SDK that we selected for the development of our application.

3.3.1 ARCore

As mentioned before ARCore is Google's AR SDK for Android devices. The Google ARCore XR Unity Plug-in is required for AR Foundation to function correctly on Android devices. This plugin's features include device tracking, plane detection, image tracking, face tracking, point cloud detection, raycast capabilities, anchor creation and tracking and AR object occlusion.

3.3.2 ARKit



As mentioned before ARKit is Apple's AR SDK for iOS devices. The Apple ARKit XR Unity Plug-in is required for AR Foundation to function correctly on iOS devices. The ARKit plug-in offers similar features to those of ARCore.

3.4 ProBuilder

ProBuilder is a hybrid of 3D modeling and level design tools, optimized for building simple geometry but capable of detailed editing and UV unwrapping. ProBuilder can be integrated into the Unity Editor, which allows us to create and edit geometry without the need for external software (eg. Blender 3D, Autodesk Maya, Autodesk 3ds Max). ProBuilder offers a variety of features for 3D modeling, while remaining relatively easy to learn.

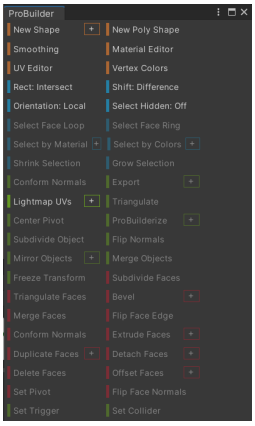


Figure 3.1: The ProBuilder window, featuring various different editing options.

ProBuilder was used in order to create some of the virtual objects used in our AR application, namely the virtual building (and it's numerous components), the bicycle lane model and the placement marker.

3.5 Krita

Krita is a free, open source painting software. It started as the Krita Project by the KDE community in 1998 and has since there received continued support and development. In 2012, the Krita community created the Krita Foundation, which supports and updates Krita to this day.

Krita was used in this thesis to create the various user interface (UI) sprites that appear in our mobile application.

3.6 Visual Studio Community 2019

Visual Studio is an integrated development environment (IDE) developed by Microsoft. It integrates with Unity via the Code Editor Package for Visual Studio, which comes pre-installed with our Unity version. Additionally, when installing Unity, we can opt to also install Visual Studio, which sets it as the default option for opening and editing scripts.

Visual Studio Community 2019 was used for the development and editing of the scrips associated with our application.

3.7 Map SDKs

In the subsections below we'll take a look at a few available Map SDKs that we considered using in order to implement a map navigation functionality for out application. The SDK that we ended up choosing is Mapbox.

3.7.1 Mapbox

Mapbox provides map SDKs for a variety of platforms, including Unity, Android and iOS. The Mapbox Unity SDK is a collection of tools for creating custom maps, integrating real world maps to applications and providing real-time navigation via the Mapbox Navigation application programming interface (API). This SDK is completely free to use and has paid tiers only after a certain amount of users and API requests are reached, which is high enough to not be a concern for us while developing the application (100,000 free temporary Geocoding API requests, 100,000 free Directions API requests and 25,000 free monthly active users).

3.7.2 Google Maps

Another option that was initially considered but quickly dismissed was using the Google Maps SDK. Sadly the Google Maps SDK for Unity had been deprecated. Quoting their website on deprecations "Since the introduction of our gaming services in 2018, we were proud to launch innovative new game experiences for developers. While we remain dedicated to supporting the developer community, we've decommissioned Google Maps Platform gaming services as of December 31, 2022 due to limited adoption."

3.7.3 ArcGIS

ArcGIS seemed like another promising option since it offers features very similar to those of Mapbox. However, their only free tier was a 21-day trial which wouldn't be viable nor accommodate the needs of our application and development process. Therefore ArcGIS was also rejected.

3.8 Hardware

Two computers were used for the development of the applications and two Android mobile phones were used for its testing.

The first computer was a desktop with the following specifications: *CPU:* Intel Core i7-4790K 4.40GHz, *GPU:* Asus Nvidia GeForce GTX 970 Strix 4Gb, *RAM:* 16GB DDR3 2133 MHz.

The second computer used was a laptop with the following specifications: *CPU:* Intel Core i5-10300K 2.50GHz, *GPU:* Nvidia GeForce GTX 1650 4Gb, *RAM:* 32GB DDR4 3200MHz

The two Android mobile phones used were 1) a Motorola Moto G 5G and 2) a Samsung A40. Sadly, we didn't have an iOS device available, but we expect that our application would function correctly on that platform as well.

3.9 Case Analysis

In this section we will present a thorough analysis of the case of the district of Nea Chora, Chania, from an urban planning perspective. Then we will go over the proposed urban planning reconstruction of the area. Lastly, we will present the three locations that were chosen for our application.

3.9.1 Urban reconstruction of the district of Nea Chora in the city of Chania

This is a thesis [21] from the School of Architecture, Technical University of Crete (TUC), in which a reconstruction of the district of Nea Chora in the city of Chania is proposed. Nea Chora is located in the western section of the city, outside of the old Venetian wall. According to the authors this location was specifically chosen because the buildings there are predominantly residential, which leaves the area devoid of other amenities, such as parks, commercial areas or sports facilities.

3.9.2 Current State of Nea Chora

The majority of the existing buildings have one to four stories, with very few having up to five. The land usage in the district of Nea Chora is as follows:

- *Residencies*: Most of the buildings are residential.
- *Educational facilities*: There's eight (8) kindergartens, three (3) primary schools, one (1) middle school and one (1) high school.
- *Sports facilities*: There only exist two facilities of hyperlocal use, the Municipal Swimming Centre and the Municipal Sports Center of Kladissos.
- *Recreational facilities*: Mostly on the coastal part of the region.
- *Commercial facilities*: Along the streets Selinou and Kissamou.
- *Parking spaces*: Few, close to the recreational facilities.

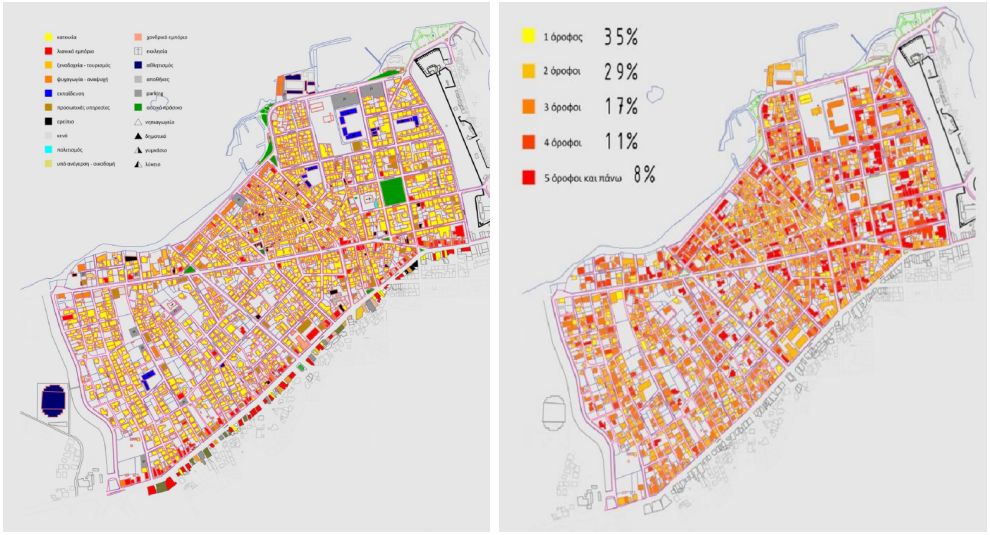


Figure 3.2: Current land usage of Nea Chora.

The district can be divided into three zones: Zone A: configured based on urban planning, Zone B: not configured based on urban planning, Zone C: under reconstruction.

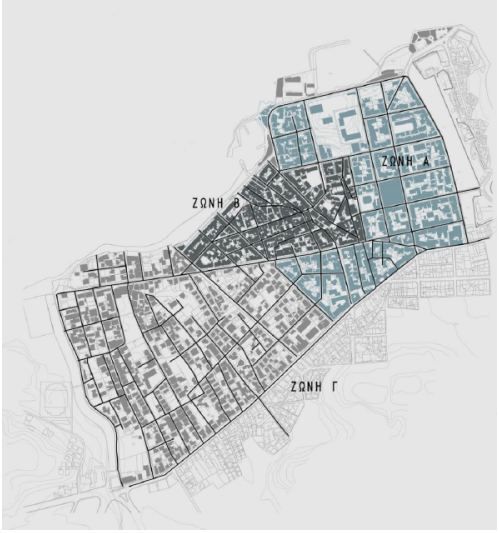


Figure 3.3: The 3 zones. Zone A: light blue, Zone B: black, Zone C: gray.

The road arteries can be categorized as: main streets, collector roads and local roads. There is a lack of pedestrian lanes and cycling paths and vehicles still pass through the few existing ones. Lastly, there's no designated parking spaces.

Figure 3.4: Current road arteries.

3.9.3 Proposed Urban Reconstruction of Nea Chora

The proposed urban reconstruction of Nea Chora [21] [22] aims to turn the district into a more viable and functional area. The proposal involves a reconstruction and redistribution of residential areas, an increase in green spaces, an overhauled road system that would declutter the area, more parking spaces, commercial hubs, new athletic facilities and entertainment areas.

The district would be divided into three neighborhoods, each of which would be accommodated by a neighborhood center. The first neighborhood would have a commercial-administrative center, the second one a cultural center and the third another commercial center.

The center of the first neighborhood would be the most important one due to its position. It would include the following:

- Various commercial stores (e.g. bookstores, clothing, food)
- Cafes
- A patisserie
- A Citizen Service Center
- A post office

The center of the second neighborhood would include:

- A library
- An elderly care/pastime center
- A multipurpose center for children and young adults
- A couple of essential food stores

The center of the second neighborhood would include:

- Various commercial stores (e.g. bookstores, clothing, food)
- A cafe
- A cafe/reading room

New residential buildings would have to be constructed in previously empty lots to account for the newly created neighborhood centers, since existing buildings there would be re-purposed.

The coastal portion of the district would retain its touristic character and its western side would be redesigned for 60% touristic coverage and 40% residential.

Figure 3.5: Proposed land usage.

The road artery system remains mostly the same, but the usage of each of them changes to accommodate safer transportation both for pedestrians and vehicles. New roads will be created at the western end of the district (Zone C) and in the central area (Zone B).

Figure 3.6: Proposed road system.

A new green spaces network that's immediately connected with the new pedestrian lanes will be created, divided into both public and private green areas. Some of the previously empty lots as well as new empty spaces, created after the demolition of old unstable buildings, will be re-purposed as part of this network. The public green spaces will include parks, athletic stadiums, playgrounds or public squares.

Figure 3.7: Proposed green spaces.

3.9.4 The target areas

After examining the proposed urban reconstruction of the area and also consulting Associate Professor of the School of Architecture, in the Technical University of Crete, Dimelli Despina we ended up choosing three areas as

the target locations of our application.

One of the dominant criteria for picking the target areas was the fact that we wanted to create a diverse augmented reality experience and as such each selected area should provide a unique and different urban planning scenario. The areas should also feature flat terrain, without many obstacles, to ensure that plane (ground) detection would be successful. Lastly, the areas and the content placed in them should conform with the local laws regarding urban development and construction and also follow the urban reconstruction plan that was analyzed in section 3.9.3.

3.9.4.1 Area 1: The coastal sidewalk in Akti Kanari

The first area that we selected is located along Akti Kanari Street of Nea Chora. This road features a long coastal sidewalk that passes right next to the Chania National Nautical-Athletic Center. The exact coordinates of the area, as it was used in our application, are: 35.51095910591866 N, 24.004459975858733 E.

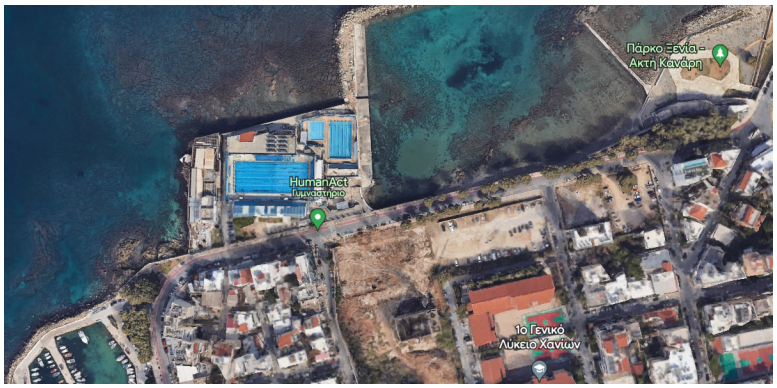


Figure 3.8: Aerial view of Area 1.

This area was selected as our target for placing trees, benches and street lamps, since 1) it features flat terrain and 2) it appears in the proposed green spaces of the urban reconstruction of Nea Chora. Due to its coastal nature and the existing, but lacking, sidewalk infrastructure this area is a perfect pick for creating a unique AR experience for our application.

3.9.4.2 Area 2: Empty lot in Ouellingkton Street

One of the very first ideas we had about the type of virtual content that we

could present was that of a virtual building. In order to do so we needed to find a flat and spacious enough area, so that our building model would fit. The task at hand was more difficult than we expected, since our search (both by foot and by use of the Google Maps Street View) showed that most of the empty lots in Nea Chora are infested with wild plants, fenced or simply not flat or big enough. Eventually we were able to locate a location that could accommodate our building in Ouellingkton Street. This location is currently used as a parking lot for a local gym, but after a brief discussion with the manager we were given permission to use it for our application and conduct any needed testing there.

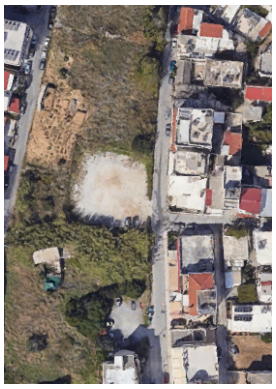


Figure 3.9: Aerial view of Area 2.

It's also worth noting that this area appears in the urban reconstruction proposal and is located in-between an area that would be allocated for residential buildings and one that would house a school. In either case a building would have to be constructed in this location, so it was perfect for our needs.

The exact coordinates of the area, as it was used in our application, are: 35.517482361372736 N, 24.01108550473443 E.

3.9.4.3 Area 3: The intersection of Selinou Street and Georgiakakidon Street

This area is located at the intersection of two major streets of Nea Chora, Selinou Street and Georgiakakidon Street, near the area of Kladisos. The exact coordinates of the area, as it was used in our application, are: 35.5118184719792 N, 24.001774574658594 E.

Figure 3.10: Aerial and street view of Area 3.

We chose to display a virtual bicycle lane and a bicycle station in this location since that is an amenity that is lacking from this area and it would be an interesting augmented reality scenario for our application. Additionally, the proposed road system of the reconstruction of Nea Chora features a bicycle lane in those two streets, which doesn't come as a surprise considering that they are two of the largest and central road arteries of the district. We should also mention that, by definition, streets provide long stretches of flat surface, which would significantly aid our plane detection.

3.10 Thesis Requirements

Our goal is to provide an interactive, mobile augmented reality experience to the user while visualizing the proposed urban planning changes in the district of Nea Chora. To do so our application would have to conform to the following requirements:

- The application should be able to access GPS coordinates, both to calculate the user's distance from each target location but also in order to display the user's position on the map.
- The application should provide a map of the user's surrounding area and navigational instructions on how to reach each target location.
- The UI of the AR locations screens should be minimal, in order to allow the user to view the scene that they're creating.
- The application should be able to access the mobile devices camera.

- The user should be able to interact with the virtual content.
- The content presented in each location should conform with existing urban planning laws and guidelines.

Chapter 4

End User Experience

4.1 Introduction

In this chapter, we will go through every screen of our application from the point of view of our end user. We will explain all the possible user interactions in each screen and also provide use-case scenario diagrams for them.

4.2 Android Permissions

In order for the application to function correctly two types of Android Permissions will need to be granted: 1) Location Service Permissions and 2) Camera Permissions.

4.2.1 Location Service Permissions

Upon first opening the application, the user will be asked to grant location service permissions. This type of permission is essential since we rely on GPS coordinates in order to calculate the user's distance from each of the AR locations. Additionally, we need GPS access to provide a map of the user's area and navigational instructions.

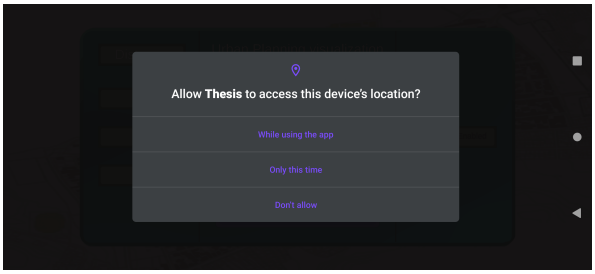


Figure 4.1: Location service permissions.

4.2.2 Camera Permissions

The first time the user enters one of the AR location screens, they will be prompted to grant camera permissions to the application. Without these permissions the AR component of the application cannot function, since it relies on utilizing the mobile devices built-in camera.

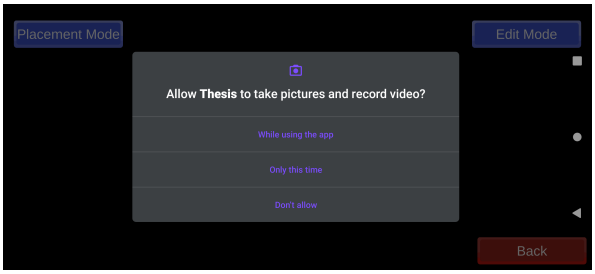


Figure 4.2: Camera permissions.

4.3 Home Screen

The Home Screen is the first screen the user enters after being prompted to grant location service permissions to the application. This screen acts as a main hub for the application, through which the user can navigate to the AR location screens, the map screen or enable the tutorial system.

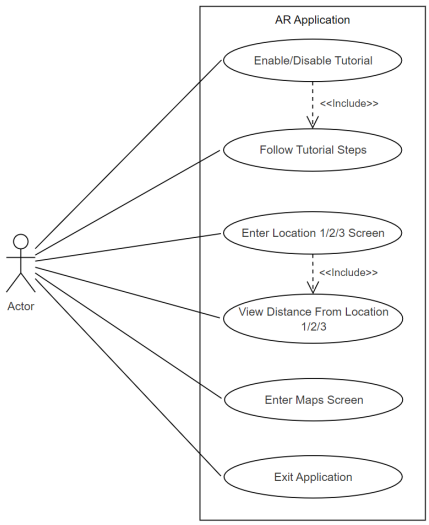


Figure 4.3: Home Screen use-case diagram.

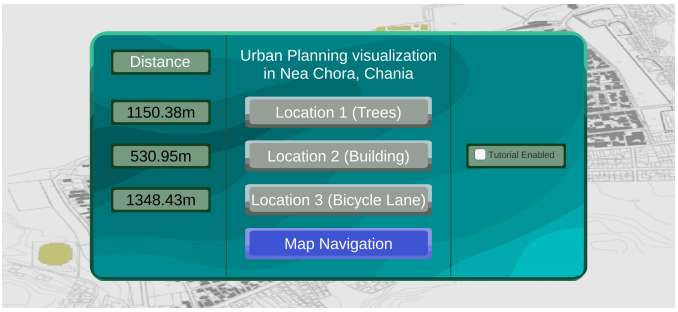


Figure 4.4: Home Screen.

By default, when entering the Home Screen after opening the application for the very first time, the tutorial system will be enabled. The tutorial can be disabled or enabled by checking or un-checking the "Tutorial Enabled" box. It is worth noting that the application "remembers" the user's selection the next time the application is opened, as well as the exact step of the tutorial that they left off. This ensures that the user is able to take a break from the application without having to re-do the entire tutorial. If the tutorial is disabled, then it is reset and has to be started from the very beginning.

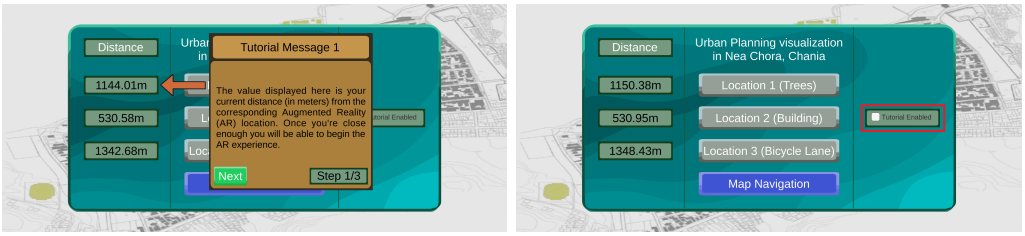


Figure 4.5: Tutorial Message example and Enabling/Disabling the tutorial system.

In this screen the user can view his current distance from each of the AR location screens. The buttons that navigate to the corresponding AR screens are disabled (gray) until the user is within a certain distance from a specific location. Once the user is close enough, the button associated with the location will be enabled (blue) allowing them to transition to that AR screen.

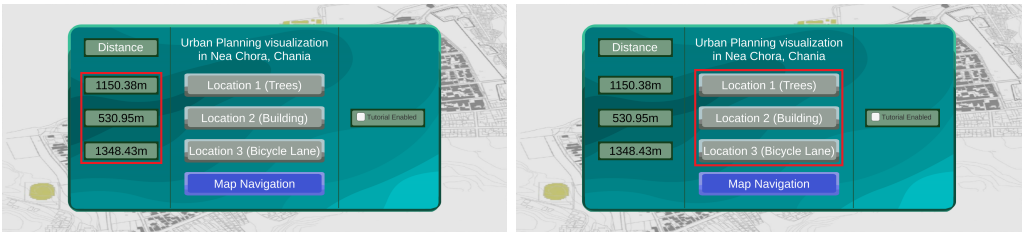


Figure 4.6: The current distance from each AR location and the AR location buttons.

Lastly, the user may also press the Map Navigation button to be redirected to the Map Screen, where they can view a map of their surrounding area and receive directions on how to reach their destination.

4.4 Map Screen

The Map Screen acts as a navigational tool, guiding the user to the AR locations of our application. If the tutorial is enabled, the corresponding messages for the map screen will appear. Otherwise, the user will immediately be able to interact with a map of their surrounding area.

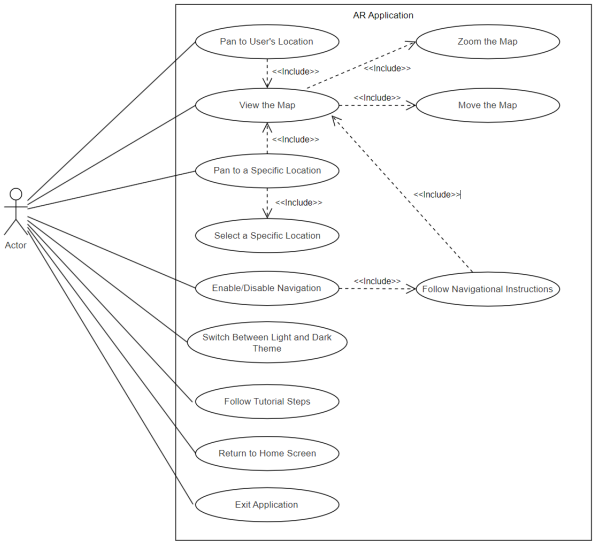


Figure 4.7: Map Screen use-case diagram.

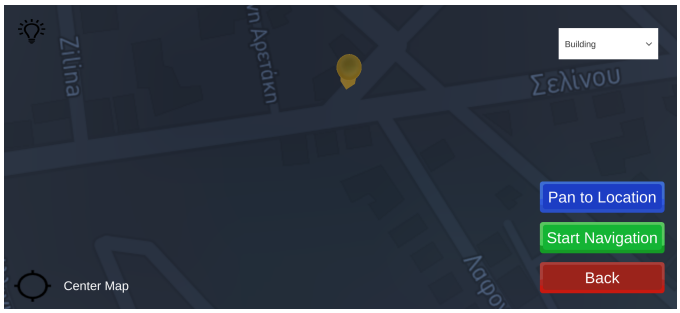


Figure 4.8: Map Screen.

The user can move or zoom the map, via dragging on the screen with their finger or pinching it, respectively. They may also pan back to their current location by pressing the corresponding icon. Another possible interaction consists of the user selecting one of the AR locations as their destination (from a dropdown list) and then pressing the "Pan to location" button to have the AR location be immediately shown on their map. All three AR locations are marked on the map with a destination icon.

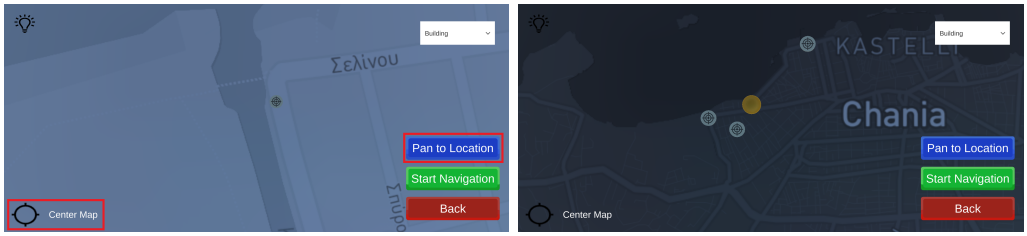


Figure 4.9: Panning back to the user's current location or to a destination and viewing all target-locations on map.

If the user wishes to be shown navigational instructions, they may enable them by pressing the "Enable navigation" button. Once enabled, a navigation line connecting the user's current location with the selected AR location will appear to help guide the user. Lastly, there is an option to switch between Light and Dark Theme for the map.

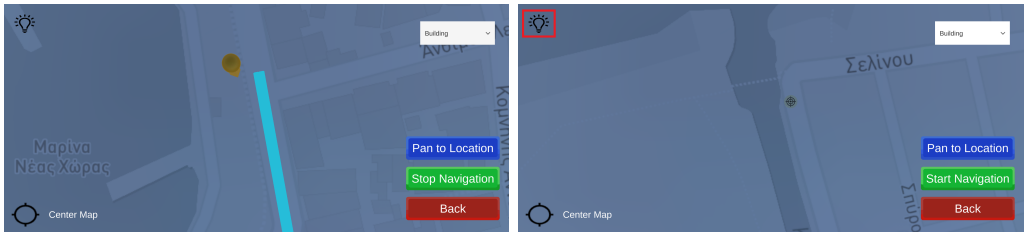


Figure 4.10: Navigational instructions and Switching Map Themes.

4.5 AR Locations Screens

Our application features three location screens, one for each of the different target areas that we have selected. In each of these screens the user can place and edit virtual content onto their real-world environment. We have split this functionality into three distinct modes:

- View Mode: This mode features minimal UI to allow the user to view the scene that they have created.
- Placement Mode: In this mode the user is tasked with scanning the ground of their environment, selecting which virtual objects to place and then placing them at a desired location on the ground.
- Edit Mode: This mode allows the user to interact with the virtual content that they have already placed.

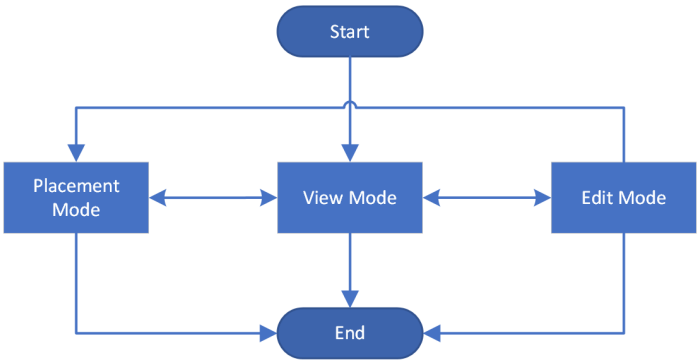


Figure 4.11: Location Screen Flow Diagram.

4.5.1 Location 1 Screen (Trees, Benches and Street Lamps)

The Location 1 Screen corresponds to our first target area (section 3.9.4.1), in which the user will be able to place virtual trees, benches and street lamps. Once again, if the tutorial is enabled the user will be greeted with a series of tutorial messages.

4.5.1.1 View Mode

The default mode, when entering one the AR location screens, is the View Mode. From this mode the user can switch to any of the other two. Once some virtual objects have been placed, the user may return to the View Mode in order to take a look at the scene they created without having their screen cluttered by UI. They may also choose to enable or disable object occlusion (which allows virtual objects to be occluded by real-world ones).

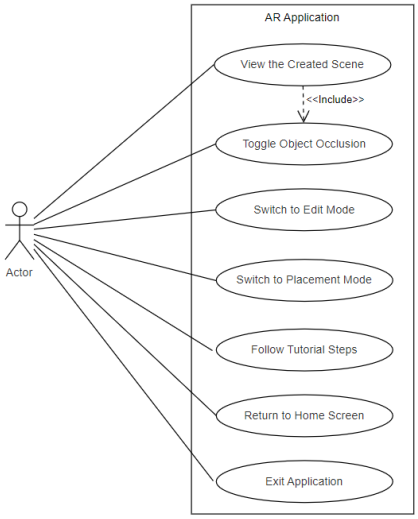


Figure 4.12: Location 1 View Mode use-case diagram.

Figure 4.13: Location 1 View Mode.

4.5.1.2 Placement Mode

This is where we attempt to detect the real-world floor and then place virtual objects on it.

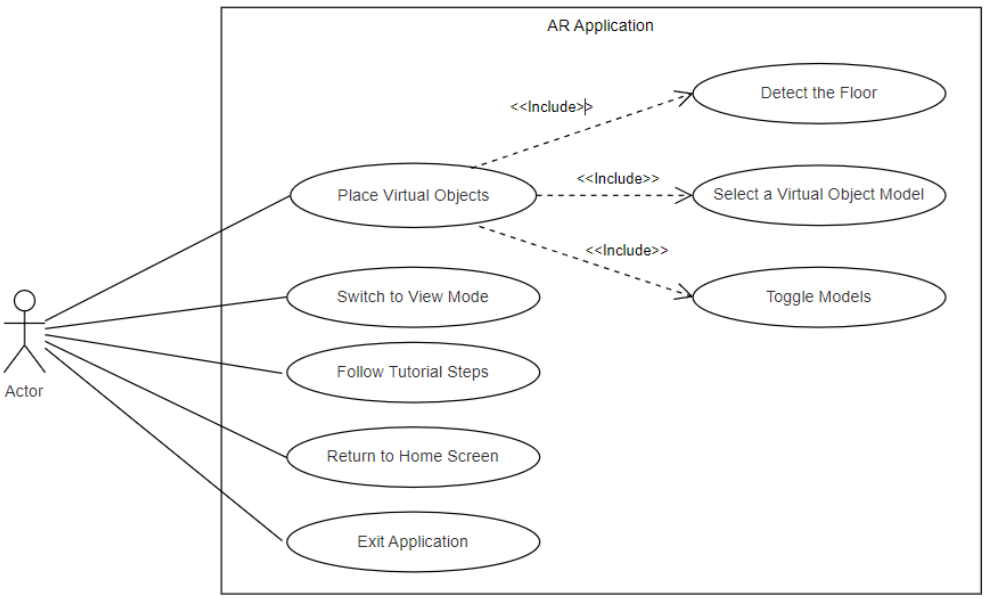


Figure 4.14: Placement Mode use-case diagram.

The process of detecting the floor begins automatically when entering this mode. The user will have to move their mobile device's camera around in order to detect different parts of the ground. Once the floor has been detected, a placement marker will appear in the middle of the user's screen, attached to the floor plane. This marker shows where the virtual content will be placed and can be replaced by the actual model if the user selects the "Toggle Models" option.

Figure 4.15: Detecting the floor.

Once the desired model (there are six different tree options, five bench options and five street lamp options) has been selected from the dropdown list the user can then place it on the ground and begin creating their AR scene.

Figure 4.16: Placing a virtual tree, bench and street lamp.

After the user is done placing virtual content they can switch back to the View Mode.

4.5.1.3 Edit Mode

The Edit Mode is where the user can interact with the virtual content that they have already placed. There are two interactions that exist regardless of the specific location: 1) Deleting an object and 2) Rotating an object. In the case of the first location the user may also change the scale of the placed virtual trees.

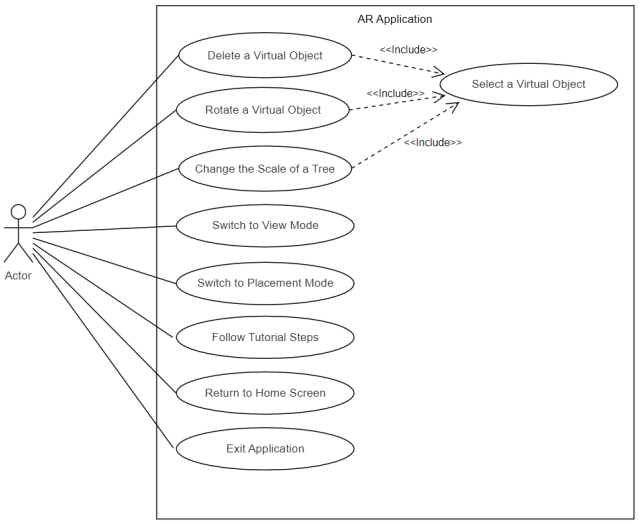


Figure 4.17: Location 1 Edit Mode use-case diagram.

Before beginning to edit a virtual object, the user has to first select it. This is done simply by pointing their mobile device's camera towards the object so that it appears in the middle of their screen. Once an object is selected, a white outline will appear around it and the UI elements corresponding to editing it will become active. The user may then perform any of the available editing operations.

Figure 4.18: Location 1 Edit Mode and selected object outline.

Deleting and rotating an object are achieved via the corresponding button press. Changing the scale of a tree is achieved by moving the slider to increase or decrease the scale of the selected tree.

4.5.2 Location 2 Screen (Building)

The Location 2 Screen corresponds to our second target area (section 3.9.4.2), in which the user will be able to place a virtual building. Once again, if the tutorial is enabled the user will be greeted with a series of tutorial messages.

4.5.2.1 View Mode

The View Mode of Location 2 functions identically to that of Location 1.

4.5.2.2 Placement Mode

The main differences between the Placement Mode of location 1 and 2 are the options given in the dropdown list for selecting virtual objects. Location 2 features six building models, each of them with a different number of floors (ranging from 1 to 6).

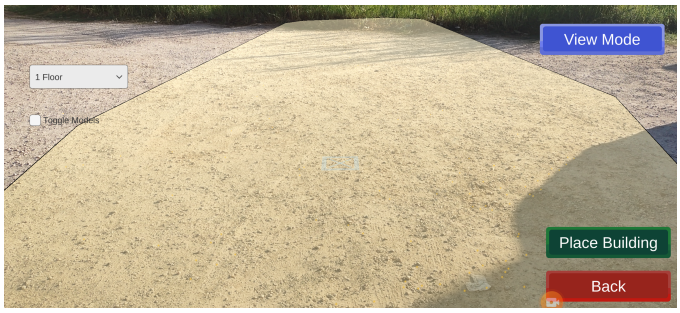


Figure 4.19: Location 2 Placement Mode.

Figure 4.20: Placing a virtual building.

4.5.2.3 Edit Mode

Just like in location 1, virtual objects can be deleted and rotated. Additionally, the number of floors that the placed building has can be changed (ranging from 1 to 6).

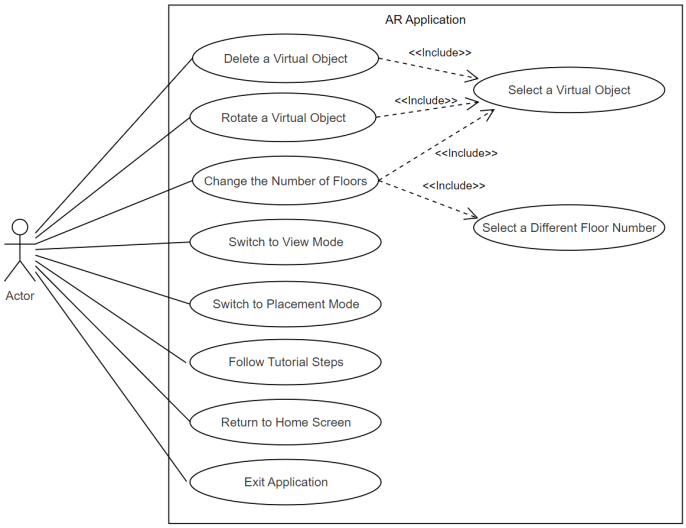


Figure 4.21: Location 2 Edit Mode use-case diagram.

After selecting the building whose floors number we want to edit, we select a different number of floors from the dropdown list and the click the replace button to commit our changes to the scene.

Figure 4.22: Location 2 Edit Mode and Floor number.

4.5.3 Location 3 Screen (Bicycle Lane and Bicycle Station)

The Location 3 Screen corresponds to our third target area (section 3.9.4.3), in which the user will be able to place a virtual bicycle lane and a virtual bicycle station. Once again, if the tutorial is enabled the user will be greeted with a series of tutorial messages.

4.5.3.1 View Mode

The View Mode of Location 3 functions identically to that of Location 1.

Figure 4.23: Location 3 View Mode.

4.5.3.2 Placement Mode

The main differences between the Placement Mode of location 1 and 2 are the options given in the dropdown list for selecting virtual objects. Location 3 features a bicycle lane model and a bicycle station model.

Figure 4.24: Location 3 Placement Mode.

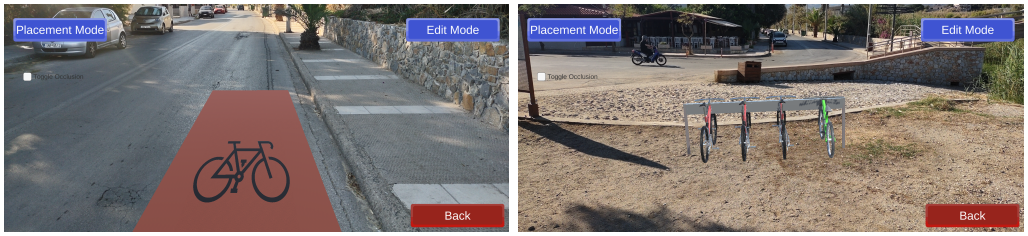


Figure 4.25: Placing a virtual bicycle lane and a virtual bicycle station.

4.5.3.3 Edit Mode

Just like in location 1, virtual objects can be deleted and rotated. Additionally, the user can add or remove bicycles from any placed bicycle station.

Figure 4.26: Location 3 Edit Mode use-case diagram.

After selecting a bicycle lane, the option to remove and add bicycles will become available. Removing bicycles is very simple, since it only required the press of a button. Adding bicycles requires the user to select a bicycle model from the available options so that it can be added to a free slot of the bicycle station.

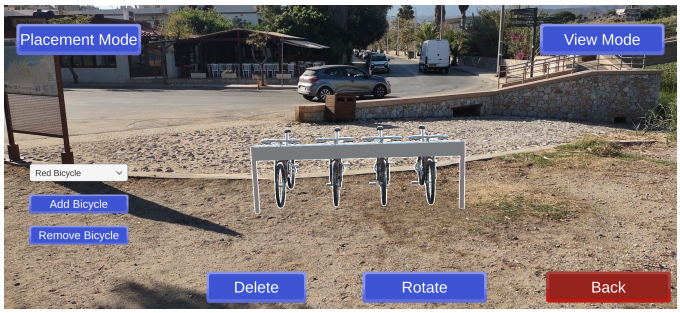


Figure 4.27: Location 3 Edit Mode.

Chapter 5

Implementation

5.1 Introduction

In this chapter we'll explain the technical implementation of our application. We'll go over each scene and its core components and examine how they work. We'll start with an overview of how to set up the basic SDKs and packages needed for developing an AR application with the Unity Game Engine and then delve deeper into how our map system, tutorial system, UI and AR functionality and interactivity was designed and implemented.

Various code snippets and other media will be shown to help give the reader a better understanding of the technical aspects of our application's implementation. We hope that by the end of this chapter the user will have a deeper understanding of how our application was developed from a more technical standpoint.

5.2 Unity

The version of Unity used for the development of this application was version 2022.3.23f1 which is one of the current long term support (LTS) versions of the Unity game engine. Additionally, the following two modules had to be installed in order to be able to build the application for Android and iOS devices: the Android Build Support module (with Open JDK and Android SDK & NDK Tools) and the iOS Build Support module.

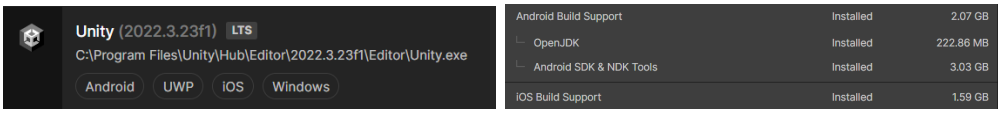


Figure 5.1: Unity version and modules.

5.2.1 Setting Up AR Foundation

The first step we have to take before beginning to develop an augmented reality application in Unity is to set up AR Foundation. To do so we must import the following packages to our project: Google ARCore XR Plug-in, Apple ARKit XR Plug-in and AR Foundation.

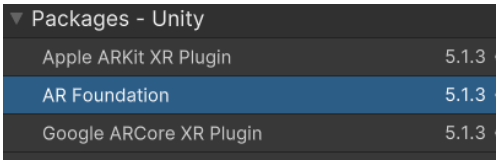


Figure 5.2: Required packages for AR Foundation.

5.2.2 Setting UP Mapbox

Our next step was to set up mapbox for Unity. The Mapbox version used in this thesis is version 2.1.1.

First we have to import the package to Unity.

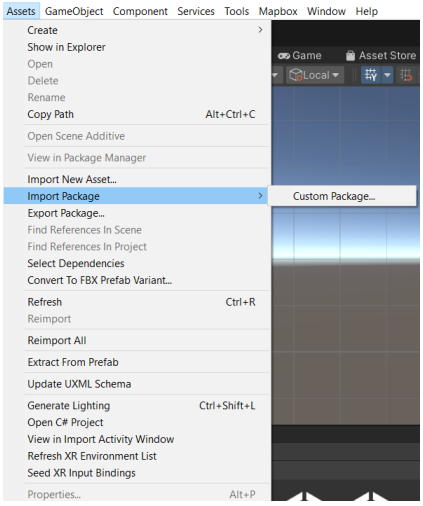


Figure 5.3: Importing a package.

After importing mapbox 2.1.1 to Unity, we have to create a public access token via the mapbox website dashboard in order to be able to access their API. Once the token is generated we need to insert it through Unity so that the mapbox SDK is activated and we have complete access to all the APIs necessary for the development of our application.

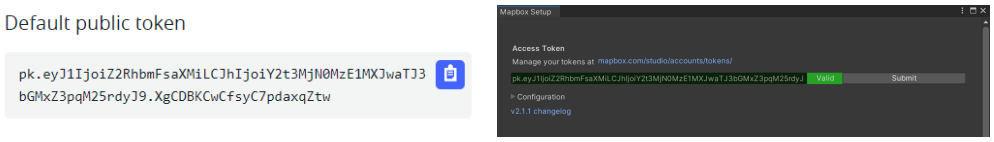


Figure 5.4: Mapbox public access token.

Various useful tutorials and samples are included with the package and become available to us once we activate the SDK.

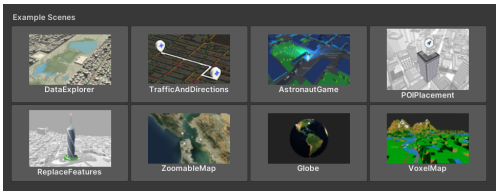


Figure 5.5: Mapbox sample scenes.

5.2.3 Player Settings

In order for our application to function correctly on mobile devices when using AR Foundation, we need to adjust a few project settings under the Player tab.

First we need to disable the Auto Graphics API and switch to using OpenGLES3.

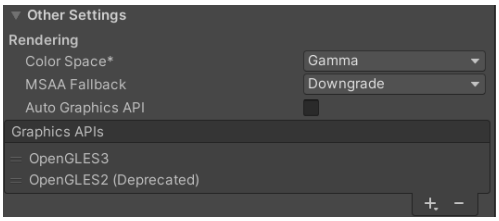


Figure 5.6: Graphics API.

Next we need to set the minimum API level to Android 8.1 'Oreo' (API level 27).

Figure 5.7: Android API.

Lastly we need to switch the scripting backend from Mono to IL2CPP and enable building for ARM64 architectures.

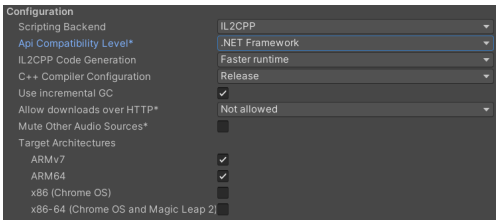


Figure 5.8: Scripting backend and target architectures.

5.3 Designing the User Interface (UI)

Before diving into the specifics of each scene of our application we should take a moment to go over the basics of creating our user interface (UI). The UI of any application is one of its most important components since that's what the user interacts with when using the application. The UI has to be simple, easy to use, cohesive and practical.

Creating the UI for the main screen and map screen was relatively straightforward. The UI for the AR location screens posed a bit of a challenge though due to the fact that it needed to be minimal in order not to obscure the users view of the scene that he is creating.

5.3.1 UI Sprites

The first step towards building our UI is creating the sprites that we are going to use. To achieve this goal the drawing software Krita was used. The version of Krita that we worked with was version 5.2.2.

5.3.1.1 Button Sprites

The first sprites that we created were the ones used for the application's buttons. We used the sprite swap transition option that Unity provides for its buttons. What this means, essentially, is that we can assign a number of sprites to a button with each of them corresponding to a specific button state. Those states are: default, highlighted, pressed, selected and disabled. Whenever the button switches states, so does the displayed sprite change.

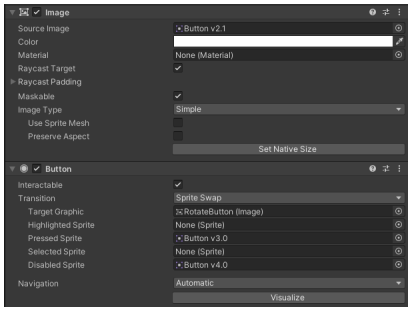


Figure 5.9: Example of a UI button using sprite swapping.

We chose to use vivid colors and adopted the rounded edges design philosophy, drawing inspiration from Android's Material design. The colors we chose for our button sprites was blue, green, gray and red. Each of these sprites has rounded edges and an outline with different shades of the base color, meant to give a sense of depth and elevation to the button.

Figure 5.10: Button sprites.

The blue button sprites was used for most buttons as the default state, the green sprite was used as the default state for buttons meant to place AR content and the red sprite was used for the back button, since the red color felt like a good way to indicate that this action could result in loss of progress in the AR scenes. The highlighted and selected states weren't of that much interest for us, since in a mobile application the user doesn't perform those actions all that often, thus we didn't assign a specific sprite for them. For the selected state we chose to use the green button sprite to signify that the user's action has a positive and immediate effect on the application and to reinforce the feeling of responsiveness. Last but not least, we used the gray button sprite for the disabled state of the buttons, since it's the least

visually stimulating color option which helps draw focus away from these specific buttons.

5.3.1.2 Miscellaneous Sprites

Apart from the Button Sprites that we mentioned previously, we also designed a couple of other sprites for the elements of the UI that the user does not interact with. We decided not to give rounded outer edges to these sprites in order to differentiate them from the intractable UI elements.

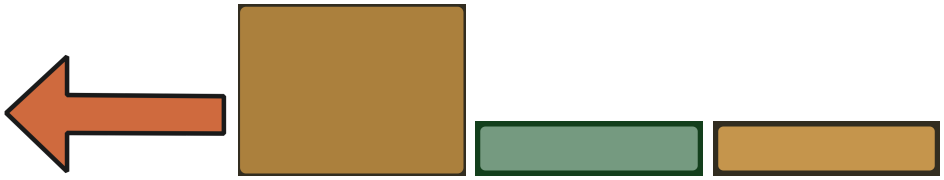


Figure 5.11: Miscellaneous sprites.

5.3.1.3 Additional Home Screen Components

We wanted our home screen to have a clean look to it and at the same time reflect that this is an application meant for architecture. For that reason we chose to use an architectural drawing of Nea Chora as the applications home background and then overlay a canvas over it which would feature all the buttons and other UI elements of the home screen.

Figure 5.12: Home Scene Background.

We used the same philosophy of having rounded edges for the home scene canvas and chose to have a wave pattern on it since it looks significantly better

than having a monochromatic canvas. We used a stock image with the wave pattern, which was then edited with Krita to give it rounded edges and a multicolor outline to add depth to it.

Figure 5.13: Home Scene Canvas.

5.4 Home Scene

The home scene is the first screen the user sees when opening the application. It is the main hub through which the user can navigate to the other screens, enable or disable the tutorial system and view his current distance from each of the AR locations. The tutorial system will be thoroughly explained in a separate section later in this thesis.

5.4.1 Home scene hierarchy and components

This scene includes the following elements: 1) a Camera, since every Unity scene requires one in order to be rendered, 2) a directional light, 3) a Canvas that contains all the tutorial messages, 4) a Canvas that contains all the UI elements of the scene, 4) an "Event System" object, which is responsible for sending events to objects in the application based on input and 5) a "Scene Manager" which handles how scene transitions work and also implements the GPS location/distance functionality of the scene.

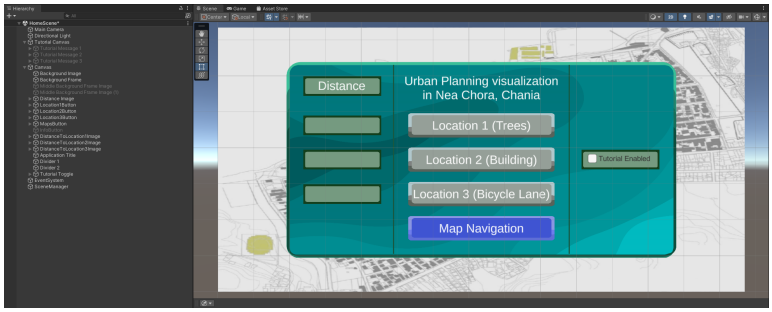


Figure 5.14: Home Scene hierarchy and components.

5.4.2 Switching Scenes

The scene gets switched whenever the user presses one of the active location buttons or the map navigation button. This functionality is implemented via the SceneManager and HomeScreenButtons scripts.

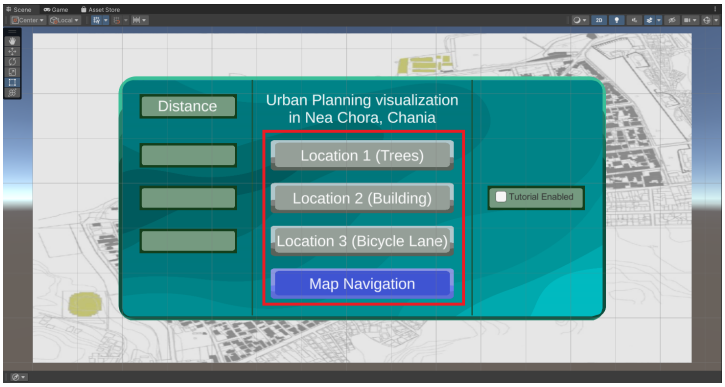


Figure 5.15: Buttons that switch scenes.

```
public static void LoadScene(int buildIndex) {
  Instance.loadedLevels.Push(GetActiveScene().buildIndex);
  UnityEngine.SceneManagement.SceneManager.LoadScene(buildIndex);
}

public static void LoadScene(string sceneName) {
  Instance.loadedLevels.Push(GetActiveScene().buildIndex);
  UnityEngine.SceneManagement.SceneManager.LoadScene(sceneName);
}
```

Listing 5.1: Loading scenes via the Scene Manager.

```
public void mapNavigation() {
  SceneManager.LoadScene("MapScene");
}
```

Listing 5.2: Example of a scene getting loaded.

5.4.3 GPS and Distance from Locations

One of the main goals of our thesis was to create a location-based AR application for urban planning. Thus, we needed a way to be able to track the user's real-world position and distance from each of the AR locations. This is achieved by utilizing the mobile device's GPS services, since all modern mobile phones are equipped with such sensors.

Three scripts are vital to getting the GPS coordinates and calculating the distances: the Native GPS Plugin, the Location Provider and the Haversine Distance.

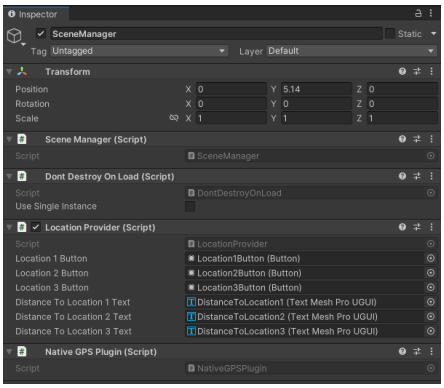


Figure 5.16: Scene Manager components.

Once the distance to each location is calculated it is displayed on the user's screen and if it is within a predefined allowed range, then the corresponding AR location button is enabled.

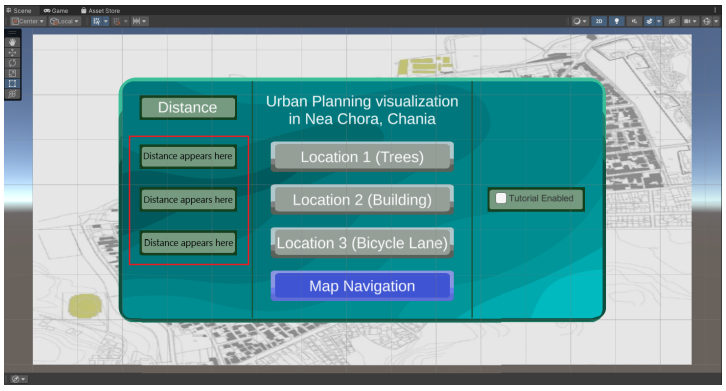


Figure 5.17: Displaying the distance from a location.

5.4.3.1 Native GPS Plugin Script

The native GPS plugin is responsible for accessing the mobile devices location service and receiving GPS coordinates. We can then access these coordinates through our other scripts and utilize them to calculate the user's distance from each AR location. By using native code for each platform we are guaranteed to get the best possible results.

```
public static bool StartLocation()
{
  Instance.Awake();

  if(!Input.location.isEnabledByUser)
  {
    Debug.Log ("Location service disabled");
    return false;
  }

  #if UNITY_IOS

  if(Application.platform == RuntimePlatform.IPhonePlayer)
  {
    startLocation();
  }

  #elif UNITY_ANDROID

  if(Application.platform == RuntimePlatform.Android)
  {
    obj.CallStatic("startLocation");
  }

  #endif

  return true;
}
```

Listing 5.3: Starting the location service based on the user's platform.

5.4.3.2 Location Provider Script

The location provider is the script that handles our main GPS coordinates functionality. It checks for location permissions and asks the user to grant them, if necessary. If the permissions have been granted then it enables the location provider service in order to start receiving GPS coordinates. Then using the HaversineDistance script and the users latitude and longitude, we can calculate their distance from each of the AR locations. If the user is within a set distance from a location, then the corresponding button on the Home Screen UI will become active (it will turn from gray to blue).

```
private void updateDistanceToAllLocations()
{
```

```
3   if (locationProviderIsReady)
4   {
5     distanceToLocation1 = HaversineDistance.getDistanceInMeters(latitude,
      longtitude, location1Latitude, location1Longtitude);
6     distanceToLocation2 = HaversineDistance.getDistanceInMeters(latitude,
      longtitude, location2Latitude, location2Longtitude);
7     distanceToLocation3 = HaversineDistance.getDistanceInMeters(latitude,
      longtitude, location3Latitude, location3Longtitude);
8             distanceToLocation1Text.text = System.Math.Round(
      distanceToLocation1, 2).ToString() + "m";
9             distanceToLocation2Text.text = System.Math.Round(
      distanceToLocation2, 2).ToString() + "m";
10            distanceToLocation3Text.text = System.Math.Round(
      distanceToLocation3, 2).ToString() + "m";
11  }
12 }
```

Listing 5.4: Updating the Distance to each AR location.

5.4.3.3 Haversine Distance Script

The Haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. The Earth isn't an exact sphere but if we were assume that it is (by ignoring ellipsoidal effects and using its volumetric radius - 6371km) we can calculate the distance between two locations on it using the Haversine formula with 0.22% percentage of error [35]. This is accurate enough for our application and therefore this is the method that we used for calculating the user's distance from a given AR location.

Let the central angle ϑ between any two points on a sphere be:

$$\theta = \frac{d}{r}$$

Where r is the radius of the chosen sphere and d is the distance between the two points along a great circle of the sphere.

The general equation for the Haversine Formula is:

$$Haversine(\theta) = sin^2(\frac{\theta}{2})$$

Thus, the distance between two coordinates over radius r is represented as follows:

$$d = 2r * arcsin\sqrt{sin^2(\frac{\phi_2 - \phi_1}{2}) + cos(\phi_1)cos(\phi_2)sin^2(\frac{\lambda_2 - \lambda_1}{2})}$$

Where r is the radius of the chosen sphere. φ and λ are the latitude and longitude respectively.

If we define:

$$a = sin^2(\frac{\theta}{2})$$

Then:

$$Haversine(\theta) = a$$

Which can be transformed into:

$$\theta = 2arcsin(\sqrt{(a)})$$

We know that the arcsine can also be expressed with the arctangent, as follows:

$$arcsin(x) = arctan(\frac{x}{\sqrt{1-x^2}})$$

By applying this to the Haversine formula we get the following equation:

$$\theta = 2arctan(\frac{a}{\sqrt{1-a^2}})$$

Therefore the distance between two points can be calculated as:

$$d = 2r * arctan(\frac{a}{\sqrt{1-a^2}})$$

In our case r will be the volumetric radius of the Earth (6371km).

```
public class HaversineDistance : MonoBehaviour
{
    private const double earthRadius = 6371;

    public static double getDistanceInMeters(double lat1, double lon1,
    double lat2, double lon2)
    {
        return getDistanceFromLatLonInKm(lat1, lon1, lat2, lon2) * 1000;
    }

    private static double getDistanceFromLatLonInKm(double lat1, double lon1
    , double lat2, double lon2)
    {
        var dLat = deg2rad(lat2 - lat1);
        var dLon = deg2rad(lon2 - lon1);
        var a = System.Math.Sin(dLat / 2) * System.Math.Sin(dLat / 2) +
          System.Math.Cos(deg2rad(lat1)) * System.Math.Cos(deg2rad(lat2)) *
          System.Math.Sin(dLon / 2) * System.Math.Sin(dLon / 2);
```

```
17          var c = 2 * System.Math.Atan2(System.Math.Sqrt(a), System.Math.Sqrt
    (1 - a));
18          var d = earthRadius * c; // Distance in km
19          return d;
20      }
21
22      private static double deg2rad(double deg)
23      {
24          return deg * (System.Math.PI / 180);
25      }
26  }
```

Listing 5.5: Calculating the Haversine Distance.

5.5 Map Scene

The map scene acts a navigational aid for the user. Through the map scene they are able to view their current position on the map, view the exact location of each of the three AR locations and also receive directions on how to reach their destination.

5.5.1 Map scene hierarchy and components

This scene includes the following elements: 1) a Camera, 2) a Directional Light, 3) a Canvas that contains all the tutorial messages, 4) a Canvas that contains all the UI elements of the scene, 5) a mapbox Map, 6) the "TargetPlayer", meaning the user's current location marker on the map, 7) a Location Provider 8) a Directions object, in charge of producing directional guidance when enabled and 9) three waypoints, each of which corresponds to one of the three AR locations.

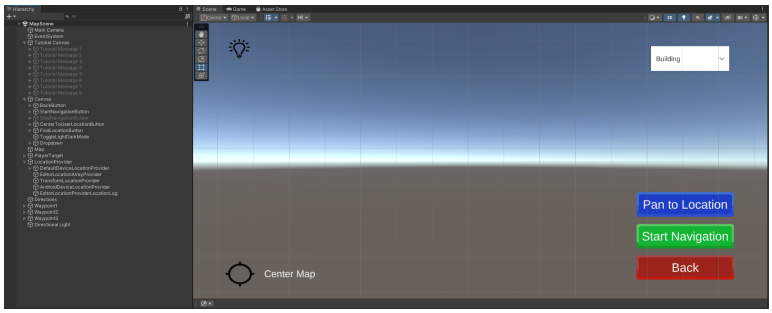


Figure 5.18: Map Scene hierarchy and components.

5.5.2 Adding Mapbox to the scene

The first step we ought to take in order to create a scene capable of displaying

a map and navigational data is to add mapbox to it.

5.5.2.1 Adding the Map

The first thing we need to add to our scene is a map prefab. This prefab will create a map of the surrounding area on the user's screen. The map prefab offers a variety of customization options, such as configuring the zoom level (set to 16), choosing the theme of the map (our default is Mapbox Light), configuring the data sources (we are using Mapbox Terrain and Mapbox Streets) and adding points of interest (POIs).

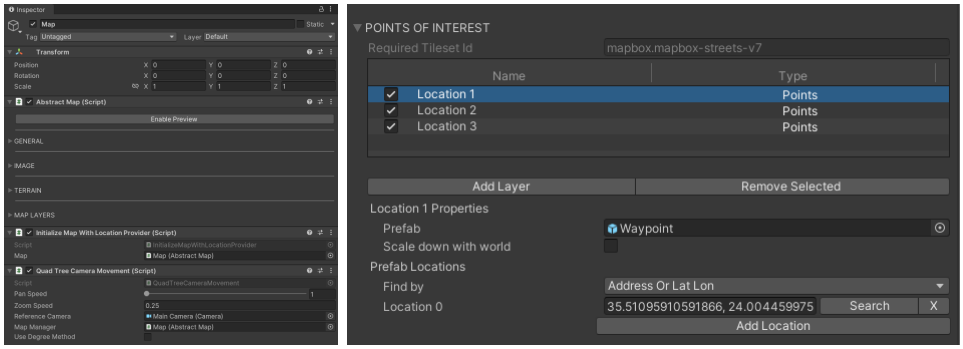


Figure 5.19: Map components and the The three points of interest (POIs).

The three POIs added to the map correspond to the three AR locations. A POI prefab (Waypoint) was created to act as a marker on the map. Their exact coordinates are (latitude, longitude):

- Location 1 (Trees, Benches and Street Lamps): 35.51095910591866, 24.004459975858733
- Location 2 (Building): 35.517482361372736, 24.01108550473443
- Location 3 (Bicycle Lane and Bicycle Station): 35.5118184719792, 24.001774574658594

Figure 5.20: Waypoint (POI marker).

The Quad Tree Movement script that is attached to map allows the user to zoom or move the map using finger gestures (slide to move, pinch to zoom in or out).

```
void HandleTouch()
{
  float zoomFactor = 0.0f;
  switch (Input.touchCount)
  {
    case 1:
      {
        PanMapUsingTouchOrMouse();
      }
      break;
    case 2:
      {
        Touch touchZero = Input.GetTouch(0);
        Touch touchOne = Input.GetTouch(1);
        Vector2 touchZeroPrevPos = touchZero.position - touchZero.
    deltaPosition;
        Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition
    ;
        float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).
    magnitude;
        float touchDeltaMag = (touchZero.position - touchOne.position).
    magnitude;
        zoomFactor = 0.01f * (touchDeltaMag - prevTouchDeltaMag);
      }
      ZoomMapUsingTouchOrMouse(zoomFactor);
      break;
    default:
      break;
  }
}
```

Listing 5.6: Moving and zooming the map using touch.

The Initialize Map With Location Provider script, initializes the map once the mapbox location provider starts working and receiving GPS coordinates. Once the location provider is active, the current location of the user will become known and therefore the map will load the correct area.

5.5.2.2 Adding the Location Provider

Next, we need to add the Mapbox Location Provider prefab to our scene. This prefab contains a LocationProviderFactory script component which is responsible for accessing the location service sensors of the mobile device.

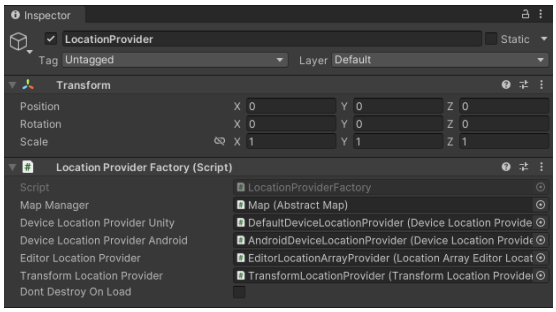


Figure 5.21: Location Provider components.

5.5.3 Tracking and displaying the user's position

As mentioned previously, though the use of the Location Provider we can start tracking the user's position on the map. In order to actually display the user's position on the map we use the PlayerTarget prefab. This prefab acts as a marker that follows the user's movements.

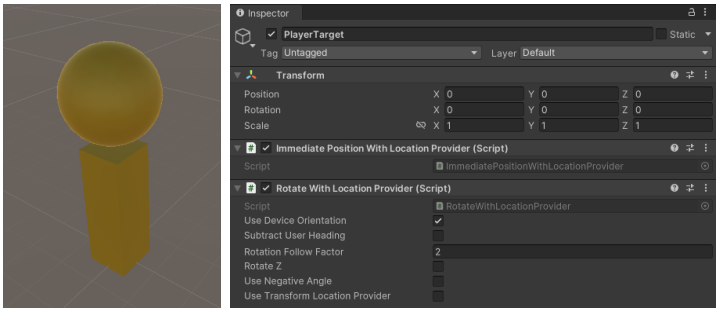


Figure 5.22: The PlayerTarget prefab and its components.

We have added two components to the PlayerTarget prefab:

- Immediate Position With Location Provider script: Moves the Player-Target prefab to the user's current location on the map (updates the location every frame).
- Rotate With Location Provider: Rotates the Player Target prefab on the map, so that it matches the user's current orientation.

5.5.4 Centering the map to the user's location or the destination

Another useful feature that has been implemented in our map scene is the capability of immediately centering the map to the user's location or to a specific AR location. This is achieved via the use of the CenterMapToLocation script, a button for centering to the user's location, a dropdown list for selecting the desired AR location and a separate button for centering to the selected location.

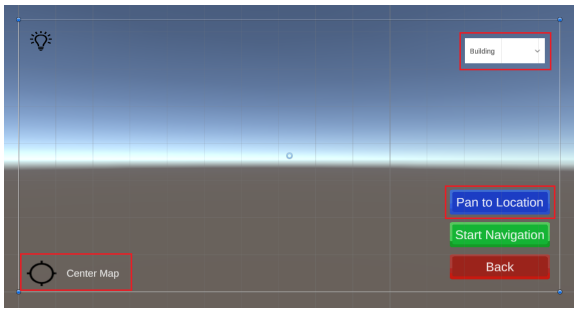


Figure 5.23: UI components for centering the map to a specific AR location or the user's position.

This script is attached as a component to the canvas of the scene.

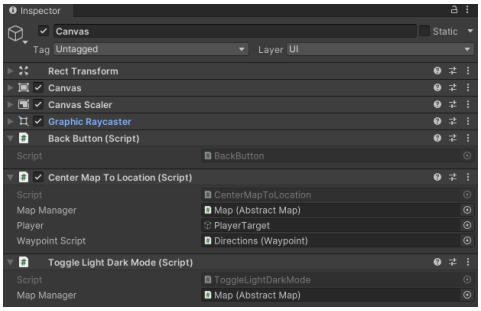


Figure 5.24: Map Scene canvas components.

```
public class CenterMapToLocation : MonoBehaviour
{
    public AbstractMap mapManager = null;
    public GameObject player = null;
    public Waypoint waypointScript;

    private GameObject destination = null;
    private double destinationLatitude = 0;
    private double destinationLongtitude = 0;

```

```
    private void Update()
    {
        destination = waypointScript.destination;
        destinationLatitude = waypointScript.destinationLatitude;
        destinationLongtitude = waypointScript.destinationLongtitude;
    }

    public void centerCameraToUser() {
        mapManager.UpdateMap(mapManager.WorldToGeoPosition(
            new Vector3(
                player.transform.localPosition.x,
                player.transform.localPosition.y,
                player.transform.localPosition.z)),
                16);
    }

    public void centerCameraToTargetLocation()
    {
        if (destination != null)
        {
            waypointScript.correctWaypointPosition(destinationLatitude,
    destinationLongtitude, destination);
            mapManager.UpdateMap(new Vector2d(destinationLatitude,
    destinationLongtitude), 16);
        }
    }
}
```

Listing 5.7: CenterMapToLocation script.

5.5.5 Displaying directions

A very important aspect of the map scene is the ability to provide navigational guidance to the user in order for them to reach one of the three AR locations. This is handled by the Directions object in our scene and the Waypoint objects.

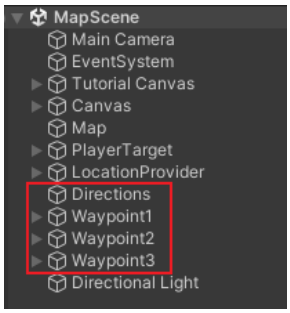


Figure 5.25: Directions and Waypoints.

Before starting to explain how the directions system works, we need to understand the difference between map geocoordinates and world position.

The geocoordinates refer to the set of coordinates of a location on the map and by utilizing the MoveToGeocoordinate() function, provided by mapbox, we can move Unity objects to specific map locations. World position refers to the position of a Unity object in the scene. Therefore an objects geoposition and world position are two very different sets of coordinates.

```
1 private void placeWaypoint(double lat, double lng, GameObject waypoint) {
2   waypoint.transform.MoveToGeocoordinate(lat, lng, map.CenterMercator, map.
      WorldRelativeScale);
3 }
```

Listing 5.8: Placing a waypoint on the map using geocoordinates.

In order to display a directions line on the user's map, connecting two geolocations (the user's position and a destination), we need to be able to translate the geolocation of both points into Unity world positions. That's why we have created the three Waypoint objects and the PlayerTarget object.

The Directions object has three components attached to it:

- JDirectionsFactory script: This script is responsible for creating the directions line that connects the user's position with the destination. In order to create the directions line visual representation we must provide a mesh modifier (Loft Modifier) and a Material (DirectionsMaterial). We also need to provide the PlayerTarget object and the Waypoint object that corresponds to the destination, so that we have access to these objects world positions.

- Waypoint script: This script is responsible for three things: 1) Placing the Waypoint objects in the correct locations on the map. 2) Switching which Waypoint is considered the user's destination when a new location is selected. 3) Ensuring that the Waypoints are always on the correct position on the map and don't get misaligned due to camera movements.

- Navigation script: This script is responsible for starting and ending the navigation process (and subsequently instructing the JDirectionsScript to stop drawing a directions line).

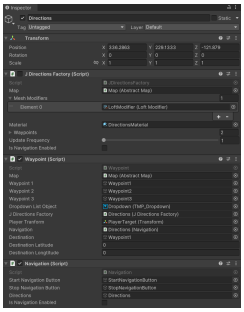


Figure 5.26: The components of the Directions object.

```
public void StartNavigation() {
  isNavigationEnabled = true;
  startNavigationButton.SetActive(false);
  stopNavigationButton.SetActive(true);
  directions.GetComponent<JDirectionsFactory>().enabled = true;
  directions.GetComponent<JDirectionsFactory>().isNavigationEnabled = true;
  PlayerPrefs.SetInt(IS_NAVIGATION_ACTIVE, IS_NAVIGATION_ACTIVE_YES);
}

public void StopNavigation() {
  isNavigationEnabled = false;
  startNavigationButton.SetActive(true);
  stopNavigationButton.SetActive(false);
  directions.GetComponent<JDirectionsFactory>().isNavigationEnabled = false;
  PlayerPrefs.SetInt(IS_NAVIGATION_ACTIVE, IS_NAVIGATION_ACTIVE_NO);
}
```

Listing 5.9: Beginning and ending the navigation process.

5.5.6 Switching between Light and Dark Map Theme

Lastly, we have also implemented a way to switch between the light and dark theme of the map via a button press.

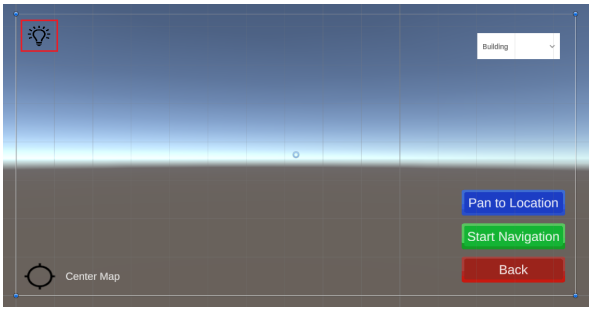


Figure 5.27: UI component for switching between Light and Dark Mode.

```
public class ToggleLightDarkMode : MonoBehaviour
{
    public AbstractMap mapManager;
    public void toggleLightDarkMode()
    {
        if (mapManager.ImageLayer.LayerSource == ImagerySourceType.
    MapboxDark)
        {
            mapManager.ImageLayer.SetLayerSource(ImagerySourceType.
    MapboxLight);
        }
        else
        {
            mapManager.ImageLayer.SetLayerSource(ImagerySourceType.
    MapboxDark);
        }
    }
}
```

Listing 5.10: ToggleLightDarkMode script.

5.6 Location Screens Overview

Below we'll take a look at features and components that are the same among all three Location Screens. These include: setting up AR Foundation for our scenes, organizing the AR functionality into three distinct modes (View Mode, Placement Mode and Edit Mode) and selecting and outlining virtual objects.

5.6.1 Adding AR Foundation to the scene

The first step we need to take in order to create an augmented reality scene is to add AR Foundation to our scene. This process is more or less the same across all three AR location scenes, so we'll go over it once before diving into the specifics of each scene.

AR Foundation requires the three following prefabs to be added to a scene: 1) an AR Session prefab, 2) an AR Session Origin prefab and 3) an AR Camera prefab.

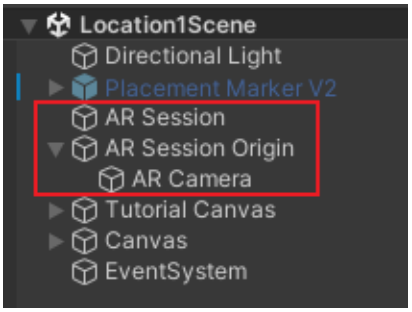


Figure 5.28: Setting up AR Foundation in our scene.

5.6.1.1 Adding the AR Session

Our first task is to add an AR Session prefab to our scene. This object refers to an instance of the AR functionality and controls the life-cycle of all AR features used in our scene by enabling or disabling AR on the target platform. The AR Session is required so that AR Foundation can start tracking features in it's environment.

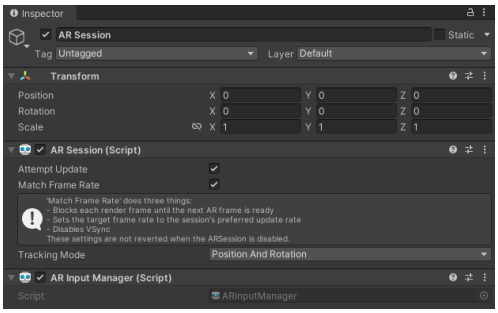


Figure 5.29: AR Session components.

The AR Input Manager script enables world tracking in our scene.

5.6.1.2 Adding the AR Session Origin and the AR Camera

Our next step is to add an AR Session Origin prefab to the scene and replace the Unity Main Camera with AR Foundation's AR Camera.

The AR Session Origin prefab is responsible for transforming trackable features, such as planar surfaces and feature points, into their final position, orientation, and scale in the Unity scene. It also allows us to scale virtual content and apply an offset to the AR Camera.

As seen in the figure below, we have attached a plethora of components to the AR Session Origin. Some of these components are location-specific and will be explained in a later section. Now, let's take a look at the attached components that are provided by AR Foundation:

- AR Point Cloud Manager: This component creates point clouds, which are, essentially, tracked feature points of the environment.
- AR Plane Manager: This component detects and keeps track of planes in the environment. Planes can be horizontal or vertical flat surfaces. For the purposes of our application we only track horizontal planes (in order to find the real-world ground surface).
- AR Raycast Manager: This component enables raycasting through AR Foundation. Raycasting allows us to determine when a ray intersects with a trackable object.
- AR Anchor Manager: This component creates anchor GameObjects and keeps track of them. An anchor is a particular point in space that we want our device to keep track of (in our application, every virtual object placed by the user is an anchor, since it is important to keep track of them and ensure that they maintain the correct position and orientation).

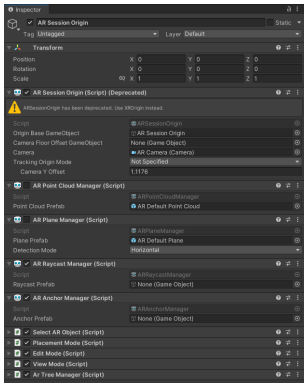


Figure 5.30: AR Session Origin components.

Lastly, we have to add an AR Camera prefab as a child object of our AR Session Origin. This replaces the Unity Main Camera and manages the device camera textures and the light estimation modes. In addition, we have added an AR Occlusion Manager component which, when enabled, will allow virtually rendered objects to be occluded by real-world ones.

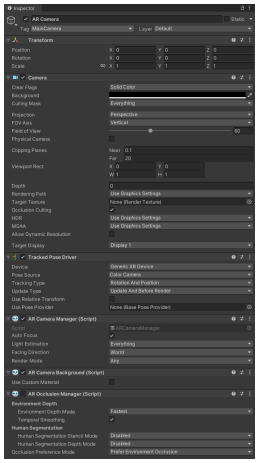


Figure 5.31: AR Camera components.

5.6.2 AR Modes

The most important aspect of our AR locations is implementing the functionality of placing, interacting and viewing objects. In order to achieve this, we have split the functionality into the following three, distinct modes:

- View Mode: In this mode the user can view the AR scene that he has created, with minimal UI.
- Placement Mode: In this mode the user will be tasked with detecting the ground surface of their environment and then place virtual objects on it.
- Edit Mode: In this mode the user may edit any placed virtual objects. The editing options given to the user depend on the specific location and the specific object that has been selecting (Deleting and Rotating objects is a constant option among all locations and objects).

5.6.2.1 ARManager

The first problem we needed to tackle was creating a way to switch between these modes, when needed, and manage the displayed UI. For this purpose, we have created the ARManager superclass. This class offers the basic functionality for switching between the three AR modes. This alone isn't enough, though, since each location has its own distinct features. Therefore, we have created an AR Manager for each of the three locations (ARTreeManager,

ARBuildingManager and ARBicycleManager) which inherits from the ARManager superclass. These subclasses implement location specific features and will be covered later on in this thesis.

```
public class ARManager : MonoBehaviour
{
    public const int VIEW_MODE = 1;
    public const int PLACEMENT_MODE = 2;
    public const int EDIT_MODE = 3;
    public const string CURRENT_MODE = "currentMode";

    public ARPlaneManager arPlaneManager = null;
    public ARPointCloudManager arPointCloudManager = null;
    public ARAnchorManager arAnchorManager = null;
    public AROcclusionManager arOcclusionManager = null;
    public EditMode editMode = null;
    public PlacementMode placementMode = null;
    public SelectARObject selectARObject = null;
    public List<GameObject> placedObjects { get; set; }

    public virtual void enablePlacementMode()
    {
        editMode.editModeEnabled = false;
        placementMode.placementModeEnabled = true;
        arPlaneManager.enabled = true;
        arPointCloudManager.enabled = true;
        PlayerPrefs.SetInt(CURRENT_MODE, PLACEMENT_MODE);
    }

    public virtual void enableEditMode()
    {
        editMode.editModeEnabled = true;
        placementMode.placementModeEnabled = false;
        arPlaneManager.enabled = false;
        arPointCloudManager.enabled = false;
        arPlaneManager.SetTrackablesActive(false);
        arPointCloudManager.SetTrackablesActive(false);
        placementMode.placementMarker.SetActive(false);
        placementMode.cleanModels();
        PlayerPrefs.SetInt(CURRENT_MODE, EDIT_MODE);
    }

    public virtual void enableViewMode()
    {
        editMode.editModeEnabled = false;
        placementMode.placementModeEnabled = false;
        arPlaneManager.enabled = false;
        arPointCloudManager.enabled = false;
        arPlaneManager.SetTrackablesActive(false);
        arPointCloudManager.SetTrackablesActive(false);
        placementMode.placementMarker.SetActive(false);
        placementMode.cleanModels();
        PlayerPrefs.SetInt(CURRENT_MODE, VIEW_MODE);
    }

    public void toggleButtonInteractability(Button button, bool
    isInteractable)
    {
        button.interactable = isInteractable;
    }
```

```
56
57 }
```

Listing 5.11: ARManager script.

We have also created a script for each of the three AR modes. The functionality implemented by these scripts will be explained in the following sections.

5.6.2.2 View Mode

The View Mode is by far the simplest of the three modes. It is designed to have minimal UI so that the user may view the scene they created without clutter in their screen. This is particularly important since the screens of mobile devices are relatively small. The location-specific AR Manager is in charge of adjusting the UI elements when the user enters View Mode.

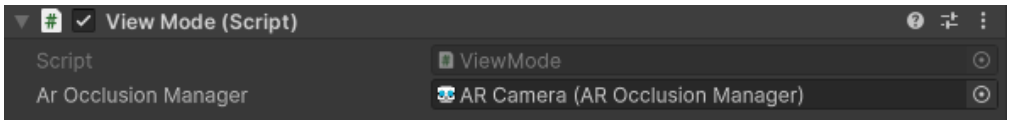


Figure 5.32: The View Mode component attached to each scene's AR Session Origin.

Additionally, when in this mode, the user may enable object occlusion via a UI check box., which will enable the AR Occlusion Manager.

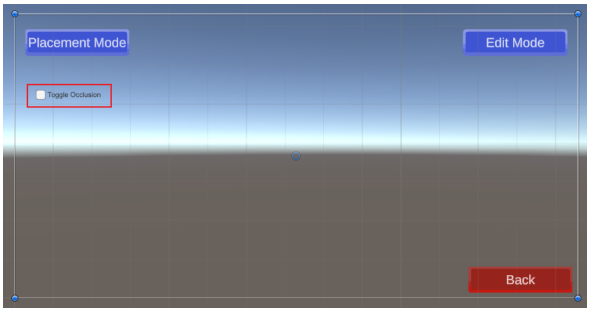


Figure 5.33: Occlusion toggle check box.

```
1 public void toggleOcclusion()
2 {
3   arOcclusionManager.enabled = !arOcclusionManager.enabled;
4 }
```

Listing 5.12: Toggling occlusion on or off.

5.6.2.3 Placement Mode

The Placement Mode is where the user starts building his AR scene. This is where we detect the user's environment's ground surface and start placing virtual objects on the real-world.

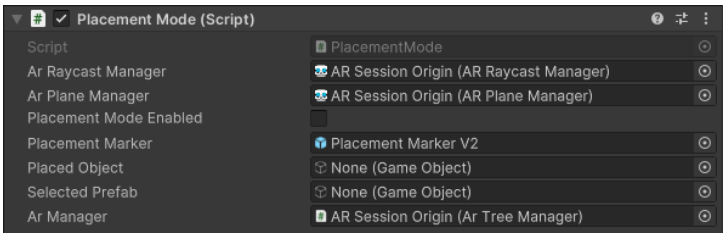


Figure 5.34: The Placement Mode component attached to each scene's AR Session Origin.

Once we enter the Placement Mode, the AR Plane Manager will be enabled and the device will automatically start detecting horizontal planes. In order to detect planes the user has to move the device around so AR Foundation can scan different parts of the ground and start expanding the interactable surface. The Placement Mode script will begin raycasting and checking if a plane has been hit (using the center of the user's mobile device screen as the raycast origin point). Once a plane is hit, a placement marker will appear, to show where the virtual object would be placed.

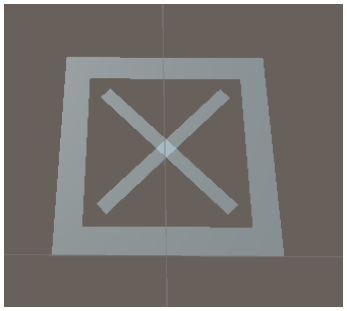


Figure 5.35: The placement marker.

```
1 private Pose tryRaycast()
2 {
3   Pose targetPose = new Pose();
4
5   if (arRaycastManager.Raycast(centerOfScreen, arRaycastHits, TrackableType.
      Planes))
6   {
```

```
7       ARPlane targetPlane;
8       targetPose = arRaycastHits[arRaycastHits.Count - 1].pose;
9       arPlaneManager.trackables.TryGetTrackable(arRaycastHits[arRaycastHits.
        Count - 1].trackableId, out targetPlane);
10      placementMarker.transform.position = targetPose.position;
11      placementMarker.transform.rotation = targetPose.rotation;
12      if (modelsEnabled)
13      {
14        if (modelPrefab == null && selectedPrefab != null)
15        {
16          modelPrefab = Instantiate(selectedPrefab, targetPose.position,
        targetPose.rotation);
17          Destroy(modelPrefab.GetComponent<ARAnchor>());
18        }
19      modelPrefab.transform.position = targetPose.position;
20      modelPrefab.transform.rotation = targetPose.rotation;
21      }
22    }
23
24    return targetPose;
25  }
```

Listing 5.13: Raycasting to place virtual objects.

The user can then choose one of the available models from a dropdown list. We have also implemented a way for the user to preview the models before actually placing them by replacing the placement marker with a temporary copy of the selected model.

Once the model and it's real-world position have been selected, the user can place it.

```
1   private void CreateObject()
2   {
3     if (selectedPrefab != null && targetPose != null)
4     {
5       if (modelsEnabled)
6       {
7         placedObject = Instantiate(selectedPrefab, targetPose.position,
        modelPrefab.transform.rotation);
8       }
9       else
10      {
11        placedObject = Instantiate(selectedPrefab, targetPose.position,
        placementMarker.transform.rotation);
12      }
13      arManager.placedObjects.Add(placedObject);
14    }
15  }
```

Listing 5.14: Creating a virtual object.

5.6.2.4 Edit Mode

The Edit Mode is where the user can interact with the virtual objects he has placed and make changes to them. The functionality of this mode depends

on the specific AR location that the user is at. The two interactions that exist in all locations are: deleting and rotating virtual objects.

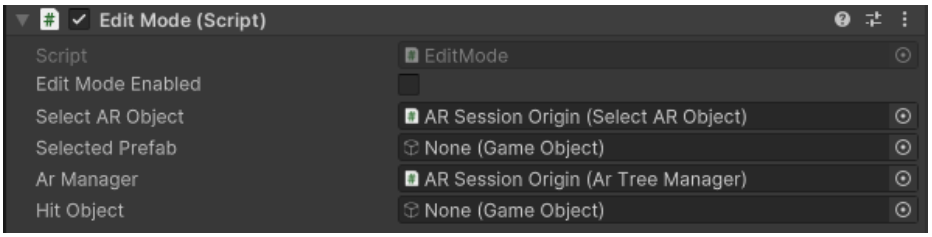


Figure 5.36: The Edit Mode component attached to each scene's AR Session Origin.

```
private void rotateObject(GameObject gameObject)
{
  if (gameObject != null)
  {
    Destroy(gameObject.GetComponent<ARAnchor>());
    gameObject.transform.Rotate(Vector3.up * 30);
    gameObject.AddComponent<ARAnchor>();
  }
}

private void deleteObject(GameObject hitObject, List<GameObject>
    placedObjects)
{
  if (hitObject != null)
  {
    placedObjects.Remove(hitObject);
    Destroy(hitObject);
  }
}
```

Listing 5.15: Rotating and Deleting a virtual object.

It is also worth noting that in order to select an object in Edit Mode, the user has to move their mobile device so that the object is in the center of the screen. The Edit Mode script continuously casts rays and if a virtual object is hit, an outline (white line) will appear around it, to show the user which object is currently selected. The corresponding editing UI will also become enabled and available for use once an object is selected. The way Outlining and Raycasting against the object works will be explained in the next section.

```
void Update()
{
  if (editModeEnabled)
  {
    hitObject = selectARObject.resetHitObjectOutline(hitObject);
    hitObject = selectARObject.shootRay(arManager.placedObjects);
```

```
  } else
  {
    if (hitObject != null)
    {
      selectARObject.resetHitObjectOutline(hitObject);
      hitObject = null;
    }
  }
}
```

Listing 5.16: Raycasting in the Edit Modes Update function.

5.6.3 Selecting and Outlining Objects

In certain scenarios of our applications use, the user may need to select a virtual object in order to edit it (primarily in our Edit Mode). Once the object is selected an outline appears around it.

In order to detect and select virtual objects, we have created a script, named SelectARObject. This script contains two functions:

- shootRay: This function shoots a ray from the center of the mobile devices screen. If an object is hit, it enabled the Outline component of the object.
- resetHitObjectOutline: If we exit Edit Mode or the previously selected object is no longer in the center of the screen (and therefore no longer selected), we can call this function to disable its Outline component.

```
public class SelectARObject : MonoBehaviour
{
    public Camera arCamera = null;

    public GameObject shootRay(List<GameObject> placedObjects)
    {
        Ray ray = arCamera.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
        RaycastHit hitData;
        GameObject hitObject = null;
        if (Physics.Raycast(ray, out hitData, 10))
        {
            hitObject = hitData.transform.gameObject;
            if (hitObject != null && placedObjects.Contains(hitObject) &&
    hitObject.GetComponent<Outline>() != null)
            {
                hitObject.GetComponent<Outline>().enabled = true;
            }
        }
        return hitObject;
    }

    public GameObject resetHitObjectOutline(GameObject hitObject)
    {
```

```
23          if (hitObject != null && hitObject.GetComponent<Outline>() != null)
24          {
25              hitObject.GetComponent<Outline>().enabled = false;
26              hitObject = null;
27          }
28          return hitObject;
29      }
30  }
```

Listing 5.17: SelectARObject script.

All the placeable virtual objects have 1) a Collider component, so that the rays can detect the object, and 2) an Outline component, responsible for creating the outline around the object when it's selected.

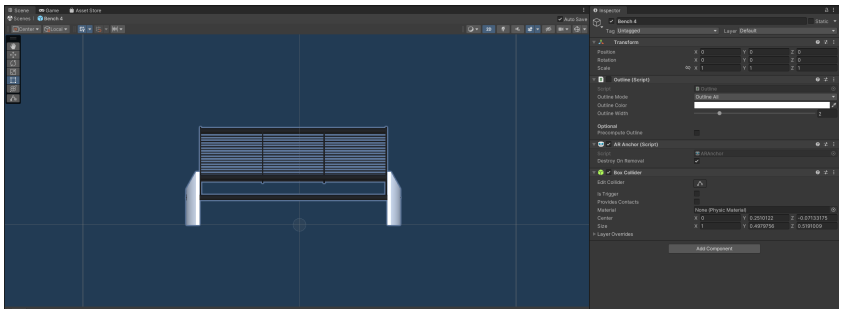


Figure 5.37: Example of the components attached to one of our bench models.

5.7 Location 1 Screen (Trees, Benches and Street Lamps)

This is the AR Location in which the user may place virtual trees, benches and street lamps along the coastal sidewalk in Akti Kanari, near the Chania Nautical-Athletic Center. After placing these objects the user may also rotate or delete them, as well as change the scale of the trees to make them bigger or smaller.

5.7.1 Location 1 scene hierarchy and components

This scene includes the following elements: 1) an AR Camera, 2) a Directional Light, 3) a Canvas that contains all the tutorial messages, 4) a Canvas that contains all the UI elements of the scene, 5) an AR Session Origin, 6) an AR Session and 7) an EventSystem. All location screens contain the aforementioned base elements. What changes between scenes is the UI elements and tutorial messages that each canvas has, as well as the components attached to each element of the scene.

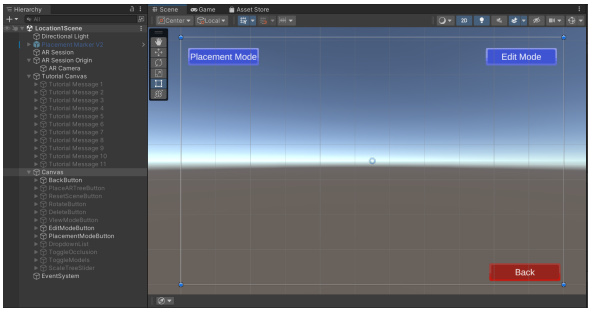


Figure 5.38: Location 1 Scene hierarchy and components.

5.7.2 Tree Models

We selected six different tree model options of our application [37] [38]. It is worth noting that model number three is a bush and not a tree and that model number two comes with a wind simulation and physics script attached to its branches. This script simulates how the branches would move if the tree was subjected to light wind.

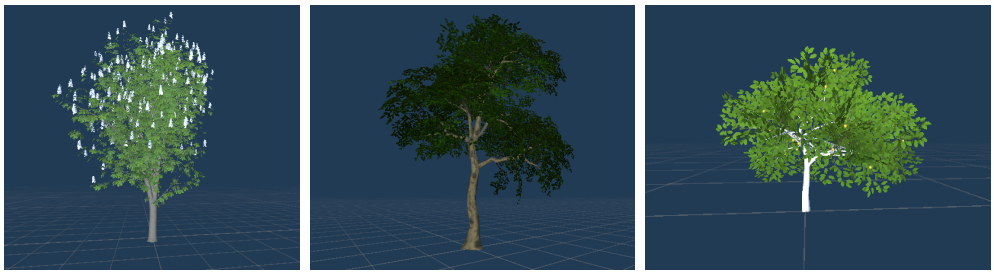


Figure 5.39: Trees 1, 2 and 3 (bush).

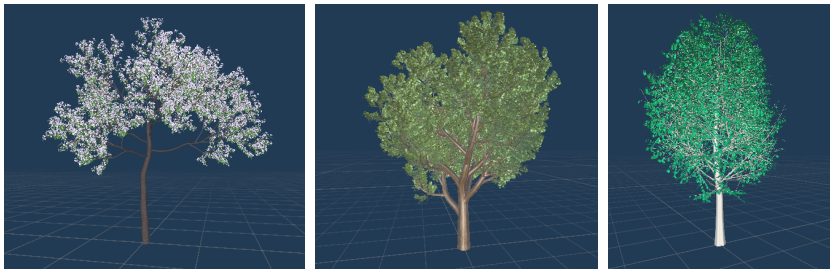


Figure 5.40: Trees 4, 5 and 6.

5.7.3 Street Lamp Models

We selected five different street lamp models, choosing three modern-looking and two "traditional" designs [37] [38]. It is worth mentioned that one of our original ideas was to try having different lighting options for the street lamps, by adding virtual light sources. However, when tested in a real, outdoors environment, during daytime (since we need the environment to have good lighting for the plane detection to function), the virtual lighting effect wasn't noticeable.

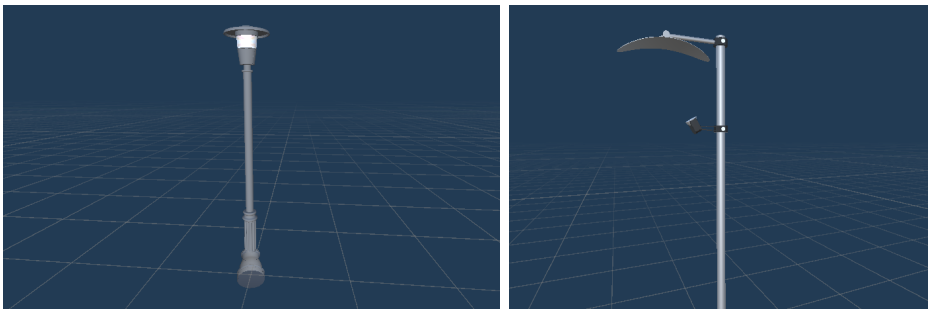


Figure 5.41: Street Lamps 1 and 2.

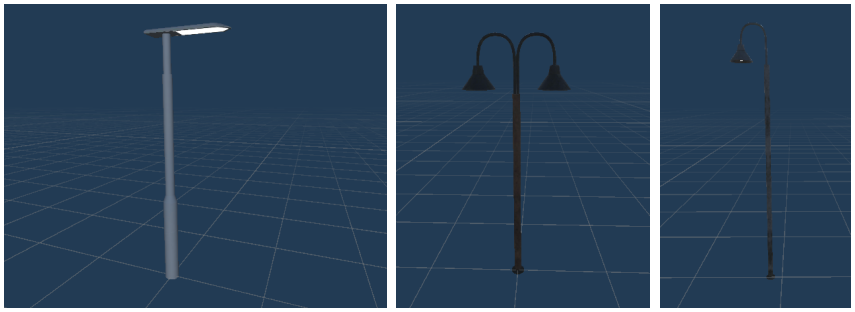


Figure 5.42: Street Lamps 3, 4 and 5.

5.7.4 Bench Models

We selected five different bench model options, opting for modern designs [37] [38].

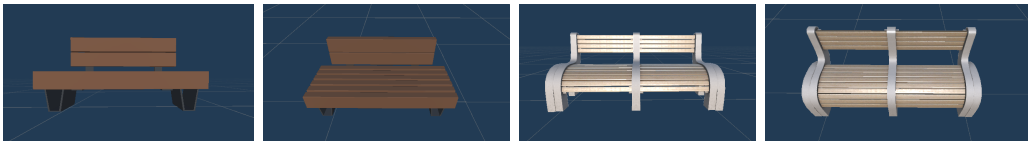


Figure 5.43: Benches 1 and 2.

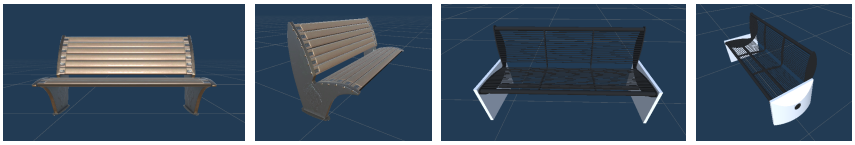


Figure 5.44: Benches 3 and 4.

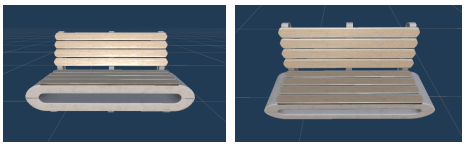


Figure 5.45: Bench 5.

5.7.5 AR Modes

This location's AR functionality has been split into three AR Modes: View Mode, Placement Mode and Edit Mode, which are controlled by the ARTree-Manager script.

5.7.5.1 ARTreeManager

In order to control how the three AR Modes behave in this scene, we have created the ARTreeManager script, which inherits from the ARManager (section 5.6.2.1) class.

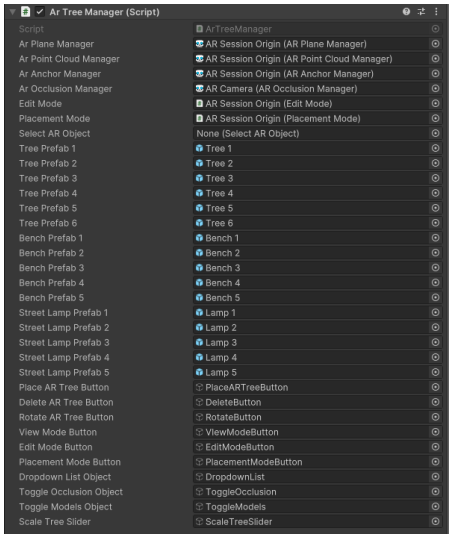


Figure 5.46: The ARTreeManager component, attached to the scene's AR Session Origin.

This script is in charge of handling the UI changes when switching modes, as well as handling how certain UI elements behave.

```
public override void enablePlacementMode()
{
  base.enablePlacementMode();
  editModeButton.SetActive(false);
  viewModeButton.SetActive(true);
  toggleOcclusionObject.SetActive(false);
  toggleModelsObject.SetActive(true);
  dropdownListObject.SetActive(true);
  placeARTreeButton.SetActive(true);
  placementModeButton.SetActive(false);
  deleteARTreeButton.SetActive(false);
  rotateARTreeButton.SetActive(false);
  arPlaneManager.SetTrackablesActive(true);
  arPointCloudManager.SetTrackablesActive(true);
  scaleTreeSlider.SetActive(false);
}

public override void enableEditMode()
{
  base.enableEditMode();
  toggleOcclusionObject.SetActive(false);
  toggleModelsObject.SetActive(false);
  dropdownListObject.SetActive(false);
  deleteARTreeButton.SetActive(true);
  rotateARTreeButton.SetActive(true);
  viewModeButton.SetActive(true);
  editModeButton.SetActive(false);
  placeARTreeButton.SetActive(false);
  placementModeButton.SetActive(true);
  scaleTreeSlider.SetActive(true);
```

```
}

public override void enableViewMode()
{
  base.enableViewMode();
  toggleOcclusionObject.SetActive(true);
  toggleModelsObject.SetActive(false);
  dropdownListObject.SetActive(false);
  deleteARTreeButton.SetActive(false);
  rotateARTreeButton.SetActive(false);
  placementModeButton.SetActive(true);
  placeARTreeButton.SetActive(false);
  viewModeButton.SetActive(false);
  editModeButton.SetActive(true);
  scaleTreeSlider.SetActive(false);
}
```

Listing 5.18: Enabling each AR Mode.

5.7.5.2 View Mode

The View Mode functionality is the same as in all other AR locations of our application.

5.7.5.3 Placement Mode

The Placement Mode functionality is more or less the same as in the other AR locations of our applications, with the only changes being the text on some UI elements and the options for the virtual object models.

In this AR location the user may choose one of the previously shown tree, street lamp or bench models from a dropdown list.

```
private void switchTreePrefab()
{
  TMP_Dropdown dropdown = dropdownListObject.GetComponent<TMP_Dropdown>();
  if (dropdown.value != dropdownValue){
    dropdownValue = dropdown.value;
    placementMode.cleanModels();
  }
  switch (dropdown.value)
  {
    case 0:
      placementMode.selectedPrefab = treePrefab1;
      break;
    case 1:
      placementMode.selectedPrefab = treePrefab2;
      break;
    case 2:
      placementMode.selectedPrefab = treePrefab3;
      break;
    case 3:
      placementMode.selectedPrefab = treePrefab4;
      break;
    case 4:
```

```
23              placementMode.selectedPrefab = treePrefab5;
24              break;
25      case 5:
26        placementMode.selectedPrefab = treePrefab6;
27        break;
28      case 6:
29        placementMode.selectedPrefab = benchPrefab1;
30        break;
31      case 7:
32        placementMode.selectedPrefab = benchPrefab2;
33        break;
34      case 8:
35        placementMode.selectedPrefab = benchPrefab3;
36        break;
37      case 9:
38        placementMode.selectedPrefab = benchPrefab4;
39        break;
40      case 10:
41        placementMode.selectedPrefab = benchPrefab5;
42        break;
43      case 11:
44        placementMode.selectedPrefab = streetLampPrefab1;
45        break;
46      case 12:
47        placementMode.selectedPrefab = streetLampPrefab2;
48        break;
49      case 13:
50        placementMode.selectedPrefab = streetLampPrefab3;
51        break;
52      case 14:
53        placementMode.selectedPrefab = streetLampPrefab4;
54        break;
55      default:
56        placementMode.selectedPrefab = streetLampPrefab5;
57        break;
58    }
59  }
```

Listing 5.19: Selecting the virtual object prefab from the dropdown list.

5.7.5.4 Edit Mode

As mentioned in section 5.6.2.4 the user may delete or rotate any placed virtual trees, street lamps or benches, after selecting them. In addition to that they may also change the scale of trees. It's worth mentioning that the scaling functionality is only implemented on trees due to the fact that benches and street lamps are required by law to have specific dimensions, so scaling them wouldn't be correct by urban planning standards.

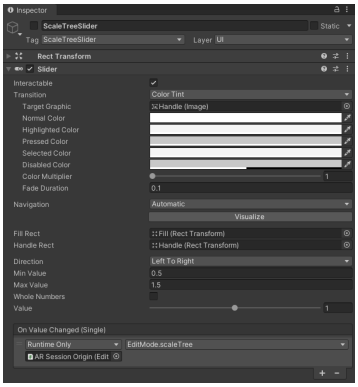


Figure 5.47: The slider used for changing the tree's scale.

A Unity Slider UI object was used in order to implement the scaling trees functionality. We have set the minimum value that the slider can have to 0.5 and the maximum to 1.5, which means that the scaling can range from half the original size up to 1.5 times bigger.

```
public void scaleTree()
{
  GameObject scaleTreeSlider = GameObject.FindGameObjectsWithTag("
    ScaleTreeSlider")[0];
  if (hitObject != null && hitObject.tag == "Tree" && scaleTreeSlider !=
    null)
  {
    Slider slider = scaleTreeSlider.GetComponent<Slider>();
    hitObject.transform.localScale = new Vector3( 1, 1, 1) * slider.value;
  }
}
```

Listing 5.20: Function for scaling trees.

It's also important that the slider's value, when the user targets a virtual object, is correct and reflects the current scale state of the object. This is achieved by the following function:

```
private void toggleSlider()
{
  if (editMode.editModeEnabled)
  {
    if (editMode.hitObject != null && editMode.hitObject.tag == "Tree")
    {
      if (!scaleTreeSlider.activeSelf || lastHitTree != editMode.hitObject)
      {
        scaleTreeSlider.SetActive(true);
        Slider slider = scaleTreeSlider.GetComponent<Slider>();
        slider.value = editMode.hitObject.transform.localScale[0];
        lastHitTree = editMode.hitObject;
      }
    }
```

```
15      else
16      {
17        scaleTreeSlider.SetActive(false);
18      }
19    }
20  }
```

Listing 5.21: Enabling the scale slider and setting it to the correct value.

5.8 Location 2 Screen (Building)

This is the AR Location in which the user may place a virtual building at Ouellington Street, in Nea Chora. After placing the building, the user may also delete or rotate it, as well as change the number of floors that the building has.

5.8.1 Location 2 scene hierarchy and components

The base elements hierarchy is the same as in the other locations. As mentioned before the main changes are the UI elements, the different tutorial messages and the different attached components.

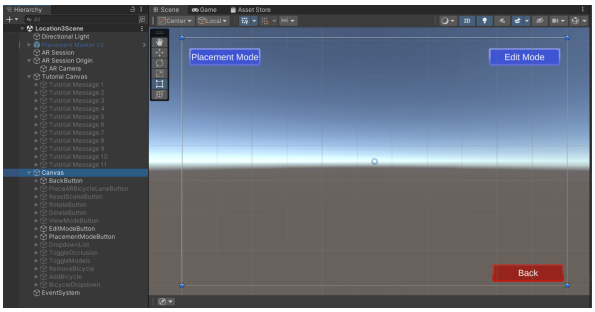


Figure 5.48: Location 2 Scene hierarchy and components.

5.8.2 Building Model

The building model was created using Unity's ProBuilder. ProBuilder equipped us with the tools needed in order to edit the geometry of Unity objects and create all the components that make up our building model. We have created models with variable numbers of floors (ranging from 1 to 6, based on the current construction laws for Nea Chora).

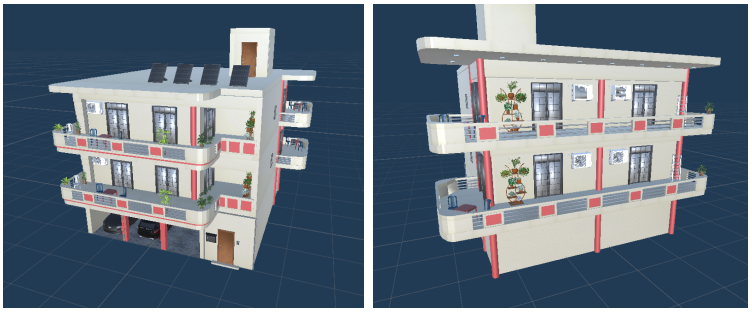


Figure 5.49: The building model, with 2 floors.

The building ground floor features a parking area with cars and a front door with an electronic doorbell. Each floor of the building features a front and a back balcony, decorations and side windows. Lastly, The roof features solar panels.

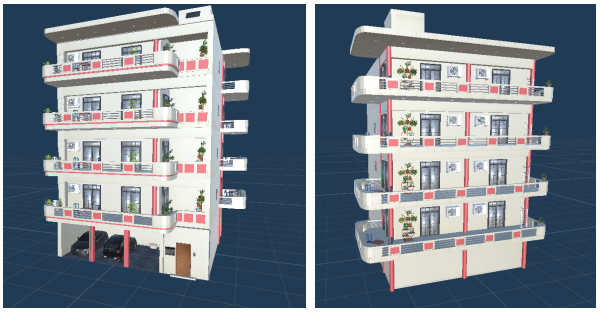


Figure 5.50: The building model, with 4 floors.

As seen in the figure below, the building model is comprised of a multitude of other models. A notable mention is the "Floors" prefab which represents each of the building's floors.

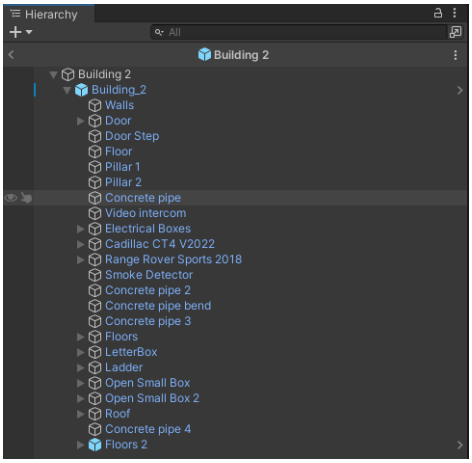


Figure 5.51: The hierarchy of the building model with 2 floors.

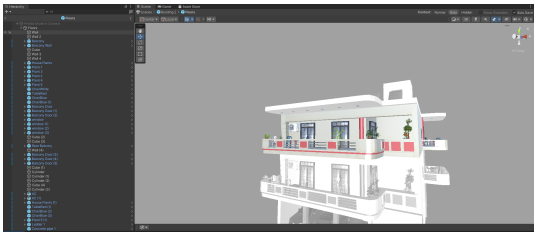


Figure 5.52: The hierarchy of the "Floors" prefab.

5.8.3 AR Modes

This location's AR functionality has been split into three AR Modes: View Mode, Placement Mode and Edit Mode, which are controlled by the ARBuildingManager script.

5.8.3.1 ARBuildingManager

The ARBuildingManager features functionality similar to the ARTreeManager, but specialized for the Building location.

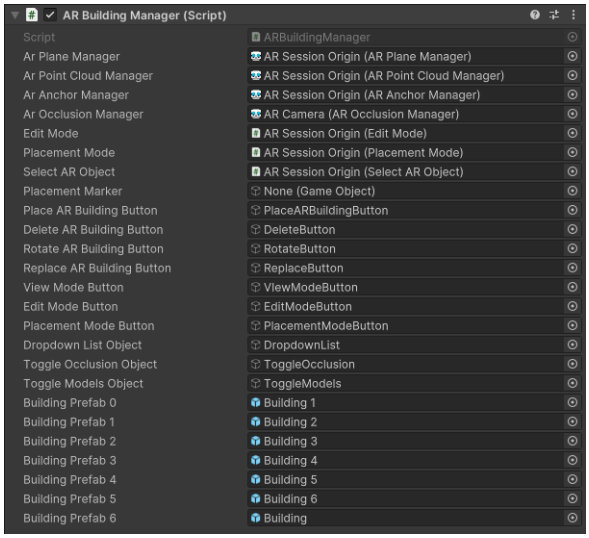


Figure 5.53: The ARBuildingManager component, attached to the scene's AR Session Origin.

5.8.3.2 View Mode

The View Mode functionality is the same as in all other AR locations of our application.

5.8.3.3 Placement Mode

The Placement Mode functionality is more or less the same as in the other AR locations of our applications, with the only changes being the text on some UI elements and the options for the virtual object models.

5.8.3.4 Edit Mode

As mentioned in section 5.6.2.4 the user may delete or rotate the placed virtual building, after selecting it. In addition to that they may also change the number of floors that the building has, ranging from 1 up to 6 floors.

This is achieved by replacing the currently placed building model with a new one that has the desired number of floors. The function that implements this functionality is the one shown below and is part of the Edit Mode script.

```
1 private void replaceObject(GameObject hitObject, List<GameObject>
      placedObjects)
2 {
3   if (hitObject != null && selectedPrefab != null)
```

```
4     {
5       Vector3 position = hitObject.transform.position;
6       Quaternion rotation = hitObject.transform.rotation;
7       placedObjects.Remove(hitObject);
8       Destroy(hitObject);
9       GameObject placedObject = Instantiate(selectedPrefab, position, rotation
        );
10      placedObjects.Add(placedObject);
11    }
12  }
```

Listing 5.22: Replacing the placed building model with a diffrent one to change the number of floors.

5.9 Location 3 Screen (Bicycle Lane and Bicycle Station)

This is the AR Location in which the user may place a virtual bicycle lane and a bicycle station at the intersection of Selinou Street and Georgiakakidon Street, in Nea Chora. After placing the bicycle lane and station, the user can delete or rotate them. Additionally, they may add or remove bicycles from the bicycle station, choosing form a selection of different bicycle models.

5.9.1 Location 3 scene hierarchy and components

The base elements hierarchy is the same as in the other locations. As mentioned before the main changes are the UI elements, the different tutorial messages and the different attached components.

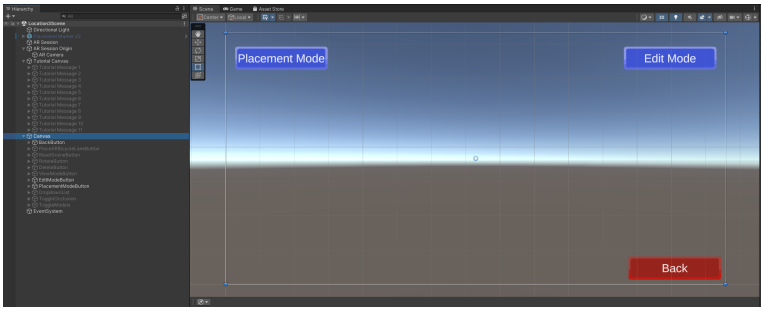


Figure 5.54: Location 3 Scene hierarchy and components.

5.9.2 Bicycle Lane Model

The bicycle lane model was created using the "Plane" Unity 3D object and then adding the correct textures to it.

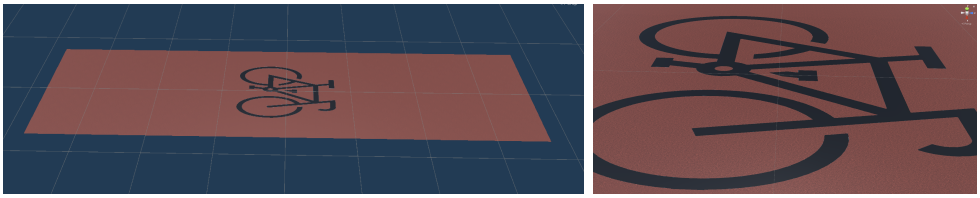


Figure 5.55: The Bicycle Lane model and a closer look at its texture.

In reality the prefab is comprised of five separate planes, stitched together to form a longer one. All those planes have the red-ish asphalt material on them, which gives them their color and asphalt texture. The middle plane has an extra material attached to it, which corresponds to the bicycle image that appears on the lane.

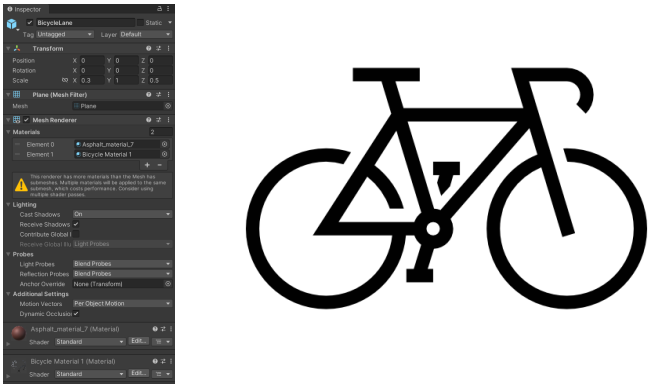


Figure 5.56: The two materials attached to the middle plane and the bicycle image.

5.9.3 Bicycle Models

Two bicycle models where selected [37] [38].

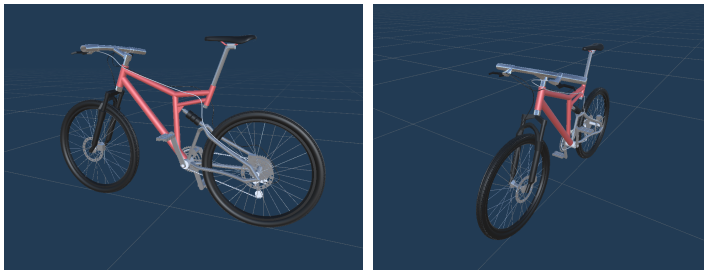


Figure 5.57: Bicycle 1.

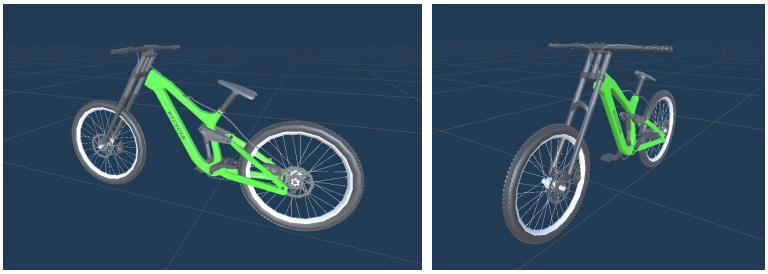


Figure 5.58: Bicycle 2.

5.9.4 Bicycle Station Model

A bicycle station model was created using ProBuilder and the bicycle models mentioned in the previous section. The model consists of a metal docking station for the bicycles, with four slots, and the bicycles themselves.

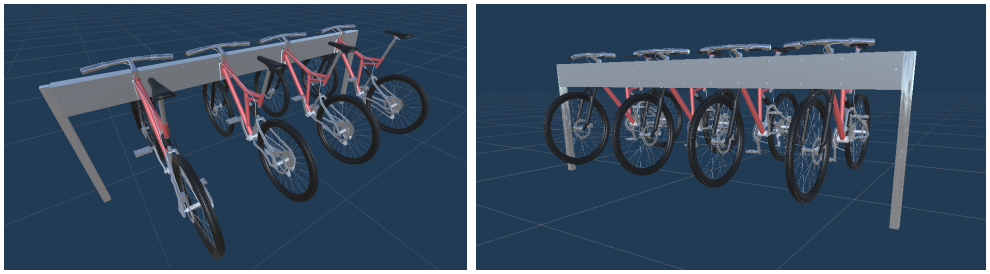


Figure 5.59: The Bicycle Station model.

5.9.5 AR Modes

This location's AR functionality has been split into three AR Modes: View Mode, Placement Mode and Edit Mode, which are controlled by the ARBicycleLaneManager script.

5.9.5.1 ARBicycleLaneManager

The ARBicycleLaneManager features functionality similar to the ARTreeManager, but specialized for the Bicycle Lane location.

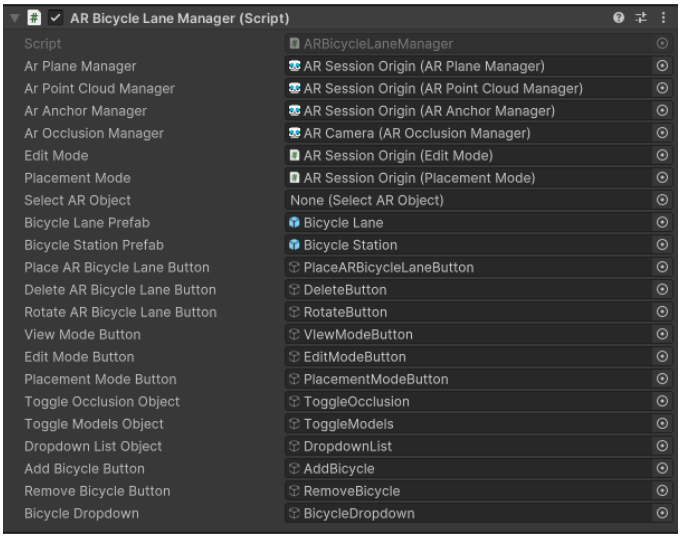


Figure 5.60: The ARBicycleLaneManager component, attached to the scene's AR Session Origin.

5.9.5.2 View Mode

The View Mode functionality is the same as in all other AR locations of our application.

5.9.5.3 Placement Mode

The Placement Mode functionality is more or less the same as in the other AR locations of our applications, with the only changes being the text on some UI elements and the options for the virtual object models.

The user can select either the bicycle lane or the bicycle station from a dropdown menu and then choose where to place them.

5.9.5.4 Edit Mode

As mentioned in section 5.6.2.4 the user may delete or rotate the placed virtual bicycle lane or bicycle station, after selecting it. In addition to that they may also add (choosing from the available selection of bicycle models) or remove bicycles from the bicycle station.

Removing and adding bicycles on the bicycle station meant being able to calculate the exact positions that the bicycles would be added in, since they have to align with the slots that the station has. For this reason we created

the "BicyclePrefabManager" script which is attached on the bicycle station model. This script stores the positions of the slots and dynamically adds each new bicycle to the next available slot.

```
public class BicyclePrefabManager  MonoBehaviour
{
    public const int BIKE_1 = 0;
    public const int BIKE_2 = 1;

    public GameObject bicycleStation = null;
    public GameObject bike1 = null;
    public GameObject bike2 = null;
    public ListGameObject bicyclePrefabs = new ListGameObject();

    private GameObject bicycleDropdown = null;

    private ListVector3 bicyclePrefabsVector3 = new ListVector3();

    void Start()
    {
        foreach (GameObject bicycle in bicyclePrefabs)
        {
            bicyclePrefabsVector3.Add(bicycle.transform.localPosition);
        }
    }

    public void removeBicycle()
    {
        if (bicyclePrefabs.Count = 1)
        {
            Destroy(bicyclePrefabs[bicyclePrefabs.Count - 1]);
            bicyclePrefabs.RemoveAt(bicyclePrefabs.Count - 1);

        }
    }

    public void addBicycle()
    {
        bicycleDropdown = GameObject.FindGameObjectsWithTag(BicycleDropdown)
    [0];
        if (bicyclePrefabs.Count  4 && bicycleDropdown != null)
        {
            TMP_Dropdown dropdown = bicycleDropdown.GetComponentTMP_Dropdown
    ();
            GameObject tmpbike = Instantiate(selectBikePrefab(dropdown.value
    ), bicycleStation.transform, worldPositionStays false);
            tmpbike.transform.localPosition = bicyclePrefabsVector3[
    bicyclePrefabs.Count];
            bicyclePrefabs.Add(tmpbike);
        }
    }

    public GameObject selectBikePrefab(int bikeOption)
    {
        switch (bikeOption)
        {
            case BIKE_1
                return bike1;
            case BIKE_2
                return bike2;
```

```
            default
                return bike1;
        }
    }
}
```

Listing 5.23: Bicycle manager script.

The final functions for removing and adding bicycles are implemented by the Edit Mode script, since we wanted to be consistent and have all the major function calls for editing AR content in the same class.

```
public void removeBicycle()
{
  if(hitObject != null && hitObject.tag == "BicycleStation")
  {
    BicyclePrefabManager bpm = hitObject.GetComponent<BicyclePrefabManager
    >();
    bpm.removeBicycle();
  }
}

public void addBicycle()
{
  if (hitObject != null && hitObject.tag == "BicycleStation")
  {
    BicyclePrefabManager bpm = hitObject.GetComponent<BicyclePrefabManager
    >();
    bpm.addBicycle();
  }
}
```

Listing 5.24: Removing and adding bicycles to the bicycle lane.

5.10 Tutorial System

The last component of our AR application is the implementation of an Tutorial System that teaches the user how to use the various features of application.

The first step towards developing this system was to create the tutorial message's appearance. As seen below, each tutorial message consists of: 1) a message title, 2) the main tutorial message text, 3) the step number and 4) the "Next" (or "Done" if it's the last message) button.

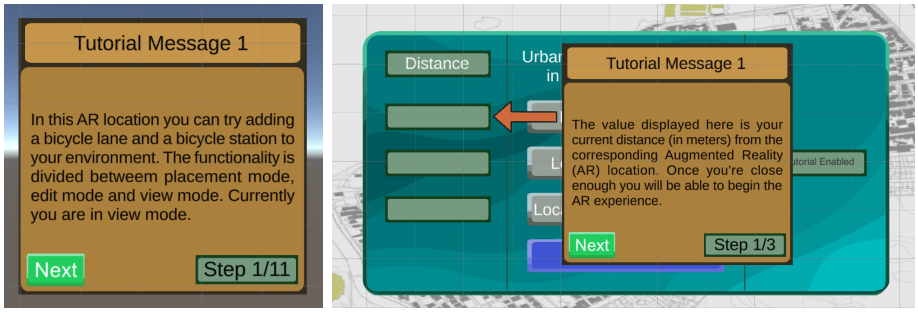


Figure 5.61: Examples of the tutorial messages' appearance.

Each of our Unity scenes has its own set of tutorial messages. We also wanted our tutorial to be persistent and not completely reset after the user closes the application, which is why utilized the Unity "PlayerPrefs". PlayerPrefs allows us to store values into the platform registry and retrieve them without loss of data between sessions.

All the tutorial messages of each scene are stored under a Tutorial Canvas object in our scene.

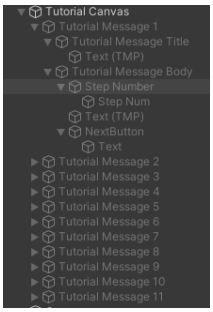


Figure 5.62: Tutorial canvas example (Location 1).

5.10.1 Enabling the tutorial system

Enabling the tutorial system is done via a checkbox that the user may "check" or "uncheck". This results in the following function being called. This function resets all tutorial steps if the tutorial is enabled anew.

```
public void toggleTutorial()
{
  if (tutorialToggle.isOn && !tutorialEnabled)
  {
    tutorialEnabled = true;
    PlayerPrefs.SetInt(TUTORIAL_STEP_HOME_SCREEN, INDEX_VALUE_ZERO);
```

```
7       PlayerPrefs.SetInt(TUTORIAL_STEP_MAP_SCREEN, INDEX_VALUE_ZERO);
8       PlayerPrefs.SetInt(TUTORIAL_STEP_LOCATION_1_SCREEN, INDEX_VALUE_ZERO);
9       PlayerPrefs.SetInt(TUTORIAL_STEP_LOCATION_2_SCREEN, INDEX_VALUE_ZERO);
10      PlayerPrefs.SetInt(TUTORIAL_STEP_LOCATION_3_SCREEN, INDEX_VALUE_ZERO);
11      PlayerPrefs.SetInt(TUTORIAL_ENABLED, TUTORIAL_ENABLED_YES);
12    } else if (!tutorialToggle.isOn)
13    {
14      tutorialEnabled = false;
15      PlayerPrefs.SetInt(TUTORIAL_ENABLED, TUTORIAL_ENABLED_NO);
16    }
17  }
```

Listing 5.25: Enabling or disabling the Tutorial System.

5.10.2 Tutorial Scripts

Since each scene has its own tutorial messages, we created a Tutorial script for each of them that acts as a manager for when each message should be displayed. These scripts keep track of which tutorial messages have already been shown, what the current AR mode is (if it's a location scene) since some tutorial messages are mode dependent and then show the appropriate (if any) message to the user.

```
1   public void nextTutorialMessage()
2   {
3     if (index >= 2 && index < 7)
4     {
5       if (PlayerPrefs.GetInt(ARManager.CURRENT_MODE, DEFAULT_VALUE) ==
        ARManager.PLACEMENT_MODE)
6       {
7         enableTutorialMessage(index, TUTORIAL_STEP_LOCATION_1_SCREEN);
8       }
9     } else if (index == 7)
10    {
11      if (PlayerPrefs.GetInt(ARManager.CURRENT_MODE, DEFAULT_VALUE) ==
        ARManager.VIEW_MODE)
12      {
13        enableTutorialMessage(index, TUTORIAL_STEP_LOCATION_1_SCREEN);
14      }
15    } else if (index >= 8 && index <= tutorialMessages.Count -1)
16    {
17      if (PlayerPrefs.GetInt(ARManager.CURRENT_MODE, DEFAULT_VALUE) ==
        ARManager.EDIT_MODE)
18      {
19        enableTutorialMessage(index, TUTORIAL_STEP_LOCATION_1_SCREEN);
20      }
21    }
22    else if (index >= 0)
23    {
24      if (PlayerPrefs.GetInt(ARManager.CURRENT_MODE, DEFAULT_VALUE) ==
        ARManager.VIEW_MODE)
25      {
26        enableTutorialMessage(index, TUTORIAL_STEP_LOCATION_1_SCREEN);
27      }
28    }
```

```
}
```

Listing 5.26: Showing the next tutorial message in location 1.

This is also where we implement the persistent messages functionality, by having our scripts access PlayerPrefs each time the scene if first accessed to check what the last shown tutorial message was before the user closed the application.

```
//if the tutorial was enabled last time the app opened, the user should
    continue from the step he left off
if (PlayerPrefs.GetInt(TUTORIAL_ENABLED, DEFAULT_VALUE) ==
    TUTORIAL_ENABLED_YES)
{
  tutorialEnabled = true;
  index = PlayerPrefs.GetInt(TUTORIAL_STEP_LOCATION_1_SCREEN,
    INDEX_VALUE_ZERO);

  if (index < tutorialMessages.Count && index >= 0)
  {
    tutorialMessages[index].SetActive(true);
  }
}
```

Listing 5.27: Resuming the tutorial from where the user last left off in location 1.

Chapter 6

Evaluation & Testing

6.1 Introduction

In this chapter, we'll go over the method used for the evaluation of our mobile AR application, our participant group and lastly the results of our evaluation.

6.2 Evaluation Method

The method that we employed for the evaluation of our mobile AR application was the "Think-Out-Loud" evaluation method. This is a usability method that entails having the participants give continuous verbal feedback while using a system, like our android application.

Our participants group consisted both of students of the Architecture Department of the Technical University of Crete and of people unaffiliated with architecture and urban planning. Each participant was given an Android phone (Motorola G 5G, Android Version 11) with the application installed. Then they were tasked with using the AR application while verbally expressing their thoughts, opinions or suggestions.

Each participant was allowed to freely explore the application and all of them chose to follow the in-app tutorial, in order to get a better understanding of how to navigate and use our system.

The evaluation's starting point was near Location 1, outside the range in which the corresponding location button gets enabled. This meant that our user's would have to utilize the application's map screen in order to navigate towards the first location. Once they reached their destination, they were

free to experiment with the augmented reality capabilities of the application and create their own unique scene. Lastly, once they were satisfied with their results, the participants would move on to navigating to the next AR Location and repeating the process there.

6.3 Evaluation Results & Limitations

The overall reception of our application by the participants of our evaluation process was very positive. Below we'll take a closer look at some of the comments and suggestions that were provided by them.

6.3.1 User Interface (UI) and User Interaction

As far as the User Interface (UI) of our application is concerned, the majority of the users were able to navigate it without issues and most of the comments we received on it were positive.

1. The participants that were students of the Architecture Department of the Technical University of Crete recognized the main screen background as an architectural plan and noted that it's a good design choice if our target audience consists mainly of architects. The rest of the participants didn't comment on it.

2. A few users commented that they would prefer it if the tutorial message pop-up windows all appeared in the same position (namely at the center of the screen). Seeing as this was not always possible due to potential obstruction of essential UI elements related to the tutorial messages, we decided to keep the implementation of the tutorial system as is.

3. Some of the participants suggested that it would be more intuitive if the rotation of the AR objects was done via touch-screen motions (e.g. scrolling with their finger) as opposed to the current implementation which utilizes a scrollbar.

4. There was some minor confusion on how to replace floors of an already placed AR building, so we edited the tutorial messages related with it to ensure that our users receive clear instructions regarding this functionality.

6.3.2 Map Navigation

The reception of our map navigation screen was also positive. The participants made very few comments on it, noting that it's more or less what they've come to expect when using a map on a modern mobile application. The motion gestures for zooming and moving the map felt natural and intuitive.

One of the problems that some participants encountered, was that the navigational line connecting their current location to their destination would sometimes not be displayed correctly. Additionally, in certain cases the application would change the proposed path mid navigation. These were both known issues and are related to how the MapBox Navigation API calculates and displays navigational data. Hopefully, future updates on their API and SDK will provide a solution to these issues and improve upon the accuracy of their navigational capabilities.

6.3.3 Floor Detection, Object Tracking and Other Technical Issues

The issues listed below are those inherently connected with the limitations of the technology used for our implementation and also the capabilities of the device used for our evaluation process, and therefore were impossible for us to improve upon currently.

1. The application was able to detect the floor plane easily when the weather conditions and lighting was good, but would take longer when dealing with poorer illumination and other conditions that hinder feature detection. This can be attributed to the fact that our implementation uses the ARFoundation SDK to implement it's plane tracking capabilities. This is a markerless approach and as such is reliant on good illumination and the existence of identifiable features in the environment.

2. The participants noticed that in some cases placed AR objects would not stay completely static but instead change positions, once again when the weather and illumination conditions were not great. This can also be attributed to our markerless approach.

3. Some virtual objects would take a long time to load.

4. Displaying a building with a lot of floors or adding a large number of them to an existing building was a resource intensive feature and oftentimes caused visual glitches and framerate drops to the application.

The issues related with the environmental conditions and the environmental illumination are sadly some of the inherent drawbacks of using a markerless, feature-based tracking technique. We hope that future research in this sector might improve upon them and create more robust systems that are less affected by their environment. Lastly, the two issues concerning the performance of the application can be eliminated by using a high end mobile device with better technical characteristics than the one that was available to us.

Chapter 7

Conclusions & Future Work

7.1 Introduction

In this chapter, we'll provide our final thoughts and overview of the developed augmented reality application, as well as present areas for future work and potential improvements.

7.2 Conclusions

In conclusion, we have developed a fully functional augmented reality application for mobile devices that can visualize proposed urban planning changes. Our system, utilizing the framework provided by ARFoundation, can accurately perform plane detection, virtual object placement, persistent tracking, anchoring and object occlusion, thus providing a robust augmented reality experience. This, combined with a wide selection of different 3D models, the three distinct target locations and the different ways the user can interact with each of the virtual objects, allows the users to create scenes that are distinct from each other and could each represent a unique, urban planning proposal for the selected location. Additionally, our map, navigation and localization implementation ensures that the user will be able to smoothly navigate to the target locations. Lastly, our tutorial system increases how accessible our application is and eliminates the need for previous familiarity with augmented reality.

As shown from our evaluation results, our system was received positively both by individuals related to the sector of architecture and urban planning but also by user's unfamiliar with those processes. All the participants agreed that our application, and by extension augmented reality, could be

a powerful supplementary tool for the urban planning process. In addition, the tutorial system, the simple but efficient user interface and the selection of mobile devices as a target platform, ensured that all our evaluation participants were able to effectively use our application regardless of their technological background. These observations lead to the conclusion that mobile augmented reality is highly suitable for applications such as this, where the target audience might not be familiar with augmented reality or more complex equipment.

Overall, our application proves that augmented reality can be used to aid and enhance the urban planning in ways that wouldn't be possible with conventional techniques. We hope that this thesis can serve as both a showcase of those capabilities of AR but also as a stepping stone for further research and advancements in this sector.

7.3 Future Work

After reviewing our evaluation results, our development process and the final state of our application, we now possess a better understanding of how we could improve upon our current implementation. As such, we would like to propose a few potential improvements, additions and avenues for future development of this application.

Additional target locations, virtual object types and interactions
First of all, one of the obvious paths for expanding our mobile AR application would be the addition of more target locations in Nea Chora or even other urban areas in Chania and beyond. Additionally, to support those new locations and also expand upon the already existing ones, we could add an even larger variety of virtual object types and new ways for the user to interact with them or existing ones. Those additions would result to even more complex representations of urban planning changes proposals.

User submitted 3D models
An improvement stemming from our previous mention of adding more 3D models, would be the implementation of a system that allows user's to submit their own models to the applications. These models would then be available for use in all the locations and scenes that the user creates or edits.

Online anchors and design sharing
Another possible way to improve upon the current design would be to imple-

ment the ability to save already created AR scenes, by utilizing cloud anchors (e.g. Google's Cloud Anchors) and storing the real-world positions of virtual objects in an online database (e.g. Google's Firebase). This would allow users to save their progress and would be the first step towards creating an online system through which users can share and view each others designs in real time. A system like that could also feature user comments, votes and various other social ways to interact with the urban planning process.

Collaborative scene creation
Expanding on the previous idea, we could implement a system through which multiple people could collaborate and edit the same scene to create one final result. This could feature both synchronous, real-time editing during which each collaborator sees their teams changes happening "live" and also asynchronous editing simply by loading the augmented reality see and reviewing the changes made by other members.

Tracking improvements
Lastly, it's worth noting that augmented reality, and especially the field related to environmental and object tracking, is a fast developing sector and new research is constantly producing more robust and reliant ways to implement markerless tracking techniques. Any such improvements can drastically enhance our applications performance in sub-optimal environmental conditions.

Bibliography

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

[2] Loomis, Jack & Golledge, Reginald & Klatzky, Roberta & Speigle, Jon & Tietz, Jerome. (1994). Personal guidance system for the visually impaired, 85-91. 10.1145/191028.191051.

[3] Delgado, Frank & Altman, Scott & Abernathy, Michael & White, Janis. (2000). Virtual cockpit window for the X-38 crew return vehicle. 10.1117/12.389361.

[4] Milgram, Paul & Kishino, Fumio. (1994). A Taxonomy of Mixed Reality Visual Displays. IEICE Trans. Information Systems. vol. E77-D, no. 12. 1321-1329.

[5] Milgram, Paul & Takemura, Haruo & Utsumi, Akira & Kishino, Fumio. (1994). Augmented reality: A class of displays on the reality-virtuality continuum. Telemanipulator and Telepresence Technologies. 2351. 10.1117/12.197321.

[6] Thomas, Bruce & Close, B. & Donoghue, J. & Squires, J. & Bondi, P. & Morris, M. & Piekarski, Wayne. (2000). ARQuake: An outdoor/indoor augmented reality first person application. 4th International Symposium on Wearable Computers. 6. 139-146. 10.1109/ISWC.2000.888480.

[7] Newman, Joseph & Ingram, David & Hopper, Andy. (2001). Augmented reality in a wide area sentient environment. Proceedings of the 2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001). 77 - 86. 10.1109/ISAR.2001.970517.

[8] Zhou, Feng & Duh, Henry & Billinghurst, Mark. (2008). Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR. 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality. 2. 193-202. 10.1109/ISMAR.2008.4637362.

[9] Koulieris, George & Akşit, Kaan & Stengel, Michael & Mantiuk, R. & Mania, Katerina & Richardt, C.. (2019). Near-Eye Display and Tracking Technologies for Virtual and Augmented Reality. Computer Graphics Forum. 38. 493-519. 10.1111/cgf.13654.

[10] Haesler, Steffen & Kim, Kangsoo & Bruder, Gerd & Welch, Greg. (2018). Seeing is Believing: Improving the Perceived Trust in Visually Embodied Alexa in Augmented Reality. 204-205. 10.1109/ISMAR-Adjunct.2018.00067.

[11] Kim, Kangsoo & Bölling, Luke & Haesler, Steffen & Bailenson, Jeremy & Bruder, Gerd & Welch, Greg. (2018). Does a Digital Assistant Need a Body? The Influence of Visual Embodiment and Social Behavior on the Perception of Intelligent Virtual Agents in AR. 105-114. 10.1109/ISMAR.2018.00039.

[12] Cai, Minya & Alfian Amrizal, Muhammad & Abe, Toru & Suganuma, Takuo. (2019). Design of an AR-Based System for Group Piano Learning. 20-21. 10.1109/ISMAR-Adjunct.2019.00020.

[13] Lin, Chengyuan & Andersen, Daniel & Popescu, Voicu & Rojas, Edgar & Cabrera, Maria & Mullis, Brian & Zarzaur, Ben & Anderson, Kathryn & Marley, Sherri & Wachs, Juan. (2018). A First-Person Mentee Second-Person Mentor AR Interface for Surgical Telementoring. 3-8. 10.1109/ISMAR-Adjunct.2018.00021.

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

[15] T. Caudell and D. Mizell. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on, volume ii, page 659-669, 1992.

[16] Arth, Clemens & Grasset, Raphael & Gruber, Lukas & Langlotz, Tobias & Mulloni, Alessandro & Wagner, Daniel. (2015). The History of Mobile Augmented Reality.

[17] Cirulis, Arnis & Brigmanis, Kristaps. (2013). 3D Outdoor Augmented Reality for Architecture and Urban Planning. Procedia Computer Science. 25. 71–79. 10.1016/j.procs.2013.11.009.

[18] Ghadirian, Payam & Bishop, Ian. (2008). Integration of augmented reality and GIS: A new approach to realistic landscape visualisation. Landscape and Urban Planning. 86. 226-232. 10.1016/j.landurbplan.2008.03.004.

[19] Huang, Wei & Sun, Min & Li, Songnian. (2016). A 3D GIS-based Interactive Registration Mechanism for Outdoor Augmented Reality System. Expert Systems with Applications. 55. 10.1016/j.eswa.2016.01.037.

[20] Carozza, Ludovico & Tingdahl, David & Bosché, Frédéric & Van Gool, Luc. (2014). Markerless Vision-Based Augmented Reality for Urban Planning. Computer-Aided Civil and Infrastructure Engineering. 29. 10.1111/j.1467-8667.2012.00798.x.

[21] Kouklaki, Despoina & Raphsomaki, Vasileia (2015), Πολεοδομική ανασυγκρότηση της συνοικίας της Νέας Χώρας στην πόλη των Χανίων. Diploma Thesis, School of Architecture, Technical University of Crete.

[22] Aggelopoulou, Maria & Matzouni, Kalliopi & Sylligardos, Dimitris & Tsiorolis, Giorgos (2018), Πολεοδομική ανασυγκρότηση της συνοικίας της Νέας Χώρας στην πόλη των Χανίων. Semester project for Urban Planning 1, School of Architecture, Technical University of Crete.

[23] Apostolopoulou-Vythoulka, Myrto & Nikou, Rafaela-Anastasia(2018), Πρόταση πολεοδομικής αναδιοργάνωσης Η περίπτωση της Κισσάμου. Diploma Thesis, School of Architecture, Technical University of Crete.

[24] K. Kim, M. Billinghurst, G. Bruder, H. B. -L. Duh and G. F. Welch, "Revisiting Trends in Augmented Reality Research: A Review of the 2nd Decade of ISMAR (2008–2017)," in IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 11, pp. 2947-2962, Nov. 2018, doi: 10.1109/TVCG.2018.2868591.

[25] L. B. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," Proceedings of IEEE Virtual Reality Annual International Symposium, Seattle, WA, USA, 1993, pp. 76-82, doi: 10.1109/VRAIS.1993.380795.

[26] Anders Henrysson, Mark Billinghurst, and Mark Ollila. 2006. AR tennis. In ACM SIGGRAPH 2006 Emerging technologies (SIGGRAPH '06). Association for Computing Machinery, New York, NY, USA, 1–es. https://doi.org/10.1145/1179133.1179135

[27] Rabbi, Ihsan & Ullah, Sehat. (2013). A Survey of Augmented Reality Challenges and Tracking. ACTA GRAPHICA. 24. 29-46.

[28] Gherghina, A., Olteanu, A., & Tapus, N. (2013). A marker-based augmented reality system for mobile devices. 2013 11th RoEduNet International Conference. doi:10.1109/roedunet.2013.6511731

[29] Caarls, J., Jonker, P., Kolstee, Y., Rotteveel, J., & van Eck, W. (2009). Augmented Reality for Art, Design and Cultural Heritage—System Design and Evaluation. EURASIP Journal on Image and Video Processing, 2009, 1–16. doi:10.1155/2009/716160

[30] Lowney, M., & Raj, A. S. (2016). Model based tracking for augmented reality on mobile devices. Computer Science, 1-5.

[31] Bibri, S. E., & Krogstie, J. (2017). Smart sustainable cities of the future: An extensive interdisciplinary literature review. Sustainable Cities and Society, 31, 183–212. doi:10.1016/j.scs.2017.02.016

[32] Kamel Boulos, M. N., Lu, Z., Guerrero, P., Jennett, C., & Steed, A. (2017). From urban planning and emergency training to Pokémon Go: applications of virtual reality GIS (VRGIS) and augmented reality GIS (ARGIS) in personal, public and environmental health. International Journal of Health Geographics, 16(1). doi:10.1186/s12942-017-0081-0

[33] Allen, M., Regenbrecht, H., & Abbott, M. (2011). Smart-phone augmented reality for public participation in urban planning. Proceedings of the 23rd Australian Computer-Human Interaction Conference on - OzCHI '11. doi:10.1145/2071536.2071538

[34] Boos, Ursina & Reichenbacher, Tumasch & Kiefer, Peter & Sailer, Christian. (2022). An augmented reality study for public participation in urban planning. Journal of Location Based Services. 17. 1-30. 10.1080/17489725.2022.2086309.

[35] Mahmoud, H., & Akkari, N. (2016). Shortest Path Calculation: A Comparative Study for Location-Based Recommender System. 2016 World Symposium on Computer Applications & Research (WSCAR). doi:10.1109/wscar.2016.16

[36] Della Porta, Giambattista. Magia naturalis. Vol. 2. 1714.

[37] Unity Asset Store. (n.d.). Unity Asset Store. `https://assetstore.unity.com/`

[38] Turbosquid. (n.d.). Shutterstock. https://www.turbosquid.com/

[39] ARCore. (n.d.). Google, Inc.. https://developers.google.com/ar

[40] ARKit. (n.d.). Apple, Inc.. https://developer.apple.com/augmented-reality/arkit/

[41] Vuforia Engine. (n.d.). PTC, Inc.. https://developer.vuforia.com/home

[42] artoolkitX. (n.d.). artoolkitX. https://www.artoolkitx.org/

[43] Wikitude. (n.d.). Qualcomm. https://www.wikitude.com/products/wikitude-sdk/

[44] EasyAR. (n.d.). VisionStar Information Technology Co.. https://www.easyar.com/