

# **Thesis**

zur Erlangung des Grades

## **Master**

im Studiengang Medieninformatik Master

an der Fakultät Digitale Medien

# **Entwicklung einer Augmented Reality App zur Visualisierung historischer Gebäude der Stadt Villingen**

Erstbetreuer: Prof. Dr. Uwe Hahne

Zweitbetreuer: Prof. Dr. Thomas Schneider

Abgegeben am: 31.08.2022

Abgegeben von: Oliver Kusch

Matrikelnummer: 268513

Erbsenlachen 6, 78050 Villingen-Schwenningen

oliver.kusch@hs-furtwangen.de



## Abstract

Durch das Projekt NISABA<sup>1</sup> und der Veranstaltung Bildverarbeitung und Computergrafik im Studiengang Medieninformatik Master im Sommersemester 2021 sind 3D-Modelle der Kasernengebäude des Lyautey- und Mangin-Geländes entstanden. Um die fertigen 3D Modelle für jeden zugänglich und auf moderner Weise präsentieren zu können, wird in dieser Master-Arbeit eine Augmented Reality Anwendung für Smartphone und Tablet Geräte entwickelt.

Diese Forschungsarbeit beschäftigt sich mit der Entwicklung dieser Augmented Reality AR Anwendung. Durch genaues Tracking und eine realistische Darstellung der Gebäude durch wetterspezifische Belichtung, Schattierung und Spiegelung soll eine hohe Immersion erschaffen werden. In der Anwendung werden die 3D-Objekte in der Größendarstellung 1:1 über das Videobild der Smartphone Kamera gelegt und der Nutzer kann vor Ort das Gebäude in Echtzeit ein- und ausblenden lassen. Bekannte Beispiele sind Pokemon GO<sup>2</sup> oder Googles Holo Lens<sup>3</sup>. In der Fachsprache werden diese Anwendungen *world-scale Augmented Reality* genannt.

Eine wesentliche Rolle bei der Immersion in AR spielt das Tracking, bei dem kontinuierlich die Positions- und Orientierungsdaten des Endgeräts erfasst werden. Es gibt mehrere Verfahren, um die Position und Orientierung der Kamera relativ zur Umgebung zu bestimmen. Im begrenzten Räumen ist das Tracking durch die einheitliche Belichtung und vorhandenen Kanten und Flächen mit kamerabasierten Tracking Methoden gut umsetzbar. Im Freien kann das Tracking je nach Umgebung Probleme bereiten. Schwierigkeiten entstehen durch hohe Dynamik und dadurch, dass die Umgebung unbegrenzt ist. Hinzu kommt, dass durch die große Distanz zwischen Kamera und virtuellem Objekt die Platzierung des 3D-Modells bereits durch kleine Bewegung der Kamera stark von der korrekten Position abweicht. Das Objekt erscheint nicht homogen in der realen Welt, wodurch die Immersion beeinträchtigt wird.

Klassischerweise wird bei AR im Freien GPS und die Neigungs- und Beschleunigungssensoren der Smartphones genutzt, um die Kameraposition und -orientierung zu ermitteln. Wie Platinsky et al.[32] bereits erörtert ist die GPS Lokalisierung insbesondere in Städten mit Störfaktoren wie Gebäuden oder Vegetation ungenau. Deshalb wird in dieser Arbeit auf Methoden eingegangen, um die Genauigkeit der Position und der Orientierung der Kamera im Freien zu verbessern.

Um die Immersion bei der Nutzung der App zu steigern, werden die Wetterbedingungen bei der Darstellung der 3D-Modelle berücksichtigt. So sollen die Fassaden bei regnerischem Wetter dunkler und feucht bzw. nass dargestellt werden. Die Schattenwürfe sollen sich anpassen, indem bei hartem Licht (z.B. durch starke Sonneneinstrahlung) auch harte Schatten und bei weichem Licht (z.B. bei einer dichten Wolkendecke) weiche Schatten dargestellt werden.

---

<sup>1</sup><https://nisaba.dm.hs-furtwangen.de/>

<sup>2</sup><https://pokemongolive.com/de/>

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

Fazit:



# Inhaltsverzeichnis

<b>Abstract</b>	<b>III</b>
<b>Abkürzungsverzeichnis</b>	<b>VIII</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Hintergrund der Arbeit . . . . .	2
1.1.1 Photogrammetrische Aufzeichnungen des SABA Geländes . . . . .	2
1.1.2 Photogrammetrische Aufzeichnungen des Lyautey Geländes . . . . .	3
1.1.3 Photogrammetrische Aufzeichnungen des Mangin Geländes . . . . .	3
1.1.4 Übersicht vorhandener 3D Modelle . . . . .	4
<b>2 Grundlagen</b>	<b>6</b>
2.1 Einordnung VR, AR und MR . . . . .	6
2.2 Immersion im AR Kontext . . . . .	7
2.3 Tracking . . . . .	8
2.3.1 Koordinatensysteme . . . . .	9
2.3.2 Tracking mit der Inertial Measurement Unit (IMU) . . . . .	9
2.3.3 Kamera-basiertes Tracking . . . . .	10
2.3.3.1 Schlüsselpunkte detektieren (Feature Detection) . . . . .	11
2.3.3.2 Feature-Matching . . . . .	16
2.3.3.3 Simultaneous Localization and Mapping (SLAM) . . . . .	17
2.3.4 Tracking in AR Core . . . . .	18
2.4 Grafik-Rendering-Pipeline . . . . .	19
2.4.1 Applikationsstufe . . . . .	20
2.4.2 Geometrieverarbeitung . . . . .	20
2.4.2.1 Vertex-Shading . . . . .	21
2.4.2.2 Projektion . . . . .	22
2.4.2.3 Clipping . . . . .	22
2.4.2.4 Window-Viewport-Transformation . . . . .	22
2.4.3 Rasterisierung . . . . .	23
2.4.4 Pixelverarbeitung (Fragment-Shader) . . . . .	23
<b>3 Konzeption der Anwendung</b>	<b>24</b>
3.1 Idee und Ablauf . . . . .	24
3.2 Platzierung auf einer beliebigen Fläche . . . . .	27

3.3 Platzierung über GPS . . . . .	28
3.4 Licht und Wetterbedingungen ändern . . . . .	28
<b>4 Vorbereitung der 3D Modelle in Blender</b>	<b>29</b>
4.1 Lückenhafte Wände füllen . . . . .	29
4.2 Export und Import . . . . .	32
<b>5 Technische Umsetzung</b>	<b>33</b>
5.1 Verwendete Software und Hardware . . . . .	33
5.2 ARCore SDK . . . . .	35
5.2.1 Technische Voraussetzungen . . . . .	35
5.2.2 Motion Tracking . . . . .	35
5.2.3 Environmental understanding . . . . .	36
5.2.4 Light estimation . . . . .	36
5.2.5 User Interaction . . . . .	38
5.2.6 Depth API . . . . .	39
5.3 Umsetzung der Anwendung . . . . .	41
5.3.1 Grundaufbau der AR Szenen in Unity . . . . .	41
5.3.2 Controller . . . . .	44
5.3.2.1 Platzieren auf einer beliebigen Fläche . . . . .	44
5.3.2.2 Objekte auswählen . . . . .	47
5.3.3 GPS Platzierung . . . . .	47
5.3.3.1 Ausrichtung der Szene nach Norden . . . . .	49
5.4 Belichtung . . . . .	52
5.4.1 Light Estimation in dieser Anwendung . . . . .	52
5.4.2 Sonne simulieren . . . . .	52
5.5 Wetterbedingungen . . . . .	52
5.5.1 Anbindung der REST-API . . . . .	52
5.5.2 Veränderungen der Wetter Parameter . . . . .	53
5.6 Verwendete Shader . . . . .	53
5.7 Probleme und Lösungen . . . . .	54
5.7.1 Grenzen der Depth API und Occlusion . . . . .	54
<b>Abbildungsverzeichnis</b>	<b>57</b>
<b>Eidesstattliche Erklärung</b>	<b>59</b>

## Abkürzungsverzeichnis

**AR** Augmented Reality

**CPU** Central Processing Unit

**DOF** Degrees of Freedom

**FAST** Features from accelerated segment test

**FPS** Frames Pro Sekunde

**GPU** Graphics Processing Unit

**GRP** Grafik-Rendering-Pipeline

**IMU** Inertial Measurement Units

**JSON** JavaScript Object Notation

**Materials** Materials affektieren das Aussehen eines 3D Objekts mit einer oder mehrerer Texturen, Farben oder den Eigenschaften von Reflexionen

**MR** Mixed Reality

**ORB** Oriented Fast and Rotated Brief

**PTAM** Parallel Tracking and Mapping

**REST** Representational State Transfer

**SDK** Software Development Kit

**SIFT** Scale Invariant Feature Transform

**SLAM** Simultaneous Localization and Mapping

**SURF** Speeded Up Robust Features

**UML** Unified Modeling Language

**VE** Virtual Environments

**VR** Virtual Reality

## 1 Einleitung

Durch vorangegangene Projekte sind mit photogrammetrischen Methoden 3D Modelle von historisch bedeutenden Gebäuden des SABA, des Lyautley- und des Mangin-Geländes der Stadt Villingen entstanden. Die Gebäude sind virtuell gespeichert, jedoch werden diese nicht weiter verwendet. In dieser Arbeit wird eine Augmented Reality Anwendung für Smartphones entwickelt, die eine Visualisierung der Modelle mit der AR Technologie bietet. Somit haben Bewohner der Stadt Villingen und interessierte Personen wie Touristen, die mehr über die Geschichte der Stadt Villingen erfahren möchten, Zugriff auf die Modelle und können damit einen Teil der Stadtgeschichte erleben.

Die Aufgabe besteht darin eine Anwendung zu entwickeln, die zum einen die Gebäude mittels GPS vor Ort platziert. Die Gebäude werden dabei im Größenverhältnis 1:1 dargestellt, sodass der Nutzer einen realistischen Eindruck bekommt, wie das Gebäude in Echt ausgesehen hat. Zum anderen ist es möglich die Gebäude auf jeder beliebigen horizontalen Fläche zu platzieren. Die Anwendung baut auf aktuelle AR Software Development Kits (ARCore bzw. ARKit) auf und wird mit der Game Engine Unity und AR Foundation entwickelt.

Ziel ist es eine hohe Immersion bei der Nutzung zu schaffen. Das Gebäude soll nahtlos im Kamerabild dargestellt werden. So wird eine Wetter REST-API angebunden (OpenWeatherMap), um aktuelle Wetterdaten abzurufen. Daraufhin wird die Darstellung der Modelle angepasst. Bei Regen wird die Szene dunkler, Materialien werden nass dargestellt und spiegeln die Umgebung. Herrscht starker Sonnenschein, so wird die Intensität des Lichts verändert und harte Schatten geworfen.

Diese Arbeit untersucht die Möglichkeiten von AR und dessen Limitierungen in einer freien Umgebung. Mit der GPS Funktionalität werden Genauigkeitsprobleme der GPS-Antennen in Smartphones untersucht.

In dieser Arbeit werden zunächst einleitend theoretische Grundlagen von Augmented Reality beschrieben. Auf der technischen Seite wird das Tracking, die Algorithmen zur Erkennung von Merkmalen, GPS Systeme und die Rendering Pipeline erläutert. Anschließend wird die Umsetzung der Anwendung beschrieben, indem auf das Konzept und die Implementierung der genannten Funktionalitäten eingegangen wird. Daraufhin werden bestehende Probleme und Limitierungen, die während der Entwicklung und dem Testing vorgekommen sind, veranschaulicht. Zuletzt werden Ideen für Erweiterungen und Verbesserungen der Anwendung vorgeschlagen.

## 1.1 Hintergrund der Arbeit

### 1.1.1 Photogrammetrische Aufzeichnungen des SABA Geländes

Das studentische Forschungsprojekt aus dem Wintersemester 19/20 [33] befasst sich mit der Photogrammetrischen Aufzeichnung des SABA-Geländes in Villingen-Schwenningen. SABA (Schwarzwälder Apparate-Bau-Anstalt) war ein deutsches Unternehmen, das unter anderem elektronische Geräte für den Rundfunk herstellte. Der Entwicklungs- und Produktionsstandort war das Gebäude in Villingen. Aufgrund der Größe des Unternehmens (mehr als 6000 Mitarbeiter), hatte SABA eine große Bedeutung für die Stadt. 1986 wurde das Unternehmen aufgelöst, bis das Gebäude am 11. August 2021 abgerissen wurde.<sup>4</sup>

Mithilfe von Drohnen- und Bodenaufnahmen wird ein 3D Modell des SABA Geländes mit photogrammetrischen Algorithmen erzeugt. *Regard3D*<sup>5</sup>, *RealityCapture*<sup>6</sup>, *VisualSFM*<sup>7</sup> und *Meshroom*<sup>8</sup> werden im Projektverlauf genutzt und verglichen, wobei *Meshroom* als kostenlose Open-Source Software hauptsächlich genutzt wird. Das Hauptgebäude wird nach der Erstellung des Meshes aus *Meshroom* neu modelliert. Aufgrund der hohen Dichte der Vertices ist die Aufteilung der einzelnen Gebäude-Elemente wie Fenster, Wand und Türen schwierig. Durch eine Neumodellierung wird das Modell klarer und die Texturierung wird vereinfacht. Als Grundlage dient dabei das generierte Mesh aus *Meshroom*. Als Modellierungssoftware wird *Blender* benutzt.<sup>9</sup> In Abbildung 1 ist das fertige 3D Modell zu sehen.



Abbildung 1: Das 3D Modell des SABA Hauptgebäudes in der 3D Karte aus den Drohnen-Aufnahmen.

<sup>4</sup><https://dewiki.de/Lexikon/SABA>, zuletzt aufgerufen am 17.03.2022

<sup>5</sup><https://www.regard3d.org/>

<sup>6</sup><https://www.capturingreality.com/>

<sup>7</sup><http://ccwu.me/vsfm/>

<sup>8</sup><https://github.com/alicevision/meshroom>

<sup>9</sup><https://www.blender.org/>

### 1.1.2 Photogrammetrische Aufzeichnungen des Lyautey Geländes

Im Projekt *NISABA* [36] aus dem Sommersemester 2020 und Wintersemester 2020/21 sind 3D Modelle von Gebäuden des ehemaligen Lyautey Kasernengeländes (das heutige "Richthofen") entstanden. Auch in diesem Projekt werden verschiedene Photogrammetrie Programme genutzt. *Meshroom*, *Pix4DMapper*<sup>10</sup>, *Agisoft Metashape*<sup>11</sup> und *WebODM*<sup>12</sup> werden für dieses Projekt verwendet. Gute Ergebnisse erzielt dabei *Pix4DMapper*, da viele Einstellungen über den Detailgrad getroffen und mehrere Projekte kombiniert werden können. Aus dem generierten Mesh wird auch hier eine Nachmodellierung in *Blender* durchgeführt.

Das Lyautey Gelände umfasst insgesamt sieben Gebäude. Für jedes Gebäude gibt es ein fertiges Modell aus *Pix4DMapper* und ein Modell, bei dem die Polygone reduziert sind. Nur das Mannschaftsgebäude gibt es nachmodelliert. Das ist das Gebäude 4 in der Abbildung 2. Alle 3D Modelle können auf der Website des *NISABA*-Projekts<sup>13</sup> begutachtet werden.



Abbildung 2: Eine Übersicht der Gebäude auf dem Lyautey-Gelände.

### 1.1.3 Photogrammetrische Aufzeichnungen des Mangin Geländes

Im Zuge der Veranstaltung Bildverarbeitung und Computergrafik im Sommersemester 2021 im Studiengang Medieninformatik Master sind weitere 3D Modelle entstanden[25]. Die Gebäude befinden

<sup>10</sup><https://www.pix4d.com/de/produkt/pix4dmapper-photogrammetrie-software>

<sup>11</sup><https://www.agisoft.com/>

<sup>12</sup><https://www.opendronemap.org/webodm/>

<sup>13</sup><https://nisaba.villingen-schwenningen.de/uebersicht/>

sich auf dem für die Stadt historisch wichtigen Kasernengelände Mangin in Villingen-Schwenningen, das sich direkt östlich vom Lyautey befindet. Dabei handelt es sich um ein verlassenes Kasernengelände mit architektonisch und historisch interessanten Gebäuden. Die Aufnahmen der Gebäude erfolgte in Zusammenarbeit mit dem Vermessungsamt Villingen-Schwenningen. Es wurden Aufnahmen von den Gebäuden 2-8 und 10-12 gefertigt. In der Abbildung x ist eine Übersichtskarte des Geländes mit Nummerierungen der Gebäude zu sehen. Die Bilder aus der Luft wurden mit einer Drohne vom Vermessungsamt gemacht, während die Aufnahmen am Boden von den Studierenden aufgenommen wurden. Als Photogrammetrie-Software wird hauptsächlich *Pix4DMapper* und *Meshroom* verwendet.



Abbildung 3: Eine Übersicht der Gebäude auf dem Mangin-Gelände.

#### 1.1.4 Übersicht vorhandener 3D Modelle

Einige Modelle sind nicht vollständig, haben aufgrund der Vegetation vor Ort Löcher im Mesh oder einen niedrigen Detailgrad. Daher können in dieser Arbeit nicht alle 3D Modelle verwendet werden. Die Tabelle 1 zeigt eine Liste der vorhandenen 3D Modelle. Die Gebäude sind wie in den Übersichtskarten nummeriert. Ein geringer Detailgrad bedeutet, dass das Mesh Löcher hat oder unvollständig ist. Ein mittlerer Detailgrad zeichnet sich durch ein vollständiges Mesh mit geringen Details aus. Gebäude mit einem hohen Detailgrad können für die Augmented Reality (AR) Anwendung genutzt werden. Händisch modellierte Gebäude haben einen sehr hohen Detailgrad.

Gelände	Bezeichnung	Nr.	Detailgrad	Vertices	Texturenanzahl
SABA	Karte	-	gering	458.311	1
SABA	Hauptgebäude Neumodellierung	1	sehr hoch	44.708	34
SABA	Heizwerk	2	mittel	142	8
Lyautey	Karte	-	mittel	2.049.590	1
Lyautey	Reithalle	1	hoch	1.386.531	1
Lyautey	Mannschaftsgebäude 1	2	hoch	1.255.734	1
Lyautey	Wirtschaftsgebäude	3	hoch	1.610.983	1
Lyautey	Mannschaftsgebäude 2	4	hoch	175.352	1
Lyautey	Mannschaftsgebäude 2 Neumodellierung	4	sehr hoch	2.046	45
Lyautey	Stabshaus	5	gering	-	-
Lyautey	Familienhaus	6	mittel	119.574	1
Lyautey	Kammergebäude	7	hoch	269.460	1
Mangin	Karte	1	mittel	502.040	1
Mangin	Casino	2	mittel	681.082	1
Mangin	-	3	hoch	381.633	1
Mangin	Pferdestall	4	mittel	242.984	1
Mangin	-	6	gering	500.463	1
Mangin	-	7	gering	31.490	1
Mangin	-	8	hoch	5.093	1
Mangin	-	10	hoch	500.307	1
Mangin	-	11	mittel	427.753	1
Mangin	-	12	mittel	497.625	1
Mangin	-	17	gering	5.179	1
Mangin	-	23	gering	998	1
Mangin	Karte der Gebäude 17-23	-	mittel	15.433	1

Tabelle 1: Eine Übersicht der vorhandenen 3D Modelle.

## 2 Grundlagen

### 2.1 Einordnung VR, AR und MR

Augmented Reality (AR) kombiniert reale Umgebungen mit virtuellen dreidimensionalen Content. Die virtuellen Objekte werden in Echtzeit dargestellt und sind interaktiv. Es handelt sich um eine Variation der Virtual Environments (VE), der Virtuellen Realität (VR) [Azuma, 5]. Gerne wird zwischen AR, VR und Mixed Reality (MR) unterschieden. Speicher et.al [39] haben zur Eingrenzung Charakteristiken definiert, die aus Interviews mit Experten hervorgehen.

VR hat die Eigenschaft eine komplett synthetisch generierte, virtuelle Welt darzustellen. So bietet sich die Möglichkeit für den Nutzer unerreichbare Orte zu erleben. Es werden hierfür VR-Headsets wie die Oculus Rift S<sup>14</sup> oder die Playstation VR<sup>15</sup> benötigt und die Bewegung des Headsets muss getrackt<sup>16</sup> werden. Der Nutzer ist während der VR-Erfahrung von der echten Umgebung isoliert, wodurch die soziale Interaktion gering ist.

Zusätzlich zu der Definition Azumas[5] von AR wird die Eigenschaft genannt, dass der virtuelle Content dazu in der Lage ist mit der echten Welt zu interagieren. Im Gegensatz zu VR ist der Nutzer im gegenwärtigen physischen Raum gebunden. Die menschliche Wahrnehmung wird durch das augmentieren und der Kreation von Content Erfahrungen verbessert[Speicher, 39]. Zu den technischen Voraussetzungen gehört die Nutzung eines Displays, um die virtuellen und realen Bilder zu kombinieren. Es wird ein Computer System gebraucht, um die Objekte zu generieren und Interaktion zu ermöglichen. Letzlich wird ein Tracking System benötigt. Zum einen um die Position des Nutzers zu bestimmen. Zum anderen wird es ermöglicht die virtuellen Objekte in der realen Welt fixiert erscheinen zu lassen[Billinghurst, 8].

Die Eingrenzung von MR ist nicht eindeutig. So wird MR als Kombination von AR und VR gesehen. Die Anwendungen bieten die Möglichkeit sowohl AR als auch VR zu nutzen. Auch wird Mixed Reality in Verbindung mit spezifischer Hardware wie die HoloLens<sup>17</sup> gebracht. Ein weiteres Beispiel ist Pokemon GO<sup>18</sup>, bei dem zwischen der AR Funktion und einer virtuellen Umgebung umgeschaltet werden kann.

In Abbildung 4 haben Milgram et.al eine Grafik definiert, die das *Reality-Virtuality Continuum* darstellt. Die linke Hälfte des Kontinuums repräsentiert jede Umgebung, die rein aus realen Objekten besteht, persönlich betrachtet wird und durch jegliche Art wie den Blick durch ein Fenster oder eines Displays erweitert wird. Die rechte Hälfte besteht aus rein virtuellen Objekten, die entweder

---

<sup>14</sup><https://www.oculus.com/rift-s/>, zuletzt aufgerufen am 27.07.2022

<sup>15</sup><https://www.playstation.com/de-de/ps-vr/>, zuletzt aufgerufen am 27.07.2022

<sup>16</sup>siehe Kapitel 2.3 Tracking

<sup>17</sup><https://www.microsoft.com/de-de/hololens>, zuletzt aufgerufen am 27.07.2022

<sup>18</sup><https://pokemongolive.com/de/>, zuletzt aufgerufen am 27.07.2022

Monitor-basierend oder immersiv sind. Durch diese Darstellung wird MR zwischen diesen beiden Extremen eingeordnet und als Umgebung definiert, die die reale und die virtuelle Welt zusammen in einem Display darstellt [29].

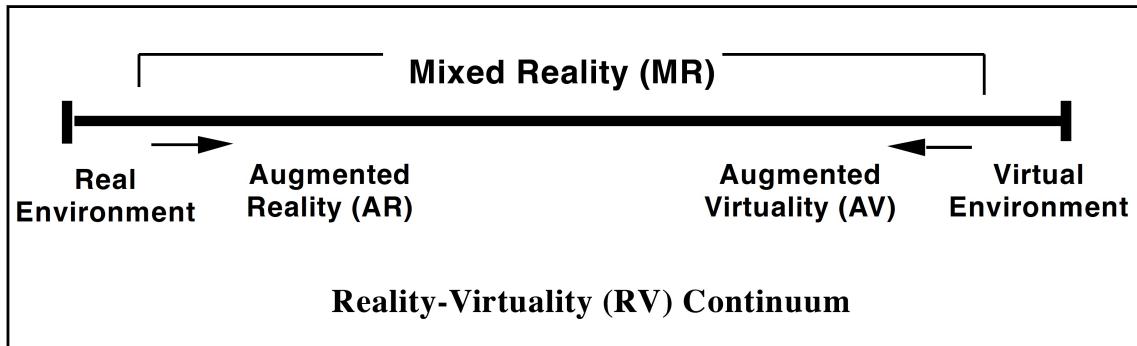


Abbildung 4: Das Reality-Virtuality Kontinuum nach Milgram[29].

Nach diesen Definitionen wird die Anwendung dieser Arbeit in die Augmented Reality eingeordnet. Die virtuellen Objekte in Form der Gebäude werden in die echte dreidimensionale Welt projiziert. Der Nutzer hat die Möglichkeit mit diesen zu interagieren. Jedoch wird es nicht möglich sein den betrachteten Raum zu verlassen und in eine reine virtuelle Welt umzusteigen.

## 2.2 Immersion im AR Kontext

### 2.3 Tracking

Die Bewegung eines starren Körpers wird mit drei Koordinaten für die Position und drei Winkeln für die Orientierung angegeben[Dörner (2019) S.119f., 13]. Diese sechs Werte werden als *Freiheitsgrade* (engl. *Degrees of Freedom (DOF)*) bezeichnet. Der Begriff Tracking beschreibt die kontinuierliche Verfolgung von Positions- und Orientierungsdaten.

Im AR Kontext wird die Position und Orientierung des Smartphones kontinuierlich berechnet. Gleichzeitig wird die Umgebung erfasst und sogenannte Schlüsselpunkte festgelegt. Anhand der Schlüsselpunkte können z.B. Flächen erkannt werden, auf denen die virtuellen Objekte platziert werden. Billinghurst[8] nennt zwei Phasen beim Tracking:

- eine Registrierungsphase, bei der die Position und Orientierung des Smartphones beim Start der Anwendung im Bezug auf einem Ankerpunkt in der realen Welt bestimmt wird,
- eine Trackingphase, bei der die Position und Orientierung des Smartphones anhand der vorherigen Position und Orientierung aktualisiert wird.

Das Ziel beim Tracking ist es, die sechs Freiheitsgrade für die Translation und Rotation der Objekte kontinuierlich zu bestimmen bzw. zu schätzen. Die Aufnahme der Daten erfolgt durch Sensoren z.B. durch die *Inertial Measurement Units (IMU)* in Smartphones und Tablets. Die IMU wird in Kapitel 2.3.2 behandelt.

Es werden zwischen zwei Trackingverfahren unterschieden. Beim *Outside-In-Tracking* befindet sich die Sensorik zur Datenaufnahme in der Umgebung innerhalb eines bestimmten Raumes, sodass das Objekt von außen getrackt wird. Dies wird bei VR-Brillen eingesetzt. Der Nachteil dieser Technik ist, dass die Sensoik an einem Ort gebunden ist und bei einem Ortswechsel neu platziert werden muss. Beim *Inside-Out-Tracking* befinden sich Sensoren im Objekt, das getrackt werden soll. Die Position und Lage des Objekts wird im Verhältnis zur Umgebung gesetzt. Bei der AR Anwendung in dieser Arbeit handelt es sich somit um ein Inside-Out-Tracking. Vorteil dieser Methode ist, dass der Nutzer nicht an einem Ort gebunden ist. Probleme treten in der Genauigkeit beim Tracking auf. Das Tracking System im Smartphone muss sich rein auf die Sensoren in der IMU und dem Kamerabild verlassen, während beim Outside-InTracking mehr Sensoren zur Verfügung stehen.

In der Unity Manual zu AR Foundation wird der Begriff Tracking mit der Bestimmung der relativen Position und Orientierung in der physichen Welt definiert. Zusätzlich wird darauf hingewiesen, dass falls die Umgebung zu dunkel ist, das Gerät Probleme beim Tracking bekommt und die Genauigkeit der Positionsbestimmung sich verringert [41]. Damit ist davon auszugehen, dass in AR Foundation Kamera-basiertes Tracking (siehe Kapitel 2.3.3) verwendet wird. Kamera-basiertes Tracking wird in Kapitel 2.3.3 näher behandelt.

### 2.3.1 Koordinatensysteme

Um eine Bestimmung bzw. Schätzung der Translationen und Rotationen durchzuführen, werden zwei Koordinatensysteme herangezogen. Ein Kamerakoordinatensystem und ein Objektkoordinatensystem. Weiterhin gibt es die Möglichkeit, dass für alle Objekte im Raum ein Koordinatensystem (Weltkoordinatensystem) verwendet wird. Voraussetzung für das Tracking ist, dass die Transformationen zwischen den Objekten bekannt sind. Dann kann die Transformation zwischen dem Objekt und dem Weltkoordinatensystem geschätzt werden.[Dörner (2019) S.124f., 13].

In Unity und AR Foundation hat das Smartphone ein eigenes Koordinatensystem. Wird eine AR-Session gestartet, so wird ein Koordinatensystem für diese spezifische Session initialisiert. Die Koordinaten (0,0,0) stehen für die Position des Gerätes, bei dem die Session gestartet ist [41]. Die z-Achse zeigt dabei in die Richtung, in die die Kamera des Smartphones gerichtet ist. Dieser Umstand ist für die Roatation der Gebäude in der GPS Platzierung relevant und wird in Kapitel 5.3.3.1 erläutert. Die *GameObjects* in der Szene haben ein lokales Objektkoordinatensystem.

### 2.3.2 Tracking mit der Inertial Measurement Unit (IMU)

Eine Inertial Measurement Unit besteht typischerweise aus Beschleunigungssensoren, Drehratensensoren. Drei Beschleunigungssensoren und drei Drehratensensoren werden orthogonal zueinander eingebaut. Die Sensoren bilden ein Trägheitsnavigationssystem. Dieses misst die Bewegungsrichtung, die Beschleunigung und die Drehung in einem kartesischen Koordinatensystem und bestimmt damit die Position und Orientierung der IMU<sup>19</sup>. Die Orientierung kann anhand der Drehratensensoren präzise bestimmt werden. Die Position wird mit den Beschleunigungswerten und der dabei vergangenen Zeit pro Aktualisierung berechnet. Die Genauigkeit dieser Berechnung ist insbesondere bei niedrigpreisigen Smartphones nicht akkurat. Es kommt zu *Drifteffekten*. Dörner beschreibt den Hintergrund des Probelms präzise:

„[...] wird beispielsweise ein Sensor aus dem Ruhezustand bewegt und anschließend wieder angehalten, so müssten sowohl die Summen der erfassten Beschleunigungswerte wie auch die der errechneten Geschwindigkeitswerte am Ende Null ergeben“[Dörner (2019) S.127f., 13]. Durch die Ungenauigkeit wird dies nicht erzielt und die errechnete Position weicht von der tatsächlichen Position ab. Zur Bestimmung der Orientierung wird die IMU oft genutzt. Da in AR Anwendungen eine hohe Präzision in der Positionsbestimmung benötigt wird, wird reines IMU-Tracking nicht genutzt. Es wird zusätzliches Kamera-basiertes Tracking benötigt, um den Drifteffekt auszugleichen[Billinghurst (2015), 8].

---

<sup>19</sup><https://www.5gpositioning.com/inertial-measurement-units-in-smartphones/>, zuletzt aufgerufen am 27.07.2022

### 2.3.3 Kamera-basiertes Tracking

Kamera-basiertes Tracking nutzt Bilder, um die relative Position und Orientierung der Objekte zur Kamera zu bestimmen. Die Position und Orientierung der Kamera in einem Weltkoordinatensystem wird von Hartley und Zisserman als extrinsische Kameraparameter bezeichnet [Hartley, Zisserman (2003) S.156, 19]. Für das Kamera-basierte Tracking wird zwischen Marker-basierten und Marker-less Tracking unterschieden.

Beim Marker-basierten Tracking werden Marker (sogenannte Kanji und Hiro Marker) wie in Abbildung 5 zu sehen genutzt. Billinghurst[8] bezeichnet dies als *Fiducial Tracking*. Dabei werden künstliche Marker in der Umgebung platziert. Sie dienen als Ursprung, sodass diese als Orientierungshilfen fungieren. Populär wurde die Methode durch ARToolkit, das von Kato und Billinghurst[(1999), 22] entwickelt wurde. Das jeweilige Muster und die Größe der Marker müssen für das Tracking bekannt sein. Der Tracking Prozess ist in Abbildung 6 zu sehen. Zunächst wird das Bild im Videodatenstrom binarisiert. Die Ränder des Markers werden identifiziert. Anschließend wird die Position und die Orientierung relativ zur Kamera berechnet. Die Symbole im Marker werden mit den bekannten Symbolen aus der Datenbank verglichen und gematched. Letzlich wird das virtuelle Objekt mit der Position und Orientierung des Markers transformiert und in das Kamerabild gerendert.



Abbildung 5: Kanji und Hiro Marker haben einfache geometrische Formen, Texte oder Buchstaben zur Erkennung in der Szene<sup>20</sup>.

<sup>20</sup>Quelle Bild: <https://stemkoski.github.io/AR-Examples/>, zuletzt aufgerufen am 29.07.2022

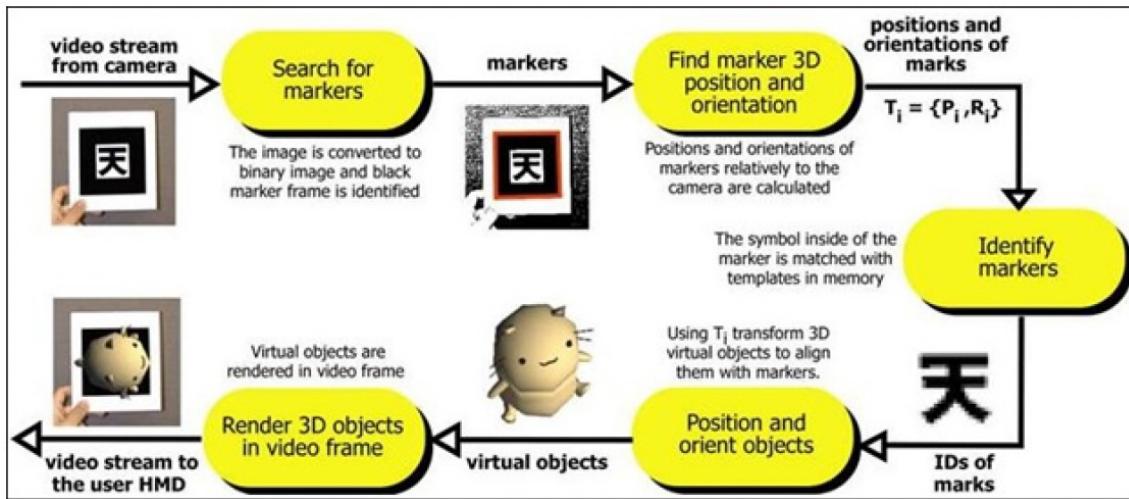


Abbildung 6: Der Tracking Prozess von ARToolkit[Billinghurst(2015), 8].

Die Methode ist simpel und bietet eine hohe Genauigkeit beim Tracking. Da die Marker aktiv in der Umgebung platziert werden müssen, gibt es Einschränkungen in der Flexibilität. Daher wird bei der zweiten Möglichkeit über Algorithmen der Computer Vision Merkmale in der realen Umgebung erfasst. Dies wird als *marker-less AR* bezeichnet. Diese Merkmale sind Punkte oder Ecken, die aus der Umgebung herausstechen und als einzigartigen Punkt gelten. Sie sind natürliche Marker in der Umgebung. In den folgenden Kapiteln werden natürliche Merkmals-Detektoren benannt und deren Funktionsweise anhand des Scale Invariant Feature Transform (SIFT) Algorithmus erklärt.

### 2.3.3.1 Schlüsselpunkte detektieren (Feature Detection)

Merkmale oder auch *keypoint features* oder *interest points* sind Bildbereiche mit einem zentralen Punkt, die einen hohen Wiedererkennungswert besitzen. In Abbildung 7 wird deutlich, dass Flächen ohne Texturen schwer identifiziert und abgeglichen werden können. Dieses Problem ist als *aperture Problem* bekannt. Erst durch starke Kontraständerungen an der Richtung der Normalen zur Kante<sup>21</sup>, werden Merkmale erkannt[Szeliski(2022) Seite 337f., 40].

<sup>21</sup>die durch Gradienten erkannt werden. Ein Gradient bestimmt die Richtung und Steigung der größten Änderung.

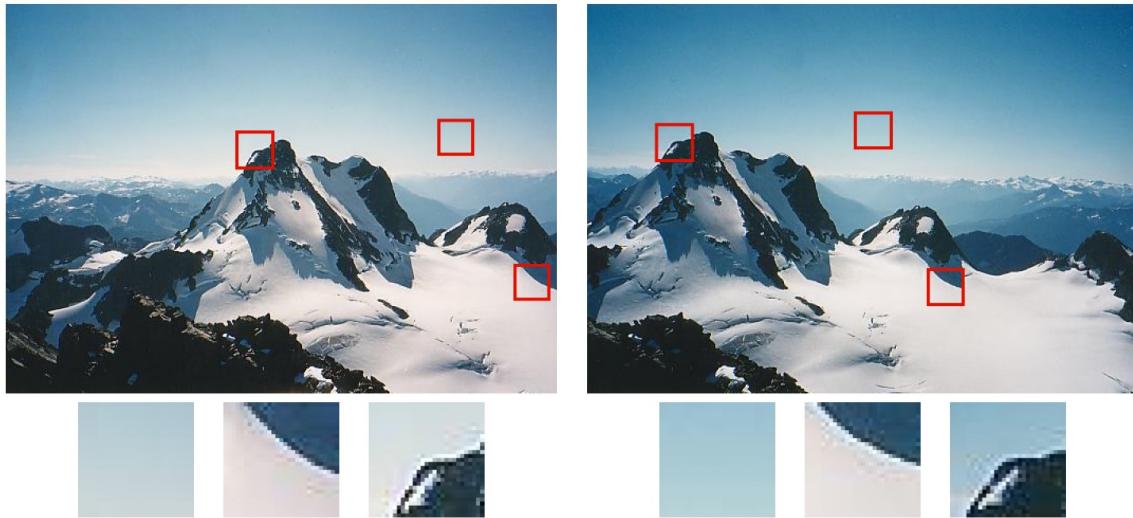


Abbildung 7: Die Wahrscheinlichkeit, dass eine Fläche aus dem linken Bild im rechten Bild genau identifiziert werden kann steigt, wenn starke Kontraständerungen (Gradienten) im Teilbild vorhanden sind<sup>22</sup>.

Es wird zwischen *Eckendetektoren* und *Blob Detektoren* unterschieden. Bekannte Kantendetektoren sind z.B.:

- Harris[18],
- Features from accelerated segment test (FAST) von Rosten und Drummond[34].

Bekannte Blob-Detektoren sind z.B.:

- Scale Invariant Feature Transform (SIFT) von Lowe[28],
- Speeded Up Robust Features (SURF) von Bay u.a.[7],
- Oriented Fast and Rotated Brief (ORB) von Rublee u.a.[35].

Herling und Broll[20] nennen Herausforderungen für Detektoren, damit eine valide Position und Orientierung berechnet werden kann:

- Da AR in Echtzeit abläuft, muss die Bestimmung von Schlüsselpunkten und Deskriptoren schnell berechnet werden,
- der Algorithmus muss robust gegenüber unterschiedlichen Lichtverhältnissen. Dies gilt insbesondere bei AR in freier Umgebung, da eine hohe Dynamik herrscht und die Lichtverhältnisse sich schnell verändern können,

<sup>22</sup>Quelle Bild: [Szeliski(2022) Seite 337, 40]

- da der Nutzer keine Einschränkungen in der Position der Kamera besitzt, kann ein schneller Perspektivwechsel erfolgen. Der Algorithmus muss daher eine Robustheit gegenüber starken Bildveränderungen haben,
- es muss eine Skalierungsinvarianz bereitgestellt werden. Das bedeutet im AR Kontext, dass das Tracking nicht auf eine bestimmte Entfernung beschränkt ist und weiter entfernte Punkte nicht verschwinden. In einem abgegrenzten Bereich ist es einfacher Schlüsselpunkte zu detektieren. Die Skaleninvarianz ist daher insbesondere bei AR im Freien relevant, da hier die Umgebung unbegrenzt ist.

Die Blob Detektoren wählen Schlüsselpunkte aus, die einen fleckenartige Eigenschaft besitzen. Es sind z.B. Punkte oder große verschwommene Flächen, die ähnliche Farbintensitäten besitzen[Herling und Broll(2011), 20]. Im folgenden Abschnitt wird die Herangehensweise anhand des SIFT Algorithmus erläutert.

Zunächst wird eine Gauss Pyramide gebildet. Dabei wird ein Eingangsbild verkleinert und in mehreren Durchläufen mit einem Gauss-Filter weichgezeichnet. Dann erfolgt eine weitere Skalierung und Gauss-Filter Ebene. Eine Ebene wird dabei *Octave* genannt. Dieser Prozess ist in Abbildung 8 zu sehen und wird so lange durchgeführt, bis das Bild so klein ist, dass es nicht weiter verkleinert werden kann. Dadurch entsteht ein Skalenraum, der die Skaleninvarianz implementiert und das Bild in mehreren Skalierungen bereitstellt. Der Sinn dabei ist es, das Bild aus verschiedenen Entfernungen zu betrachten. So ist ein Baum aus der Entfernung durch seinen Aufbau als Feature zu erkennen. Ist der Baum im Bild groß skaliert, werden die Blätter des Baumes als Feature erkannt. Da die Detektoren keine Vorkenntnisse über das Bild haben, wird dieser Skalenraum erstellt. Der Gauss Filter wird benötigt, um feine Strukturen wie z.B. Bildrauschen oder zu feine Strukturen herauszufiltern.

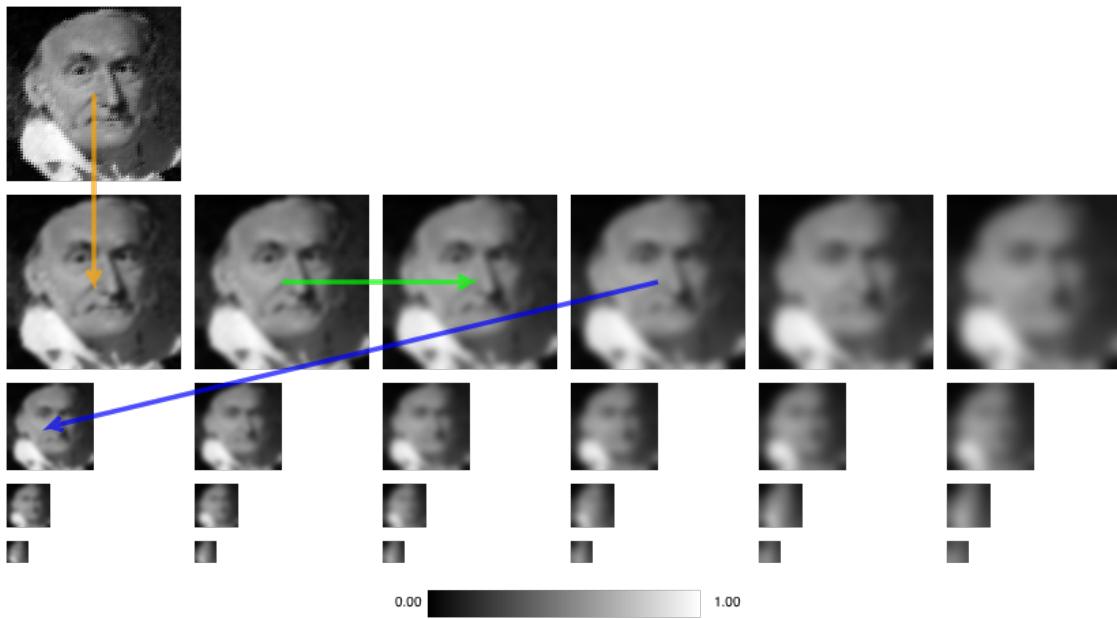


Abbildung 8: Die Gauß-Pyramide für ein Eingangsbild. Das Bild wird runterskaliert und mit steigender Stärke mit einem Gauss-Weichzeichner versehen<sup>23</sup>.

Dann werden nach *Schlüsselpunkten* (engl. *Key Points*, *Feature Points*) in einem Bild gesucht. Es wird von je zwei benachbarten Bildern in der Gauß-Pyramide die Differenz gebildet (*Difference of Gaussians (DoG)*). Dadurch entstehen Grauwerte, die diskrete Maximas erkennen lassen. Für jeden Punkt wird der Grauwert des mittleren Pixels mit den Grauwerten der umliegenden Pixel und die Grauwerte der benachbarten Bilder an der gleichen Stelle verglichen. Ist ein starker Unterschied bemerkbar, gilt der Punkt als Kandidat für einen Schlüsselpunkt. In Abbildung 9 werden Schlüsselpunkte gefunden. Die roten Punkte enthalten starke Extrema, die im weiteren Verlauf betrachtet werden. Die gelben Punkte enthalten zwar Extrema, jedoch sind die Unterschiede der Grauwerte so gering, dass sie unter dem eingestellten Schwellwert liegen und aussortiert werden. Diese werden beispielsweise als Rauschen erkannt. Je nachdem wie groß das Bild ist und wie stark die Schwellwerte beim jeweiligen Detektor eingestellt sind, verändert sich die Anzahl an Schlüsselpunkten. Dies beeinflusst die Tracking Performance und die Genauigkeit. Wie?

<sup>23</sup>Quelle Bild: <http://weitz.de/sift/>, zuletzt aufgerufen am 01.08.2022

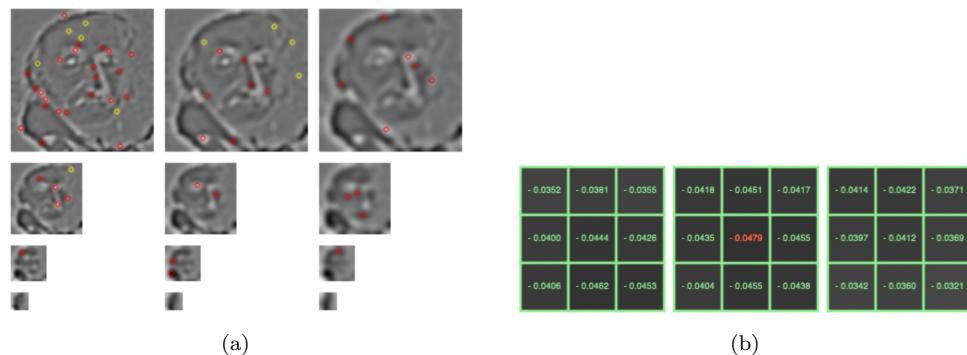


Abbildung 9: Detektierte Schlüsselpunkte anhand der Extremas der DoG Bilder und deren Nachbarbilder (a) und die Grauwerte des Punktes auf der Nasenspitze der Person (b). Der Grauwert an der Stelle der Nase unterscheidet sich stark von seinen benachbarten Pixeln und beherbergt daher ein Extrema<sup>24</sup>.

Nach der ersten Auswahl der Schlüsselpunkte wird die Positionen der Schlüsselpunkte verbessert. Dazu werden mit der Taylor-Formel Taylorpolynome gebildet. Diese werden in der Mathematik genutzt, Punkte in der Umgebung einer Funktion anzunähern<sup>25</sup>. Mit den Polynomen werden die diskreten x- und y-Koordinaten durch Nachkommastellen verfeinert. Mit feineren Koordinaten wird der Punkt genauer erfasst und die Auswahl der Schlüsselpunkte wird ebenfalls verfeinert.

Ist eine finale Auswahl der Schlüsselpunkte erfolgt, erhält jeder Punkt einen sogenannten *Deskriptor*. Dieser enthält lokale Eigenschaften auf dem Bild in der Umgebung des Punktes. Ziel eines Deskriptors ist es Schlüsselpunkte eindeutig zu beschreiben. Der Algorithmus nimmt jeden Schlüsselpunkt, berechnet in einem bestimmten viereckigen Bereich um den Punkt Gradienten. Ein Gradient bestimmt die Richtung und Steigung der größten Änderung<sup>26</sup>. Die Gradienten werden in einem Histogramm gespeichert. Aus dem Histogramm wird ersichtlich, in welche Richtung die meisten Gradienten zeigen und es wird daraus eine Referenzrichtung dieses Punktes abgeleitet. Der gleiche Prozess wird im nächsten Schritt nochmals durchgeführt. Jetzt ist die Auswahl kreisförmig und das genutzte Koordinatensystem orientiert sich an der zuvor abgeleiteten Referenzrichtung. Es wird wieder eine Hauptrichtung der Gradienten bestimmt und die Gradienten werden in weitere Histogramme abgespeichert. Die Informationen sind dazu da, die Schlüsselpunkte rotationsinvariant zu machen [Herling und Broll(2011), 20] [Weitz(2022), 43].

Der Deskriptor hat am Ende des Algorithmus folgende Informationen für jeden Schlüsselpunkt:

- Die verfeinerten x- und y-Koordinaten,
- der Skalierungsfaktor,

<sup>24</sup>Quelle Bild: <http://weitz.de/sift/>, zuletzt aufgerufen am 01.08.2022

<sup>25</sup><https://de.wikipedia.org/wiki/Taylor-Formel>, zuletzt aufgerufen am 01.08.2022

<sup>26</sup><https://de.wikipedia.org/wiki/Gradient>, zuletzt aufgerufen am 01.08.2022

- der Grad an Unschärfe,
- die Hauptrichtungen der Gradienten,
- normalisierte  $4 \times 4 \times 8 = 128$  8-bit integer Werte, die die Histogramme aus dem letzten Schritt darstellen.

Diese Informationen beschreiben jeden Schlüsselpunkt eindeutig. Der Punkt ist durch die Skalierungen skaleninvariant, wenig anfällig für Rauschen oder feine Strukturen und rotierungsinvariant.

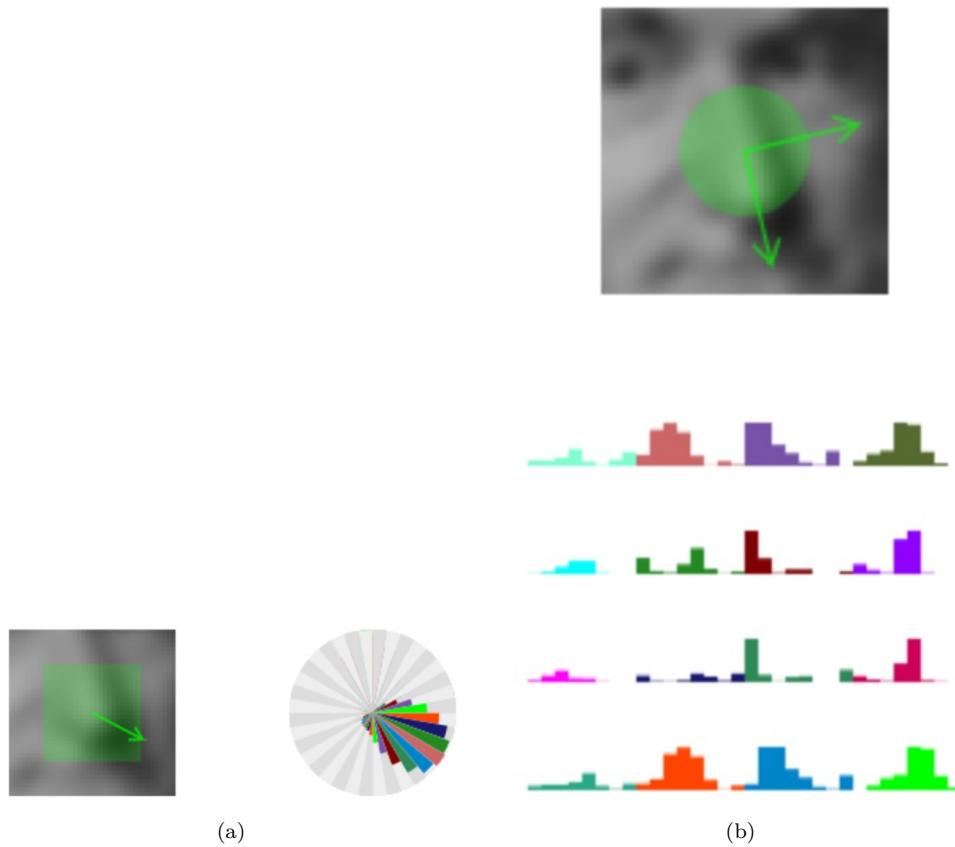


Abbildung 10: Das Histogramm und die Referenzrichtung im ersten Durchlauf (a) und das Koordinatensystem nach der Referenzrichtung und die entstandenen Histogramme(b)<sup>27</sup>.

### 2.3.3.2 Feature-Matching

Sind die Deskriptoren vorhanden, kann im Anschluss ein *Feature-Matching* durchgeführt werden. Dabei werden Schlüsselpunkte mit ihren Deskriptoren verglichen und zugeordnet. Mithilfe der zuge-

<sup>27</sup>Quelle Bild: <http://weitz.de/sift/>, zuletzt aufgerufen am 01.08.2022

orndeten Schlüsselpunkten kann die Pose der Kamera berechnet werden [Dörner(2019) 13][Herling und Broll(2011) 20].

Als Beispiel wird mit Python<sup>28</sup> und OpenCV<sup>29</sup> ein Feature-Matching durchgeführt. Hierfür wird ORB verwendet. Abbildung 11 zeigt das Feature-Matching Ergebnis mit der Brute Force Methode. Bei der Brute Force Methode werden die Deskriptoren verglichen. Der Deskriptor der, der am nächsten zum vergleichenden Deskriptor ist, wird zurückgegeben und als Match identifiziert<sup>30</sup>. In diesem Beispiel wird die Skaleninvarianz und die Rotierungsinvianranz ersichtlich.

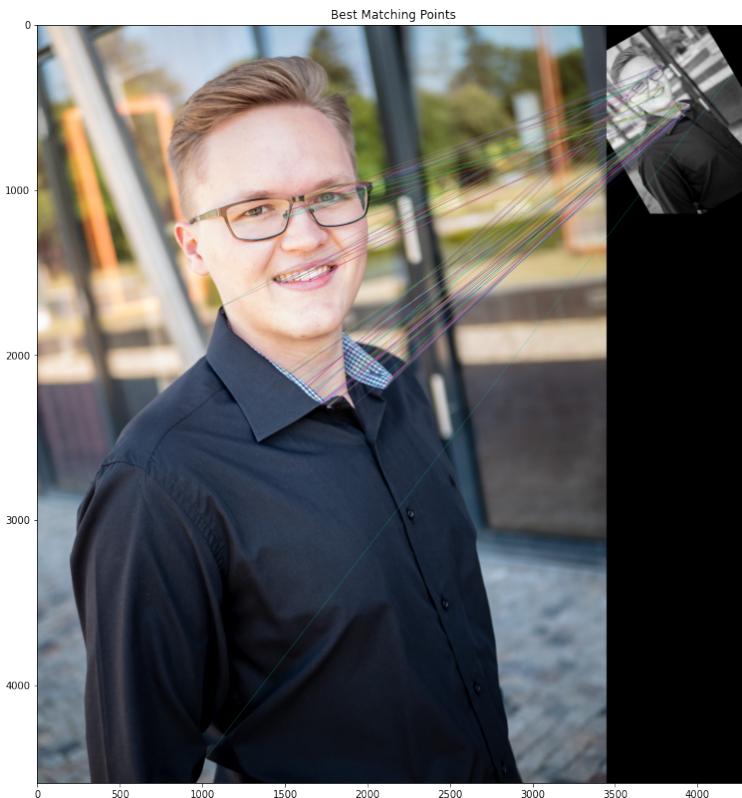


Abbildung 11: Das Ergebnis des Feature-Matching mit ORB und OpenCV.<sup>31</sup>

### 2.3.3.3 Simultaneous Localization and Mapping (SLAM)

Während bei Feature-Matching Methoden eine Feature-Map für das Tracking vorhanden sein muss, verfolgt SLAM[12][6] einen weiterführenden Ansatz. So wird simultan zum Tracking eine Feature Map

<sup>28</sup><https://www.python.org/>

<sup>29</sup><https://opencv.org/>

<sup>30</sup>[https://docs.opencv.org/3.4/dc/dc3/tutorial\\_py\\_matcher.html](https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html), zuletzt aufgerufen am 02.08.2022

<sup>31</sup>Link zum Source-Code:

<https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf>, und das Beispiel: [https://github.com/Koliyoshii/masterarbeit/blob/main/opencv\\_orb\\_feature\\_matching.ipynb](https://github.com/Koliyoshii/masterarbeit/blob/main/opencv_orb_feature_matching.ipynb)

erstellt und kontinuierlich erweitert und verbessert. Während dem Tracking werden dreidimensionale Schlüsselpunkte ermittelt. Dabei wird entweder kamerabasiert (Visual SLAM) nach Merkmalen gesucht oder es kommen Sensoren zur Generierung von Tiefeninformationen, wie zum Beispiel *LIDAR-Sensoren* (engl. *light detection and ranging*), zum Einsatz[Liu et al. 27]. Vorteil dieser Technik ist, dass bereits ab dem ersten Start eine grobe Positions- und Orientierungsbestimmung durchgeführt werden kann. Zusätzlich verbessert sich diese im Laufe des Trackings, indem immer weiter eine Karte mit 3D Schlüsselpunkten der Umgebung erstellt wird. Ursprünglich stammt SLAM aus der Robotertechnik, damit diese sich frei in einer Umgebung bewegen können. Da im AR-Kontext Smartphones weniger und unpräzisere Sensoren haben (z.B. keine Tiefensensoren) wird auf Visual SLAM gesetzt[Dörner(2019) Seite 143 f., 13][Herling und Broll(2011) 20].

Weiterhin schlagen Klein und Murray[23] das *Parallel Tracking and Mapping (PTAM)* für AR Anwendungen vor. Dabei wird die Idee von SLAM mit einer einzelnen Kamera durchgeführt. Hierfür wird das Tracking über FAST und die Kartenerstellung aufgeteilt. Für jeden neuen Frame wird die Karte erweitert und 3D Schlüsselpunkte verbessert. Zusätzlich wird eine Hauptebene festgelegt, auf der die AR Objekte platziert werden. Diese Herangehensweise ist performant genug, um auf mobilen Geräten anwendung zu finden[Klein und Murray(2009) 24]. Auch ARCore richtet sich nach diesem Prinzip, indem Ebenen wie Tische oder Wände erkannt werden.

#### 2.3.4 Tracking in AR Core

## 2.4 Grafik-Rendering-Pipeline

Die Grafik-Rendering-Pipeline (GRP) dient dazu die dreidimensionalen Objekte mit einer virtuellen Kamera auf ein zweidimensionales Bild zu *rendern*<sup>32</sup>. Es handelt sich um eine Pipeline (Befehlskette), die aus mehreren Teilaufgaben bestehen. Die Geometrie, die Charakteristiken der Umgebung und die Platzierung der virtuellen Kamera in der Szene beeinflussen die Lokalisierung und die Form der 3D-Objekte. Die Materials, *Texturen*, Lichtquellen und Shader beeinflussen die Erscheinung [1, Moeller (2019)]. Materials affektieren das Aussehen eines 3D Objekts mit Texturen, Farben oder der Eigenschaften von Reflexionen. Texturen sind Bilder, die die Oberfläche des Objekts benetzen. Abbildung 12 zeigt mehrere dreidimensionale Objekte, die virtuelle Kamera und das daraus generierte zweidimensionale Bild.

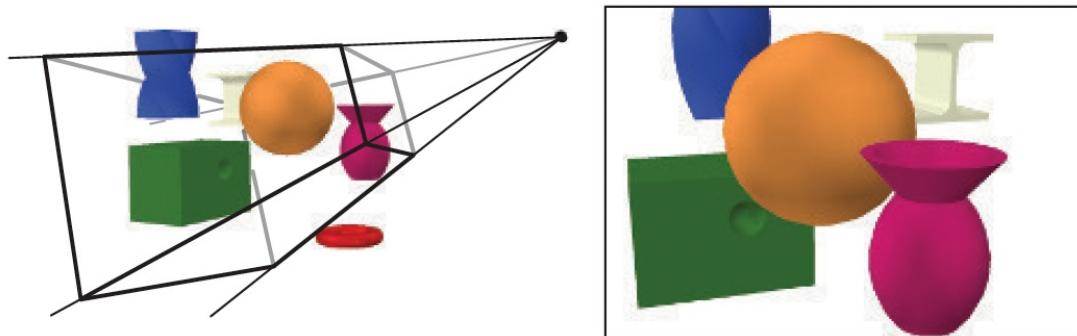


Abbildung 12: Die dreidimensionale Szene (links) und die zweidimensionale Sicht der virtuellen Kamera. Wird das Objekt perspektivisch gerendert, gibt es ein Raumvolumen (frustum). Es werden nur die Objekte gerendert, die sich innerhalb dieses Volumens befinden. So wird der rote Donut nicht und das blaue Objekt abgeschnitten dargestellt[1, Moeller (2019)].

Die GRP wird in vier Hauptstufen unterteilt:

- Applikation,
- Geometrieverarbeitung,
- Rasterisierung,
- Pixelverarbeitung.

<sup>32</sup>„ein Bild (eine Grafik) aus Rohdaten ([..] aus einer 3D-Szene[..]) durch einen Webbrower/ein Programm erzeugen.“  
<https://de.wiktionary.org/wiki/rendern>

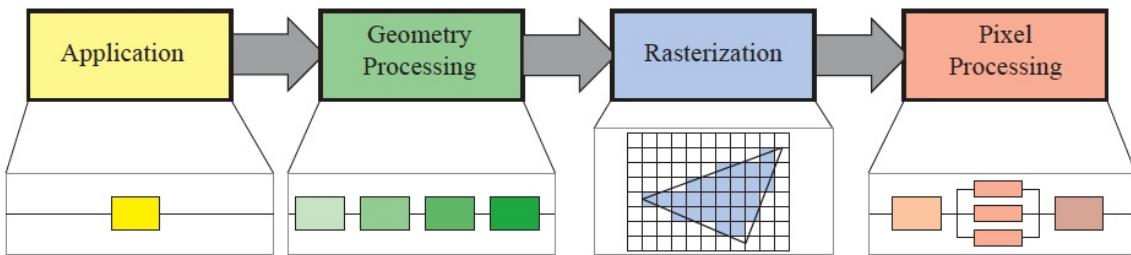


Abbildung 13: Die Basis der GRP, die in weitere Sub-Pipelines eingeteilt und parallel ausgeführt werden können [1].

Die Abarbeitung der Aufgaben in den verschiedenen Stufen wird parallel ausgeführt. Durch die Parallelisierung ist eine verbesserte Performance möglich, da nicht jeder Prozess Schritt für Schritt durchgeführt werden muss. Jede Stufe kann gleichzeitig eine weitere Sub-Pipeline benutzen. So hat die Stufe *Geometrieverarbeitung* in der Abbildung 13 die Aufgabe in eine Sub-Pipeline gegliedert, die wiederum parallel ausgeführt werden kann. Die render Geschwindigkeit wird in *frames pro sekunde* (FPS) angegeben und repräsentiert die Anzahl an Bildern, die pro Sekunde gerendert werden[1, Moeller (2019)].

#### 2.4.1 Applikationsstufe

Die Hauptaufgabe der Applikationsstufe besteht darin die benötigten Daten in die Pipeline zu schicken und zu aktualisieren. Die Daten bestehen aus den Render-Primitiven, die beispielsweise Punkte, Linien, Dreiecke oder Polygone (Vielecke) darstellen. Am Ende der Applikationsstufe werden die Rendering-Primitive an die Geometrieverarbeitungsstufe weitergegeben. Weiterhin finden Interaktionen wie z.B. Kollisionsabfragen statt oder es werden Input-Daten von Maus und Tastatur verarbeitet. Auch werden Culling-Algorithmen<sup>33</sup> ausgeführt. Die Applikationsstufe wird im Gegensatz zu den anderen Stufen meist auf der CPU ausgeführt. Entwickler haben dadurch eine hohe Kontrolle über die Prozesse in dieser Stufe. Sie können die Implementierung selbst bestimmen und im nachhinein modifizieren, um z.B. die Performance weiter zu verbessern[1, Moeller (2019)].

#### 2.4.2 Geometrieverarbeitung

Auf dieser Stufe finden Operationen an den Polygonen und Vertices (Eckpunkte) statt. Sie wird auf der GPU ausgeführt und lässt sich in eine Sub-Pipeline in Vertex-Shading, Projektion, Clipping und Window-Viewport-Transformation unterteilen.

<sup>33</sup>ein Algorithmus, bei dem bestimmte (z.B. verdeckte) Polygone nicht gerendert werden, um eine bessere Performance zu erzielen [10].

### 2.4.2.1 Vertex-Shading

Ein Vertex-Shader ist ein programmierbarer Shader in der Rendering-Pipeline, der die Eigenschaften jedes einzelnen Vertex bestimmt. Ein Vertex beinhaltet die Koordinaten von Punkten im dreidimensionalen Raum. Mithilfe von Informationen, welche Vertices miteinander verbunden sind, werden Linien und Polygone definiert [31, Vgl. Nischwitz (2012) S.48.]. Die Aufgabe des Vertex-Shaders ist es, die Modellkoordinaten der Vertices durch Translation, Rotation oder Skalierung zu transformieren, um die Modelle zunächst im Weltkoordinatensystem, dann im Kamerakoordinatensystem und zuletzt im Clipping-Koordinatensystem zu lokalisieren. Die Transformationen werden mit 4x4 Transformationsmatrizen durchgeführt. Der Vertex-Shader berechnet die Position eines Vertex von seinem Modellkoordinatensystem zum Clipping-Koordinatensystem:

$$p' = P * K * W * p \quad (1)$$

wobei:

$p'$  = Vertex in Clip-Koordinaten

$P$  = Projektionsmatrix

$K$  = Kamera-Transformation

$W$  = Welt-Transformation

$p$  = Vertex in Modellkoordinaten

Die Reihenfolge der Transformationen spielt eine wichtige Rolle, da die Matrizenmultiplikation nicht kommutativ ist. Die Transformationen in der Gleichung 1 müssen daher von rechts nach links durchgeführt werden.

Die Vertices der Modelle befinden sich ursprünglich in ihrem eigenen Modellkoordinatensystem. Die Szene hat ein Weltkoordinatensystem und ein Kamerakoordinatensystem. Die Vertices werden zunächst in das Weltkoordinatensystem transformiert. Die Kamera in der Szene hat Koordinaten im Weltkoordinatensystem und eine Richtung, in die die Kamera zeigt. Mit der Kamera-Transformation werden die Vertices und die Kamera so platziert, dass die Kamera in der Position des Ursprungs des Weltkoordinatensystems liegt und (meist) in negativer z-Richtung zeigt, während die y-Achse nach oben und die x-Achse nach rechts zeigt. Die genaue Definition der Richtungen der Koordinatenachsen hängt von der jeweiligen Anwendung ab. Das daraus folgende Koordinatensystem wird Kamerakoordinatensystem genannt. Somit befinden sich die Vertices nach den Kamera-Transformationen im Kamerakoordinatensystem. Schließlich wird eine Projektionsmatrix angewandt, um die Kamerakoordinaten der Vertices in Clipping-Koordinaten zu transformieren.

Zusätzlich zu den Transformationen bestimmt ein Vertex-Shader Eigenschaften wie die Farbe, Texturkoordinaten oder Normalenvektoren für jeden Vertex. Diese beschreiben das Material des Objekts und Effekte von Lichtquellen, die auf die Oberfläche scheinen. Die Wirkung von Licht auf

ein Material wird Shading genannt. Es werden "[..]Normalenvektoren und Materialeigenschaften der Objekte auf der einen Seite und den Eigenschaften der Lichtquellen auf der anderen Seite für jeden Vertex die Beleuchtungsrechnung durchgeführt und somit eine Farbe ermittelt"[Nischwitz (2012) S.48, 31]. Die Ergebnisse der Berechnungen werden zur Rasterisierung und zur Pixelverarbeitung weitergeschickt[1, Moeller (2019)].

Mit Vertex-Shadern ist es möglich Oberflächen realistisch darzustellen, ohne die Polygonanzahl zu erhöhen. So nutzen Mitchell[30] und Isidoro [21] Vertex-Shader, um mit Heightmaps und Normalmaps einen realistischen Ozean zu rendern.

#### 2.4.2.2 Projektion

Bei der Projektion werden zwei Aufgaben durchgeführt. Zum einen wird ein Clipspace (Kanonisches-Sichtvolumen) definiert, das meist in Form eines Einheitswürfels<sup>34</sup> gegeben ist. Der Würfel ist für das Clipping von Bedeutung.

Zum anderen wird eine Projektionsmethodik je nach Anwendungsfall bestimmt. Beispielsweise gibt es eine orthographische Projektion, bei der parallele Linien auch parallel erscheinen, während bei der perspektivischen Projektion parallele Linien Fluchtpunkten folgen. Die Projektionsmethodik bestimmt die Projektionsmatrix für die Transformation in die Clipping-Koordinaten.

#### 2.4.2.3 Clipping

Nur mit den im Vertex-Shader transformierten Clipping-Koordinaten der Vertices kann Clipping durchgeführt werden. Es werden nur die Primitiven zur Window-Viewport-Transformation weitergeschickt, die sich innerhalb des kanonischen-Sichtvolumens befinden. Clipping wird dann benötigt, wenn sich ein Teil eines Primitiven innerhalb und ein anderer Teil außerhalb des Sichtvolumens befinden. Die Primitiven, die sich mit dem Einheitswürfel schneiden, erhalten an der Schnittkante neue Vertices[Moeller (2019) 1].

#### 2.4.2.4 Window-Viewport-Transformation

Im nächsten Schritt wird mit den Primitiven eine Window-Viewport-Transformation durchgeführt. Dadurch wird es möglich die x- und y-Koordinaten der Primitiven auf ein Bildschirmausschnitt (Viewport) darzustellen. Die Primitiven erhalten Bildschirm- bzw. Window-Koordinaten<sup>35</sup>. Es handelt sich dabei um eine Translation gefolgt von einer Skalierung[Moeller (2019) S.20f., 1] [Nischwitz (2012) S.150, 31].

---

<sup>34</sup>ein Würfel mit der Skalierung (1,1,1)

<sup>35</sup>Bildschirm-Koordinaten sind zweidimensionale (x,y)-Koordinaten, während Window-Koordinaten auch die z-Koordinate beinhaltet (x,y,z).

### 2.4.3 Rasterisierung

Bei der Rasterisierung werden mit einer perspektivischen Projektion Pixel gesucht, die sich auf dem Viewport innerhalb der gerenderten Primitiven befinden. Es findet eine Übersetzung der 3D-Koordinaten in 2D-Pixelkoordinaten statt. Klassische Rasterisierungs-Algorithmen definieren einen zugehörigen Pixel genau dann, wenn ein Pixel sich innerhalb eines Primitiven befindet. Es werden sogenannte *Fragmente* für den Teil des Pixels erzeugt, der sich mit dem Dreieck überschneidet[37, zuletzt aufgerufen am 11.07.2022] cite\*[Moeller (2019) S.21f.,,] moeller2019.

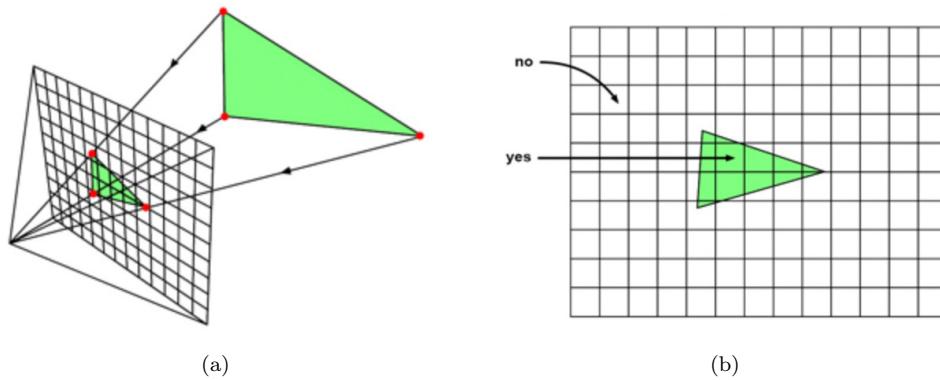


Abbildung 14: Die Primitiven werden in eine 2D-Ebene projiziert(a). Befindet sich ein Pixel innerhalb des Primitiven, werden diskrete Fragmente gebildet, die im in der Pixelverarbeitung weitergeschickt werden(b) [37, zuletzt aufgerufen am 11.07.2022].

### 2.4.4 Pixelverarbeitung (Fragment-Shader)

Bei der Pixelverarbeitung werden die diskreten Fragmente aus der Rasterisierung mit einem Fragment-Shader verarbeitet. So werden im einfachsten Fall die endgültigen Farben der Fragmente festgelegt oder auch Texturen an das Modell angeheftet.

Nach dem Fragment-Shader wird überprüft, ob sich im dreidimensionalen Raum zwei oder mehrere Fragmente überlappen. Für diesen Fall gibt es einen z-Buffer, der für jedes Fragment einen z-Wert gespeichert hat. Dieser z-Wert repräsentiert wie weit das Fragment von der Kamera entfernt ist. Ein z-Buffer Algorithmus entscheidet dann, welches Fragment näher zur Kamera liegt und welche Farbe das Fragment letztendlich haben soll. Dieser Algorithmus ist simpel und hat den Vorteil, dass die render Reihenfolge der Primitiven trivial ist. Probleme gibt es jedoch bei transparenten Objekten. Hier ist die Reihenfolge für eine korrekte Darstellung wichtig[Moeller (2019) S.21f., 1]. Für eine ausführliche Erklärung für den Umgang mit transparenten Objekten wird auf Möller, Kapitel 5.5 verwiesen[Moeller (2019) S.148ff., 1].

## 3 Konzeption der Anwendung

In diesem Abschnitt wird das Konzept vorgestellt. Es werden die Interaktionsmöglichkeiten des Nutzers erläutert und ein grober Ablauf der Anwendung aufgezeigt. Mithilfe eines Use-Case-Diagramms wird der Ablauf grafisch dargestellt. Ein Use-Case Diagramm ist ein Verhaltensdiagramm der *Unified Modeling Language (UML)*. Es repräsentiert grafisch Anwendungsfälle für eine Software und beinhaltet eine Beschreibung der Funktionen, der Akteure und deren Beziehungen in einem System<sup>36</sup>. Das Kapitel ?? geht dann auf die technische Umsetzung ein.

### 3.1 Idee und Ablauf

Das Ziel der Anwendung ist es dem Nutzer die historischen Gebäude sowohl vor Ort als auch an einem beliebigen Ort zu zeigen. Die Gebäude werden so dargestellt, dass sie sich nahtlos in das reale Bild einfügen. Dafür wird auf die richtige Belichtung und auf korrekte Schattenwürfe geachtet.

Je nach Tageszeit verändert sich die Farbtemperatur des Lichts. Zum Sonnenaufgang und zum Sonnenuntergang färbt sich z.B. das Hauptlicht röthlich, während im Tageslicht die Farbtemperatur neutral ist. Ist es Nacht, scheint kein Licht auf die Gebäude. Zusätzlich wird das Gebäude den herrschenden Wetterbedingungen angepasst.

Eine Anbindung zur Wetter Representational State Transfer (REST) API *Open-Weather-Map*<sup>37</sup> fragt das aktuelle Wetter am Ort des Nutzers ab.

Eine REST API ermöglicht den Austausch von Informationen von unterschiedlichen Systemen. Die Informationen, die in diesem Fall die Wetterdaten sind, liegen auf Servern, die mit Hilfe eines HTTP-Requests angefordert werden<sup>38</sup>. Mit der Antwort der API werden dann Parameter für das Licht wie in Kapitel 5.4 und des verwendeten Fragment-Shaders verändert. Die genaue technische Umsetzung wird in Kapitel 5.5 behandelt.

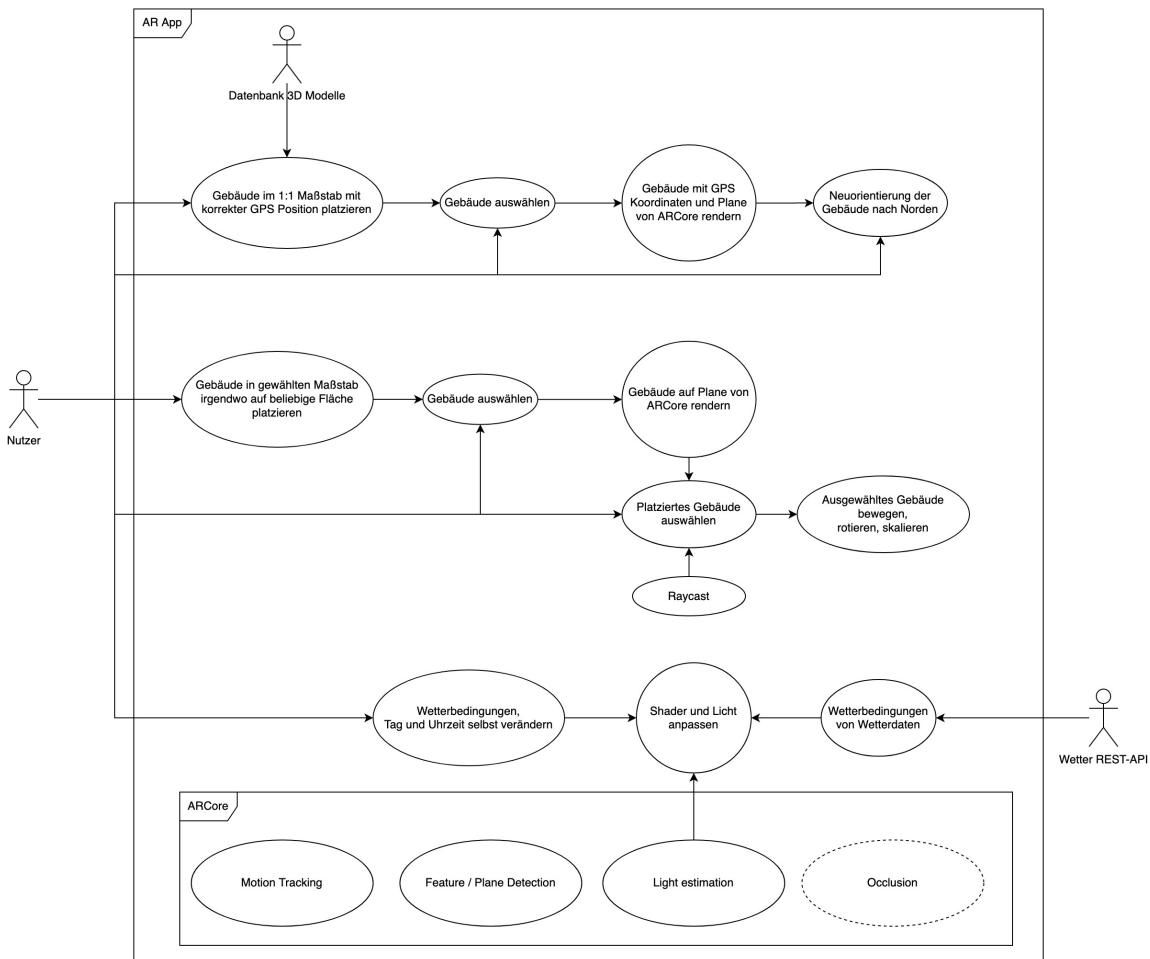
Das Use-Case Diagramm in Abbildung 15 zeigt den Ablauf der Anwendung.

---

<sup>36</sup><https://de.wikipedia.org/wiki/Anwendungsfalldiagramm>, zuletzt aufgerufen am 10.08.2022

<sup>37</sup><https://openweathermap.org/>, zuletzt aufgerufen am 10.08.2022

<sup>38</sup><https://www.codecademy.com/article/what-is-rest>, zuletzt aufgerufen am 19.08.2022

Abbildung 15: Das Use-Case-Diagramm zur Anwendung.<sup>39</sup>

In der Anwendung gibt es drei Akteure: Der Nutzer, die REST-API für die Wetter-Abfrage und eine interne Datenbank mit den Daten zu den Gebäuden. Die interne Datenbank enthält die Informationen zu den Gebäuden, die für die Anwendung benötigt werden. Die Informationen sind:

- die 3D Modelle als *Prefab*. Ein Prefab ist ein GameObject in Unity, dass alle Komponenten, Werte und angehängte GameObjects beinhaltet. Sie erlauben dieses GameObject zu speichern und mit den gleichen Informationen wieder zu verwenden<sup>40</sup>,
- der Name des Gebäudes, wie z.B. das SABA Hauptgebäude oder das Mannschaftsgebäude,
- eine Kurzbeschreibung. Diese wird noch nicht genutzt. Die Idee dazu ist, zusätzlich zum 3D Modell noch eine Kurzbeschreibung des Gebäudes und dessen ehemalige Funktion in AR einzublenden,

<sup>39</sup>Quelle Bild: <https://drive.google.com/file/d/1ZhIIEN6V9Jt-uwVcFVNkeFbu1qxbGkaN/view?usp=sharing>

<sup>40</sup><https://docs.unity3d.com/Manual/Prefabs.html>, zuletzt aufgerufen am 13.08.2022

- die Längen- und Breitengrade für die GPS-Platzierung,
- eine Winkelangabe. Diese wird benötigt, um bei einer Ausrichtung nach Norden die Gebäude mit der korrekten Rotation darzustellen,
- eine ID-Nummer zur Identifizierung.

Die Datenbank wird in Unity mit `ScriptableObject`<sup>41</sup> realisiert. Das ist ein Datencontainer, indem individuelle Daten gespeichert werden können. Für jedes Objekt wird ein `BuildingGPS` Scriptable Object erzeugt, indem die jeweiligen Daten gespeichert sind. Die Nutzung der Daten wird in Kapitel 5.3.2.1 und 5.3.3 behandelt.

Zu Beginn der Anwendung kann der Nutzer in einem Menü sich zwischen zwei Möglichkeiten der Platzierung entscheiden:

- das Gebäude wird über GPS Koordinaten platziert,
- das Gebäude wird auf einer von ARCore getrackten Flächen platziert.

Anschließend wird die jeweilige AR Szene gestartet. An diesem Zeitpunkt wird ARCore gestartet. Die Funktionen wie z.B. das Motion Tracking oder die Plane Detection aktivieren sich. Der Nutzer wählt daraufhin das Gebäude aus, das platziert werden soll. Eine Vorschau des Modells und der Name des Gebäudes werden angezeigt. Dieses Menü ist in Abbildung 16 zu sehen.

---

<sup>41</sup><https://docs.unity3d.com/Manual/class-ScriptableObject.html>, zuletzt aufgerufen am 13.08.2022

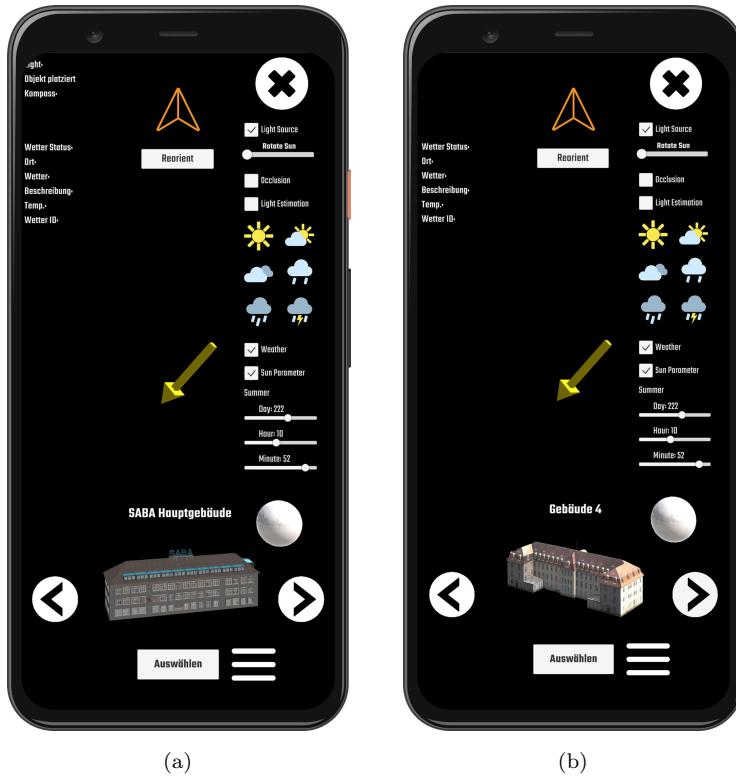


Abbildung 16: Das Auswahl-Menü. Der Nutzer kann mit dem Burger-Icon das Menü jederzeit öffnen und wieder schließen. Mit den Pfeilen kann zwischen den vorhandenen Gebäuden geschaltet werden.

Ist ein Gebäude ausgewählt, ist die Platzierung des Gebäudes möglich. Voraussetzung ist, dass ARCore eine Fläche gefunden hat. Sobald eine Fläche vorhanden ist, wird über Raycast ein Hit-Testing durchgeführt (siehe Kapitel 5.2.5). Ist ein Hit registriert, ist eine Fläche für die Platzierung vorhanden. Der Nutzer erhält Feedback über ein Pfeil-Icon, das auf die jeweilige Fläche platziert wird. Dieser hilft dem Nutzer bei der Orientierung und der Platzierung der Objekte.

### 3.2 Platzierung auf einer beliebigen Fläche

Der Pfeil auf dem Boden zeigt an, wo die Position des Gebäudes sein wird. Nach der Auswahl des Modells kann der Nutzer über ein Tap auf dem Bildschirm das Gebäude platzieren. Über einen Raycast wird das platzierte Modell erkannt und der Nutzer kann mit einem Button das platzierte Gebäude auswählen. Dieses kann per Drag Gesten über erkannte Flächen verschoben werden. Mit einer Pinch Geste wird das Gebäude skaliert. Über einen Slider wird die Rotation verändert. Nur ausgewählte Gebäude können bewegt, skaliert, rotiert und gelöscht werden. Es ist nur möglich eine Instanz von einem Gebäude zu platzieren.

### 3.3 Platzierung über GPS

Bei der GPS Platzierung wird die aktuelle GPS Position abgefragt. Mit den Längen- und Breitengraden aus der Datenbank werden die Gebäude bezüglich der GPS Position des Smartphones platziert. Damit das Gebäude nicht in der Luft schwebt, wird es auf eine Fläche platziert, die ARCore detektiert hat. Der Nutzer hat hier nicht die Möglichkeit ein Gebäude auszuwählen, da diese Funktion nur vor Ort funktioniert. Der Sinn dieser Funktion ist es, dem Nutzer die Gebäude so zu zeigen, wie sie vor Ort ausgesehen haben. Die Gebäude werden an der jeweiligen Position im Größenverhältnis 1:1 dargestellt.

Der Nutzer hat zusätzlich die Möglichkeit mithilfe des Kompass des Gerätes die Szene nach Norden auszurichten. Damit wird sichergestellt, dass das Gebäude in der korrekten Rotation dargestellt wird. In der Datenbank ist das 3D Modell zusammen mit einer Winkelangabe gespeichert. Diese wird benötigt, um das Gebäude so zu drehen, dass es in die korrekte Richtung zeigt. Die technische Umsetzung wird in Kapitel 5.3.3 behandelt.

### 3.4 Licht und Wetterbedingungen ändern

Die Wetterbedingungen werden beim Start der Anwendung von der REST-API abgefragt. Hierfür wird eine Internetverbindung benötigt. Die Informationen zu Wetter und Tageszeit werden dazu, genutzt das Licht und die Shader der Gebäude anzupassen. Zusätzlich wird die Light estimation Funktion von ARCore genutzt, um die Farbtemperatur und die Helligkeit anzupassen.

## 4 Vorbereitung der 3D Modelle in Blender

Einige 3D Modelle haben durch die umliegende Vegetation unvollständige oder mit der Vegetation verschmolzene Meshes. In der Projektarbeit des Wintersemesters 2020/21 wird das Problem näher beschrieben [25].

### 4.1 Lückenhafte Wände füllen

Das Gebäude 2 des Lyautey Geländes hat eine Haushälfte, die mit einem Baum verschmolzen ist und es fehlt daher ein Teil der Wand. Da das Gebäude symmetrisch und die gespiegelte Seite fehlerfrei ist, wird ein Teil dieser Wand als Lückenfüller eingesetzt. Im folgenden wird die Nachbearbeitung an diesem Modell beschrieben.



Abbildung 17: Ein Baum behinderte bei den Aufnahmen die Sicht auf die Wand, sodass diese mit dem Baum verschmolzen ist.

Zunächst wird das Modell kopiert. Anschließend werden die fehlerhaften Polygone im *Edit-Mode* ausgewählt und gelöscht. Beim kopierten Modell werden alle Polygone gelöscht, bis auf die Wand, die eingesetzt werden soll. Diese wird mit dem *Mirror-Modifier* gespiegelt. Das gespiegelte Modell wird an die Position der fehlerhaften Wand gesetzt und die beiden Modelle werden mit *STRG + J* als ein Modell zusammengefügt. Die Lücke zwischen den beiden Modellen wird anschließend händisch mit neuer Geometrie gefüllt. Dieser Prozess ist sehr zeitaufwendig, da je nach Polygonanzahl viele Kanten einzeln ausgewählt werden müssen. Daher wird zunächst ein *Decimate-Modifier* benutzt, um die Anzahl an Polygone zu reduzieren. Bei diesem Gebäude wird eine Reduzierung um 70% vorgenommen, sodass sich die gesamte Anzahl an Polygonen von über 2.5 Millionen auf ca 750.000 Polygone verringert, ohne einen starken Verlust im Detailgrad festzustellen.

Um die Lücken zu schließen werden benachbarte Kanten im *Edit-Mode* ausgewählt und mit der *F-Taste* mit neuen Flächen aufgefüllt. An einigen Stellen sind kleine Polygone vorhanden, bei dem

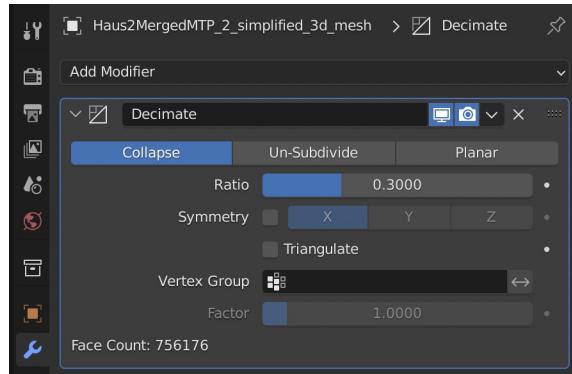


Abbildung 18: Der Decimate-Modifier reduziert die Anzahl an Polygonen. Ein Wert von 0.3 bedeutet, dass die Polygonenanzahl 30% der ursprünglichen Polygonanzahl entspricht.

dieser Prozess erschwert wird. Diese werden gelöscht und mit größeren Flächen ersetzt.

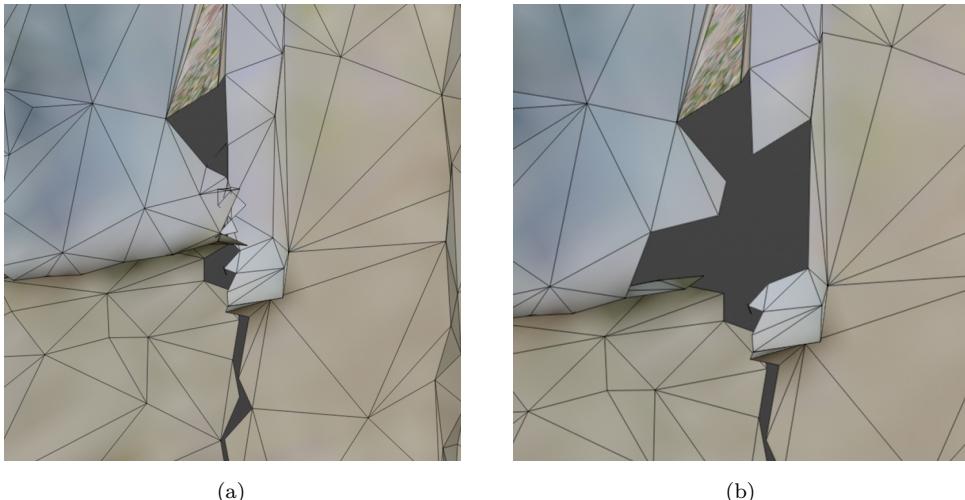


Abbildung 19: Kleine Polygone erschweren das Markieren von Kanten (a). Das entstandene Loch wird mit größeren Flächen gefüllt (b).

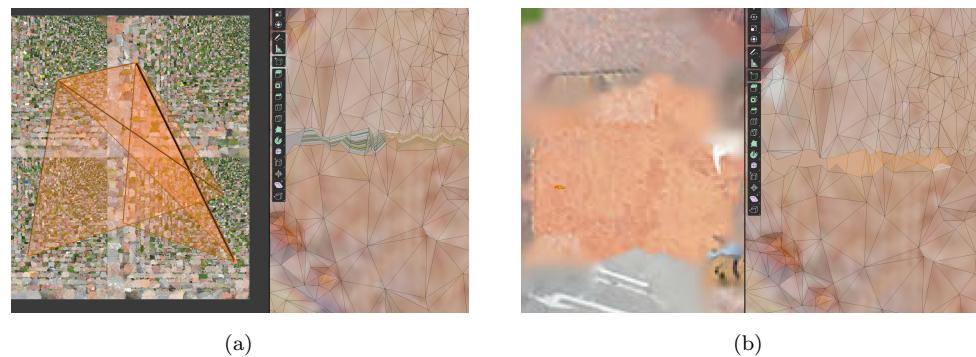
Ist der Füllprozess abgeschlossen, ist die Naht dennoch erkennbar (siehe Abbildung 20), da die Textur auf den neuen Polygone verzerrt wird. Aufgrunddessen werden im letzten Schritt im UV-Editing Tab die UV-Koordinaten der neuen Polygone neu vergeben, sodass die Texturen den umliegenden Flächen entsprechen. Hierfür werden die neuen Polygone im *Edit-Mode* markiert. Im *UV-Editor* werden mit dem Befehl *U + Unwrap* die markierten Polygone bestmöglich auf die 2D Textur gelegt.<sup>42</sup> Die Polygone werden auf der Textur so skaliert, bewegt und rotiert, dass sie eine ähnliche Textur wie die umliegenden Flächen besitzen. Zur Hilfe werden die umliegenden

<sup>42</sup><https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html>, zuletzt augerufen am 29.05.2022



Abbildung 20: Das zusammengesetzte Gebäude.

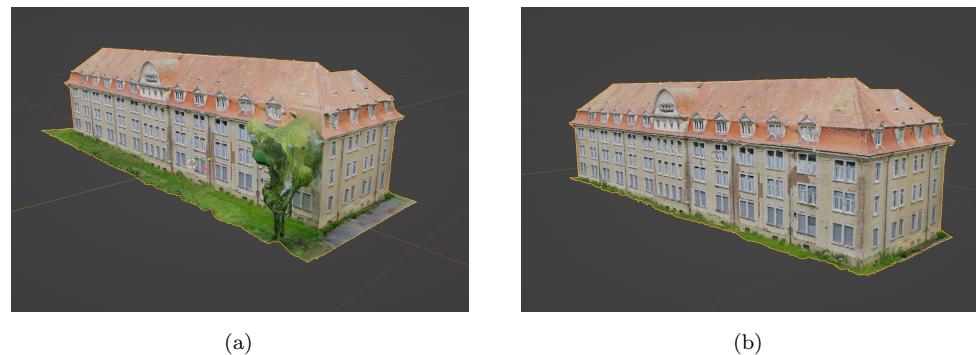
Flächen markiert und deren UV-Koordinaten auf der Textur verglichen. So wird sichergestellt, dass die Farbunterschiede an den Kanten nicht zu stark abweichen. Das fertige Modell ist im Vorher-Nachher-Vergleich in Abbildung 22 zu sehen.



(a)

(b)

Abbildung 21: Die Polygone werden markiert(a), unwrapped und anschließend auf eine geeignete Fläche auf der Textur gelegt(b).



(a)

(b)

Abbildung 22: Das Mannschaftsgebäude vor(a) und nach der Bearbeitung(b).

## 4.2 Export und Import

Damit Unity die 3D Modelle korrekt darstellen kann, werden Anpassungen in Blender vorgenommen:

- Vor dem Export werden Texturen in einem Ordner gespeichert. In Blender muss dafür die Option unter *File>External Data>Automatically Pack Resources* ausgeschaltet sein. Anschließend wird mit *File>External Data>Unpack Resources* ein Ordner im Projekt mit den Namen *textures* erstellt, in dem sich die Texturen des Projekts befinden. Nun kann Unity die fehlenden Materials und Texturen im Projekt-Ordner finden,
- Unity kann die Texturen nur anwenden, wenn die Textur direkt im *Shader Node* als *Base Color* verbunden ist. Dazwischen befindliche Knoten funktionieren nur in Blender und müssen entfernt werden,
- die in Blender genutzten *Modifier*<sup>43</sup> werden angewendet,
- damit die Textur korrekt dargestellt werden, werden die Richtungen der *Normalen*<sup>44</sup> überprüft, ob diese in die richtige Richtung zeigen. Die Normalen müssen zur korrekten Darstellung nach außen zeigen. In Blender können die Normalen im *Edit Mode* im Viewport Overlay angezeigt und unter *Mesh>Normals* angepasst werden,
- Blender und Unity haben unterschiedliche Koordinatensysteme. In Blender zeigt die Z-Achse nach oben und die Y-Achse in die Tiefe, während in Unity die Y-Achse nach oben und die Z-Achse in die Tiefe zeigt. Damit die Modelle aus Blender mit der korrekten Orientierung dargestellt werden, wird das Modell in Blender um -90° gedreht. Mit *STRG + A* und *Rotation* wird die Rotation eingebettet,
- beim Export wird das Export-Format *.fbx* benötigt.

Jetzt wird das 3D Modell als *.fbx* Datei in Unity importiert. Im *Inspector*-Fenster der *.fbx* Datei werden unter Materials mit *Extract Materials* Materials in Unity generiert, die die Texturen beinhalten. Der Texturen-Ordner muss im gleichen Ordner liegen. Werden die Texturen nicht gefunden, so können diese händisch in das Projekt eingefügt werden. Um die Texturen den von Unity generierten Materials zu geben, wird im *Inspector* die Textur in das Feld neben *Albedo* eingefügt werden.

---

<sup>43</sup><https://docs.blender.org/manual/en/latest/modeling/modifiers/index.html>

<sup>44</sup><https://docs.blender.org/manual/en/latest/modeling/meshes/editing/mesh/normal.html?highlight=normals>

## 5 Technische Umsetzung

Dieses Kapitel erläutert, mit welcher Software und Hardware die Anwendung entwickelt wird. Damit wird ein Überblick geschaffen, welche Funktionen die Anwendung verwendet und wie die Implementierung durchgeführt wird. Es werden die Vorteile in der Entwicklung mit Unity und ARCore aufgeführt, um die Wahl der Software zu begründen. Anschließend werden die Funktionen von ARCore vorgestellt.

Es wird der Aufbau der AR Szene und dessen Grundelemente erklärt, um einen Überblick zu erhalten, welche Elemente in der Szene bestimmte Funktionen ausführen. Die Implementierung der verwendeten ARCore Funktionen in dieser Anwendung wird kurz erläutert, um den State of the Art der Entwicklung von AR näher zu bringen. Im Anschluss erfolgt eine Bewertung der Funktionen. Die Kriterien dafür sind die Nutzbarkeit für diese Anwendung und inwiefern die jeweilige Funktion die Nutzererfahrung erweitert.

Danach wird veranschaulicht, wie die Anbindung zu Open Weather Map umgesetzt wird und wie eine Antwort der API aussieht. Die Implementierung der Anpassungen an die verschiedenen Wetterbedingungen wird im Code gezeigt. Um die Veränderungen in der Erscheinung der Gebäude nachzuvollziehen, werden die Ergebnisse miteinander verglichen.

### 5.1 Verwendete Software und Hardware

Diese AR Anwendung wird mit Unity<sup>45</sup> in der Version *2021.2.14* und AR Foundation<sup>46</sup> in der Version *4.2.2* entwickelt. Unity ist eine Spiel-Engine, die eine Laufzeit- und Entwicklungsumgebung für Spiele und 3D Grafik Anwendungen bietet. Unity ist überwiegend kostenlos. Es gibt kostenpflichtige Unity Komponenten, die für diese Arbeit nicht relevant sind. AR Foundation ist ein Framework von Unity für die Entwicklung von Augmented-Reality Anwendungen. Mit AR Foundation werden offiziell Plug-Ins für Googles ARCore, ARKit und weiteren Mixed Reality SDK's für Unity angeboten. Die Nutzung von Unity und AR Foundation bietet diverse Vorteile, die im folgenden Abschnitt erläutert werden. Aufgrund dieser Vorteile wird in dieser Arbeit auf die Entwicklung mit unity und ARFoundation gesetzt.

Das verwendete Smartphone ist ein Oneplus A6003<sup>47</sup> aus dem Jahr 2018. Das Gerät läuft mit der Android Version 11 und der OxygenOS-Version 11.1.2.2.

**Entwicklungsumgebung** Die Benutzeroberfläche von Unity ist verständlich und übersichtlich gestaltet. Der Einstieg wird dadurch vereinfacht. Funktionen und Packages von Unity können

---

<sup>45</sup><https://unity.com/>

<sup>46</sup><https://unity.com/de/unity/features/arfoundation>

<sup>47</sup>[https://de.wikipedia.org/wiki/OnePlus\\_6](https://de.wikipedia.org/wiki/OnePlus_6), zuletzt aufgerufen am 11.08.2022

auch für AR Anwendungen genutzt werden. Beispielsweise wird die Erstellung des User-Interface vereinfacht. Durch die Nutzung von einzelnen Szenen wird die Wahl der Positionierung (GPS oder lokal auf einer Fläche) umgesetzt und es können Buttons, Texte und Slider mit den jeweiligen Interaktionen verwendet werden. Dies beschleunigt den Entwicklungsprozess und der Fokus der Arbeit kann mehr auf die Umsetzung der Licht und Wetterveränderung gesetzt werden.

**Kompatibilität** Unity kann für viele verschiedene Plattformen genutzt werden. Neben den Plattformen für die Spieleentwicklung, ist es möglich sowohl für Android als auch für iOS zu entwickeln. Auch bietet sich eine große Auswahl an Augmented Reality SDK's. Neben ARCore[16] und ARKit[2] ist es z.B. möglich Vuforia[42], Wikitude[44] oder ARToolkit[4] zu verwenden. Hierfür werden Plug-Ins des jeweiligen Anbieters benötigt. Der Vorteil von ARFoundation ist die offizielle Plug-In Unterstützung für ARCore und ARKit. Die Plug-Ins werden regelmäßig geupdatet und die Implementierung ist einfach gestaltet. Diese Vielfältigkeit der Plattformauswahl ist in Hinblick auf die Zielgruppe ein Vorteil. So ist die Anwendung nicht auf ein Betriebssystem wie Android oder iOS beschränkt und es können dadurch mehr Nutzer die App ausprobieren.

**Dokumentation** Die Unity Dokumentation[41] ist ausführlich. Die Funktionen von ARCore und ARKit werden vorgestellt. Die Installation der benötigten Packages und der Szenenaufbau einer AR Anwendung wird erklärt. Neben der Dokumentation für die AR-spezifischen Funktionen gibt es eine umfangreiche Dokumentation für die Nutzung von Unity selbst<sup>48</sup>. Damit wird der Einstieg in die Entwicklungsumgebung vereinfacht und die Implementierung von Funktionalitäten kann anhand der gegebenen Beispielen in der Dokumentation besser nachvollzogen werden.

**Programmiersprache C# und die große Community** Unity verwendet die Programmiersprache C#<sup>49</sup>, die laut PYPL-Index im August 2022 hinter Python, Java und JavaScript als viertbeliebteste Programmiersprache gilt[PYPL, 11]. Damit kann während der Entwicklung auf die Erfahrung von anderen Nutzern zurückgegriffen werden, wenn es Probleme gibt. Auch ist eine ausführliche Dokumentation vorhanden. Da im Studiumsverlauf bereits mit C# gearbeitet wurde, fällt zudem der persönliche Einstieg leicht.

**Asset Store** Unity bietet im Asset Store<sup>50</sup> eine Vielfalt an kostenfreien und kostenpflichtigen Modellen und Skripten, die den Entwicklungsprozess beschleunigen. Anstatt Funktionen mit hohen Aufwand selbst zu schreiben, wird auf den Asset Store zurückgegriffen. Es wird nicht nur Zeit eingespart, sondern auch Fehler in der Entwicklung minimiert. In der Regel werden die veröffent-

---

<sup>48</sup><https://docs.unity3d.com/2023.1/Documentation/Manual/index.html>, zuletzt aufgerufen am 09.08.2022

<sup>49</sup><https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/>, zuletzt aufgerufen am 09.08.2022

<sup>50</sup><https://assetstore.unity.com/>, zuletzt aufgerufen am 09.08.2022

lichten Skripte getestet und regelmäßig mit Bug-fixes verbessert. Damit wird die Fehleranfälligkeit minimiert und der Nutzer hat ein besseres Benutzererfahrung. Im Kapitel 5.4 wird beispielsweise die Positionierung der Sonne vereinfacht implementiert.

## 5.2 ARCore SDK

Ein Software Development Kit (SDK) ist eine Ansammlung von Entwicklungswerkzeugen, die zur Entwicklung von Software dient<sup>51</sup>. Sie beinhalten Softwarebibliotheken, -pakete und Frameworks, um die Entwicklung auf bestimmten Plattformen wie z.B. Smartphones oder Spielekonsolen zu ermöglichen. Softwareentwickler greifen auf diese Bibliotheken zu, um den Entwicklungsprozess zu beschleunigen. Mit SDK's werden z.B. Algorithmen bereitgestellt, die immer wieder gebraucht werden. In der Entwicklung für AR Anwendungen betrifft dies insbesondere das Tracking (Kapitel 2.3).

Für AR gibt es mehrere SDK's, die für die Entwicklung von Smartphone Anwendungen geeignet sind. Diese sind z.B. ARCore[16], ARKit[2], Vuforia[42], Wikitude[44] oder ARToolkit[4].

ARCore ist die AR Plattform von Google. Sie verwendet drei wichtige Funktionen, um virtuelle Inhalte in die reale Welt zu integrieren, die auch in dieser Anwendung eine wichtige Stütze für eine gute Nutzererfahrung sind: *Motion Tracking*, *Environmental understanding* und *Light estimation*.

### 5.2.1 Technische Voraussetzungen

ARCore kann von vielen Geräten mit Android 7.0 Nougat mit dem API Level 24 oder höher genutzt werden. Auch iOS Geräte mit iOS 11.0 oder höher sind mit ARCore kompatibel. Lediglich die Tiefen-API wird nicht von allen Geräten unterstützt. Die Dokumentation von ARCore[3] bietet eine Tabelle mit allen kompatiblen Geräten.

### 5.2.2 Motion Tracking

*Motion-Tracking* ermittelt die Position und Orientierung des Smartphones im Bezug zur Umgebung. Laut der offiziellen Dokumentation[3] und dem Patent zu Googles AR Core[14] wird ein SLAM System genutzt. Es werden Schlüsselpunkte in der Umgebung identifiziert. Diese werden mit den Daten aus der IMU kombiniert, um die relative Position und Orientierung des Smartphones zum Weltkoordinatensystem zu bestimmen. Mit Motion Tracking ist nur die Ermittlung der Position und Orientierung des Smartphones gemeint. Die Erstellung der Karte von der Umgebung wird ähnlich zu PTAM von Klein und Murray[23] getrennt. Die Karte wird in der Funktion *Environmental understanding* erstellt.

---

<sup>51</sup>[https://de.wikipedia.org/wiki/Software\\_Development\\_Kit](https://de.wikipedia.org/wiki/Software_Development_Kit), zuletzt aufgerufen am 09.08.2022

### 5.2.3 Environmental understanding

*Environmental understanding* erkennt die Größe und die Position aller Oberflächen in der Umgebung. Damit werden Oberflächen wie Tische, Wände oder Böden erfasst. Die Umgebung wird im Laufe der Anwendung kontinuierlich mit Schlüsselpunkten erweitert. Bestehende Schlüsselpunkte werden verbessert. Es wird eine Karte der Umgebung erzeugt, die aus Schlüsselpunkten besteht. AR Core ist dann dazu in der Lage zusammenhängende Schlüsselpunkte zu erkennen, um damit Flächen und Wände zu identifizieren. Es werden horizontale als auch vertikale Flächen erkannt.

Motion Tracking und Environmental understanding ist als Grundfunktion für diese Anwendung essentiell.

### 5.2.4 Light estimation

ARCore schätzt Informationen über die Lichtverhältnisse in der realen Umgebung. Es werden *lighting cues* in der Umgebung erkannt. Diese light cues beeinflussen die Farbe und die Helligkeit von jedem Pixel eines Bildes. Light cues sind:

- das Ambient light. Das ist das diffuse Licht aus der Umgebung,
- die Schatten bzw. Schattenwürfe, die Aufschluss darüber geben, aus welcher Richtung das Hauptlicht kommt,
- Shading, das die Intensität des Lichts bestimmt,
- spiegelnde Highlights. Diese entstehen, wenn das Licht direkt auf eine spiegelnde Fläche scheint,
- die Reflektionen von Objekten.

Light estimation hat zwei Modi: den *Ambient Intensity mode* und den *Environmental HDR Mode*.

**Ambient Intensity mode** Dieser Modus ermittelt mit dem Bild der Kamera die durchschnittliche Helligkeit und die durchschnittliche Farbtemperatur der Umgebung. Das Hauptlicht erhält damit eine Einfärbung in das Umgebungslicht.

**Environmental HDR Mode** Dieser Modus nutzt zusätzliche APIs mit Machine Learning Algorithmen. Das Kamerabild wird dazu genutzt die Beleuchtung der Umgebung zu verstehen. Environmental HDR Mode basiert auf die Arbeiten von LeGendre et al.[26]. Dabei wird das Bild nach den Light cues analysiert.

Environmental HDR berechnet drei Komponenten. Das *Main directional light* ist das Hauptlicht, dass die reale Szene aufhellt. In Outdoor AR ist dies in der Regel die Sonne. Diese API berechnet die Richtung und die Intensität des Hauptlichtes. Mit dieser Information werden z.B. spiegelnde

Highlights auf den Modellen berechnet. Außerdem wird die Intensität und die Richtung der Schatten angepasst.

Zusätzlich zum Hauptlicht, werden *Ambient spherical harmonics* berechnet. Sphärisch-harmonische Beleuchtung wird in Detail von Green[17] erklärt und ist eine Technik zur Berechnung von globaler Beleuchtung. Es wird aus dem Kamerabild Informationen zum Umgebungslicht berechnet, das zusätzlich zum Hauptlicht die Objekte beleuchtet.

Letztlich werden *HDR Cubemaps* in Echtzeit erstellt. Eine Cubemap ist ein Panoramabild, das Bilder einer Umgebung enthält. Das Panoramabild wird auf die sechs Flächen eines Würfels verteilt. Diese Cubemap wird verwendet, um zum einen Reflektionen der Umgebung bereitzustellen. Damit erhalten stark reflektierende Objekte, wie z.B. Metall, eine Spiegelung der Umgebung. Zum anderen wird das Umgebungslicht simuliert[Szelsiki Seite 410ff., 40]. Dies sorgt dafür, dass virtuelle Objekte harmonischer in das Bild integriert werden. In Abbildung 23 wird das Ergebnis aller Komponenten gezeigt.

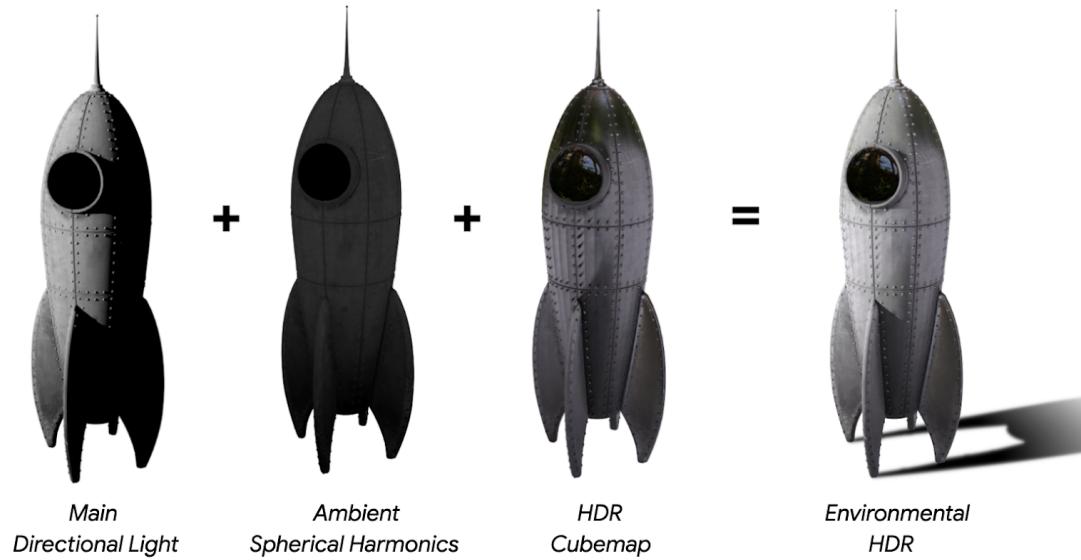


Abbildung 23: Die Beleuchtungseinschaf-ten und das kombinierte Endergebnis des Environmental HDR Modus<sup>52</sup>.

In dieser Arbeit wird dieser Modus verwendet, um die Reflektionen der Umgebung zu implementieren. Es werden in Kapitel 5.4 eigene Cubemaps aus Panoramabildern des SABA Gebäudes erstellt und benutzt. Dies ermöglicht das manuelle Einstellen der Wetterbedingungen. Damit wird in Kapitel 5.5 auch die Veränderung der Cubemaps sichtbar.

<sup>52</sup>Quelle Bild: <https://developers.googleblog.com/2019/05/ARCore-IO19.html>, zuletzt aufgerufen am 16.08.2022

Nachteil dabei ist, dass nur Cubemaps des SABA-Geländes genutzt werden. Da das SABA und Lyautey Gelände zum Zeitpunkt dieser Arbeit bebaut wird, verändert sich die Umgebung. Eine Erstellung von Cubemaps für jedes Gebäude würde sich demnach nicht lohnen, da sich der Ort verändert. Für eine weiterführende Arbeit wird daher empfohlen den Environmental HDR Mode zu verwenden. Die Cubemaps für die Reflektionen und das Environmental Lighting wird sich dann an den realen Gegebenheiten anpassen.

### 5.2.5 User Interaction

Um Interaktionen des Nutzers zu den virtuellen Objekten zu implementieren, wird *hit-testing* durchgeführt. In AR Foundation bzw. Unity wird dies *Raycast* genannt. Daher wird diese Funktion nicht mit ARCore, sondern mit Unity implementiert. Das Prinzip ist gleich. Beim hit-testing wird ein virtueller Strahl vom Smartphone aus in die Umgebung erzeugt. Es werden Schnittpunkte zwischen diesen Strahl und virtuellen Objekten gefunden. Das *Hit-Ergebnis* beinhaltet drei Informationen:

- die Distanz des Smartphones zum Objekt,
- die Position und die Orientierung des hits im Weltkoordinatensystem,
- die 3D Gemoetrie des getroffenen Objekts.

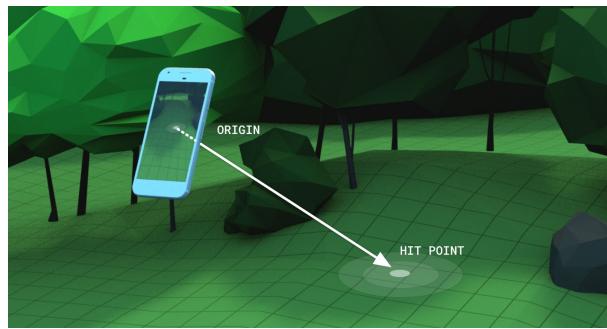


Abbildung 24: Der hit-testing Strahl vom Smartphone trifft auf eine virtuelle Fläche und gibt ein Hit Ergebnis zurück.<sup>53</sup>

In dieser Anwendung wird hit-testing dafür verwendet die Modelle auf den Flächen zu platzieren. Wird ein Hit zurückgegeben, wird dem Nutzer ein Pfeil auf der jeweiligen Fläche angezeigt. Der Pfeil dient als *Placement Indicator*. Es hilft dem Nutzer bei der Platzierung der Objekte. Tippt der Nutzer auf den Bildschirm, wird das Gebäude an der Position des Hits platziert. Außerdem wird das hit-testing dazu genutzt bereits platzierte Gebäude zu detektieren. Die Implementierung in dieser Anwendug wird in Kapitel 5.3.2 erläutert.

<sup>53</sup>Quelle Bild: <https://developers.google.com/ar/develop/hit-test>, zuletzt aufgerufen am 09.08.2022

### 5.2.6 Depth API

Die Depth API generiert Tiefenbilder (*depth maps*) um Tiefeninformationen der Umgebung zu erhalten. Es basiert auf der Arbeit von Flynn et al.[15]. Mit den Tiefenbildern werden die Position und die Größe von Objekten besser erkannt. Dies wird z.B. dafür genutzt, um *Verdeckung* (*engl. Occlusion*) umzusetzen. Occlusion wird dann eingesetzt, wenn reale Objekte, die Näher am Betrachter liegen, das virtuelle Objekt verdecken. Wie in Abbildung 25 zu sehen wird die Katze vom realen Objekt verdeckt. Die Glaubwürdigkeit, dass eine echte Katze im Bild ist, steigt.

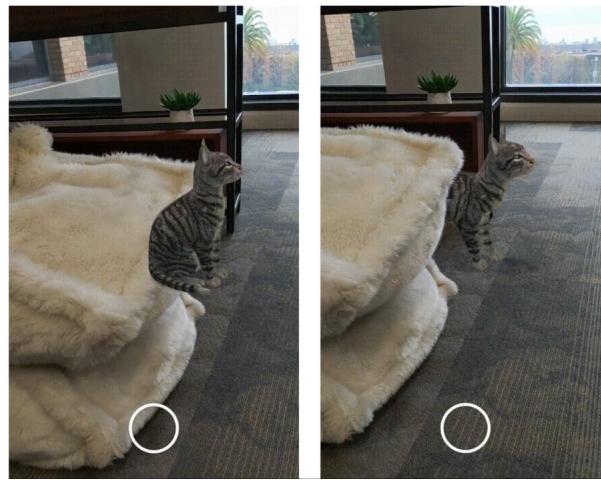


Abbildung 25: AR ohne Occlusion (l.) und mit Occlusion (r.). Die virtuelle Katze wird durch das Objekt verdeckt.<sup>54</sup>

ARCore bietet zwei Ausgabemöglichkeiten für das Tiefenbild:

- Raw Depth API liefert ein Tiefenbild mit hoher Genauigkeit. Dabei kann es vorkommen, dass nicht alle Pixel im Bild auch ein Tiefenwert erhält. Zusätzlich wird ein Konfidenzbild erzeugt.
- Full Depth API liefert ein geglättetes Tiefenbild. Pixel, die keinen Tiefenwert haben, erhalten einen geschätzten Wert.

Ein Konfidenzbild gibt jedem Pixel mit einem Tiefenwert einen Konfidenzwert zwischen 0 und 1. Je höher dieser Wert, desto höher ist die Wahrscheinlichkeit, dass die Tiefeninformation korrekt ist. Im Konfidenzbild wird dies durch die Pixelhelligkeit dargestellt. Je heller der Pixel im Konfidenzbild ist, desto höher ist der Konfidenzwert. Diese Funktion ist nützlich, um mit einem Threshold ungenaue Tiefeninformation auszuschließen.

Die Occlusion der Depth API zeigt in dieser Anwendung Probleme. Diese werden in Kapitel 5.7.1 aufgezeigt. Für den Anwendungszweck dieser Arbeit wird Occlusion nur zum Testen für Entwickler

<sup>54</sup>Quelle Bild: <https://developers.google.com/ar/develop/depth>, zuletzt aufgerufen am 13.08.2022

in die Anwendung implementiert, da Ungenauigkeiten und Artefakte die Nutzererfahrung negativ beeinflussen.

### 5.3 Umsetzung der Anwendung

Dieser Abschnitt erläutert den Aufbau der Szenen in Unity, um einen Überblick der erforderlichen Komponenten zu erhalten. Zunächst wird darauf eingegangen, wie eine Szene in AR Foundation aufgebaut ist. Dann wird ein Überblick über die verwendeten Skripte und deren Funktion in der Anwendung geschaffen, damit sich interessierte Entwickler schnell zurecht finden.

Im Anschluss wird die Anbindung mit der REST-API erklärt. Die Antwort der API und die damit erzeugten Effekte werden aufgezeigt.

#### 5.3.1 Grundaufbau der AR Szenen in Unity

Bei der Erstellung eines neuen Projekts in Unity kann ein Standard Template für die Entwicklung von AR ausgewählt werden. Dieses beinhaltet Beispielszenen, Assets und die benötigten Packages. Diese können in normalen Unity Projekten im Package Manager installiert werden. Die benötigten Packages sind wie in Abbildung 26 zu sehen in einem Gesamtpaket für AR vorhanden.

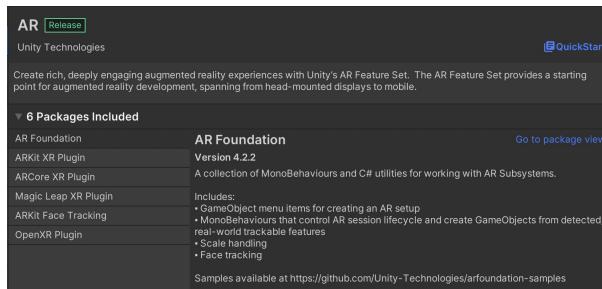


Abbildung 26: Die benötigten Packages im Package Manager.

Damit ARCore Zugriff auf die Packages hat, muss in den Projekteinstellungen unter *XR Plug-in Management* Einstellungen vorgenommen werden. Unter dem Tab für Android muss ARCore als Plug-in Provider aktiviert werden. Damit wird das Plug-In für ARCore aktiviert und es kann auf die Kamera des Smartphones zurückgegriffen werden.

Eine einfache AR Szene besteht aus drei wichtigen Szenenobjekten, die in Abbildung 27 zu sehen sind: die *AR Session*, die *AR Session Origin* und die *AR Camera*.

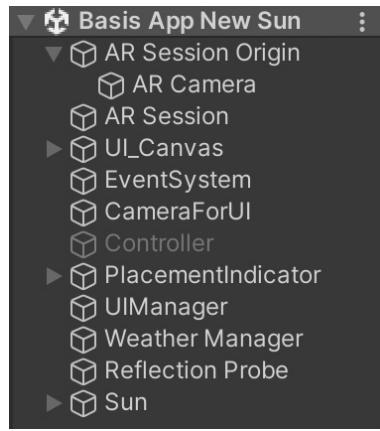


Abbildung 27: Der Szenenaufbau für die Position auf beliebigen Flächen.

Die *AR Session* ist ein *GameObject*, dass das *AR Session Skript* enthält. Dieses sorgt für das Tracking. Wird das *AR Session Skript* deaktiviert, werden keine Schlüsselpunkte mehr erzeugt. Wird das Skript wieder aktiviert, wird versucht die bestehenden Schlüsselpunkte weiterhin zu nutzen.

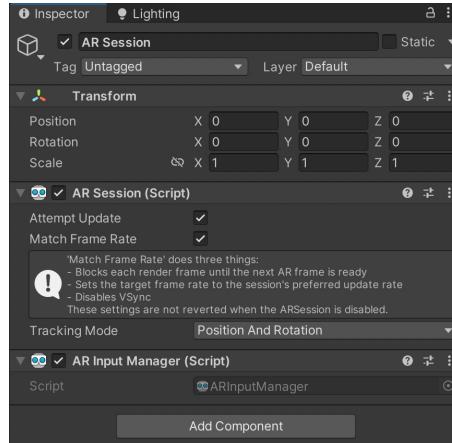


Abbildung 28: Die AR Session.

Die *AR Session Origin* ist auch ein *GameObject*, an dem das *AR Session Origin Skript* hängt. Es transformiert die Trackable Objekte, wie z.B. Schlüsselpunkte und Flächen, in ihre finale Position, Orientierung und Skalierung in das Weltkoordinatensystem von Unity. Die Skalierung der *AR Session Origin* bestimmt damit auch die Skalierung der Objekte in der Szene. Ist die Skalierung auf 0.1 gesetzt, so werden die Objekte in der Szene um den Faktor 0.1 verkleinert. Dies ist ein wichtiger Aspekt für die korrekte Skalierung der Gebäude, damit diese in der GPS Platzierung in der korrekten Größe dargestellt werden. Außerdem werden an der *AR Session Origin* die Skripte angehängt, die für einige *ARCore* Funktionen zuständig sind. So hängt für diese Anwendung der *AR Plane Manager* und der *AR Raycast Manager* daran.

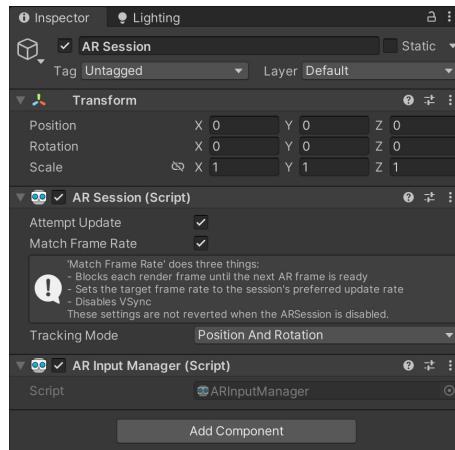


Abbildung 29: Die AR Session Origin mit dem AR Plane Manager und dem AR Raycast Manager.

Das dritte Szenenobjekt ist die *AR Camera*. Dies ist die Kamera von Unity mit zusätzlichen Komponenten. Diese sind der *AR Camera Manager*, der *AR Camera Background* und der *Tracked Pose Driver*. Im AR Camera Manager wird die Light estimation und dessen Modus aktiviert. Die *Facing Direction* bestimmt, ob die Front- oder die Rückkamera des Smartphones genutzt wird. Das AR Camera Background Skript sorgt dafür, dass das Kamerabild des Smartphone als Hintergrund angezeigt wird. Im Tracked Pose Driver werden Einstellungen vorgenommen, die für die Bestimmung der Position und Orientierung benötigt werden. Im Falle von AR im Smartphone wird hier festgelegt, dass die Pose über die Kamera berechnet wird. Die AR Camera wird als Kindelement der AR Session Origin eingefügt.

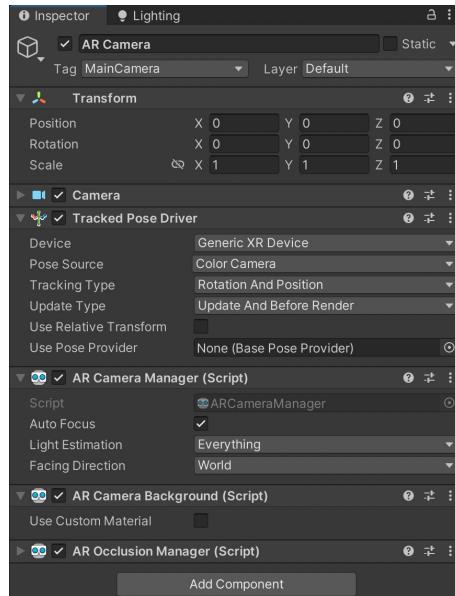


Abbildung 30: Die AR Camera. Der Occlusion Manager aus der Depth API wird hier als Komponente eingefügt.

Diese drei Elemente werden für jede AR Anwendung in AR Foundation benötigt. Ausführliche Erklärungen zu den einzelnen Komponenten sind in der AR Foundation Dokumentation zu finden [41].

### 5.3.2 Controller

Der *Controller* in der Szene aus Abbildung 27 ist ein *GameObject*, das die Logik der Interaktionen beinhaltet. Alle Interaktionsmöglichkeiten (bis auf das User Interface) werden über dieses *GameObject* geregelt. In den folgenden Abschnitt werden diese Funktionen kurz erläutert, damit ein Überblick über die verwendeten Skripte geschaffen wird.

#### 5.3.2.1 Platzieren auf einer beliebigen Fläche

Für die Platzierung der Gebäude ist das Skript `ARPlaceObjectAndMove.cs` zuständig. Wie in Kapitel 5.2.5 erwähnt, wird in dieser Anwendung Raycast verwendet, um bei einem Hit einen Pfeil auf einer erkannten Ebene zu zeigen. Dieser dient als Placement Indicator. Wird eine Fläche erkannt, wird mit einem Tip auf dem Bildschirm das Gebäude platziert. Die Position und Orientierung wird vom letzten registrierten Hit bestimmt. Die Implementierung ist mit dem Code von Kris Schultz erfolgt<sup>55</sup>. Die Abbildung 31 zeigt den Placemet Indicator und wie die das Gebäude platziert wird.

---

<sup>55</sup><https://github.com/TheUnityWorkbench/tuw-arfoundation-demo/blob/master/Assets/Demo/ARTapToPlaceObject.cs>, zuletzt aufgerufen am 13.08.2022

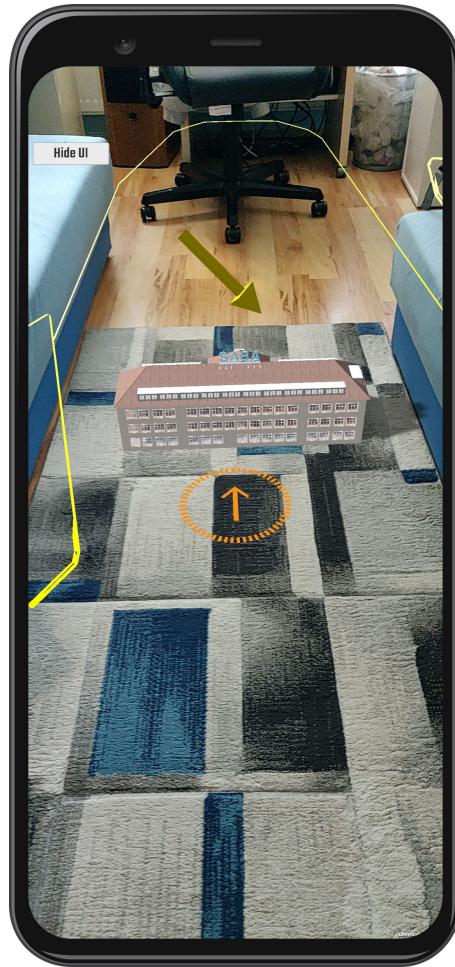


Abbildung 31: Der Pfeil auf dem Boden signalisiert, dass auf dieser Fläche Objekte platziert werden können.

Die Funktion `ARPlaceObject()`<sup>56</sup> wird für diese Anwendung modifiziert. Dies ist nötig, um die Funktion zu implementieren, dass das vom Nutzer ausgewählte Objekt platziert wird. Außerdem wird überprüft, ob es bereits eine Instanz des Gebäudes in der Szene gibt. Es soll nicht möglich sein ein Gebäude mehrmals zu platzieren.

Hierfür werden die aus Kapitel 3.1 erwähnten BuildingGPS Scriptable Objects benötigt. Es werden zwei Listen geführt. Die Liste in Zeile 11 im unteren Code beinhaltet nur die Prefabs, die für die `Instantiate()` Funktion gebraucht wird. Die zweite Liste in Zeile 19 wird geführt, um das Scriptable Object des ausgewählten Gebäudes zu speichern. Tippt der Nutzer nach der Platzierung nochmal auf den Bildschirm, wird die ID des ausgewählten mit den ID's aus der Liste verglichen. Stimmt eine ID überein, ist das ausgewählte Gebäude bereits in der Szene vorhanden. Die `ARPlaceObject()` Funktion wird nicht ausgeführt.

<sup>56</sup>unter Schultz als `PlaceObject()` benannt

```

1 public void ARPlaceObject()
2 {
3     //Get the selected Object from the ObjectSelection Script. This Script is
4     //attached to the UIManager.
5     choosedObject = selectionMenuScript.selectedObject;
6     foreach (var currentBuilding in buildings)
7     {
8         //Check if there is an instance of the choosed building in the scene.
9         if (currentBuilding.id == choosedObject.id)
10        {
11            //Add the object to a list.
12            listOfSpawnedObjects.Add(
13                Instantiate(
14                    currentBuilding.prefab,
15                    placementPose.position,
16                    placementPose.rotation
17                )
18            );
19            //add the object to the building info list, in order to check earlier,
20            //if the building already exists
21            listOfSpawnedObjectsBuildingInfo.Add(choosedObject);
22        }
23    }
24 } // end function ARPlaceObject()

```

Listing 1: Die ARPlaceObject() Funktion.

**Pinch Funktion zur Skalierung** Neben der Platzierung der Gebäude wird im ARPlaceObjectAndMove.cs Skript ermöglicht, die Gebäude zu Skalieren und auf einer Fläche zu bewegen. Die Skalierung wird dann ausgeführt, wenn zwei Finger gleichzeitig den Bildschirm berühren. Die Distanz der beiden Finger während der Pinch Bewegung bilden den Wert für die Skalierung<sup>57</sup>.

**Gebäudewahl und Positionsveränderung auf einer Fläche** Die Bewegung auf einer Fläche wird mit einem Raycast implementiert. Dabei wird ein Strahl von der Position des Touches erzeugt. Das Gebäude wird an die Position verschoben, bei der der Strahl die Fläche geschnitten hat. Der verwendete Code ist aus dem Github Repository von Dilmer Valecillos<sup>58</sup>.

<sup>57</sup>Der verwendete Code: <https://www.youtube.com/watch?v=ISBIu6Jzfk8>, zuletzt aufgerufen am 13.08.2022

<sup>58</sup><https://github.com/dilmerv/UnityARFoundationEssentials/blob/master/Assets/Scripts/PlacementWithDraggingDroppingController.cs>, zuletzt aufgerufen am 13.08.2022

### 5.3.2.2 Objekte auswählen

Für das auswählen der Objekte im `SelectObject()` Skript wird wieder ein Raycast genutzt. Von der Position der Mitte des Bildschirms wird ein Strahl geschickt. Wird ein Hit registriert, so wird zunächst überprüft, ob das Objekt ein Gebäude ist. Zur Identifikation werden *Tags* genutzt. Unity bietet die Möglichkeit jedem GameObject einen Tag zu geben. Ein Tag ist ein String, dass als Referenz für das GameObject genutzt werden kann<sup>59</sup>. Wie in Abbildung 32 zu sehen, bietet Unity vorgefertigte Tags. Es können auch eigene Tags definiert werden, wie in diesem Anwendungsfall der *Selectable* Tag. Wird ein Gebäude identifiziert, erhält das Gebäude einen Outline Shader. Dieser fügt dem Gebäude eine Umrandung hinzu, die dem Nutzer signalisiert, dass das Gebäude ausgewählt werden kann. Mit den Button *Select* und *Deselect* kann der Nutzer die Gebäude auswählen und anschließend verschieben, rotieren, skalieren und entfernen.

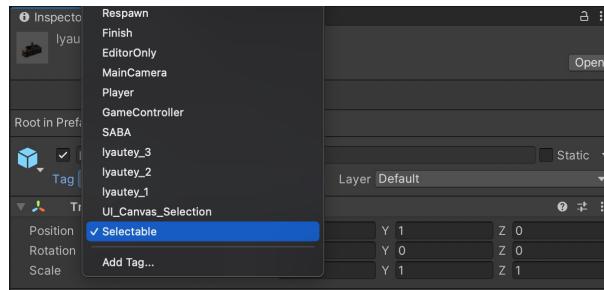


Abbildung 32: Die Tag-Auswahl im Unity Inspector.

**Gebäude entfernen** Zur Entfernung der Gebäude ist das Skript `DestroyObject.cs` zuständig. Mit dem Löschen Button wird die Funktion gestartet. Damit der Nutzer nicht aus versehen Gebäude löscht, wird ein Gebäude erst dann entfernt, wenn der Button für zwei Sekunden gedrückt wird. Es werden die zwei Listen aus dem `ARPlaceObjectAndMove.cs` Skript bearbeitet. Mit dem Befehl `Remove()` wird das jeweilige Gebäude aus den Listen entfernt und die Instanz des Gebäudes wird mit `Destroy()` zerstört.

### 5.3.3 GPS Platzierung

Die Platzierung von Objekten über die GPS Informationen des Smartphones und der Angabe von GPS Längen- und Breitengraden ist nativ nicht mit AR Foundation möglich. Es können zwar GPS Informationen des Smartphones mit dem *LocationService*<sup>60</sup> von Unity abgerufen werden. Das Platzieren von Objekten über GPS Koordinaten muss selbst implementiert werden. Hierfür wird das Asset *AR + GPS Location* aus dem Asset Store benutzt<sup>61</sup>. Dieses Asset hat eine umfangreiche

<sup>59</sup><https://docs.unity3d.com/Manual/Tags.html>, zuletzt aufgerufen am 15.08.2022

<sup>60</sup><https://docs.unity3d.com/ScriptReference/LocationService.html>, zuletzt aufgerufen am 15.08.2022

<sup>61</sup><https://assetstore.unity.com/packages/tools/integration/ar-gps-location-134882>, zuletzt aufgerufen am 15.08.2022

Dokumentation<sup>62</sup>. Für das Asset werden weitere GameObjects in der Szene benötigt. In der *AR Session Origin* wird ein *AR Location Root* Objekt erzeugt. Außerdem wird in die Szene ein *GPS Stage Object* gebraucht, das das *PlaceAtLocation.cs* Skript trägt. Dieses Objekt dient als Parent für jedes Objekt, das in der Szene platziert werden soll. Die jeweiligen GPS Koordinaten können im Inspector eingetragen werden.

Da der Nutzer keine Auswahl hat, welches Gebäude platziert wird, wird jedes Gebäude aus der hinterlegten Datenbank platziert. Voraussetzung ist, dass das Gebäude innerhalb der eingestellten Render Distanz von Unity liegt. Ansonsten wird das in Kapitel 2.4.2.3 eingeführte Clipping durchgeführt und die Gebäude außerhalb des Einheitswürfels werden abgeschnitten. In dieser Anwendung ist die Render Distanz auf 250 Unity Einheiten gesetzt, was 250 Metern entspricht<sup>63</sup>. Dies genügt, um drei Gebäude nebeneinander zu platzieren. So ist es vor Ort mit den Gebäuden 2, 3 und 4 der Fall ist.

Für die GPS Platzierung wird das *ARPlaceObjectAndMove* Skript angepasst. Die *MoveObject()* Funktion wird nicht mehr benötigt. Der untere Code zeigt die Anpassungen an der *ARPlaceObject()* Funktion. Für jedes Gebäude wird zunächst eine *Location* erzeugt. Dies ist ein Scriptable Object vom Asset<sup>64</sup>. Hier werden die Längen-, Breiten- und Höhengrade eingetragen. Diese Angaben stammen aus der Datenbank der Gebäude. Der *AltitudeMode* bestimmt, ob der eingegebene Höhengrad verwendet werden soll, oder ob eine detektierte Fläche von ARCore für die Platzierung genutzt werden soll. Da die Höhengrade in der Regel nicht sehr genau sind, werden die erkannten Flächen von ARCore genutzt. Anschließend werden Optionen gesetzt, die für die Platzierung im nächsten Schritt benötigt werden. Mit der Funktion *CreatePlacedInstance()*, die vom Asset bereitgestellt wird, wird das jeweilige Gebäude platziert.

```

1 void ARPlaceObject()
2 {
3     foreach (var currentBuilding in buildings)
4     {
5         //set new location data
6         var location = new Location()
7         {
8             Latitude = currentBuilding.latitude,
9             Longitude = currentBuilding.longitude,
10            Altitude = currentBuilding.altitude,
11            AltitudeMode = AltitudeMode.GroundRelative
12        };
13        //set options
14        var options = new PlaceAtLocation.PlaceAtOptions()

```

<sup>62</sup><https://docs.unity-ar-gps-location.com/>, zuletzt aufgerufen am 15.08.2022

<sup>63</sup><https://docs.unity3d.com/ScriptReference/Camera-farClipPlane.html>, zuletzt aufgerufen am 15.08.2022

<sup>64</sup><https://docs.unity-ar-gps-location.com/guide/#locationdata>, zuletzt aufgerufen am 15.08.2022

```

15     {
16         HideObjectUntilItIsPlaced = true,
17         MaxNumberOfLocationUpdates = 2,
18         MovementSmoothing = 0.1f,
19         UseMovingAverage = false
20     };
21     //create Instance
22     spawnedObject = PlaceAtLocation.CreatePlacedInstance(
23         currentBuilding.prefab,
24         location,
25         options,
26         false
27     );
28     //set Rotation to fit real facing direction
29     spawnedObject.transform.rotation = Quaternion.Euler(0, 0, 0);
30     var currentHeading = locationProvider.CurrentHeading.heading;
31     spawnedObject.transform.Rotate(
32         0,
33         (float)currentHeading - currentBuilding.realWorldOrientation,
34         0,
35         Space.World
36     );
37     listOfSpawnedObjects.Add(spawnedObject);
38 }
39 }
```

Listing 2: Die angepasste `ARPlaceObject()` Funktion für die Platzierung mit GPS.

### 5.3.3.1 Ausrichtung der Szene nach Norden

Wie in Kapitel 2.3.1 erwähnt, wird beim Start einer Unity Szene die Richtung bestimmt, in die die z-Achse des Weltkoordinatensystems zeigen wird. In AR Foundation ist dies die Richtung, in die die Kamera des Smartphones zeigt. Damit die Gebäude in die richtige Richtung zeigen, müsste der Nutzer die Szene dann starten, wenn das Smartphone genau nach Norden zeigt. Es wird ein Skript gebraucht, um die Szene nach Norden auszurichten.

Mit Unity wird auf den Kompass des Smartphones zugegriffen. Da der Kompass in Smartphones ungenau ist, wird ein Tiefpassfilter verwendet. Dieser berechnet den Durchschnitt der letzten 10 Kompasswerte, um grobe Ausreißer der schwankenden Kompasswerte zu ignorieren. Die Kompasswerte liegen zwischen  $0^\circ$  und  $360^\circ$ . Falls der Wert bei Nordkurs zwischen  $359^\circ$  bis  $1^\circ$  schwankt, kann es zu Diskontinuitäten kommen. Der Durchschnittswert liegt dann bei  $180^\circ$ . Um dies zu vermeiden wird der Tiefpassfilter mit Sinus und Kosinus berechnet. Dies ist im unteren Code in der Zeile 24 zu sehen.

Mit den Button *Reorient* kann der Nutzer die Szene neu ausrichten. Das Ausrichten wirkt sich

sowohl in der GPS Anwendung auf die Rotation der Gebäude aus, als auch auf die Position der Sonne. Die Sonne wird in Kapitel 5.4 behandelt.

```

1 public float CalculateAverage()
2 {
3     int dataSize = data.Count;
4     //use low pass filter to determine the average compass direction
5     float average = (float)(180 / Math.PI) * (float) Math.Atan2(sumSin / dataSize,
6     sumCos / dataSize);
7     return average;
8 }
```

Listing 3: Der Tiefpassfilter im `ReorientToNorth.cs` Skript

**Ausrichtung der Gebäude** Wenn die Szene nach Norden ausgerichtet ist, zeigt das Prefab des Gebäudes in die Richtung, mit der das Gebäude aus Blender exportiert wird. Die z-Achse des lokalen Koordinatensystems des SABA Gebäudes ist z.B. in Abbildung 33(a) zu sehen. Damit das SABA Gebäude in die korrekte Richtung zeigt, wird das Gebäude um  $72.47^\circ$  in der y-Achse gedreht. Dieser Wert wird mithilfe einer Karte des Vermessungsamts Villingen-Schwenningen berechnet. Dieser Wert wird mit der Karte vom Gelände berechnet.

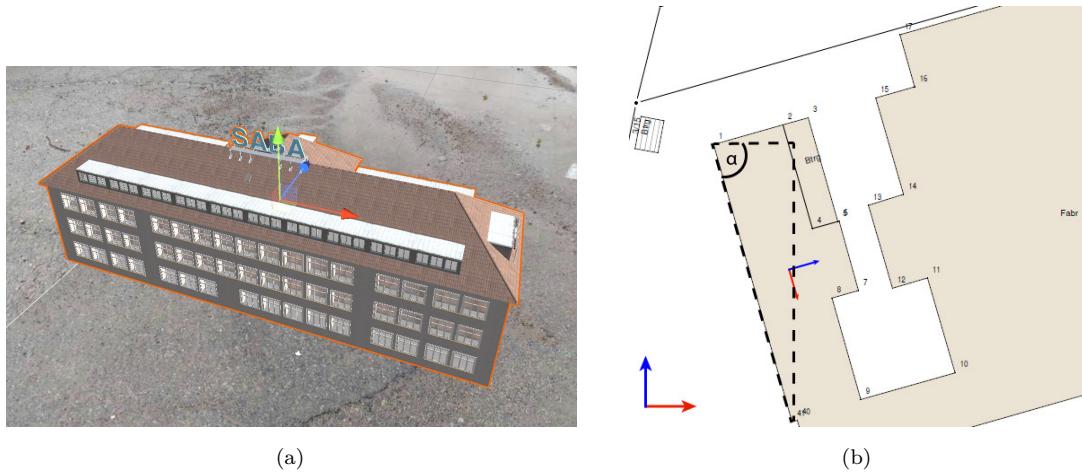


Abbildung 33: Das lokale Koordinatensystem des SABA Gebäudes zeigt mit der z-Richtung (blauer Pfeil) nach Norden(a). Um den Winkel  $\alpha$  wird das Gebäude gedreht, damit die Rotation des Gebäudes mit den realen Gebäude übereinstimmt(b).

Folgende Winkel werden für die Gebäude in dieser Anwendung verwendet:

Gebäude	Nr.	lokales Koordinatensystem	Winkel
SABA	1		74,32°
Reithalle	2		33,54°
Mannschaftsgebäude 1	3		208,00°
Wirtschaftsgebäude <sup>65</sup>	4		23,10°
Mannschaftsgebäude 2	5		20,40°

Tabelle 2: Die lokalen Koordinatensysteme der Gebäude und deren Winkelangaben zur korrekten Rotation.

<sup>65</sup>Wie am Bild in der Tabelle zu sehen, ist das Gebäude bereits um den Winkel rotiert.

## 5.4 Belichtung

Die Belichtung der Szenen besteht aus drei Komponenten. Mit der Light Estimation API wird der *Ambient Intensity Mode* genutzt, um die durchschnittliche Helligkeit und die durchschnittliche Farbtemperatur zu bestimmen. Mit diesen Werten wird das Hauptlicht gefärbt, um die realen Lichtbedingungen einwirken zu lassen. Das Hauptlicht ist ein direktionales Licht, das die gesamte virtuelle Szene aufhellt. Dieses Licht simuliert die Sonne. Letzlich werden HDR Panoramabilder des SABA Geländes angefertigt, um Cubemaps zu generieren und *Environmental Lighting* zu ermöglichen. In den folgenden Abschnitten wird die Implementierung der verwendeten Techniken gezeigt. Vorher und Nachher Vergleiche zeigen die Unterschiede bei der Belichtung.

### 5.4.1 Light Estimation in dieser Anwendung

### 5.4.2 Sonne simulieren

## 5.5 Wetterbedingungen

Dieses Kapitel behandelt die Umsetzung der Veränderungen an der Szene anhand von Wetterdaten. Zunächst wird die auf die Anbindung an die Wetter REST-API in Unity eingegangen. Danach wird veranschaulicht, welche Parameter angepasst werden. Ein anschließender Vergleich der einzelnen Wetterbedingungen zeigt die Auswirkungen auf die Optik der Gebäude.

### 5.5.1 Anbindung der REST-API

Um Wetterdaten in *Open Weather Map* abzufragen, wird ein Account benötigt. Genutzt wird die *Current Weather API*<sup>66</sup>. Nach der Anmeldung wird im Account ein API Key generiert, der später im Skript für die Anfrage benötigt wird.

Die Antwort der API erfolgt über eine JavaScript Object Notation (JSON) Datei. Es ist ein Datenformat, das zum Datenaustausch zwischen Anwendungen genutzt wird. JSON Daten liegen in einfach lesbare Textform bereit und ist Programmiersprachen unabhängig<sup>67</sup>. Um die Daten der JSON Datei zu nutzen, wird ein JSON-Parser in Unity benötigt. In dieser Anwendung wird SimpleJSON<sup>68</sup> genutzt. Das Skript wird als Asset in das Unity Projekt eingefügt.

Das Wetter wird in der Szene vom *Weather Manager* und dem *WeatherManager.cs* Skript geregelt. Zunächst wird in einer *Coroutine* durchgeführt, um die Längen- und Breitengrade der aktuellen Position zu ermitteln. Mit einer *Coroutine* werden in Unity Arbeitsschritte durchgeführt. Zum Beispiel wird für eine Funktion 1 das Ergebnis einer anderen Funktion 2 benötigt. Mit einer

<sup>66</sup><https://openweathermap.org/current>, zuletzt aufgerufen am 19.08.2022

<sup>67</sup>[https://de.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://de.wikipedia.org/wiki/JavaScript_Object_Notation), zuletzt aufgerufen am 19.08.2022

<sup>68</sup><https://github.com/Bunny83/SimpleJSON>, zuletzt aufgerufen am 19.08.2022

*Coroutine* wird dafür gesorgt, dass die Funktion 2 ausgeführt wird und Funktion 1 erst dann ausgeführt wird, wenn Funktion 2 vollständig durchlaufen ist. In diesem Fall wird für die Anfrage der Daten die Längen- und Breitengrade benötigt. Erst wenn diese vorhanden sind, erfolgt eine Anfrage der Daten.

```

1  private IEnumerator FetchWeatherDataFromApi(string latitude, string longitude)
2  {
3      string url = currentWeatherApi + "lat=" + latitude + "&lon=" + longitude + "
4      &appid=" + apiKey + "&units=metric";
5      UnityWebRequest fetchWeatherRequest = UnityWebRequest.Get(url);
6      yield return fetchWeatherRequest.SendWebRequest();
7      if (fetchWeatherRequest.isNetworkError || fetchWeatherRequest.isHttpError)
8      {
9          //Check and print error
10         statusText.text = fetchWeatherRequest.error;
11     }
12     else
13     {
14         Debug.Log(fetchWeatherRequest.downloadHandler.text);
15         var response = JSON.Parse(fetchWeatherRequest.downloadHandler.text);
16
17         //set UI Text
18         location.text = "Ort: " + response["name"];
19         weatherIDText.text = "WetterID: " + response["weather"][0]["id"];
20         mainWeather.text = "Wetter: " + response["weather"][0]["main"];
21         description.text = "Beschreibung: " + response["weather"][0]["
22         description"];
23         temp.text = response["main"]["temp"] + " C";
24
25         //fetch data for further functions
26         weatherID = response["weather"][0]["id"];
27
28         if (IsWeatherImpactEnabled.isOn)
29         {
30             ChooseWeatherCase(weatherID);
31         }
32     }
33 }
```

Listing 4: Die ARPlaceObject() Funktion.

### 5.5.2 Veränderungen der Wetter Parameter

## 5.6 Verwendete Shader

## 5.7 Probleme und Lösungen

### 5.7.1 Grenzen der Depth API und Occlusion

Wie in Kapitel 2.2 beschrieben trägt eine gute Verdeckung der virtuellen Objekte dazu bei, die Immersion in AR zu steigern[Breen et al. (1995) 9][Shah et al. (2012) 38]. Umso wichtiger ist es, dass diese Funktion auch in dieser Anwendung gut funktioniert. Dabei gibt es drei Problematiken, die bei der Entwicklung in dieser Arbeit zum Vorschein kommen.

Das erste Problem ist, dass diese API nicht für jedes Gerät verfügbar ist. Die Dokumentation von ARCore[3] bietet eine Tabelle mit allen kompatiblen Geräten. Damit kann nicht sichergestellt werden, dass die Anwendung auf jedem Gerät gleich funktioniert.

Ein weiteres Problem wird in Abbildung 34 deutlich. In diesem Beispiel wird Occlusion mit dem *AR Occlusion Manager* implementiert, der an die *AR Camera* angehängt wird. Die AR Camera wird in Kapitel 5.3.1 erläutert. Das Beispiel ist vor Ort an einer Mauer vor dem Gebäude 4 aufgenommen. Das rote Quadrat ist die erzeugte Depth Map. Je dunkler die rote Farbe, desto näher ist das Objekt zur Kamera. Es wird deutlich, dass erst ab einer Entfernung von 1m überhaupt Tiefe erkannt wird. Aus einer Entfernung ab 1,5m wird keine Tiefe mehr erkannt. Dies ist insbesondere für AR im Freien problematisch.

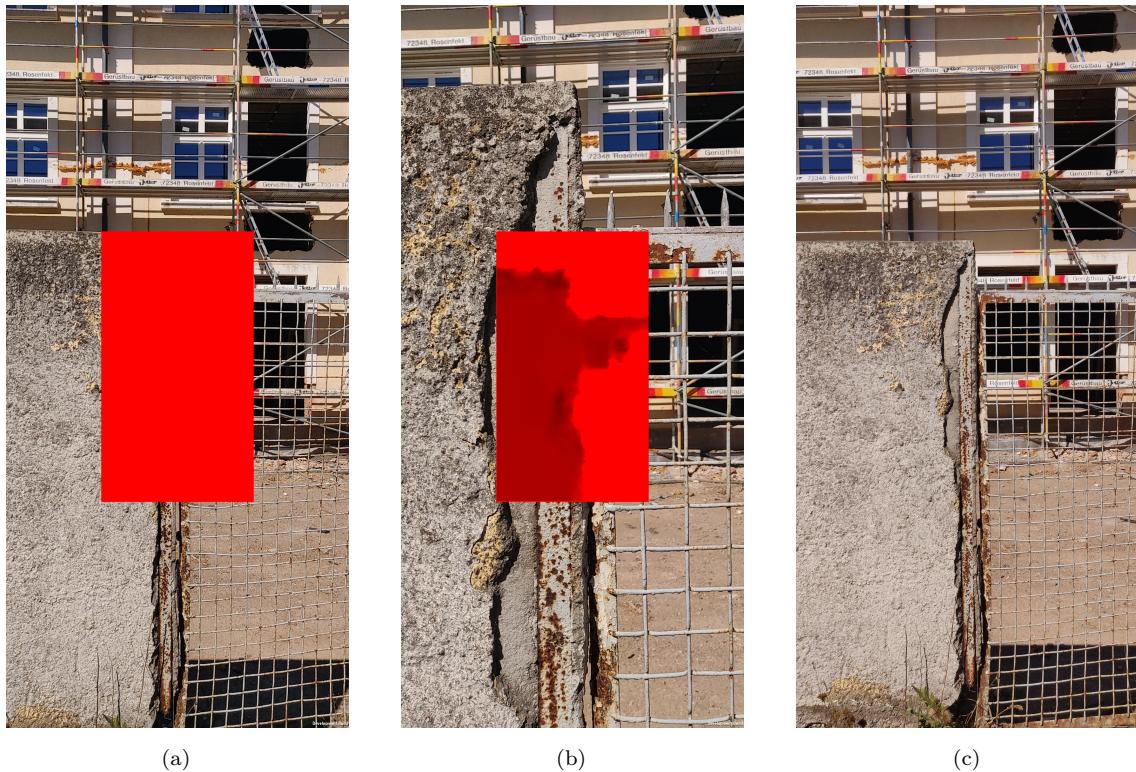


Abbildung 34: Die Depth Map aus einer Entfernung von 1,5m (a) im Vergleich zu der Depth Map aus 1m (b) und die Szene selbst mit der Mauer und dem Zaun aus 1,5m Entfernung(c).

In der Regel sind bei der Nutzung von GPS Positionsabweichungen von bis zu 10 Metern möglich. Eine genaue Platzierung der Gebäude ist daher problematisch.

Um die Tracking-Genauigkeit zu erhöhen, gibt es mehrere Methoden. *DGPS (engl. Differential GPS)* verbessert GPS-Signale, indem es ein Korrektursignal durch eine ortsfeste Referenzstation mit bekannter Lokalisierung berechnet. Da es in Deutschland lediglich acht solcher Stationen gibt und einige Anbieter nur kommerziell die Daten bereitstellen, ist diese Methode nicht für diese Arbeit geeignet<sup>69</sup> <sup>70</sup>. Eine weitere bekannte Möglichkeit bietet *SBAS (engl. Satellite Based Augmentation System)*, bei dem mehrere geostationäre Satelliten das GPS Signal auf bis zu einem Meter Genauigkeit zu verbessern [13].

Platinsky et al.[32] erstellen für ein besseres Tracking bei fehlender GPS Genauigkeit ein 3D-Modell der Umgebung. Bei der anschließenden AR Nutzung in diesem Gebiet wird auf dem Smartphone SLAM betrieben. Die Daten vom Smartphone werden mit der 3D-Karte verglichen, um ein genaueres Tracking durchzuführen.

<sup>69</sup><https://www.heise.de/newsticker/meldung/Differential-GPS-und-WLAN-RTT-Praeziere-Ortung-mit-Android-P-4046935.html>

<sup>70</sup> Liste von DGPS-Sendern: <https://www.ndblist.info/datamodes/worldDGPSdatabase.pdf>



## Abbildungsverzeichnis

1	Das 3D Modell des SABA Hauptgebäudes in der 3D Karte aus den Drohnen-Aufnahmen.	2
2	Eine Übersicht der Gebäude auf dem Lyautey-Gelände.	3
3	Eine Übersicht der Gebäude auf dem Mangin-Gelände.	4
4	Das Reality-Virtuality Kontinuum nach Milgram[29].	7
5	Kanji und Hiro Marker haben einfache geometrische Formen, Texte oder Buchstaben zur Erkennung in der Szene.	10
6	Der Tracking Prozess von ARToolkit[Billinghurst(2015), 8].	11
7	Die Wahrscheinlichkeit, dass eine Fläche aus dem linken Bild im rechten Bild genau identifiziert werden kann steigt, wenn starke Kontraständerungen (Gradienten) im Teilbild vorhanden sind.	12
8	Die Gauß-Pyramide für ein Eingangsbild. Das Bild wird runterskaliert und mit steigender Stärke mit einem Gauss-Weichzeichner versehen.	14
9	Detektierte Schlüsselpunkte anhand der Extremas der DoG Bilder und deren Nachbarbilder (a) und die Grauwerte des Punktes auf der Nasenspitze der Person (b). Der Grauwert an der Stelle der Nase unterscheidet sich stark von seinen benachbarten Pixeln und beherbergt daher ein Extrema	15
10	Das Histogramm und die Referenzrichtung im ersten Durchlauf (a) und das Koordinatensystem nach der Referenzrichtung und die entstandenen Histogramme(b).	16
11	Das Ergebnis des Feature-Matching mit ORB und OpenCV.	17
12	Die dreidimensionale Szene (links) und die zweidimensionale Sicht der virtuellen Kamera. Wird das Objekt perspektivisch gerendert, gibt es ein Raumvolumen (frustum). Es werden nur die Objekte gerendert, die sich innerhalb dieses Volumens befinden. So wird der rote Donut nicht und das blaue Objekt abgeschnitten dargestellt[1, Moeller (2019)].	19
13	Die Basis der GRP, die in weitere Sub-Pipelines eingeteilt und parallel ausgeführt werden können [1].	20
14	Die Primitiven werden in eine 2D-Ebene projiziert(a). Befindet sich ein Pixel innerhalb des Primitiven, werden diskrete Fragmente gebildet, die im in der Pixelverarbeitung weitergeschickt werden(b) [37, zuletzt aufgerufen am 11.07.2022].	23
15	Das Use-Case-Diagramm zur Anwendug.	25
16	Das Auswahl-Menü. Der Nutzer kann mit dem Burger-Icon das Menü jederzeit öffnen und wieder schließen. Mit den Pfeilen kann zwischen den vorhandenen Gebäuden geschaltet werden.	27

17	Ein Baum behinderte bei den Aufnahmen die Sicht auf die Wand, sodass diese mit dem Baum verschmolzen ist. . . . .	29
18	Der Decimate-Modifier reduziert die Anzahl an Polygonen. Ein Wert von 0.3 bedeutet, dass die Polygonanzahl 30% der ursprünglichen Polygonanzahl entspricht. . . . .	30
19	Kleine Polygone erschweren das Markieren von Kanten (a). Das entstandene Loch wird mit größeren Flächen gefüllt (b). . . . .	30
20	Das zusammengesetzte Gebäude. . . . .	31
21	Die Polygone werden markiert(a), unwrapped und anschließend auf eine geeignete Fläche auf der Textur gelegt(b). . . . .	31
22	Das Mannschaftsgebäude vor(a) und nach der Bearbeitung(b). . . . .	31
23	Die Beleuchtungseinschaften und das kombinierte Endergebnis des Environmental HDR Modus. . . . .	37
24	Der hit-testing Strahl vom Smartphone trifft auf eine virtuelle Fläche und gibt ein Hit Ergebnis zurück. . . . .	38
25	AR ohne Occlusion (l.) und mit Occlusion (r.). Die virtuelle Katze wird durch das Objekt verdeckt. . . . .	39
26	Die benötigten Packages im Package Manager. . . . .	41
27	Der Szenenaufbau für die Position auf beliebigen Flächen. . . . .	42
28	Die AR Session. . . . .	42
29	Die AR Session Origin mit dem AR Plane Manager und dem AR Raycast Manager. . . . .	43
30	Die AR Camera. Der Occlusion Manager aus der Depth API wird hier als Komponente eingefügt. . . . .	43
31	Der Pfeil auf dem Boden signalisiert, dass auf dieser Fläche Objekte platziert werden können. . . . .	45
32	Die Tag-Auswahl im Unity Inspector. . . . .	47
33	Das lokale Koordinatensystem des SABA Gebäudes zeigt mit der z-Richtung (blauer Pfeil) nach Norden(a). Um den Winkel $\alpha$ wird das Gebäude gedreht, damit die Rotation des Gebäudes mit den realen Gebäude übereinstimmt(b). . . . .	50
34	Die Depth Map aus einer Entfernung von 1,5m (a) im Vergleich zu der Depth Map aus 1m (b) und die Szene selbst mit der Mauer und dem Zaun aus 1,5m Entfernung(c). . . . .	55

## **Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben.

Villingen-Schwenningen, 31.08.2022

---

Oliver Kusch

## Literatur

- [1] Tomas Akenine-Moeller, Eric Haines und Naty Hoffman. *Real-Time Rendering*. A K Peters/CRC Press, Jan. 2019. ISBN: 9781315365459. DOI: 10.1201/9781315365459.
- [2] Apple. *ARKit*. 2022. URL: <https://developer.apple.com/augmented-reality/> (besucht am 09.08.2022).
- [3] *ARCore - Fundamental concepts*. 2022. URL: <https://developers.google.com/ar/develop/> (besucht am 02.08.2022).
- [4] *ARToolkit X*. 2022. URL: <http://www.artoolkitx.org/> (besucht am 09.08.2022).
- [5] Ronald T. Azuma. „A Survey of Augmented Reality“. In: *Presence: Teleoperators and Virtual Environments* 6 (4 Aug. 1997), S. 355–385. ISSN: 1054-7460. DOI: 10.1162/pres.1997.6.4.355.
- [6] T. Bailey und H. Durrant-Whyte. „Simultaneous localization and mapping (SLAM): part II“. In: *IEEE Robotics and Automation Magazine* 13 (3 Sep. 2006), S. 108–117. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1678144.
- [7] Herbert Bay u. a. „Speeded-Up Robust Features (SURF)“. In: *Computer Vision and Image Understanding* 110 (3 Juni 2008), S. 346–359. ISSN: 10773142. DOI: <https://doi.org/10.1016/j.cviu.2007.09.014>.
- [8] Mark Billinghurst, Adrian Clark und Gun Lee. „A Survey of Augmented Reality“. In: *Foundations and Trends® in Human–Computer Interaction* 8 (2-3 2015), S. 73–272. ISSN: 1551-3955. DOI: 10.1561/1100000049.
- [9] David E. Breen, Eric Rose und Ross T. Whitaker. „Interactive Occlusion and Collision of Real and Virtual Objects in Augmented Reality“. In: *European Computer-Industry Research Centre GmbH (Forschungszentrum)* (1995).
- [10] Satyan Coorg und Seth Teller. „Real-time occlusion culling for models with large occluders“. In: ACM Press, 1997, 83–ff. ISBN: 0897918843. DOI: 10.1145/253284.253312.
- [11] *Die beliebtesten Programmiersprachen weltweit laut PYPL-Index im August 2022*. 2022. URL: <https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index/> (besucht am 09.08.2022).
- [12] H. Durrant-Whyte und T. Bailey. „Simultaneous localization and mapping: part I“. In: *IEEE Robotics and Automation Magazine* 13 (2 Juni 2006), S. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022. URL: <https://ieeexplore.ieee.org/document/1638022/>.
- [13] Ralf' Dörner u. a. *Virtual und Augmented Reality (VR/AR)*. Bd. 2. Springer Berlin Heidelberg, 2019. ISBN: 978-3-662-58860-4. DOI: 10.1007/978-3-662-58861-1.

- [14] NERURKAR ESHA, LYNEN SIMON und ZHAO SHENG. „SYSTEM AND METHOD FOR CONCURRENT ODOMETRY AND MAPPING“. Pat. 2017. URL: <https://lens.org/137-618-961-961-779>.
- [15] John Flynn u. a. „DeepView: View Synthesis With Learned Gradient Descent“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [16] Google. *ARCore*. 2022. URL: <https://developers.google.com/ar/> (besucht am 09.08.2022).
- [17] Robert Green. „Spherical Harmonic Lighting: The Gritty Details“. In: 2003.
- [18] Chris Harris und Mike Stephens. *A combined Corner and Edge Detector*. 1988. DOI: 10.1.1.434.4816.
- [19] Richard Hartley und Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Bd. 2. Cambridge University Press, März 2004. ISBN: 9780521540513. DOI: 10.1017/CBO9780511811685.
- [20] Jan Herling und Wolfgang Broll. *Markerless Tracking for Augmented Reality*. Springer New York, 2011, S. 255–272. DOI: 10.1007/978-1-4614-0064-6\_11.
- [21] John Isidoro, Alex Vlachos und Chirs Brennan. „Rendering ocean water“. In: *Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks, Wordware* (2002).
- [22] H. Kato und M. Billinghurst. „Marker tracking and HMD calibration for a video-based augmented reality conferencing system“. In: IEEE Comput. Soc, 1999, S. 85–94. ISBN: 0-7695-0359-4. DOI: 10.1109/IWAR.1999.803809.
- [23] Georg Klein und David Murray. „Parallel Tracking and Mapping for Small AR Workspaces“. In: IEEE, Nov. 2007, S. 1–10. ISBN: 978-1-4244-1749-0. DOI: 10.1109/ISMAR.2007.4538852.
- [24] Georg Klein und David Murray. „Parallel Tracking and Mapping on a camera phone“. In: IEEE, Okt. 2009, S. 83–86. ISBN: 978-1-4244-5390-0. DOI: 10.1109/ISMAR.2009.5336495.
- [25] Oliver Kusch u. a. *Kreation von 3D Modellen der alten Kaserne in Villingen-Schwenningen - Eine Analyse der Arbeitspipeline zur Erstellung von 3D Modellen mittels Pix4d*. Hochschule Furtwangen, 2021.
- [26] Chloe LeGendre u. a. „DeepLight: Learning Illumination for Unconstrained Mobile Mixed Reality“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
- [27] Weiquan Liu u. a. „Learning to Match 2D Images and 3D LiDAR Point Clouds for Outdoor Augmented Reality“. In: IEEE, März 2020, S. 654–655. ISBN: 978-1-7281-6532-5. DOI: 10.1109/VRW50115.2020.00178.

- [28] David G. Lowe. „Object recognition from local scale-invariant features“. In: IEEE, 1999, 1150–1157 vol.2. ISBN: 0-7695-0164-8. DOI: 10.1109/ICCV.1999.790410.
- [29] Paul Milgram u. a. „Augmented reality: a class of displays on the reality-virtuality continuum“. In: Hrsg. von Hari Das. Dez. 1995, S. 282–292. DOI: 10.1117/12.197321.
- [30] Jason L. Mitchell. „Real-Time Synthesis and Rendering of Ocean Water“. In: *ATI Research Technical Report* (2005), S. 121–126.
- [31] Alfred Nischwitz u. a. *Computergrafik und Bildverarbeitung*. Vieweg+Teubner Verlag, 2012. ISBN: 978-3-8348-1304-6. DOI: 10.1007/978-3-8348-8323-0.
- [32] Lukas Platinsky u. a. „Collaborative Augmented Reality on Smartphones via Life-long City-scale Maps“. In: (Nov. 2020). URL: <http://arxiv.org/abs/2011.05370>.
- [33] Andreas Reich u. a. *Photogrammetric recordings of the SABA area*. Hochschule Furtwangen University, 2020.
- [34] E. Rosten und T. Drummond. „Fusing points and lines for high performance tracking“. In: IEEE, 2005, 1508–1515 Vol. 2. ISBN: 0-7695-2334-X. DOI: 10.1109/ICCV.2005.104.
- [35] Ethan Rublee u. a. „ORB: An efficient alternative to SIFT or SURF“. In: IEEE, Nov. 2011, S. 2564–2571. ISBN: 978-1-4577-1102-2. DOI: 10.1109/ICCV.2011.6126544.
- [36] Andre Schaefer u. a. *NISABA: Photogrammetrische Erfassung historischer Gebäudekomplexe in Villingen-Schwenningen*. Hochschule Furtwangen University, 2021.
- [37] Scratchapixel. *Rasterization: a Practical Implementation*. 2022. URL: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm>.
- [38] Manisah Mohd Shah, Haslina Arshad und Riza Sulaiman. „Occlusion in augmented reality“. In: *2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012)*. Bd. 2. 2012, S. 372–378.
- [39] Maximilian Speicher, Brian D. Hall und Michael Nebeling. „What is Mixed Reality?“ In: ACM, Mai 2019, S. 1–15. ISBN: 9781450359702. DOI: 10.1145/3290605.3300767.
- [40] Richard Szeliski. *Computer Vision*. Springer International Publishing, 2022. ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9.
- [41] *Unity Dokumentation, About AR Foundation*. 2021. URL: <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html> (besucht am 27.07.2022).
- [42] *Vuforia Engine*. 2022. URL: <https://www.ptc.com/en/products/vuforia/vuforia-engine> (besucht am 09.08.2022).

- [43] Edmund Weitz. *SIFT - Scale-Invariant Feature Transform*. 2022. URL: <http://weitz.de/sift/>.
- [44] Wikitude. 2022. URL: <https://www.wikitude.com/> (besucht am 09.08.2022).