

Bearbeitungsbeginn: 01.09.2018

Vorgelegt am: 21.02.2019

# Thesis

zur Erlangung des Grades

**Master of Science**

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

***Kathrin Fuhrer***

***Matrikelnummer: 256343***

**Entwicklung einer Anwendung zur Einbindung und  
Programmierung von AR-Applikationen mit ARCore für die  
Integration in einen webbasierten Game-Editor (FUDGE)**

*Erstbetreuer: Prof. Jirka Dell'Oro-Friedl*

*Zweitbetreuer: Prof. Dr. Norbert Schnell*



---

## Abstract

Virtual Reality, Augmented Reality, Mixed Reality – Begriffe, die mehr und mehr an Bedeutung erlangen. Es gibt mittlerweile unzählige Anwendungen, die diese Technologien nutzen und dadurch dem Nutzer neue bzw. andere Erlebnisse ermöglichen. In Web-Anwendungen sind diese jedoch noch nicht im Einsatz. Es gibt kaum Websites, die Inhalte in virtueller oder augmentierter Realität darstellen. Allerdings gibt es einige Anwendungsbereiche dafür und die Technologie könnte in hybriden Anwendungen verwendet werden. Mit FUDGE (Furtwangen University Didactic Game Editor) soll ein Game-Editor entstehen, der (vor allem im Unterricht und beim ersten Kontakt mit der Spieleentwicklung) die Grundlagen dafür mit einfachen Funktionen erklärt. Dabei sollen Spiele für die unterschiedlichsten Plattformen entwickelt werden können.

Das Ziel dieser Arbeit ist es, eine Bibliothek zu entwickeln, welche die Funktionalität für die Erstellung von AR-Anwendungen im Web und damit auch in FUDGE bietet. Es soll möglich sein, über einfache Funktionsaufrufe Inhalte mit erweiterter Realität für mobile Anwendungen zu generieren.

Anhand einer Analyse werden AR-API untersucht und daraus eine passende für die Entwicklung der Anwendung ausgewählt. Zudem wird recherchiert, wie existierende Game-Engines die Erstellung von AR-Anwendungen handhaben. Erstellt wird daraus eine AR-Bibliothek für die Webtechnologien.

Virtual reality, augmented reality, mixed reality – terms that become more and more important. There are innumerable applications that use these technologies and allow the user new or different experiences. But these technologies are not yet in use in web applications. There are hardly any websites that present content in virtual or augmented reality. However, there are some application areas for this and the technology could be used in hybrid applications. The idea of FUDGE (Furtwangen University Didactic Game Editor) is to create a game editor that explains the basics of game developing with simple functions (especially in the classroom). It should be able to develop games for a variety of platforms.

The aim is to develop a library which provides the functionality for creating AR applications on the web and thus in FUDGE. It should be possible to use simple function calls to generate content with augmented reality for mobile applications.

Based on an analysis, AR-API will be examined and a suitable one selected for the development of the application. In addition, it will be researched how existing game engines handle the creation of AR applications. The result is an AR library for the web technologies.



---

## Inhaltsverzeichnis

Abstract.....	I
Inhaltsverzeichnis .....	III
Abbildungsverzeichnis .....	VII
Tabellenverzeichnis .....	IX
Listingverzeichnis .....	XI
Abkürzungsverzeichnis .....	XIII
1    Einleitung .....	1
1.1    Ausgangssituation.....	1
1.2    Zielsetzung .....	2
1.3    Problemdarstellung und Fragestellung .....	2
2    Grundlagen.....	3
2.1    Virtual Reality (VR), Augmented Reality (AR) und Mixed Reality (MR) .....	3
2.2    AR-Tracking.....	5
2.3    „Cross-platform“-Anwendungen.....	7
2.3.1    Allgemein .....	7
2.3.2    Adobe PhoneGap .....	8
2.3.3    Electron.....	9
2.4    FUDGE .....	9
2.5    ARCore .....	10
2.6    WebGL - 3D im Web.....	11
3    Analyse vorhandener API für AR .....	12
3.1    Allgemein .....	12

3.2	Untersuchung von AR-Plugins in Unity .....	12
3.2.1	Vuforia for Unity .....	12
3.2.2	ARCore for Unity .....	16
3.2.3	Ergebnisse dieser Untersuchungen .....	23
3.3	Vorstellung vorhandener Web-API für AR .....	24
3.3.1	Three.js und three.ar.js .....	24
3.3.2	AR.js und A-Frame .....	28
3.3.3	Tracking.js .....	30
3.3.4	WebXR .....	32
3.4	Kriterien für die Analyse .....	37
3.5	Beschreibung der Web-API bezüglich der Kriterien .....	39
3.6	Auswertung der Analyse .....	43
4	Konzeption der Anwendung .....	45
4.1	Anforderungen .....	45
4.2	Das Konzept .....	46
4.2.1	Technologien und Zusammenhänge .....	46
4.2.2	Vorgehensweise .....	49
4.2.3	Ablaufdiagramm .....	51
4.3	Struktur .....	52
4.4	Klassendiagramm .....	55
5	Umsetzung der Anwendung .....	56
5.1	Beschreibung der Umsetzung .....	56
5.2	Herausforderungen .....	66
6	Zusammenfassung .....	68

---

7	Fazit und Ausblick .....	70
8	Literatur- und Quellenverzeichnis.....	72
9	Eidesstattliche Erklärung .....	76
10	Anhang .....	77





---

## Abbildungsverzeichnis

Abbildung 2.1 VR-Brille und CAVE .....	3
Abbildung 2.2 Screenshot von Pokémon GO (links) und Snapchat (rechts).....	4
Abbildung 2.3 Microsoft Hololens.....	5
Abbildung 2.4 Beispiele für AR-Marker .....	7
Abbildung 2.5 Darstellung einer hybriden vs. nativen Anwendung.....	8
Abbildung 3.1 Ausschnitt der Vuforia-Website mit Verwaltung der Bilder-Datenbank .	13
Abbildung 3.2 Vuforia-Konfiguration für die Bilder-Datenbank.....	14
Abbildung 3.3 Screenshot für die Image-Target-Einstellungen mit Vuforia.....	14
Abbildung 3.4 Ergebnis der Vuforia-Beispiel-Anwendung.....	15
Abbildung 3.5 ARCore-Komponenten für die FirstPersonCamera/ AR-Kamera .....	16
Abbildung 3.6 Screenshot der ARCore-Anwendung mit PointCloud.....	17
Abbildung 3.7 Screenshot der ARCore-Anwendung mit PlaneGenerator .....	18
Abbildung 3.8 Visualisierung des Raycasts / Hit-Tests.....	19
Abbildung 3.9 Screenshot der ARCore-Anwendung mit visuellen Objekten.....	20
Abbildung 3.10 ARCore ImageDatabase erstellen in Unity .....	21
Abbildung 3.11 AugmentedImagesSessionConfig von ARCore .....	21
Abbildung 3.12 Screenshot einer Anwendung mit ar.js .....	29
Abbildung 4.1 Zusammenhänge der verwendeten Technologien.....	47
Abbildung 4.2 Ablaufdiagramm der Anwendung.....	51
Abbildung 4.3 Struktur der Anwendungen .....	54
Abbildung 5.1 Von der Anwendung zur XR-Bibliothek .....	58
Abbildung 5.2 Screenshot der Anwendung mit Terminal-Ausgabe .....	64



---

## Tabellenverzeichnis

Tabelle 2.1 Markerbasierende Tracking-Methoden.....	6
Tabelle 3.1 Übersicht der Web-API (Stand: 16.01.2019) .....	40



---

## Listingverzeichnis

Listing 3.1 Raycast (Hit-Test) mit ARCore in Unity (C#).....	20
Listing 3.2 Erkennung von Bild-Markern in Unity mit ARCore (C#) .....	22
Listing 3.3 Platzierung virtueller Objekte auf einem Marker in Unity mit ARCore (C#) .....	23
Listing 3.4 Beispiel einer einfachen three.js-Anwendung .....	25
Listing 3.5 Animation eines Würfels in three.js .....	26
Listing 3.6 Initialisierung eines AR-Displays mit three.ar.js.....	27
Listing 3.7 Initialisierung der Marker-Arten in three.ar.js.....	27
Listing 3.8 Initialisierung der ARView und AR-Kamera in three.ar.js.....	28
Listing 3.9 THREE.VRControls zur AR-Kamera hinzufügen.....	28
Listing 3.10 Erkennen eines Markers in three.ar.js.....	28
Listing 3.11 Einfache A-Frame AR-Anwendung mit AR.js .....	30
Listing 3.12 Zugriff auf ARToolkit mit AR.js und three.js.....	30
Listing 3.13 Implementierung der Nutzer-Kamera-Ansicht über JavaScript .....	31
Listing 3.14 HTML-Video-Element für die Kamera-Ansicht mit JavaScript.....	31
Listing 3.15 Verwendung des tracking.js-Tracking-Objekts .....	32
Listing 3.16 Abfrage des XR-Devices, wenn Voraussetzungen für WebXR erfüllt sind ..	33
Listing 3.17 Starten einer WebXR-XRSession .....	34
Listing 3.18 WebXR Rendering mit WebGLLayer .....	34
Listing 3.19 WebXR – requestFrameOfReference() und requestAnimationFrame().....	35
Listing 3.20 WebXR requestAnimationFrame-Callback (mit three.js) .....	36
Listing 3.21 Verwendung des Hit-Tests in WebXR .....	37
Listing 5.1 Erstellen eines neues PhoneGap-Projekts.....	61
Listing 5.2 readFile-Funktion zum Öffnen eines PhoneGap-Projekts .....	62
Listing 5.3 Funktion, um ein PhoneGap-Projekt auf einem lokalen Server zu starten.....	63



---

## Abkürzungsverzeichnis

API	Application Programming Interface
AR	Augmented Reality (Erweiterte Realität)
CAVE	Cave Automatic Virtual Environment (Virtuelle Umgebung in einem Käfig)
CSS	Cascading Style Sheet
ES	ECMAScript (JavaScript Standard)
FUDGE	Furtwangen University Didactic Game Editor
GPS	Global Positioning System (Globales Positionierungssystem)
HTML	HyperText Markup Language
JS	JavaScript
MR	Mixed Reality (Vermischte Realität)
NDK	Native Development Kit
SDK	Software Development Kit
SLAM	Simultaneous Localization And Mapping (Gleichzeitige Lokalisierung und Kartierung/Abbildung)
SRC	Source (= Quelle)
TS	TypeScript
UI	User Interface (Benutzeroberfläche)
VR	Virtual Reality (Virtuelle Realität)





# 1 Einleitung

## 1.1 Ausgangssituation

Nichts unterliegt dem Wandel der Zeit mehr als der Fortschritt der Technik. Es werden neue Technologien entwickelt und alte Technologien durch neue Methoden und andere Techniken aufbereitet – neue Begriffe werden geprägt.

Was derzeit mehr und mehr an Bedeutung gewinnt, sind Virtual Reality (VR), Augmented Reality (AR) und Mixed Reality (MR). Es gibt mittlerweile unzählige Anwendungen, die diese Technologien nutzen und dadurch dem Nutzer neue bzw. andere Erlebnisse ermöglichen. Ein jeder kann sein mobiles Endgerät nehmen, eine entsprechende Applikation installieren und damit virtuelle oder erweiterte Realitäten erleben.

Eine weitere Technologie, die aus dem Alltag nicht mehr weg zu denken ist, ist das World Wide Web. Weltweit werden täglich etwa 5,6 Milliarden Suchanfragen auf Google getätigt (vgl. Löffler kein Datum). Dabei werden mehr als 1,9 Milliarden Websites durchsucht (Stand 05.02.2019), wobei die Zahlen stetig wachsen (vgl.

InternetLiveStats.com kein Datum). Auch die Webstandards wie HTML, CSS und JavaScript werden ununterbrochen weiterentwickelt. Die gängigen Browser unterstützen derzeit HTML 5, CSS 3 und ES 6 für JavaScript (vgl. Deveria 2019). Es wird angestrebt, das Web immer weiter zu verbessern und andere Medien und Technologien darin zu integrieren.

Ein weiteres aktuelles Projekt befasst sich mit der Aufgabe einen Game-Editor auf der Basis der Webtechnologien zu entwickeln. Dieser soll sowohl im Web als auch auf den gängigen Betriebssystemen Windows, MacOS und Linux funktionieren. Diese Funktionalität wird mit Hilfe der hybriden Entwicklung in Electron ermöglicht. Dieser

Game-Editor hat den Namen FUDGE (Furtwangen University Didactic Game Editor). Weitere Details zum Game-Editor werden in Kapitel 2.4 beschrieben.

## 1.2 Zielsetzung

Ziel dieser Arbeit ist es, eine Anwendung in Electron zu entwickeln, die es ermöglicht AR-Anwendungen zu erstellen. Diese Anwendung soll in einer Form geschrieben werden, die eine einfache Implementierung dieser in den Game-Editor FUDGE möglich macht. Benötigt wird daher eine Bibliothek zur Einbindung von AR in den Game-Editor FUDGE.

Für die Bereitstellung von AR-Inhalten soll die Software ARCore von Google verwendet werden. Diese ermöglicht das Erkennen von Oberflächen, Wänden, Ecken und Kanten (siehe Kapitel 2.5). Dazu soll die bereits von Google entwickelte Erweiterung für ARCore in Unity untersucht werden, ebenso wie andere Unity-Packages und bereits existierende Web-AR-Bibliotheken.

## 1.3 Problemdarstellung und Fragestellung

Derzeit gibt es kaum Websites, deren Inhalt mit virtueller oder augmentierter Realität dargestellt wird, jedoch einige native Anwendungen für Smartphones oder Tablets. Daher ist es eine Herausforderung, eine stabile Bibliothek zu finden, welche die Funktionalität für AR und VR in die Webtechnologien integriert. Anhand von AR-Erweiterungen für Unity soll untersucht werden, wie AR-Anwendungen in einer Game-Engine bzw. einem Game-Editor integriert werden, damit diese für den neu entwickelten FUDGE übernommen werden können.

Die meisten AR-Anwendungen werden außerdem für mobile Endgeräte entwickelt. Daher muss eine Möglichkeit gefunden werden, AR-Anwendungen aus Electron heraus in mobile Anwendungen zu exportieren.

## 2 Grundlagen

### 2.1 Virtual Reality (VR), Augmented Reality (AR) und Mixed Reality (MR)

In der „Virtual Reality“, also der virtuellen Realität, wird eine computergenerierte Wirklichkeit dargestellt. Das bedeutet, dass alles was gesehen wird von einem Computer generiert und abgebildet wird. Diese virtuelle Welt wird meist in einem Raum auf großen Leinwänden dargestellt (CAVE) oder mit Hilfe einer speziellen (VR-)Brille (Head-Mounted-Display). (Dieser Abschnitt vgl. Bendel 2018)



Abbildung 2.1 VR-Brille<sup>1</sup> und CAVE<sup>2</sup>

Bei „Augmented Reality“ wird von der erweiterten Realität gesprochen. Hierbei wird die Realität in der Wahrnehmung durch virtuelle Objekte erweitert. Ein Merkmal von AR ist die immer vorhandene Darstellung der Realität, meist in Form von einer Kamera-Ansicht in Applikationen. Diese wird durch Textinformationen oder Abbildungen, wie 3D-Objekten, ergänzt. (Dieser Abschnitt vgl. Markgraf 2018) Ein bekanntes Beispiel einer AR-Anwendung ist das Spiel „Pokémon GO“ und die Social-Media-Plattform „Snapchat“ (Abbildung 2.2).

---

<sup>1</sup> Quelle: <https://www.oculus.com/>

<sup>2</sup> Quelle: <https://www.werigi.com/cave>



Abbildung 2.2 Screenshot von Pokémon GO (links) und Snapchat (rechts)

AR findet allerdings nicht nur in Smartphones oder Tablets Anwendung: Es gibt auch „AR-Brillen“ (oft Datenbrillen genannt), die eine erweiterte Realität ermöglichen. Hier wird dann von „Mixed Reality“ gesprochen. Die reale Welt wird dabei nicht auf einem oder mehreren Displays angezeigt, sondern es werden nur die virtuellen Elemente auf den Gläsern der Brille angezeigt und wirken dadurch als wären sie in der realen Szene dargestellt. Ein Beispiel für eine solche Brille ist die Microsoft Hololens. (Dieser Abschnitt vgl. Länger 2017)

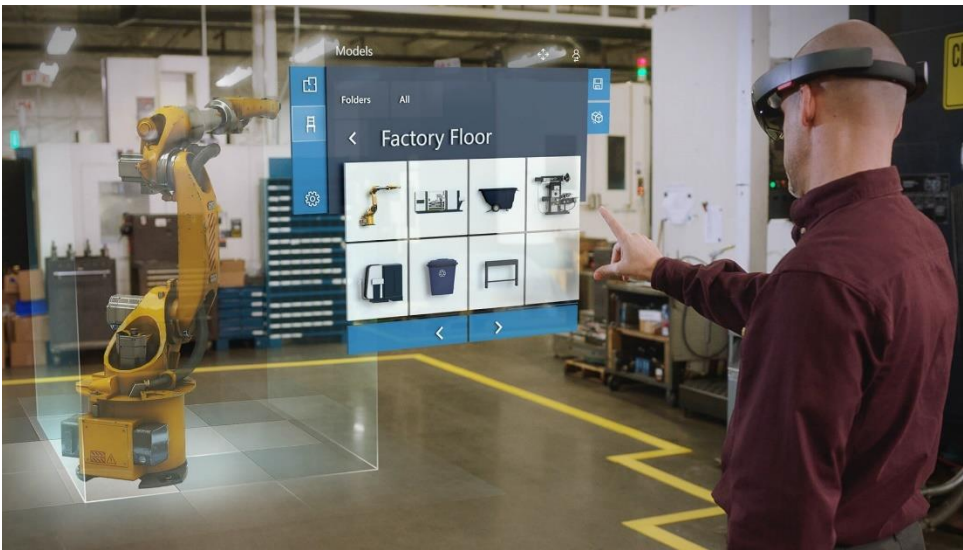


Abbildung 2.3 Microsoft Hololens<sup>3</sup>

## 2.2 AR-Tracking

Das Tracking bildet die Grundlage für die Funktionalität von AR. Es gibt verschiedene Arten von Tracking, meist wird jedoch das optische Tracking verwendet. Dieses lässt sich in zwei Bereiche unterteilen: Markerbasierendes (markerbased) und markerloses (markerless) Tracking. Beim markerbasierenden Tracking werden, wie der Name schon sagt, Marker verwendet.

*„Als Augmented Reality Marker oder AR-Marker werden die visuellen Trigger bezeichnet, welche die Anzeige der virtuellen Zusatzinformation auslösen.“  
(AnyMotion GmbH 2019)*

Diese Marker werden mit Hilfe des Trackings in der Realität wahrgenommen und an deren Position wird der virtuelle Kontext dargestellt. Jeder Marker, der in einer Anwendung verwendet wird, muss dieser zunächst beigebracht werden, damit die Software den Marker während der Laufzeit erkennt. Die Marker sind daher für jede Anwendung speziell angefertigt. Es gibt unterschiedliche Arten von Markern und dazugehörige Tracking-Methoden, die jedoch nicht alle von jedem System unterstützt

---

<sup>3</sup> Quelle: Microsoft <https://www.microsoft.com/de-de/hololens>

werden. Einige markerbasierende Tracking-Methoden werden in Tabelle 2.1 aufgelistet. Beispiele für mögliche Marker werden in Abbildung 2.4 dargestellt.

Marker/Methode	Kurzbeschreibung
<b>Frame-Marker/ Border-Marker</b>	Ein 2D-Marker, der meist einem QR-Code ähnelt und zudem einen dicken schwarzen oder weißen Rand hat. Dabei wird von der Anwendung zuerst nach dem dicken Rand gesucht und anschließend nach dessen Inhalt. Diese Marker sind meist quadratisch und werden oft auf Papier gedruckt.
<b>Image-Marker</b>	Diese Marker sind die logische Weiterentwicklung der Frame-Marker. Der Inhalt des Markers besteht dabei aus einem Bild. Dazu kann jedes beliebige Bild verwendet werden. Wird oft auch NFT-Marker (Natural Feature Tracking) genannt, da hierbei die „natürlichen Eigenschaften“ des Bildes extrahiert werden, um dieses wieder zu erkennen.
<b>Object-Marker</b>	Die Object-Marker erlauben es als Vergleichsbild ein 3D-Objekt zu verwenden und ist damit ein 3D-Marker. Eine Technologie namens SLAM (simultaneous localization and mapping) ermöglicht das Abgleichen von Eckpunkten und Kanten mit diesem gespeicherten 3D-Objekt.
<b>GPS-Marker</b>	GPS-Marker durch die Verwendung von GPS-Koordinaten relativ ungenau. Daher wird diese Art der Tracking-Methode von kaum einem Framework unterstützt und auch nur wenige AR-Anwendungen verwenden diese Methode. Virtuelle Objekte werden hierbei mit bestimmten GPS-Koordinaten verknüpft und anhand dieser und der Bewegung des Endgeräts angezeigt.

Tabelle 2.1 Markerbasierende Tracking-Methoden<sup>4</sup>

<sup>4</sup> (Diese Tabelle vgl. AnyMotion GmbH 2019)



Abbildung 2.4 Beispiele für AR-Marker<sup>5</sup>

Beim markerlosen Tracking ist das Erlernen von Markern im System nicht notwendig, allerdings muss das System hierbei die Umgebung erkennen. Dabei wird ein breites Spektrum an Bildverarbeitungsmethoden verwendet, um Oberflächen, Kanten, Eckpunkte und Farben zu erkennen und daraus ein eigenes Abbild der Umgebung zu schaffen, das zur Berechnung von Positionen relativ zur Welt verwendet werden kann. (Dieses Kapitel vgl. Tönnis 2010)

## 2.3 „Cross-platform“-Anwendungen

### 2.3.1 Allgemein

„Cross-platform“ oder auch „plattformübergreifend“ bedeutet, dass ein Programm von einem Programmierer in nur einer Programmiersprache entwickelt und anschließend auf verschiedenen Betriebssystemen genutzt werden kann. Diese Fähigkeit wird oft auch als „multi-platform“ bezeichnet. Eine „cross-platform“-Anwendung wird außerdem auch als hybride Anwendung bezeichnet. (Dieser Abschnitt vgl. WebFinance, Inc. 2019)

Soll ein Programmierer beispielsweise eine Anwendung für Android und iOS entwickeln, muss er für Android das Android SDK verwenden (Google LLC 2018) und für iOS Swift - eine speziell für Apple-Produkte entwickelte Programmiersprache (Apple Inc. 2019).

---

<sup>5</sup> Quellen: <https://jeromeetienne.github.io/AR.js/data/images/HIRO.jpg>; <https://xml3d.github.io/xml3d-examples/examples/xflowAR/img/64.png>

Oder er verwendet eine Programmiersprache, mit der es möglich ist diese Anwendung als „cross-platform“-Anwendung zu erstellen z. B. mit PhoneGap.

### 2.3.2 Adobe PhoneGap

Adobe PhoneGap ist ein Open Source-Framework, das ein Teil des Apache Cordova Projekts ist. Es bietet die Möglichkeit hybride, mobile Applikationen auf der Basis von Webtechnologien (HTML, CSS und JavaScript) zu erstellen. (Vgl. Adobe Systems Inc. 2016)

Die PhoneGap API bietet einen nativen Container für die unterschiedlichen Betriebssysteme, in dem eine Instanz eines Browserfensters dargestellt wird. Der Code, welcher vom Entwickler in HTML, CSS und JavaScript erstellt wird, wird in diesem Browserfenster angezeigt. Eine Veranschaulichung dafür zeigt Abbildung 2.5. Mit diesem Container ist es außerdem möglich auf die Hardware-Komponenten des Endgeräts zuzugreifen. (Dieser Abschnitt vgl. Trice 2012)

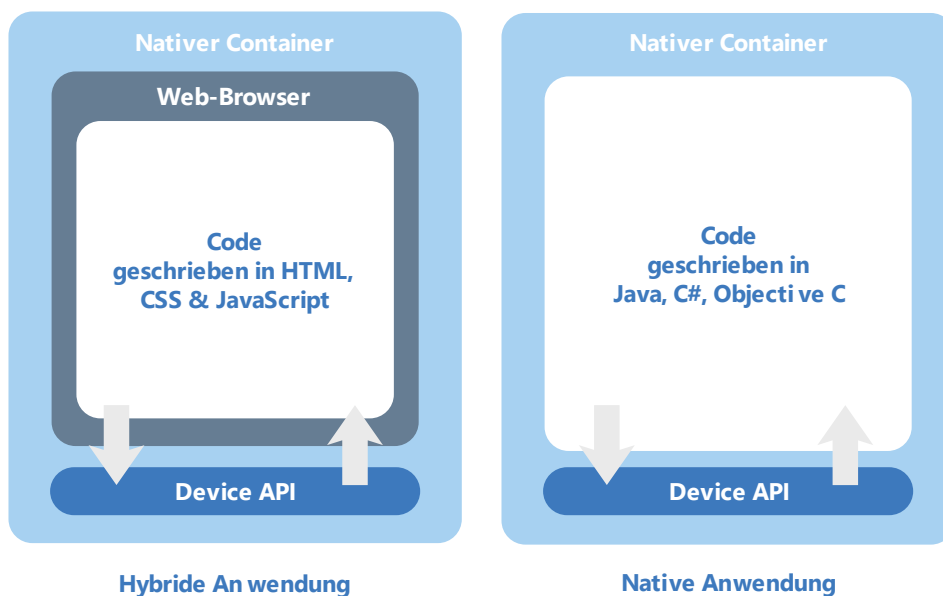


Abbildung 2.5 Darstellung einer hybriden vs. nativen Anwendung

Diese Art von Frameworks gibt es nicht nur für mobile Applikationen, sondern auch für Desktop-Anwendungen. Das wohl bekannteste Beispiel hierfür ist Electron.



### 2.3.3 Electron

Wie PhoneGap ist auch Electron ein Framework um plattformübergreifende Anwendungen auf der Basis der Webtechnologien zu entwickeln. Anwendungen, die mit Electron entwickelt wurden, sind z. B. Skype, GitHub Desktop, Atom und Wordpress.com. (GitHub, Inc. kein Datum)

Electron wird mit Node.js und Chromium entwickelt. Node.js ermöglicht serverseitiges JavaScript mit Hilfe der V8 JavaScript Engine von Chrome (Node.js Foundation kein Datum). Chromium ist ein Open Source Browser Projekt von Googles Browser Chrome (Google LLC kein Datum). Durch das Verwenden von Node.js ist es des Weiteren möglich die verschiedensten Node-Module über den Node-Package-Manager (NPM) zu installieren (GitHub Inc. 2019).

Mit diesen Bestandteilen erstellt Electron einen Container für den Chromium-Browser. In diesem Container ist es (wie auch bei PhoneGap) möglich auf die Hardware-Komponenten des Endgeräts zuzugreifen.

Das Interface der Anwendungen hat immer das Aussehen eines klassischen Programms des aktuellen Betriebssystems - so kann das Menü auch immer im nativen Look angelegt werden. (Dieser Abschnitt vgl. GitHub, Inc. kein Datum)

## 2.4 FUDGE

FUDGE ist ein Akronym und steht für Furtwangen University Didactic Game Editor. Der Game-Editor ist ein Projekt von Professor Jirka Dell'Oro-Friedl an der Hochschule Furtwangen. Mit FUDGE soll ein Open-Source Game-Editor entstehen, der hauptsächlich für die Lehre von 2D und 3D audiovisueller Spieleentwicklung genutzt werden soll. Da die meisten Studenten mit ihren eigenen PC arbeiten, soll dieser Editor für alle gängigen Plattformen entwickelt werden. Studenten sollen lernen, wie Spiele und interaktive Anwendungen erstellt werden. Daher sollen auch die Ergebnisse der Anwendung in einem gut leserlichen Format gespeichert werden und für die

meistgenutzten Plattformen entwickelt werden können. Um diesen Anforderungen gerecht zu werden, wird FUDGE mit Hilfe der folgenden Technologien entwickelt:

- HTML und CSS
- JavaScript bzw. TypeScript für die Entwicklung
- Node.js
- WebGL
- JSON
- NoSQL
- WebAssembly
- Electron und PhoneGap/Cordova

Das Projekt rund um FUDGE befindet sich derzeit in der Konzeptions- und Findungsphase. Das Ergebnis dieser Arbeit ist ebenfalls Teil des Projekts. (Dieses Kapitel vgl. Dell'Oro-Friedl 2018)

## 2.5 ARCore

ARCore ist eine von Google entwickelte Software, um AR-Anwendungen zu erstellen. Je nach Entwicklungsumgebung muss ein entsprechendes SDK integriert werden. Folgende SDK stehen derzeit zur Verfügung (Stand 13.01.2019):

- Android SDK
- Android NDK
- Unity (Android und iOS)
- iOS
- Unreal

Mit dieser Software ist es möglich die Umgebung bzw. die unterschiedlichen Oberflächen zu erkennen und mit diesen Informationen zu interagieren. Dafür werden von Google drei Schlüssel-Kapazitäten genannt:

*„**Motion tracking** allows the phone to understand and track its position relative to the world.*

***Environmental understanding** allows the phone to detect the size and location of all type of surfaces: horizontal, vertical and angled surfaces like the ground, a coffee table or walls.*

***Light estimation** allows the phone to estimate the environment's current lighting conditions.” (Google LLC 2018)*

Mit „Motion tracking“ ist es also möglich die Position des Endgeräts relativ zur Welt herauszufinden, „Environmental understanding“ ist demnach für das Erkennen der Oberflächen zuständig, und mit „Light estimation“ wird das Umgebungslicht eingeschätzt und kann so für eine real wirkende Darstellung der virtuellen Objekte verwendet werden. In der Bewegung des Endgeräts werden die Punkte und Kombinationen, welche die Software zur Interpretation verwendet, immer wieder aktualisiert und neu interpretiert. ARCore bietet damit die Algorithmen und Technologien für AR-Entwickler um markerloses Tracking zu ermöglichen. (Gesamter Abschnitt vgl. Google LLC 2018)

## 2.6 WebGL - 3D im Web

WebGL ist ein auf OpenGL ES basierender JavaScript Web-Standard, der es ermöglicht Low-Level-3D-Grafiken über ein HTML5-Canvas-Element zu integrieren. Durch diese Erweiterung der Browser-Funktionen können 3D-Objekte, ohne die Installation bzw. Integration weiterer Plugins, angezeigt werden. (Dieser Abschnitt vgl. The Khronos Group Inc 2019)

## 3 Analyse vorhandener API für AR

### 3.1 Allgemein

Es gibt sehr viele Entwickler, die sich dafür einsetzen, AR in das World Wide Web zu integrieren. Einige haben dafür eine berufliche Motivation, wie zum Beispiel das Google AR-Team, andere haben ein persönliches Interesse daran. Um die Möglichkeiten von AR im Web sehen und verwirklichen zu können, ist es notwendig bereits vorhandene API für AR zu untersuchen und sich dabei nicht nur auf Web-API zu beschränken, sondern zudem API von Game-Engines wie Unity anzusehen. Diese Arbeit soll beleuchten, welche Einflussmöglichkeiten ein Game-Developer auf Parameter einer von ihm erstellten AR-Anwendung haben muss, da es Ziel dieser Arbeit ist, eine Entwicklungsumgebung für AR-Anwendungen zu schaffen. Diese notwendigen Einflussmöglichkeiten werden im Folgenden definiert, um die Auslegung der Entwicklungsumgebung dafür in Kapitel 5 optimal zu gestalten.

### 3.2 Untersuchung von AR-Plugins in Unity

#### 3.2.1 Vuforia for Unity

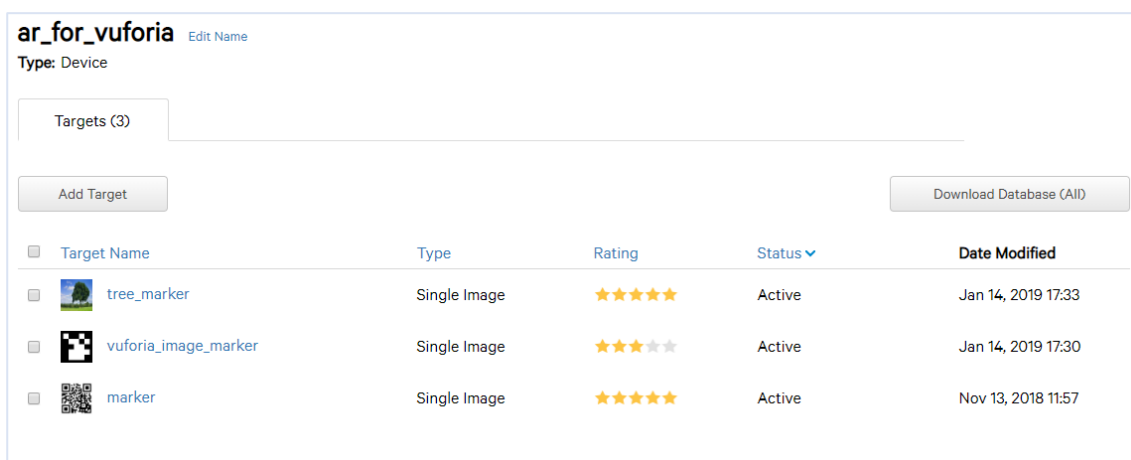
Vuforia ist ein SDK zur Erstellung von AR-Anwendungen. Dazu können verschiedene Editoren verwendet werden: Android Studio, Xcode, Visual Studio oder Unity (vgl. PTC Inc. 2019). In diesem Abschnitt wird eine Anwendung in Unity betrachtet, welche mit Vuforia erstellt wird. Somit kann ein Eindruck gewonnen werden, wie AR-Anwendungen in einem Game-Editor erstellt werden können.

In einem Vuforia-Unity-Projekt muss zunächst eine AR-Kamera zur Szene hinzugefügt werden. Diese AR-Kamera ist ein Prefab<sup>6</sup> von Vuforia und hat grundlegend die gleichen Eigenschaften wie eine normale Kamera in Unity mit dem Zusatz, dass das Bild der

---

<sup>6</sup> Ein Prefab ist ein vorgefertigtes/ gespeichertes Unity-GameObject, bei dem alle Komponenten (Transformationen, Skripte, Shader, etc.) wie gewünscht erhalten bleiben (vgl. Unity Technologies 2019).

Kamera des Endgeräts angezeigt wird. Die standardmäßig in die Szene integrierte „MainCamera“ entfällt hiermit.<sup>7</sup> Ab jetzt ist es möglich die Ansicht der WebCam zu sehen, wenn die Anwendung gestartet wird. Anschließend müssen die Marker, die innerhalb der Anwendung erkannt werden sollen, in einer Bilder-Datenbank gespeichert werden. Diese Datenbank wird über die Vuforia Website verwaltet. Abbildung 3.1 zeigt einen Ausschnitt der Website mit einer Beispiel-Datenbank.



**ar\_for\_vuforia** [Edit Name](#)  
Type: Device

Targets (3)

[Add Target](#) [Download Database \(All\)](#)




<input type="checkbox"/> Target Name	Type	Rating	Status ▾	Date Modified
<input type="checkbox"/>  tree_marker	Single Image	★★★★★	Active	Jan 14, 2019 17:33
<input type="checkbox"/>  vuforia_image_marker	Single Image	★★★★☆	Active	Jan 14, 2019 17:30
<input type="checkbox"/>  marker	Single Image	★★★★★	Active	Nov 13, 2018 11:57

Abbildung 3.1 Ausschnitt der Vuforia-Website mit Verwaltung der Bilder-Datenbank

In die Datenbank können sowohl komplexe Bilder als auch QR-Codes oder eigens erstellte Bildmotive hochgeladen werden. Die Sterne unter dem Punkt „Rating“ zeigen an, wie gut das Bild als Marker verwendet werden kann bzw. wie viele Eckpunkte und damit Referenzpunkte ein Bild für die Erkennung hat. Das Rating zeigt, kurz gesagt, die Qualität des Markers. Nachdem die Datenbank über den Package-Manager in das Unity-Projekt importiert wurde, muss diese noch in der Vuforia-Konfiguration geladen und aktiviert werden (Abbildung 3.2). Anschließend wird ein „ImageTarget“-Prefab in die Szene geladen, dessen Verhalten auf die entsprechende Datenbank und ein bestimmtes

<sup>7</sup> Für Vuforia ist außerdem die Angabe eines Lizenz-Schlüssels notwendig, was für die Untersuchung dieser Unity-Erweiterung jedoch keine Rolle spielt.

Bild darin angepasst werden muss. In diesem Beispiel ist der Name der Datenbank „ar\_for\_vuforia“ und das gewünschte Bild hat die id „tree\_marker“ (Abbildung 3.3).

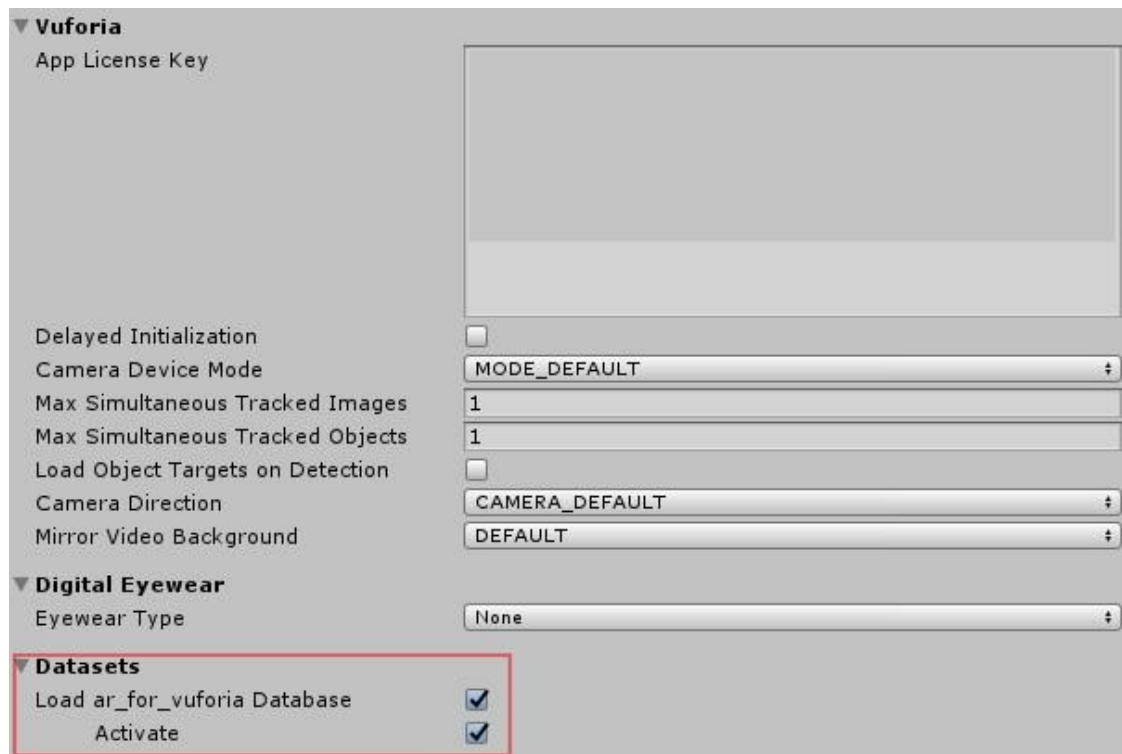


Abbildung 3.2 Vuforia-Konfiguration für die Bilder-Datenbank

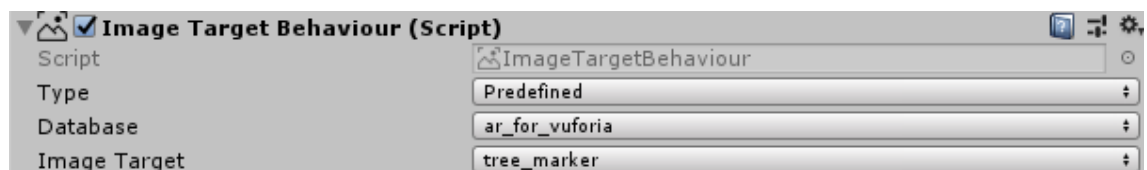
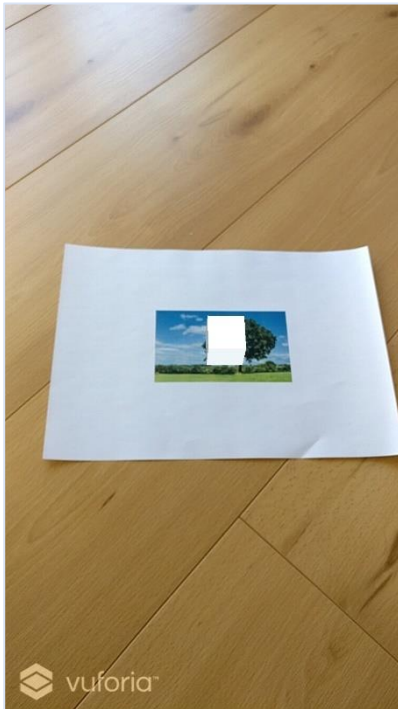


Abbildung 3.3 Screenshot für die Image-Target-Einstellungen mit Vuforia

Zum Schluss muss dem ImageTarget-Objekt nur noch das Objekt untergeordnet werden, das auf dem Marker abgebildet werden soll. Das Ergebnis dieses Beispiels wird in Abbildung 3.4 abgebildet.



*Abbildung 3.4 Ergebnis der Vuforia-Beispiel-Anwendung*

Anhand dieser einfachen Beispiel-Anwendung können wir festlegen, dass für Vuforia folgende Objekte von elementarer Bedeutung sind:

- ARCamera
- Database
- Target
  - GameObject (Ziel-Objekt)

Die Datenbank muss dabei nicht zwingend eine Bilder-Datenbank sein. Es ist auch möglich eine Objekt-Datenbank zu erstellen. Daraus lässt sich allerdings ableiten, dass immer Referenz-Objekte für die Verwendung mit Vuforia vorhanden sein müssen und dadurch nur markerbasierende Tracking-Methoden möglich sind.

Es gibt die Möglichkeit mit dem sogenannten „SmartTerrain“-Prefab die Umgebung der Marker zu scannen. Dieses wird als virtuelles Objekt beschrieben, mit dem auch interagiert werden kann. Allerdings werden immer Marker benötigt, die mit diesem

„SmartTerrain“ verknüpft sind, und virtuelle Objekte können nur anhand von Markern dargestellt werden. Durch das „SmartTerrain“ können lediglich Effekte erzeugt werden, wie das Verdecken von virtuellen Objekten durch reale Objekte. (Dieser Abschnitt vgl. PTC Inc. 2018)

### 3.2.2 ARCore for Unity

In Kapitel 2.5 wird beschrieben, was ARCore ist – nun wird untersucht, wie es funktioniert und welche Komponenten in Unity integriert werden. Da es das Ziel dieser Arbeit ist, markerlose AR-Anwendungen erstellen zu können, betrachten wir in ARCore eine markerlose Anwendung. Zuerst muss, wie auch bei Vuforia, eine AR-Kamera in die Szene geladen werden. Dazu gibt es auch von ARCore ein Prefab „ARCoreDevice“. Diesem Prefab ist das GameObject „FirstPersonCamera“ von Unity untergeordnet. Die Kamera ist mit zwei weiteren Komponenten ausgestattet, die für das Tracking der Position der Kamera bzw. des Endgeräts („Tracked Pose Driver“) und für die Abbildung der Kamera-Ansicht („AR Core Background Renderer“) notwendig sind (Abbildung 3.5).

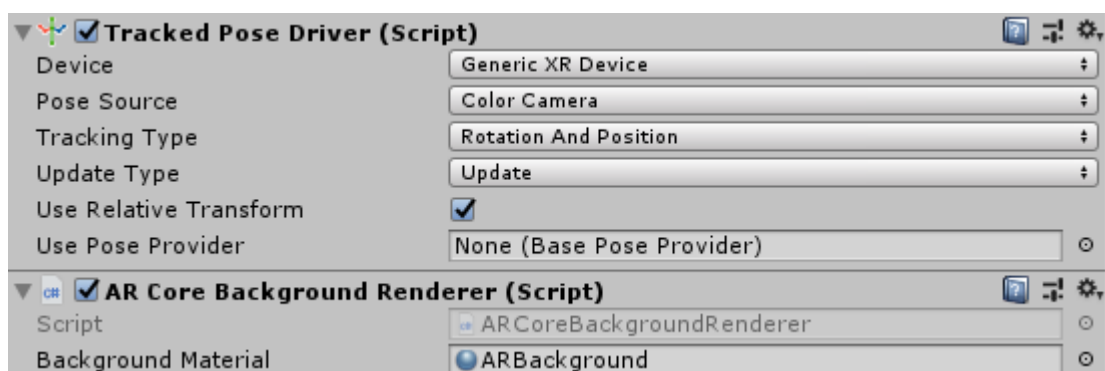


Abbildung 3.5 ARCore-Komponenten für die FirstPersonCamera/ AR-Kamera

Für die AR-Anwendung wird nur noch ein Controller (im weiteren Verlauf „ExampleController“ genannt) benötigt: Ein GameObject, dessen Skript die AR-Szene interpretiert. Dazu sind Referenzen zu folgenden Objekten notwendig:



- FirstPersonCamera
- Prefabs für die dynamisch erstellten visuellen Objekte (Ziel-Objekte)

Des Weiteren bietet ARCore noch zwei zusätzliche Prefabs an, die vor allem in der Entwicklung behilflich sein können: zum einen die „PointCloud“ und zum anderen den „PlaneGenerator“. Die „PointCloud“ (übersetzt Punkte-Wolke) stellt mit kleinen Punkten (standardmäßig in türkis) die Positionen dar, die als markante Punkte im aktuellen Frame erkannt wurden („Feature Points“). Für die Berechnung und das Erstellen der Objekte wird das „Pointcloud Visualizer“-Skript verwendet (Abbildung 3.6).



Abbildung 3.6 Screenshot der ARCore-Anwendung mit PointCloud

Der „PlaneGenerator“ erstellt, wie der Name schon sagt, Ebenen – und zwar immer an der Stelle, an der die Software eine Fläche erkannt hat. Abbildung 3.7 zeigt die Anwendung mit dem „PlaneGenerator“. Dass die Software unterschiedliche Flächen erkennt, ist mit diesem Prefab leicht zu erkennen, da jede Fläche ihre eigene Farbe erhält.

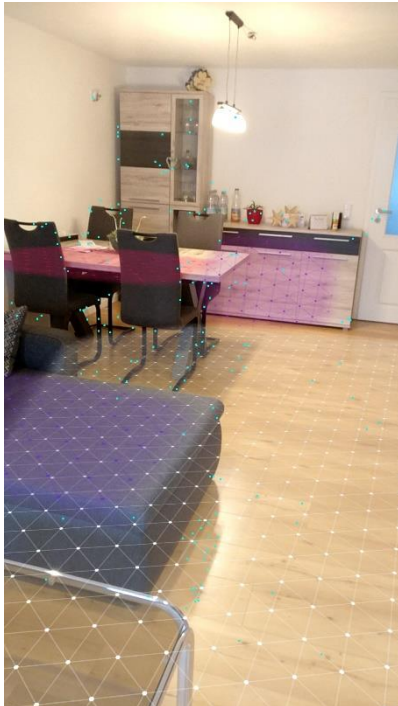


Abbildung 3.7 Screenshot der ARCore-Anwendung mit PlaneGenerator

Diese vier Aspekte genügen, um eine ARCore-Anwendung in Unity zu erstellen.

Zusätzlich wirbt Google bei dieser Software noch mit der „light estimation“. Auch hierfür gibt es ein Prefab. Es nennt sich „Environmental Light“. Das Skript, mit dem dieses Prefab verknüpft ist, schätzt das Licht jedes Pixels bei der Bilderfassung und erstellt daraus einen Durchschnittswert. Dieser Wert wird in der Variable `_GlobalColorCorrection` gespeichert und wird jedem Shader der Szene hinzugefügt. So wird das Licht für jedes virtuelle Objekt an das Umgebungslicht angepasst.

Die Funktion, um die virtuellen Objekte im Raum zu platzieren, befindet sich im `ExampleController`. Daher muss diesem Controller das Ziel-Objekt zugewiesen werden, welches zur Interaktion zur Verfügung steht. Mit einem Event-System vom Typ „Standalone Input Module“ können die Klicks (bei mobilen Endgeräten Taps) des Nutzers abgefangen werden.

Mit Hilfe eines Raycasters<sup>8</sup> wird anschließend überprüft, mit welcher Ebene die Klick-Position kollidiert, und anschließend wird das Objekt an der berechneten Stelle positioniert (Abbildung 3.8). Ein Auszug dieses Skripts wird in Listing 3.1 dargestellt. Wie die Anwendung mit einigen visuellen Objekten aussieht, kann Abbildung 3.9 entnommen werden.

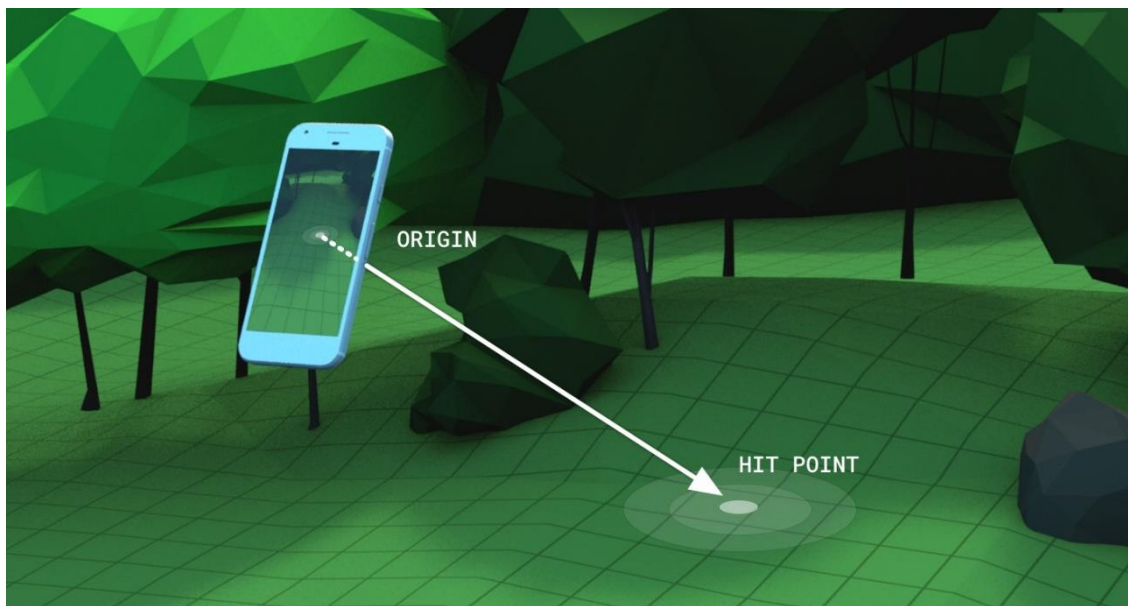


Abbildung 3.8 Visualisierung des Raycasts / Hit-Tests<sup>9</sup>

---

<sup>8</sup> Beim Raycasting wird von einem zweidimensionalen Punkt aus ein dreidimensionaler Vektor erstellt (Ray). Von diesem Richtungsvektor wird ein imaginärer Strahl erstellt und überprüft, ob dieser Schnittpunkte mit bestimmten angegebenen Elementen (in diesem Fall erkannte Ebenen in der Umgebung) hat. Diese Methode wird oft auch „hit test“ genannt. (vgl. Google Inc 2018)

<sup>9</sup> Quelle: <https://codelabs.developers.google.com/codelabs/ar-with-webxr/#4>

```

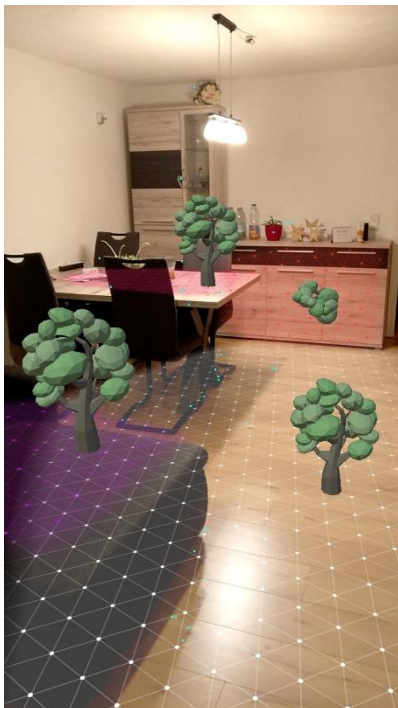
if (Frame.Raycast(touch.position.x,touch.position.y,raycastFilter,out hit)) {
    if ((hit.Trackable is DetectedPlane) &&
        Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
            hit.Pose.rotation * Vector3.up) < 0) {
        Debug.Log("Hit at back of the current DetectedPlane");
    } else {
        GameObject prefab;
        if (hit.Trackable is FeaturePoint) {
            prefab = AndyPointPrefab;
        } else {
            prefab = AndyPlanePrefab;
        }

        var treeObject = Instantiate(prefab, hit.Pose.position, hit.Pose.rotation);
        treeObject.transform.Rotate(0, k_ModelRotation, 0, Space.Self);

        var anchor = hit.Trackable.CreateAnchor(hit.Pose);
        treeObject.transform.parent = anchor.transform;
    }
}

```

*Listing 3.1 Raycast (Hit-Test) mit ARCore in Unity (C#)*



*Abbildung 3.9 Screenshot der ARCore-Anwendung mit visuellen Objekten*

Mit ARCore in Unity sind auch markerbasierende Anwendungen möglich. Dazu wird, wie auch bei Vuforia, eine Bilddatenbank benötigt. Um diese zu erstellen, wird ein Ordner mit den Bildern im Projekt angelegt. Anschließend werden alle Bilder markiert,

und über einen Rechtsklick auf die markierten Bilder kann automatisiert eine Datenbank angelegt werden. Diese Automation wird durch das ARCore-Package mitgeliefert (Abbildung 3.10).

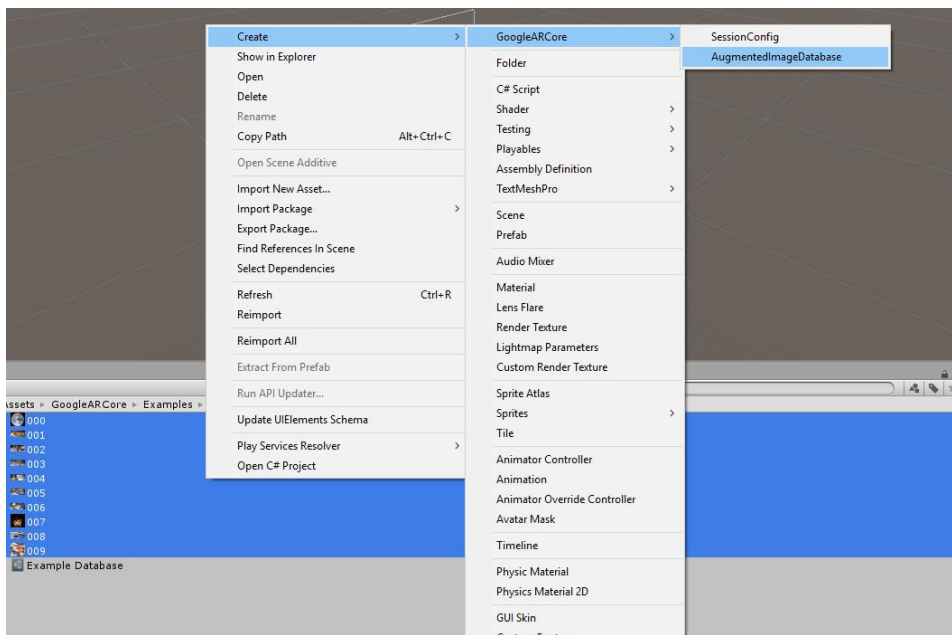


Abbildung 3.10 ARCore ImageDatabase erstellen in Unity

Dem ARCore Device muss dann diese Datenbank noch zugewiesen werden. Dazu wird die „AugmentedImagesSessionConfig“ bearbeitet, wie in Abbildung 3.11 abgebildet.

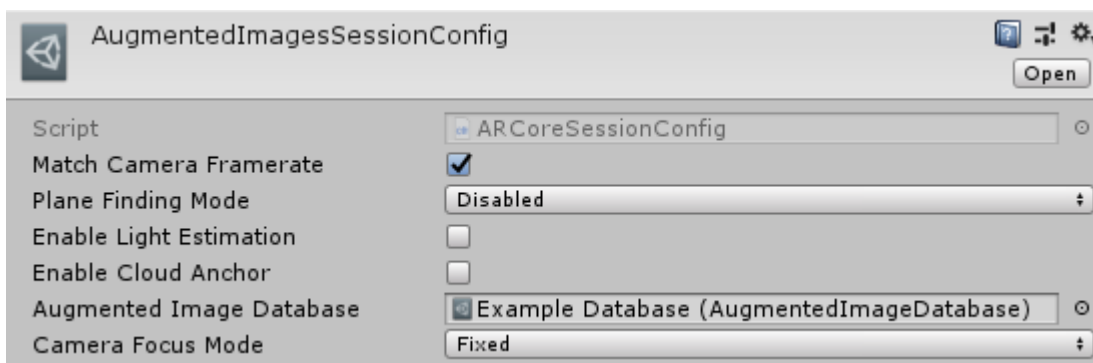


Abbildung 3.11 AugmentedImagesSessionConfig von ARCore

In diesem Fall ist das notwendige Prefab für den ExampleController das „AugmentedImageVisualizer“-Prefab. Dieses wird durch das Package implementiert und

ist dafür zuständig, dass die Marker erkannt werden. Teile dieses Skripts werden in Listing 3.2 gezeigt. Diesem Visualizer muss dann wiederum ein Skript zugewiesen werden, welches das gewünschte Objekt über den erkannten Marker legt (Listing 3.3).

```
foreach (var image in m_TempAugmentedImages) {  
    AugmentedImageVisualizer visualizer = null;  
    m_Visualizers.TryGetValue(image.DatabaseIndex, out visualizer);  
    if (image.TrackingState == TrackingState.Tracking && visualizer == null) {  
        // Create an anchor to ensure that ARCore keeps tracking this  
        // augmented image.  
        Anchor anchor = image.CreateAnchor(image.CenterPose);  
        visualizer = (AugmentedImageVisualizer)  
        Instantiate(AugmentedImageVisualizerPrefab, anchor.transform);  
        visualizer.Image = image;  
        m_Visualizers.Add(image.DatabaseIndex, visualizer);  
    }  
    else if (image.TrackingState == TrackingState.Stopped && visualizer != null) {  
        m_Visualizers.Remove(image.DatabaseIndex);  
        GameObject.Destroy(visualizer.gameObject);  
    }  
}
```

*Listing 3.2 Erkennung von Bild-Markern in Unity mit ARCore (C#)*

```
public AugmentedImage Image;
public GameObject FrameLowerLeft;
public GameObject FrameLowerRight;
public GameObject FrameUpperLeft;
public GameObject FrameUpperRight;

public void Update() {
    if (Image == null || Image.TrackingState != TrackingState.Tracking) {
        FrameLowerLeft.SetActive(false);
        FrameLowerRight.SetActive(false);
        FrameUpperLeft.SetActive(false);
        FrameUpperRight.SetActive(false);
        return;
    }

    float halfWidth = Image.ExtentX / 2;
    float halfHeight = Image.ExtentZ / 2;
    FrameLowerLeft.transform.localPosition = (halfWidth * Vector3.left) +
        (halfHeight * Vector3.back);
    FrameLowerRight.transform.localPosition = (halfWidth * Vector3.right) +
        (halfHeight * Vector3.back);
    FrameUpperLeft.transform.localPosition = (halfWidth * Vector3.left) +
        (halfHeight * Vector3.forward);
    FrameUpperRight.transform.localPosition = (halfWidth * Vector3.right) +
        (halfHeight * Vector3.forward);

    FrameLowerLeft.SetActive(true);
    FrameLowerRight.SetActive(true);
    FrameUpperLeft.SetActive(true);
    FrameUpperRight.SetActive(true);
}
```

*Listing 3.3 Platzierung virtueller Objekte auf einem Marker in Unity mit ARCore (C#)*

### 3.2.3 Ergebnisse dieser Untersuchungen

Anhand der hier beschriebenen Recherchen für die Unity-Plugins lässt sich einiges für die Entwicklung von AR-Anwendungen im Web ableiten. Eine der wichtigsten Funktionen für die AR-Entwicklung ist die AR-Kamera, denn die reale Umgebung des Nutzers muss gezwungenermaßen mit einer Kamera dargestellt werden. Sowohl in Vuforia als auch in ARCore erhält die Kamera einige Konfigurationen. Dazu gehören unter anderem die Festlegung einer Bilddatenbank und die Festlegung der Tracking-Methode (bei ARCore). Mit einem Controller - egal ob ImageTargetBehaviour (Vuforia), AugmentedImage (ARCore) oder einem markerlosen AR-Controller (ARCore) – also einem Skript, wird anschließend das Tracking übernommen. Die Skripte bzw. Prefabs dazu werden jeweils von den Plugins zur Verfügung gestellt. Das sollte daher auch das Ziel in einem Web-

Editor sein. Wie mit dem Ergebnis dieser Skripte anschließend umgegangen wird, sollte dann jeder Entwickler selbst entscheiden können. Den Skripten müssen nur das Ziel-Objekt und die AR-Kamera zugewiesen werden, damit die Ziel-Objekte nach erfolgreichem Tracking in der richtigen Relation zum Nutzer (also zur Kamera) angezeigt werden können. Da in dieser Arbeit ARCore zum Einsatz kommen soll, ist die abschließende Erkenntnis zu diesem Kapitel, dass die Web-Anwendung auf jeden Fall an die Struktur von ARCore angelehnt werden soll. Damit besteht die Möglichkeit das Umgebungslicht zu beeinflussen. Ein wichtiger Aspekt, der beim Beispiel des Vuforia-Plugins außen vor bleibt, ist die Nutzerinteraktion. Bei markerbasierenden Anwendungen kann der Nutzer fast nur mit den Markern interagieren, jedoch nicht mit der Anwendung direkt. Bei markerlosen Anwendungen ist eine Interaktion mit der Anwendung jedoch zwingend erforderlich. Zusammengefasst werden also für eine markerlose AR-Anwendung folgende Objekte benötigt:

- AR-Kamera
  - AR Einstellungen (Tracking)
- Tracking-Skript
  - AR-Kamera / Position des Nutzers
  - Ziel-Objekt
- Interaktionsmöglichkeit / Event-System
- Umgebungslicht (optional)

### 3.3 Vorstellung vorhandener Web-API für AR

#### 3.3.1 Three.js und three.ar.js

Three.js ist eine JavaScript Bibliothek, die es mit Hilfe von Canvas 2D, SVG, CSS3D und WebGL ermöglicht 3D-Content in Websites zu integrieren. Dabei sind Objekte wie eine Kamera, eine Szene, ein Renderer, Geometrie, Material und Mesh vorhanden, die auch in



3D-Programmen zum Einsatz kommen. (Dieser Abschnitt vgl. Cabello, GitHub. three.js 2019)

In Listing 3.4 wird eine einfache, kleine three.js-Anwendung dargestellt.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);

camera.position.z = 5;

function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
animate();
```

*Listing 3.4 Beispiel einer einfachen three.js-Anwendung*

Für einen 3D-Kontext ist immer eine Szene, eine Kamera und ein Renderer notwendig. Nur mit diesen Bestandteilen ist es möglich dem Nutzer den Inhalt zu zeigen. Das Renderer-Objekt erstellt ein `canvas`-Element. In diesem Element wird das abgebildet, was die Kamera in der Szene beobachtet. In diesem Fall wird der Szene ein grüner Würfel der Größe 1 hinzugefügt. Anschließend wird die Kamera so positioniert, dass diese den Würfel sehen kann. In der `animate()`-Funktion wird dann das Bild gerendert, damit es im `canvas` angezeigt werden kann. `requestAnimationFrame()` sorgt dafür, dass für jedes Frame diese Funktion erneut aufgerufen wird. Soll sich der Würfel jetzt zum Beispiel noch drehen, werden in der `animate()`-Funktion die entsprechenden Parameter angepasst (Listing 3.5).

```
cube.rotation.x += 0.01;  
cube.rotation.y += 0.01;
```

*Listing 3.5 Animation eines Würfels in three.js*

(Dieser Abschnitt vgl. Cabello, three.js / docs kein Datum)

Das Google AR-Team, unter anderem bestehend aus Clayton Wilkinson, Fred Sauer, Ian Muldoon und Jordan Santell, hat zu dieser Bibliothek eine Helfer-Bibliothek erstellt, die es ermöglicht AR-Content mit three.js zu erstellen. Diese Bibliothek funktioniert allerdings nur in zwei experimentellen Apps: WebARonARKit (für iOS) und WebARonARCore (für Android). Diese beiden Apps nutzen eine Erweiterung für die „WebVR 1.1 API“<sup>10</sup> um AR im Web zu ermöglichen. (Dieser Abschnitt vgl. Wilkinson, Sauer, et al., GitHub. Google AR/three.ar.js 2018)

Es gibt einige Beispiele zur Verwendung von three.ar.js. Ein Beispiel für eine markerbasierende Anwendung vom Unternehmen Google wird in Anhang 1 dargestellt. Die wichtigsten Funktionen für die Erkenntnisse in dieser Arbeit werden in diesem Abschnitt kurz erläutert. Zunächst sollte immer überprüft werden, ob das Endgerät für die Darstellung von AR geeignet ist. Dazu wird die Funktion `getARDisplay()` verwendet. Bei Erfolg der Funktion wird die Anwendung initialisiert (Listing 3.6).

---

<sup>10</sup> Die WebVR API ermöglicht den Zugriff auf VR-Geräte und Sensoren, sowie Head-Mounted-Displays im Web. (Vgl. Vukicevic, et al. 2017)

```
THREE.ARUtils.getARDisplay().then(function (display) {  
  if (display) {  
    if (display.getMarkers) {  
      vrDisplay = display;  
      init();  
    } else {  
      THREE.ARUtils.displayUnsupportedMessage("This prototype browser app for  
        WebAR does not support marker detection yet.");  
    }  
  } else {  
    THREE.ARUtils.displayUnsupportedMessage();  
  }  
});
```

*Listing 3.6 Initialisierung eines AR-Displays mit three.ar.js*

Bei der Initialisierung muss, wie in Listing 3.7 abgebildet, die Marker-Art festgelegt werden, eine `ARView` und eine AR-Kamera zur Szene (Listing 3.8) und die `THREE.VRControls` zur Kamera (Listing 3.9) hinzugefügt werden. Die `ARView` ist für das Rendering der Szene verantwortlich. Mit der `ARPerspectiveCamera` wird sichergestellt, dass die Kamera-Ansicht des Endgeräts mit der angezeigten AR-Ansicht übereinstimmt und die `VRControls` sorgen dafür, dass die Position und Ausrichtung der Kamera mit der Kamera-Ansicht während der Laufzeit übereinstimmt. Dieses Objekt kann ohne Schwierigkeiten mit seinen Funktionen von VR auf AR übertragen werden, da hierbei grundsätzlich egal ist welches Endgerät verwendet wird: Ein Nutzungsgerät muss sich in seiner Position und Ausrichtung mit einer Kamera-Ansicht synchronisieren.

```
var datGUI = new dat.GUI();  
gui = new GUI();  
datGUI.add(gui, "markerTypeString", ["AR",  
  "QRCODE"]).onFinishChange(function (value) {  
  if (!vrDisplay) {  
    return;  
  }  
  if (value === "QRCODE") {  
    gui.markerType = vrDisplay.MARKER_TYPE_QRCODE;  
  }  
  else if (value === "AR") {  
    gui.markerType = vrDisplay.MARKER_TYPE_AR;  
  }  
}).name("Marker type");
```

*Listing 3.7 Initialisierung der Marker-Arten in three.ar.js*

```
arView = new THREE.ARView(vrDisplay, renderer);  
camera = new THREE.ARPerspectiveCamera(vrDisplay, 60, window.innerWidth /  
window.innerHeight, 0.01, 100);
```

*Listing 3.8 Initialisierung der ARView und AR-Kamera in three.ar.js*

```
vrControls = new THREE.VRControls(camera);
```

*Listing 3.9 THREE.VRControls zur AR-Kamera hinzufügen*

In der `update()`-Funktion werden dann üblicherweise die Renderer aufgerufen.

Außerdem wird hier auch das Objekt auf die Position des Markers gesetzt, sobald dieser erkannt wurde (Listing 3.10).

```
var markers = vrDisplay.getMarkers(gui.markerType, MARKER_SIZE_IN_METERS);  
if (markers.length > 0) {  
    model.matrix.fromArray(markers[0].modelMatrix);  
    model.matrix.decompose(model.position, model.quaternion, scale);  
}
```

*Listing 3.10 Erkennen eines Markers in three.ar.js*

(Dieses Kapitel vgl. Google Inc. 2017)

### 3.3.2 AR.js und A-Frame

AR.js ist eine markerbasierende JavaScript-AR-Bibliothek. In Abbildung 3.12 wird die Demo-Anwendung von AR.js dargestellt, wobei auf dem Marker ein Würfel mit einer Schlange darin abgebildet wird. Für das Tracking verwendet diese Bibliothek das SDK „ARToolkit“<sup>11</sup>.

---

<sup>11</sup> ARToolkit ist ein Open-Source-Projekt, das im Herbst 2017 zu ARToolkitX umbenannt wurde. Es ist ein SDK, welches die Entwicklung von AR- und MR-Anwendungen für MacOS, iOS, Android und Linux ermöglicht. In der nächsten Version soll auch ARCore in das ARToolkitX integriert werden, daher ist das aktuell leider noch nicht der Fall. (Vgl. Lamb, et al. 2018)



Abbildung 3.12 Screenshot einer Anwendung mit *ar.js*

In den meisten Beispielen wird *AR.js* zusammen mit *A-Frame* verwendet. Es ist auch möglich *AR.js* mit *three.js* zu verwenden.

*A-Frame* ist ein Framework um VR-Experimente mit den Webtechnologien und damit für das Web zu erstellen. Dabei wird auf die Funktionen von *WebVR* zurückgegriffen, wie es auch bei *three.js* der Fall ist. *A-Frame* erweitert die HyperText Markup Language (HTML) um eigene Elemente. Mit diesen Elementen hat der Entwickler n erster Linie die Möglichkeit VR-Kontext zu kreieren. In Zusammenspiel mit *AR.js* wird damit natürlich AR-Kontext erstellt. Ein einfaches Beispiel für eine AR-Anwendung mit *A-Frame* und *AR.js* wird in Listing 3.11 dargestellt.

```

<!-- Implementierung von a-frame -->
<body style='margin : 0px; overflow: hidden;'>
  <a-scene embedded arjs>
    <a-marker preset="hiro">
      <a-box position='0 0.5 0' material='color: black;'></a-box>
    </a-marker>
    <a-entity camera></a-entity>
  </a-scene>
</body>

```

Listing 3.11 Einfache A-Frame AR-Anwendung mit AR.js

Mit `<a-entity camera>` wird die Umgebung, also die Kamera-Ansicht abgebildet. Diese befindet sich, wie auch ein Marker `<a-marker preset="hiro">` in einer Szene, die wiederum AR.js implementiert: `<a-scene embedded arjs>`. Das Attribut „preset“ in `<a-marker>` bestimmt, welcher Marker verwendet werden soll: in diesem Fall ein Marker mit der id „hiro“. Wird der Marker durch das Tracking erkannt, wird der Inhalt des Marker-Elements dargestellt: im gezeigten Beispiel eine schwarze Box.

Auf die detaillierte Beschreibung von AR.js mit three.js wird im Umfang dieser Arbeit verzichtet, da three.js mit three.ar.js bereits im vorherigen Kapitel beschrieben wurde. Zur Vollständigkeit wird ein Code-Beispiel in Anhang 2 beigefügt. Dabei wird über das Objekt mit dem Namen THREEx direkt auf das SDK ARToolKit zugegriffen. Ein Beispiel für einen solchen Zugriff wird in Listing 3.12 aufgeführt. (Dieses Kapitel vgl. Etienne 2018)

```

arToolkitSource = new THREEx.ArToolkitSource({
  sourceType : 'webcam',
});

```

Listing 3.12 Zugriff auf ARToolKit mit AR.js und three.js

### 3.3.3 Tracking.js

Tracking.js beschreibt sich selbst nicht als AR-Bibliothek, sondern als Bibliothek, die verschiedene Algorithmen und Technologien für maschinelles Sehen ermöglicht:

*„The tracking.js library brings different computer vision algorithms and techniques into the browser environment.“ (Lundgren, Rocha, et al., tracking.js kein Datum)*

Mit tracking.js ist es möglich Farben oder Gesichter bzw. Objekte zu erkennen.

Außerdem können eigene Tracking-Algorithmen integriert werden. Das Referenz-Objekt kann dabei sowohl ein Bild als auch ein Video oder das Rendering einer Kamera sein. Da es möglich ist eine Kamera-Ansicht mit dieser Bibliothek zu tracken, kann tracking.js also ohne weitere Schwierigkeiten für AR-Anwendungen verwendet werden.

Um zum Beispiel einen ObjectTracker mit tracking.js zu verwenden, sind nur wenige Zeilen Code notwendig. Zunächst muss ein HTML-Video-Element erstellt werden, in dem dann das Kamera-Bild zu sehen ist, welches, wie in Listing 3.13 zu sehen, über den JavaScript-Navigator einbaubar ist. Das HTML-Video-Element hat in diesem Beispiel die id „myVideo“ (Listing 3.14).

```
navigator.mediaDevices.getUserMedia({video: true}).then(function(stream) {  
  document.getElementById('myVideo').src = URL.createObjectURL(stream);  
}).catch(function() {  
  alert('could not connect stream');  
});
```

*Listing 3.13 Implementierung der Nutzer-Kamera-Ansicht über JavaScript*

```
<video id="myVideo" width="400" height="300" preload autoplay loop  
muted></video>
```

*Listing 3.14 HTML-Video-Element für die Kamera-Ansicht mit JavaScript*

Zum Tracken der Objekte müssen zunächst Daten eingelesen werden bzw. das System auf bestimmte Objekte trainiert werden. Standardmäßig stehen in tracking.js Gesichter, Münder und Augen zur Verfügung. Diese Trainingsdaten müssen zusätzlich implementiert werden, da diese nicht zwingend im Einzelnen, jedoch in der Masse große Datenmengen mit sich bringen und die Performance darunter leiden könnte. Daher gilt: Nur die tatsächlich benötigten Trainingsobjekte integrieren.

Nachdem der `ObjectTracker` mit den angegebenen Daten initialisiert wurde, können, wie in Listing 3.15 dargestellt, über einen `EventListener` die Daten abgefangen werden.

```
var objects = new tracking.ObjectTracker(['face', 'eye', 'mouth']);
objects.on('track', function(event) {
  if (event.data.length === 0) {
    // No objects were detected in this frame.
  } else {
    event.data.forEach(function(rect) {
      // rect.x, rect.y, rect.height, rect.width
    });
  }
});
tracking.track('#myVideo', objects);
```

Listing 3.15 Verwendung des tracking.js-Tracking-Objekts

Im Objekt `rect`, welches vom `EventListener` bei der Erkennung der Objekte zurückgegeben wird, steht die Information, an welcher Position und wie groß die Fläche des erkannten Objekts ist. (Dieses Kapitel vgl. Lundgren, Rocha, et al., tracking.js. Trackers kein Datum)

### 3.3.4 WebXR

In den vorherigen beschriebenen API war das eine oder andere Mal die Rede von WebVR, einer API, die den Zugriff auf VR-Geräte und Sensoren sowie Head-Mounted Displays im Web ermöglicht. (Vgl. Vukicevic, et al. 2017)

WebVR wird derzeit um einige Komponenten erweitert und nennt sich seither WebXR. Das X steht dabei nicht für ein Akronym, wie es zum Beispiel bei AR der Fall ist, sondern es stellt eine algebraische Variable dar, die sowohl mit einem A für augmented als auch mit einem V für virtual ersetzt werden kann (vgl. Cannon, et al. 2019). Mit WebXR soll die Integration von AR in das World Wide Web ermöglicht werden:

*„This specification describes support for accessing virtual reality (VR) and augmented reality (AR) devices, including sensors and head-mounted displays, on the Web.“ (Jones und Waliczek 2019)*

WebXR ist eine Erweiterung der Browser-Funktionalität (geschrieben in C++ und JavaScript) und erweitert damit das JavaScript-Navigator-Objekt um ein Objekt mit dem Namen „xr“. Über dieses Objekt wird eine Reihe von Funktionen und Objekten implementiert, die dann den Zugriff auf XR-Endgeräte und -Sensoren ermöglichen. Zum Zeitpunkt dieser Arbeit befindet sich die WebXR-Spezifikation in einem nicht stabilen



und sich stetig ändernden Zustand. In diesem Kapitel wird die Vorgehensweise vom 07. Februar 2019 beschrieben – bei der beigefügten und in Kapitel 5.1 beschriebenen Anwendung musste jedoch auf eine ältere Version aus dem Dezember 2018 zurückgegriffen werden. Der Lebenszyklus einer XR-Anwendung sollte sich jedoch nicht großartig verändern.

Um WebXR nutzen zu können, muss zunächst überprüft werden, ob das Endgerät die entsprechenden Voraussetzungen dafür erfüllt. Da WebXR das Navigator-Objekt erweitert, kann hierzu einfach überprüft werden, ob diese Erweiterung vorhanden ist. Außerdem muss überprüft werden, ob der Chrome-Flag für den Hit-Test gesetzt wurde. Anschließend wird mit `supportsSessionMode("inline")` überprüft, ob der im Programm benötigte Modus (also „immersive-vr“ [VR-Brille], „immersive-ar“ [AR-Brille] oder „inline“ [Standard-Kamera-Ansicht]) mit dem XRDevice möglich ist.

```
if (navigator.xr && XRSession.prototype.requestHitTest) {  
  return new Promise((resolve, reject) => {  
    navigator.xr  
      .supportsSessionMode("inline")  
      .then(() => {  
        resolve(new ReturnMessage(true, "Can create XR Session"));  
      })  
      .catch(error => {  
        reject(this.onNoXRDevice(error.message));  
      });  
  });  
}
```

*Listing 3.16 Abfrage des XR-Devices, wenn Voraussetzungen für WebXR erfüllt sind*

Aufgrund der Abfrage für die Berechtigungen der Kamera und die erhöhte Belastung der Grafikkarte, muss bzw. sollte der Nutzer AR immer manuell „aktivieren“. Hat der Nutzer die Berechtigungen zugelassen und hat er ein XR-fähiges Endgerät, kann eine XRSession, wie in Listing 3.17 abgebildet, erstellt werden.

```
const ctx: XRPresentationContext = this.xrCanvas.getContext("xrpresent");
return new Promise((resolve, reject) => {
  navigator.xr
    .requestSession({
      mode: "inline",
      outputContext: ctx,
      environmentIntegration: true })
    .then(session => {
      this.session = session;
      this.xrCanvas.classList.add("active");
      this.onSessionStarted();
      resolve(new ReturnMessage(true, "Started Session"));
    })
    .catch(err => {
      reject(this.onNoXRDevice(err.message));
    });
});
```

*Listing 3.17 Starten einer WebXR-XRSession*

Sobald die Session gestartet wurde, kann mit dem Aufbau der Szene begonnen werden. Dazu muss der XRSession ein „baseLayer“ zugewiesen werden. Verwendet wird in WebXR hierzu ein WebGLLayer. Diesem muss der Kontext eines WebGL-Renderers sowie die XRSession übergeben werden. Dieser „baseLayer“ wird zur Aktualisierung des Render-Status des Frames verwendet. Hierbei muss erneut sichergestellt werden, dass der Inhalt, der geschrieben werden soll, mit dem XRDevice kompatibel ist:

```
this.gl = this.xrCanvas.getContext("webgl");
return this.gl.makeXRCompatible().then(() => {
  this.session.updateRenderState({
    baseLayer: new XRWebGLLayer(this.session,
      this.gl)
  });
});
```

*Listing 3.18 WebXR Rendering mit WebGLLayer*

Die virtuell anzuzeigende Szene sollte ebenfalls nach dem Start der Session aufgebaut werden. Hierzu kann WebGL direkt oder eine Bibliothek, wie zum Beispiel three.js, verwendet werden. Anschließend kann dann Frame für Frame gerendert werden. Da WebXR nicht nur für Smartphones ausgerichtet ist, sondern vor allem auch für Head-Mounted-Displays, muss vor dem Rendering definiert werden, wie und in welchem Kontext das Frame benötigt wird. Hierbei wird von dem „ReferenceSpace“ gesprochen.

Der Funktionsaufruf, der die Einrichtung des Layers und die Konfiguration der virtuellen Welt beinhaltet, sieht wie folgt aus:

```
this.session
  .requestReferenceSpace({ type: "stationary", subtype: "eye-level" })
  .then((referenceSpace: XRReferenceSpace) => {
    this.referenceSpace = referenceSpace;
  })
  .then(() => {
    // Setup WebGL Context
    // Create virtual scene
  })
  .then(() => {
    this.session.requestAnimationFrame(this.onXRFrame);
  });
```

*Listing 3.19 WebXR – requestFrameOfReference() und requestAnimationFrame()*

Die Callback-Funktion des `requestAnimationFrame`-Aufrufs beinhaltet die benötigten Variablen, um „die Magie geschehen zu lassen“. In dieser Funktion wird durch einen rekursiven Aufruf immer das nächste Bild gerendert. Listing 3.20 zeigt eine mögliche Callback-Funktion. Zur einfacheren Darstellung wird für das Rendering der virtuellen Szene `three.js` verwendet.

```

onXRFrame(time: number, frame: XRFrame) {
  if (this.session) {
    let session = frame.session;
    let pose = frame.getViewerPose(this.referenceSpace);

    // Check for surfaces
    if (!this.surfaces) {
      this.checkForSurfaces(frame);
    }

    this.gl.bindFramebuffer(this.gl.FRAMEBUFFER,
      this.session.baseLayer.framebuffer);

    // Keep real world camera view and virtual world camera in sync
    if (pose) {
      for (let view of pose.views) {
        const viewport = session.baseLayer.getViewport(view);
        this.gl.viewport(viewport.x, viewport.y, viewport.width,
          viewport.height);
        this.renderer.setSize(viewport.width, viewport.height);
        this.camera.projectionMatrix.fromArray(view.projectionMatrix);
        const VIEW_MATRIX = new Matrix4().fromArray(view.viewMatrix);
        this.camera.matrix = VIEW_MATRIX;
        this.camera.updateMatrixWorld(true);
        this.renderer.render(this.scene, this.camera);
      }
    }
    // Queue up the next frame
    session.requestAnimationFrame(this.onXRFrame);
  }
}

```

Listing 3.20 WebXR requestAnimationFrame-Callback (mit three.js)

Das `frame`-Objekt vom Typ `XRFrame` beinhaltet je nach `XRDevice` bzw. je nach Angabe des Parameters in `requestReferenceSpace` eine oder mehrere Views. Head-Mounted-Displays benötigen je eine Ansicht für das linke und das rechte Auge. Daher müssen in einer Schleife alle Views abgearbeitet werden. In der `pose`-Variable wird die Position des `XRDevices` zum Zeitpunkt des Frames gespeichert. Nur wenn diese Position vorhanden ist, können die Ansicht der virtuellen und realen Kamera synchronisiert werden. Dazu wird natürlich auch die Größe des Renderers angepasst. Anschließend werden die Ansichten synchronisiert. Das heißt, die Projektions- und Ansichts-Matrizen der `XRDevice-View` werden auf die virtuelle Kamera übertragen. Danach kann die Szene mit dem WebGL-Renderer abgebildet werden.

Zur Interaktion mit der Szene können die JavaScript-Events verwendet werden. Mit dem sogenannten Hit-Test kann dann ein Objekt auf einer erkannten Oberfläche platziert werden. Dazu wird dem Event ebenfalls das `XRFrame` übergeben, mit dem die Klick-Position herausgefunden werden und anschließend an die Funktion `requestHitTest()` des `XRSession`-Objekts übergeben werden kann. Die Verwendung dieses Hit-Tests wird in Listing 3.21 dargestellt.

```
onClick(event) {
  let rayOrigin = new THREE.Vector3();
  let rayDirection = new THREE.Vector3();

  let inputPose = event.frame.getInputPose(event.inputSource,
    this.frameOfRef);
  let devicePose = event.frame.getDevicePose(this.frameOfRef);

  if (!inputPose) {
    return;
  }

  if (inputPose.targetRay) {
    rayOrigin.set(inputPose.targetRay.origin.x, inputPose.targetRay.origin.y,
      inputPose.targetRay.origin.z);
    rayDirection.set(inputPose.targetRay.direction.x,
      inputPose.targetRay.direction.y, inputPose.targetRay.direction.z);

    event.frame.session
      .requestHitTest(new Float32Array(rayOrigin.toArray()),
        new Float32Array(rayDirection.toArray()), this.frameOfRef)
      .then(results => {
        if (results.length > 0) {
          this.addARObjectAt(results[0].hitMatrix,
            devicePose.poseModelMatrix);
        } else {
          // Object could not be placed, no hits found
        }
      });
  }
}
```

Listing 3.21 Verwendung des Hit-Tests in WebXR

### 3.4 Kriterien für die Analyse

Nachdem die vorhandenen API in den vorherigen Kapiteln vorgestellt wurden, wird hier beschrieben, auf welche Kriterien bei der Auswahl der API ein besonderes Augenmerk gelegt wurde:

- Abhängigkeiten/ Basistechnologien

Unter diesem Punkt wird darauf geachtet, welche Technologien und Abhängigkeiten zu weiterer Software der API zu Grunde liegen. Müssen zum Beispiel zusätzliche Bibliotheken integriert werden, dass die API verwendet werden kann?

- Aktualität der Software

Die Software sollte in einer aktuellen Version verfügbar sein und auch in neuen Browsern funktionieren. Alte Software bringt oft Sicherheitsrisiken mit sich, ebenso steigt die Fehleranfälligkeit von alter, nicht mehr weiter entwickelter Software.

- Dokumentation

Die Dokumentation und der Support von Software ist für Entwickler von elementarer Bedeutung. Durch eine gut beschriebene Dokumentation kann ein Entwickler sich sehr schnell in einer neuen Umgebung zurechtfinden. Außerdem wird durch Foren- oder Blogbeiträge die Fehlersuche deutlich vereinfacht und Fehler, die eventuell schon von anderen gemacht wurden, können dadurch verhindert werden.

- Nutzererfahrung, Ersterfahrungen im Umgang mit der API

Der erste Eindruck zählt - auch bei Software. Entstehen zum Beispiel schon bei der Implementierung der API Schwierigkeiten, ist oft auch die Entwicklung zum Scheitern verurteilt. Das gleiche gilt, wenn das Starten von Demo-Anwendungen oder Beispielen zur Herausforderung wird.

- Technische Einschränkungen

Bei neuer Software und neuen Technologien ist es nicht selten der Fall, dass für die Nutzung einige technische Einschränkungen vorhanden sind. Gerade im Bereich der Entwicklung für mobile Endgeräte werden häufig neue Sensoren oder Funktionen verwendet, die in älteren Geräten noch nicht vorhanden sind.

- Umfang und Funktionen

Aus Kapitel 3.2 ergeben sich einige Funktionen für AR-Anwendungen, die auch in der Entwicklung von AR-Anwendungen im Web zur Verfügung stehen sollten. Es sollte möglich sein, diese in gleicher oder ähnlicher Form zu integrieren.

- Zielerreichbarkeit

In Kapitel 1.2 wird beschrieben, was das Ziel dieser Arbeit und damit auch das Ziel dieser Anwendung ist. Bei der Auswahl der Software ist es daher wichtig zu überprüfen, ob es möglich ist, das Ziel zu erreichen. In dieser Arbeit muss die Anwendung die Einbindung von ARCore ermöglichen, ebenso wie die Integration in Electron möglich sein muss. Außerdem soll mit der Anwendung in erster Linie markerloses Tracking implementiert werden.

### 3.5 Beschreibung der Web-API bezüglich der Kriterien

In Tabelle 3.1 wird eine Übersicht der Web-API in Verbindung mit den Kriterien dargestellt. In den folgenden Abschnitten wird die Tabelle genauer erläutert. Das Ergebnis der Bewertung wird in Kapitel 3.6 „Auswertung der Analyse“ detailliert beschrieben. Die Tabelle ist außerdem in Anhang 3 verfügbar.

Kriterium	three.ar.js <sup>12</sup>	AR.js <sup>13</sup>	tracking.js <sup>14</sup>	WebXR <sup>15</sup>
<b>Aktualität</b>	2018-02-26	2018-12-02	2018-05-17	2019-01-15
<b>Basis-technologien</b>	WebGL, WebVR, three.js	WebGL, ARToolKit, three.js oder A-Frame, [WebVR]	-	WebVR, WebGL
<b>Dokumentation</b>	Sehr gut	Okay	Sehr gut	Sehr gut (noch nicht stabil)
<b>Nutzererfahrungen</b>	Schlecht	Okay	Sehr gut	Sehr gut
<b>Technische Einschränkungen</b>	Browser: WebARonARCore, WebARonARKit Endgeräte: Google Pixel 1 + 2, Samsung Galaxy S8	-	-	Browser: Chrome Dev, Chrome Canary + 2 Chrome Flags Android: Version > 8.0
<b>Umfang / Funktionen</b>	AR-Kamera: ja Tracking: ja Referenz-Objekt: ja Event-System: ?	AR-Kamera: ja Tracking: ja Referenz-Objekt: nicht direkt Event-System: nein	AR-Kamera: ja Tracking: ja Referenz-Objekt: nicht direkt Event-System: ja	AR-Kamera: ja Tracking: ja (ARCore) Referenz-Objekt: nicht direkt Event-System: ja
<b>Zielerreichbarkeit</b>	ARCore: ja Markerlos: ja Electron: nein	ARCore: nein Markerlos: nein Electron: nein	ARCore: nein Markerlos: bedingt Electron: ja	ARCore: ja Markerlos: ja Electron: ja (in naher Zukunft)
<b>Erstveröffentlichung</b>	2017-08	2017-02	2012-06	2015-01

Tabelle 3.1 Übersicht der Web-API (Stand: 16.01.2019)

Diese Tabelle beruht auf Erfahrungen und Nachforschungen zu den einzelnen Themen, die im Zuge dieser Arbeit getätigt wurden.

<sup>12</sup> Informationen, die nicht auf eigener Erfahrung beruhen stammen von folgender Quelle:

<https://github.com/google-ar/three.ar.js/>

<sup>13</sup> Informationen, die nicht auf eigener Erfahrung beruhen stammen von folgender Quelle:

<https://github.com/jeromeetienne/AR.js/>

<sup>14</sup> Informationen, die nicht auf eigener Erfahrung beruhen stammen von folgender Quelle:

<https://github.com/eduardolundgren/tracking.js/>

<sup>15</sup> Informationen, die nicht auf eigener Erfahrung beruhen stammen von folgenden Quellen:

<https://github.com/immersive-web/webxr/> und <https://codelabs.developers.google.com/codelabs/ar-with-webxr/#1>



Bis auf `tracking.js` bauen alle API auf WebGL auf. Da dies der Web-Standard für 3D-Rendering-Content ist, ist das nicht weiter verwunderlich. `Three.ar.js`, sowie `WebXR` bauen auf `WebVR` auf, ebenso wie `AR.js` in der Verwendung mit `A-Frame`. `Tracking.js` hingegen ist ein reines JavaScript-Framework. Daher kann es problemlos in sämtliche webbasierten Programme integriert werden. Die Dokumentationen der einzelnen API sind ausführlich und leicht verständlich. Bei `AR.js` gibt es allerdings nicht direkt eine Dokumentation, dafür aber einen kurzen Anfänger-Kurs. Da hier `three.js` bzw. `A-Frame` verwendet werden muss und diese Dokumentationen ebenfalls sehr gut und leicht verständlich sind, sollte das jedoch nicht zu Schwierigkeiten führen.

Technische Einschränkungen gibt es bei `three.ar.js` und `WebXR`. Bei `WebXR` sind diese jedoch nur bedingt problematisch. Die Bedingungen für `three.ar.js` sind deutlich schlechter, denn es wird, je nach Endgerät, zwingend einer von zwei experimentellen Browsern (`WebARonARCore` für Android oder `WebARonARKit` für iOS) benötigt. `WebARonARCore` integriert zwar `ARCore`, jedoch scheint die Software nicht weiterentwickelt zu werden – die letzten Änderungen stammen vom Februar 2018 und waren daher schon beim Start dieser Arbeit ein halbes Jahr alt. Diese Browser wurden außerdem nur für eine geringe Anzahl von Endgeräten entwickelt. Bei Android beschränken sich diese auf das Google Pixel 1 und 2 sowie das Samsung Galaxy S8. (Dieser Abschnitt vgl. Wilkinson, Sauer, et al., GitHub. [google-ar/WebARonARCore](https://github.com/google-ar/WebARonARCore) 2018) Dadurch war es mit den zur Verfügung stehenden Mitteln bezüglich dieser Arbeit nicht möglich, `three.ar.js` zu testen.

Die Einschränkungen durch `WebXR` beziehen sich auf ein `ARCore`-fähiges Endgerät mit Version Android 8.0 oder später. Da `WebXR` sich derzeit in der Entwicklung und damit in einem nicht stabilen Zustand befindet, kann es auch nur in den Entwicklungsbrowsern von Chrome (Chrome Dev und Chrome Canary) mit zwei gesetzten Flags, deren Einstellungen über die URL `chrome://flags` aufgerufen und freigeschalten werden können, genutzt werden. Wie beschrieben, soll `WebXR` ein Browser-Standard werden,

womit diese Einschränkungen in Zukunft wohl ausbleiben werden.

Bei AR.js und tracking.js gibt es keine technischen Einschränkungen.

Der erforderliche Funktionsumfang wird von keinem der API direkt erfüllt. Allerdings ist es mit JavaScript durchaus möglich ein Event-System zu integrieren und mit WebGL oder einer Erweiterung dafür, wie three.js, ist es keine große Schwierigkeit ein Ziel-Objekt für das Tracking zu implementieren. Bezogen auf den Game-Editor FUDGE, für den diese Anwendung erstellt wird, ist es sogar von Vorteil, wenn kein Ziel-Objekt in einem bestimmten Format angegeben werden muss und dieses durch den Game-Editor bestimmt werden kann.

Die Funktionalität von ARCore wird bei zwei der analysierten Bibliotheken integriert: three.ar.js, der Browser integriert in diesem Fall ARCore für Android, und WebXR. Da diese beiden ARCore integriert haben, ist eine Erstellung von markerlosen AR-Anwendungen möglich. Markerbasierende Anwendungen sind derzeit mit three.ar.js, AR.js und tracking.js möglich.

Außer dem nicht gelungenen Testen der three.ar.js Bibliothek konnten weitere spannende Erfahrungen mit den API gemacht werden. Dazu gehört auch die (versuchte) Implementierung der Software in Electron. Ohne Weiteres ist nur die Implementierung von tracking.js in Electron möglich. Für die Verwendung von three.ar.js und AR.js sind Webserver notwendig. Da WebXR zur Zeit der Bearbeitung dieser Arbeit in einem nicht stabilen Zustand ist, jedoch – wie es auch bei WebGL der Fall ist - als Standard direkt in den Chromium-Browser integriert werden soll, ist es in absehbarer Zeit möglich, WebXR in Electron zu verwenden.

Bei AR.js war es, trotz der nicht vorhandenen technischen Einschränkungen, nicht möglich die Bibliothek zu verwenden. Es sind keine Probleme beim Erforschen von bereits bestehenden Anwendungen entstanden, das selbstständige Entwickeln von Anwendungen gestaltete sich allerdings als eine große Herausforderung. Daher hat diese API nur ein „okay“ in Sachen Nutzererfahrung verliehen bekommen. Die „schlechte“

Beurteilung von `three.ar.js` ist auf die oben beschriebene Problematik des nicht möglichen Testens zurückzuführen. Mit den Frameworks `tracking.js` und `WebXR` gab es keine Schwierigkeiten in der Kennenlern-Phase.

Die Aktualität der API kann nur bei `three.ar.js` in Frage gestellt werden. `WebXR` ist in dieser Disziplin der absolute Spitzenreiter. Im Vergleich zur Erstveröffentlichung sollte aber auch `tracking.js` Beachtung geschenkt werden, denn hier wurde fünf Jahre lang fleißig weiterentwickelt und ausgebessert.

### 3.6 Auswertung der Analyse

Jede der, auf den vergangenen Seiten, beschriebenen Software hat sowohl Vor- als auch Nachteile. Trotzdem konnte durch die Zielsetzung dieser Arbeit recht eindeutig eine Entscheidung getroffen werden.

Durch die Schwierigkeiten beim Ausführen der Beispiel-Anwendungen von `three.ar.js` und den anschließend auftretenden Problemen bei der Erstellung einer eigenen Anwendung konnte diese Software nicht für diese Arbeit verwendet werden. Dies ist jeweils hauptsächlich auf die technischen Einschränkungen durch die Verwendung der experimentellen Browser zurückzuführen.

Ähnlich ist es mit `AR.js` verlaufen. Dabei konnten die Demo-Anwendungen ohne Schwierigkeiten im Web angesehen werden, die Entwicklung einer AR-Anwendung mit `A-Frame` ist jedoch nicht auf Anhieb gelungen. Für die Entwicklung mit `A-Frame` oder `three.js` sind jeweils Webserver notwendig (vgl. Marcos, McCurdy und Ngo kein Datum, Pettit 2014), damit gestaltet sich die Einbindung von `AR.js` in `Electron` eher schwierig. Es können in `Electron` zwar `node-Server` verwendet werden, wie und ob das in Zusammenspiel mit `AR.js` jedoch funktioniert konnte im Rahmen dieser Arbeit nicht genauer untersucht werden.

`Tracking.js` kann hingegen sehr einfach in `Electron` integriert werden. Markerloses Tracking ist allerdings nur mit selbstgeschriebenen Algorithmen möglich, damit ist auch

ARCore nicht eingebunden. Für eine Anwendung wie Snapchat ist dieses Tool sicher in die engere Auswahl zu nehmen, da es auch sehr schlank ist – für einen AR integrierenden Game-Editor ist das jedoch eher untauglich.

Damit steht die Entscheidung fest: Die am besten geeignete Software ist WebXR. Die Verwendung von WebXR wird als sehr gute Ausgangsbasis betrachtet, da WebXR zum einen der Standard für Web in Bezug auf AR- und VR-Anwendungen werden soll, sowie die Basis für einen Großteil der anderen Bibliotheken darstellt. Mit WebXR steht die zu verwendende Software ARCore zur Verfügung und auch die Implementierung in Electron scheint durch die direkte Verknüpfung mit Chromium kein Problem zu sein. Der einzige Nachteil ist, dass sich diese Spezifikation derzeit im Entwicklungszustand befindet und dadurch einige Überraschungen während der Programmierung der Anwendung entstehen könnten.

## 4 Konzeption der Anwendung

### 4.1 Anforderungen

Aus der Analyse und dem Ziel dieser Arbeit ergeben sich einige Anforderungen, welche die Anwendung erfüllen soll. Diese werden in diesem Abschnitt zusammengefasst.

Der erste Punkt auf dieser Liste der technischen Anforderungen ist der Game-Editor FUDGE. Diese Anwendung soll so programmiert werden, dass sie zu einem späteren Zeitpunkt in FUDGE integriert werden kann. Dazu ist es notwendig, dass die Anwendung in den Webtechnologien entwickelt wird und in Electron funktioniert. Außerdem soll nicht reines JavaScript, sondern TypeScript in der Entwicklung verwendet werden.

Da für eine AR-Anwendung ein mobiles Endgerät benötigt wird, muss die Anwendung aus Electron heraus als App gespeichert werden können. Auch diese müssen dann in den Webtechnologien geschrieben sein. Daher bietet sich die Verwendung von Adobe PhoneGap an.

Ein Ziel dieser Arbeit ist es, die Software ARCore in die Anwendung zu integrieren, um markerloses Tracking zu ermöglichen. Daraus und aus dem Kapitel 3 „Analyse vorhandener API für AR“ geht die Verwendung von WebXR, als Hilfsmittel für die Verbindung zwischen ARCore und den Webtechnologien, hervor. Für die Darstellung der virtuellen Objekte kann eine beliebige Bibliothek (z. B. three.js) oder aber auch eine eigenständig programmierte (FUDGE-)3D-Bibliothek verwendet werden. Zugrunde liegen muss dieser nur WebGL, da WebXR darauf zurückgreift.

## 4.2 Das Konzept

### 4.2.1 Technologien und Zusammenhänge

In diesem Kapitel wird kurz erläutert, wie die verschiedenen genannten Technologien zusammenhängen. Zum besseren Verständnis werden die Zusammenhänge in Abbildung 4.1 dargestellt.

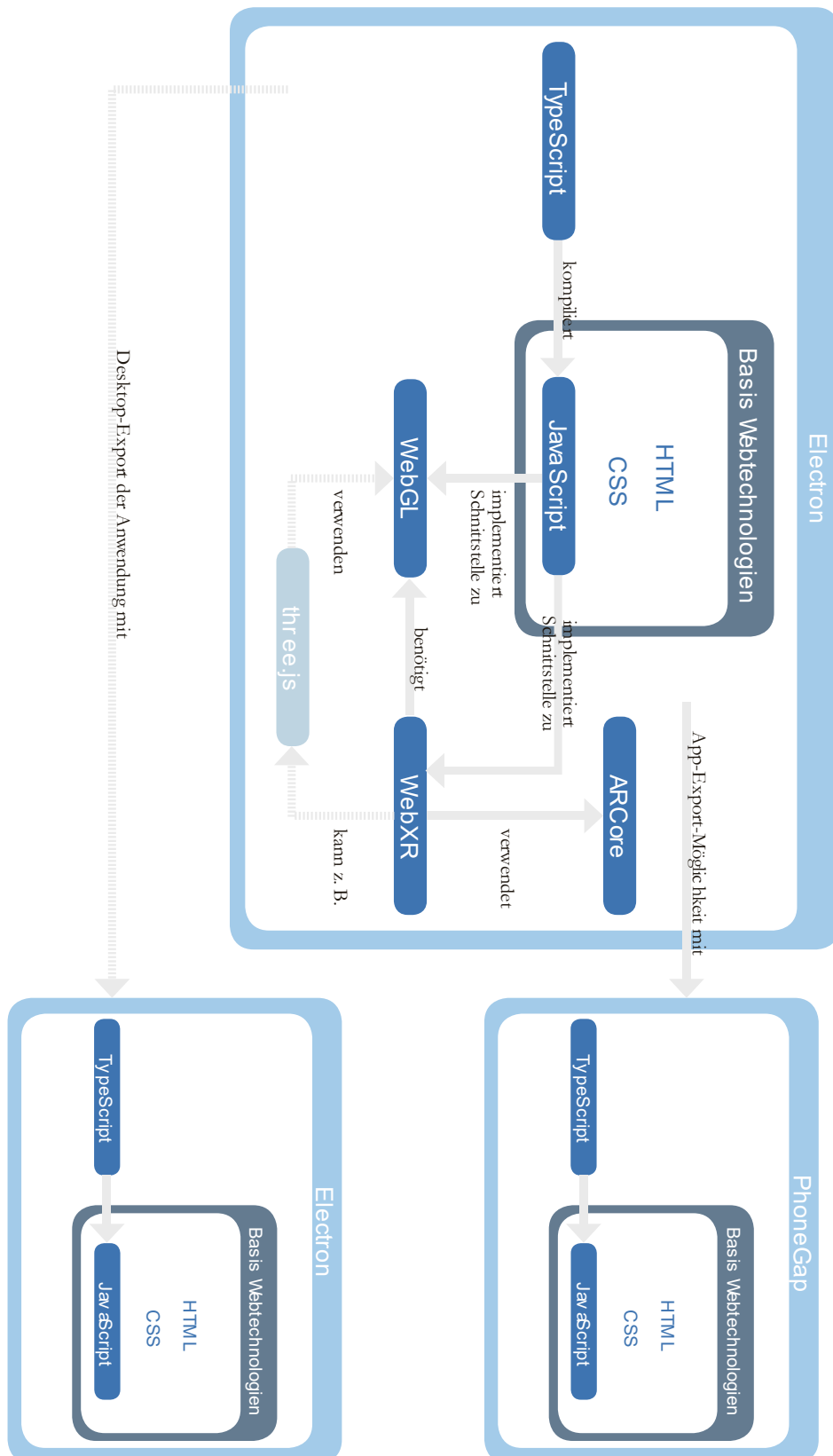


Abbildung 4.1 Zusammenhänge der verwendeten Technologien

Das Grundgerüst für die gesamte Anwendung bildet Electron. Innerhalb von Electron bzw. bei der Programmierung mit Electron stehen den Entwicklern die Webtechnologien HTML, CSS und JavaScript zur Verfügung. Um den Umgang mit JavaScript zu vereinfachen und um objektorientierte Programmierung in JavaScript zu vereinfachen sowie die Typisierung in JS zu ermöglichen, wird die JS-Erweiterung TypeScript (TS) verwendet. Diese steht jedoch nur zum Zeitpunkt der Entwicklung zur Verfügung. TS muss vor dem Ausführen des Programms zu JavaScript kompiliert werden.

Auch die Webbrowser werden von Zeit zu Zeit um einige Komponenten erweitert. Dazu werden Standards definiert und anschließend als Schnittstelle in JavaScript zur Verfügung gestellt. So verhält es sich mit WebGL und WebXR. WebXR greift zudem auf Funktionen von WebGL zurück (Details in Kapitel 3.3.4). Diese Funktionsaufrufe können mit Bibliotheken für WebGL, wie z. B. three.js, vereinfacht werden. Die Verwendung ist optional, daher ist die Verbindung im Schaubild schraffiert.

Für die Erkennung der Umgebung, also die Wahrnehmung der Oberflächen, Kanten und Ecken, und den Funktionsumfang des Hit-Tests verwendet WebXR die Google-Software ARCore.

Da AR-Anwendungen und -Spiele für Desktop-Computer nur begrenzt von Nutzen sind, muss es außerdem möglich sein, mobile Anwendungen aus FUDGE, bzw. aus dem darin integrierten Electron heraus, exportieren zu können. Hier muss daher eine Verknüpfung zu PhoneGap hergestellt werden. Für die Erstellung von den mobilen Anwendungen fiel die Entscheidung bewusst auf PhoneGap, denn es ist einfacher mit HTML, CSS und JavaScript dieselben Dateien zu schreiben und ein hybrides App-Konstrukt zu generieren, als die genannten Formate in das Android SDK und Swift (iOS) umzuwandeln.

Im Schaubild wird auch der Export für Desktop-Anwendungen dargestellt. Dieser wurde zum Zweck der Vollständigkeit mit aufgenommen, wird in dieser Arbeit jedoch nicht genauer betrachtet und ist daher ebenfalls schraffiert.



### 4.2.2 Vorgehensweise

In der vorliegenden Arbeit werden zwei Anwendungen parallel entwickelt mit dem Ziel diese am Ende zusammenfügen zu können. Eine der beiden Anwendungen befasst sich mit der Herausforderung PhoneGap in Electron zu integrieren, während sich die andere damit auseinandersetzt eine Bibliothek in JavaScript zu erstellen, um AR-Anwendungen mit WebXR und den Webtechnologien erstellen zu können.

#### a) WebXR-Bibliothek

In der WebXR-Bibliothek sollen die Funktionen von WebXR durch klare, verständliche Klassen-, Funktions- und Variablennamen aufbereitet werden. Dazu soll zunächst der Umgang mit WebXR verstanden werden, indem eine AR-Anwendung mit WebXR erstellt wird. Aus dieser Anwendung werden dann die entsprechenden Funktionen und Variablen extrahiert und eine Klasse erstellt. Diese Klasse soll dann in FUDGE integriert werden, um mit einfachen Funktionsaufrufen mobile AR-Anwendungen erstellen zu können.

#### b) Integration von PhoneGap in Electron

Das Adobe-PhoneGap-Team befindet sich derzeit in der Entwicklung einer Electron-Anwendung, um PhoneGap-Projekte einfacher und ohne das PhoneGap CLI verwenden zu können. Eine erste Version dieser Anwendung wurde bereits veröffentlicht. (vgl.

Adobe Systems Inc. 2018)

Über diese Anwendung konnte herausgefunden werden, dass es möglich ist PhoneGap in Electron zu integrieren. Dabei werden über Kind-Prozesse von Node.js PhoneGap CLI-Befehle ausgeführt. Auf eben diesem Weg soll die Anwendung in dieser Arbeit PhoneGap in Electron integrieren.

Dabei sollen vorerst jedoch nicht alle PhoneGap CLI-Befehle integriert werden, sondern lediglich die wichtigsten. Eine Übersicht über die PhoneGap-CLI Befehle befindet sich in Anhang 4. Zu den wichtigsten gehören:

- `create <path>`
- `build <platforms>`
- `install <platforms> [deprecated]`
- `run <platforms>`
- `serve <platforms>`

Diese fünf Befehle sollen in einem ersten Schritt in die Electron-Anwendung integriert werden. Damit können dann PhoneGap-Anwendungen erstellt werden, die für bestimmte Plattformen erstellt, auf bestimmten Plattformen installiert und auf einem lokalen Server getestet werden können.

Zusätzlich soll auch die Möglichkeit bestehen, bereits vorhandene PhoneGap-Projekte zu öffnen. Diese Funktion ist gerade in Anbetracht von FUDGE durchaus notwendig, da Schüler meist ein Projekt starten und dieses von Unterrichtsstunde zu Unterrichtsstunde weiterentwickeln.

## 4.2.3 Ablaufdiagramm

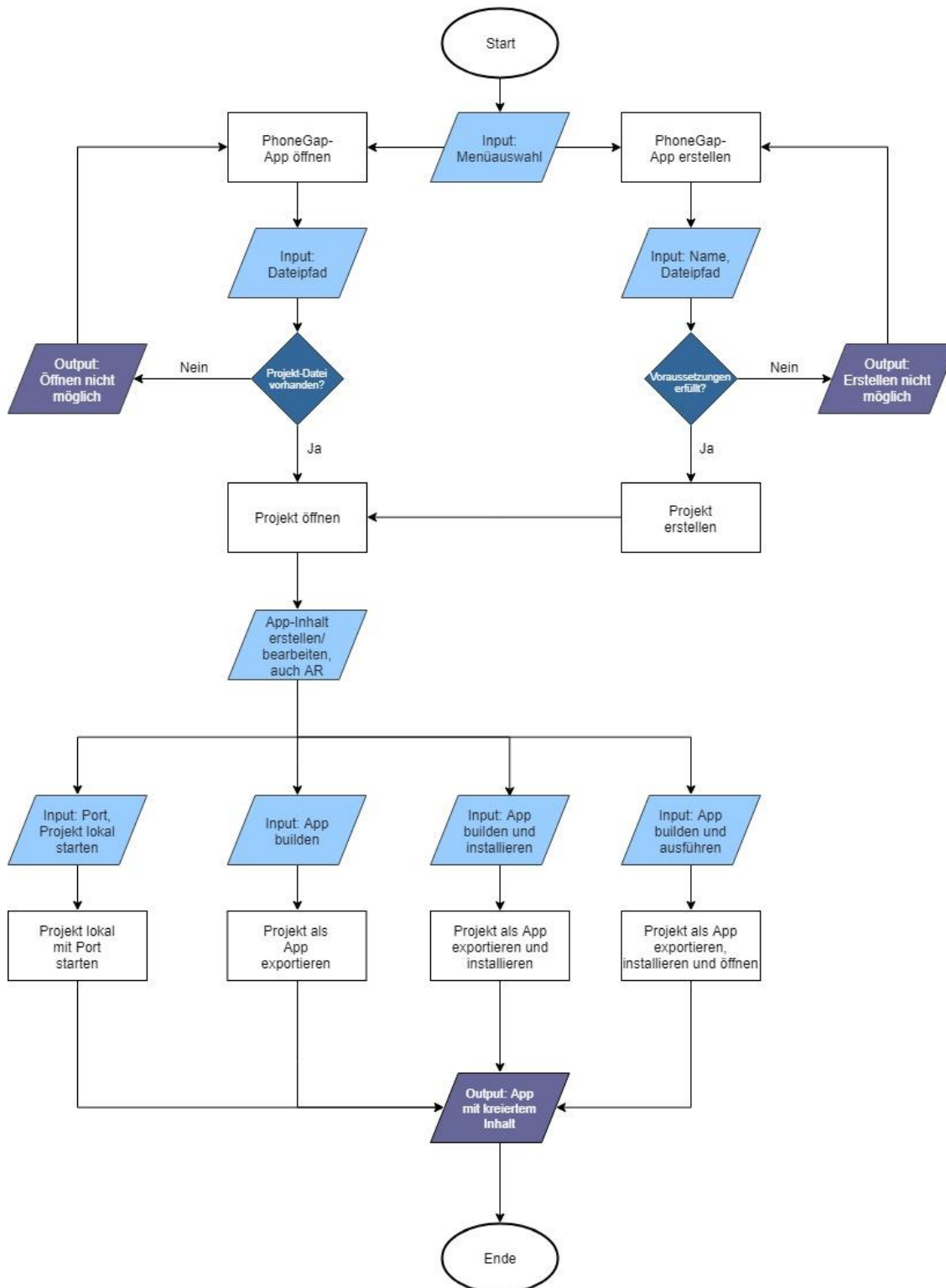


Abbildung 4.2 Ablaufdiagramm der Anwendung

In Abbildung 4.2 wird das Ablaufdiagramm der hier beschriebenen Anwendung dargestellt. Eine bessere Darstellung dieser ist auf dem beigefügten Datenträger zu finden.

Über eine Menüauswahl kann sich der Nutzer einen der Punkte „App erstellen“ und „App öffnen“ heraussuchen. Soll eine neue App erstellt werden, so muss der Nutzer einen Namen für das Projekt und einen Dateipfad für den Speicherort angeben. Erfüllt das System alle Voraussetzungen (dazu gehört unter anderem, ob PhoneGap installiert ist) kann das Projekt erstellt und anschließend geöffnet werden. Möchte der Nutzer die App öffnen, so muss er den Ordner auswählen, der das Projekt enthält. Konnte ein PhoneGap-Projekt im angegebenen Verzeichnis gefunden werden, so kann das Projekt geöffnet werden.

Nachdem ein Projekt geöffnet wurde, können die Inhalte für die Anwendung erstellt werden. Dazu gehört auch die Möglichkeit eine AR-Anwendung zu erstellen. Hat der Nutzer den Inhalt erstellt, hat er vier Möglichkeiten eine ausführbare App daraus zu erstellen:

- Das Starten der App auf einem lokalen Server mit variablem Port
- Das Erstellen einer ausführbaren Datei (builden)
- Das Erstellen einer ausführbaren Datei mit ihrer anschließenden Installation auf einem Endgerät, das über ein USB-Kabel mit dem PC verbunden ist
- Das Erstellen einer ausführbaren Datei mit anschließender Installation und automatischer Ausführung auf einem Endgerät, das über ein USB-Kabel mit dem PC verbunden ist

### 4.3 Struktur

Bei beiden Anwendungen wurde eine identische Struktur verwendet, sodass diese auch problemlos kombiniert werden können. Für die Benennung der Dateien wurde ein einfaches Muster eingehalten. Klassen erhalten immer den Prefix „class“, während bei der Electron Anwendung alle Renderer-Skripte den Zusatz „renderer“ erhalten. Alle Skript-

---

Dateien befinden sich in einem Ordner mit dem Namen „src“. Die HTML-Dateien/ Template-Dateien gehören in einen Ordner mit dem Namen „templates“. Für Assets wurde außerdem ein gleichnamiger Ordner angelegt. Hierin befinden sich z. B. 3D-Modelle. Für Teilbereiche, die zu einem großen Ganzen gehören, aufgrund der Übersichtlichkeit jedoch in einzelne Dateien unterteilt werden, wird der Teilbereich mit einem Bindestrich im Namen angehängt (z. B. „renderer.phonegap-create.ts“).

Die einzelnen Bereiche werden zudem in einem Ordner zusammengefasst. So befinden sich mögliche Renderer und Klassen sowie Teilbereiche und die kompilierten JS-Dateien in diesem Ordner. Einen Einblick zur Verdeutlichung der Struktur der Anwendungen zeigt Abbildung 4.3.

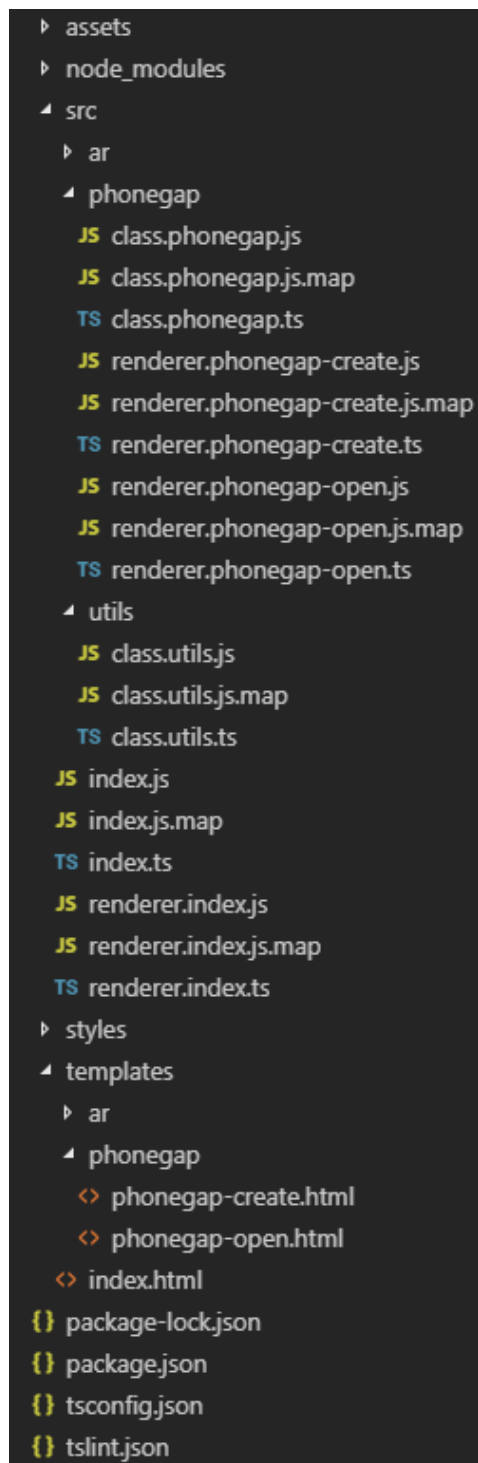


Abbildung 4.3 Struktur der Anwendungen

---

## 4.4 Klassendiagramm

In Anhang 5 ist das Klassendiagramm der Anwendung dargestellt. Hier werden die Klassen dargestellt, welche für die Arbeit von größter Bedeutung sind. Für die Implementierung von PhoneGap wurde eine PhoneGap-Klasse erstellt – für die AR-Komponente wurde die Klasse „XR“ erstellt. Zur Hilfestellung wurden zudem die Klassen „ObjectLoader“ und „ReturnMessage“ erstellt. Wozu genau die Klasse „Reticle“ in „XR“ benötigt wird, wird im folgenden Kapitel beschrieben.

## 5 Umsetzung der Anwendung

### 5.1 Beschreibung der Umsetzung

Nachdem die Voraussetzungen für die Entwicklung mit WebXR alle definiert und die Entwicklungsumgebung dementsprechend eingerichtet war, konnte mit der Umsetzung gestartet werden. Der Quellcode zur Anwendung ist auf dem beigelegten Datenträger zu finden. Die Anwendung musste aufgrund der noch instabilen Entwicklung der WebXR-Device-API leider mit einer älteren Version (Dezember 2018) umgesetzt werden, wie sie in Kapitel 3.3.4 (Februar 2019) beschrieben wurde. Mit einem mobilen Endgerät und dem entsprechenden Browser (Chrome Version 71.0.3578.5) kann die Website mit der beschriebenen AR-Anwendung unter der URL

*<https://kathrinfuhrer.de/masterthesis/webxr-201812/> aufgerufen werden<sup>16</sup>.*

Zunächst wurde eine einfache AR-Anwendung auf einem lokalen Server entwickelt. Diese Anwendung erstellt mit Zugriffen auf die interne Kamera des mobilen Endgeräts eine Ansicht der realen Umgebung, wie es für AR notwendig ist. Dazu wird bei der Initialisierung durch die Funktion `requestDevice()` der WebXR-Device API sichergestellt, dass eine Anzeige von AR-Inhalten mit dem verwendeten Endgerät möglich ist. Anderenfalls wird eine Ausgabe auf dem Screen mit der Nachricht „Your browser does not support WebXR“ erzeugt. Wurde ein AR-fähiges Endgerät gefunden, wird ein Button erstellt, über den eine sogenannte „XRSession“ gestartet werden kann. Beim Starten der Session wird ein canvas-Element erzeugt, dessen Kontext mit einem „xrpresent“-Parameter generiert wird. Dieser Kontext wird `XRPresentationContext` genannt und muss der Session zugewiesen werden. Anschließend wird dieser Kontext

---

<sup>16</sup> Bitte beachten: Es muss die angegebene Chrome Version verwendet werden (71.0.3578.5), die Chrome-Flags `#webxr` und `#webxr-hit-test` müssen unter `chrome://flags` aktiviert werden, die Verwendung einer sicheren Verbindung (https) ist zwingend erforderlich und das Endgerät muss Android 8 oder später verwenden (für ARCore) ansonsten kann die Anwendung nicht funktionieren.



dem HTML-Body angeheftet. Wurde die Session auf diese Weise gestartet, kann mit dem Aufbau der virtuellen Szene begonnen werden. Wichtig dabei ist, dass der zu erstellende Kontext mit dem mobilen Endgerät (dem XRDevice) kompatibel bleibt. Dazu wird die Funktion `setCompatibleXRDevice` auf den Rendering-Context angewendet.

Um die Anzeige der Kamera mit jedem Frame zu aktualisieren, muss eine Schleife über die Funktion `requestAnimationFrame` erzeugt werden, die zu jedem Frame die übergebene Funktion (`onXRFrame`) aufruft. Zudem wird das aktuelle „FrameOfReference“-Objekt gespeichert, damit eine Interaktion mit den Frames möglich ist. Für einen reibungslosen Ablauf der Bilddarstellung werden die Bilder in einem Puffer zwischengespeichert:

```
this.gl.bindFramebuffer(this.gl.FRAMEBUFFER, this.session.baseLayer.framebuffer);
```

Diese Frames werden dann von der Software ARCore im Hintergrund und ganz ohne einen extra Programmier-Aufruf gescannt. Dabei werden Ecken und Kanten, sämtliche Oberflächen wie Wände, Tische und Decken erkannt und gespeichert. Um diese für den Nutzer sichtbar zu machen, wurde ein sogenanntes `Reticle` verwendet – ein Fadenkreuz, welches in der Mitte des Bildschirms angezeigt und immer auf einer erkannten Oberfläche angeheftet dargestellt wird.

In jedem Frame muss dann noch sichergestellt werden, dass sich die virtuelle und die reale Welt an den gleichen Positionen befinden. Dazu wird die `XRDevicePose`, also die Position des Endgeräts in jedem Referenz-Frame mit dem Renderer der virtuellen Welt bzw. dessen Kamera synchronisiert.

Natürlich sollte die Anwendung, die als Basis für die Erstellung der XR-Klasse dient, eine Nutzerinteraktion beinhalten. Dazu wurde ein `EventListener` auf das Canvas-Element erstellt. In dieser Funktion wird dann die gleiche „Technik“ verwendet, um ein Objekt auf einer Oberfläche darzustellen, wie es auch beim `Reticle` verwendet wird. Dazu wird auf die Funktion `requestHitTest` der `XRSession` zurückgegriffen. Damit wird ein 3D-Vektor in der Mitte des Bildschirms mit der Richtung des Endgeräts erzeugt. Anhand

dieses Vektors können die Schnittpunkte mit den von ARCore erkannten Flächen ermittelt und zurückgegeben werden. Darauf wird dann das Objekt positioniert.

Entstanden ist so die Grundlage für die WebXR-Device-API-Klasse, bestehend aus vier elementar wichtigen Funktionen:

- onEnterAR,
- onSessionStarted,
- onXRFrame und
- onClick.

Um das weitere Vorgehen zu verdeutlichen, wurde Abbildung 5.1 erstellt.

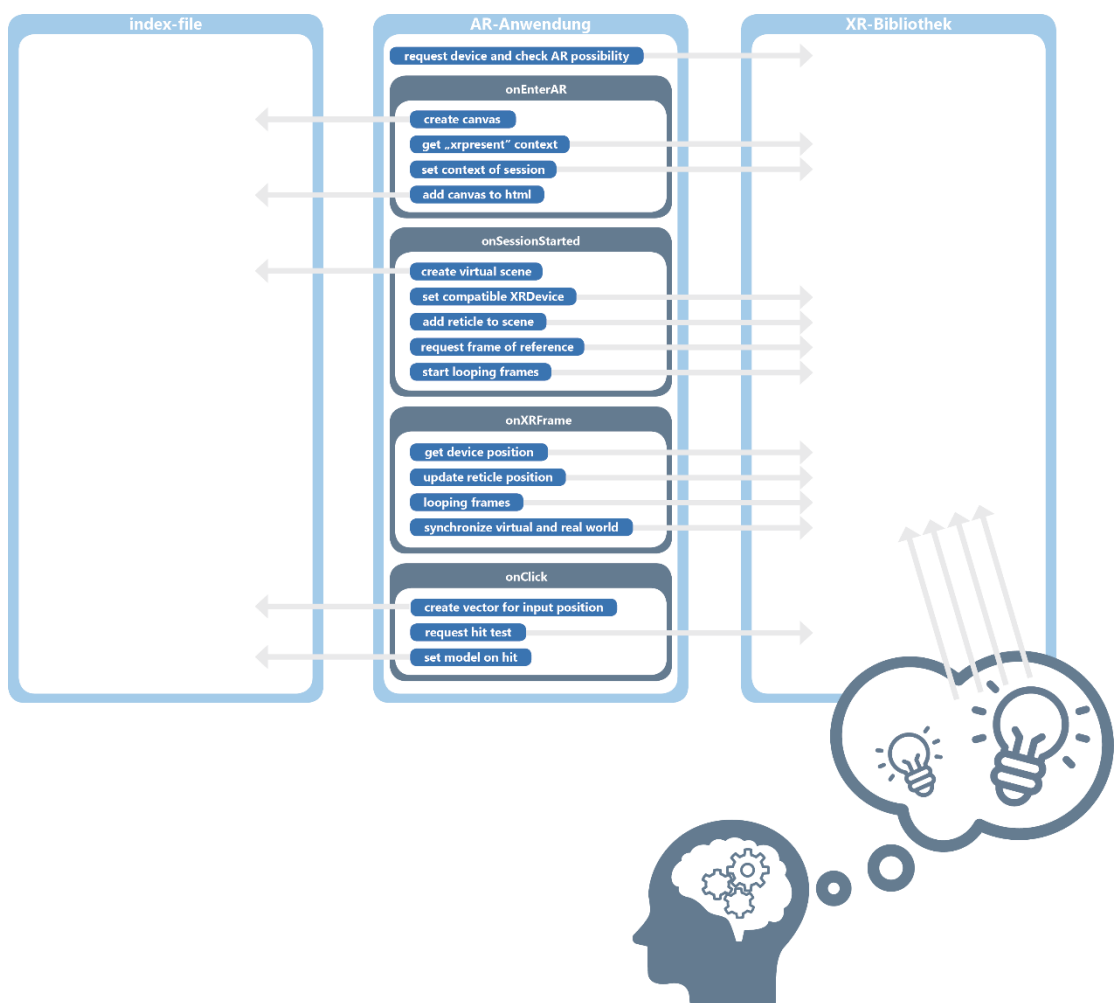


Abbildung 5.1 Von der Anwendung zur XR-Bibliothek

Die Anwendung wurde in zwei Bereiche eingeteilt. Ein Teil befasst sich mit der Veranschaulichung der Anwendung, also all das, was später von einem Nutzer des Game-Editors veränderbar sein soll. Dazu gehört zum Beispiel das Erstellen des canvas-Elements oder das Kreieren der virtuellen Scene. Der andere Teil beinhaltet all das, was im Hintergrund passieren soll, also das was nicht von einem Nutzer des Game-Editors veränderbar sein soll. Dazu gehört unter anderem das Abfragen des XRDevices und die Synchronisation zwischen realer und virtueller Welt. Außerdem wurde für die Initialisierung des XRDevices eine Initialisierungsfunktion geschrieben.

Anschließend wurde die Bibliothek um weitere nützliche Funktionen ergänzt. So kann eine Session z. B. über einen cancel-Button auch wieder beendet werden. Die Funktionen wurden zudem teilweise vereinfacht, indem Funktionalitäten in eine eigene Funktion ausgelagert wurden. Dazu gehört unter anderem die Funktion `setupWebGLLayer`, die für die Kompatibilität mit dem XRDevice zuständig ist, sowie die Klasse „Reticle“, die alle Informationen zum Reticle enthält. Außerdem wurde die Funktion `checkForSurfaces` umgesetzt, die überprüft, ob Oberflächen erkannt wurden. Diese löst ein Event aus, auf welches die Nutzer zugreifen können, sobald Oberflächen gefunden wurden. Die `onClick`-Funktion wurde ebenfalls angepasst. Wird kein Reticle verwendet, so kann auf jede Stelle des Screens geklickt werden, um dort ein Objekt zu platzieren. Dafür wird das `XRInputSourceEvent` der WebXR API verwendet. Um dieses zu verwenden, muss der `EventListener` der `XRSession` zugewiesen werden. Wird das Reticle verwendet, so wird immer die Position des Reticles als Referenzpunkt verwendet.

Da nicht grundsätzlich davon ausgegangen werden kann, dass beim Auslösen eines Input-Events ein Objekt platziert werden soll, sondern der Nutzer zunächst nur die Position der Schnittstelle benötigt, wird ein „hit“-Event ausgelöst, das die Schnittstellen beinhaltet. Die Funktion zur Platzierung des Events wurde daher ebenfalls ausgelagert und kann über die Klasse `XR` aufgerufen werden, wenn es vom Nutzer benötigt wird.

Dazu wurden noch einige setter-Funktionen implementiert:

- `enableHitTest`:  
Ermöglicht oder verhindert das Ausführen von Hit-Tests. Das ist nützlich, wenn zum Beispiel nur eine virtuelle Scene in der realen Welt angezeigt werden muss.
- `enableReticle`  
Ermöglicht oder verhindert das Anzeigen des Reticles
- `reticle`  
Ersetzt das Standard-Reticle-Objekt durch das übergebene Objekt
- `enableReticleDistance`  
Ermöglicht oder verhindert das Anzeigen der Distanz zwischen Reticle und Nutzer bzw. dessen Endgerät
- `enableSurfaceChecking`  
Ermöglicht oder verhindert das Überprüfen, ob Oberflächen erkannt wurden

Die WebXR-Bibliothek wurde damit erstellt. Nun sollte diese Bibliothek in Electron integriert werden. Electron verwendet in der aktuell 3. Version die Chromium-Version 69. Die Funktionalitäten für WebXR werden jedoch erst ab Chromium Version 71 implementiert. Außerdem müssten, wie in der Analyse beschrieben, für die WebXR Device API zwei experimentelle Flags gesetzt werden. Eine Einbindung in Electron direkt ist daher noch nicht möglich.

Was jedoch möglich ist, ist die Integrierung von PhoneGap in eine Electron-Anwendung. Für den FUDGE ist es sehr wichtig, dass die Anwendung von Electron heraus auf mobilen Endgeräten angezeigt werden kann – ansonsten ist eine Implementierung der oben beschriebenen XR-Bibliothek überflüssig. Es wurde daher eine weitere Anwendung entwickelt, die genau diese Funktionalität bietet. Der Quellcode dieser Anwendung sowie eine ausführbare Datei befinden sich ebenfalls auf dem beigefügten Datenträger.

Wie in Kapitel 4.2.2 beschrieben, wurden dazu fünf PhoneGap-CLI Befehle integriert. Hierzu wurde ebenfalls eine kleine Bibliothek geschrieben, die in den FUDGE integriert werden kann, sobald dieser die Grundfunktionalitäten enthält. Alle Funktionen der PhoneGap-Klasse können dem Klassendiagramm in Anhang 5 entnommen werden. Eine

Übersicht über die Funktionalität der Bibliothek zeigt das Aktivitätsdiagramm in Abbildung 4.2 auf Seite 51.

Über node.js-Kind-Prozesse können jegliche CLI-Befehle ausgeführt werden, also auch diejenigen, die durch PhoneGap integriert werden. Dazu muss zunächst sichergestellt werden, dass sowohl node.js als auch PhoneGap auf der lokalen Maschine des Nutzers installiert sind. Diese Funktionalität bietet die Bibliothek mit der Funktion `checkDependencies()`. Mit `ChildProcess.exec` wird der Befehl „node –version“ ausgeführt, ebenso wie der Befehl „phonegap –version“. Wird eine Versionsnummer zurückgegeben, sind die Voraussetzungen erfüllt. Ist das nicht der Fall, so wird der Nutzer aufgefordert diese zu installieren.

Mit der Funktion `createProject()` kann ein neues PhoneGap-Projekt erstellt werden. Dazu muss ein Name und ein Pfad zum gewünschten Speicherort angegeben werden. Beim Erstellen wird zunächst überprüft, ob das Verzeichnis vorhanden ist und anschließend über den Befehl „phonegap create“ mit Angabe des Projektnamens und unter Anweisung des Speicherorts erzeugt (Listing 5.1).

```
let command: string = "phonegap create " + this._appFolderName + ' --name "' +
+ this._appName + '"';
try {
  const { stdout, stderr } = await this._exec(command, { cwd: this._dirPath });
  if (stdout) {
    this._pathToProject = this._dirPath + "\\\" + this._appFolderName;
    console.log("Created Phonegap Project");
    return new ReturnMessage(true, "Created phonegap project.");
  } else if (stderr) {
    return new ReturnMessage(false, "Could not create project.\n" + stderr);
  }
} catch (error) {
  return new ReturnMessage(false, "Project could not be created. Check if your project path already exists.");
}
```

Listing 5.1 Erstellen eines neues PhoneGap-Projekts

Um ein bestehendes Projekt zu öffnen, wird ebenfalls der Pfad zum Speicherort des Projekts benötigt. Anschließend wird über den Befehl `readFile` versucht die config.xml-

Datei des Projekts zu öffnen. Gelingt das, wird aktuell davon ausgegangen, dass das verwendete Verzeichnis ein phonegap-Verzeichnis ist. Gelingt es nicht, wird der Nutzer darauf hingewiesen, dass es sich eventuell nicht um ein PhoneGap-Projekt handelt bzw. die config.xml-Datei nicht gelesen werden konnte. Beim Lesen der config.xml-Datei werden außerdem die benötigten Variablen gelesen, um den Projektnamen herauszufinden (Listing 5.2).

```
try {
  fs.readFile(infoFile, (error, data_xml) => {
    if (!error) {
      let parser = new xml2js.Parser();
      parser.parseString(data_xml, (error: any, data: any) => {
        if (!error) {
          this._fileInfo = data.widget;
          this._appName = this._fileInfo.name[0];
          resolve(new ReturnMessage(true, "Opened project."));
        } else {
          reject(new ReturnMessage(false, "Could not open config.xml. Is this
a PhoneGap-Project?"));
        }
      });
    } else {
      reject(new ReturnMessage(false, "Error while reading config.xml " +
error));
    }
  });
} catch (error) {
  reject(new ReturnMessage(false, "Could not open file. " + error));
}
```

*Listing 5.2 readFile-Funktion zum Öffnen eines PhoneGap-Projekts*

Der Aufbau der Funktionen für die Generierung einer erstellten App auf einem lokalen Server (`serveProject`) als ausführbare Datei (`buildProject`) oder auf einem mobilen Endgerät (`runProject`) verhält sich sehr ähnlich. Hierbei handelt es sich um Kind-Prozesse, die im Hintergrund weiterhin ausgeführt werden. Daher muss zum Ausführen dieser Befehle die `ChildProcess.spawn` Funktion verwendet werden. Listing 5.3 zeigt die Verwendung dieser Funktion in der `serveProject` Funktion.

```
public async serveProject(port: number = 1234, terminal: HTMLElement):
Promise<ReturnMessage> {
    let command: string;
    if (os.platform() === "win32") {
        command = "phonegap.cmd";
    } else {
        command = "phonegap";
    }

    let options: object = {
        cwd: this._pathToProject
    };

    let args: Array<any> = [];
    args.push("serve");
    args.push("--port");
    args.push(port);

    try {
        this._serveProcess = await child_process.spawn(command, args, options);

        this.doTerminalOutput(this._serveProcess, terminal);
        console.log(this._serveProcess.pid);

        return new Promise((resolve, reject) => {
            this._serveProcess.stdout.on("data", (data: any) => {
                if (data.indexOf("listening on") !== -1 &&
                    data.indexOf(":" + port) !== -1) {
                    this.createProjectWindow("http://localhost:" + port);
                    resolve(new ReturnMessage(true, "Run project locally on port " +
                        port + " pid: " + this._serveProcess.pid));
                }
            });
            this._serveProcess.stderr.on("data", (data: any) => {
                reject(new ReturnMessage(false, "Could not run project locally. See
                    terminal output for details."));
            });
        });
    } catch (error) {
        return new ReturnMessage(false, "Could not run project locally: " +
            error);
    }
}
```

Listing 5.3 Funktion, um ein PhoneGap-Projekt auf einem lokalen Server zu starten

Werden Daten aus diesem spawn-Prozess zurückgegeben, müssen diese über Events an den Nutzer übergeben werden. Dies geschieht durch den Aufruf der Funktion `doTerminalOutput`. Die Daten werden hierin für die Darstellung in einem HTML-Element mit der Funktion `consoleStyleToHtmlString` aufbereitet und anschließend ein entsprechendes Event („process-data“ oder „process-end“) mit dem Inhalt als HTML-

String ausgelöst. Dieser Inhalt kann anschließend vom Nutzer in einem „Terminal“, also einer Art Ausgabefenster, angezeigt werden. Wie das aussehen kann zeigt Abbildung 5.2. Wird der `serveProcess`-Befehl ausgeführt, kann außerdem ein Port für die Ausführung auf dem lokalen Server bestimmt werden und die Anwendung wird in einem neuen Fenster geöffnet.

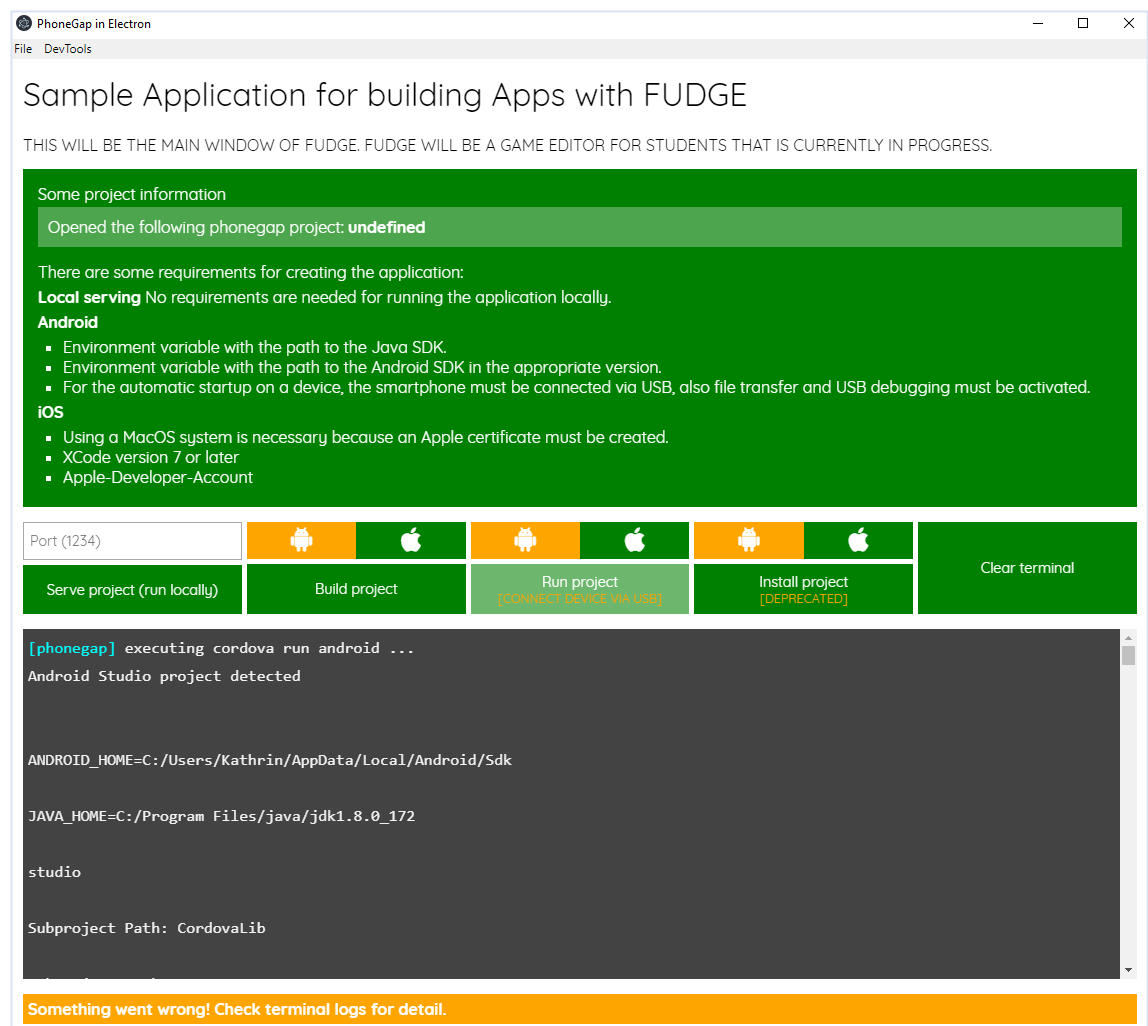


Abbildung 5.2 Screenshot der Anwendung mit Terminal-Ausgabe

Die Funktion `installProcess` wurde im Rahmen dieser Arbeit noch umgesetzt, ist von PhoneGap jedoch als deprecated, also veraltet, markiert worden und wird mit einer neuen Version nicht mehr funktionieren. Die Befehle `buildProcess` und `runProcess` können außerdem mit einem entsprechenden Parameter für die Plattform ausgeführt werden.



Implementiert wurden bisher die Plattformen Android und iOS. Die Ausführung für iOS-Anwendungen konnten nur bedingt (auf einem Windows-Gerät) getestet werden. Bei iOS-Applikationen ist es jedoch zwingend erforderlich ein MacOS-Gerät zu nutzen.

Außerdem gibt es weitere Eigenschaften, die bei der Erstellung von ausführbaren Anwendungen beachtet werden müssen. Diese sind bei Android-Anwendungen:

- Eine Umgebungsvariable mit dem Pfad zur Java SDK muss gesetzt sein.
- Eine Umgebungsvariable mit dem Pfad zur Android SDK in der gewünschten Version muss gesetzt sein.
- Für die Verwaltung der unterschiedlichen Plattform-Versionen wird am besten Android Studio verwendet.
- Um die Anwendung automatisch auf einem mobilen Endgerät installieren und starten zu können, muss dieses über ein USB-Kabel mit dem PC verbunden sein und auf dem Endgerät müssen die Dateiübertragung sowie USB-Debugging aktiviert sein.

Bei der Entwicklung von iOS-Anwendungen muss folgendes beachtet werden:

- Das Verwenden eines MacOS-Systems ist notwendig, da ein Apple-Zertifikat erstellt und verwendet werden muss.
- XCode muss in Version 7 oder später auf dem System vorhanden sein.
- Es wird ein Apple-Developer-Account benötigt.

Damit wurden alle Funktionen der PhoneGap-Klasse beschrieben. Die beiden hier beschriebenen Anwendungen bzw. Klassen können zu einem späteren Zeitpunkt mit nur wenig Aufwand in den FUDGE integriert und verwendet werden.

## 5.2 Herausforderungen

Die in dieser Arbeit größte Herausforderung war der Umgang mit der WebXR Device API. Wie schon mehrfach erwähnt, ist diese API in der Entwicklungsphase und ändert sich stetig. Schon ein Update des Browsers konnte die Anwendung zerstören und es mussten die „Fehler“ behoben bzw. die neu formulierten Funktionen umbenannt werden. Um in dieser Arbeit eine stabile Anwendung entwickeln zu können wurde letztendlich die Entscheidung getroffen auf einer lauffähigen Version der API und damit auch des Browsers stehen zu bleiben. Die beschriebene XR-Anwendung wurde mit dem Browser *Chrome in der Version 71.0.3578.5* auf einem *Android Gerät der Version 8* entwickelt und getestet. Außerdem müssen die Chrome-Flags `#webxr` und `#webxr-hit-test` gesetzt sein. Die Anwendung muss des Weiteren über eine gesicherte Verbindung (SSL-zertifiziert/ https) aufgerufen werden. Beim Ausführen dieser wird der Nutzer darauf hingewiesen, dass ARCore installiert werden muss, wenn das nicht der Fall ist.

Außerdem sollte eine zusätzliche Funktion beschrieben werden, die es ermöglicht die erkannten Flächen darzustellen – ähnlich wie es im Screenshot der Unity ARCore-Anwendung in Abbildung 3.7 zu sehen ist. Nach den derzeitigen Möglichkeiten in der WebXR Device API und aufgrund detaillierter Überlegungen wurde davon jedoch abgesehen. Der Rechenaufwand wäre dafür leider zu groß, denn es müssten unzählige Hit-Tests gemacht werden. Nur über Hit-Tests besteht aktuell die Möglichkeit eine Fläche zu finden.

CLI-Befehle in einer node.js-Anwendung ausführen zu können, schien zunächst keine Schwierigkeit zu sein. Allerdings musste auch hier auf Ausnahmen Rücksicht genommen werden. Diese beschränkten sich auf die Ausführung der nebenläufigen Prozesse (`ChildProcess.spawn`) und auf Windows-Geräte – hier musste der auszuführende Befehl mit dem Zusatz „`cmd`“ angegeben werden. Außerdem mussten diese Prozesse in Variablen gespeichert werden, damit auch zu einem späteren Zeitpunkt auf die Rückgabewerte des Prozesses zugegriffen werden kann.

---

Zu guter Letzt ist und bleibt die größte Herausforderung der aktuelle Stand der Technik, der eine Zusammenführung der beiden Anwendungen zum Zeitpunkt der Bearbeitung dieser Arbeit nicht möglich macht.

## 6 Zusammenfassung

Sogenannte „cross-platform“-Anwendungen sind Anwendungen, die mit einer Technologie bzw. Programmiersprache entwickelt und anschließend für unterschiedliche Plattformen exportiert werden können. Electron ist eine dieser Technologien und bildet die Grundlage für den Game-Editor (FUDGE). FUDGE ist ein Projekt der Hochschule Furtwangen und soll ein Game-Editor werden, mit dem Schülern die Basis für die Entwicklung von Spielen nähergebracht werden soll.

Um AR-Anwendungen mit diesem Game-Editor entwickeln zu können muss AR in die Webtechnologien integriert werden. Es gibt ein paar wenige API und Bibliotheken, welche die Integration ermöglichen sollen, jedoch sind diese Entwicklungen bisher experimentell und in keiner stabilen Verfassung. Die Analyse dieser Bibliotheken kam zu dem Ergebnis, dass für die Umsetzung dieser Anwendung die WebXR Device API genutzt werden soll. Die Basis der Untersuchung bildeten die Kriterien Abhängigkeiten, Aktualität, Dokumentation, Nutzererfahrung, technische Einschränkungen, Umfang und Funktionen und die Einhaltung des Ziels. Zusätzlich wurden AR-Anwendungen zum Vergleich mit Unity und den Erweiterungen Vuforia und ARCore untersucht.

Die Anforderungen an die Anwendung ergeben sich aus der Analyse und dem Ziel dieser Arbeit. Aufgrund der Verwendung von Electron müssen die Webtechnologien verwendet werden, wobei für die Entwicklung TypeScript eingesetzt wird. Für den Export auf mobile Endgeräte aus Electron heraus wird PhoneGap verwendet. Mit WebXR wird eine Bibliothek für AR im Web erstellt, wobei ARCore als Grundlage für die Erkennung der Umgebung dient. Zur Darstellung in HTML wird WebGL verwendet.

Bei der Umsetzung wurden zunächst zwei Anwendungen entwickelt, mit der Idee diese am Ende zusammenfügen zu können. Die Erste der Anwendungen integriert eine Bibliothek zur Erstellung von AR-Anwendungen im Web. Dazu wurde zunächst eine AR-Web-Anwendung mit WebXR erstellt, welche dann zu einer Bibliothek umgebaut

---

wurde. In der zweiten Anwendung werden PhoneGap-CLI-Befehle in eine Electron Anwendung, ebenfalls in Form einer Bibliothek bzw. Klasse, implementiert. Dazu wurde der node.js-ChildProcess verwendet.

Eine große Herausforderung war die Verwendung einer nicht stabilen API. Dadurch hat die Funktionalität der AR-Anwendung häufig gelitten und schon ein Browser-Update sorgte zum Zusammenbruch der Anwendung. Eine Funktion zur Darstellung der erkannten Oberflächen durch ARCore konnte aufgrund von mangelnden Rückgabewerten und dadurch erhöhtem Rechenaufwand nicht umgesetzt werden. Des Weiteren gab es eine Ausnahme für nebenläufige Kind-Prozesse auf Windowsgeräten bei der Programmierung der PhoneGap-Anwendung, die den Fortschritt der Arbeit aufhielten. Der aktuelle Stand der Technik, hauptsächlich zurückzuführen auf die Version von Electron, verhindert eine Zusammenführung der beiden Anwendungen.

## 7 Fazit und Ausblick

Im Rahmen dieser Arbeit sind zwei Anwendungen entstanden, welche die Zukunft des Webs beinhalten. Aktuell ist es nicht möglich, diese beiden Anwendungen zu einer zu verschmelzen. Jedoch bieten diese Anwendungen eine gute Grundlage für die Integration von AR und mobilen Anwendungen in den Game-Editor FUDGE. Diese Bibliotheken bieten allerdings nicht nur viel Potenzial für Erweiterungen, sondern sind auch mit Bedacht zu verwenden.

Sollen Anwendungen für mobile sowie für Desktop-Anwendungen gleichzeitig entwickelt werden, muss dabei beachtet werden, dass zum Beispiel Funktionen für die Hardware-Zugriffe (Zugriff auf das Dateisystem, Zugriff auf die Kamera, etc.) unterschieden werden müssen. Außerdem muss die Größe des Endgeräts sowie dessen Auflösung beachtet und entsprechend angepasst werden. Die Integrierung dieser Klasse in FUDGE sollte leicht von der Hand gehen. Lediglich das UI muss in FUDGE entsprechend erstellt werden.

Aufgrund der Tatsache, dass die ARCore-Erweiterung für Unity die Anzeige von den Oberflächen ermöglicht, wird es in naher Zukunft wohl möglich sein, diese Funktion in die XR-Bibliothek ohne erhöhten Rechenaufwand zu integrieren. Jedoch kann man auch davon ausgehen, dass einige Funktionen, wie zum Beispiel das Erstellen der XRSession durch andere Funktionen der API (wenn diese offiziell verwendet werden kann) ersetzt werden müssen. Bevor diese Klasse in FUDGE integriert werden kann, sollte WebXR daher offiziell verwendbar und stabil sein, und bezogen auf diese Version sollten die Zugriffe der XR-Klasse überarbeitet werden.

Durch diese Arbeit konnte ich mir einiges Neues an Wissen aneignen sowie bereits vorhandenes Wissen vertiefen. Dazu gehören unter anderem Informationen über die Spiele-Entwicklung mit AR-Inhalten und das Entwickeln von hybriden Anwendungen

---

mit Electron, sowie die Vertiefung in der Verwendung von TypeScript. Ich konnte außerdem mein Wissen in der Webentwicklung gut in diese Arbeit einbringen.

Eine neue, persönliche Herausforderung war die Entwicklung einer Bibliothek mit einer nicht stabilen API für einen noch nicht vorhandenen Game-Editor. Dabei war einiges an Vorstellungskraft sowie logisches Denken erforderlich, wobei ich teilweise über meine bisherigen Grenzen hinausgewachsen bin.

Alles in allem hat mir dieses Projekt sehr viel Freude bereitet und ich werde die Entwicklung der WebXR Device API sowie die Entwicklung von FUDGE auch über das Ende meines Studiums hinaus beobachten und nach Möglichkeit damit und daran weiterarbeiten.

## 8 Literatur- und Quellenverzeichnis

Adobe Systems Inc. 2016. *Adobe PhoneGap*. Zugriff am 07. Januar 2019.

<https://phonegap.com/>.

—. 2016. *Adobe PhoneGap. About - A high-level summary of what PhoneGap ist all about*. Zugriff am 07. Januar 2019. <https://phonegap.com/about/>.

—. 2018. *PhoneGap. Desktop App*. Zugriff am 21. Januar 2019.

<http://docs.phonegap.com/references/desktop-app/>.

AnyMotion GmbH. 2019. *AnyMotion. Marker - Augmented Reality*. Zugriff am 12.

Januar 2019. <https://anymotion.com/wissensgrundlagen/augmented-reality-marker>.

Apple Inc. 2019. *Swift 4*. Zugriff am 07. Januar 2019.

<https://developer.apple.com/swift/>.

Bendel, Prof. Dr. Oliver. 2018. *Gabler Wirtschaftslexikon*. 19. 02. Zugriff am 28.

Dezember 2018. <https://wirtschaftslexikon.gabler.de/definition/virtuelle-realitaet-54243/version-277293>.

Cabello, Ricardo. 2019. *GitHub. three.js*. Zugriff am 07. Januar 2019.

<https://github.com/mrdoob/three.js/>.

—. kein Datum. *three.js / docs*. Zugriff am 14. Januar 2019.

<https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>.

Cannon, Ada Rose, Chris Wilson, Dominique Hazael-Massieux, Nell Waliczek, Brandon Jones, und Trevor F. Smith. 2019. *webxr-reference*. Zugriff am 14. Januar 2019.

<https://immersive-web.github.io/webxr-reference/webxr-device-api/>.

Carpignoli, Nicolò. 2018. *AR.js - The Simplest Way to get Cross-Browser Augmented Reality on the Web*. 19. 04. Zugriff am 12. Januar 2019.



<https://medium.com/chialab-open-source/ar-js-the-simpliest-way-to-get-cross-browser-ar-on-the-web-8f670dd45462>.

Dell'Oro-Friedl, Prof. Jirka. 2018. *GitHub*. *FUDGE Home*. 24. 11. Zugriff am 19. Januar 2019. <https://github.com/JirkaDellOro/FUDGE/wiki>.

Deveria, Alexis. 2019. *Can i use*. 30. 01. Zugriff am 05. Februar 2019. <https://caniuse.com/#compare=edge+18,firefox+65,chrome+72,safari+12,opera+57>.

Etienne, Jerome. 2018. *GitHub*. *AR.js*. Zugriff am 12. Januar 2019. <https://github.com/jeromeetienne/AR.js>.

GitHub Inc. 2019. *GitHub*. *Electron*. Zugriff am 07. Januar 2019. <https://github.com/electron/electron>.

GitHub, Inc. kein Datum. *Electron*. Zugriff am 07. Januar 2019. <https://electronjs.org/>.

Google Inc. 2018. *Building an augmented reality (AR) application using the WebXR Device API*. Zugriff am 15. Januar 2019. <https://codelabs.developers.google.com/codelabs/ar-with-webxr/#4>.

Google Inc. 2017. *GitHub*. *google-ar/three.ar.js*. Zugriff am 13. Januar 2019. <https://github.com/google-ar/three.ar.js/blob/master/examples/markers.html>.

Google LLC. 2018. *ARCore Overview*. 02. 08. Zugriff am 07. Januar 2019. <https://developers.google.com/ar/discover/>.

—. 2018. *Create an Android project*. 17. 04. Zugriff am 07. Januar 2019. <https://developer.android.com/training/basics/firstapp/creating-project>.

—. kein Datum. *The Chromium Projects*. Zugriff am 07. Januar 2019. <https://www.chromium.org/chromium-projects>.

InternetLiveStats.com. kein Datum. *internetlivestats. Total number of Websites*. Zugriff am 05. Februar 2019. <http://www.internetlivestats.com/total-number-of-websites/#>.

Jones, Brandon, und Nell Waliczek. 2019. *WebXR Device API*. 10. 01. Zugriff am 14. Januar 2019. <https://immersive-web.github.io/webxr/>.

Lamb, Philip, Ben Vaughan, Daniel Bell, und Thorsten Bux. 2018. *GitHub. artoolkitx*. Zugriff am 12. Januar 2019. <https://github.com/artoolkitx/artoolkitx>.

Länger, Klaus. 2017. *IT-Business - Was ist Virtual, Augmented und Mixed Reality?* 06. 10. Zugriff am 28. Dezember 2018. <https://www.it-business.de/was-ist-virtual-augmented-und-mixed-reality-a-650442/>.

Löffler, Robert. kein Datum. *live-counter.com. Google-Suchanfragen weltweit*. Zugriff am 05. Februar 2019. <https://www.live-counter.com/google-suchen/>.

Lundgren, Eduardo, Thiago Rocha, Zeno Rocha, Pablo Carvalho, und Maira Bello. kein Datum. *tracking.js*. Zugriff am 13. Januar 2019. <https://trackingjs.com/>.

—. kein Datum. *tracking.js. Trackers*. Zugriff am 13. Januar 2019. <https://trackingjs.com/docs.html#trackers>.

Marcos, Diego, Don McCurdy, und Kevin Ngo. kein Datum. *A-Frame School. Set up a Web Development Environment*. Zugriff am 19. Januar 2019. <https://aframe.io/aframe-school/#/2/5>.

Markgraf, Prof. Dr. Daniel. 2018. *Gabler Wirtschaftslexikon*. 16. 02. Zugriff am 03. Januar 2019. <https://wirtschaftslexikon.gabler.de/definition/augmented-reality-53628/version-276701>.

Node.js Foundation. kein Datum. *Node.js*. Zugriff am 07. Januar 2019. <https://nodejs.org/en/>.

Pettit, Nick. 2014. *treehouse. The Beginner's Guide to three.js*. Zugriff am 19. Januar 2019. <https://blog.teamtreehouse.com/the-beginners-guide-to-three-js>.

- 
- PTC Inc. 2018. *vuforia Developer Library. Smart Terrain*. Zugriff am 14. Januar 2019.  
<https://library.vuforia.com/articles/Training/Getting-Started-with-Smart-Terrain>.
- . 2019. *vuforia Developer Portal. Downloads*. Zugriff am 14. Januar 2019.  
<https://developer.vuforia.com/downloads/sdk>.
- The Khronos Group Inc. 2019. *Khronos Group. WebGL Overview*. Zugriff am 11. Januar 2019. <https://www.khronos.org/webgl/>.
- Tönnis, Marcus. 2010. „Optisches Tracking.“ In *Augmented Reality. Einblicke in die Erweiterte Realität*, Herausgeber: Prof. Dr. O. P. Günther, Prof. Dr. R. Lienhart, Prof. Dr. W. Karl und Prof. Dr. K. Zeppenfeld, 44 - 52. Berlin: Springer-Verlag.
- Trice, Andrew. 2012. *Adobe PhoneGap. PhoneGap Explained Visually*. 02. 05. Zugriff am 07. Januar 2019. <https://phonegap.com/blog/2012/05/02/phonegap-explained-visually/>.
- Unity Technologies. 2019. *Unity documentation. Prefabs*. 08. 01. Zugriff am 15. Januar 2019. <https://docs.unity3d.com/Manual/Prefabs.html>.
- Vukicevic, Vladimir, Brandon Jones, Kearwood Gilbert, und Chris Van Wiemeersch. 2017. *WebVR*. 12. 12. Zugriff am 11. Januar 2019. <https://immersive-web.github.io/webvr/spec/1.1/>.
- WebFinance, Inc. 2019. *BusinessDictionary.com*. 02. 01. Zugriff am 07. Januar 2019.  
<http://www.businessdictionary.com/definition/cross-platform.html>.
- Wilkinson, Clayton, Fred Sauer, Ian Muldoon, und Jordan Santell. 2018. *GitHub. Google AR/three.ar.js*. Zugriff am 11. Januar 2019. <https://github.com/google-ar/three.ar.js>.
- Wilkinson, Clayton, Fred Sauer, Ian Muldoon, und Santell Jordan. 2018. *GitHub. google-ar/WebARonARCore*. Zugriff am 11. Januar 2019.  
<https://github.com/google-ar/WebARonARCore>.

## 9 Eidesstattliche Erklärung

Ich, Kathrin Fuhrer, versichere, dass ich die vorliegende Arbeit selbstständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

---

Furtwangen, den 21.02.2019

---

## 10 Anhang

Anhang 1 Three.ar.js Code-Beispiel für eine Marker-Anwendung .....	78
Anhang 2 Code-Beispiel AR.js mit three.js .....	82
Anhang 3 Übersicht der untersuchten Web-API.....	85
Anhang 4 PhoneGap-CLI Befehle .....	86
Anhang 5 Wichtigste Klassen der Anwendung als UML-Diagramm.....	87

Alle Anhänge befinden sich außerdem auf dem beigefügten Datenträger. Darauf sind unter anderem die Klassendiagramme in besserer Qualität und Größe aufzufinden.

---

*Anhang 1 Three.ar.js Code-Beispiel für eine Marker-Anwendung*

```
<!--
/*
 * Copyright 2017 Google Inc. All Rights Reserved.
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <title>three.ar.js - Marker Detection</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, user-scalable=no,
    minimum-scale=1.0, maximum-scale=1.0">
  <style>
    body {
      font-family: Monospace;
      background-color: #000000;
      margin: 0px;
      overflow: hidden;
    }

    #info {
      background-color: rgba(40, 40, 40, 0.4);
      bottom: 0;
      box-sizing: border-box;
      color: rgba(255, 255, 255, 0.7);
      left: 50%;
      line-height: 1.3em;
      padding: 0.75em;
      position: absolute;
      text-align: center;
      transform: translate(-50%, 0);
      width: 100%;
      z-index: 10;
    }

    .divider {
      color: rgba(255, 255, 255, 0.2);
      padding: 0 0.5em;
    }

    #info a {
      text-decoration: none;
      color: white;
    }
  </style>
</head>
<body>
```

```

    }

    #dat {
        user-select: none;
        position: absolute;
        left: 0;
        top: 0;
        z-index: 200;
    }

    #glFullscreen {
        width: 100%;
        height: 100vh;
        position: relative;
        overflow: hidden;
        z-index: 0;
    }

    a { color: skyblue }
</style>
</head>
<body>
<div id="info">
    <a href="https://github.com/google-ar/three.ar.js">three.ar.js</a><span
class="divider">|</span>Point device at a marker or QR code
</div>
<script src="../../third_party/three.js/three.js"></script>
<script src="../../third_party/three.js/VRControls.js"></script>
<script src="../../third_party/dat.gui/dat.gui.js"></script>

<script src="../../dist/three.ar.js"></script>

<script>
function GUI() {
    this.markerTypeString = "AR";
    this.markerType = 1;
    return this;
}
// Global variables
var vrDisplay;
var vrControls;
var arView;

var canvas;
var camera;
var scene;
var renderer;

var stats;
var cameraOrtho, cameraScene, renderer, cameraMesh;
var vrDisplay = null;
var scene, cameraPersp, model = null;
var MODEL_SIZE_IN_METERS = 0.2;
var vrControls = null;
var gui;
var scale = new THREE.Vector3();

```

```

var MARKER_SIZE_IN_METERS = 0.06;

/**
 * Use the `getARDisplay()` utility to Leverage the WebVR API
 * to see if there are any AR-capable WebVR VRDisplays. Returns
 * a valid display if found. Otherwise, display the unsupported
 * browser message.
 */
THREE.ARUtils.getARDisplay().then(function (display) {
  if (display) {
    if (display.getMarkers) {
      vrDisplay = display;
      init();
    } else {
      THREE.ARUtils.displayUnsupportedMessage("This prototype browser app for
WebAR does not support marker detection yet.");
    }
  } else {
    THREE.ARUtils.displayUnsupportedMessage();
  }
});

function init() {
  // Initialize the dat.GUI.
  var datGUI = new dat.GUI();
  gui = new GUI();
  datGUI.add(gui, "markerTypeString", ["AR",
"QRCODE"]).onFinishChange(function(value) {
    if (!vrDisplay) {
      return;
    }
    if (value === "QRCODE") {
      gui.markerType = vrDisplay.MARKER_TYPE_QRCODE;
    }
    else if (value === "AR") {
      gui.markerType = vrDisplay.MARKER_TYPE_AR;
    }
  }).name("Marker type");

  // Setup the three.js rendering environment
  renderer = new THREE.WebGLRenderer({ alpha: true });
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setSize(window.innerWidth, window.innerHeight);
  renderer.autoClear = false;
  canvas = renderer.domElement;
  document.body.appendChild(canvas);
  scene = new THREE.Scene();

  // Creating the ARView, which is the object that handles
  // the rendering of the camera stream behind the three.js
  // scene
  arView = new THREE.ARView(vrDisplay, renderer);

  // The ARPerspectiveCamera is very similar to THREE.PerspectiveCamera,
  // except when using an AR-capable browser, the camera uses
  // the projection matrix provided from the device, so that the

```



```
// perspective camera's depth planes and field of view matches
// the physical camera on the device.
camera = new THREE.ARPerspectiveCamera(vrDisplay, 60, window.innerWidth /
window.innerHeight, 0.01, 100);

// Create a model to be placed in the world using picking
model = new THREE.Mesh(new THREE.ConeBufferGeometry(
    MODEL_SIZE_IN_METERS / 2, MODEL_SIZE_IN_METERS),
    new THREE.MeshLambertMaterial( {color: 0x888888 } ));

// Apply a rotation to the model so it faces in the direction of the
// normal of the plane when the picking based reorientation is done
model.geometry.applyMatrix(
    new THREE.Matrix4().makeTranslation(0, MODEL_SIZE_IN_METERS / 2, 0));
model.geometry.applyMatrix(
    new THREE.Matrix4().makeRotationX(THREE.Math.degToRad(90)));
model.position.set(0, 0, -1);
scene.add(model);
// Add some Lighting
var directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.position.set(0, 1, 0);
scene.add(directionalLight);

// VRControls is a utility from three.js that applies the device's
// orientation/position to the perspective camera, keeping our
// real world and virtual world in sync.
vrControls = new THREE.VRControls(camera);

// Correctly handle window resize events
window.addEventListener( 'resize', onWindowResize, false );

update();
}

/**
 * On window resize, update the perspective camera's aspect ratio,
 * and call `updateProjectionMatrix` so that we can get the latest
 * projection matrix provided from the device
 */
function onWindowResize () {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
}

function update() {
    // Render the device's camera stream on screen first of all.
    // It allows to get the right pose synchronized with the right frame.
    arView.render();

    // Update our camera projection matrix in the event that
    // the near or far planes have updated
    camera.updateProjectionMatrix();

    // Update our perspective camera's positioning
    vrControls.update();
}
```

```

// Render our three.js virtual scene
renderer.clearDepth();
renderer.render(scene, camera);

var markers = vrDisplay.getMarkers(gui.markerType, MARKER_SIZE_IN_METERS);
if (markers.length > 0) {
  model.matrix.fromArray(markers[0].modelMatrix);
  model.matrix.decompose(model.position, model.quaternion, scale);
}

// Kick off the requestAnimationFrame to call this function
// when a new VRDisplay frame is rendered
vrDisplay.requestAnimationFrame(update);
}
</script>
</body>
</html>

```

Link zur Anwendung (Funktioniert nur mit WebARonARCore oder WebARonARKit):

<https://google-ar.github.io/three.ar.js/examples/markers.html>

#### Anhang 2 Code-Beispiel AR.js mit three.js

```

<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, user-scalable=no,
minimum-scale=1.0, maximum-scale=1.0">
    <title>Hello, AR Cube!</title>
    <!-- include three.js library -->
    <script src='js/three.js'></script>
    <!-- include jsartoolkit -->
    <script src="jsartoolkit5/artoolkit.min.js"></script>
    <script src="jsartoolkit5/artoolkit.api.js"></script>
    <!-- include three.js artoolkit -->
    <script src="threejs/threejs-artoolkitsource.js"></script>
    <script src="threejs/threejs-artoolkitcontext.js"></script>
    <script src="threejs/threejs-arbasecontrols.js"></script>
    <script src="threejs/threejs-armarkercontrols.js"></script>
  </head>

  <body style='margin : 0px; overflow: hidden; font-family: Monospace;'>
    <!--
      Example created by Lee Stemkoski: https://github.com/stemkoski
      Based on the AR.js Library and examples created by Jerome Etienne:
      https://github.com/jeromeetienne/AR.js/
    -->
    <script>
      var scene, camera, renderer, clock, deltaTime, totalTime;
      var arToolkitSource, arToolkitContext;
      var markerRoot1, markerRoot2;
      var mesh1;
    </script>

```

```

initialize();
animate();

function initialize() {
    scene = new THREE.Scene();
    let ambientLight = new THREE.AmbientLight( 0xcccccc, 0.5 );
    scene.add( ambientLight );

    camera = new THREE.Camera();
    scene.add(camera);

    renderer = new THREE.WebGLRenderer({
        antialias : true,
        alpha: true
    });
    renderer.setClearColor(new THREE.Color('lightgrey'), 0)
    renderer.setSize( 640, 480 );
    renderer.domElement.style.position = 'absolute'
    renderer.domElement.style.top = '0px'
    renderer.domElement.style.left = '0px'
    document.body.appendChild( renderer.domElement );

    clock = new THREE.Clock();
    deltaTime = 0;
    totalTime = 0;

    ////////////////////////////////////////////
    // setup arToolkitSource
    ////////////////////////////////////////////
    arToolkitSource = new THREE.ArToolkitSource({
        sourceType : 'webcam',
    });

    function onResize() {
        arToolkitSource.onResize()
        arToolkitSource.copySizeTo(renderer.domElement)
        if ( arToolkitContext.arController !== null )
        {
            arToolkitSource.copySizeTo(arToolkitContext.arController.canvas)
        }
    }

    arToolkitSource.init(function onReady(){
        onResize()
    });

    // handle resize event
    window.addEventListener('resize', function(){
        onResize()
    });

    ////////////////////////////////////////////
    // setup arToolkitContext
    ////////////////////////////////////////////
    // create arToolkitContext
    arToolkitContext = new THREE.ArToolkitContext({

```

```

        cameraParametersUrl: 'data/camera_para.dat',
        detectionMode: 'mono'
    });

    // copy projection matrix to camera when initialization complete
    arToolkitContext.init( function onCompleted(){
        camera.projectionMatrix.copy(
arToolkitContext.getProjectionMatrix() );
    });
    //////////////////////////////////////
    // setup markerRoots
    //////////////////////////////////////
    // build markerControls
    markerRoot1 = new THREE.Group();
    scene.add(markerRoot1);
    let markerControls1 = new THREE.ArMarkerControls(arToolkitContext,
markerRoot1, {
        type: 'pattern', patternUrl: "data/hiro.patt",
    })
    let geometry1 = new THREE.CubeGeometry(1,1,1);
    let material1 = new THREE.MeshNormalMaterial({
        transparent: true,
        opacity: 0.5,
        side: THREE.DoubleSide
    });

    mesh1 = new THREE.Mesh( geometry1, material1 );
    mesh1.position.y = 0.5;

    markerRoot1.add( mesh1 );
}
function update() {
    // update artoolkit on every frame
    if ( arToolkitSource.ready !== false )
        arToolkitContext.update( arToolkitSource.domElement );
}

function render() {
    renderer.render( scene, camera );
}

function animate() {
    requestAnimationFrame(animate);
    deltaTime = clock.getDelta();
    totalTime += deltaTime;
    update();
    render();
}
</script>
</body>
</html>

```

Link zur Anwendung: <https://stemkoski.github.io/AR-Examples/hello-cube.html>

## Anhang 3 Übersicht der untersuchten Web-API

Kriterium	Three.ar.js	Ar.js	Tracking.js	WebXR
<b>Aktualität</b>	2018-02-	2018-12-02	2018-05-17	2019-01-10
<b>Basis-technologien</b>	WebGL, WebVR, three.js	WebGL, ARToolKit, three.js oder A-Frame, [WebVR]	-	WebVR, WebGL
<b>Dokumentation</b>	Sehr gut	Okay	Sehr gut	Sehr gut (noch nicht stabil)
<b>Nutzererfahrungen</b>	Schlecht	Okay	Sehr gut	Sehr gut
<b>Technische Einschränkungen</b>	Browser: WebARonARCore, WebARonARKit Endgeräte: Google Pixel 1 + 2, Samsung Galaxy S8	-	-	Browser: Chrome Dev, Chrome Canary
<b>Umfang / Funktionen</b>	AR-Kamera: ja Tracking: ja Referenz-Objekt: ja Event-System: ?	AR-Kamera: ja Tracking: ja Referenz-Objekt: nicht direkt Event-System: nein	AR-Kamera: ja Tracking: ja Referenz-Objekt: nicht direkt Event-System: ja	AR-Kamera: ja Tracking: ja (ARCore) Referenz-Objekt: nicht direkt Event-System: ja
<b>Zielerreichbarkeit</b>	ARCore: nein Markerlos: ? Electron: nein	ARCore: nein Markerlos: nein Electron: nein	ARCore: nein Markerlos: bedingt Electron: ja	ARCore: ja Markerlos: ja Electron: ja (in naher Zukunft)

*Anhang 4 PhoneGap-CLI Befehle*

Befehle	Beschreibung
help [command]	Ausgabe der Nutzungsinformationen
create <path>	Ein phonegap-Projekt erstellen
build <platforms>	Das Projekt für eine bestimmte Plattform erstellen
install <platforms>	Das Projekt auf einer bestimmten Plattform installieren (Anzeige auf einem Endgerät)
run <platforms>	Das Projekt für eine bestimmte Plattform erstellen und installieren
platform [command]	Eine Plattform Version aktualisieren
plugin [command]	Plugins hinzufügen, löschen und auflisten
template [command]	Verfügbare App-Templates auflisten
info	Projekt-Informationen anzeigen
serve	Das Projekt auf einem lokalen node-Server starten (Anzeige im Browser)
version	Ausgabe der Versionsnummer von PhoneGap
analytics	Analytics ein oder ausschalten, oder den aktuellen Status anzeigen
report-issue	Öffnet ein Browserfenster mit einem vorgefertigten github.com-Issue
<b>Weitere Befehle</b>	
local [command]	Entwicklung auf einem lokalen System
remote [command]	Entwicklung in der Cloud mit phonegap/build
prepare <platforms>	Kopiert den www-Ordner in den platform-Ordner, bevor das Projekt kompiliert wird
compile <platforms>	Kompiliert das Projekt für eine Plattform ohne es vorzubereiten
emulate <platforms>	Startet das Projekt in einem Emulator
cordova	Ausführung von jedem möglichen cordova Befehl

## Anhang 5 Wichtigste Klassen der Anwendung als UML-Diagramm

