Hochschule für Technik Stuttgart

University of Applied Sciences

Master of Science Programme
Photogrammetry and Geoinformatics
Master Thesis
Winter Term 2020/2021

Outdoor Mobile Augmented Reality in Urban Planning;

Concepts of Visualizing Focus and Context Using Grayscale Filter in Video-based AR.

by

Justin Jing Hui

Supervisors: Prof. Dr. -Ing. Volker Coors
Prof. Dr. -Ing. Eberhard Gülch

Outdoor Mobile Augmented Reality in Urban Planning; Concepts of Visualizing Focus and Context Using Grayscale Filter in Video-based AR.

by

Justin Jing Hui

A dissertation presented in partial fulfillment of the requirements for the degree of Master of Science in the Department of Geomatics, Computer Science and Mathematics, Stuttgart University of Applied Sciences

Declaration

The following Master thesis was prepared in my own words without any additional help. All used sources of literature are listed at the end of the thesis.

I hereby grant to Stuttgart University of Applied Sciences permission to reproduce and to distribute publicly paper and electronic copies of this document in whole and in part.

Stuttgart, 28.02.2021 (Justin Jing Hui)

Approved by:

(Prof. Dr. -Ing. Volker Coors)

Acknowledgement

A lot of help was given to me to finish this thesis. Without these generous help, the last six months would have been a much harder journey to take on. I would like to express my gratitude to Prof. Dr. Volker Coors for being the supervisor of this thesis, giving me guidance, inspiration and support with great patience and kindness.

Thank you also Prof. Dr. Eberhard Gülch for sharing your insights. Your generosity helps me to complete this thesis.

Martin Storz, Habiburrahman Dastageeri and Simone Weil have provided me knowledge and advice since the start of this thesis with their expertise and experience in the field of augmented reality development. I sincerely appreciate the generous help.

I have been extremely grateful to complete this research as well as my master studies with the unfailing support and assistance from all Professors and my dear classmates. Thank you for all your encouragement.

Justin Jing Hui

20 February 2021

Master Course Photogrammetry and Geoinformatics

Outdoor Mobile Augmented Reality in Urban Planning; Concepts of Visualizing Focus and Context Using Grayscale Filter in Video-based AR.

Abstract

Urban planners and designers often use grayscale masking filters to highlight a selected urban feature and its context without the overwhelming information. With the rising interest in applying augmented reality (AR) technology into the field of urban planning for increasing social engagement, this thesis aims to develop an outdoor mobile AR application that visualizes the urban environment using the mentioned grayscale masking filters. The application is able to apply the grayscale masking filter to the designated buildings in Nordbahnhof, Stuttgart when the device's camera is pointing to them. State of the art of outdoor mobile AR technologies are studied and reviewed. This thesis proposes the concepts and methodology of the video-based AR visualization and the development of the app. The implementation of the concepts and methodology is documented and the implementation process and result are evaluated.

Keywords: Augmented Reality, Outdoor Tracking, Real time visualization, Urban Planning

Table of Content

Table of Figures

Table of Tables

Abbreviations

API	Application Programming Interface
AR	Augmented Reality
GIS	Geographic Information System
LOD	Level of Detail
MARS	Mobile Augmented Reality System
SDK	Software Development Kit

1 Introduction

Figure 1 Visualization of a building in Stuttgart Nordbahnhof Building with grayscale filter and mask edited with Photoshop

Architects, landscape architects, urban designers and urban planners use visual as their primary means of communication. While dealing with the complex and dynamic situation of urban development along with numerous involving stakeholders and physical and abstract processes, the professionals rely strongly on visualization to convey their findings, ideas, and aspirations. (Smith, Dodge and Doyle, 1998) Throughout the years, different visualization technologies, methods and skills have been tested, developed and adopted into the field. Photos, drawings, and diagrams have always been important media. As a landscape architect, Ian McHarg introduced his map-overlay method in the 1960s and later became one of the most significant pioneers of Geographic Information System (GIS). (Turner, 2017) Decades later, landscape architects and planners continue to explore the possibility of using GIS to aid regional planning and visualizing analysis. (Nijhuis, 2016) 3D city modeling and photorealistic rendering are also common visualization techniques applied to the field which provides users an immersive experience and a better understanding of what the design would turn out to be.

Architects and planners develop their visual communication skills through learning, experience, and experiment. Through experimentation and exploration, effective visualization methods would be adopted and utilized. (Pingale, Damugade and Jirge, 2017) In this thesis, one visualization technique, which is commonly adopted by professionals, is selected to be the focus. During the site analysis process, architects and planners would often use photos to

provide a first brief impression of the site. Instead of using the photos directly, professionals would choose to apply a grayscale filter to the image, while leaving the select feature to be in color, when they are trying to guide the readers to focus on the said feature. This visualization technique creates attention towards a selected building or urban element while still providing surround context information without overwhelming the reader. In current practice, this visualization is usually created with photo editing software such as Adobe Photoshop with still images.

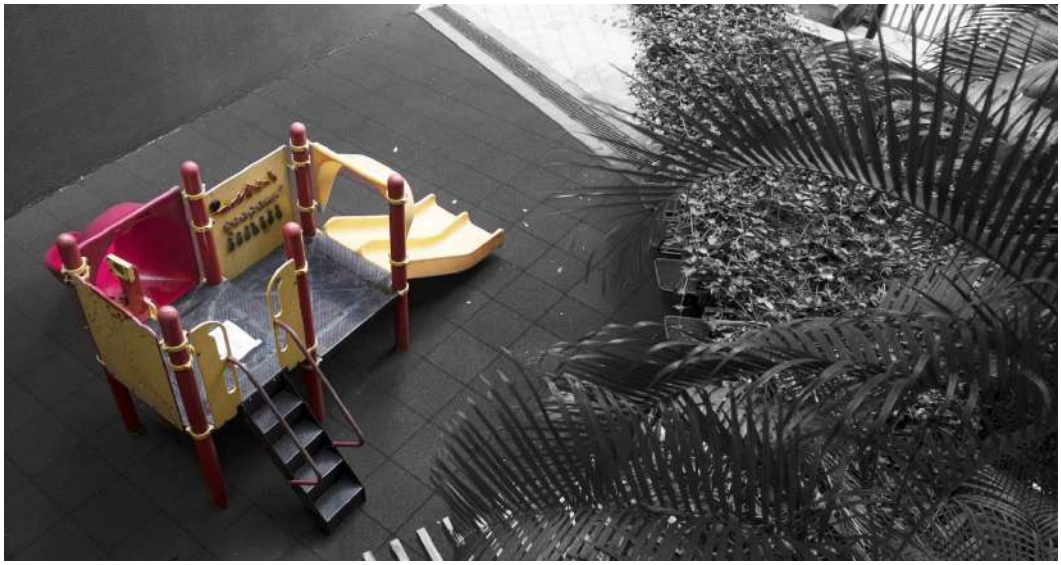

Figure 2 Landscape Architect using the grayscale filter to highlight urban features. (Leung, 2020)

While social participation has become more important yet more complicated in the process of urban planning, the quality of communication is crucial. Unlike the professionals, usual citizens are not trained to read and understand complex diagrams and drawings. Presenting accessible information that is understandable for the general public could reduce the difficulties and complexity of social participation. (Kirkhaug, 2016) Augmented reality (AR) could be an answer to this call. Mobile AR application provides users an interactive immersive experience that connects to the real world which would be able to talk to the user in an understandable presentation. (ARPost, 2019) Mobile AR applications have been and are being developed for use in the urban planning and design field in recent years. Examples can be seen that professionals are using mobile AR technologies to showcase historical buildings or their own design. Utilizing AR technology in the field is a rising trend and will gain further attention and investigation from now on. (Lypchenko, 2019)

Seeing the possibility of the fast-developing AR technology and the need for better-presented visualization in the urban planning and design field, this project aims to develop a mobile AR application that allows users to apply the grayscale masking filter visualization to buildings and special urban features. The application should let the users find the designated building and in real-time apply the grayscale filter to the surrounding context. The selected building would be left in color. This project would use the Nordbahnhof District in Stuttgart as the selected site. Using the district as the testing site on one hand would be a good chance to test the application in a well-constructed urban area with several different kinds of buildings. With the application, users would be able to visualize the building in the said area with the mentioned style. The state of the art of outdoor mobile AR technologies will be studied and reviewed at the early stage of the thesis to find out applicable existing technology. The thesis would provide the concepts and methodology of the app development, and documentation of its implementation. An evaluation of the application will also be carried out in order to seek the range of possible improvements.

1.1 Related research

AR experience in urban planning and design

Augmented reality has been applied to the field of urban planning and design. Lee et al (2012) created the CityViewAR application providing AR information visualization in Christchurch, New Zealand. The city experienced several major earthquakes in 2010 and 2011. The app was designed to aid the recovery of the destroyed buildings and the future development of the city after the natural disaster. It provides geolocated information of the destroyed historical buildings as well as the 3D model of the buildings on site. This project provides the general public a richer experience of the city and also provided a guideline for the future development of outdoor mobile AR applications.

Figure 3 CityViewAR showing buildings in the city before the damage. (Lee et al, 2012)

Anagnostou and Vlamos (2011) designed Square AR, an application that also aims to encourage civic participation in the process of urban planning. This application allows users to overlay layers such as vegetation, transportation and buildings on top of printed 2D maps virtually with a simple gridded system, which allows different individuals to contribute their envisions towards the city's future.

As a more commercially focused project, YARD is an application developed by Structure Studios (2018). The application allows users to place pre-built 3D models onto a designated site, visualizing their design in real life. The realistic visualization allows designers and planners to communicate with their clients more effectively and efficiently. The application also provides a

series of sample 3D features such as benches and planters in different styles, allowing them to brainstorm new ideas and demonstrate them in a fast exchange.

Apart from the practical implementation of AR technology by developers, urban designers would also envision what the technology could bring to the community from their own point of view. Malishev (2014) gave his vision on how AR could possibly change the urban landscape in the conceptual project CELOSCAPE. In the project, he envisioned that AR could play opportunities inside a dense urban center, which creates an extension of the public space. The play opportunities would also encourage human interaction and build a more cohesive community.

Real time masking

The concept of applying masks onto the surrounding environment to alter the way the world looks had been applied to projects from different fields of practice. The mentioned project by Blanco-Pons et al (2019) is masking the buildings in the site with their historical facade. It renders the facade according to the real time lighting and environment to provide a better feel of realism. Paint company Sherwin-Williams developed the application Colorsnap Visualizer with Google's AR Core SDK. (2019) The application allows users to change the color of the paint on the wall of their home to test out different color combinations and figure out which paint to buy. This visualization method is similar to the one that is being focused in this project and could be a useful reference point in terms of performance and user experience.

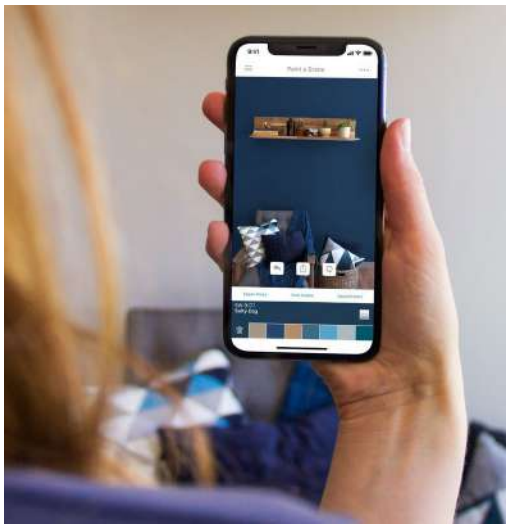

Figure 4 Real time swapping of wall paint color using Colorsnap Visualizer (Sherwin-Williams, 2019)

1.2 Objectives/ Key questions

Seeing the contextual and technological opportunities, this project aims to develop an outdoor mobile AR application that visualizes buildings and context with the grayscale masking filter that could cater to the need of urban planners and designers. The project will have the following focus questions:

1. What is the current state of the art development in terms of outdoor mobile augmented reality technology and its possible application to the urban planning and design field?

2. How to apply an outdoor tracking system for building recognition in the case of Stuttgart Nordbahnhof District?

3. Is it possible to apply a mask grayscale filter on buildings and context with video-based AR, that the visualization would assist the communication of urban planners and designers?

2 State of the Art

2.1 Mobile AR

Defined by Azuma (1997) and widely acknowledged later, Augmented Reality as a system has three characteristics: it combines real and virtual, is interactive in real time, and is registered in 3-D. The integration of the real world and virtual contents allows users to receive sensory experiences beyond reality in real time. (Qiao et al, 2019) The development of AR technologies began in the 1960s. In 1968, Ivan Sutherland created the world's first AR system. The system operates on a head-mounted see-through display with one mechanical and one ultrasonic sensor. This AR system was able to display simple wireframe graphics on top of the optics in real time. (Arth et al, 2015)

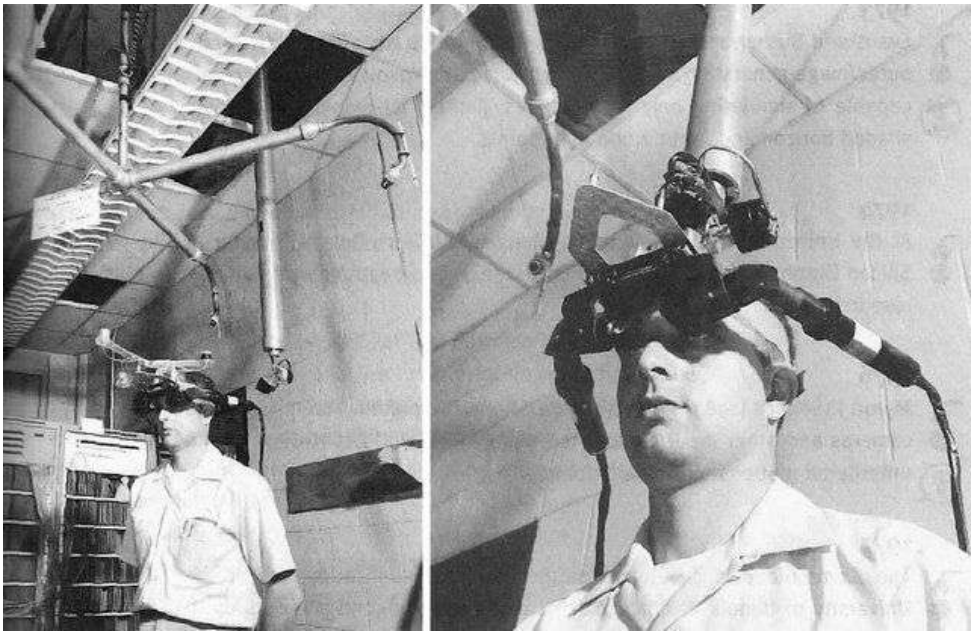

Figure 5 Ivan Sutherland's world's first AR system in 1968 (Basu, 2019)

In the following decades, the development of laptop and handheld computers, mobile phones, navigation systems and outdoor sensors laid the foundation for the possible future development of mobile augmented reality. In 1997, a "Touring Machine" was created by Feiner et al, which is known as the first mobile AR model prototype. The prototypical model attempts to combine AR technology and the mobile computer interface that was under development at the time. AR was usually applied for fields such as assisting surgery operation or flight control, while this Touring Machine was aiming for providing a campus tour in Columbia University. The model

contains a headworn see-through display, which overlays textual labels for different buildings in the campus. The display is driven by a computer worn as a backpack. The user would also hold a handheld display and a stylus for interaction. The model was mobile yet bulky. Due to the computing power required and the technological limitation, the whole system weighed just about 40 pounds. It also faces problems such as insufficient quality of display and tracking. (Feiner et al, 1997) Coors and Jasnoch later used a smaller wearable computer to develop a wearable GIS that accompanies tourists in Heidelberg, a city in Germany with rich historical features. The smaller computer provides users higher mobility which allows them to navigate outdoors more easily. (1998)

Figure 6 The Touring Machine (Feiner et al, 1997)

Figure 7 Wearable GIS (Coors and Jasnoch, 1998)

Scientists continue the development of mobile AR systems. Efforts and experiments have been put into creating the devices in the form of helmets, headsets, or glasses. Devices dedicated to AR emerged, such as Google Glass, Microsoft Hololens, and Magic Leap. These devices fueled

the further development of mobile AR technology. (Qiao et al, 2019) In the late 2000s and early 2010s, the introduction of high-performance smartphones provides an opportunity for AR systems to be built and run on these handheld mobile devices. The computing and graphic processing power, built-in camera and inertial sensor allow the AR system to prevail. (Yuan, 2019) In 2016, over 2.5 billion of the population around the world are smartphone users and the number keeps on increasing. (O'Dea, 2020) The popularization of smartphone usage allows mobile AR technologies to flourish commercially in recent years, applied in a wider range of fields including entertainment, advertisement, education, navigation, maintenance, etc. One popular example of the commercial success of AR applications is Pokémon GO. Pokémon GO is a location based AR capture game built for a popular Japanese intellectual property that has a fanbase all over the world. The game was released worldwide in mid-2016 and recorded 755 million downloads within the first year. (Bhasin, 2017) The phenomenal financial success of Pokémon Go attracts investors and capital. The AR and VR market is projected to reach 18.8 billion USD in 2020 and will continue to grow. (Bloomberg, 2020) The influx of investment allows the technology to continue to develop.

As smartphones are now the most popular platform for running mobile AR systems, scientists continue the development of improving the technology. Qiao et al suggested that Web-based AR could be the “promising future of AR”. (2019) While current AR applications can run on high-performance smartphones, they face the challenge that the flexibility is limited. The performance of the AR system depends on each individual devices’ hardware specification. For example, Google’s ARCore SDK is only supported by a limited number of devices. Older devices or devices with lower computation capability are not able to run the applications developed with ARCore. The variety of operating systems (eg. Android, iOS, Linux) also poses an increase in development cost when cross-platform support is needed. Seeing current limitations and also the opportunities of faster internet networks in the future, the web-based AR system is proposed.

The WebAR technology can be categorized into two approaches: the self-contained method and the computation outsourcing method. The self-contained WebAR system utilizes either a JavaScript Library or an extension of a web browser kernel. With more standardization efforts, these methods could be a solution for the current cross-platform interoperability problem and also allows a cross user AR experience. The computation outsourcing method is where the AR computation is done in a cloud server. With the fifth generation (5G) networks’ infrastructure starting to take over the coverage, a 1GB per second internet speed allows the AR system to operate with real time interaction and less delay. The computation outsourcing method tackles

the problem of devices' disparity. (Qiao et al, 2019) Sukhmani et al's survey suggested that the edge caching technology will further boost the performance of WebAR. (2019) The temporary storage of a selection of content in a server closer to the users can further increase the computation speed and reduce the latency. While the WebAR system is promising under circumstances, it still faces several challenges at the moment such as rendering and energy efficiency, privacy and security problems, and interoperability.

2.2 Mobile AR Apps development

2.2.1 Development Kits for Mobile AR Apps

Apart from the emergence and optimization of hardware devices, the advancement of powerful development kits also fueled further research and development of mobile AR. (Qiao et al, 2019) In the 1990s, mobile AR systems were developed with a combination of available software and extensive coding. Using the "Touring Machine" mentioned in the previous section as an example. The application of the system was implemented in approximately 3600 lines of commented Repo code. The software was running on a Windows NT/95 PC. The graphic package was a modified version of Obliq 3D, a display-list based 3D graphics package. (Feiner et al, 1997)

ARtoolkit was first presented by Kato and Billinghurst in 1999. ARtoolkit started as an open source software library for AR application development written in C and C++ language. The tool kit tackled the difficulties of establishing orientation and position of the user's camera at the time. (Lamb, 2003) The software library allows developers to easier create applications with marker based video AR tracking and real time 3D visualization, powered by the algorithms developed by Kato and Billinghurst. (1999) ARtoolkit continued to develop as an open source software library with several other contributors. In 2003, ARtoolkit supported multi-platform development (PC Windows, Mac OS X, PC Linux, etc.), contained an extensive collection of supported markers, and supported tracking with simple black squares. ARtoolkit has stemmed into a commercial licensing business since 2001. The company ARToolworks now provides toolkits for desktop AR development as well as toolkits for Android and iOS, which supports image based tracking on top of the original marker based tracking method. (ARToolworks, 2021) As the pioneer of AR development tool kits, ARToolkits is commonly used nowadays.

Wikitude is another currently available SDK for mobile AR. Wikitude was launched in 2008 initially as an Android mobile application for world browsing with real time AR camera view

overlaid with information retrieved from Wikipedia. (Arth et al, 2015) The company later switched focus and launched the Wikitude SDK for multiplatform AR development supporting Android and iOS devices and more. The Wikitude SDK is a paid license service, providing functionalities such as image tracking with a capacity of up to 1000 trackable images in the reference library, object tracking which tracks 3D objects such as toys and furniture, geo-location based tracking with GPS, and cloud database service. With their marketed customizable enterprise licenses and support team, Wikitude SDK is popular amongst the commercial sector with companies such as BMW and Disney on the customer list. (Wikitude, 2021)

Vuforia AR SDK was launched in 2015. Being also a paid license service, Vuforia AR SDK also offers image tracking and 3D model tracking with CAD models as input for app development. On top of that, it also offers an object tracking system, where the developers can scan a small object in real life such as a toy with Vuforia's Android Object Scanner app. The scanned object will turn into an Object Data (.od) file and can be used as a reference input. Vuforia provides a cloud server for each customer to store their reference database online. The SDK is optimized for a wide range of devices when only a vision based tracking is used, even for several different AR headsets and glasses and some older smartphone models. (Vuforia, 2021)

As smartphones and tablets are becoming the most popular devices for the implementation of mobile AR applications, companies have developed their own AR SDK designed and optimized for their own mobile devices' operating systems. Google launched ARCore in 2017. As a follow-up development to their project Tango in 2014, the new ARCore SDK does not require any additional hardware. With their mission statement "to bring AR to everyone", ARCore is free to use. (Burke, 2017) Currently, ARCore supports features such as image tracking, cloud anchor that allows users to place objects through a cloud server and visualize across different devices, face recognition, plane tracking for detection flat surfaces such as the floor or a tabletop, and immersive visualization services such as the Depth API and Lighting Estimation. (Google ARCore, 2021)

Apple on the other hand launched ARKit in 2017. Since the company has a more focused lineup of mobile devices, the ARKit SDK is able to suit the functionality of their iPhones and iPads more perfectly. The SDK has a yearly update to optimize for the new devices they put on the market. As per 2021, on top of the similar service as ARCore such as image tracking and face recognition, ARKit 4 contains functionalities such as 3D scanning with the LiDAR scanner

installed in the iPhone 12 Pro, iPhone 12 Pro Max, and iPad Pro models, and Location Anchor that provides location based AR built on their map service. (Apple, 2021)

The SDKs and toolkit mentioned above are all available to work in Unity. Unity is a multi-platform game engine that extends into the fields of industry, cinema and architecture. Its ability that allows developers and artists to create immersive experiences makes it popular with a wide range of applications. (Unity Technologies, 2020) Using AR SKDs in Unity allows developers with less coding knowledge to develop a robust mobile AR application for multiple and simultaneously create a user interface and other functionality for the app visually. To further streamline the process of mobile AR app development, Unity has launched the ARFoundation SDK. ARFoundation wrapped the common functionalities of ARCore, ARKit, Magic Leap, and Windows AR Plugin into one and allows developers to create applications for all these platforms and operating systems in one go.

SDKs	2D Trackings	3D Trackings	Location Based AR	Mobile Platforms	Cost
ARToolkit	Marker Based, Image Based	-	No	Android, iOS	Open Source/ Commercial license
Wikitude	Image Based	3D Object Based, Plane and Surface Detection	Yes	Android, iOS, Windows, Epson Moverio, HoloLens, Vuzix	~3600 USD / yr up for updating SDK
Vuforia	Marker Based, Image Based	3D Object Based, Self-scanned Model, Area Target, Plane Detection	No	Android, iOS, Windows, Lumin OS	504 USD/ yr up for Basic Plan
ARCore	Image Based	Plane Detection, Face Detection	No	Android, iOS	Free
ARKit	Image Based	3D Object Based, Plane Detection, Face Detection, Depth API with LiDAR	Yes	iOS	Free

Table 1 Comparison of common AR SDKs

The table above is a comparison of several common AR SDKs mentioned. ARToolkit, ARCore and ARKit are the options that are free to use or a free option is provided. ARToolkit however has a smaller range of functions provided. The commercial options support more platforms

including AR glasses and headsets. They also provide a customization plan for their customer to pick the services they needed to be included in the subscription.

2.2.2 Mobile AR App Development Guideline

The SDKs are convenient tools to help the development of mobile AR apps. However, developing a proper application requires not only the suitable tool but also a proper design to well suit the designated users' situation. Yuen proposed a design guideline for mobile AR app for building reconstruction. (2019) The guideline could be summarized in 6 aspects:

1) Selecting a suitable tracking method;
2) Having quality 3D models;
3) Deciding on the level of interactivity;
4) Storytelling;
5) Providing user guide and feedback;
6) Preventing cognitive overload.

Although the guideline was designed specifically for building reconstruction and conservation purposes, ideas from that could be taken and apply to a broader field of mobile AR application. Selecting a suitable tracking method should always be important in any AR development. A well selected and implemented tracking method can reduce the latency and increase the accuracy of the augmentation, which helps achieve the ultimate goal of an augmented reality system: to provide an immersive sensory experience beyond reality. On the contrary, delay and inaccuracy in tracking will cause positional and temporal misplacements of augmentations, withdrawing the users from the world building. More about tracking methods for outdoor mobile AR systems will be discussed in the next section.

"Having quality 3D models" is referring to the reconstructed building models in this case for building reconstruction and conservation. To visualize the historical outlook of a damaged or renovated building, a detailed computer generated building model has to be prepared. A better built model will have a better representation of the original building when added to the rendered AR experience. Details like the facade's texture and detailed artifacts can also help users to gain a more impactful understanding of the old building. To apply this concept to a broader implication possibility, the rule can be modified into "having quality augmentation objects". Different applications of mobile AR are aiming for a different visualization outcome. And the outcomes are not limited to visualizing 3D models. Videos, animations, analyzed data,

information, and 2D images are all possible objects to be added in an AR experience. The quality of these objects is crucial to the AR experience as they are the things to be added on top of reality.

“Deciding on the level of interactivity” and “Storytelling” could be generalized as the higher conceptual design idea of the AR app. This process is better to be sorted out and done before the start of the development phase, deciding the purpose of the app. Decisions should be made to decide what kind of AR experience that the app will bring to the users. For example, in the field of urban planning, if the app is used for collaborative design, the interactivity should be high. If the app is aiming for showing the real world with a specified visualization style, how to present the visualization will then become the main emphasis. “Preventing cognitive overload” also presents a similar concept of having a careful conceptual design for the app. If the AR experience from the app is well defined and executed in the development, the app should work according to the concept and not overload the users with information.

“Providing user guide and feedback” can be separated into two different concepts. Providing a user guide is helpful for improving the usability of the app. Users should be able to operate the app and gain the desired experience from the app with information inside the app without any additional help. This can be achieved by designing a good user interface and providing clear instruction within the app. On the other hand, “user feedback” can be one important element in the evaluation process. Evaluating the developed app is useful for maintenance and helps the updating development of the app.

Overall, this guideline can be summarized into three major stages for AR app development: conceptual design and planning, technical implementation, and evaluation. These stages can be applied to a vast majority of mobile AR application development projects. The detailed rules on the other hand can need some modification in order to apply for different development processes from case to case.

2.3 Outdoor AR Tracking

As mentioned in the previous section, AR is a combination of virtual content and the real world. To create an immersive experience with the surrounding environment and proper interactions, the AR system and device need to understand its position and relationship between itself and the real world. To acquire its registration in the real world, a tracking system is required. A tracking system uses the sensors or cameras in the device to look for specific patterns or points of reference. By matching those with the stored reference in the AR system, it registers itself a position and is able to find out the relationship between the user and the world. (Kaiser, 2019)

If the mobile AR system is to be used in an outdoor urban setting, implementing adequate outdoor tracking is important. In general, tracking approaches could be divided into two general categories: marker-based and markerless tracking. The former utilizes fiducial marks and the latter uses natural features on the object. (Blanco-Pons et al, 2019) For outdoor tracking, marker-based tracking is usually harder to implement as the fiducial marks cause possible damages to the building. In earlier development of outdoor mobile AR systems, GPS, magnetic compass and inertial sensors are heavily relied on for position measurement and orientation. (Reitmayr and Drummond, 2006) As the first mobile AR system (mentioned in section 2.1), the Touring Machine carries a Trimble DSM GPS receiver. The tour application continuously takes the GPS reading as input. Based on the GPS input and the database of buildings' information, the computer generates graphics to overlay on the real world. (Feiner et al, 1997)

The GPS tracking systems are still applicable in more recent outdoor AR systems and could be seen in urban planning related implementation. Cirulis and Brigmanis (2013) suggested using AR technology to enhance the "immersion level of urban planning solution". The City 3D-AR project uses a system that allows users to place 3D building models in the real world environment using GPS longitude and latitude coordinates. By matching the data gathered from the GPS and orientation sensors in the mobile device to the coordinates configured in the user's model, the application allows OBJ or OFF files to be placed and rendered in real space. Using GPS tracking is efficient and lightweight in the urban planning field, as usually there are more than one building or urban feature that needs to be tracked. The system requires only the coordinates and orientation in the database to match with the GPS readings. This technique however is facing difficulties in accuracy in urban settings. Although civilian GPS can have satisfactory performance in open space, the accuracy reduces in an urban setting, especially in a dense urban environment where signals are shaded by buildings and the GPS sensors might not fully perform. (Reitmayr and Drummond, 2006)

Outdoor AR tracking is also possible without a GPS sensor in the device. Behringer, Park, and Sundareswaran (2002) developed a model based approach to outdoor tracking. The system uses 3D CAD models as the reference database. With a camera attached to the headset, the AR tracking system captures 2D images and performs a gradient scan in real time to extract edges. The system then matches the extracted edges and matches them with the 3D CAD model in the database. This tracking method has high accuracy. However, it relies heavily on the precision and the level of detail of the 3D model. While Reitmayr and Drummond (2006) had a similar model-based approach to their outdoor AR tracking, they further incorporated the GPS and inertial sensors to their implementation for a more robust system. They suggested that the GPS readings could act as an index to limit the search range inside the database, further increasing the efficiency of the tracking system. Comparatively, the model-based and hybrid tracking system has higher accuracy than the GPS tracking systems. The possible precision of these tracking methods can be up to one pixel. The actual error is however up to the camera lens. (Behringer, Park, and Sundareswaran, 2002)

Figure 8 Computing processes of model based tracking(Behringer, Park, and Sundareswaran, 2002)

Model-based and hybrid tracking methods provide a high accuracy. However, they require large memory space for the database as highly detailed 3D models are crucial for effective tracking. Image tracking on the other hand is a lighter weight option as only raster images are needed to build the reference library. Image tracking is a markerless tracking method often used in fields like advertisement, tourism and education, etc. An image tracking system detects points of interest in the reference images and matches them with the real time input from the camera of the device. (Google ARCore, 2019) Image based tracking is optimized for tracking 2D images and therefore not optimised for tracking buildings or urban features. Blanco-Pons et al however developed a system to apply a lightweight image-based outdoor tracking for buildings. (2019) An application was developed for visualizing the historical outlook of the buildings in Parliament Hill, Ottawa. The project introduces the multi-image system, where multiple images

taken at the same location at different time points are used to tackle the problem of the changing environment, weather and lighting at different times of the day. An analysis of possible environmental changes was carried out to limit the number of images used for one spot. This approach is comparatively lightweight and could be applied to different situations.

Artificial intelligence (AI) has had a prolific development and gained interest in recent years. The technology can be used for object detection and target tracking when applied to certain devices such as Unmanned Aerial Vehicles. (Rohan, Rabah and Kim, 2019) Sahu, Young and Rai suggested that "AI is a promising solution to increase the adaptability and versatility of AR systems." (2020) The AI subset Deep Learning (DL) technique has shown a better performance in the last decade in terms of object detection compared to decades of development in classic AR detection methods. A DL based tracking system can perform well in tracking natural features despite the change of environment. The technique can delimit the AR system from controlled objects and environment. AI can also reduce the latency of an AR system with its possibility of accomplishing detecting, tracking, and other tasks simultaneously. Google, as one of the major driving forces of AR development in the commercial sector, has released the AR navigation function within Google Map in 2019. Having the advantage of possessing a large amount of data collected from their street view service, Google is able to utilize the data with the help of deep learning technology to provide accurate real time outdoor tracking for AR navigation. Applying AI in the AR system is still in its beginning phase of development. But with the promising opportunity and development pace, AI assisted AR systems can be more widely adopted and matured in the foreseeable future. (Sahu, Young and Rai, 2020)

2.4 Application of AR Visualization Methods in Urban Planning

Approaches to urban planning have been continuously evolving. The methods keep on changing to tackle modern urban problems such as the rapid pace of urbanization globally and the need for sustainability. One of the most mentioned approaches in the field is called the bottom-up approach. Although there a clear guideline is not defined and followed universally, the bottom-up approach in urban planning generally means that the planning process is initiated from the general citizens starting from the individuals and community and is revolving around their opinion, idea, and needs. With the involvement of the local individuals and communities that are not urban planners or designers, the planning process can potentially identify the actual social, environmental and economic problems in quality details rather than generating analytics only based on quantitative data. With the insights from the users that are actually living and participating in the urban space, they can help fill the gaps between the perception of a handful

of professionals and the actual problems that exist in the area. Solutions generated from the bottom-up approach are locally grounded and therefore can address the local problem more properly. (Dias et al, 2017) Augmented reality can be a very helpful tool to achieve a bottom-up urban design as it provides interactivity and improved sensory experiences that can bring the general public into the planning process. Different AR visualization techniques can be applied to different stages in the planning process, including the gathering and presenting of the site's information, designing collaboratively, and presenting the design in an enhanced reality, etc.

To increase the participation of the general public in the urban planning process to achieve a bottom-up design, the transparency of data and information is important. Akahoshi et al developed an AR prototype called the "AR Consensus-formation system" to visualize geospatial data gathered by the Japanese government's i-UR project. The project gathered data such as transportation infrastructure, population density, and urban sprawl trends and visualized the data with the incorporation of the CityGML data. The "AR Consensus-formation system" is developed to visualize the data through Hololens at the same time for multiple users. The visualization is controlled by a tablet and allows a 1:10000 map with overlaid 3D data such as a population density grid to project on a 3x3meter surface. The visualization can be modified in real time and allows a large group of people to browse and experience the shared contents. (2020) This AR visualization method is helpful at the beginning stage of social engagement in urban planning. The sharing of information in an easy to understand format with customization which allows contribution from the group can empower the community and encourage them to contribute their insight.

Figure 9 The AR Consensus-formation system of the i-UR project (Akahoshi et al, 2020)

Apart from the indoor visualization with AR system, outdoor mobile AR visualization is also applied for the early phases of the urban planning process. Bednarczyk and Templin had developed an AR app that visualizes geospatial data on top of buildings' facades. (2020) The

AR app uses the GPS/ GNSS tracking method combined with the image tracking method and visualizes information of buildings such as construction elements, defects, and historical facades. The app works in client-server architecture and calls the web based app to the mobile device through an HTTP request. The infrastructure of this project is robust but the visualization of it is relatively simple. The visualization is mainly text and line based data. The web based system however allows the app to link to websites that contain more information about the buildings on site. This app presents data that are more professional oriented, but the applied technology shows the opportunity of visualizing data outdoors on designated buildings. If the visualization style for this system is designed for a wider target audience, it could invite the general public to discover and navigate the urban space and share information similar to the i-UR project.

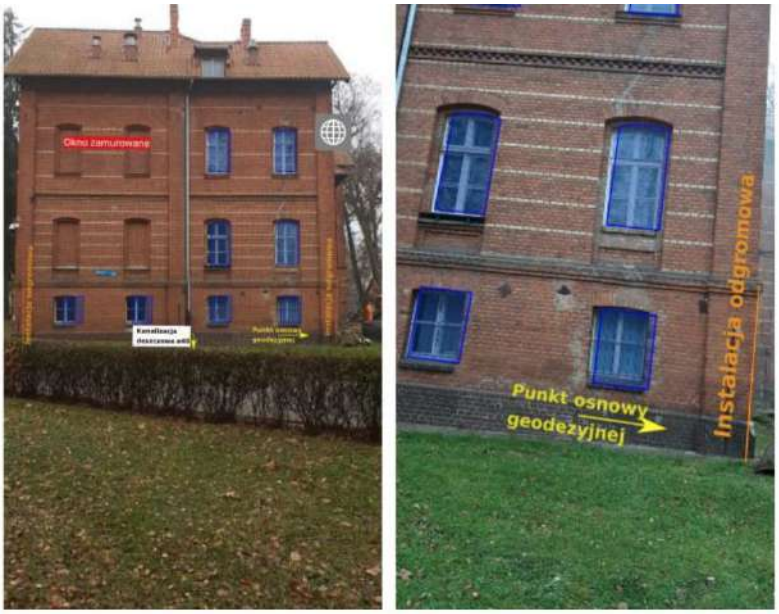

Figure 10 Visualization of information on buildings facade with AR (Bednarczyk and Templin, 2020)

AR visualization can also be applied to the later stage of urban planning and design than the initial phase of gathering, analyzing, and presenting the information of the site to encourage public involvement. Imottesjo et al presented the prototype of Urban CoBuilder in 2020. The prototype allows several users to contribute to an urban design piece by adding design elements on the site through the AR system. The project utilized the marker based tracking method in an outdoor setting. Being site specific allows the markers to be placed on the position that a design element could be added. Different types of design elements such as buildings, open spaces, and trees are stored in the system as 3D models. The users can add the design elements real time on a grid based system on site at wish and apply a different texture to the models. They can then

switch to a bird eye view with a fully rendered 3D CG scene to spectate the design from a different point of view. This prototype creates a game like system that can be easily used by anyone who has minimum design knowledge to present and express their idea on improving the urban space, which can further improve the public involvement rate throughout the bottom-up urban planning process.

Figure 11 Adding design element on site through the Urban CoBuilder AR app

The reviewed AR visualization app for urban planning shows a range of possibilities for the technologies to be applied to practical urban planning processes to encourage involvement from individuals and communities. The AR app to be developed in this thesis will aim to visualize the buildings as a focal point in the urban context. The enhanced visual perception of the buildings through AR can help urban planners present their idea more efficiently and allow the general public to understand the context more easily. The visualization can be used for the early stage of the urban planning process to provide context to the project.

3 Concepts and Methodology

3.1 Design Guideline

The application to be developed will be using outdoor mobile AR technologies to visualize buildings and context with the grayscale masking filter. The app will run on Android mobile devices. The devices should act like a viewport for the user. When the app is on, users are able to observe the real world through the camera of the device. The augmentation comes when the user is looking at one of the designated buildings in the area of Stuttgart Nordbahnhof. The AR system should recognize the building, and instead of overlaying information or graphics on the building, it turns the surrounding context and background into monochrome, leaving the designated building in color. With this visualization, the building is highlighted. It provides context to the building but also reduces the amount of information feeding to the users' perception. Users should be able to tell from the information in the app where the designated buildings are and where to find them on site.

Figure 12 Demonstration of the desired visualization method for this AR app

Modified from the AR app design guideline by Yuen mentioned in section 2.2, the development of this application will follow the following considerations:

1) Choosing and implementing a suitable tracking method
2) Gathering and creating suitable models for visualization
3) Defining and executing the visualization
4) Clarity and usability for users
5) Evaluation

The guideline is modified to better suit the nature of this AR app, which emphasizes more heavily on the visualization method than the other aspects. "Choosing and implementing a suitable tracking method" is important for almost every AR project and is directly transplanted from the original guideline. A high accuracy and low latency tracking method can guarantee the

users an enhanced sensory experience. Having quality digital models is important for AR visualization as it provides the foundation for the rendering. The models that will be used in this app will be overlaid on the building in the real world and attempt to match the outline of them.

"Defining and executing the visualization" is conceptually combining the "storytelling" and "interactivity" aspects in Yuen's original guideline. This AR app aims to apply and showcase the desired visualization style for urban planners, which does not require a lot of interactivity and has a set "story" to tell once the visualization style is defined. The visualization style for this app is already defined above. More concepts about executing the visualization style will be discussed in the later sections.

"Clarity and usability for users" is modified from the "providing user guide" rule and will be achieved by designing a suitable user interface and experience for the app. Users should be able to learn and use the app easily within the application without any additional help. The user interface should be kept minimalistic for minimizing the interruption to the visualization. Other functions can be added to cater to the practical needs of urban planners.

"Evaluation" is important for the maintenance of an app and can provide insights for future development. Further discussion about the evaluation criteria of this particular app will also be discussed in the later sections.

3.2 Project Setup

The application will be built on Unity with the ARFoundation SDK and ARCore extension. As reviewed in section 2.2, Unity is a multi-platform game engine that extends into the fields of industry, cinema and architecture. Its ability that allows developers and artists to create immersive experiences makes it popular with a wide range of applications. (Unity Technologies, 2020) While the game engine has endless potential for rendering and visualization, developing with Unity does not require as much programming and coding knowledge as other developing platforms such as using Android NDK. Unity is free to use, which can make this methodology applicable in more different circumstances. Moreover, Unity allows developers to develop and publish mobile applications in one sitting. Apart from creating the AR system, other functions and elements of the app can also be done in the same engine.

Unity supports multiple common AR SDKs. This mobile AR application will be built with the ARFoundation SDK and its ARCore extension. ARCore provides several 2D and 3D tracking services and is optimized for a list of Android mobile devices. ARCore is also free to use, which again benefits the idea of giving this development method a wider implementation possibility. Being wrapped inside ARFoundation, using this SDK creates the opportunity for cross platform publishing in the future.

As a game engine, the setup of a Unity project starts with scenes. Each scene in a project could be understood as a stage in a game. A default scene started with the main camera, which directs and controls the field of view of the player, and a light source called Directional Light. Developers can then add any objects, obstacles, decoration and command on top of the scene and start building the world in the game. (Unity Technologies, 2020) As a mobile AR application, the main scene of the app will naturally be the AR scene. The most basic setup of an AR scene with the ARFoundation SDK will consist of an ARSession and an ARSessionOrigin. ARSession controls the lifecycle of the AR system. When an ARSession starts, the device will start tracking the surrounding environment. ARSessionOrigin on the other hand transforms the trackables into information such as position and orientation. For example, when a plane surface is detected in the scene, ARSessionOrigin transforms the points and vortices on the plane into coordinates and rotation degrees. The ARSessionOrigin is also the parent of the ARCamera. The ARCamera represents the camera of the users' device. In an AR scene, the ARCamera should be set as the "main camera". (Unity, 2021) The image below shows a basic set up of an AR scene in Unity. The origin point represents the position of the ARCamera. The projected white wireframe is the field of view of the camera. The blue and gray is the Skybox in Unity. Once the app is built and running on an Android device, it will be covered by the rendered recording from the active camera.

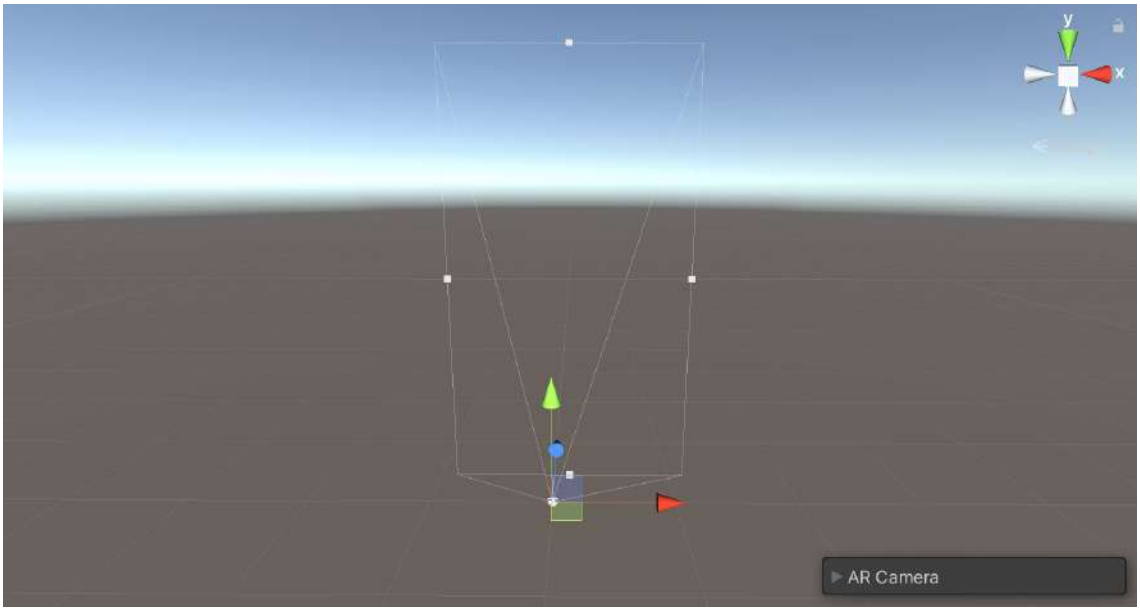

Figure 13 Basic setup of an AR scene in Unity

With the main AR scene, ARSession and ARSessionOrigin set up, the project is ready for mobile AR app development. Command and configurations for AR tracking can be added and

set under ARSessionOrigin. Objects that need to be rendered can be put inside the scene. Unity is able to build and run the app on a plugged in Android device and testing could be done upon that.

3.3 Image Tracking System

To achieve the visualization for this app, the tracking method has to be working in an outdoor environment with densely built buildings, as the site is in Stuttgart Nordbahnhof, where high rise office buildings and several residential buildings are located. Since the app is designed to cater to urban planners, and they often work on a district wide scale instead of a single building, the tracking system will have to be able to track on multiple buildings. The accuracy of tracking needs to be relatively high to provide a more effective visualization. Low accuracy in tracking might result in gaps and shifting in the placement of the masking filter.

On the other hand, the tracking method for this app also has to be kept lightweight. Since the system has to recognize multiple buildings, there should not be too much reference data for the AR system to run through. Run time for tracking each building should be kept Otherwise, latency is likely to occur when the system has to run through more than 10 buildings in real time.

A series of available outdoor tracking methods have been reviewed in section 2.3. In most cases, the developed methods aim to very precisely track a building. However, the reviewed technologies often are just implementing the tracking system on one designated building. For example, the mentioned hybrid method that combined GPS position and 3D model provides a robust and accurate outdoor tracking for buildings. However, it requires detailed 3D building models as input reference and can take up a lot of memory space. The complicated computation method can also possibly slow down the system and impose latency when there are multiple targets that need to be tracked. A tracking method that requires a smaller data input will be more ideal for this particular AR app.

While it is not optimized for outdoor tracking with buildings, Blanco-Pons et al's project prove that it is possible to implement an image based tracking system for tracking buildings. Unlike some other moving urban features such as vehicles and trains, buildings sit still with a relatively

static surrounding context. Also, from a pedestrian's point of view, when holding a camera, they are only able to observe the whole silhouette of a building from a certain standpoint and direction. These criteria create the opportunity to treat buildings and their contexts as "2D images" when they are framed in a certain way through a camera. Making it therefore possible to implement an image based tracking system to track buildings.

The ARFoundation SDK and ARCore extension come with the "Augmented Image" function, which makes it easier to implement an image based tracking system in the AR app. Image based tracking requires only raster images as input. Once implemented, the system will run through the images and extract points of interest from them. Once the tracking session starts, the application will compare and match the points of interest in the reference images and the real time input from the AR camera. If the system recognizes that the user is looking at an "image" that appears also in the reference database, it will register the orientation and location of the user and the tracked object. Since only raster images are needed as input and only the extracted features are stored in the database, each reference image will take up only 6kb of memory. The maturity and relative simplicity of this tracking method provides it with adequate accuracy and allows fast computing with less possibility of latency.

3.4 Reference Image Library

To implement an image based tracking system with ARFoundation, a reference image library needs to be built. The developers' guide for ARCore has provided requirements and suggestions on how to properly gather reference images and build the library. Reference images are converted into grayscale in the system and features in the image will be extracted. Therefore, the color information of the image is not used although color images can be used. The images should be at least 300x300 pixels in resolution, but using images with high resolution does not improve performance. The images have to be formatted as JPEG or PNG. Compression on the image has to be avoided because it will affect the feature extraction process. Images with too many or too few geometrical features should also be avoided as they can result in poor detection and tracking performance.

In order for the image tracking to work, the image must follow the following requirements:

- *It must fill at least 25% of the camera frame to be initially detected.*
- *It must be flat (for example, not wrinkled or wrapped around a bottle).*
- *It must be in clear view of the camera. They should not be partially obscured, viewed at a highly oblique angle, or viewed when the camera is moving too fast due to motion blur. (ARCore developers' guide, 2021)*

If the image tracking system is used for tracking smaller objects, such as logos printed on a product, the common practice is to hint the user to move the object closer to the camera, until it fills at least 25% of the camera. In this case, the app will be used in an outdoor setting with buildings that cannot be moved around. The application should guide the user to walk up to a spot where the target object is framed correctly in the camera.

The image must be flat without being wrinkled or curved is mentioned in the developers' guide. This rule is applied to, for example, printed images in the real world. If the images are printed on paper or some other soft materials, wrinkling or bending of the material and the image will obscure the detection process. In this case for outdoor tracking, the target objects are buildings being treated as a "framed image/ scene" instead of actual printed images. The scenery can not be wrinkled or curved. Therefore, this rule should not be a worry in this application.

To assure the object to be in clear view of the camera, several considerations must be taken during the reference image collection process. When capturing the reference image for the buildings, make sure the buildings are framed properly in the photo. The photos should be taken in an accessible spot for pedestrians with an angle that can be comfortably reached. Avoid tilting the camera too much to avoid a highly oblique angle. Since this app is designed for users to walk around the district and visualize the buildings, the camera moving too fast or motion blur should not be a problem. The developers' guide also suggests that when tracking large objects, the dimension of the image in the real world should be specified to improve tracking and detection performance.

In order to create an adequate reference image library and implement a functioning image tracking system for this app, the following steps will be followed in the process of capturing reference images:

1. Set up a route for the users

   The app requires the users to walk around the district to find the designated buildings for visualization. If the designated buildings are scattered all around the place, it will impose difficulties for the user to find them. Selecting a route that allows the users to observe multiple buildings with a clear view can help narrow down the search, provide better tracking, and improve the overall AR experience.

2. Select the observation points

   Once the route is defined, observation points have to be picked. On each observation point, a building should be in a clear view on the level of the user's camera or eye level. The outline of the building should be seen without looking up too much to avoid excessive tilting of the camera and overly oblique angle.

3. Capture the images in day time with neutral weather and lighting

   Lighting and weather plays a big role in capturing a high quality reference image. Lighting should be mild but adequate when capturing the image. Insufficient lighting could reduce the clarity of the features on the building. Strong lighting or shadow on the other hand will induce over or under exposure and obscure the feature detection. The weather should also be neutral when capturing the reference image. Some weather conditions will create unnecessary features in the image. For example, snowing will add features to the roof and the ground. Cumulus clouds will also give the sky extra features that might not be seen in the next visit.

   Images will be captured by an Android smartphone as it provides convenience and adequate resolution. By doing so, the viewing angle of the users can also be tested while capturing the reference image.

4. Import the images and estimate the dimensions

   After the images are captured, they are imported to the development platform. Dimensions have to be specified according to the size of the object in the real world to

improve the tracking performance. The height of the building can be estimated from reference data, 3D models, or from the numbers of stories the buildings have.

With these steps, the reference image for this app should be optimal for the tracking system. The features in the images are at an adequate level of complexity. The angles of observation are not obscure. The users should be able to find the viewing point easily and the system can perform the tracking efficiently.

3.5 Gathering and creating suitable models for visualization

A set of CityGML data is available for the Stuttgart Nordbahnhof area. The dataset contains 3D building models that are at the Level of Detail 2 (LOD2). LOD2 buildings model is defined by the CityGML specification as differentiated roof structure and differentiated boundary surfaces. (Gröger et al., 2012) The CityGML data can be extracted with FME and exported into individual building models that are on the same scale as the actual buildings. The geometry of the LOD2 models can at least partially match the outline of the building in the real world and be used as the masking object to be placed in the AR app. Since the 3D MOD2 models are readily available and only file conversion and exporting are needed, using CityGML can potentially be an efficient option to be used to build the AR app with the desired visualization.

Figure 14 Building models from LOD0 to LOD4 (OGC, 2012)

A LOD2 3D building model might however not contain a sufficient amount of detail when a very accurate mask is wanted for the visualization. LOD2 models do not contain detailed artifacts such as chimneys and antennas. When they are being placed to mask the desired building in the real world, those artifacts will be excluded in the visualization. Another difficulty of using the LOD2 models would be the positional placement of them. Image based tracking is a 2D system. The matching of 2D images with tilted perspective and 3D models can be challenging.

LOD2 3D building model will be used and tested in the development process of the app. If these models are not sufficient for effective visualization, alternative methods will be sought. One possible alternative will be manually creating the surface that contains the buildings in a CAD software according to the reference image. The concept behind this method is that as the image based tracking system is used, the buildings will be recognized by the AR system only from a certain point of view from the users. From that point of view, the static building's outline can be found. That outline can be used to create a polygon that overlays the building perfectly in the "framed scene" from the AR camera. That can lead to more precise masking of the building and execute the visualization more effectively.

3.6 Authoring Shaders and their application for visualization

To create and apply the desired visualization style to the AR application, several shaders are created. According to Unity's official documentation, shaders are "small scripts that contain the mathematical calculations and algorithms for calculating the color of each pixel rendered, based on the lighting input and the Material configuration." (2021) In other words, shaders are the pieces of programs that render pixels on the screen of a device taking meshes, vertices, etc. and other configurations as inputs. (Hocking, 2020) The diagram below shows a brief description of the rendering workflow of Unity which helps further describe the role of a shader in the rendering process. Every object placed in unity requires an assigned material to be rendered. And every material contains a shader. The material acts as a wrapper for the shader and allows the values to be set for each property and perimeter in the shader. (Zucconi, 2015) Ultimately the shader communicates with the GPU and renders the object according to its geometry and the values set in the assigned material.

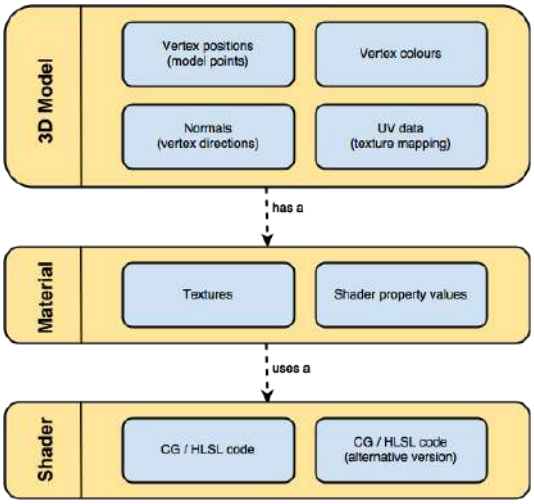

Figure 15 Diagram showing the relationship between shaders, materials, and 3D models in Unity (Zucconi, 2015)

Shaders in Unity are written in the High-Level Shading Language (HLSL) and wrapped in the ShaderLab syntax. (Unity, 2021) Every piece of shader program in Unity consists of two things, Properties and SubShader. In the Properties block, variables declared will then become the values and parameters to be tweaked and adjusted in the materials. Some common property types include 2D raster texture, Color (RGBA), Vector (XYZW) on top of Int, Float, and Range. The SubShader blocks are the actual commands that will be given to the GPU to perform the rendering. Multiple SubShader blocks could be found in a piece of shader program containing different arguments. (Zucconi, 2015)

Shaders allow customization on a large variety of qualities, including but not limited in: the color, saturation, brightness, contrast, hue, texture, and the interaction of the 3D model with light, creating blurring effect, bokeh, refraction and distortion, etc. Shaders are even capable of rendering animation. By imposing a shifting range of value in the script, the shader can be able to animate the 3D object with effects such as tideling water and illuminating light. The image below demonstrates the capability of a shader in Unity. With the shader created by Loghin (2019) applied, a 3D mesh is rendered with the texture of water. The "water surface" reflects the features above including the sky. The texture of foam could be added and customized. The underwater features are rendered through the transparency of the water with diffusion and refraction of light. And in gameplay, there are moving waves with customizable intensity and steepness.

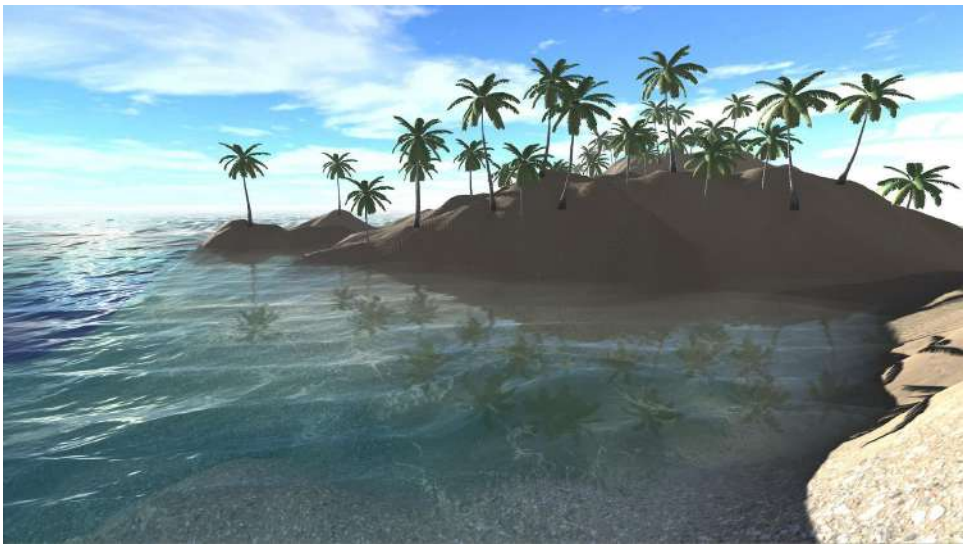

Figure 16 Demonstrating the Shader's Visualization Possibility (Loghin, 2019)

To cater to this complexity of developing a shader, Unity had created a visual shader editor in 2018 named Shader Graph. Shader Graph allows developers to use a node based shader network to develop shaders visually. Instead of writing codes, developers can connect nodes and assign values in a graph network. The editor visualizes the result in real-time. Shader Graph provides an opportunity for artists or other developers with less programming skills to be able to author shaders at ease. (Cooper, 2018) While Shader Graph is an adequate tool for authoring complex and sophisticated shaders, the visualization style to be achieved in this project is relatively simple. Creating a grayscale filter to visualize buildings and context in an urban environment should be able to be achieved by coding with image processing equations. The buildings which will remain in color should be able to achieve through programming logic. To avoid complications and showcase the logic behind the visualization method, the shader in this project will be created by coding with ShaderLab rather than being constructed in Shader Graph. The shaders should remain simple and streamlined.

Theoretically, the targeted visualization can be achieved with only 2 shaders: one for creating a grayscale filter, and one for masking the building and leaving them to be rendered in color. Each shader will be wrapped in a material. The material with the color rendering shader will be applied to the 3D building model or manually constructed building contour mentioned in the previous section. The material with the grayscale filter shader will need a separately built model that surrounds the AR camera and provides a filter for the scene all around the user.

3.7 Hardware

This project will be implemented and tested on an Android mobile device. The mobile device is important for an AR app as the computation, tracking and visualization are going to run on it. The specification on the device needs to support real time computation. The performance of Android devices is improving rapidly in recent years. However, despite the increasing number, not all Android mobile devices are supported by ARCore. The Google ARCore Developers Team provides a list of supported devices on their website. (2020) The device that would be used in this project for running and testing is:

Model	CPU Speed	CPU Type	GPU	Rear Camera - Resolution	RAM Size (GB)	ROM Size (GB)	External Memory Support	OS
 Samsung Galaxy A20S	1.8 GHz	Octa-core Cortex-A53	Adreno 506	13 MP + 8 MP + 5 MP	3	32	MicroSD (Up to 512GB)	Android 9 (Pie)

Table 2 Specs of Galaxy A20

This device has one of the lower specifications on the list of ARCore supported devices. If the developed AR app can run smoothly on this device, most of the other devices on the list theoretically can also support it.

3.8 Workflow

Summarizing the previous sections in this chapter, the development of the app will follow the following workflow with the concepts discussed applied:

1. **Capturing Reference Images and Building the Reference Image Library**

At the first phase of the development, reference images will be captured following the steps mentioned in section 3.4 for building the reference image library. A high quality reference image library can guarantee a better performance for the deployed image tracking system and improve the users' experience in the later stage. This process could require several site visit sessions for scouting and capturing. The reference image library can be updated according to the later development in the next steps.

2. **Implementing and Testing of the Outdoor Image Based Tracking System**

With the reference image library ready, the next step is to build the tracking system and put it to test. As discussed in section 3.3, this app will adopt the image based tracking method. The tracking system will be built with the Augmented Image API in ARFoundation within Unity. It has to be able to track multiple outdoor buildings and spawn different objects according to the

detected building. The system will be tested indoor for the basic logic and later outdoor on site to see its performance on tracking the actual buildings.

3. **Gathering, Formatting and Creating Suitable Models for Visualization**

The CityGML data will be extracted in FME for the 3D LOD2 building models. The building models will then be exported as .OBJ files as individual buildings. The models will be orientated and tested with the tracking system to see if they are adequate for the visualization in terms of the level of details and placement. If the quality of these models could not meet the desired result, new models will be built manually to create the best outcome for the visualization method.

4. **Developing the mask filter visualization method**

As mentioned in section 3.6, visualization styles will be achieved by authoring shaders in Unity. 2 different shaders are planned to be written. The first one is for achieving the grayscale filter applied to the AR camera. The second one is for the mask applied to the buildings to exclude them from the grayscale filter and let them be rendered in color. Each shader will be applied to suitable 3D models to achieve the visualization. The style of the filter should effectively convey the idea behind the visualization style, which is to focus on a selected building without overwhelming the viewer with surrounding information nor cutting out all the context. Testing will be done on site to actually see and feel the visualization in the real world.

5. **Improve the user experience of the application**

The users know how to use the app based on information within it. The app should provide information for the users on the location of the buildings and their viewing point in order to let the system track them and visualize them. The user interface should be clean and minimalistic to keep the visualization the main focus. Other functions can be added to the app to improve the usability of urban planners.

6. **Evaluation**

Evaluation is important for the improvement of an app and can provide insights for future development. The evaluation will be done for this app in terms of its development method and the final product. The criteria of evaluation will be discussed in the next section.

3.9 Evaluation Criteria

The evaluation of the app is going to be divided into three general categories:

1) Selection and performance of the SDK

   The ARFoundation SDK and the ARCore extension are selected from a wide range of available AR development toolkits. The evaluation section will review their performance in the app development process, pointing out the bugs and glitches found during the process, discuss the limitations of the documentation and help provided, and discuss the pros and cons of selecting the SDK for a mobile AR app development.

2) Selection and performance of the selected tracking method

   The multi-image based tracking method is chosen for this outdoor mobile AR app. Although the method is chosen due to its lightweightness and robustness, image tracking is not optimized for tracking bit outdoor 3D features such as buildings. The evaluation will review the challenge and limitations faced in the development process when the tracking system is implemented.

3) Comparison of the two building model types

   As mentioned in section 3.5, two different approaches will be done for gathering and creating the suitable building models for the visualization. The evaluation will compare the two approaches in terms of time-cost effectiveness, preciseness, and visualization effectiveness.

   The time of implementation of each model type will be recorded and compared. Calculation will be done to analyze how many pixels lay inside and outside the building as decided, as an account for the preciseness of using each type of the building model. Comments will be made for the visualization effectiveness based on the AR experience they can provide to the users.

4 Implementation

4.1 Implementing the Image Tracking System

With the concepts and guidelines explained in chapter 3, this section will focus on the implementation of the image based tracking system for tracking multiple buildings outdoors in the area of Stuttgart Nordbahnhof. The implementation is broken down into 3 phases:

1) Building the reference image library
2) Modifying the "Augmented Image" API in ARFoundation to track multiple targeted buildings
3) Testing and modification

4.1.1 Reference Image Library

The capture of the reference image and construction of the reference image library will follow the steps listed in section 3.4 above. The steps optimize the database for effective outdoor tracking and also provide a better users' AR experience when navigating with the app.

1. **Set up a route for the users**

The route selected for this AR app starts from Löwentorbrücke and ends at Presselstrasse. The route started on a lifted footbridge and slowly declined to ground level. It passes through a series of office high rise buildings in the block with a clear view. The route then turns to a street where some residential buildings can be found.

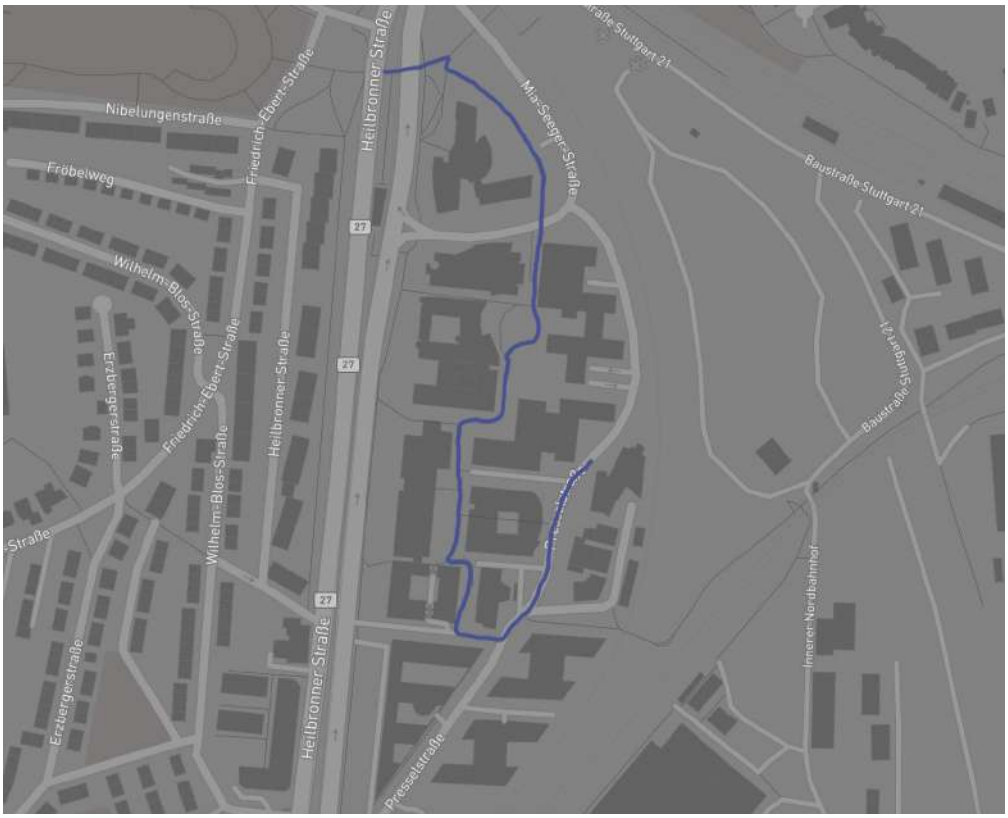

Figure 17 Map for the Selected Route for the AR app starting at Löwentorbrücke and ending at Presselstrasse

The route is picked for the following reasons:

- It provides clear views of a series of buildings on a pedestrian level. This is the most crucial criterion. Clear views of buildings allow the system to properly detect and track them from the users' point of view.
- The selection of buildings has varieties. It contains high rise office buildings with unique forms and also more standardized residential buildings. This provides the opportunity to test out the capacity of the tracking system.
- It lies on a relatively quiet walk path. The route that sets on a less busy street can increase the level of safety for the users, while they are partially focused on the mobile device for the AR experience.
- It mainly consists of a "pedestrian only" path. The absence of cars and other moving objects can reduce the disturbance and increase the accuracy of the tracking.

2. Select the observation points

Defined in section 3.4 the observation points should contain a clear view of a building through the camera of a smartphone. The point should not be too hard to reach to ensure the safety of the users. The viewing angle should not be too tilted. For example, it should not require the users to look with their heads up. Otherwise, the overly tilted angle will obscure the detection from the tracking system.

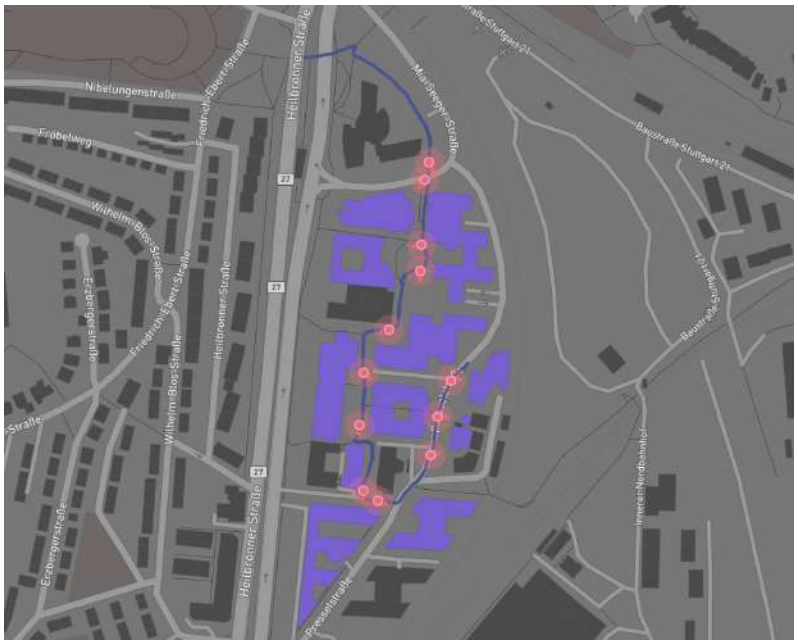

Figure 18 Map showing the selected observation points, view direction, and designated buildings.

The red circles in the image above show the selected observation points. The pink arrowhead represents the direction for viewing. The purple buildings in the maps show the targeted buildings that are going to be traced and visualized in this AR app. There are some buildings in the area that will not be included in the app. The reasons include:

- Too much obstruction such as plants or other buildings is sitting in front of the building.
- Some buildings are too tall to be seen whole without tilting the camera too much.
- The Locations of the buildings are too close to the route. Only part of the facade can be seen.

In total, 12 observation points are selected, meaning that there will be 12 designated buildings in the app that can be visualized with the grayscale masking filter.

3. Capture the images in the daytime with neutral weather and lighting

Once the observation points and designated buildings are selected, the next step will be to actually capture the images for the reference image library. Following the rules stated in section 3.4, the images need to be taken in a daytime with neutral weather and adequate lighting. Images are taken and selected at the 12 observation points.

Figure 19 Captured reference images for the reference image library

The images are taken on a cloudy afternoon. A cloudless sky could have been a better alternative, but it is rare in the winter of Stuttgart. In this case, the sky is covered by a layer of stratus cloud. The homogeneity of the cloud cover provides an acceptable result for the images.

No precipitation was recorded on the day of capturing the images. Each image is 3499 × 1968 in resolution with the 16:9 proportion.

Moving objects such as other pedestrians and vehicles are avoided when capturing the images. However, features such as tree branches and road signs are difficult to avoid. But since these features are mostly going to stay there when the users are using the app, they should not pose any serious problem to the tracking system.

4. Import the images and estimate the dimensions

If the ARFoundation SDK is properly installed and set up in the Unity project following the steps in section 3.1, a Reference Image Library prefab can be added in the project. The Reference Image Library prefab can store up to 1000 images and track up to 20 images at one time. After importing the captured images to Unity as assets, they can be added to the Reference Image Library prefab by holding and dragging. For each image, a custom dimension can be specified. Specifying the size of the images according to their size in the real world can help the tracking system to more easily recognize and register the object.

The size of the images in the real world was roughly estimated by the height of the buildings. The height of the buildings is referenced to the CityGML data. After knowing the height of the building, the length of the space outside the building is estimated and added to the total height of the entire image. The system will automatically calculate the width of the images after the height is input in meters.

4.1.2 Tracking multiple images

The ARFoundation SDK provides the "Augmented Images" API for image based tracking in Unity. It can be added as a default component called "AR Tracked Image Manager" under the ARSessionOrigin as a child. The API supports a Reference Image Library as an input that can contain up to 1000 images and can track up to 20 moving images at one time as mentioned in the previous subsection. The number of tracked images at one time is customizable by the developer for optimizing hardware performance. There is also a field to set the "Tracked Image Prefab", that will instantiate when each of the images is being tracked. However, the API only allows 1 prefab to be spawned even when different images are being tracked.

This app requires a different model to be instantiated when a different building is being detected. Each designated building requires its own model to be masked. To achieve that, a C# script is written to provide extra commands to the Augmented Image API. The script will allow the API to spawn different prefabs when a different image is being tracked according to the file names of the prefabs and the images.

To start with, the script needs to import some namespaces that are required to access and use the classes related to AR tracking. Importing these namespaces allows the use of functions such as ARTrackedImageManager in the script.

```
using UnityEngine.XR;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
```

After importing the needed namespaces, a few variables are declared. A serialized field that is an array of game objects is declared as placeablePrefabs. This will later allow the prefabs that will spawn with the tracked buildings to be added to the list. A private dictionary is declared as spawnedPrefabs to contain the name of the prefabs in the list. A Private ARTrackedImageManager is also declared.

```
[SerializeField]
private GameObject[] placeablePrefabs;

private Dictionary<string, GameObject> spawnedPrefabs = new Dictionary<string,
GameObject>();

private ARTrackedImageManager trackedImageManager;
```

To popularize the dictionary declared above, a for each loop is used to go through all and each of the placeable prefabs in the serialized list as the script instance. The loop prespawns the prefabs, gets the names of them, and adds the game objects and the names into the dictionary.

```
private void Awake()
        {
                trackedImageManager = FindObjectOfType<ARTrackedImageManager>();

                foreach(GameObject prefab in placeablePrefabs)
                {
                        GameObject newPrefab = Instantiate(prefab, Vector3.zero,
Quaternion.identity);
                        newPrefab.name = prefab.name;
                        spawnedPrefabs.Add(prefab.name, newPrefab);
                }
        }
```

The next part of the script is to set a function to get the status of the tracked image and change the behaviour of the program when the status is changed. The status of the tracked images is obtained from the class ARTrackedImagesChangedEventArgs. It tells if the status of the image is added, updated or removed. This function gives a command that if a new image is detected or moved during the tracking in the app, the image will be updated to the script for the next function. If the tacked image is removed from the camera, the spawned prefab will be set inactive and removed from the scene.

```
private void ImageChnaged(ARTrackedImagesChangedEventArgs eventArgs)
        {
                foreach(ARTrackedImage trackedImage in eventArgs.added)
                {
                        UpdateImage(trackedImage);
                }
                foreach(ARTrackedImage trackedImage in eventArgs.updated)
                {
                        if (trackedImage.trackingState == TrackingState.Tracking)
                {
                                UpdateImage(trackedImage);
                }

                else
                {
                        Destroy(spawnedPrefabs[trackedImage.name]);
                        spawnedPrefabs[trackedImage.name].SetActive(false);
                         }
                }
                foreach(ARTrackedImage trackedImage in eventArgs.removed)
                {
                        spawnedPrefabs[trackedImage.name].SetActive(false);
                }
        }
```

The last function of the script is to get the name of the tracked image that is inside the reference image library. If the name of the image matches the name of the prefab in the placeable prefabs list, the prefab will spawn according to the position and the rotation of the tracked image. The spawned prefabs will move and rotate when the camera or the tracked image moves. If a prefab has a name that does not match any of the tracked images, it will be set as inactive and will not be rendered in the scene.

```
private void UpdateImage(ARTrackedImage trackedImage)
        {
                string name = trackedImage.referenceImage.name;
                Vector3 position = trackedImage.transform.position;
                Quaternion rotation = trackedImage.transform.rotation;

                GameObject prefab = spawnedPrefabs[name];
                prefab.transform.position = position;
                prefab.transform.rotation = rotation;
                prefab.SetActive(true);

                foreach(GameObject go in spawnedPrefabs.Values)
                {
                        if(go.name != name)
                        {
                                go.SetActive(false);
                        }
                }
        }
```

The finished script is added as a new component to the ARSessionOrigin. The declared serialized field of placeable prefabs now allows the number of placeable prefabs to be set and added to the array on Unity's user interface. Once the placeable prefabs are added to the list, the script can now be tested with a built app running on the device.

4.1.3 Testing and Modification

The multi-images tracking system is implemented and tested indoors with smaller objects. Two different cuboids in different sizes are built in Unity to spawn when a paper box and a mint candy box are being tracked. The test result shows that the system is able to track the two boxes and spawn their corresponding object successfully. The following figure shows the reference object and the test result on the device.

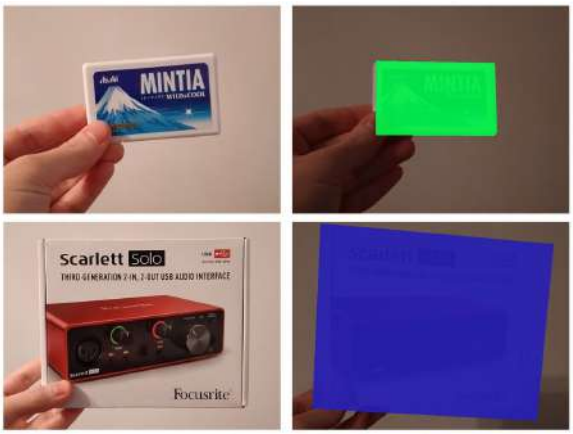

Figure 20 Testing result of the multi-image tracking implementation

However, a problem is shown during the test. Although being included in the function in the previous script, the system sometimes fails to deactivate the spawned object even if the tracked image is removed from the scene. The spawned prefab will stay and hover at the edge of the AR scene. The problem is likely a glitch in the ARFoundation SDK. To tackle the problem, an extra button and C# script are added to the app. By pushing the button, the app will terminate the current scene and ARSession and restart them. By resetting the ARSession, all tracking progress will be reset, the origin points of the AR device will be recalculated, and all spawned objects will be destroyed. The system will start tracking from the ground up again.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.SceneManagement;

public class resets : MonoBehaviour{

	public ARSession session;

	public void resetsession()
	{
		SceneManager.LoadScene("ImageTracking");
		session.Reset();
	}
}
```

The script simply used the LoadScene class in the SceneManagement namespace and the Reset class for ARSession to restart the whole session. The public ARSession variable is defined in the Unity interface as the current using ARSession. The public void variable is linked to a button to activate on click.

4.2 Gathering and creating suitable models for visualization

As discussed in section 3.1, a collection of high quality and suitable models will benefit the visualization as they provide the foundation and geometry for the rendering. In this case, a suitable building model can help to outline the designated building and provide a more precise masking for the masked grayscale visualization. In this project, 2 different types of models will be tested. The first type is the 3D LOD2 building models from a readily available set of CityGML data. The second type will be a set of manually created building models if the level of detail in the CityGML data is not sufficient.

4.2.1 Using the CityGML LOD2 Building Model

Data Extraction and Conversion

A set of CityGML data is available for the Stuttgart Nordbahnhof. To utilize the data in this app, the building models in the CityGML dataset need to be extracted and converted into individual building models as .OBJ files, which is a file format supported by Unity. To start with, an FME workbench is set up to extract the building models from the dataset. The results from the FME workbench are a 3D city model that only contains the buildings.

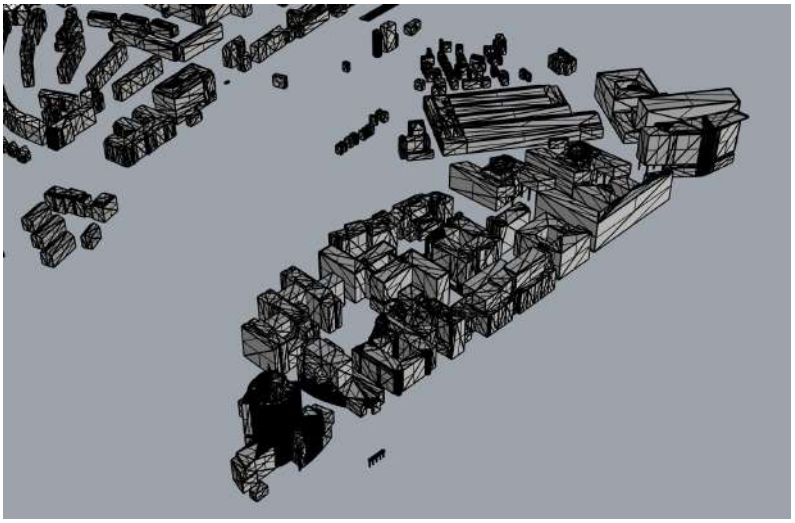

Figure 21 Building Models extracted from the CityGML Data

The models are imported into a 3D CAD software. The individual building models that are in the list of designated buildings are selected and exported into individual OBJ files. An OBJ file stores the geometry of the building model and is supported by Unity. The exported file has to

use a filename that is the same as its according reference image in order for the multi image tracking system to work.

Orientating the Model

Once the individual OBJ files are ready and imported into Unity, the next step is to orientate them so that they will be spawned in the correct location with the correct rotation when the according building is detected by the image tracking system. The placement of spawned objects with the ARFoundation SDK's Augmented Image API is not very clearly documented. There are also no visual aids that reference the position of the reference images for the orientation of the placeable prefabs. Tests have to be done to find out the exact position of the spawned object in relation to the reference image.

After some tests, it was found out that the spawned object will be placed in a position that is as if the reference image is placed on the XZ plane. The center of the reference image will be on the origin of the plane (0,0,0). To properly orientate the building models, a quad is placed on the XZ plane with the dimensions of the reference image. The reference image was used as a 2D texture placed on the quad. Now, with the reference image visible in the system at the correct position, the building models can be orientated accordingly and saved as a placeable prefab.

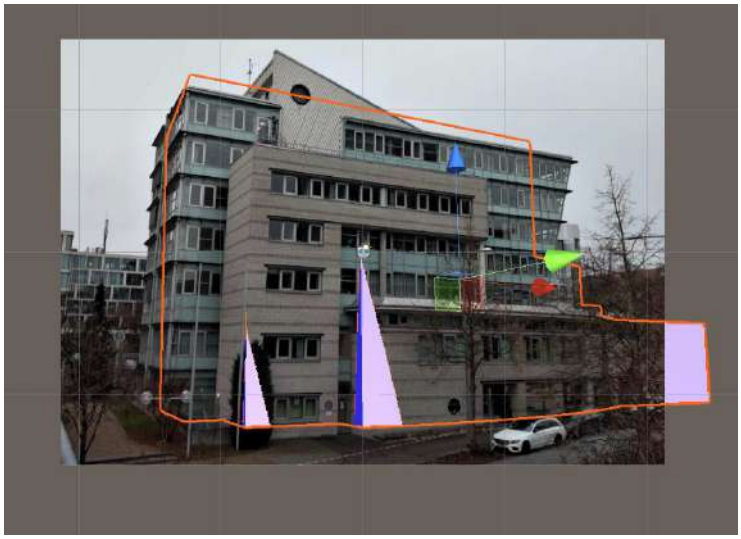

Figure 22 Orientating the building model in Unity with its reference image

However, even when the LOD2 models can be orientated with the reference images located at the correct location in the system, it is still difficult to perfectly match the 3D model with a 2D image at a slightly tilted angle with a perspective. There are also parts of the building that are

not constructed in the 3D model. As a result, as shown in the image above, there are some parts of the building that are not included in the building while some parts of the background are included.

4.2.2 Manually Built Models

The second type of model tested for this app is a manually created building model. The aim of building the model is to create a more suitable model that can fit the entirety of the building's silhouette and therefore provide a more precise visualization. The manually created models are built in the 3D CAD software.

In the software, the reference image of each designated building was imported as a background bitmap. The bitmap is scaled and positioned in the system according to their position in Unity mentioned in the previous subsection. With the reference image correctly placed, the outline of the buildings was traced as a polyline. The enclosed polyline is then converted to a surface. The surface should cover the buildings as shown in the reference image perfectly. Each surface can then be exported in OBJ files and imported into Unity and saved as a placeable prefab.

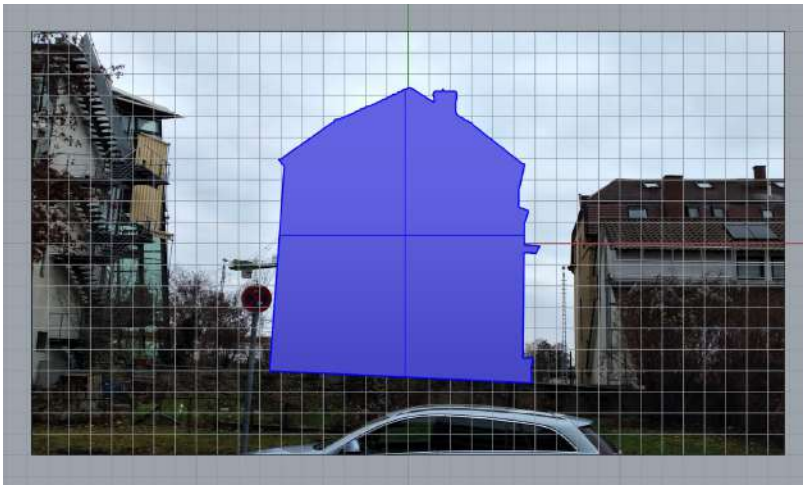

Figure 23 Created surface that covers the whole building in the reference image

The image above shows an example of one of the surfaces created as a building model for the app. Since the model is manually built, all the visible artifacts of the buildings seen in the reference image can be included, including chimneys, windowsills and clothes racks. All of these artifacts will be included in the visualization in the app after deployment. However, since the model is only created according to the reference image, the visualization will work perfectly but only in the selected viewpoint for the users.

4.3 Implementing the Visualization Method

As discussed in section 3.6, the desired visualization is achieved by authoring shaders in Unity. Shader commands the GPU to perform the desired rendering method. In Unity, shaders are wrapped in a material and applied to a 3D object. To achieve the grayscale filter with masked building visualization, two shaders are authored. They are both wrapped in their corresponding materials and applied to the suitable 3D objects. One of the shaders is going to be a grayscale filter that will surround the AR camera and turn the whole scene into monochrome. The other shader is going to be applied to the building models created in the previous section to exclude the buildings from the grayscale rendering.

4.3.1 Creating a grayscale filter for AR camera

The concept of creating a grayscale filter is straightforward. The shader needs to get the RGB data of the pixels behind the filter and render them with a simple equation that turns the RGB values into monochrome. The filter needs to be universally applied to the AR camera. A 3D model needs to be built for that purpose to cover all the directions from the camera.

The Shader

The shader for creating the grayscale filter uses a specific pass type called “Grab Pass”. It grabs things that lay behind the object when rendering and turns them into a texture. The texture can then be used for other passes commands for different image based effects.

The first part of the shader sets the rendering order of the object. Setting it as “Transparent” allows it to be rendered after any other geometries and the background, providing the object a chance to grab what stands behind it after they are rendered. The Grabpass command is then declared in the subshader. The content that it grabs will turn into the texture called "_BackgroundTexture".

```
Tags { "Queue" = "Transparent" }

GrabPass
{
  "_BackgroundTexture"
}
```

With the texture generated by the GrabPass command, the next step is to render the texture with a grayscale effect. To do that the pass needs to first calculate the clip-space of the vertex and get the correct coordinates of the texture so that the texture will be rendered in the correct position on the screen. UnityObjectToClipPos from UnityCG.cginc is used to calculate the vertex position. And ComputeGrabScreenPos is used to get the coordinates of the texture sampled from Grabpass.

```
CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag
    #include "UnityCG.cginc"

    struct v2f
    {
      float4 grabPos : TEXCOORD0;
      float4 pos : SV_POSITION;
    };

    v2f vert(appdata_base v) {
      v2f o;
      o.pos = UnityObjectToClipPos(v.vertex);
      o.grabPos = ComputeGrabScreenPos(o.pos);
      return o;
    }
```

The last part of the shader uses the texture form Grabpass and turns the color into a monochrome. The half4 vector named bgcolor is used to get the R, G, B, and alpha values from

the _BackgroundTexture. The return uses a simple image processing equation, calculating the average value of R, G, and B to create a monochrome value for each pixel.

```
sampler2D _BackgroundTexture;

half4 frag(v2f i) : SV_Target
{
    half4 bgcolor = tex2Dproj(_BackgroundTexture, i.grabPos);
    return (1*bgcolor.r + 1*bgcolor.g + 1*bgcolor.b) / 3;
}
```

The shader is tested in Unity using a color image. A cube is created on top of the image and the shader is applied to the cube. The result shows that the shader is able to successfully render the texture from the color image and turn it into grayscale. The following image demonstrates the result. The red box represents the coverage of the cube with the applied shader. The area covered by the cube is rendered in monochrome and anywhere outside is in color.

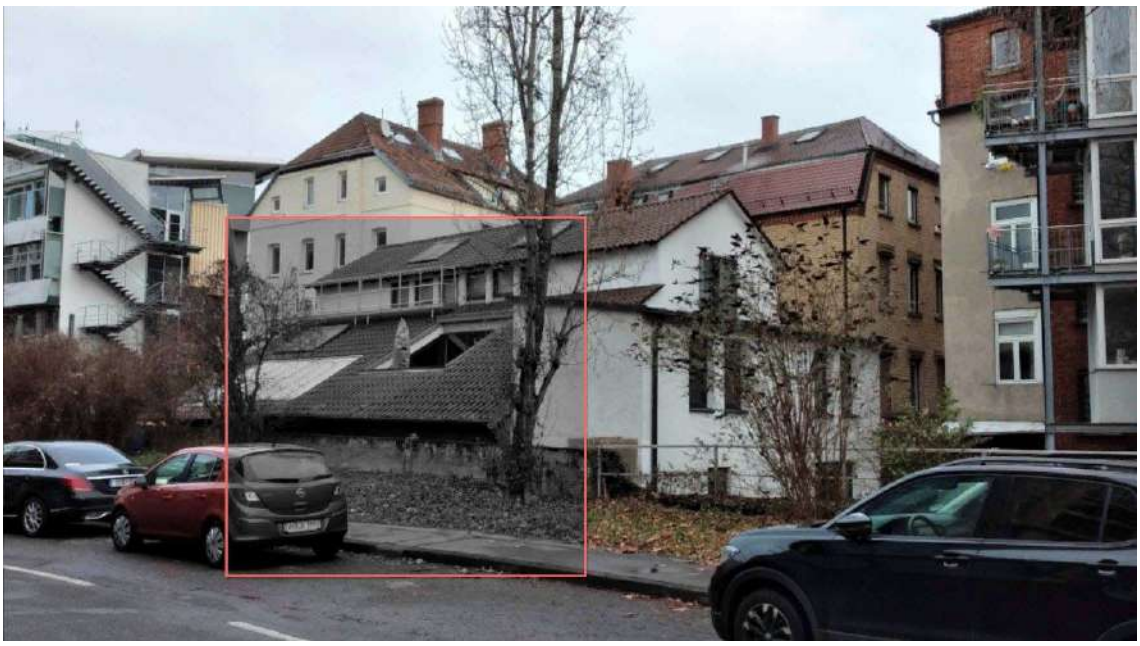

Figure 24 Demonstration of the rendering result of the grayscale filter shader.

Achieving a more effective grayscale filter

The test result shows that the shader is able to get the background and render it into a grayscale texture. However, in a visual point of view, the average method calculating the mean of the R, G, and B values is a basic equation for converting color images into grayscale. But the visualization is a bit bland and uninteresting to the human eyes.

The average method for color to grayscale conversion:
$$(R + G + B) \div 3$$

One alternative to the average method for converting color images to grayscale is called the luminosity method. The luminosity method is a weighted average of the RGB values with the color perception of the human's eyes taken into account. As the human's eyes are more sensitive to the color green, the equation weighted the R value higher than the other two. In theory, using the luminosity method can result in a more vibrant and contrasting grayscale image. (Cook, 2009)

The luminosity method for color to grayscale conversion:
$$0.21\ R + 0.72\ G + 0.07\ B$$

The luminosity method is added to the shader and tested with the same image above. The image below shows the comparison between the result of the conversion with the average method and the conversion with the luminosity method. The comparison shows that the result from the luminosity method shows a sharper visualization with a more defined rendering. It will provide a more effective grayscale filter to the app and is applied to the shader for the filter.

Grayscale Conversion Using the Average Method *Grayscale Conversion Using the Luminosity Method*

Figure 25 Comparison of the results between the grayscale conversion of the average method and the luminosity method

The equation is added to the last part of the shader and replaced the average conversion equation so that the final rendering will return with the result luminosity conversion.

```
half4 frag(v2f i) : SV_Target
        {
          half4 bgcolor = tex2Dproj(_BackgroundTexture, i.grabPos);
          return (0.21*bgcolor.r + 0.72*bgcolor.g + 0.07*bgcolor.b) ;
        }
```

Building the 3D Model for Rendering the Grayscale Filter

As mentioned in section 3.6, each shader needs to be wrapped in a material and applied to a 3D model in order to actually render on the screen of the device. The shader of the grayscale filter is now completed and a generic material is created to wrap the shader. No extra setting is needed on the material. The material now needs a suitable 3D model in order to provide a grayscale filter universally to the AR camera.

The idea for the 3D model in this case is to surround and enclose the AR camera with 6 outward facing “windows”. Each window will be using the grayscale filter shader. With that, no matter in which direction the camera is looking at, it will always look into one of the outward facing filters and the scene will be rendered in grayscale.

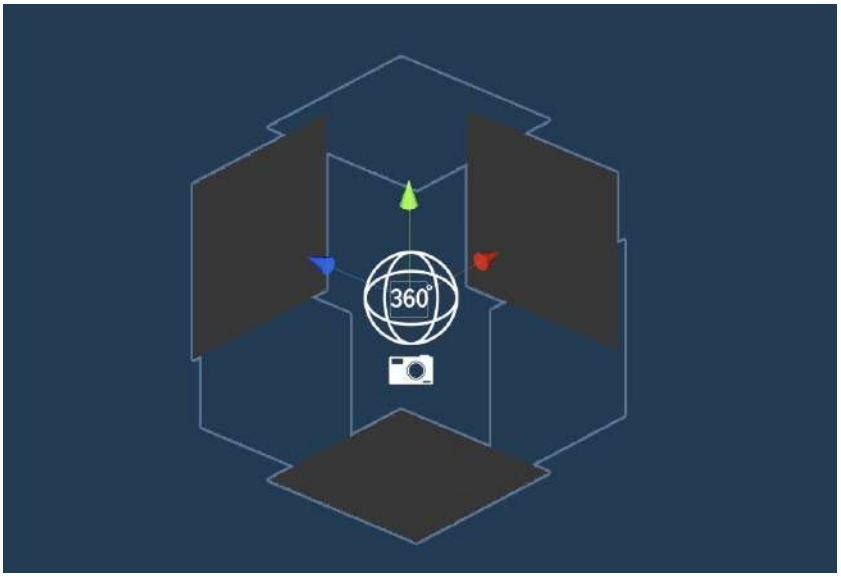

Figure 26 Demonstration of the idea of creating outward facing windows for the grayscale filter

The 3D model is built in Unity with 6 quads. Each quad is facing outwards. The dimensions of them are 50x50 meters. The size of the model prevents the users from walking outside the enclosed filter box. The borders of the 6 quads are intersecting with each other, forming a 50x50x50 meters cube. The centroid of the cube is located at (0,0,0), which is at the same location as the AR camera.

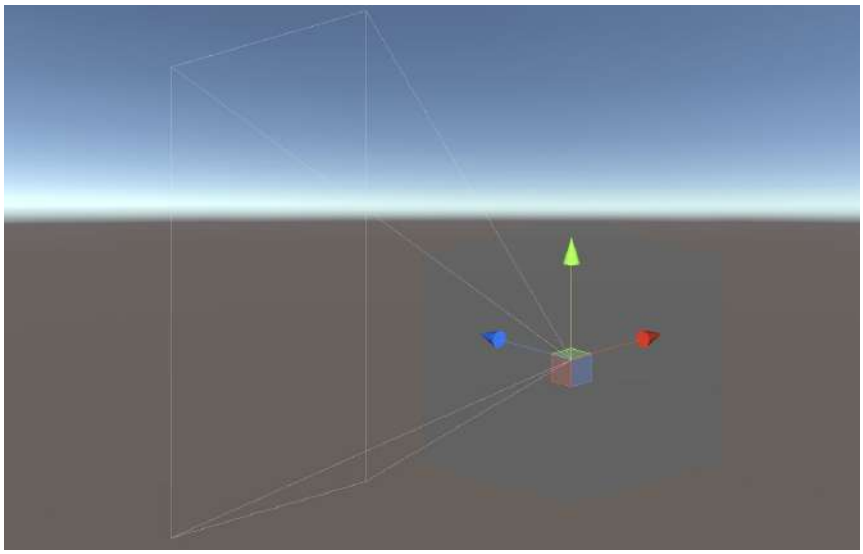

Figure 27 Demonstration of the enclosed cube built in Unity

With the model built and the shader applied, the app is built and run on the Android device for testing. The image below shows the final testing result of the grayscale filter applied to the AR camera. The filter applies to all the directions around the camera and successfully turns the surrounding environment into monochrome on the device's display.

Figure 28 The testing result of the grayscale filter applied to the AR camera

4.3.2 Creating a shader to exclude placed object from the grayscale filter

The second shader authored for this app is a shader that will exclude the buildings from the grayscale filter. This will be achieved by running a stencil buffer with the shader applied to the building models. A stencil buffer will compare the reference number set in each of two different shaders and test if they are passing the condition set. If the condition is passed, the overlapping pixels of the two objects applied with those two shaders will be rendered normally. But if it does not pass the stencil test, the pixel will be omitted and will not be rendered.

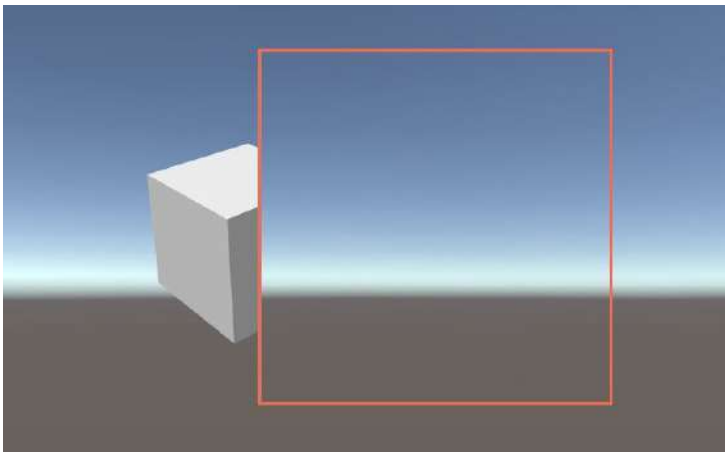

Figure 29 A failed stencil test will omit the rendering where two objects overlap

The image above demonstrates the possible functionality of the stencil buffer. In this case the white cube and the red box are applied with two different shaders. In both of the shaders, the reference number for the stencil test is set as 1. However, the passing condition in the shader of the white cube is set as “NotEqual”, meaning that the overlapping pixels will be rendered only when the reference number in the two shaderes are not equal to each other. In this case, since the reference number in the two shaders are both set as 1, it can not pass the stencil test. As a result, when the white cube is lying behind the red box on the scene, the pixel of the overlapped white cube will not be rendered, making that part of the cube disappear behind the red box.

The idea of this is to apply the same stencil buffer test to the grayscale filter and the shader for the building models in a way that they do not pass the condition test. As a result, the rendering queue will ignore the pixel where the grayscale filter is covered by the building model, leaving the area of the building out of the grayscale filter. With this, the visualization of the masked building with a grayscale context can be achieved.

The shader for the building models is a rather simple one. As it does not require any spatial effect when rendered, the shader only needs to specify the reference number for the stencil test and the pass type of the stencil buffer. In this case, the reference number is set as 1 and the pass type is set as “replace” in order to give the whole area of the object the value 1.

```
SubShader
    {
        Zwrite off
        Colormask 0
        Cull off

        Stencil{
            Ref 1
            Pass replace
        }
    }
```

The stencil test would not work without two different shaders participating. Therefore, a stencil buffer function also needs to be added to the grayscale buffer shader mentioned in the previous subsection. In the grayscale filter shader, a property is added so that the condition for the stencil

test could be set in the material interface later. A stencil function is added to the subshader. The reference number is also set as 1.

```
Properties{
      [Enum(Equal,3,NotEqual,6)] _Stencil("Stencil Test", int) = 6
   }
.
.
.
Stencil{
      Ref 1
      Comp[_Stencil]
      }
```

The test condition is then set as “NotEqual” in the material for the grayscale filter. With that set, since the both shaders could not pass the stencil test, the part of the grayscale filter covered by the building model will not be rendered, creating a masked filter for the buildings.

4.3.3 Testing the result of the visualization on site

At this point, all of the components for achieving the AR visualization are ready. The shader for the grayscale is applied to the constructed cube. The model is set as a prefab that will spawn when any one of the reference images is being tracked. The stencil buffer shader is applied to the manually created building models mentioned in section 2.3. Each of the building models is turned into a prefab in Unity with a filename that is the same as their corresponding reference image. These prefabs are added to the list of placeable prefabs in the multi-image tracking system mentioned in subsection 4.1.2.

With these elements all set and ready to go, the app is built and run in the Android device. A testing session was carried out in the Nordbahnhof district to try visualize the buildings with the masked grayscale filter. The following are some of the images captured during the test showing the results of the visualization.

Figure 30 Examples of the visualization result tested on site

The result shows that the system is able to detect the designated buildings and apply a masking filter to them. The buildings are rendered in color while the surrounding context is in grayscale. The overall precision of the placement of the mask is high, although some of the smaller artifacts on some buildings are missed.

4.4 User Experience

The AR tracking system and visualization is already completed at this stage. However a few additional functions are needed to increase the usability and improve the AR experience for the users. Mentioned in section 3.8, the app should provide information for the users on the location of the buildings and their viewing point in order to let the system track them and visualize them.

The design of the user interface should be minimalistic to keep the users' focus on the visualization. In this case, an image capture function is also added to the app for urban planners to record the visualization or keep the images for printing or further editing.

4.4.1 Addition of navigation map

The tracking method chosen for this app is the image based tracking method. With this method, the tracked buildings are treated as a "framed scene". The system is only able to detect the buildings when the user is able to find and put the "framed scene" within the camera of their device. For this reason, it is important for the users to know where and how to find the designated buildings. A navigation map is added to the app to cater this need. The map uses the GPS sensor in the device of the users and allows them to navigate and know their current position. The observation points are added to the map with their view direction. The users can use the map to find the observation points, look into the correct direction, and allow the system to detect and visualize the designated buildings.

To achieve that, the Mapbox SDK. The Mapbox SDK provides the service for building location based games in Unity. It provides an API to access the styled maps through the application. There is an online platform for customizing the maps, adding geolocation data, and publishing them. The Mapbox SDK is free to use with a monthly limited access.

In this case, a simple map is created on their online platform. The data of the observation points are added to their server along with the viewing direction. The map is published and accessed through the SDK. The app will ask for the user's permission to use the GPS in theri device. The map in the app will update according to the current location of the user and tell them the location of the observation point. The image below shows the navigation map in the app. The pink navigator shows the current location of the user. The blue dots are the location of the observation points and the arrows are the direction of view. The users can walk around the area, look for the blue dots and see the visualization through the AR scene.

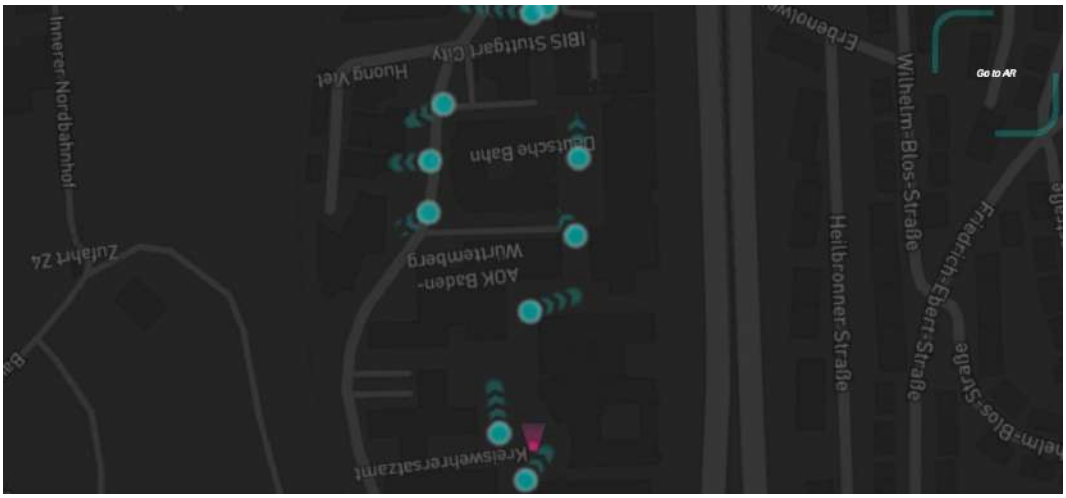

Figure 31 Navigation Map added in the app

A button and a simple C# script is also added to allow the users to switch between the AR scene and the map. The script simply uses the class "LoadScene" under the Scene Manager namespace to allow the users to switch through the two scenes on click.

```
public void loadthear()
        {
                SceneManager.LoadScene("Imagetracking");
        }
```

4.4.2 Image capture

Considering that the users might need to save the visualization for future presentation, sharing, further editing, and printing, an image capture function is added to the app as the final addition. The function captures the AR scene in a higher resolution and saves the file as a PNG raster into the app's folder locally in the device. The function is achieve by a piece of C# script.

The Unity scripting API has the CaptureScreenshot class that allows the app to capture a screenshot and save it in a local file. To write a unique file every time without overwriting the previous image, the time and date of the screenshot is added in the filename. The file extension ".png" is also added to specify the file type when the screenshot is saved. The integer 4 is for specifying the resolution of the saved image. In this case it will be 4 times the resolution of the screen of the device. As the date and time of the system will only update when the scene is loaded, the LoadScene class is added to the script to reload the scene every time a screenshot is saved to update the time and prevent overwriting of files.

```
String thisDate1 = DateTime.Now.ToString("MMddyyyyHmmss");


public void capture()
{
        ScreenCapture.CaptureScreenshot("arsc"+thisDate1+".png", 4);
        SceneManager.LoadScene("ImageTracking");
}
```

The captured screenshot will be saved in the path "/storage/emulated/0/ Android/data/<packagename>/files", named with a specific time and date and formatted as a PNG file. The users will be able to access, share, or edit the images once the file is saved.

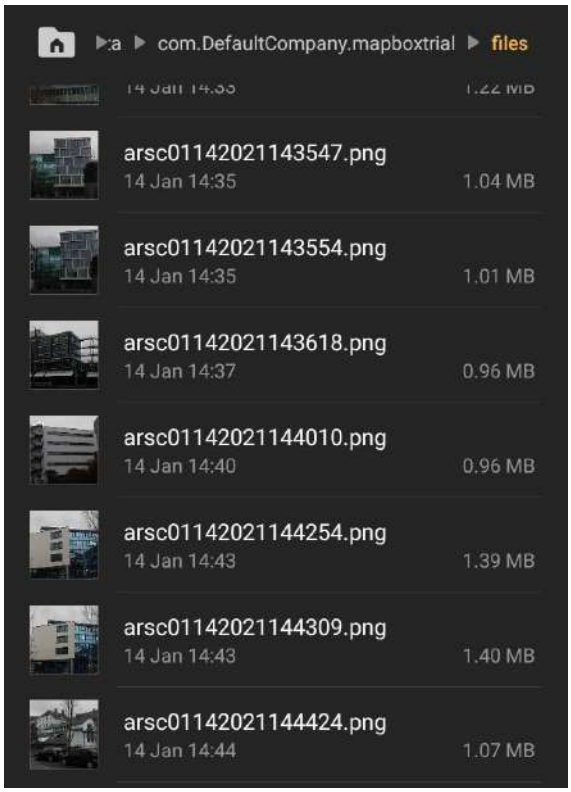

Figure 32 Images saved in the local file of the device with the image capture function

5 Evaluation

5.1 Performance of the image based tracking method

Being able to track the object in real time and spawn the visualization without latency is crucial to an AR visualization app to provide an immersive experience. To achieve that, an efficient and effective tracking method needs to be implemented. For this app, an image based tracking method was chosen for its lightweightness. About a dozen different buildings needed to be tracked on the site for this app. An image based tracking can reduce the size of the reference database, reduce the time of the hardware searching through it and possibly reduce the latency of the tracking process.

To trial and evaluate the implemented image based tracking, a tracking speed test was conducted to calculate the time needed for the system to track the building and visualize the effects. The app was built and installed to the device mentioned in section 3.7. The app was tested on site. While the app is running, a screen recording is performed for each buildings' visualization. The screen will be recorded at a 30 frame per second frame rate. The recorded video will then be analysed for the number frames needed for the system to go from starting the AR session to the visualization is completed. Each frame needed is equal to 1/30 of a second. For that the time needed for each tracked building to be visualized can be recorded.

The result of the test is listed as following:

Building name	Frames needed (30fps)	Time needed (S)
Kreiswehrersatzamt	4	0.1333333333
Karrierecenter	3	0.1
Diakonisches Werk	5	0.1666666667
AOK	6	0.2
Schulentwicklung	4	0.1333333333
Deutsche Bahn	5	0.1666666667
Schulentwicklung Hinter	3	0.1
HDI	4	0.1333333333
Takkt	3	0.1
House1	3	0.1
House2	3	0.1
Weckherlin Haus	5	0.1666666667

Table 3 The time needed for initiating the visualization on each building

The average frame needed to all the designated buildings to get recognized and visualized is 3.92. With the 30 fps frame rate, the average time used for the visualization is about 0.13 second. The latency is only a tenth of a second and on the app it feels like the visualization is happening in real time. The tracking system is successful in terms of keeping the computation fast and the latency of visualization low.

Limitation of Using a Image Based Tracking System for Outdoor Building Tracking

The implemented tracking system is efficient in terms of speed and tracking efficiency. However, to track and visualize objects in an outdoor setting, the method still poses a few notable limitations.

The image based tracking method is a 2D based tracking. To implement that for tracking 3D buildings, the buildings have to be treated as a "framed picture". As a result, the system can only recognize a building from one specific direction. The users are therefore required to search for those specific points and directions in order to visualize the building in the app. This reduces the usability of the app and constrains the angle that the buildings are being visualized. The users cannot walk around the same building and apply the same effect to the different facades unless there are reference images in the database for that particular angle. The image based tracking method also cannot track an image from an overly tilted angle. Meaning that if a building is too tall and requires the users to look up, the tracking may fail.

A 2D tracking method also applies that the system does not understand 3D space. Depth of the object related to the surrounding is not registered in the system since the 3D building in the real world is not treated as an 3D object in the app. As a result, the tracking method cannot avoid the artifacts standing in front of the designated buildings. If there are artifacts such as trees, cars, and pedestrians standing in front of the building, it will include those artifacts into the colored visualization instead of turning them to monochrome.

The following image shows the result of the visualization applied to one of the buildings. As there is a car parked in front of the building and is overlapping with the building's outline, part of the car is rendered in color. This problem is hard to be fixed with the 2D image tracking method applied to track 3D buildings.

Figure 33 Part of the car is rendered in grayscale

5.2 Comparison of the two building model types

Accuracy test

Two different types of building models were tested for the visualization during the development process. One is the LOD2 extracted from the CityGML data and the other one is the manually created model specifically made for this visualization method. In the app, the manually created model is adopted for the final result as it provides a more accurate and precise visualization. Since the model is created for each viewpoint specifically, it can cover the entire building when it is viewed from that viewpoint. As a result it provides a more convincing visual effect.

The images below show one of the results of the visualization using both of the model types. The building being tracked in this case is a comparatively smaller residential building in the site. The LOD2 model performs the best on cases like these when the building is smaller and its form is relatively simple. The result using the LOD2 model (left) shows that there are still artifacts on the building, such as the chimney on the roof and the balcony on the side, that are left in grayscale due to the insufficiency of the level of detail. There are also parts of the sky that are rendered in color due to the placement of the model. However, the overall result is still visually acceptable. On the other hand, the one using the manually created model (right) shows that almost the entirety of the building is rendered in color and no big area of the surrounding is included.

Figure 34 Comparison of the visualization result using LOD2 (left) and manually created (right) building models with a residential building

The visualization result using the LOD2 model is imperfect yet promising when it is applied to smaller buildings with simpler forms. However, if the buildings are larger with a more complicated design, the visualization result will be varied. One of the major challenges when implementing the LOD2 models is to properly orientate the 3D models with the 2D reference image when the angle of taking the images is tilted and perspective is present. The challenge becomes more noticeable when the building is bigger and is harder to be captured in one frame without tilting the camera angle. The images below show an example where the LOD2 model is applied to an office building on the site. A tiled angle of view is needed to capture the silhouette of the building. As a result, orientating the model is more difficult and the visualization shows a larger area of the sky being included in the color mask.

Figure 35 Visualization result using LOD2 models on an office building

There are also cases where the data comes in insufficient. The LOD2 models used in the app is from the Stuttgart CityGML dataset. The following images show the visualization result of the building Karrierecenter on Heilbronner Straße 188, 70191 Stuttgart. In this case, the model of this particular building built for the CityGML data was built in LOD1. The image shows that

there is a large part of the roof structures that is not included in the model. As a result, that part of the building can not be included in the visualization, causing a spike in the impreciseness.

Figure 36 Comparison of the visualization result using LOD2 (left) and manually created (right) building models on Karrierecenter, Heilbronner Straße 188, 70191 Stuttgart

To more accurately survey the "impreciseness" of using each type of model, the following equation is used:

(*Area that should be included* + *Area that is wrongly included to the color mask*) ÷ *Area of the building* × 100%

The equation calculates the percentage of area that is wrongly visualized in the app compared to the surface area of the building in the AR scene. The result of the calculation is listed in the following table:

Building name	Impreciseness using LOD2 Models (%)	Impreciseness using Manually Created Models (%)
Kreiswehrersatzamt	7.230	0.6909
Karrierecenter	10.184	0.7256
Diakonisches Werk	7.484	0.7353
AOK	6.676	0.7191
Schulentwicklung	7.545	0.7880
Deutsche Bahn	6.295	0.6141
Schulentwicklung Hinter	6.636	0.6186
HDI	6.982	0.6401
Takkt	7.179	0.8340
House1	3.767	0.4792
House2	3.184	0.5432
Weckherlin Haus	6.961	0.6125

Table 4 Comparison of the impreciseness of using the two model types

The average impreciseness of the LOD2 model is about 6.67% while the average for the manually created models stands at around 0.67%. The number suggested the manually created model is 10 times more precise in masking the building compared to the readily available LOD2 3D models.

Time cost record

The difference in terms of visualization preciseness is obvious between the two model types tested for the app. However, the preciseness should not be the only consideration when choosing which type of models to adopt. The cost of implementation should also be taken into account when making the decision. In this case, the time cost for implementing one LOD2 model is obviously going to be lower than creating a manually created model as the former is already readily available and only orientation in Unity is required. For a more detailed comparison, the time needed for implementing the both types of models are recorded and listed on the table below:

Building name	Time needed for implementing LOD2 Models (nearest minute)	Time needed for implementing manually created models (nearest minute)
Kreiswehrersatzamt	4	23
Karrierecenter	6	19
Diakonisches Werk	5	18
AOK	4	24
Schulentwicklung	7	22
Deutsche Bahn	5	16
Schulentwicklung Hinter	4	18
HDI	5	20
Takkt	6	17
House1	3	14
House2	3	14
Weckherlin Haus	4	22

Table 5 Comparison of time needed to implement each model type

The average time needed to implement one LOD2 model into the AR system is about 5 minutes, while the average time needed for each manually created model is 19 minutes. Moreover, since the LOD2 models are 3D models, the same implemented model can be used for different view directions towards the same building. However, as the manually created models are created

according to the silhouette of the building in one specific viewpoint, if the app requires to visualize the building from more than one direction, more models are needed to be created for the same building. The implementation time will easily multiply when there are more than one observation point established for a single building.

Overall, the LOD2 models are a more time efficient option for this visualization. The manually created models can achieve a more precise and immersive visualization. However, it requires a more labour intensive process. The time needed for this method can be a few times more than using the LOD2 models. If the development process has a limited time period or budget, the CityGML data can be an efficient substitution although the accuracy will be sacrificed.

5.3 Performance of the ARFoundation SDK

The ARFoundation SDK and ARCore extension provides a wide range of features that helps the development of this app. The SDK is easy to adopt and use. If a simple AR app is needed to be built without many customization, the developer does not need much coding knowledge to lay the foundation. Unity and the SDK allows simple “drag and drops” of components and a few adjustments and the app is ready to build and run in the device. The SDK provides handy and readily available APIs such as Plane Detection and Augmented Images for a quick and convenient implementation. The system has clear notification for bugs in the customized script and provides clear instructions for where to check and how to fix them.

However, despite the user friendliness, a few shortcomings are found during the app development phases with ARFoundation SDK. One of the major problems discovered is that there are some lacking parts of documentation provided by Google and Unity. The Augmented Image API in ARFoundation is used for tracking the buildings in the app. One of the most crucial parts of an image tracking system is to place the object on a desired position according to the tracked image. For some other SDK available in the market, a visual aid is usually provided to match the position of the reference image and the spawned object. A proxy reference image will be added to the scene and the initial spawn position and rotation of the object can be adjusted and saved to the system. However, the ARFoundation SDK does not provide this aid. The position and orientation of the tracked image and their relationship to the device’s orientation is not stated in any documentation provided. Experiments had to be done to find out how to correctly position the placeable objects according to the reference image. The results show that the trackable images are projected to the XZ plane in Unity. Once the

dimensions for the images are set, the center of the image will be referenced to the origin point (0,0,0). The height of the image will be the length on Z axis and the width will be length on the X axis. To visualize the reference images for visual aids to the positioning of models, a quad can be created on the XZ plane. Scale the quad according to the size of the image and apply an unlit material with the original image used as a 2D texture. The quad will be placed centered at the origin point and the image will be visualized as a reference.

Another notable limitation of the ARFoundation is the bugs and glitches found during the app development process. Already mentioned in section 4.1.2, the ARFoundation namespace has a class that tells the status of the tracked images. The class is supposed to be able to tell when a tracked image is no longer observable through the camera. However, in the development process, it is found out that the system sometimes fails to recognize that the image is not being tracked anymore. In the app, the script is written that when the image is not being tracked anymore, the spawned object will be set inactive in the scene. However, since the system fails to tell when the image is no longer in the scene, the spawned object will stay at the edge of the scene. This problem also leads to a failure when the system needs to track the next image. The status of the first object can not change to be “removed” and it can not proceed to detect the next object.

A major glitch also appeared in the development process. The app was first built with ARFoundation Version 3.1.10, which is the verified version for Unity 2020.1.9f1. During the development and testing process, one of the built versions of the app experienced a glitch that gives the AR camera a flashing green color. Part of the screen was flickering between a normal camera view and a bright green color. The bug was reported also by some other ongoing projects. The problem was solved at the end by updating the SDK to Version 4.1.1, which is a newer but unverified version.

6 Conclusion

In this thesis, an mobile Augmented Reality application is developed to showcase the technicality of visualizing focus and context on buildings with grayscale masking filters. The visualization application is oriented towards the needs of urban planners and designers and it successfully incorporated the visualization methods with the video based mobile AR technologies.

The state of the art of the mobile AR technologies was reviewed alongside with the outdoor tracking methods, development platforms and methods, and their application to the field of urban planning and design. Suitable technologies, concepts and methodologies are selected and applied to the implementation of this application. The image based tracking method was selected for this app. Despite not being optimised for tracking outdoor 3D objects such as buildings, the development of this project proves that with a proper implementation and planning, the image based tracking method can be applied for tracking buildings in an urban environment. Building the reference image library with selected buildings and viewpoints and adding a clear navigation map can allow users to find the specific locations and points of view where the designated buildings can be found and detected. Inputting the dimensions of the buildings also helps the system to detect the buildings more easily. The result of the implication is a system with a lightweight database that can track buildings and visualize the visual effect in real time with low latency.

The visualization method was implemented by authoring two different shaders. One shader is using the Grabpass method to gather information behind an object and recalculate the RGB values into a grayscale value. Applied that shader to a 3D model surrounding the AR camera and a universal grayscale filter can be achieved. The masking effect can be achieved by a shader using the stencil buffer technique. And when the shader is applied to a suitable building model, the building in real life will be omitted from the grayscale filter. Combining the two shaders, this app is able to visualize the buildings in Nordbahnhof, Stuttgart with the grayscale masked filter.

Using suitable building models is also important for applying this visualization method into an AR system. With the two different model types tested in this thesis, it can be concluded that the LOD2 3D building models is a more cost efficient solution when the CityGML data is available. The precision of visualization with this solution is however lower. If a more accurate and

precise visualization is needed, models have to be manually created for each building from each viewpoint according to the reference images. In this case, the labour time cost will be higher than using the LOD3 models.

Future Work Recommendation

This thesis mainly focuses on the technicality of applying a masked filter to buildings in an urban setting with mobile AR technologies. The implemented visualization methods such as the grayscale filter and the luminosity color to grayscale conversion method was selected based on the assumption that they are helpful to urban planners and designers. The assumption was made by observing the works from urban planners and designers. However, in actual practices, the professionals often need to face everchanging problems and can demand different visualization styles for different situations. To make this developed technology more versatile and applicable in more different situations, a higher level of customization can be provided, meaning that the users can tweak the visualization style in real time when using the app. For example, sliders can be added for the users to adjust the brightness, contrast, saturation level, or sharpness of the video from the AR camera. With these customization available, the urban planners and designers will be able to adapt a different visualization style if desired. That can also tackle the problem of inadequate lighting in the outdoor setting such as situations when the building is sitting in a poor lighting or is being shadowed.

Figure 37 Using a blurring effect the provide focus and context

The grayscale filter is a common method to put focus on a selected future by reducing the amount of information coming from the surrounding. However, it is not the only available and adopted visualization method. While this thesis showcased the possibility of applying the grayscale masking filter with mobile video based AR systems, more different visualization

methods can be explored to achieve a similar goal with these technologies. For example, a blurring effect can also reduce the complexity of the context of a building. (Example in the image above) It is also commonly adopted as a visual communication technique amongst urban designers and is aesthetically neutral compared to the grayscale filter. With the availability of the AR masking technology, visualization styles like this can possibly be applied in the future if it is beneficial to the urban planning process.

References

Akahoshi, K. et al., 2020. I-Urban Revitalization: Conceptual Modeling, Implementation, and Visualization Towards Sustainable Urban Planning Using Citygml. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-4-2020. Available at: https://pdfs.semanticscholar.org/a2e2/9361ab4070e87afd420a6e3eead084b6b441.pdf [Accessed 2021].

Anagnostou, K. & Vlamos, P., 2011. Square AR: Using Augmented Reality for Urban Planning. *2011 Third International Conference on Games and Virtual Worlds for Serious Applications*.

Apple Inc., 2021. ARKit - Augmented Reality. *Apple Developer*. Available at: https://developer.apple.com/augmented-reality/arkit/ [Accessed February 16, 2021].

ARPost, 2019. The Role Of AR And VR In Urban Planning. *ARPost*. Available at: https://arpost.co/2019/01/10/role-ar-vr-urban-planning/ [Accessed September 2020].

Arth, C. et al., 2015. The history of mobile augmented reality. arXiv preprint arXiv:1505.01319.

ARToolworks, 2021. *ARToolworks*. Available at: https://artoolworks.com/products/commercial-software.html [Accessed 2021].

Azuma, R.T., 1997. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4), pp.355–385. Available at: https://www.cs.unc.edu/~azuma/ARpresence.pdf [Accessed 2021].

Basu, A., 2019. A brief chronology of Virtual Reality. arXiv preprint arXiv:1911.09605.

Bednarczyk, M. & Templin, T., 2020. Mobile Augmented Reality Application Supporting Building Facades Visualization. *International Conference on Durability of Building Materials and Components*, 15. Available at: https://www.researchgate.net/profile/Michal_Bednarczyk/publication/345426112_Mobile_Augmented_Reality_Application_Supporting_Building_Facades_Visualization/links/5fb3bd4245851518fdad0039/Mobile-Augmented-Reality-Application-Supporting-Building-Facades-Visualization.pdf [Accessed 2021].

Behringer, R., Park, J. & Sundareswaran, V., 2002. Model-based visual tracking for outdoor augmented reality applications. *Proceedings. International Symposium on Mixed and Augmented Reality*. [online] Available at: http://www.geocities.ws/r_behringer/publications/2002_ISMAR_proc_RB.pdf [Accessed September 2020]

Bhasin, K., 2017. Pokémon Go Never Went Away—and Neither Did Its Technical Woes. *Bloomberg*. Available at: https://www.bloomberg.com/news/articles/2017-07-24/pokemon-go-never-went-away-and-neither-did-its-technical-woes [Accessed 2021].

Blanco-Pons , S. et al., 2019. Augmented Reality Markerless Multi-Image Outdoor Tracking System for the Historical Buildings on Parliament Hill. *Multimodal Technologies and Interaction*, 11(16). Available at: https://www.mdpi.com/2071-1050/11/16/4268 [Accessed 2020].

Bloomberg, 2020. [Press Release]. Why Augmented and Virtual Reality (AR/VR) Markets are Projected to Exceed $18 Billion This Year. Available at: https://www.bloomberg.com/press-releases/2020-01-15/why-augmented-and-virtual-reality-ar-vr-markets-are-projected-to-exceed-18-billion-this-year [Accessed 2021].

Burke, D., 2017. ARCore: Augmented reality at Android scale. *Android Developers Blog*. Available at: https://android-developers.googleblog.com/2017/08/arcore-augmented-reality-at-android.html [Accessed 2021].

Cirulis, A. & Brigmanis, K.B., 2013. 3D Outdoor Augmented Reality for Architecture and Urban Planning. *Procedia Computer Science*, 25, pp.71–79. [online] Available at: https://www.sciencedirect.com/science/article/pii/S1877050913012143 [Accessed September 2020].

Cook, J.D., 2019. Converting color to grayscale. *Applied Mathematics Consulting*. Available at: https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/#:~:text=The%20luminosity%20method%20is%20a,%2B%200.72%20G%20%2B%200.07%20B. [Accessed 2021].

Cooper, T., 2018. Introduction to Shader Graph: Build your shaders with a visual editor. *Unity Technologies Blog*. Available at: https://blogs.unity3d.com/2018/02/27/introduction-to-shader-graph-build-your-shaders-with-a-visual-editor/ [Accessed 2021].

Coors, V. and Wiedmann, B., 1998. Using Wearable GIS in outdoor applications. Available at: http://papers.cumincad.org/data/works/att/624d.content.pdf [Accessed 2021].

Dias, N. et al., 2018. Critical success factors of a bottom up urban design process to deliver sustainable urban designs. *International Journal of Strategic Property Management*, 22(4), pp.265–277. Available at: https://journals.vgtu.lt/index.php/IJSPM/article/view/3683 [Accessed 2021].

Feiner, S. et al., 1997. A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment. *Personal Technologies*, 1, pp.208–217. Available at: https://sites.cs.ucsb.edu/~holl/pubs/feiner-1997-perstech.pdf [Accessed 2021].

Fortes, D., 2019. Unity AR+GPS Location. *Unity AR+GPS Location | Unity AR+GPS Location Docs (v3.0+)*. Available at: https://docs.unity-ar-gps-location.com/ [Accessed September 2020].

Google ARCore Developers Team, 2020. *ARCore supported devices*. Google. Available at: https://developers.google.com/ar/discover/supported-devices [Accessed September 2020].

Google ARCore, 2019. Build new augmented reality experiences that seamlessly blend the digital and physical worlds. *Google*. Available at: https://developers.google.com/ar [Accessed September 28, 2020].

Google ARCore, 2021. Choose your development environment | ARCore. *Google Developers*. Available at: https://developers.google.com/ar/develop [Accessed February 16, 2021].

Gröger, G. et al., 2012. OGC City Geography Markup Language (CityGML) Encoding Standard. *Open Geospatial Consortium*. Available at: https://portal.ogc.org/files/?artifact_id=47842 [Accessed 2021].

Hocking, J., 2020. Introduction to Shaders in Unity. *raywenderlich.com*. Available at: https://www.raywenderlich.com/5671826-introduction-to-shaders-in-unity#toc-anchor-003 [Accessed 2021].

Imottesjo, H. et al., 2020. Iterative Prototyping of Urban CoBuilder: Tracking Methods and User Interface of an Outdoor Mobile Augmented Reality Tool for Co-Designing. *Multimodal Technologies and Interaction*, 4(2). Available at: https://www.mdpi.com/2414-4088/4/2/26 [Accessed 2021].

Kaiser, S., 2020. Tracking Technologies in AR. *Onirix*. Available at: https://www.onirix.com/image-tracking-vs-world-tracking-in-ar/ [Accessed 2021].

Kato, H. & Billinghurst, M., 1999. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. *University of Washington*. Available at: http://www.hitl.washington.edu/artoolkit/Papers/IWAR99.kato.pdf [Accessed 2021].

Lamb, P., 2003. ARToolKit. *ARToolKit Home Page*. Available at: http://www.hitl.washington.edu/artoolkit/ [Accessed February 16, 2021].

Lee, G.A. et al., 2012. CityViewAR: A mobile outdoor AR application for city visualization. *2012 IEEE International Symposium on Mixed and Augmented Reality - Arts, Media, and Humanities (ISMAR-AMH)*. [online] Available at: https://ir.canterbury.ac.nz/bitstream/handle/10092/8693/12642991_ismar2012-lee-cityviewar%20(2).pdf;jsessionid=E7E13BF0CE577682EACA478FCEE2B997?sequence=1 [Accessed September 2020].

Leung, H.C., 2020. UrbanStation - Redesigning Pocket Parks in Hong Kong. Master's thesis. Hochschule für Wirtschaft und Umwelt Nürtingen-Geislingen.

Loghin, 2019. tuxalin/water-shader. *GitHub*. Available at: https://github.com/tuxalin/water-shader [Accessed February 16, 2021].

Lypchenko, S., 2019. How To Build A Location-Based AR Application. *ARPost*. Available at: https://arpost.co/2019/10/29/how-to-build-a-location-based-ar-application/ [Accessed September 2020].

Malishev, A., 2014. AUGMENTED REALITY CELOSCAPE. *antonmalishev.com*. Available at: https://www.antonmalishev.com/portfolio/augmented-reality-celoscape/ [Accessed September 2020].

Nijhuis, S., 2016. Applications of GIS in landscape design research. *Research in Urbanism Series*, 4, pp.43-56. Available at: https://core.ac.uk/download/pdf/268414024.pdf [Accessed September 2020].

O'Dea, P., 2020. Number of smartphone users worldwide from 2016 to 2021. *Statista*. Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Accessed 2021].

Piekarski, W., 2017. Tinmith Augmented Reality Project: Home Page. *Tinmith Augmented Reality Project*. Available at: http://www.tinmith.net/index.htm [Accessed February 16, 2021].

Pingale, T., Damugade, S. & Jirge, N., 2017. Visual Communication in Architecture. *International Journal of Engineering Research and Technology*, 10(1). Available at: https://www.ripublication.com/irph/ijert_spl17/ijertv10n1spl_20.pdf [Accessed September 2020].

Qiao, X. et al., 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. *Proceedings of the IEEE*, 107(4). Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8643424 [Accessed 2021].

Rohan, A., Rabah, M. & Kim, S.-H., 2019. Convolutional Neural Network-Based Real-Time Object Detection and Tracking for Parrot AR Drone 2. *IEEE Access*, 7. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8723347 [Accessed 2021].

Sahu, C.K., Young, C. & Rai, R., 2018. Artificial intelligence (AI) in augmented reality (AR)- assisted manufacturing applications: a review. *International Journal of Production Research*. Available at: https://doi.org/10.1080/00207543.2020.1859636 [Accessed 2021].

Smith, A., Dodge, M. & Doyle, S., 1998. Visual communication in urban planning and urban design, *London: University College London, Centre for Advanced Spatial Analysis (CASA)*. Available at: https://discovery.ucl.ac.uk/id/eprint/220/1/urbanplan.pdf [Accessed September 2020].

Structure Studios, 2018. YARD - Augmented Reality Designer - From Structure Studios. *Structure Studios*. Available at: https://www.structurestudios.com/yard [Accessed September 2020].

Sukhmani, S., 2018. Edge Caching and Computing in 5G for Mobile AR/VR and Tactile Internet. *IEEE MultiMedia*, 26(1), pp.21–30. Available at: https://par.nsf.gov/servlets/purl/10125893 [Accessed 2021].

Turner, T., 2017. Ian McHarg's legacy to landscape architecture & landscape urbanism. *Landscape Architects LAA*. Available at: http://www.landscapearchitecture.org.uk/ian-mchargs-legacy-landscape-architecture-landscape-urbanism/ [Accessed September 2020].

Unity Technologies, 2020. *Unity*. Available at: https://unity.com/ [Accessed September, 2020].

Unity, 2021. About AR Foundation: AR Foundation: 4.1.5. *AR Foundation*. Available at: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html [Accessed 2021].

Unity, 2021. Meshes, materials, shaders and textures. Available at: https://docs.unity3d.com/Manual/Shaders.html [Accessed February 16, 2021].

Vuforia, 2021. Getting Started. *VuforiaLibrary*. Available at: https://library.vuforia.com/ [Accessed 2021].

Wikitude, 2021. Wikitude Augmented Reality: the World's Leading Cross-Platform AR SDK. *Wikitude*. Available at: https://www.wikitude.com/ [Accessed 2021].

Yuan, P., 2019. Design Guidelines for Mobile Augmented Reality Reconstruction. Master's Thesis. Technical University of Munich. Available at: https://cartographymaster.eu/wp-content/theses/2019_Yuan_Thesis.pdf [Accessed 2021].

Zucconi, A., 2020. A gentle introduction to shaders in Unity - Shader tutorial. *Alan Zucconi*. Available at: http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/ [Accessed 2021].