Department of
Media Computing and
Human-Computer
Interaction

RWTH AACHEN
UNIVERSITY

FLAPP: Bringing Augmented Reality to Mobile Tourist Guide Systems

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Jan Kaßel

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Kuhlen

Registration date: 03/10/2016
Submission date: 06/17/2016

Eidesstattliche Versicherung

Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum	Unterschrift

Contents

List of Figures

List of Code Listings

List of Tables

Abstract

Augmented Reality is a technology which superimposes digital objects over one's perception of the real world. Thus, captivating user experiences can be realized, where digital artifacts can interact with real-world objects. Today's smartphones are capable of computing such experiences, due to their processing power and sensory equipment. By providing capabilities for communication and navigation, these devices are useful when traveling abroad. Furthermore, previous research showed that users enjoyed Augmented Reality applications in the context of tourism, as they can interact with historical artifacts and heritage sites in an intriguing way. In this thesis, we present the design, development, and documentation of a framework for the Apple iOS mobile operating system. This framework uses commercial technology to realize an Augmented Reality experience. During conception, we evaluated several tracking approaches, 3D rendering methods, and third-party Augmented Reality frameworks. In a post hoc evaluation, we measured the framework's rendering performance on various devices. Although 3D rendering was stable, the results of this evaluation showed that there is a possible need for optimization.

Überblick

Augmented Reality ist eine Technologie, die digitale Inhalte über die Wahrnehmung der realen Welt legt. Damit können fesselnde Erlebnisse umgesetzt werden, in denen digitale Objekte mit solchen aus der Realität interagieren. Heutige Smartphones sind dank ihrer Rechenleistung und Sensoren in der Lage, solche Erlebnisse umzusetzen. Indem sie nützliche Fähigkeiten für Kommunikation und Navigation bieten, sind Smartphones nutzvolle Begleiter auf Reisen. Desweiteren hatten, wie vorherige Forschung aufzeigt, Nutzer Gefallen an Anwendungen von Augmented Reality: Im Tourismus werden neue Interaktionen mit historischen Gegenständen und kulturellen Stätten möglich. In dieser Arbeit stellen wir das Konzept, die Implementierung und die Dokumentation eines Frameworks für das iOS-Betriebssystem von Apple vor. Dieses Framework verwendet kommerzielle Software, um eine Anwendung mit Augmented Reality umzusetzen. Während der Konzeptionierung wurden mehrere Tracking-Verfahren, 3D-Rendering-Methoden und Augmented Reality Frameworks Dritter untersucht. In einer abschließenden Auswertung wurde die Rendering-Leistung des Frameworks auf verschiedenen Geräten getestet. Obwohl das 3D-Rendering stabil war, zeigten die Ergebnisse der Auswertung einen möglichen Bedarf nach Optimierung auf.

Acknowledgements

First, I want to thank my supervisor, Philipp Wacker, M. Sc., for his valuable advice, time and input during the work on this thesis. I thank him and Prof. Dr. Jan Borchers for the possibility to write this thesis at i10.

I thank this thesis' examiners, Prof. Dr. Jan Borchers and Prof. Dr. Torsten Kuhlen, for their patience and time.

I thank Akira Iwaya from Japan for translating his article "SceneKit + VuforiaでARアプリを作ろう" [Iwaya, 2016] upon our request from Japanese to English. The solution he proposed—as discussed in section 3.2.3—was a major contribution to the technical implementation of our work.

I thank my family for their ongoing support of my studies, and my dear friends for their helpful aid and support during the work on this thesis.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English. We use the plural form for the first person.

Chapter 1

Introduction

Recent years have brought remarkable success in research of Mixed Reality, a conception where reality and virtual surroundings are merged gradually [Azuma, 1997]. This continuum includes the concepts of *Virtual Reality*, where one perceives purely virtual content, as well as *Augmented Reality*, where reality is augmented by virtual content in one's perception [Milgram and Kishino, 1994]. Augmented Reality holds promising experiences while providing new interaction techniques between the real world and virtual artifacts [Wellner et al., 1993].

Augmented Reality merges reality with digital content

AUGMENTED REALITY:
While perceiving in Augmented Reality, "virtual objects [are] superimposed upon or composited with the real world" [Azuma, 1997]. In contrast to *Virtual* Reality, real and virtual objects exist in the same sphere, while Virtual Reality tends to focus on virtual content (figure 1.1).

Definition:
Augmented Reality

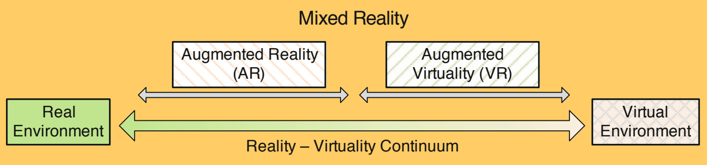


Figure 1.1: The Mixed Reality continuum [Olsson et al., 2013]

Handheld computing devices, such as smartphones, are enjoying enormous popularity [Falaki et al., 2010]. These devices are equipped with various sensors, some of which facilitate the development of context-aware applications [Lane et al., 2010, Olsson et al., 2013]. Thus, the technical support for deploying an Augmented Reality system is quite extensive.

FLApp will bring a modern touristing experience for the City of Aachen

Future Lab Aachen[1] is an initiative which is organized by the City of Aachen. The project aims to present scientific research. Furthermore, it is intended to serve as a cultural institution, promoting university-related affairs to businesses, tourists, as well as inhabitants of Aachen. The *Future Lab Aachen App* (hereinafter designated as FLApp) is a mobile application which will accompany this effort. The application will provide rich media content and information about pivotal places on the go [Future Lab Aachen Website].

FLApp will also include an Augmented Reality feature, which shall provide unique experiences alongside traditional digital media. This feature will complement digital information in form of text and pictures. Augmented Reality brings a unique user experience [Olsson et al., 2013]. By building upon this, we explore a new way of interacting with local research and the city's history.

We provide a survey of Augmented Reality frameworks and developed an Augmented Reality feature

The goal of this thesis is to provide an overview of existing Augmented Reality frameworks and to create an easy to use framework for the development of FLApp, called FLAppAR. To realize this, we performed a small survey of various Augmented Reality frameworks. Furthermore, we investigated suitable tracking methods for our particular Augmented Reality application. Therefore, we compared a selection of Augmented Reality frameworks performance-wise and designed a system that is capable of rendering interactive 3D content in an Augmented Reality context. This system is powered by Vuforia[2] on devices that run the Apple iOS mobile operating system.

Subsequently, we implemented the system we designed beforehand, taking our conceptions of 3D rendering and

[1]http://www.futurelab-aachen.de
[2]https://vuforia.com

Augmented Reality tracking into account. We created a standalone framework that exposes an Application Programming Interface which enables developers to use this framework in an uncomplicated manner. We provide documentation and guidelines on how to work with the system for both application developers as well as designers.

In chapter 2 "Related Work" we review literature and software that is related to our work. This covers fundamental research of Augmented Reality as well as rather specific topics—for example, deployment of Augmented Reality experiences in tourism as well as user experience design for Augmented Reality.

Our own work is discussed in chapter 3, which is divided into three sections: After outlining the Augmented Reality system design in section 3.1 "Designing the Augmented Reality System", we describe its implementation in the subsequent section, 3.2 "Implementing the Application Design". This implementation is evaluated briefly in the third section, 3.3 "Evaluating the System Performance".

A detailed explanation on how to use the system is provided in chapter 4 "Guidelines for Developers and Designers", to assist both developers and designers through the process of setting up an Augmented Reality experience using the system presented here.

In the fifth chapter, "Summary and Future Work", we conclude our research and implementation. We give an outlook on how the system is going to be used, as well as possible improvements for the technical implementation.

Chapter 2

Related Work

Augmented Reality is a field with over 25 years of active research: Over time, that branch developed into a state of the art research field with applications in many areas, such as medicine, communication and military [Azuma, 1997]. Fundamental research is discussed in section 2.1. Subsequently, research on the capabilitiy of Augmented Reality to present contextual information is outlined in section 2.2, while section 2.3 will provide an overview on deployment of Augmented Reality in tourism. Section 2.4 deals with studies on User Experience Design (UX) in Augmented Reality systems. The two latter sections, 2.5 "Augmented Reality Frameworks" and 2.6 "Related Software", will cover software which is related to this thesis and Augmented Reality specificially.

2.1 Fundamental Research in the Field of Augmented Reality

Head-Mounted Displays (HMD) enabled early research on changing one's visual perception by presenting arbitrary three-dimensional information [Sutherland, 1968]. Fundamental research projects performed first Augmented Reality experiments using such devices in the early 1980s [Azuma, 1999]. The survey curated by Azuma [1997] examines such

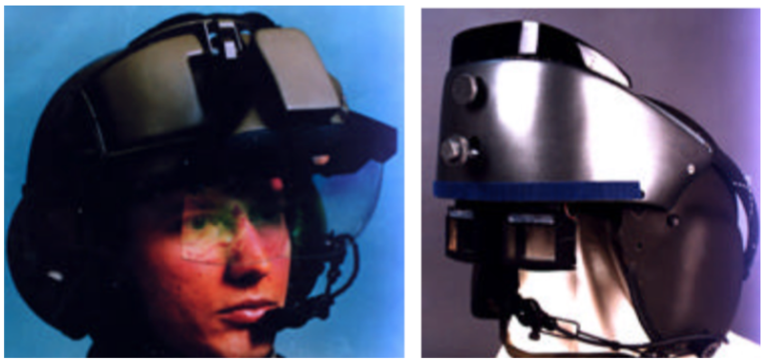


Figure 2.1: Head-Mounted Displays (HMD) have already been used in early Augmented Reality research [Azuma, 1997].

research and represents one of the major research achievements in early history of Augmented Reality. Not only is the terminology of Augmented Reality itself determined: Furthermore, Azuma provides a detailed overview of early Augmented Reality research, hardware and theoretical foundations. Issues with tracking stability and the dialog of the place of Augmented Reality in the Mixed Reality continuum are being discussed.

In follow-up research, Azuma approached the issues Augmented Reality exposed when used outdoors [Azuma, 1999]. Contemporary tracking approaches such as Electronic Compass and GPS yield errors in their exactness, especially in uncontrolled outdoor environments. Azuma discusses solutions to these inaccuracy issues, e.g. a *hybrid tracking* approach that incorporates several tracking methods to overcome their individual weaknesses.

Kato and Billinghurst [1999] proposed a system that was capable of tracking special marker images—opposed to tracking landmark images of the surroundings—to realize an Augmented Reality conferencing system. With this visual approach, augmenting content is associated with these marker images, and not reference images of the real world. By using a Head-Mounted Display for their evaluation, Kato and Billinghurst found that marker images may provide good tracking results, but their tracking method lacked ro-

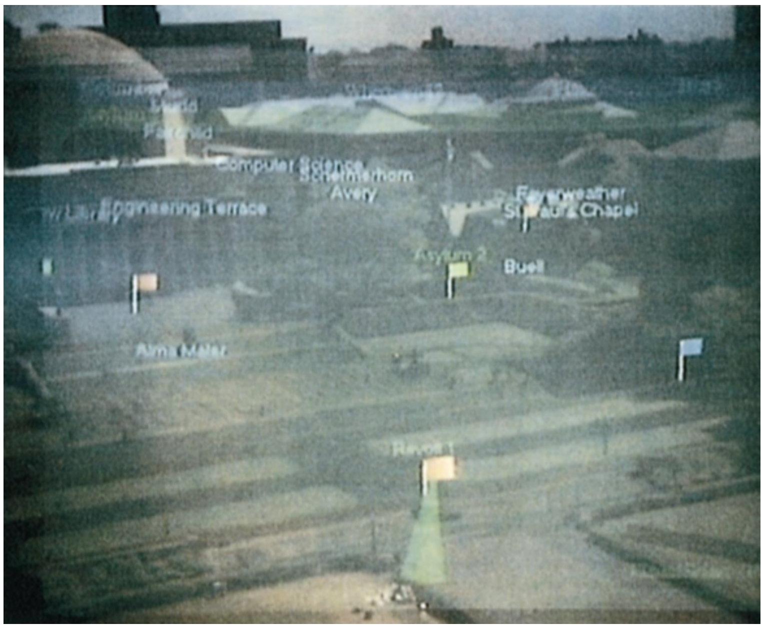


Figure 2.2: View through a Head-Mounted Display with the MARS system [Höllerer et al., 1999].

bustness.

2.2 Contextual Information and Augmented Reality

In recent research, several attempts have been made to show the capability of Augmented Reality to provide useful contextual information to users. In their work, Feiner et al. [1997] seeked to provide such context by developing an application that presented contextual information about their university campus. While using a Head-Mounted Display and a handheld device, the system's tracking capabilities, however, have been prone to error due to computational limitations [Feiner et al., 1997].

Also focusing on contextuality, Höllerer et al. [1999] propose a system that presents information that is "spatially registered [in] the real world" [Höllerer et al., 1999]. By using mobile computing devices, a Head-Mounted Display, users can annotate indoor and outdoor locations (figure 2.2).

Bergig et al. [2010], too, studied the concept of providing contextual information with Augmented Reality. For their approach, additional digital information about a real-world object is encoded in this object's appearance. The object then offers both its physical and digital information, of which the latter can be obtained by using an Augmented Reality system. By adapting existing computer vision methods, information about the augmented 2.5d data is extracted from the graphical target—objects in 2.5d are not represented fully three-dimensional, but rather use mimicking methods to appear so. A study found that users performed better on tasks regarding the information from such a target by using the additional Augmented Reality information [Bergig et al., 2010].

Ajanki et al. [2011] discovered that contextual information, that is provided by an Augmented Reality interface, can prove useful when performing certain tasks. They developed a system that runs on both a Head-Mounted Display and handheld devices. When using the system, the user will be presented contextual information as soon as certain objects or persons are recognized. The system performed well in a user study, where participants found use in the additional virtual information while performing research-related tasks [Ajanki et al., 2011].

2.3 Augmented Reality in Tourism

The use of Augmented Reality as a possibility to change the tourist experience has already been proposed by Azuma [1999]. While Virtual Reality gained a lot of popularity, as Azuma states, Augmented Reality was prone to error because of tracking issues. Due to the difficulty of visual tracking in uncontrolled environments, such as outdoor environments as opposed to controlled indoor settings, the deployment of Augmented Reality in outdoor settings has always been challenging. Achieving more stable optical trackers and all-around Augmented Reality systems would

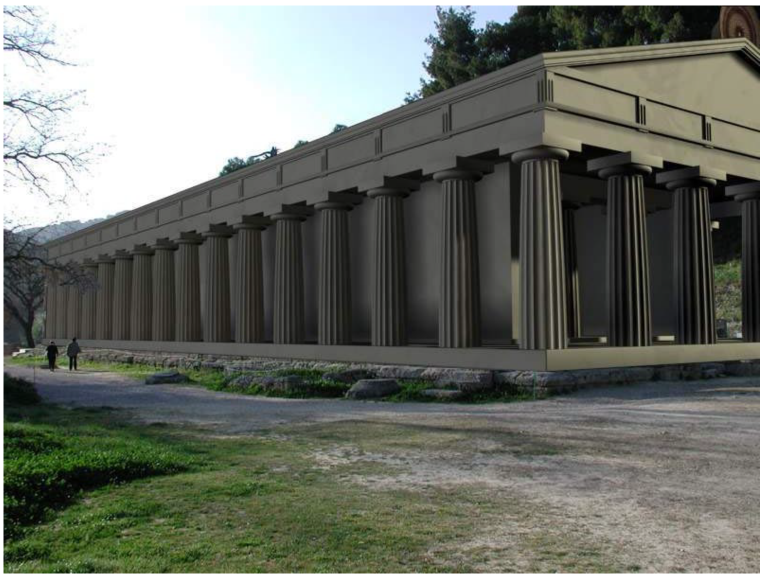


Figure 2.3: Augmentation of 3D models of ancient greek buildings [Vlahakis et al., 2001].

benefit visitors of heritage sites, where unique experiences could be gained with Augmented Reality [Azuma, 1999].

In their research, Fritz et al. [2005] found out that while most contemporary Augmented Reality systems had been in an academic stage, tourists could gain a lot of additional information by Augmented Reality systems. Especially at cultural and heritage sites, artifacts such as historical buildings could be reconstructed. Visitors of these sites often yield particular interest in their sourroundings which can be complemented by gaining views of their initial outlooks [Fritz et al., 2005]. Kounavis et al. [2012], too, encourage the use of Augmented Reality in tourism. While such systems can provide useful valuable information to tourists, presenting digital content in an interactive manner can attract a wider audience, as they state [Kounavis et al., 2012]. Kounavis et al. furthermore provide an overview of Augmented Reality software, as well as list up several deployments of Augmented Reality systems in tourism and cultural institutions like, for example, museums.

The ARCHEOGUIDE project pursued research on a system that can augment 3D objects, video and audio [Gleue and Dähne, 2001, Vlahakis et al., 2002]. While some Augmented Reality systems performed well with the use of digital markers, ARCHEOGUIDE was intended to work markerless. To overcome the limitations of these marker, researchers developed an image recognition system that performs feature detection on reference images [Stricker and Kettenbach, 2001]. An evaluation at Olympia, Greece showed that the participants were capable of using the system and were interested in exploring other cultural heritage sites by using Augmented Reality [Vlahakis et al., 2001].

In a follow-up research project to ARCHEOGUIDE, the LIFEPLUS project has been introduced as a modern Augmented Reality experience at the site of ancient Pompeii [Papagiannakis et al., 2002, Vlahakis et al., 2003, Papagiannakis et al., 2005]. With the goal of "push[ing] the limits of curent Augmented Reality (AR) technologies" [Papagiannakis et al., 2002], an approach with rich features in 3D augmentation as well as audiovisual content has been developed and implemented. Real-time camera tracking has been conducted to estimate the user's position, so 3D content can be positioned within the real-world scene accurately. The concept does not only include the augmentation of ancient buildings as part of an Augmented Reality guide of the site: Detailed models and animations of human beings have been developed to recreate ancient life in an Augmented Reality storytelling experience [Papagiannakis et al., 2005].

2.4 User Experience Design in Augmented Reality

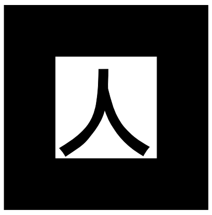
Augmented Reality, as an interface between physical and virtual artifacts, can bring new interaction designs and unforeseen experiences. To better understand Augmented Reality as medium, Macintyre et al. [2001] considered it as novelty to media theory. They investigated on how to deploy narrative Augmented Reality experiences, such as movie-like formats. While characterizing it as a new medium, three main features have been outlined: The blending of virtual and physical space, users' control over the point of view, and interactivity of Augmented Reality experiences [Macintyre et al., 2001]. Furthermore, Macintyre et al. pointed out that due to the user's interaction with his surroundings, even non-interactive digital content yields an interactive experience in Augmented Reality.

In a study to gain insight about expectations and requirements of users of Augmented Reality systems, Olsson et al. [2013] found that Mobile Augmented Reality (MAR) yields "great potential of facilitating positive user experience and emotions" [Olsson et al., 2013]. The study negotiated various expectancies of users to Augmented Reality services, such as captivation, liveliness, and playfulness [Olsson et al., 2013]. After analyzing these terms, Olsson et al. derived a collection of such expectations, garnished with summarized descriptions.

Han et al. [2013] approached a similar study, by investigating into expectations and requirements of possible users of Augmented Reality systems. To gather insight on how to realize a tourism Augmented Reality system in the city of Dublin, a study with tourists has been conducted. The interviews showed that Augmented Reality is asked to serve as interface to local information as well as to data from social networks. Also, factors of usability, such as the application's responsiveness and multi-language features have been identified to be important [Han et al., 2013].

2.5 Augmented Reality Frameworks

Most Augmented Reality research from the 1990s and early 2000s has been realized with special hardware, such as mobile PCs and Head-Mounted Displays, e.g. Feiner et al. [1997], Billinghurst et al. [2001], Gleue and Dähne [2001]. These systems often were running proprietary software that suited just the particular setup. With the popularity of mobile computing devices, device-agnostic software for Augmented Reality systems became evident and made deployment of Augmented Reality experiences much more prevalent.


Kanji marker proposed by ARToolKit

Emerged from research by Kato and Billinghurst [1999], Kato et al. [2000], ARToolKit[1] is an open source Augmented Reality Software Development Kit. The software is available, among others, for iOS and Android mobile operating systems. Besides tracking of traditional imagery, ARToolKit is capable of tracking specifically designed markers that are similar to barcode markers and QR codes. These images yield special properties, such as square and fixed size [Kato and Billinghurst, 1999], for being able to recognize them reliably by computer vision systems.

Vuforia[2] and Wikitude[3] are commercial contenders in the Augmented Reality market. Both offer similar features, Image Recognition and Natural Feature Tracking (NFT) among them, and are offered for iOS and Android operating systems as well. Additionally to conventional 2D tracking, both frameworks feature 3D object recognition and can be powered by Internet-based services.

2.6 Related Software

OpenGL[4] (Open Graphics Library) is an open source graphics library that is supported on various different devices, ranging from embedded systems to desktop computers. Its descendant for embedded computing, OpenGL for Embedded Systems (OpenGL ES), is supported by both Apple iOS as well as Android mobile operating systems.

On iOS, SceneKit[5] is a framework with a high-level 3D graphics API. Development of 3D scenes is streamlined due to a simplified API design, while a lot of functionality of Open GL ES is maintained, such as shaders, materials, and lighting. Due to its compatibility with OpenGL, SceneKit content can seamlessly be rendered into OpenGL ES contexts. Furthermore, SceneKit is capable of importing 3D

[1]http://www.artoolkit.org
[2]http://www.vuforia.com
[3]http://www.wikitude.com
[4]https://www.opengl.org
[5]https://developer.apple.com/scenekit/

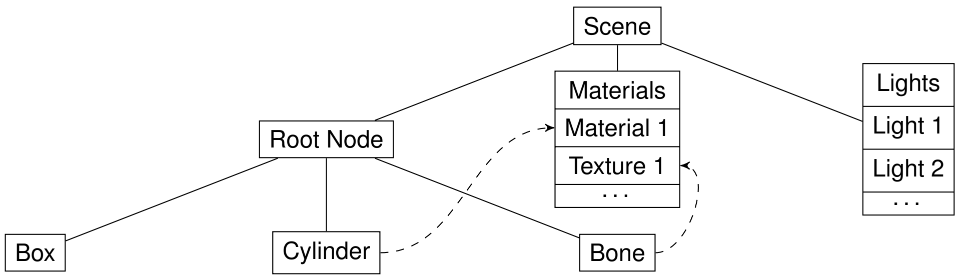


Figure 2.4: Possible structure of a Collada model; A scene contains, for example, nodes, materials, and lights. Nodes, too, are structured as tree, and each node may contain meshes and references to other properties, such as textures [Collada 1.4.1 Specification].

scenes in the Collada[6] format—an open source, XML-based file format for storing 3D scenes. In Collada, properties of 3D scenes, such as materials, nodes and lights, are structured in a tree-like manner (figure 2.4).

JSON[7] is a format for data interchange that is readable and writable easily for humans as well as machines. Its support in a lot of programming languages and systems facilitates data interchange on different platforms. Structure of JSON documents can be specified by JSON Schemata[8], which empower the possibility of validating such documents. KiteJSONValidator[9] is an open source library for Objective-C that provides functionality for such validation.

A popular system for documenting C++ source code, among others, is Doxygen[10]. For Objective-C, appledoc[11] provides similar functionality while adding further support for embedding generated documentation of source code into Apple's Xcode software.

[6]https://www.khronos.org/collada/
[7]http://www.json.org
[8]http://json-schema.org
[9]https://github.com/samskiter/KiteJSONValidator
[10]http://www.doxygen.org
[11]https://github.com/tomaz/appledoc

Chapter 3

Developing an Augmented Reality Application

The work we faced during this thesis consists of three major tasks. First, analyzing the Future Lab Aachen App project requirements and designing an Augmented Reality system that runs efficiently on the Apple iOS mobile operating system. Secondly, implementing this design in such a way that it can be deployed on various Apple iPhone devices and evaluate its performance. And third, providing documentation to both designers and developers that will work with this system on how to use it and how to tune the settings to achieve the system's optimal performance.

We had to design and implement an Augmented Reality application

This chapter discusses the first two tasks, *Design* and *Implementation* of the system. In section 3.1 "Designing the Augmented Reality System", we outline the design challenge we faced. We present our decisions regarding the selection of a suitable tracking method as well as an Augmented Reality framework that is supported on iOS. Following that section, section 3.2 "Implementing the Application Design" gives a detailed description of the implementation of the design that was pictured beforehand. We illustrate the software design and its workflow, and highlight crucial components. In section 3.3 "Evaluating the System Performance", the performance of the software is evaluted and discussed.

3.1 Designing the Augmented Reality System

Future Lab Aachen App is intended as mobile tourist guide application

The Future Lab Aachen App, short FLApp, is intended as a mobile companion to the Future Lab Aachen[1] project—an initiative that aims to gain interest for local research projects in the city of Aachen. The FLApp mobile application will offer information to several POIs (Points of Interest) in the city center, such as Aachen cathedral and buildings of university chairs [Future Lab Aachen Website].

Regarding the Augmented Reality feature, several requirements were given:

Display 3D content The system should be capable of displaying 3D objects in an Augmented Reality context.

Pursue a cross platform approach Since an application for the Android mobile operating system with the same functionality has been scheduled, porting the System to Android should be as easy as possible. Hence, most of the major software libraries used should be available cross platform.

Make it run on a range of iPhone devices Instead of optimizing it for the newest generation of Apple iPhone devices, the software should be available for Apple iPhone 4S and newer.

Provide stable tracking Image targets should be recognized reliably by the Augmented Reality system, and the estimated pose of the augmenting 3D object must be stable.

Recognize frame-like target images Originally, the system was intended to be able to recognize frame-like targets, i.e. images with a cut-out center. Augmenting content

[1]http://www.futurelab-aachen.de

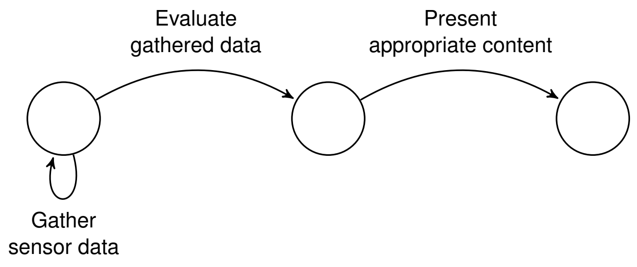


Figure 3.1: Augmented Reality cycle.

had to be displayed within that frame, while through the frame, a certain landmark was visible. Therefore, a certain distance will be between the user's device and the target. *This requirement has been considered in the design stage, but was dropped on a further iteration of the project. It has been considered within the Design stage nevertheless, see section 3.1.3 "Recognizing Frame-Like Targets".*

We began by analyzing how Augmented Reality actually works. It depicts a relatively simple cycle, as illustrated in figure 3.1. Sensor data—for example, GPS position or camera input—, that is continuously gathered will be analyzed subsequently. Based on the results of the analysis, appropriate digital content will be presented.

Derived from that cycle, there are three components to be discussed. In the following, we outline several ways of sensor input for Augmented Reality. Section 3.1.2 will explain our choice and evaluation of different Augmented Reality frameworks, that bring Computer Vision components for evaluating device sensor data. The presentation of 3D content, as the last component, as required, is discussed in section 3.1.4.

We had to discuss sensor input, computer vision and presentation of 3D content

3.1.1 Comparing Augmented Reality Input Methods

While learning about the technical specifications of the devices we targeted, namely Apple iPhone 4S and newer, we gathered data about the device's sensor and video specifications [iPhone 4S Technical Specifications]. Intuitively, we gathered several of the device's hardware components and distinguished them into two categories:

The targeted device type contains suffient hardware to perform tracking for Augmented Reality

Visual sensors The iPhone's camera has 8-megapixel resolution and supports video streaming in Full HD (1920×1080 pixels) resolution with up to 30 frames per second.

Non-Visual sensors The iPhone contains several sensors, namely assisted GPS, a digital compass, a three-axis gyroscope, and an accelerometer.

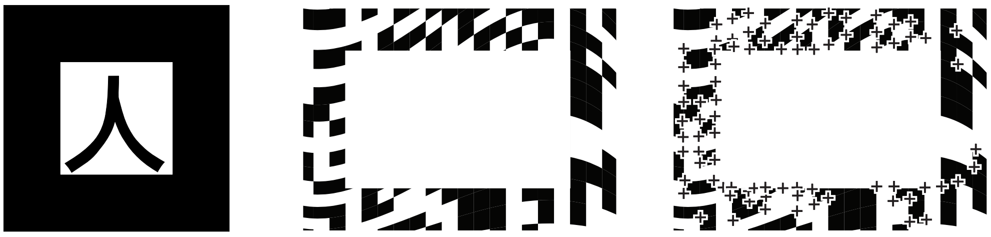
To realize in-place rendering of 3D objects robust tracking is a necessary requirement. Sensors often bring inaccuracies, for example, GPS due to connection failures and occlusions as well as electronic compasses due to noise in the magnetic field [Azuma, 1999]. Therefore, we tended to a visual approach, by using live camera input. Feature recognition will then be done by Computer Vision functionality. Yet, we still considered a *hybrid approach* which has been proposed by Azuma [1999], which embodies a combination of multiple sensors for tracking.

We settled with a visual approach for tracking

Visual tracking comes in two types: With *Marker Tracking* on the one hand, recognition is done by specifying certain properties a trackable image has to include—these may be, for example, color, shape and sizes. While these properties restrict the trackable image in its design, such tracking systems may perform reliably due to their optimization for these particular types of images. A sample marker image is illustrated in figure 3.2 (a).

Natural Feature Tracking gives more artistic freedom than Marker Tracking

Natural Feature Tracking (NFT) on the other hand leaves more freedom regarding the choice of the trackable imagery to



(a) Kanji marker (b) Sample target image (c) Image features

Figure 3.2: Figure 3.2 (a) shows a marker image by ARToolKit. Figure 3.2 (b) depicts a sample image intended for Natural Feature Tracking, and figure 3.2 (c) shows a recreated analysis of features for Natural Feature Tracking by Vuforia Target Manager. "+" depicts one or more features.

the designer. Images that are trackable by Natural Feature Tracking are analyzed by certain *features* in their appearance, such as corners and edges. The choice of imagery, however, is limited by its trackability: distribution and amount of such features may result in worse or better trackability. A sample distribution of features on an image is depicted in figure 3.2 (b).

3.1.2 Evaluating Augmented Reality Frameworks

From conducting the Wikipedia-like, crowd-sourced comparison platform Social Compare[2], we obtained a selection of frameworks that met our needs: ARToolKit, Wikitude, and Vuforia. To gain further insight in the differences between the frameworks—regarding performance and usability—, we performed a small performance review. This required us to work with each system, learn the sample code, and perform target analysis.

We chose three Augmented Reality frameworks for testing

The frameworks beared the following capabilities:

ARToolKit ARToolKit is a free and open source Augmented Reality SDK. Among many desktop operating systems, the software is also available for the iOS and Android

[2]http://socialcompare.com/en/comparison/augmented-reality-sdks

operating systems. Besides tracking of natural features, ARToolKit features special square-sized marker images. The SDK comes with a selection of tools, notably `genTexData` for creating Natural Feature Tracking datasets [ARToolKit Documentation].

Vuforia Vuforia is a closed-source, commercial contender of ARToolKit. The software, too, is available for iOS and Android, among other platforms, notably Microsoft HoloLens. Various tracking targets are offered, such as text, images, and primitive three-dimensional objects. Recognition of targets can be computed either locally or online, using Vuforia cloud services [Vuforia Developer Library]. Analysis of targets is performed with a web-based tool, Vuforia Target Manager, which performes feature detection and provides download and management of datasets.

Wikitude Similar to Vuforia, Wikitude is a commercial product. Powering recognition of 2D and 3D targets as well as cloud services for target recognition, Wikitude's set of features resembles Vuforia's capabilities. Additional to a native API, a Java-Script powered API enabled developers to create Augmented Reality experiences using web technologies [Wikitude Documentation]. Notably, this web-powered code runs cross platform without modification.

The commercial frameworks provide many tools and good usability

While ARToolKit attracts with being free of charge and open source, commercial options Vuforia and Wikitude tempt with rich ecosystems, such as cloud services. While Vuforia is promoted with high performance [Vuforia Developer Library], the cross platform approach Wikitude provides may ease the development of Augmented Reality experiences on multiple platforms.

Comparison of Image Recognition Features

Since reliable tracking on a distance as well stable rendering was a project requirement, we conducted a small survey to evaluate performances of the frameworks we introduced

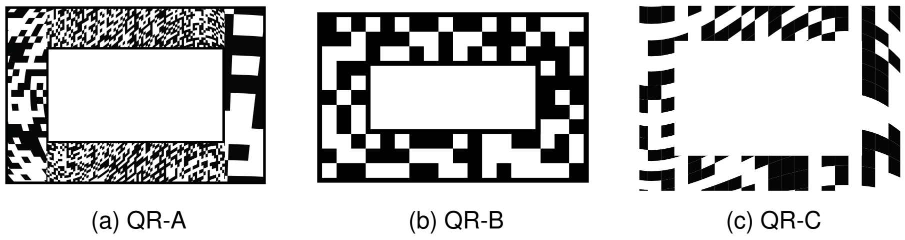


Figure 3.3: Images with a frame-like appearance used for comparison; Figure 3.3 (a) contains many features with high detail and distortion (achieved by bending it perspectively), figure 3.3 (b) consists of a simple geometrical shape with quite even distribution, and figure 3.3 (c) adds bending to uneven this distibution.

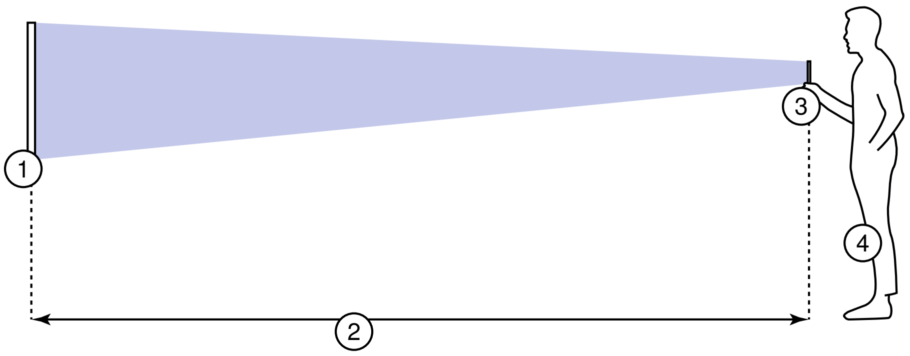


Figure 3.4: Setup of the framework evaluation; The user (4) points his smartphone (3) at the specified target image (1). While keeping the target image in focus, he approaches it from a distance (2) where the system is not capable to recognize the target.

in section 3.1.2. With this survey, we intended to gain an overview of the systems' tracking stability and tracking range. Furthermore, we investigated in setting up tracker imagery and importing such into each framework's Augmented Reality.

We conducted a survey to compare the frameworks' performance

Therefore, we created a collection of image targets, ranging from ARToolKit's *Hiro* and *Kanji* markers to own designs that followed our requirements, such as having a frame-like appearance. Such designs are illustrated in figure 3.3. A list of all target images used is provided in appendix A.

We designed markers specificially for this comparison

We set up a testing environment as depicted in figure 3.4. The target image to be recognized was positioned at eye

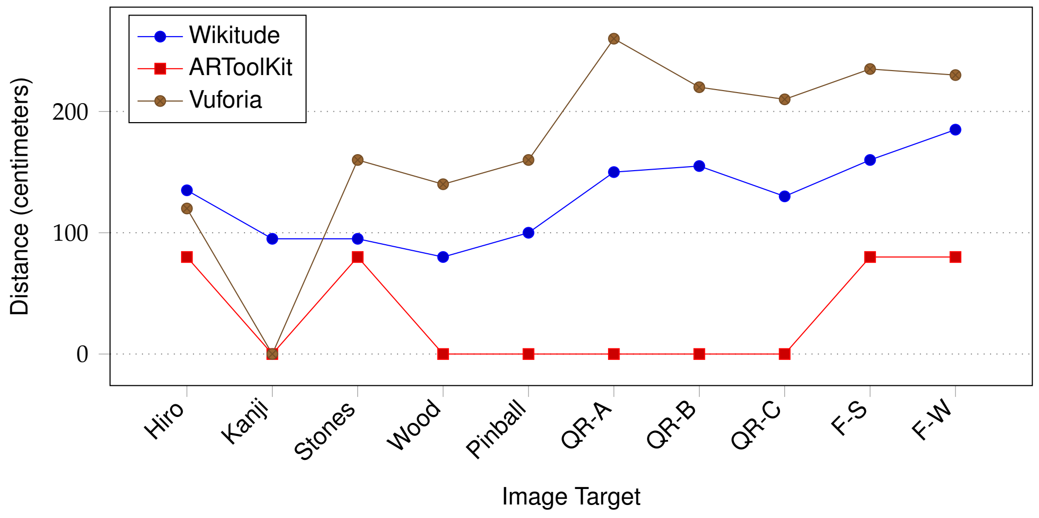


Figure 3.5: Results of the framework comparison; Detailed results are available in table B.1 of appendix B. A distance value of zero centimeters indicates that the device was not capable of recognizing the target, using the respective framework.

level, on, for example, a wall. Standing in front of that image, we started moving towards it. We began moving from a distance from which the current framework to be tested was not capable of recognizing the image from. As soon as the image had been recognized and virtual content appeared, we stopped and conducted our distance measure to note the distance. The results of this test are listed in table B.1. These results are to be considered as estimates with a precision of ±5 centimeters.

Vuforia outperformed Wikitude as well as ARToolKit

As illustrated in figure 3.5, Vuforia and Wikitude both performed well, achieving recognition distances from 1 meter and upwards. ARToolKit, however, was not capable of recognizing the targets properly—in contrast to prior tests with the default marker images, where tracking had been very reliable. Even when the latter frames, *F-S* and *F-W*, had been recognized, the image projection was unstable and flickered. Vuforia, however, exceeded our expectations and was capable of detecting targets from up to 2.5 meters, while rendering of the sample 3D object was reliable and stable. Wikitude achieved good results throughout, but was outperformed by Vuforia.

3.1.3 Recognizing Frame-Like Targets

The project's requirements asked to realize an approach where 3D objects are rendered within the cut-out center of an image target. The window-like target was supposed to keep visual focus on a building located behind the frame, while displaying Augmented Reality context.

We were asked to realize a window-like approach for image targets

Unfortunately, we were not able to find official resources regarding transparency in image targets. However, by testing in Wikitude and Vuforia, we learned that due to the feature recognition system, transparent parts can be easily included. Continuously, the system tracks position and density of features—that are, for example, edges and corners [Vuforia Developer Library]. During tracking, those features of the input image that are not included in the reference image are omitted. The feature distribution of such an image is illustrated in figure 3.2 (c). Hence, supposed transparent areas of a target image can be realized by filling the area with solid color to prevent feature recognition in this particular area.

Blank areas do not contain features and may be considered transparent

3.1.4 Displaying 3D Content

Nowadays, 3D content can be rendered on mobile devices by a range of technologies—one of which is OpenGL[3]. OpenGL is an open source technology which is available on a range of devices. Its descendant for embedded systems, OpenGL for Embedded Systems (OpenGL ES), is provided as the default 3D framework on iOS as well as the Android operating system. Therefore, OpenGL is the fundamental layer of our 3D rendering system.

While Vuforia does not ship a feature of rendering 3D objects itself, this has to be performed in the application. Having the OpenGL ES support on iOS in mind, we phrased several requirements that our 3D rendering system should follow: The system should be capable of rendering the given content in a considerable amount of time, bringing no visual or com-

OpenGL was mandatory to be used with iOS and Vuforia

[3]https://www.opengl.org

putational lags to the system. Integration with iOS' OpenGL context is necessary, since Vuforia feeds the captured camera content via the OpenGL context onto the screen. Lastly, the system should bring no or just little overhead to our setup. It is rather encouraged to simplify the OpenGL workflow by relying on frameworks that apply shaders and further 3D adjustments in a straightforward manner.

We seeked simpler approaches than using plain OpenGL

From this perspective, we did research and conducted several open source 3D rendering frameworks. Simplified, our goal was to find a 3D framework that performs well on iOS, runs within an OpenGL context and simplifies the OpenGL workflow. Finally, we settled for three possible solutions for our design approach:

OpenGL does not provide functionality to import 3D objects

1. Purely relying on OpenGL and do not using any additional frameworks in the rendering system. Since OpenGL ES does not provide any functionality for importing 3D models from various formats, e.g. Collada[4], model import has to be done by oneself. Assimp[5], an open source library which supports import of various 3D formats, Collada among them, has been considered for this task.

2. Relying on a third party 3D framework that is capable of rendering 3D content and importing such content. We considered NinevehGL[6] and the open-source Irrlicht Engine[7], which both run under iOS.

Metal and SceneKit are 3D frameworks provided on iOS

3. Using features that are built-in into the iOS operating system, namely the low-level graphics framework Metal[8] and the high-level 3D framework SceneKit[9]. Both frameworks are developed by Apple and bring less overhead to the application. While Metal is kept low-level to provide good performance in both games and 3D applications, SceneKit is built on top of OpenGL ES as a high level, simplified framework. Still,

[4]https://www.khronos.org/collada/
[5]http://assimp.org
[6]http://nineveh.gl
[7]http://irrlicht.sourceforge.net
[8]https://developer.apple.com/metal/
[9]https://developer.apple.com/scenekit/

> SceneKit provides similar extended functionality to OpenGL ES, such as shaders and physics [Rönnqvist, 2014]. Furthermore, import functionality of Collada models is supported in SceneKit.

Considering these three options, we chose SceneKit as our primary solution, adhering to the third option. Its simple API makes setting up a 3D scene for both us and developers who work with the system a pleasant experience, while features such as animation, physics and scene import are provided [SceneKit Framework Reference]. Since SceneKit is based on OpenGL ES, rendering into an arbitrary OpenGL context can be achieved easily—but unfortunately, further steps have to be done to make SceneKit render its 3D content properly onto a target that has been recognized by Vuforia. These steps are discussed in section 3.2.3.

We settled with a combination of OpenGL and SceneKit

3.2 Implementing the Application Design

The application is implemented using the Objective-C and Objective-C++ programming languages. We used Apple's Xcode Software[10] for development and debugging of the software. Hardware-wise, iPhone 4S, iPhone 5 and iPhone 6S[11] devices have been used for testing the software.

In the following, we will outline the structure of the implementation and the purpose of each component in chapter 3.2.1. Furthermore, we will discuss particular aspects of the implementation: Section 3.2.2 illustrates the system's lifecycle and possible delegation techniques for responding to lifecycle events as well as providing 3D scenery to the system. Section 3.2.3 shortly discusses a programmatical approach to combine Vuforia and SceneKit rendering. In section 3.2.4, we detail configuration capabilities for provided 3D scenery, including an approach to normalize scaling of such scenery automatically. Section 3.2.5 "Improving Rendering Stability" covers approaches to improve the stability

[10]https://developer.apple.com/xcode/
[11]http://www.apple.com/de/iphone/

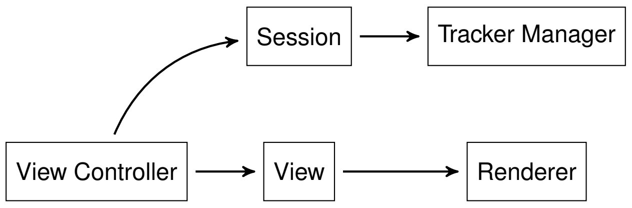


Figure 3.6: Structure of the implementation; arrows highlight dependencies.

of the pose matrices which are estimated by the Vuforia engine.

3.2.1 System Structure

Originally, the system's structure draws several implications from Vuforia sample applications[12]. By iterating over the initial structure, we found a design more suited for our particular approach, where communication with external components by using delegates and rendering are incorporated.

The structure is illustrated in figure 3.6. The view controller serves as entry point to the system, upon whose creation the backend components—session and tracker manager—are initialized. To overcome components' thickness in the initial approach, we separated tracker management and rendering into each their own component.

3.2.2 System Lifecycle

`ARSession` is the system's central component

Major calls to the Vuforia API—such as initialization, launching the camera, and shutdown—are handled by the system's central component, `ARSession`. By responding to the Vuforia API, a certain lifecycle consisting of initialization and pausing is enforced. Figure 3.7 illustrates that lifecycle.

[12]https://developer.vuforia.com/downloads/samples

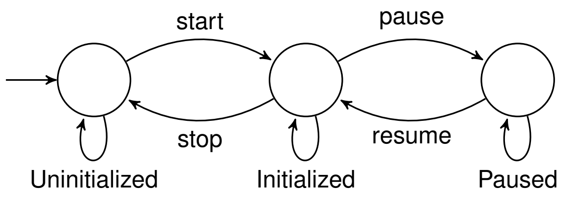


Figure 3.7: Lifecycle of the system

Initially, the Augmented Reality system is not initialized. The *start* transition causes `ARSession` to initialize the Vuforia engine, start the tracker, and load its tracking database. A temporary pause of the system is handled by the *pause* respectively *resume* transitions. The *stop* transition will shut down the system, making Vuforia free all its resources and uninitialize its components.

The system's lifecycle consists of three states

To handle such events, `FLAppAR` exposes several delegates. The lifecycle delegate, which is of type `ARLifecycleDelegate`, consists of methods that are called on each lifecycle transition. In addition to lifecycle delegation, a delegate of type `ARErrorDelegate` can be supplied to catch possible errors and respond accordingly—therefore, several error types with distinct levels of severity are provided. Both delegates can be set via the view controller's `lifecycleDelegate` respectively `errorDelegate` properties. All delegates are discussed more detailed in section 4.1.6.

The system communicates its lifecycle through delegates

To ensure a seamless execution within `FLAppAR`, all lifecycle and error delegate methods are called in a background thread. It is noteworthy that updates to the user interface have to be dispatched from the main thread, for example by using Grand Central Dispatch [iOS Developer Library].

Scene Delegation

In addition to lifecycle and error delegation, `FLAppAR` delegates requests for scene content to a given scene delegate. This design empowers developers to handle scene allocation and release independently of the Augmented Reality system,

3D objects are supplied by a scene delegate for example, by relying on GPS data or iBeacon[13] technology. Other than lifecycle or error delegates, a scene delegate is required for the system to work—therefore, `ARViewController` holds a strong reference to it.

The `ARSceneDelegate` protocol consists of one method, `provideSceneForTarget:`. As soon as the Renderer detects a new target, the Scene Delegate is asked to provide a matching scene for the given target. The target's name may be set at the Vuforia Target Manager. To be able to handle possible errors gently during scene allocation, this step should be performed *before* an appropriate target may possibly be recognized. This can be achieved by either relying on spatial data (GPS, etc.) or by simply loading all scenes upon initalization of the system.

3.2.3 Joining Vuforia and SceneKit

SceneKit and Vuforia can not be integrated ad-hoc

SceneKit is built on top of OpenGL ES, hence arbitrary SceneKit content can be rendered into an existing OpenGL ES context, `EAGLContext` [SceneKit Framework Reference, OpenGL ES Framework Reference]. Therefore, taking advantage of SceneKit's features such as `SCNScene` and physics in `FLAppAR` is not conceptually impossible, yet combining it with Vuforia yielded some difficulties.

A coordinate in 3D space is projected onto the camera by matrix multiplication

The traditional approach to translate coordinates of the three-dimensional space onto the camera consists of the multiplication of three matrices, namely *model matrix*, *view matrix*, and *projection matrix*. While the model matrix describes translation, rotation and scaling of an object in 3D space, the view matrix transforms these coordinates from the model's own perspective into a global coordinate system. Subsequent to that, the projection matrix projects the coordinates onto points of the 2D camera plane.

Figure 3.8 illustrates said approach that is used by Vuforia in conjunction with OpenGL: The *extrinsic matrix* here describes translation and rotation of the tracked target—scaling is dispensable here—, and is multiplied onto the *in-*

[13] https://developer.apple.com/ibeacon/

$$\lambda \underbrace{\begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix}}_{\text{Point in viewpoint}} = \underbrace{\begin{pmatrix} f_x & 0 & x_{c0} \\ 0 & f_y & y_{c0} \\ 0 & 0 & 1 \end{pmatrix}}_{K} \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}}_{[R|t]} \underbrace{\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}}_{\text{Point in target}}$$

Figure 3.8: Coordinate translations in 3D [Iwaya, 2016]; K is the intrinsic matrix, which describes properties such as focal length. $[R|t]$ is the extrinsic matrix and describes the camera's position in global coordinates, as well as its rotation.

trinsic matrix that describes camera-specific attributes, such as focal length. The extrinsic matrix is also named *pose matrix*. While the extrinsic matrix represents the view matrix, the intrinsic matrix depicts the projection matrix.

When a target has been recognized by Vuforia, its pose matrix can be acquired by calling `getPose()` of `Vuforia::TrackableResult` [Vuforia API Reference]. This matrix represents the extrinsic matrix. The translation to camera coordinates is still required.

Vuforia provides an extrinsic matrix

SceneKit, however, does not expose such matrices. Since its approach to set up and render 3D content is simplified, a `SCNNode` object's model matrix can rather be adjusted by setting its `position`, `rotation`, `scale` or `transform` properties. SceneKit's representation of a viewport, `SCNCamera`, is attached to a `SCNNode` as well. OpenGL's approach from figure 3.8 is not applicable here.

SceneKit does not expose intrinsic and extrinsic matrices

```
- (void)setCameraMatrix:(Vuforia::Matrix44F)matrix {
    SCNMatrix4 extrinsic =
      [self SCNMatrix4FromVuforiaMatrix44:matrix];
    SCNMatrix4 inverted = SCNMatrix4Invert(extrinsic);

    self.cameraNode.transform = inverted;
}
```

Listing 3.1: Transforming SceneKit's camera[Iwaya, 2016]

Rather than following the traditional approach, we pursue to derive the camera's position and rotation in the 3D scene

We rather assign the extrinsic matrix to the camera's position

and assign them to the camera node's `transform` property. This is achieved by first transforming the model view matrix into a SceneKit 4×4 matrix, `SCNMatrix44`. Then, *inverting* the extrinsic matrix will result in the camera's pose matrix (listing 3.1) [Iwaya, 2016].

```
Vuforia::Matrix44F modelViewMatrix =
  Vuforia::Tool::convertPose2GLMatrix(result->getPose());

[self setCameraMatrix:modelViewMatrix];
[self.renderer renderAtTime:CFAbsoluteTimeGetCurrent()];
```

Listing 3.2: Rendering SceneKit content [Iwaya, 2016]

By then using SceneKit's `SCNRenderer` in our rendering loop, SceneKit content will be rendered into the OpenGL ES context at each render call, as seen in listing 3.2.

3.2.4 Scene Configuration

We approached normalization of 3D models

Scenes from different vendors not only differ in their size—they yield differences in rotation, position and size. Hence, normalization of those models was required. Since we intended the framework's API to be as simple as possible, we introduced configurable scenes, whose settings are encoded in the JSON format.

```json
{
  "rotation": [0, 1, 0, 90],
  "autoscale": true,
  "scale": 0.6,
  "visibleNodes": ["licht", "_01.5_-_animiert"],
  "sceneName": "Scenes.scnassets/Kunstherz.dae"
}
```

Listing 3.3: Sample configuration of a scene

3D objects can be configured with JSON files

Such a configuration, as depicted in figure 3.3, can easily be created and read by both humans and machines, due to the text-based approach of the JSON format. Alternatively to creating a new `ARScene` by providing a file name or a scene that has already been imported, scenes that are preconfigured can be initialized with the `initWithConfig` constructor.

To validate the correctness of such a configuration file, we provide a JSON Schema that describes the configuration properties. This schema contains definitions of the following properties:

- `autoscale` (boolean value, optional). Whether to normalize the scene's size by automatically re-scaling it to target dimensions. The autoscale value defaults to `true`.
- `offset` (3-element vector of integers, optional). Position offset added to the scene's position.
- `position` (3-element vector of integers, optional). Position of the scene. Since the position of all immediate children of the scene's root node is set to this value, it is recommended to use this property only if one child node is present—for example, by setting the `visibleNodes` property.
- `rotation` (4-element vector of integers, optional). If supplied, the scene is rotated by the provided three- or four-dimensional vectors – according to the number of array items. The first three items describe the rotation direction (either 0 or 1), and the fourth item describes the rotation angle, in degree. If omitted, the rotation angle is set to 90°.
- `scale` (decimal number, optional). Additional scaling to be applied to the scene. Since `autoscale` will fit the scene to the image target's dimensions, an augmenting 3D scene may occlude the target. Setting an additional scaling of 0.5 to 0.8 will avoid that.
- `sceneName` (character string, *required*). The relative path to the Collada scene file within the main bundle.
- `visibleNodes` (array of character strings, optional). In Collada, a model is structured like a tree—beginning with the root node, each node may have its children. If supplied, only a subset of immediate children of

the scene's root node remains after import; all other children are removed. Therewith, particular parts of a model may be dismissed during presentation.

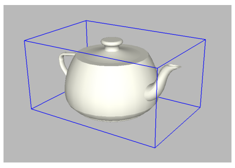
These properties may be changed after import by their respective class properties in `ARScene`. All elements with character string values are of type `NSString`, while vector values are represented by `SCNVector3` respectively `SCNVector4`.

Animation of objects as well as material transparency are supported by SceneKit out-of-the-box. If a Collada scene contains animated objects, SceneKit will by default start performing the animations in a loop.

Automatic Scaling

Size normalization was necessary due to different sizes of 3D models

When importing several sample scenes into our system, we discovered that their sizes differed from each other: Since the vendors of these scenes worked with various scales, some models appeared very small and were hardly visible, while others exceeded the viewpoint of the Augmented Reality window and occupied the device's whole screen. To prevent such incidents an automatic scaling functionality has been implemented. This feature anticipates manual calculation of appropriate scaling of scenes and can be applied to every scene that is imported into the system.



The minimum bounding box of the infamous Utah Teapot

The default dimensions of an image target are calculated automatically when uploading an image for processing to Vuforia Target Manager. Vuforia's C++ API exposes the method `getSize()` of `QCAR::ObjectTarget` that returns a three-dimensional `float` vector, `QCAR::Vec3F` [Vuforia API Reference]. Although being a 3-element vector, a `QCAR::ImageTrackable` only uses the `x` and `y` parameters for X respectively Y axis.

The minimum bounding box of a three-dimensional object is the quboid with the smallest size, in terms of volume, containing the whole object . By calculating this box, we can determine the size of any scene and scale it appropriately. By using `getBoundingBoxMin:max:` [SceneKit Framework Reference] of the SceneKit API, we receive two vectors describing the corners of the bounding box. From these corners, we can easily derive the bounding box' size with arithmetical operations.

By acquiring the sizes of both the image target and the 3D scene we can finally calculate their respective scaling factor and perform appropriate scaling of the scene.

3.2.5 Improving Rendering Stability

Although pose estimation of the Vuforia engine appeared quite reliable, rendered objects often had some *jitter* in their appearance. Applying *Extended Tracking*—a tracking mode that takes as well into account features of the environment—resulted in just some slight improvements. Altough the device is held quite stable in a person's hand, the presented 3D object's position yields small changes. Vincent et al. [2013] were aware of this inaccuracy and conducted a study to understand said jitter in different user pointing techniques.

Rendering was instable, therefore we introduced filtering

To overcome this jitter, we conducted two filtering techniques:

Filtering of small differences By applying a discrete filter that omits small changes in position, the object's pose can be stabilized under certain circumstances. Small inaccuracies due to hand tremor may be filtered, yet at the cost of lags appearing during rendering when making slight hand movements intentionally. *We tested this approach as ad-hoc solution to this problem, but dismissed it due to the rendering lag issues.*

Filtering of different movement directions Intuitively, jitter yields movement of an object into two different diretions. By applying a directional filter, intentional movements can be detected by providing an appropriate threshold angle, while unintended noisely movements will get filtered. This filtering is powered by using the *cosine similarity* between the last ($\vec{a}$) and the

Cosine similarity brought a fitting approach to filter unintended movements current ($\vec{b}$) movement direction:

$$cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|}$$

While $cos(180°) = -1$ predicts totally diverging directions, a result between zero and one (i.e., a difference between 0° and 90°) will describe a movement in a *similar* direction. Testing by filtering changes with $cos(\theta) \leq 0.2$ showed quite stable results.

We settled with the second approach, which relies on *cosine similarity* and filters more selective than the ad-hoc approach.

3.3 Evaluating the System Performance

We experienced a performance decrease in 3D rendering

Although SceneKit is supposed to integrate into OpenGL ES seamlessly, we experienced a performance decrease when using the approach introduced in section 3.2.3 compared to rendering directly with OpenGL ES. To get an more detailed overview of the performance issue, we considered Apple's Core Animation Instrument [iOS Developer Library].

Therefore, we used two devices—an iPhone 5 and an iPhone 6S—to evaluate their rendering performance. In a well-illuminated setting, we used these devices to recognize the `flapp-frame` target image (figure A.8).

iPhones rendered stable with an average of 53 frames per second

Using the "CoreAnimation" instrument, we gathered comparative video performance data by measuring frames per second over a duration of 20 seconds: While the iPhone 5 had an average of 53 frames per second, the iPhone 6 also achieved an average frame rate of 53 frames per second. This frame rate is scheduled internally by Vuforia [Vuforia API Reference], hence this is no actual performance indication. However, we also compared runtime durations of the render function, which will give insight of how fast the devices are able to perform rendering.

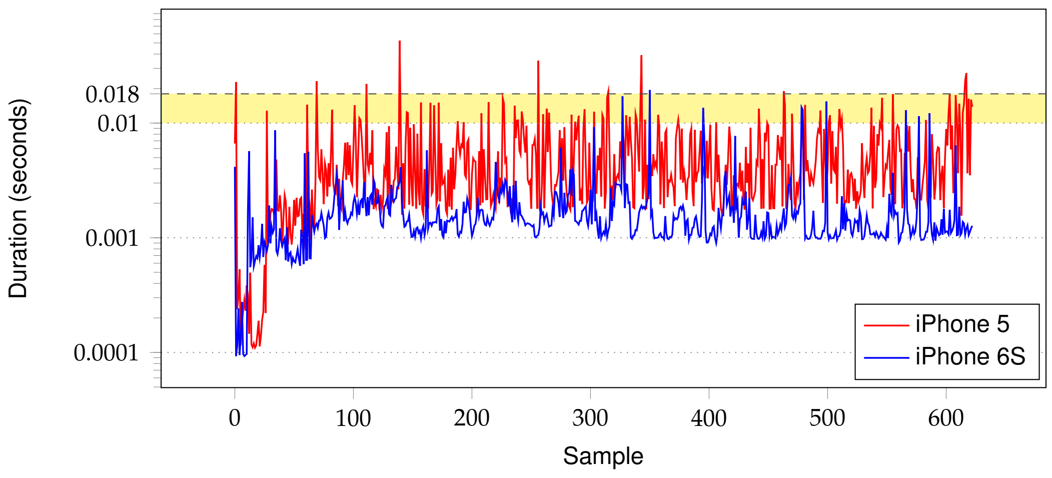


Figure 3.9: Rendering performance of FLAppAR in the sample application during intended use: After focusing an image target (samples 1—70), it is recognized immediately and the smartphone is used to view the augmented 3D object; therefore, reder computation duration increases. The yellow area depicts a duration which should be avoided; otherwise, the current render call might interfere with the next one.

Having an overall average frame rate of 53 frames per second, we can derive that a single rendering of 3D content is supposed to happen in less than 1 second $\div$ 53 $\approx$ 18.87 milliseconds. By achieving this performance, each individual rendering will not interfere with others. We declare a buffer zone of 8 milliseconds within each render which should be avoided to prevent this interference.

We then calculated runtime durations of the render function. After saving timestamps before and after method call, we calculated the timestamps' differences and compared the data. Results of this comparison are illustrated in figure 3.9. iPhone 5 achieved an average of 5.12 milliseconds (min. 0.1 ms, max. 52.29 ms) and iPhone 6S achieved an average of 2 milliseconds (min. 0.09 ms, max. 26.39 ms). Furthermore, an amount of 1.44 % of all iPhone 5 samples is higher than 18.87 ms, which is higher than an average rendering loop duration. In addition, 10.91 % of all iPhone 5 samples are located in the buffer area of 10 to 18.87 milliseconds.

iPhone 5 lacks performance; need for improvement

These results show that, while iPhone 6S performs well, iPhone 5 yields a possible need for optimization—prospects for such improvements are discussed in section 5.1, "Summary and Contributions".

Chapter 4

Guidelines for Developers and Designers

We provide guidelines for developers as well as designers who are working with `FLAppAR`. This chapter is divided into two parts: Section 4.1 will give detailed descriptions on how to set up an iOS application that uses `FLAppAR` to provide an Augmented Reality service. We will give hints on which components of the Application Programming Interface to use and detail several approaches to import of 3D content. Section 4.2 provides insight on the design of trackable images and we conduct suggestions by the Vuforia Developer Library for designing well-trackable target images.

4.1 Development Guidelines

The `FLAppAR` Application Programming Interface has been designed with common Objective-C patterns in mind, such as delegates, views and property lists, while preserving some amount of flexibility for easy porting to other operating systems, e.g. the Android mobile operating system. The framework itself, `FLAppAR`, is intended to run on iOS 8.3 and newer.

For ad-hoc development, a pre-built version of FLAppAR is available. Alternatively, current versions of the code may be obtained from the official repository, as detailed in section 4.1.1.

FLAppAR is released as open source

All code—that is, framework, sample application, and documentation—is licensed under the MIT License[1]. This allows developers to freely use the system or copy parts of it for their own use, as long as the original license is included.

In the following, we will cover setting up the Vuforia library. Sections 4.1.2 and 4.1.3 provide an overview of how to accomplish the necessary prerequisites. Furthermore, common tasks for building an Augmented Reality application with FLAppAR are discussed: Import of scenes in section 4.1.4, and setup of an Augmented Reality view in Sections 4.1.5 and 4.1.6.

4.1.1 Repository Structure

Instead of using a pre-built version of the FLAppAR framework, its source code can be acquired by cloning the official Git[2] code repository. This repository contains an Xcode project which consists of three different targets.

The Xcode project consists of framework, sample application and documentation

FLAppAR The framework itself can be built by running this target.

FLAppAR Sample Alongside the FLAppAR framework, we developed a sample application that provides a simple Augmented Reality experience using self-designed image targets and freely available 3D objects. The application is designed to show different features of the framework and provide insight for developers on how to approach an Augmented Reality application with FLAppAR.

FLAppAR Documentation The code documentation can be generated by building this target. A local version of

[1]https://opensource.org/licenses/MIT
[2]https://git-scm.com/

the Appledoc software is required. The generated documentation is embedded into Xcode as well as stored in the “Documentation” folder of the repository.

Git tags are used to identify versions

In the repository, Git tags are used to identify versions. Therefore, fallbacks to certain versions are easily possible by issuing `git checkout tags/<tag>`. Versions and their respective tag names are structured as `v<major>.<minor>`. Since version `0.10`, we began using Semantic Versioning[3] for declaring the severity of changes in the version numbering.

The `lab` branch contains changes used to gather measurements for, for example, the performance evaluation (section 3.3).

4.1.2 Vuforia API Key Acquisition

A license key is required to run Vuforia

To either run the sample application or use the `FLAppAR` framework, a valid Vuforia license key is required. Before acquiring such license key, a registration for the Vuforia Developer Portal[4] is needed. Several pricing plans are available on the Vuforia pricing website[5]—including the free *Starter* plan, which brings a watermarked Vuforia engine but is free of charge.

After successful registration, the license key must be placed in the main configuration file of `FLAppAR`. The property list file `FLAppAR.plist` is supposed to be placed in the root directory of the application’s main bundle. Setting the character string value of the key `VuforiaLicenseKey` to the given license key will enable `FLAppAR` to properly initialize the Augmented Reality system.

[3]http://semver.org
[4]https://developer.vuforia.com
[5]https://developer.vuforia.com/pricing

4.1.3 Management of Image Targets

Image targets can be managed with a web-based tool

For analysis and compilation of image targets, Vuforia provides a web-based tool, Vuforia Target Manager[6]. By using the web interface, images can be uploaded and grouped into databases. Once a selection of image targets is finished, the database containing these images is available for download and may be, afterwards, loaded by a Vuforia application.

`FLAppAR` is capable of loading a single database file. This database can be referred to by setting the `TargetDatabase` key's value of the main configuration file, `FLAppAR.plist`. The key's value is considered as path, relative from the application's main bundle's root directory.

4.1.4 Importing 3D Scenes

3D scenes may be loaded using multiple approaches

In `FLAppAR`, 3D scenes are represented by the `ARScene` class. This class offers three initialization methods for various allocation strategies. `ARScene` has been designed with three strategies in mind:

Programmatical allocation Initialize the scene, but allocate and assign the *SceneKit* scene programmatically. This can be done by using default constructor, `init`, and setting the `scene` property either by yourself, or by calling the `importSceneNamed:` method.

Implicit allocation By using the `initWithSceneNamed:` constructor, the given scene is automatically loaded.

Pre-configured allocation Initializing the scene with a given configuration, as outlined in section 3.2.4. The third constructor of `ARScene`, `initWithConfig:`, can be used to let `ARScene` conduct such a configuration file with initial adjustments, such as scaling and offsetting. The referenced scene file is loaded implicitly.

[6]https://developer.vuforia.com/target-manager

Furthermore, there are several configuration properties provided which are described in section 3.2.4. These properties can be set either in a configuration file, or after initialization, via their class properties.

4.1.5 Setting up an Augmented Reality View

As described in section 3.2.2 "System Lifecycle", the system's view controller, `ARViewController`, serves as entry point to `FLAppAR`. By setting a `UIView`'s view controller to `ARViewController`, or an extension of `ARViewController`, the view then will become capable of presenting Augmented Reality content. As soon as the view controller's `loadView` method is called, the Augmented Reality system will get initialized. Be aware of properly setting up the view controller's delegates, as outlined subsequently in section 4.1.6.

4.1.6 Setting up Delegates

Delegates are used to communicate with the application

`FLAppAR` brings three delegates to provide asynchronuous interactivity between the system and the application encapsuling it: Lifecycle, error, and scene delegates. Each delegate corresponds to a separate property of `ARViewController`: `lifecycleDelegate`, `errorDelegate`, and `sceneDelegate`.

These properties are supposed to be set before initialization of the view controller, by either overloading the view controller's `loadView` method, or by setting the properties upon transitioning into the view controller, for example during `prepareForSegue:`.

Delegates inform about events and request data

In the following, we will detail the protocols, which are separated into two categories: The lifecycle and error delegates function just as handlers for receiving events perform on their own, independently from the remaining system. The scene delegate, on the other hand, actually interacts with the system by returning 3D scene data.

Lifecycle and Error Delegates

The lifecycle delegate informs about updates in the framework's status

As outlined in section 3.2.2 “System Lifecycle”, `FLAppAR` follows a certain lifecycle at runtime. To update elements of the user interface or handle internal resources, the error delegate provides functionality to keep track of the system's current state.

```
@protocol ARLifecycleDelegate <NSObject>
@optional

- (void)handleARInitBegin;
- (void)handleARInitDone;
- (void)handleARDeinitBegin;
- (void)handleARDeinitDone;

- (void)handleARSceneLoad;
- (void)handleARSceneLoadDone;
- (void)handleARSceneUnload;

@end
```

Listing 4.1: LifecycleDelegate protocol

The methods provided by this delegate are listed in listing 4.1. While the first set of methods, including `handleARInitBegin`, are called upon actual changes in the system's lifecycle, the second set of methods is called in response to 3D rendering events.

Since complexity of 3D objects and device performance may vary, loading such content may result in small delays. To inform the user about possible loading times, user interface elements, such as loading indicators, can easily be presented in such situations using said methods.

```
@protocol ARErrorDelegate <NSObject>
@required
- (void)handleARError:(NSError *)error;
@end
```

Listing 4.2: ErrorDelegate protocol

Errors are catched with the error delegate

The error delegate, sketched in listing 4.2, is designed to respond in events or error. Since error fallbacks are handled internally, either in FLAppAR or Vuforia, this approach enables developers to handle errors in the user interface and return to previous views or recover and restart the system.

The code property of the supplied error parameter contains appropriate error codes which are of various severities. For their respective details, we refer to the documentation of FLAppAR.

Scene Delegate

The scene delegate provides on-demand 3D scenes

The purpose of the scene delegate was already outlined in section 3.2.2: To support dynamic allocation of 3D scenes, the renderer of FLAppAR asks for such scenes on-demand. Therewith, unused scenes may be deallocated asynchronously to save memory and computation time.

```
@protocol ARSceneDelegate <NSObject>
@required
- (ARScene *)provideSceneForTarget:
    (NSString *)targetName;
@optional
- (void)dismissSceneForTarget:(NSString *)targetName;
@end
```

Listing 4.3: SceneDelegate protocol

The scene delegate's ARSceneDelegate protocol is outlined in listing 4.3: While the provideSceneForTarget: method is called if a certain scene is desired, the optional dismissSceneForTarget: method is called if the currently recognized target changed and the renderer switched scenes.

4.2 Image Target Guidelines

Image targets must yield to certain constraints

To ensure optimal tracking capabilities, image targets have to fulfill certain constraints: On the one hand, their designs should conform with the feature detection methods used. On the other hand, the physical appearances of these targets

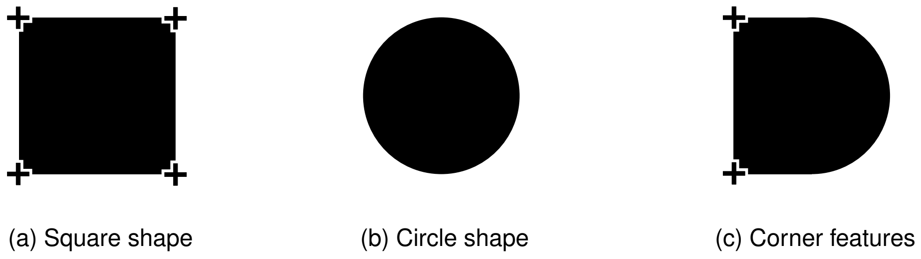


Figure 4.1: The feature detection method of Vuforia recognizes features on an edge- and corner-based approach. Figure 4.1 (a) shows how a simple square shape is recognized, where all four corners each depict a feature. Round shapes, as shown in figure 4.1 (b), do not feature corners, and hence will not have features. Figure 4.1 (c) shows a combination of both round and square shapes. Here, "+" depicts a feature. [Vuforia Developer Library].

imply certain properties, such as glossiness, which affect tracking as well.

Although particular feature recognition methods are not detailed, as Vuforia is proprietary software, several recommendations are provided in the Vuforia Developer Library to optimize tracking performance.

Regarding the shape of a particular image target, the following properties should be considered:

Features are details that are combine edges and corners

Features Features are details in an image which yield to certain constraints—preferably edges and corners, which have appropriate mathematical implications for computational detection [Li and Allinson, 2008]. Features are described similarly in the Vuforia documentation, as depicted in figure 4.1. Designers are encouraged to omit "organic shapes" and rather incorporate "sharp, spiked, chiseled detail[s]" in target images [Vuforia Developer Library]. High amounts of features are desirable.

Higher contrast can increase the number of features

Contrast While low contrast will result in blurry images, higher contrast increases the differences in color between areas of an image [Vuforia Developer Library]. Therefore, high contrast might increase the amount of features by enabling details classified as edges and corners.

Feature distribution After acquiring a set of features for a given image, their distribution within the image creates a unique footprint. Therefore, an uneven yet not centralized distribution of these features across the image canvas is mandatory for a good trackability. Hence, repetitive patterns decrease the uniqueness of such a distrubution and are obliged to avoid [Vuforia Developer Library].

A unique feature distribution is important

In essence, rich details, high contrast, and a well-distributed set of features are good conditions to facilitate good tracking performance.

Since feature detection in Vuforia is achieved with a cloud-based tool, Vuforia Target Manager, supplied images are obliged to be in "8- or 24-bit PNG and JPG formats" with "less than 2MB in size", while "JPGs must be RGB or greyscale (no CMYK)" [Vuforia Developer Library]. Vuforia Target Manager provides performance results of feature detection with a non-detailed 5-star rating system, where zero stars represent a non-trackable image and five stars mean an optimal condition. Since a *feature-exclusion buffer* is applied to an uploaded image to enable tracking of details in surfaces with continuous patterns, an area of 8% of the image's width is excluded from feature tracking and should be considered for feature distribution [Vuforia Developer Library].

Vuforia provides a star-rating system for estimated tracking quality

Additionally, a target's physical appearance may also affect its tracking quality. While its surface may have certain implications such as, for example, to reflect lights, the following physical properties should be considered:

Solidness and Flatness Vuforia performs best if a surface actually is flat [Vuforia Developer Library]—therefore, it is encouraged to use solid materials that hardly bend or get wrinkled, such as thin paper.

Flat and properly sized targets are important

Size The image target's size should be selected thoroughly, taking the estimated distance between the scanning device and the target into account. Since the amount of recognizable features shrinks with decreasing image quality, the target should be sized appropriately [Vuforia Developer Library].

Glossy surfaces might reflect light

Surface finish Glossy surfaces should be avoided if direct light sources, such as the sun, are present. Glossy reflections will affect the tracking quality similar to an occlusion and can create issues with feature recognition [Vuforia Developer Library].

If a target image is designed carefully and its physical appearance is realized with said guidelines in mind, its tracking quality should be considerably increased.

Chapter 5

Summary and Future Work

In this chapter, we summarize the contents and contributions of this thesis. In section 5.1 "Summary and Contributions", we conclude previous chapters and briefly discuss our findings. The last section, 5.2 "Future Work", outlines possible future prospects regarding our work.

5.1 Summary and Contributions

In this thesis, we presented our process of developing FLAppAR. In section 2 "Related Work", we discussed research related to our context of Augmented Reality experiences in tourism. This includes fundamental research in Augmented Reality (section 2.1), research regarding Augmented Reality in tourism (section 2.3), and related software (section 2.6).

Section 3 "Developing an Augmented Reality Application" outlines the work we did on the application. This work is divided into three parts: First, in section 3.1 "Designing the Augmented Reality System", we describe our approach to design the Augmented Reality application. During examination of Augmented Reality tracking methods (section 3.1.1), we took visual and non-visual tracking methods into account, favoring a visual approach. Relying on Natural Feature Tracking instead of Marker Tracking was appealing for us, since artistic freedom for target images was ensured that way. We then performed an evaluation of several Augmented Reality frameworks for the Apple iOS mobile operating system (section 3.1.2). The evaluation's results revealed that one particular framework, Vuforia, outperformed the other frameworks regarding the maximum distance between the device and the target image. Afterwards, in section 3.1.4, we discussed our approach to present 3D content: A combination of OpenGL ES and iOS' SceneKit framework provided enough flexibility.

The second part, section 3.2 "Implementing the Application Design", consists of multiple descriptions of implementational details: Foremost, we detail general concepts such as the structure (section 3.2.1) and the lifecycle (section 3.2.2) of the system. In addition, we discuss particular solutions to the issues we faced: Joining OpenGL ES and SceneKit for 3D rendering is outlined in section 3.2.3. Furthermore, normalization of 3D scenes via configuration files is portrayed in section 3.2.4 and a solution to distortions in the estimated pose matrix of 3D scenes is provided in section 3.2.5.

Facing slight performance issues, we shortly evaluate our own system in section 3.3 "Evaluating the System Performance". With the intention of identifying possible causes for this decrease in performance, we analyzed graphics performancce and system performance.

Accompanying the implementational details of FLAppAR, we formulate multiple guidelines for developing applications with the system as well as design image targets to be used with Vuforia in section 4 "Guidelines for Developers and Designers". Several recommendations for developers (section 4.1) cover tasks such as acquiring an API Key for Vuforia (section 4.1.2) and importing 3D scenes into FLAppAR (section 4.1.4). In section 4.2, we congregated guidelines for improved Vuforia tracking performance, such as improving contrast in imagery and paying attention to target images' physical appearance.

Recalling the performance issues we faced, the evaluation in section 3.3 showed that there is a possible need for optimization in the rendering method. Although iPhone 5 renders 3D content reliably, observations and measurements demonstrated that 10 % of render calls should finish faster. This may be, partly, due to the processing power of iPhone 5—the 2016 iteration of iPhone, iPhone 6S, performed significantly better in the evaluation. To solve this issue, FLAppAR might be further optimized. We are convinced that by analyzing and tuning the approach how 3D content is rendered increased performance may be achieved. Alternatively, the integration of SceneKit and OpenGL ES could be discarded in favor of more efficient solutions, using, for example, iOS' Metal[1] framework.

5.2 Future Work

We see possibilities of future work in three distinguished domains: Design, implementation and physical deployment.

Although we partly evaluated our approach and the resulting application, all work was based on an implementation that was derived from several requirements given (as listed in section 3.1). Research from Kounavis et al. as well as the ARCHEOGUIDE project [Vlahakis et al., 2002] showed that tourism provides an interesting application for Augmented Reality and is well received by users [Vlahakis et al., 2001]. However, we did not evaluate the user experience from either our setting or from FLAppAR itself—mainly, because the main contribution of this thesis is the technical implementation of a framework, not a particular application. In a follow-up study, though, the user experience of the Augmented Reality experience of FLApp could be investigated.

As we remarked in the performance evaluation (section 3.3), FLAppAR yields a possible need for increased 3D rendering performance. In the summary (section 5.1), we already referred to possible solutions: By either opting for more performant devices or optimizing the 3D rendering approach, a significant increase in performance may be achieved. The

[1]https://developer.apple.com/metal/

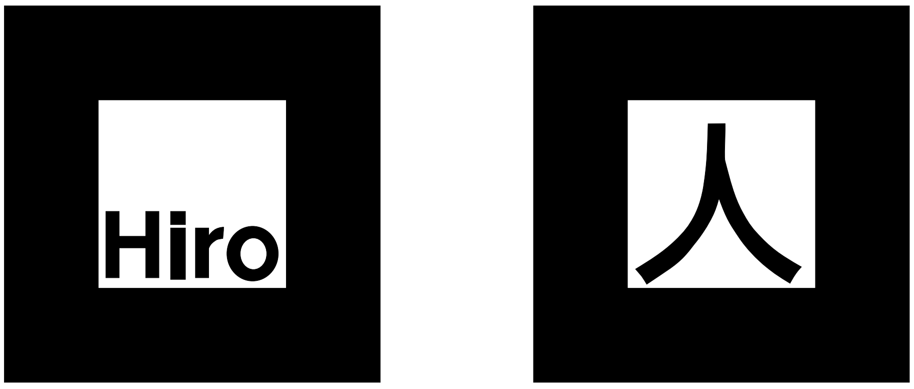
latter, optimization of 3D rendering, could be realized by using purely OpenGL ES or taking iOS' Metal framework into account, which is supported in Vuforia since version 5.5.9, which has been released this spring [Vuforia Developer Library].

With a possible deployment in the center of the city of Aachen in the near future, the physical realization of FLApp (and, therewith, FLAppAR) could also contribute to interesting follow-up research. With the guidelines given in section 4.2, actual Augmented Reality tracking stability in outdoor settings could be evaluated, as well as the comparison of multiple surface materials.

Appendix A

Sample Image Targets

We created several shapes and images to learn about frameworks' performance when using different image targets. Image markers (figure A.1) have been used to test recognition of such targets. Sample image targets from Vuforia's sample applications provided well-distributed image features for testing of Natural Feature Tracking capabilities. To get insight in even and uneven distributions of such features, we designed target images based on QR codes (figures A.3, A.4, and A.5). With figures A.6 and A.7, we created a combination of both frame-like targets and well-distributed image features.



(a) Hiro (b) Kanji

Figure A.1: Image markers from the ARToolKit sample application [ARToolKit Documentation].

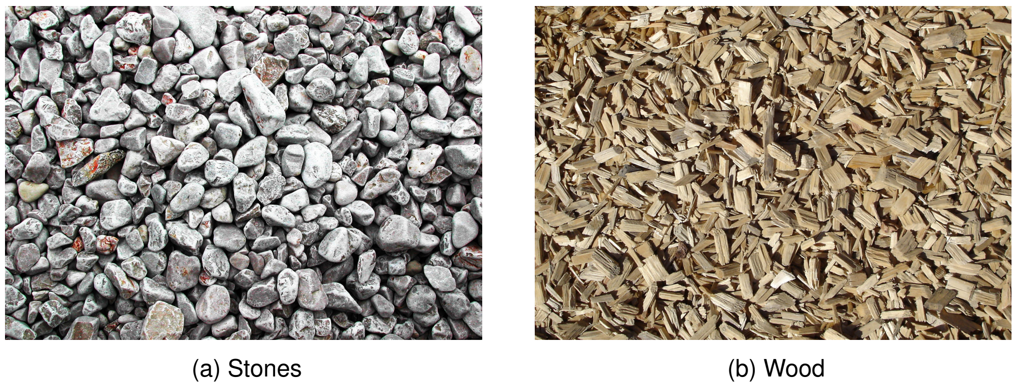


Figure A.2: Sample target images with different patterns, taken from the Vuforia sample application [Vuforia Developer Portal]

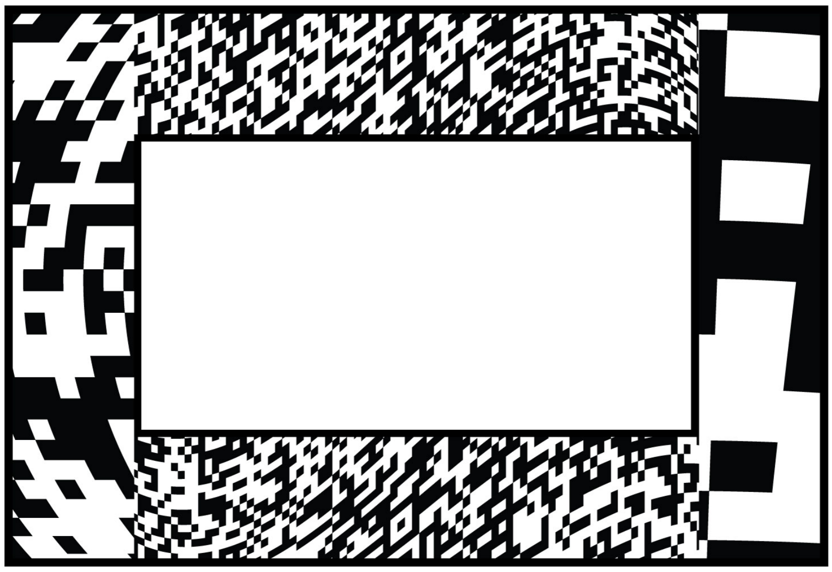


Figure A.3: **QR-A** features a combination of multiple QR codes to create an uneven distribution of edges and corners.

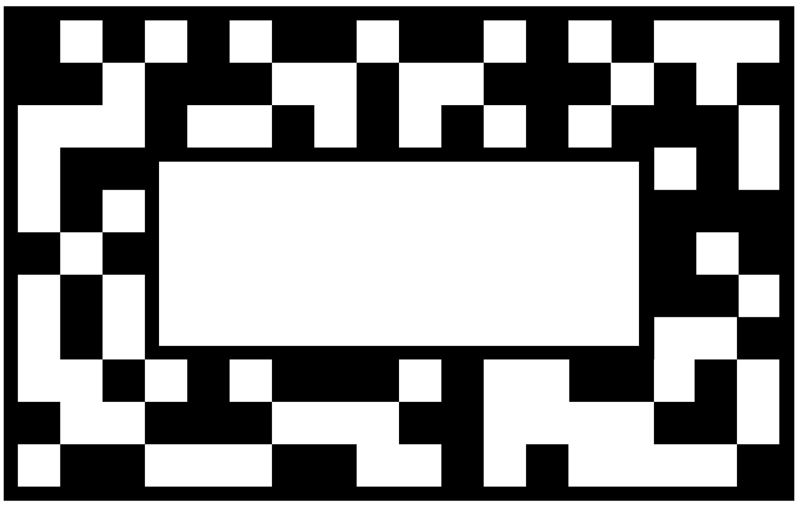


Figure A.4: QR-B features a relatively even and small distribution of edges and corners.

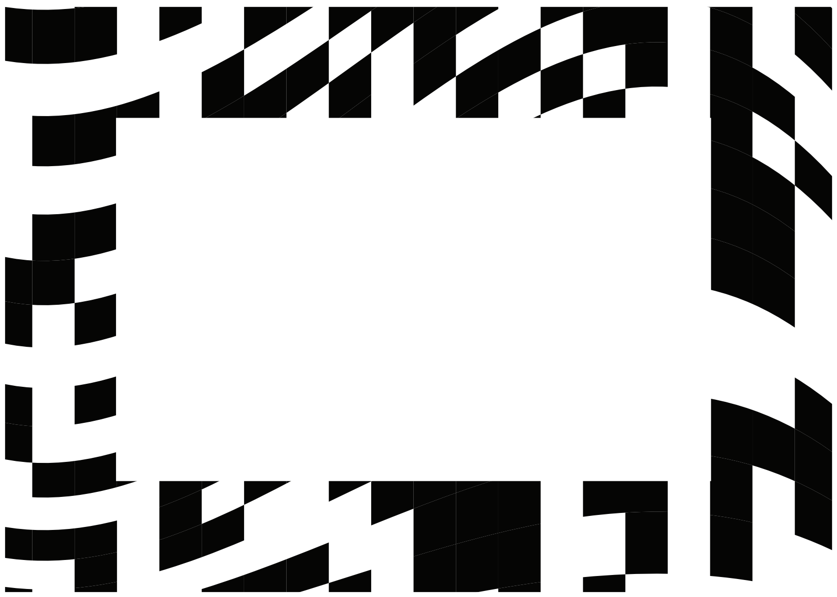


Figure A.5: QR-C features some edges corners, whose distribution is uneven due to warping.

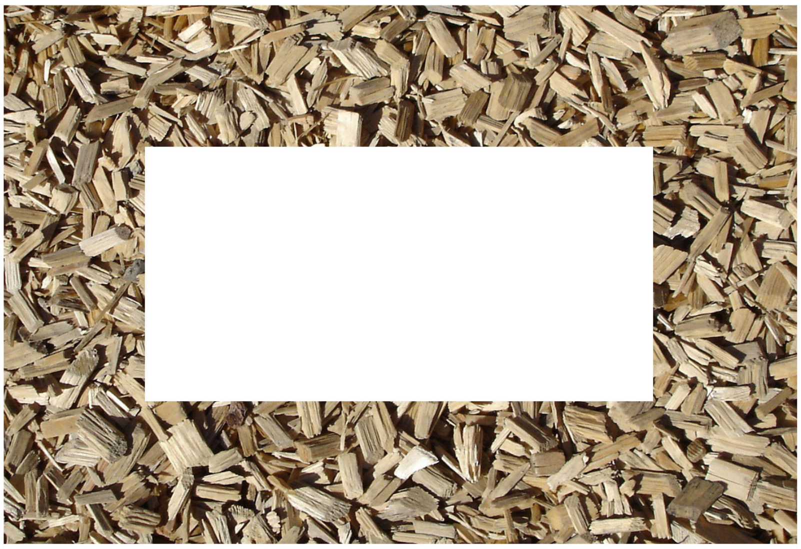


Figure A.6: F-S is based on the Stones pattern image from the Vuforia sample application (figure A.2 (a)), but features a window-like cutout.

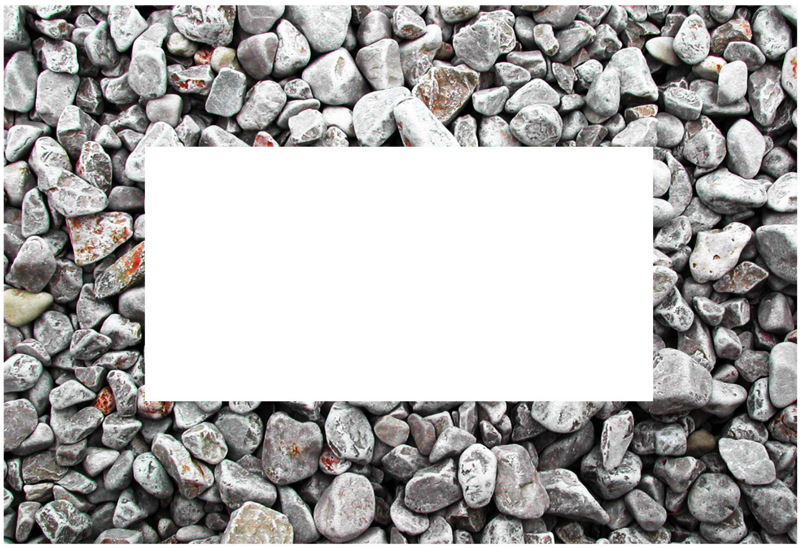


Figure A.7: F-W is based on the Wood pattern image from the Vuforia sample application (figure A.2 (b)), but features a window-like cutout.

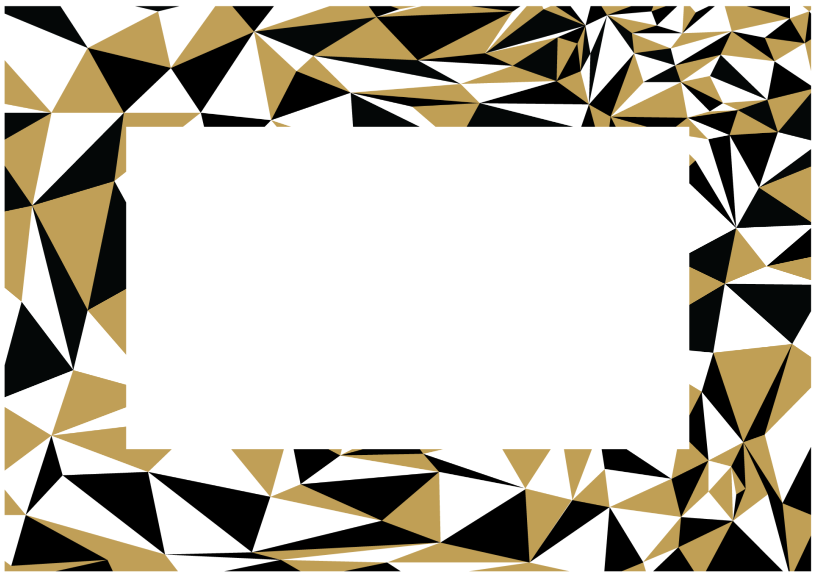


Figure A.8: flapp-frame is based on previous image targets, like figure A.5, but contains colors and a more arbitrary pattern.

Appendix B

Results of the Framework Comparison

Target Image	Wikitude	ARToolKit	Vuforia
Hiro*	135	80	120
Kanji*	95	0	0
Stones†	95	80	160
Wood†	80	0	140
Pinball†	100	0	160
QR-A‡	150	0	260
QR-B‡	155	0	220
QR-C‡	130	0	210
F-S‡	160	80	235
F-W‡	185	80	230

Table B.1: Results from the framework performance comparison survey

The the results of table B.1 are discussed in section 3.1.2 "Comparison of Image Recognition Features". All values are measured in centimeters. The results are to be considered as estimates with a precision of $\pm 5cm$. For the individual target images, the following settings have been used for printing:

*	Tags:	Printed on DIN A4
†	Images:	Printed on DIN A4
‡	Frames:	Printed on DIN A3 (held on white background, for cut-out parts)

Bibliography

Antti Ajanki, Mark Billinghurst, Hannes Gamper, Toni Järvenpää, Melih Kandemir, Samuel Kaski, Markus Koskela, Mikko Kurimo, Jorma Laaksonen, Kai Puolamäki, Teemu Ruokolainen, and Timo Tossavainen. An augmented reality interface to contextual information. *Virtual Real.*, 15 (2-3):161–173, June 2011. ISSN 1359-4338. doi: 10.1007/s10055-010-0183-5. URL `http://dx.doi.org/10.1007/s10055-010-0183-5`.

ARToolKit Documentation. `http://artoolkit.org/documentation/`. Accessed: 2016-05-11.

Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

Ronald T. Azuma. The challenge of making augmented reality work outdoors. In *In Mixed Reality: Merging Real and Virtual*, pages 379–390. Springer-Verlag, 1999.

Oriel Bergig, Nate Hagbi, Jihad El-Sana, Klara Kedem, and Mark Billinghurst. In-place augmented reality. *Virtual Reality*, 15(2):201–212, 2010. ISSN 1434-9957. doi: 10.1007/s10055-010-0158-6. URL `http://dx.doi.org/10.1007/s10055-010-0158-6`.

Mark Billinghurst, Hirkazu Kato, and Ivan Poupyrev. The magicbook—moving seamlessly between reality and virtuality. *IEEE Comput. Graph. Appl.*, 21(3):6–8, May 2001. ISSN 0272-1716. doi: 10.1109/38.920621. URL `http://dx.doi.org/10.1109/38.920621`.

Collada 1.4.1 Specification. `https://www.khronos.org/files/collada_spec_1_4.pdf`. Accessed: 2016-05-11.

Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. doi: 10.1145/1814433.1814453. URL http://doi.acm.org/10.1145/1814433.1814453.

S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 74–81, Oct 1997. doi: 10.1109/ISWC.1997.629922.

F Fritz, A Susperregui, and Maria Teresa Linaza. Enhancing cultural tourism experiences with augmented reality technologies. 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST), 2005.

Future Lab Aachen Website. http://www.futurelab-aachen.de. Accessed: 2016-06-08.

Tim Gleue and Patrick Dähne. Design and implementation of a mobile device for outdoor augmented reality in the archeoguide project. In *Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage*, VAST '01, pages 161–168, New York, NY, USA, 2001. ACM. ISBN 1-58113-447-9. doi: 10.1145/584993.585018. URL http://doi.acm.org/10.1145/584993.585018.

Dai-In Han, Timothy Jung, and Alex Gibson. *Information and Communication Technologies in Tourism 2014: Proceedings of the International Conference in Dublin, Ireland, January 21-24, 2014*, chapter Dublin AR: Implementing Augmented Reality in Tourism, pages 511–523. Springer International Publishing, Cham, 2013. ISBN 978-3-319-03973-2. doi: 10.1007/978-3-319-03973-2_37. URL http://dx.doi.org/10.1007/978-3-319-03973-2_37.

Tobias Höllerer, Steven Feiner, Tachio Terauchi, Gus Rashid, and Drexel Hallaway. Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23:779–785, 1999.

iOS Developer Library. ios developer library. `https://developer.apple.com/library/ios/navigation/`. Accessed: 2016-03-20.

iPhone 4S Technical Specifications. iphone 4s technical specifications. `https://support.apple.com/kb/SP643?locale=en_US`. Accessed: 2016-04-25.

Akira Iwaya. Making augmented reality app easily with scenekit + vuforia (in english). `http://qiita.com/akira108/items/a743138fca532ee193fe`, 2016. Accessed: 2016-03-04.

H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94, 1999. doi: 10.1109/IWAR.1999.803809.

Hirokazu Kato, Mark Billinghurst, Ivan Poupyrev, Kenji Imamoto, and Keihachiro Tachibana. Virtual object manipulation on a table-top ar environment. In *Augmented Reality, 2000.(ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 111–119. Ieee, 2000.

Chris D Kounavis, Anna E Kasimati, and Efpraxia D Zamani. Enhancing the tourism experience through mobile augmented reality: Challenges and prospects. *International Journal of Engineering Business Management*, 4, 2012.

N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications Magazine*, 48(9):140–150, Sept 2010. ISSN 0163-6804. doi: 10.1109/MCOM.2010.5560598.

Jing Li and Nigel M. Allinson. A comprehensive review of current local features for computer vision. *Neurocomput.*, 71(10-12):1771–1787, June 2008. ISSN 0925-2312. doi: 10.1016/j.neucom.2007.11.032. URL `http://dx.doi.org/10.1016/j.neucom.2007.11.032`.

Blair Macintyre, Jay David Bolter, Emmanuel Moreno, and Brendan Hannigan. Augmented reality as a new media experience. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, ISAR '01, pages 197–, Washington, DC, USA, 2001. IEEE Computer

Society. ISBN 0-7695-1375-1. URL http://dl.acm.org/citation.cfm?id=582828.881323.

Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

Thomas Olsson, Else Lagerstam, Tuula Kärkkäinen, and Kaisa Väänänen-Vainio-Mattila. Expected user experience of mobile augmented reality services: A user study in the context of shopping centres. *Personal Ubiquitous Comput.*, 17(2):287–304, February 2013. ISSN 1617-4909. doi: 10.1007/s00779-011-0494-x. URL http://dx.doi.org/10.1007/s00779-011-0494-x.

OpenGL ES Framework Reference. Opengl es framework reference. https://developer.apple.com/library/ios/documentation/OpenGLES/Reference/OpenGLES_Framework/. Accessed: 2016-04-20.

George Papagiannakis, Michal Ponder, Tom Molet, Sumedha Kshirsagar, Frederic Cordier, M Magnenat-Thalmann, and Daniel Thalmann. Lifeplus: revival of life in ancient pompeii, virtual systems and multimedia. In *Proceedings of VSMM 2002*, number VRLAB-CONF-2007-038, 2002.

George Papagiannakis, Sébastien Schertenleib, Brian OKennedy, Marlene Arevalo-Poizat, Nadia Magnenat-Thalmann, Andrew J. Stoddart, and Daniel Thalmann. Mixing virtual and real scenes in the site of ancient pompeii. *Journal of Visualization and Computer Animation*, 16: 11–24, 2005.

David Rönnqvist. Scene kit. https://www.objc.io/issues/18-games/scenekit/, 2014. Accessed: 2016-03-18.

SceneKit Framework Reference. Scenekit framework reference. https://developer.apple.com/library/ios/documentation/SceneKit/Reference/SceneKit_Framework/. Accessed: 2016-03-18.

D Stricker and T Kettenbach. Real-time and markerless vision-based tracking for outdoor augmented reality applications. In *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 189–190, 2001. doi: 10.1109/ISAR.2001.970536.

Ivan E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA, 1968. ACM. doi: 10.1145/1476589.1476686. URL http://doi.acm.org/10.1145/1476589.1476686.

Thomas Vincent, Laurence Nigay, and Takeshi Kurata. Handheld augmented reality: Effect of registration jitter on cursor-based pointing techniques. In *Proceedings of the 25th Conference on L'Interaction Homme-Machine*, IHM '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2407-6. doi: 10.1145/2534903.2534905. URL http://doi.acm.org/10.1145/2534903.2534905.

Vassilios Vlahakis, John Karigiannis, Manolis Tsotros, Michael Gounaris, Luis Almeida, Didier Stricker, Tim Gleue, Ioannis T. Christou, Renzo Carlucci, and Nikos Ioannidis. Archeoguide: First results of an augmented reality, mobile computing system in cultural heritage sites. In *Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage*, VAST '01, pages 131–140, New York, NY, USA, 2001. ACM. ISBN 1-58113-447-9. doi: 10.1145/584993.585015. URL http://doi.acm.org/10.1145/584993.585015.

Vassilios Vlahakis, Nikolaos Ioannidis, John Karigiannis, Manolis Tsotros, Michael Gounaris, Didier Stricker, Tim Gleue, Patrick Daehne, and Luís Almeida. Archeoguide: an augmented reality guide for archaeological sites. *IEEE Computer Graphics and Applications*, (5):52–60, 2002.

Vassilios Vlahakis, Thomas Pliakas, Athanasios M Demiris, and Nikolaos Ioannidis. Design and application of an augmented reality system for continuous, context-sensitive guided tours of indoor and outdoor cultural sites and museums. In *VAST*, pages 155–164, 2003.

Vuforia API Reference. https://developer.vuforia.com/library/sites/default/api/cpp/index.html. Accessed: 2016-05-11.

Vuforia Developer Library. https://developer.vuforia.com/library/. Accessed: 2016-05-11.

Vuforia Developer Portal. `https://developer.vuforia.com/library/`. Accessed: 2016-05-11.

Vuforia Target Manager. `https://developer.vuforia.com/targetmanager`. Accessed: 2016-05-11.

Pierre Wellner, Wendy Mackay, and Rich Gold. Back to the real world. *Commun. ACM*, 36(7):24–26, July 1993. ISSN 0001-0782. doi: 10.1145/159544.159555. URL `http://doi.acm.org/10.1145/159544.159555`.

Wikitude Documentation. `http://www.wikitude.com/documentation/`. Accessed: 2016-05-11.

Index

Typeset June 17, 2016