



# CARL VON OSSIEZKY UNIVERSITÄT OLDENBURG

INFORMATIK  
BACHELORARBEIT

---

## Entwicklung einer Augmented Reality Anwendung zum Tracken und Erstellen von Markern für den Bildungsbereich

---

*Autor:*  
Johannes Scheibe

*Erstgutachter:*  
Prof. Dr.-Ing. Jürgen Sauer

*Zweitgutachter:*  
M. Sc. B. Eng. Nils Hartmann

Abteilung Systemanalyse und -optimierung  
Department für Informatik

Oldenburg, 23. September 2020

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>iv</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>Abkürzungsverzeichnis</b>	<b>vi</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Ziele der Arbeit . . . . .	1
1.3. Aufbau der Arbeit . . . . .	1
<b>2. Grundlagen</b>	<b>2</b>
2.1. Augmented Reality . . . . .	2
2.1.1. Einsatzbereiche . . . . .	3
2.1.2. Interfaces . . . . .	4
2.1.3. Technische Grundlagen . . . . .	4
2.2. Android Entwicklung . . . . .	7
2.2.1. Architektur . . . . .	7
2.2.2. Grundlegende Konzepte . . . . .	8
2.2.3. Fragments . . . . .	9
2.3. OpenGL . . . . .	10
2.3.1. Grundlagen von OpenGL . . . . .	10
2.3.2. Rendering-Pipeline . . . . .	10
2.3.3. OpenGL ES . . . . .	12
<b>3. Verwandte Arbeiten</b>	<b>13</b>
<b>4. Anforderungsanalyse</b>	<b>14</b>
4.1. Vision . . . . .	14
4.1.1. Mögliche Anwendungsfälle . . . . .	14
4.2. Der Prototyp . . . . .	15
4.3. Anforderungsanalyse . . . . .	15
4.3.1. Beschreibung der Systemumgebung . . . . .	16
4.3.2. Anwendungsfälle des Prototyps . . . . .	16

4.3.3. Anforderungsliste . . . . .	17
<b>5. Entwurf</b>	<b>20</b>
5.1. Anwendungsart . . . . .	20
5.1.1. Betriebssystem . . . . .	20
5.1.2. Entwicklungsumgebung . . . . .	21
5.2. Trackingverfahren . . . . .	21
5.2.1. Tracking Bibliothek . . . . .	21
5.3. Rendering . . . . .	22
5.3.1. Modellformat . . . . .	22
5.4. Userinterface . . . . .	22
5.5. Datenspeicherung . . . . .	23
<b>6. Implementierung</b>	<b>24</b>
6.1. Laden der Modelldateien . . . . .	24
6.1.1. ObjLoader . . . . .	24
6.1.2. TextureLoader . . . . .	25
6.1.3. Model . . . . .	25
6.2. Augmented Reality . . . . .	26
6.2.1. CameraFragment . . . . .	26
6.2.2. EducationARRenderer . . . . .	26
6.3. Markergenerierung . . . . .	26
6.3.1. MarkerGenerator . . . . .	27
<b>7. Evaluation</b>	<b>28</b>
7.1. Tests . . . . .	28
7.1.1. Testdurchführung . . . . .	28
7.1.2. Testfälle . . . . .	29
7.1.3. Zusammenfassung . . . . .	31
<b>8. Fazit</b>	<b>32</b>
8.1. Fazit . . . . .	32
8.2. Ausblick . . . . .	32
<b>Literaturverzeichnis</b>	<b>33</b>
<b>A. Beispielanhang</b>	<b>36</b>

# **Abbildungsverzeichnis**

2.1. RV Kontinuum . . . . .	2
2.2. Snapchat AR . . . . .	3
2.3. Barcode Marker . . . . .	6
2.4. Android Architektur . . . . .	7
2.5. Lebenszyklus einer Aktivität . . . . .	9
2.6. OpenGL Rendering Pipeline . . . . .	10
2.7. OpenGL Triangle . . . . .	11
4.1. Use Cases des Prototyps . . . . .	17
5.1. UI Mockup . . . . .	23
6.1. Obj-Dateiformat . . . . .	25
6.2. Markererstellung . . . . .	27
7.1. Testaufbau . . . . .	29
7.2. Perspektiven eines Markers . . . . .	30
7.3. Skalierungen eines Markers . . . . .	30
7.4. Rotationen eines Markers . . . . .	30
7.5. Belichtungen eines Markers . . . . .	31

# **Tabellenverzeichnis**

# **Abkürzungsverzeichnis**

**AR** Augmented Reality

**VR** Virtual Reality

**API** Application Programming Interface

**ART** Android Runtime

**IDE** Integrated Development Environment

**IDE** Identifikator

# **Kapitel 1**

## **Einleitung**

**1.1. Motivation**

**1.2. Ziele der Arbeit**

**1.3. Aufbau der Arbeit**

# Kapitel 2

## Grundlagen

Dieses Kapitel behandelt die Grundlagen, die für die Realisierung des Prototyps notwendig sind. Dabei werden Methoden und Eigenschaften der Augmented Reality erläutert, sowie ein Überblick über OpenGL gegeben.

### 2.1. Augmented Reality

In der Literatur lassen sich für den Begriff der Augmented Reality (AR, deutsch: „angereicherte Realität“) viele unterschiedliche Definitionen finden, die meisten stützen sich dabei auf das von Milgram u. a. (1994) definierte Reality-Virtuality (RV) Kontinuum, welches in Abbildung 2.1 dargestellt ist.

Um dieses zu verstehen muss zunächst der Begriff der Virtual Reality (VR, deutsch: „virtuelle Realität“) definiert werden. Nach Klein (2006, S.1) beschreibt die VR eine völlig künstliche, computergenerierte Welt, in die der Nutzer eintauchen kann.

Milgrams Definition fasst nun die Virtual Reality und die reale Welt als zwei, sich gegenüberliegende Enden eines Kontinuums auf. Dabei ist die reale Welt an die physikalischen Gesetze gebunden, während die virtuelle Welt diese überschreiten und sich von ihnen lösen kann (Milgram u. a., 1994, S. 283). Nach dem RV Kontinuum bewegt sich die Augmented Reality zwischen beiden Welten und stellt ein Kombination beider dar. Eine weitere Definition, die sich auch mit der von Milgram

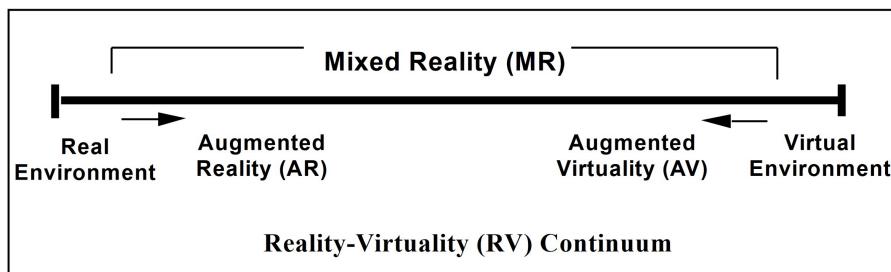


Abbildung 2.1.: Das Reality-Virtuality (RV) Kontinuum. (Quelle: Milgram u. a. (1994, S. 283))

vereinbaren lässt, beschreibt die Augmented Reality als eine computergestützte Erweiterung der wahrnehmbaren Realität um virtuelle Objekte (Kind u. a., 2019, S. 9). Auf dieser Grundlage kann man Augmented Reality als das Einbinden und Visualisieren digitaler, computergenerierte Objekte in der realen Welt auffassen. Oftmals wird dabei das Ziel verfolgt eine möglichst realistische Illusion für den Nutzer zu schaffen.

### 2.1.1. Einsatzbereiche

Die erste Assoziation, die die meisten mit dem Begriff Augmented Reality verbinden ist vermutlich der Unterhaltungsbereich. Große Firmen wie zum Beispiel Snapchat nutzen die Technologie um kleine Gimmicks für ihre Nutzer bereitzustellen (siehe Abbildung 2.2) Die meisten Personen, die den Begriff Augmented Reality hören, werden vermutlich an ein lustiges Gimmick zur Unterhaltung denken. Doch auch neben dem Bereich der Unterhaltung wird AR an vielen weiteren Stellen eingesetzt. Beispielsweise zu nennen wären hier der Bereich der Produktion, in welchem AR unter Anderem als Hilfsmittel zum Prototyping (Kind u. a., 2019, S. 44) genutzt werden kann, oder der Bereich der Medizin. Im letzteren können mittels AR Therapiemaßnahmen für psychische Erkrankungen oder Assistenzsysteme zur Diagnose und Operation entwickelt werden (Kind u. a., 2019, S. 52, 54).

Der Einsatzbereich in dessen Rahmen sich diese Arbeit bewegt ist jedoch der Bereich der Bildung.

Hier bietet Augmented Reality die Möglichkeit eines neuen Informationsmediums, welches vor allem zur Betrachtung dreidimensionaler Objekte genutzt werden kann. Dieses ermöglicht ein verbessertes, räumliches Verständnis des Lerninhaltes. Laut einer systematischen Analyse der Universität Stockholm, in welchem basierend auf der Anzahl der wissenschaftlichen Veröffentlichungen Schlüsse für das Lernen mit AR gezogen wurden, sind vor allem die Naturwissenschaften relevant für den Einsatz von AR (Hedberg u. a., 2018, S. 81).



Abbildung 2.2.: AR Gimmick aus der Anwendung Snapchat. (Quelle: Screenshot aus der Anwendung Snapchat, 01.08.2020)

## 2.1.2. Interfaces

### Head-Mounted-Displays

## 2.1.3. Technische Grundlagen

Zur Umsetzung der Augmented Reality ist eine Umgebungserfassung und -analyse des Systems mit Hilfe einer Tracking Software (auch Tracker genannt) notwendig. Auf Grundlage dieser können dann im Anschluss computergenerierte virtuelle Objekte in die Umgebung eingefügt werden.

Zur Analyse der Systemumgebung können verschiedene Eingabesysteme genutzt werden, mit deren Hilfe die Eigenschaften der Umgebung und der in ihr vorhandenen Objekte wahrgenommen werden können (Kind u. a., 2019, S. 22). Zu diesen Eigenschaften zählen neben statischen, wie der Größen und Position, auch dynamische Eigenschaften, welche die Veränderung der statischen Attribute einzelner Objekte umfassen. Beispielhafte Technologien, die zur Erfassung genutzt werden können, wären Kamerasysteme, Laser, Infrarot oder sonstige Sensoren (Kind u. a., 2019, S. 22). Neben der Umgebungserfassung ist auch die Verfolgung einzelner, in der Umgebung enthaltender Objekte notwendiger Bestandteil der Tracking Software, um die dynamischen Eigenschaften der realen Umgebung auf die virtuellen Objekte zu übertragen.

Umformulieren,  
wegen Zitat

Eine wichtige Rolle beim Tracking spielt die Genauigkeit, mit der die Eigenschaften der Umgebung wahrgenommen werden, sie bestimmt wie akkurat virtuelle Objekte in die reale Umgebung eingefügt werden können und wie realistisch die Illusion erscheint (Klein, 2006, S. 2).

### Trackingverfahren

Grundsätzlich kann bei der Tracking Software zwischen zwei Verfahren unterschieden werden, dem nicht visuellen und dem visuellen Tracking (Mehler-Bicher und Steiger, 2014, S. 26). Während bei letzterem auf Daten von zum Beispiel einer Kamera zurückgegriffen wird, beruht das nichtvisuelle Tracking auf Sensoren, die direkten Zugriff auf die Eigenschaften der Umgebung, wie der Position, liefern.

Bei dem für diese Arbeit relevantem visuellen Tracking, müssen die Eigenschaften der Umgebung aus dem Kamerabild abgeleitet und verarbeitet werden. Dazu werden in der Regel nach Mehler-Bicher und Steiger (2014, S. 26) zwei Schritte benötigt:

1. Die Initialisierung, bei welcher nach einem bestimmten Muster im Kamerabild gesucht wird. Dieses kann dabei im Vorfeld definiert sein oder aus dem Kamerabild abgeleitet werden.
2. Die Verfolgung bzw. Antizipation der möglichen Bewegung, bei welcher das

gefundene Muster in den einzelnen, aufeinanderfolgenden Videoframes verfolgt wird und eine Prognose der zukünftigen Position des Muster berechnet wird, um den Rechenaufwand zu verkleinern.

Des Weiteren kann beim Visuellen Tracking zwischen Marker Based Feature Tracking und Natural Feature Tracking unterschieden werden.

**Marker Based Feature Tracking** verwendet feste, im Vorfeld definierte Muster, die in der Umgebung platziert werden. Diese Muster werden als Marker bezeichnet. Meistens handelt es sich dabei um schwarze Quadrate, in deren Mitte eine ID als ein Muster aus schwarzen und weißen Vierecken codiert ist. Ein beispielhafter Marker ist in Abbildung 2.3 zu sehen. Die Marker sind dabei optisch so angepasst, dass sie sehr einfach von einem Tracker erfasst werden können, um die Erkennungsgeschwindigkeit und somit die Performanz des gesamten Systems zu optimieren (Mehler-Bicher und Steiger, 2014, S. 28).

Owen u. a. stellen dazu folgende Anforderungen an einen optimalen Marker:

- Ein idealer Marker sollte die eindeutige Bestimmung seiner Position und Orientierung im Verhältnis zur Kamera unterstützen.
- Der Marker sollte alle Ausrichtungen unterstützen.
- Der Marker sollte Teil einer Reihe an Markern sein, die sich eindeutig von einander unterscheiden lassen.
- Ein Marker muss einfach lokalisierbar und identifizierbar sein.
- Die Marker müssen über einen weiten Aufnahmebereich funktionieren

(Nach Owen u. a. (2002, S. 2))

Durch diese Eigenschaften ist die Erkennung von einem Marker relativ simpel und funktioniert im Allgemeinen in drei Schritten:

1. Zunächst werden müssen die Kanten aus dem Bild extrahiert werden, um mit deren Hilfe alle Vierecke aus dem Bild zu filtern.
2. Dann kann mittels Template Matching oder ähnlichen Verfahren der Marker erkannt werden und die ID, falls vorhanden, dekodiert werden.
3. Im letzten Schritt muss dann noch die Position, Größe und Orientierung des Markers berechnet werden.

(In Anlehnung an Ćuković u. a. (2015))

link ins Literaturverzeichnis?



Abbildung 2.3.: Ein Barcode Marker für ARToolKit mit der ID 10.  
(Quelle: Generiert mit dem Marker Generator <http://au.gmented.com/app/marker/marker.php>)

**Natural Feature Tracking** greift hingegen auf sogenannte Keypoints zurück, die natürlich im Bild enthalten sind. Keypoints beschreiben dabei markante Bildpunkte, die sich durch zum Beispiel starke Helligkeitsveränderungen in der direkten Nachbarschaft auszeichnen, wie es unter anderem bei Kanten oder Ecken der Fall ist. Zur Extraktion dieser Keypoints gibt es verschiedene Verfahren wie den SIFT-Algorithmus (Scale-Invariant-Feature-Tracking), welcher die Grundlage für viele weitere aktuelle Verfahren bildet und Merkmale extrahiert, die invariant gegenüber der Rotation, Translation, Skalierung und partiell invariant gegenüber Helligkeitsveränderungen sind (Nischwitz u. a., 2011, S. 345). Diese Invarianzen sind dabei auch für den Bereich der Augmented Reality sehr wichtig, da die virtuellen Objekte auch bei einer sich verändernden Umgebung korrekt eingebunden werden sollen.

Bei diesem Verfahren müssen in einem ersten Schritt zunächst die Keypoints für den Bildbereich, in dem das Objekt platziert werden soll, extrahiert und gespeichert werden. Diese Bildpunkte können dann mit Hilfe von Matchingverfahren in den folgenden Frames gefunden werden. Bei einer Übereinstimmung muss dann folglich noch die Position, Größe und Orientierung des Bildbereiches berechnet werden.

Eine Verallgemeinerung des gesamten Tracking Prozesses kann dabei wie folgt beschrieben werden:

Nachdem das System einen Frame von der Kamera bekommt wird dieser meist vorverarbeitet, um die wichtigsten Informationen zu extrahieren und den Tracking Prozess zu beschleunigen (Ćuković u. a., 2015, S. 5). Eine Möglichkeit ist dabei das Bild in ein Graustufenbild umzuwandeln. Dadurch lassen sich Helligkeitsveränderungen leichter detektieren und die Anzahl der Farbkanäle wird von drei oder mehr auf einen reduziert, wodurch die Geschwindigkeit der Tracking Algorithmen deutlich erhöht werden kann.

Nach der Vorverarbeitung kann dann mittels Natural Feature Tracking oder Marker Based Feature Tracking die Position, Größe und Orientierung der gesuchten Fläche, beziehungsweise des Markers ermittelt werden und das virtuelle Objekt entsprechend transformiert werden.

## 2.2. Android Entwicklung

Android ist ein Open Source Software Plattform, die vor allem bei Smartphones und anderen mobilen Geräten zum Einsatz kommt und ein Produkt der von Google gegründeten Open Handset Alliance ist (Gargenta, 2011, S. 4). Bekannt ist vor allem das Betriebssystem Android, das auf vielen Geräten ihren Einstaz findet.

Als Programmiersprache zur Entwicklung von Android Anwendungen stehen Kotlin, Java und C++ zur Verfügung Android (a).

### 2.2.1. Architektur

Abbildung 2.4 zeigt eine Übersicht der Software Architektur von Android.

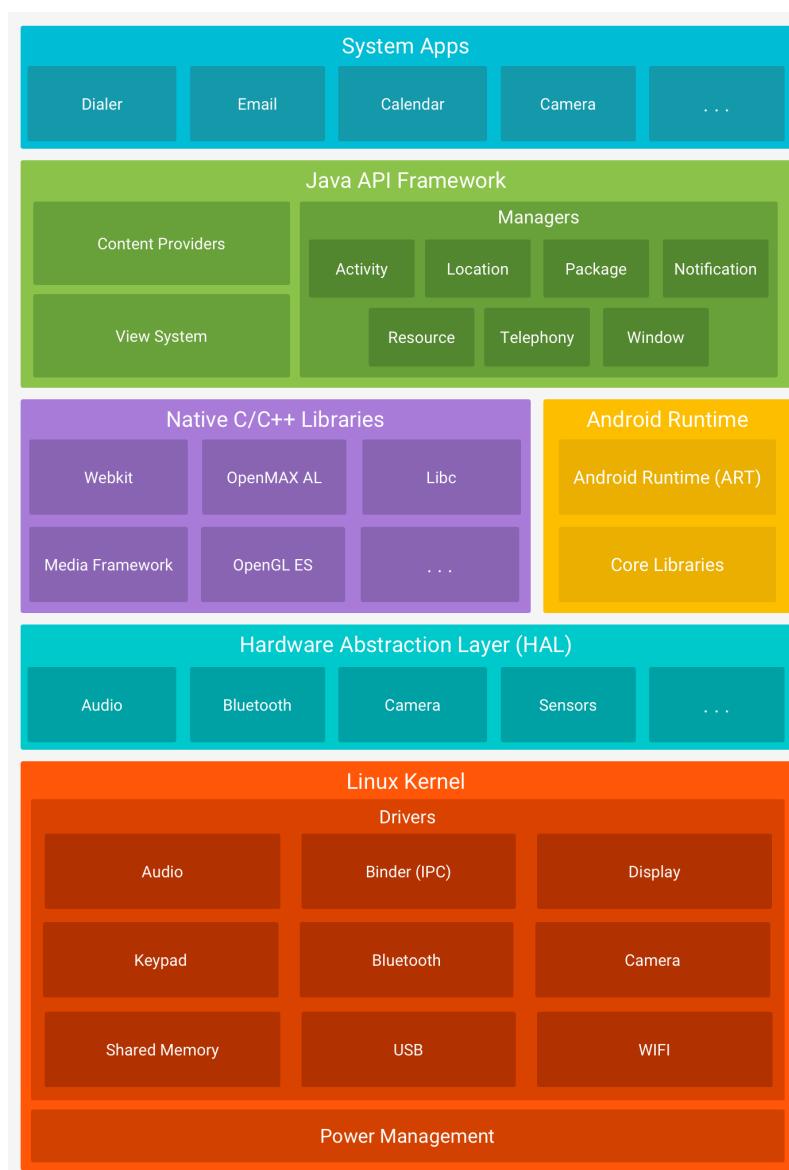


Abbildung 2.4.: Die Android Architektur. (Quelle: Android (c))

Die Grundlage eines Android Systems bildet dabei der Linux-Kernel, der es Android

unter anderem erlaubt auf die Sicherheitsfunktionen des Kernels zuzugreifen (Android, c).

Die Hardware Abstraktionschicht (HAL) stellt dem übergeordneten Java-API-Framework Standardschnittstellen zur Verfügung, über die eine Anwendung auf die Hardwarefunktionen des Gerätes zugreifen kann (Android, c). Übergeordnet finden sich die Laufzeitumgebung Android Runtime (ART) und nativen Bibliotheken auf. Bei letzteren handelt es sich um Bibliotheken, welche in C und C++ geschrieben sind (Android, c) und dem Entwickler unterschiedliche Funktionen zur Verfügung stellen. Die in C oder C++ geschriebenen Bibliotheken sind vor allem aus Geschwindigkeitsaspekten relevant.

Die Entwicklung von Android Anwendungen erfolgt dabei über das Java-API-Framework, welches dem Entwickler ein Grundgerüst sowie alle wichtigen Funktionen und Anwendungsbauusteine die für die Entwicklung relevant sind zur Verfügung stellt (Android, c).

Die oberste Schicht stellen die System Anwendungen dar, bei welchen es sich um Standard-Anwendungen handelt, die Android dem Nutzer zur Verfügung stellt.

### **2.2.2. Grundlegende Konzepte**

Alle Android Anwendungen basieren auf den selben Komponenten und Konzepten, die im folgenden einmal grundlegend betrachtet werden. Diese Komponenten sind als Klassen in Android verankert und können mittels Vererbung in Projekte eingebunden und angepasst werden

#### **Activities**

Activities (deutsch: „Aktivitäten“) stellen die Grundlage jeder Android Anwendung dar und werden durch die Klasse Activity implementiert. Dabei erzeugt jede Activity ein Fenster der Anwendung und ersetzt gleichzeitig die aus der Programmierung mit unter anderem Java bekannte main() Methode (Android, b). Anstelle einer main() Methode werden verschiedene callback Methoden aufgerufen, die einen bestimmten Zustand im Lebenszyklus einer Activity repräsentieren (Android, b). Der gesamte Lebenszyklus einer solchen Activity wird in Abbildung 2.5 dargestellt.

#### **Services**

Ein Service (deutsch: „Dienstleistung“) ist eine Komponente die im Hintergrund läuft und dazu konzipiert wurde langfristige Operationen auszuführen (Android, a). Services können dabei unabhängig jeglicher Aktivitäten im Hintergrund laufen, auch wenn die Anwendung geschlossen wurde (Murphy, 2009).

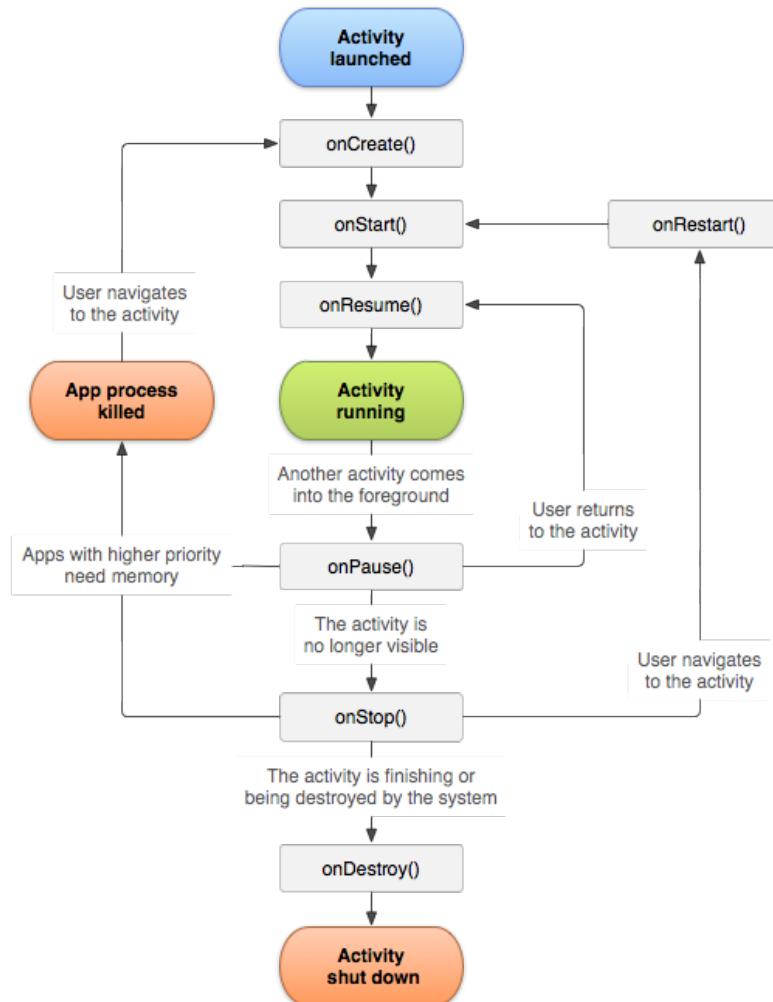


Abbildung 2.5.: Der Lebenszyklus einer Aktivität. (Quelle: Android (d))

## Content Providers

Content Providers (deutsch: „Inhaltsanbieter“) stellen ein Abstraktionslevel für Daten dar, auf die aus mehreren Anwendungen zugegriffen wird (Murphy, 2009). Mit ihrer Hilfe ist es möglich eigene Daten anderen Anwendungen zur Verfügung zu stellen (Murphy, 2009).

Dabei wird das Ziel verfolgt das Zusammenspiel zwischen verschiedenen Anwendungen zu verbessern, und den Datenaustausch zwischen ihnen zu verbessern.

## Intents

Bei einem Intent (deutsch: „Absicht“) handelt es sich um eine asynchrone Systemnachricht, die dazu genutzt werden kann bestimmte Komponenten oder bestimmte Komponentenarten zu aktivieren (Android, a).

### 2.2.3. Fragments

[schreiben](#)

## 2.3. OpenGL

OpenGL stellt die industriell meistgenutzte Programmierschnittselle (API) zur Entwicklung von 2D und 3D Anwendungen dar (Khronos Group, b). Mit Hilfe des Interfaces lassen sich komplexe dreidimensionale Objekte darstellen.

### 2.3.1. Grundlagen von OpenGL

Alle Objekte werden in OpenGL aus einem oder mehreren Polygons zusammengezetzt (Shreiner u. a., 2006, S. 5).

Dabei wird jedes Poligon durch eine Menge an Eckpunkten (Vertices) definiert und jeder Eckpunkt(Vertex) stellt dabei eine Menge an Daten für einen bestimmten Punkt in einem dreidimensionalen Koordinatensystem dar (de Vries, J.).

### 2.3.2. Rendering-Pipeline

Die folgende Abbildung 2.6 zeigt die Rendering-Pipeline von OpenGL. Alle Objekte und Modelle, die mit Hilfe von OpenGL dargestellt werden, durchlaufen diese Pipeline.

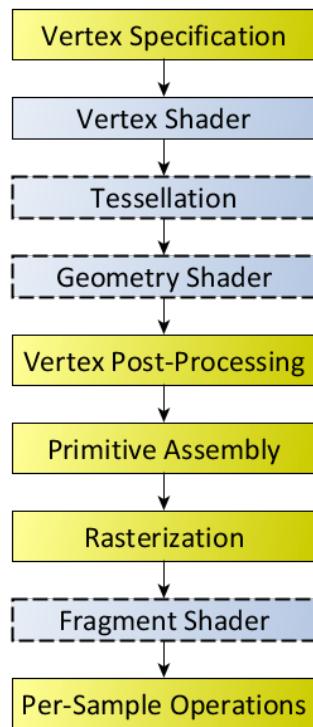


Abbildung 2.6.: Die Rendering-Pipeline von OpenGL. (Quelle: Khronos Group (c))

Die Objekte werden dabei durch eine Menge an Vertices definiert. Die zwei wichtigsten Eingriffspunkte in der Rendering-Pipeline sind der sec:vertex-shader und der sec:fragment-shader. Beide können von Programmierer implementiert werden und

dienen dem Zweck verschiedene Operationen auf den einzelnen Vertices beziehungsweise den Pixeln durchzuführen. Um beide nutzen zu können müssen sie mit einem Shader Programm verknüpft werden, welches den Output von jedem Shader mit dem Input des nächsten Shaders verbindet (de Vries, J.).

Im folgenden werden einmal die einzelnen Schritte der Pipeline genauer betrachtet. Bevor das Objekt dargestellt werden kann muss es zunächst definiert werden. Dieser Schritt wird als **Vertex Specification** bezeichnet. Dazu muss das Modell in viele einzelne Dreiecke unterteilt werden, welche zusammen das gesamte Objekt bilden. Diese einzelnen Flächen werden als Grundelemente (Primitives) bezeichnet (Khronos Group, c) und können durch jeweils drei Eckpunkte (Vertices) beschrieben werden (siehe Abbildung 2.7).

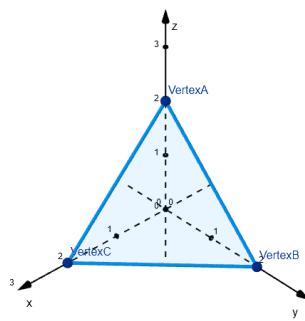


Abbildung 2.7.: Zusammensetzung eines einzelnen Dreiecks. (Quelle: Eigene Darstellung)

Dabei können jedem Vertex neben der Position, auch weitere Eigenschaften, welche für die Berechnungen in den nächsten Schritten der Rendering-Pipeline notwendig sind, wie Texturkoordinaten zugeordnet werden.

Tool angeben?

Alle Informationen werden in einer Listenstruktur gespeichert und werden im Anschluss an den **Vertex Shader** übergeben. Dieser verarbeitet jeden einzelnen Eckpunkt, indem er bestimmte Operationen auf diesem durchführt bevor er an den nächsten Schritt der Rendering Pipeline weitergegeben wird. (de Vries, J.). Dabei können einzelne Attribute des Punktes transformiert werden, während andere einfach nur weitergeleitet werden.

Die **Tessellation** stellt einen weiteren, optionalen Schritt in der Pipeline dar. Er arbeitet mit einer Menge an Vertices, die als Patch bezeichnet wird und erzeugt aus diesen kleinere Grundelemente (Khronos Group, d). Mit Hilfe dieser Zerkleinerung können Geometrien, wie zum Beispiel Rundungen verstärkt werden.

Der nächste Schritt in der Rendering Pipeline ist der **Geometry Shader**, welcher ebenfalls eine optionale Station darstellt und vor allem für das Layered Rendering, bei welchem ein Grundelement in mehreren Bildern gerendert wird, genutzt werden kann (Khronos Group, a). Im Anschluss folgt das **Vertex Post-Processing**,

welches aus einer Reihe fester Funktionen besteht (Khronos Group, c), die im Kontext dieser Arbeit nicht zu brachten sind. Der nächste Schritt ist das **Primitive Assembly** bei welchem aus der Reihe an Vertices eine geordnete Sequenz an Grundelementen geformt wird (Khronos Group, c). Während der **Rasterization** werden dann die Grundelemente in eine Pixeldarstellung für den entsprechenden Bildschirm umgewandelt (de Vries, J.).

Der **Fragment Shader** ordnet dann jedem Pixel eine Farbe zu, in deren Berechnung meist Werte wie Schatten, Licht und dessen Farbe einfließen (de Vries, J.). Zu letzt werden dann noch eine Reihe an **Per-Sample Operations** durchgeführt, welche Tests beschreiben die testen, ob ein Pixelwert aktualisiert werden muss oder nicht (Khronos Group, c).

### 2.3.3. OpenGL ES

# **Kapitel 3**

## **Verwandte Arbeiten**

# Kapitel 4

## Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an die Anwendung und den zu entwickelnden Prototypen erhoben. Dazu werden verschiedene Anwendungsfälle herbeigezogen und daraus Anforderungen abgeleitet.

### 4.1. Vision

Die Hauptzielgruppe der Anwendung ist der Bildungsbereich. Sie soll Schülern, Studieren und auch Professoren die Möglichkeit bieten mit Hilfe von Augmented Reality das Lernen sowie Lehren zu bereichern.

Das Grundprinzip ist dabei folgendes: Die App soll dem Anwender die Möglichkeit geben 3D-Modelle hochzuladen, für die dann ein einzigartiger Marker generiert wird. Dieser kann dann ausgedruckt oder anderweitig angezeigt werden. Mit Hilfe der Kamera lässt sich dann das entsprechende 3D-Modell im Augmented Reality Bereich darstellen.

#### 4.1.1. Mögliche Anwendungsfälle

Im folgenden werden mögliche Anwendungsfälle für zwei verschiedene Personas definiert.

##### Der Professor

Das Anwendungsszenario, welches für einen Professor oder Lehrer denkbar wäre, orientiert sich an der bereits existierenden Webseite Socrative, welche bereits im Universitätsalltag häufig eingesetzt wird und Lehrenden die Möglichkeit gibt Umfragen zu erstellen, die von Studierenden über einen Raumcode beantwortet werden können.

Ein solches Raumsystem wäre auch für die AR Anwendung denkbar. Mit Hilfe der Anwendung könnte dann ein Professor einen Raum erstellen, in welchem er verschiedene 3D-Modelle speichern könnte, die Marker könnten im Anschluss dann Online

ref socrative

zur Verfügung gestellt, über Ausdrucke vervielfältigt oder in die Präsentation eingebunden werden.

Die Studierenden können dann über einen Zugangscode dem Raum beitreten und dann mit Hilfe der Kamera die entsprechenden Modelle des Raumes, in dem sie sich aktuell befinden, anzeigen lassen.

### **Die Studierenden**

Den Studierenden soll neben der Möglichkeit öffentlichen Räumen beizutreten auch die Möglichkeit gegeben werden selbst Modelle hochzuladen und die entsprechenden Marker zum Beispiel auf Lernzettel zu drucken. Dazu könnte jeder einen privaten Raum besitzen. Die Modelle die der Anwender hier hochlädt, wären dann nur lokal gespeichert und nicht öffentlich über einen Code oder Ähnlichem zugänglich.

## **4.2. Der Prototyp**

Der zu entwickelnde Prototyp fokussiert sich dabei auf die Umsetzung der Augmented Reality und das Erstellen der zu trackenden Marker. Die Modelle, die hochladen werden, werden dabei lediglich lokal gespeichert.

Dadurch handelt es sich bei dem Prototyp um die Umsetzung des „privaten Raumes“. Er bietet dem Anwender nicht die Möglichkeit die Modelle zuteilen.

Eine ansonsten notwendige Datenspeicherung auf einem Server, das Implementieren eines Raumsystems mit Zugangscode und weitere Funktionen fallen dadurch bei diesem ersten Prototyp weg.

## **4.3. Anforderungsanalyse**

Im folgenden wird eine Anforderungsanalyse für den zu entwickelnden Prototyp durchgeführt. Dazu wurde das in der Softwaretechnik I Vorlesung vorgestellte Verfahren zur Anforderungsdefinition genutzt (Winter, 2018, Folie 209-214). Dieses Verfahren sieht den folgenden Aufbau vor:

1. Vision
2. Machbarkeitsstudie
3. Beschreibung der Systemumgebung
4. Anwendungsfälle
5. Anforderungsliste

## 6. Prototypen

## 7. Glossar

Im folgenden wird jedoch nur eine vereinfachte, an diese Arbeit angepasste Version dieses Verfahrens genutzt.

### **4.3.1. Beschreibung der Systemumgebung**

Bei dem Prototyp handelt es sich um eine mobile Anwendung. Das Endgerät für den Nutzer ist also ein Smartphone.

Der relevante Sensor, den das Smartphone für das Trackingverfahren bereitstellt, ist die Kamera. Die Anwendung muss die Videoframes analysieren und eine Ausgabe in Form von einem gerenderten 3D-Modell auf dem Display darstellen.

Insgesamt besitzt der Prototyp drei Unterfunktionen:

1. Das Generieren von Markern. Hierbei erstellt der Prototyp Marker für den Anwender, die alle eine unterschiedliche ID besitzen und aus jeder Rotation eindeutig zu erkennen sind. Dafür müssen sich zwei Marker auch unterscheiden, wenn einer von ihnen beliebig rotiert wurde.
2. Das Tracken von Markern. Bei dem die Anwendung einen Marker in einem Videoframe erkennt und seine Position und Transformation im Videoframe berechnet. Dabei liegt der Marker in gedruckter Form auf einem Blatt Papier vor oder wird auf einem Bildschirm angezeigt.
3. Das Rendern von Modellen. Das anzuzeigende Modell wurde dabei vom Nutzer als Datei hochgeladen und muss zunächst vom System verarbeitet werden. Um im Anschluss das Modell realistisch in Relation zum Marker zu platzieren, muss die beim Tracking berechnete Position und Transformation des Markers verwendet werden.

Da der Prototyp auf einem mobilen Endgerät genutzt wird, variiert die Umgebung in welcher dieser zum Einsatz kommt stark. Deshalb ist es notwendig, dass das System robust gegenüber verschiedenen Umwelteinflüssen ist.

### **4.3.2. Anwendungsfälle des Prototyps**

Ein Beispielhaftes Anwendungsszenario des Prototypen, das sich aus dem Anwendungsfall des Studierenden ableitet (siehe Abschnitt 4.1.1), ist die mit dem Prototyp

generierten Marker auf einem Lernzettel einzufügen und ein vorlesungsrelevantes Modell zu verlinken. Dazu wird das Modell in dem Prototyp hochgeladen und der generierte Marker in ein Dokument eingefügt. Im Anschluss kann dann das Dokument mit der Kamera gefilmt werden, um sich das 3D-Modell anzeigen zu lassen. Allgemein lassen sich die Funktionen und Anwendungsfälle in dem in Abbildung 4.1 gezeigtem Anwendungsfalldiagramm visualisieren.

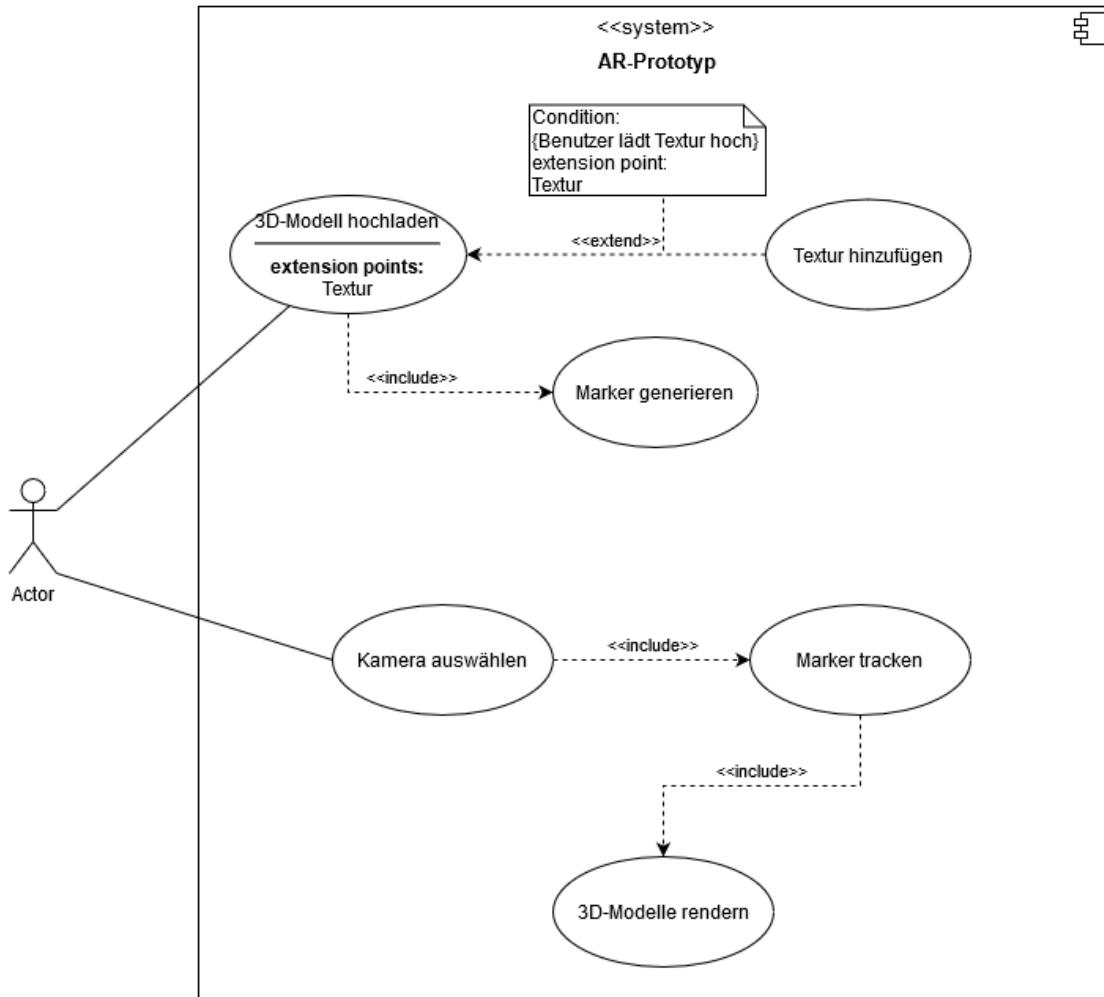


Abbildung 4.1.: Use Case Diagramm des Prototyps. (Quelle: Eigene Darstellung)

### 4.3.3. Anforderungsliste

Im folgenden Abschnitt werden die Anforderungen an den Prototyp definiert. Dabei wird zwischen funktionalen und nichtfunktionalen Anforderungen unterschieden.

**Funktionale Anforderungen** beschreiben die Funktionen eines Systemes, die notwendig sind, damit dieses seine Aufgaben erfüllen kann (Robertson und Robertson, 2012, S. 10).

**Nichtfunktionale Anforderungen** beschreiben Eigenschaften, die das System besitzen muss, um den Nutzer zufrieden zu stellen (Robertson und Robertson, 2012, S. 10). Diese Eigenschaften beziehen sich vor allem auf die Bereiche Service Level, Zugriffsbeschränkungen, Sicherheit, Monitoring, Kontrolle, Schnittstellen, Archivierung, Benutzerfreundlichkeit und Konversion (Böhm und Fuchs, 2002, S. 139)

Die folgende Anforderungsliste leitet sich aus den Anwendungsfällen des Prototyps ab.

### Funktionale Anforderungen

- FA1 Die Anwendung soll ein Set von mindestens 10 Markern generieren können.
- FA2 Die Anwendung soll die generierten Marker in einem Kamerabild erkennen können.
- FA3 Die Anwendung soll die generierten Marker unterscheiden können.
- FA4 Der Benutzer soll eigene 3D-Modelle als OBJ-Dateien hochladen können.
- FA5 Die Anwendung soll OBJ-Datei laden und verarbeiten können.
- FA6 Die Anwendung soll Textur-Dateien im Bildformat(jpeg, png) laden und verarbeiten können.
- FA5 Die Anwendung soll den hochgeladenen Modellen einen generierten Marker zuordnen.
- FA6 Die Anwendung soll die 3D Modelle im Kamerabild anzeigen können.
- FA7 Die Anwendung soll die 3D-Modelle basierend auf der Transformation und Position der Marker im Kamerabild rendern.

### Nichtfunktionale Anforderungen

- NF1 Die Anwendung soll auf einem Android Smartphone(Huawei P30 Pro) laufen.
  - NF1.1 Die Anwendung soll in Java mit Hilfe von Android Studio entwickelt werden.
- NF2 Das Tracking soll mittels eines markerbasierten Verfahrens realisiert werden.
- NF3 Das Tracking soll mit 30FPS laufen.
- NF4 Das Tracking soll bei vollständig erkanntem Marker robust gegenüber Rotation, Skalierung, Perspektive und Belichtung sein.

NF4 Die Anwendung soll ein simples User-Interface bereitstellen

NF4 Mit Hilfe der UI soll ein Wechsel zwischen Kamera und Einstellungen möglich sein.

NF4 Über die Einstellungsseite soll ein hochladen von 3D-Modellen möglich sein.

# **Kapitel 5**

## **Entwurf**

Dieses Kapitel beschäftigt sich mit dem Entwurf des Prototyps. Die Aufgabe des Entwurfes eines Softwaresystems ist es aus den gegebenen Anforderungen softwaretechnische Lösungen zu entwickeln (Balzert, 2011). Die Grundlage bilden dabei die in Kapitel 4 definierten Anforderungen.

### **5.1. Anwendungsart**

Die Anwendung soll Lernenden eine weitere Zugriffsmöglichkeit auf Wissensinhalte inform von Augmented Reality bieten. Wie bereits in den Anforderungen festgelegt, soll dafür die Anwendung auf einem mobilen Endgerät laufen.

Dieses bietet im Bezug auf AR mehrere Vorteile. Zum einen ist es wichtig, dass das Endgerät frei im Raum bewegbar ist, damit der Anwender gut mit den AR-Objekten interagieren kann. Dieses ist durch ein mobiles Endgerät gewährleistet. Eine Alternative dazu wären sogennante Head-Mounted-Displays (siehe Kapitel 2.1.2). Diese besitzen aber im Vergleich eine deutlich geringere Verfügbarkeit. Ein Handy oder Tablet besitzt dagegen heutzutage fast jeder Student und hat dieses auch so gut wie immer parat. Dadurch ermöglicht ein Smartphone zusätzlich einen schnellen, ortsunabhängigen Zugriff.

Ein weiteren Vorteil den ein mobiles Endgerät mit sich bringt ist die Kamera verfügt, welche für die meisten Trackingverfahren essentiell ist.

#### **5.1.1. Betriebssystem**

Für mobile Anwendungen gibt es vor allem zwei Plattformen, die für diese Arbeit in Frage kommen. Auf der einen Seite steht Apples IOS und auf der Anderen Android von Google. Letztendlich gibt es im Bezug auf die AR-Anwendung nicht viele Argumente, die für eine bestimmte Seite sprechen. Letztendlich wurde Android als Zielsystem gewählt, da bereits ein Endgerät, welches dieses Betriebssystem nutzt,

vorliegt. Dadurch kann die entwickelte Software direkt auf dem Gerät getestet werden.

### 5.1.2. Entwicklungsumgebung

Als Entwicklungsumgebung (IDE) wurde Android Studio festgelegt, welches die offizielle Entwicklungsumgebung für Android Anwendungen darstellt.

## 5.2. Trackingverfahren

Für Augmented Reality kommen vor allem visuelle Tracking Verfahren, bei denen das Kamerabild als Eingabe für den Tracker genutzt wird, in Frage. Dabei gibt es jedoch nochmal die Unterscheidung zwischen Natural Feature Tracking und Markerbased Feature Tracking (vgl. Kapitel 2.1.3).

Für den konkreten Anwendungsfall dieser Arbeit ist das markerbasierte Verfahren am besten geeignet. Es ermöglicht, dass Speichern von Informationen über den Identifikator (ID) der einzelnen Marker. Dadurch kann man jedem individuellem Marker ein AR-Objekt zu ordnen. Diese Zuordnung ist mit Hilfe von natürlichen Bildpunkten nicht direkt möglich, da bei diesem Verfahren lediglich ein Objekt im Bild positioniert wird und anhand markanter Bildpunkte der Umgebung ausgerichtet und transformiert wird.

Ein weiterer Vorteil den die Marker bieten ist dass sie sich einfach in verschiedene Arten von Dokumenten einbinden lassen und dann im Anschluss lediglich mit Hilfe einer Kamera eingescannt werden müssen.

### 5.2.1. Tracking Bibliothek

Mittlerweile gibt es viele Bibliotheken, die Methoden und Klassen zur Umsetzung von Augmented Reality bereitstellen und die Tracking Verfahren müssen nicht selbst erstellt werden. Jedoch unterstützen nicht alle von Ihnen auch das Markerbasierte Verfahren. Eines der vermutlichen bekanntesten Bibliotheken, die die Anforderungen erfüllen, ist ARToolKit. Es stellt umfangreiche Klassen zur Implementierung von Augmented Reality zur Verfügung und unterstützt dabei die verschiedensten Markervarianten von simplen Hiro Markern bis hin zu Barcode Markern mit verschiedenen IDs. Im Rahmen dieser Arbeit wurde die Version ARToolKitX genutzt.

Weitere mögliche Bibliotheken wären Googles ARCore, welches jedoch keine explizite Marker Unterstützung besitzt, oder OpenCV, welches viele Methoden zum Tracken bereitstellt, jedoch im Vergleich zu ARToolKit einen deutlich aufwendigeren

Implementierungsprozess benötigt. ARToolKitb bietet zusätzlich den Vorteil, dass es bereits an die Konzepte von Android angepasst wurde und sich somit gut in die bestehende oder neue Android Anwendungen einfügen lässt.

## 5.3. Rendering

ARToolKit arbeitet mit OpenGL ES als API zum darstellen von AR-Objekten im Kamerabild. Demzufolge wird auch die Anwendung auf OpenGL ES aufbauen.

### 5.3.1. Modellformat

Da es dem Nutzer mit der Anwendung möglich sein soll eigene Modelle hochzuladen, muss die Software die entsprechenden Dateien einlesen und verarbeiten können. Dazu wurde im Entwurfsprozess festgelegt, dass die Modell-Datei für den ersten Prototyp im OBJ-Format hochgeladen wird und die Textur als einzelne JPG-Datei hinzugefügt werden kann. Die Entscheidung beruht darauf, dass dieses Format von den meisten Grafikprogrammen unterstützt wird. So ist es auch mit dem kostenlosen Tool Blender, welches zum Erstellen der Testmodelle genutzt wurde, möglich die Objekte im OBJ-Format zu exportieren.

Für den ersten Prototyp wurde die Entscheidung getroffen, die MTL-Datei, welche Materialinformationen des Objektes speichert nicht zu betrachten, da sie zunächst nicht essentiell für die Darstellung des Objektes ist und in großen Teilen durch eine Textur in Form einer Bilddatei ersetzt werden kann.

Eine weitere Anforderung an das Modell ist, dass die sogenannten Faces, welche die einzelnen Flächen (Polygons), trianguliert werden. Dieses ist in den meisten Grafikprogrammen ohne Probleme vor dem Export möglich. Auch hier könnte in späteren Versionen des Prototyps über eine erweiterte Unterstützung nachgedacht werden.

## 5.4. Userinterface

Der Prototyp soll ein simples Userface besitzen, welches dem Nutzer grundlegende Interaktionsmöglichkeiten bietet. Dazu wurde das in Abbildung 5.1 gezeigte Mockup entwickelt. Es zeigt die drei Ansichten (Fragments) der Anwendung, welche von der Navigation überlagert werden.

Die Navigation besteht aus zwei Teilen am oberen Bildschirmrand ist der Titel des aktuellen Fragments zusehen und am unteren Rand befindet sich die eigentliche Navigation. Diese besteht aus drei repräsentativen Icons, welche jeweils ein Fragment

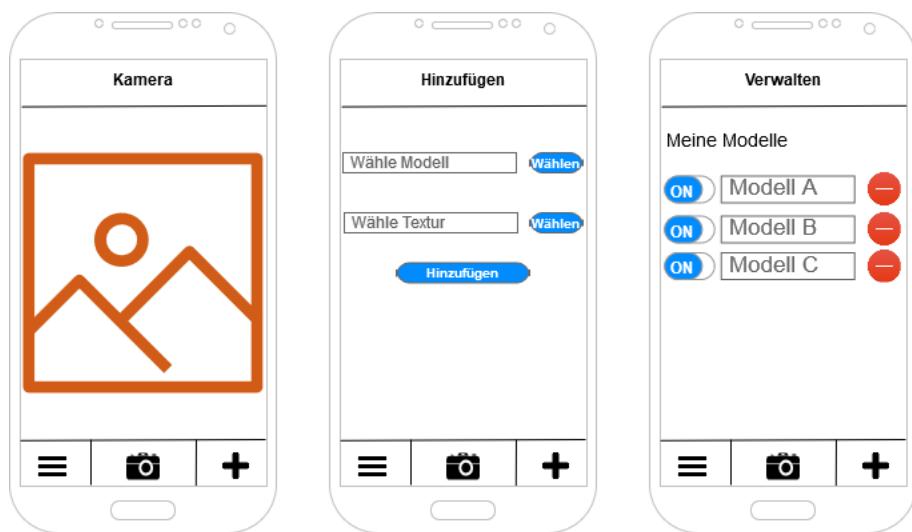


Abbildung 5.1.: Das Mockup des User Interfaces. (Quelle: Eigene Darstellung)

repräsentieren. Über einen Druck auf den entsprechenden Knopf erscheint die zugehörige Ansicht.

Das Hauptfragment stellt die Kamera dar. Hier wird das Kamerabild analysiert und mittels Augmented Reality, um die entsprechenden Objekte erweitert. Zusätzlich ist noch ein Fragment zum Hinzufügen von Modellen und eins zum Verwalten der Fragments vorgesehen.

Letzteres bietet dem Nutzer die Möglichkeit bestehende Modell wieder zu entfernen.

## 5.5. Datenspeicherung

Die Anwendung muss die Modelle, die vom Anwender hochgeladen werden auf geeignete Weise intern abspeichern.

Überlegung  
zur korrek-  
ten Daten-  
speicherung

# Kapitel 6

## Implementierung

### 6.1. Laden der Modelldateien

Um die Dateien zu laden, die vom Nutzer hochgeladen wurden, mussten drei Klassen implementiert werden. Die *Model*-Klasse repräsentiert ein Modell und speichert die relevanten Daten des Modells für das Rendering. Zum Laden der Daten greift es auf die *ObjLoader*-Klasse und die *TextureLoader*-Klasse zurück.

#### 6.1.1. ObjLoader

Die *ObjLoader*-Klasse ist dafür zuständig die Daten aus der Obj-Datei des Modells einzulesen und in das passende Format für OpenGL umzuwandeln.

Bei ihrer Instanziierung wird der Klasse der Pfad zu der Datei die eingelesen werden soll übergeben. Nachdem diese geladen wurde, muss die Datei Zeile für Zeile durchlaufen werden und anhand der Abkürzungen, mit denen jede Zeile beginnt, der Datentyp bestimmt werden.

Der Aufbau einer Obj-Datei unterliegt dabei dem folgenden Muster (vergleiche dazu auch Abbildung 6.1):

Jede Zeile enthält ein Datenobjekt, welches über die die Kennung zu einem bestimmten Datentyp zugeordnet werden kann. Zu Beginn der Datei werden alle Einzelteile des Modells definiert:

**Vertex (v):** Ein Vertex repräsentiert einen Eckpunkt des Objektes. Dazu werden in der Zeile die drei Koordinaten des Punktes gespeichert.

**Textur (vt):** Diese Zeile besteht aus zwei Koordinaten, welche einen Punkt in einem zweidimensionalen Bild, der Texturdatei, repräsentieren.

**Normalenvektor (vn):** Dieser Kennung folgenden die drei Koordinaten eines Normalenvektors

Damit OpenGL mit diesen Einzelteilen arbeiten kann, müssen diese noch zusammengesetzt werden.

```

30 vn 0.0000 -1.0000 0.0000
31 vn 1.0000 0.0000 0.0000
32 vn 0.0000 0.0000 -1.0000
33 usemtl Material
34 s off
35 f 5/1/1 3/2/1 1/3/1
36 f 3/2/2 8/4/2 4/5/2

```

Abbildung 6.1.: Ausschnitt aus einer Obj-Datei. (Quelle: Eigene Darstellung)

Die dafür nötigen Informationen speichern die sogenannten Faces (deutsch: „Gesichter“), welche die Kennung f besitzen. Sie beschreiben die einzelnen Flächen des Modells. Da zuvor im Entwurf) die Anforderung an die Obj-Dateien gestellt wurde, dass diese trianguliert werden, handelt es sich bei den Flächen immer um Dreiecke. Die Zeile eines dieser Dreiecke speichert dafür für jeden Eckpunkt der Fläche drei Indices die auf jeweils einen der zuvor definierten Vertices, Textrukordinaten und Normalenvektoren verweisen. Dadurch wird neben der Koordinaten der Fläche, auch der zugehörige Ausschnitt der Textur und der Normalenvektor der Datei definiert. Das Programm erstellt aus diesen Daten drei Listen (jeweils eine für Vertices, Textrukordinaten und Normalenvektoren), indem es für jede Fläche die Daten jedes Punktes an die entsprechende Liste anhängt.

Auf diese Listen kann dann über die Klassenvariablen des *ObjLoader*-Objektes zugegriffen werden.

### 6.1.2. TextureLoader

Die *TextureLoader*-Klasse hat die Aufgabe die Textur-Datei einzulesen und OpenGL zur Verfügung zu Stellen. Dazu wird zunächst mit *GLES20 glGenTextures* eine Textur erzeugt und eine Referenz in *textureHandle* gespeichert. Anschließend wird die Texturdatei des Modells als Bitmap an die Textur gebunden und die Filter zur Texturverarbeitung gesetzt.

### 6.1.3. Model

Die *Model*-Klasse beruht auf dem Grundgerüst von der *Cube*-Klasse, welche ein Teil von ARToolKit ist und einen simplen Würfel darstellt.

Diese Klasse wurde im Rahmen der Arbeit so modifiziert, dass sie beliebige Modelle speichern kann.

Dazu werden mittels der *ObjLoader*-Klasse die Daten aus der Obj-Datei geladen und in Buffern gespeichert. Des weiteren wird mit Hilfe der *TextureLoader*-Klasse die Textur geladen und eine Referenz auf diese erzeugt.

Um das Modell zu Rendern verfügt es über eine *draw*-Methode, welche die Daten des Modells an das Shader Programm übergibt.

## 6.2. Augmented Reality

Die Struktur, die zur Umsetzung der Augmented Reality genutzt wurde, beruht zum Großteil auf dem Konzepten und Klassen von ARToolKit und wurde dann an die Anforderungen dieser Arbeit angepasst.

### 6.2.1. CameraFragment

Das *CameraFragment* bildet den Grundbaustein des Augmented Reality Fragments. Es erweitert das *ARFragment*, eine Klasse die im Laufe der Arbeit implementiert wurde. Sie ist von ihrer Funktionalität identisch zu der *ARActivity*-Klasse von ARToolKit und wurde lediglich an die Anforderungen eines Fragments angepasst, um in das Userinterface eingefügt werden zu können.

Zum einen wird in dieser Klasse die in C++ geschriebene, native ARToolKit-Bibliothek geladen, zum anderen dient die Klasse dazu alle wichtigen Objekte zu initialisieren und mit einander zu verbinden. Dafür wird unter anderem die visuelle Oberfläche in Form eines *FrameLayouts*, welches ein *GLSurfaceView* enthält, auf dem später das Kamerabild, sowie die Modelle projiziert werden, erzeugt. Des weiteren wird in dieser Klasse der *ARRenderer* an das *ARFragment* gebunden.

### 6.2.2. EducationARRenderer

Die *EducationARRenderer*-Klasse erweitert ARToolKits *ARRender*. Die Aufgabe des Renderes ist es die AR-Szene zu konfigurieren und die Visualisierung der AR-Modelle einzuleiten. Dazu werden zunächst in der *configureARScene*-Methode die Marker gesetzt. In der *draw*-Funktion werden dann die zuvor definierten Marker getrackt und die Model-, View- und Projektionsmatrix berechnet. Diese Matrizen werden dann an die *draw*-Funktionen des entsprechenden Modells übergeben, um die Modelle mit der richtigen Transformation zu visualisieren.

## 6.3. Markergenerierung

Damit der Nutzer die Modelle nutzen kann, muss die Anwendung dazu in der Lage sein selber Marker zu erzeugen. Dazu wurde die folgende Klasse im Rahmen der Arbeit implementiert.

### 6.3.1. MarkerGenerator

Die *MarkerGenerator*-Klasse ist eine Hilfsklasse die über die *generateMarker*-Methode einen Marker mit einer bestimmten ID und Patterngröße  $n$ , die über die Methodenparameter übergeben werden, erzeugt. Dazu wird zunächst eine quadratische Bitmap der Größe  $n \times n$  erzeugt. Anschließend muss aus der übergebenen ID ein Barcodemuster erzeugt werden.

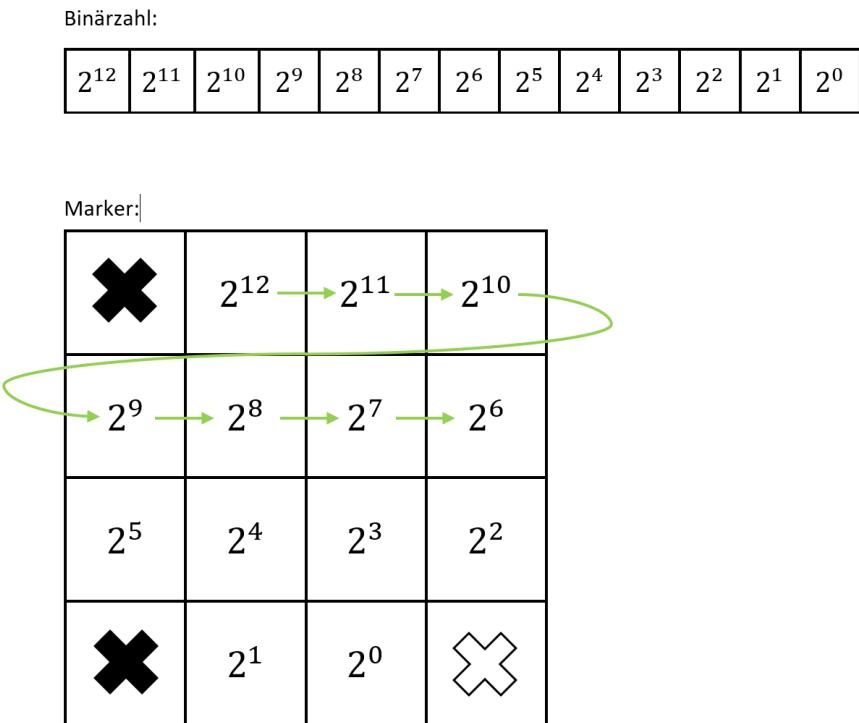


Abbildung 6.2.: Visualisierung des Vorgehens zur Erstellung eines 4x4 Barcodepatterns. (Quelle: Eigene Darstellung)

Dieses geschieht indem die ID zunächst in eine Binärzahl mit der Länge  $n^2 - 3$  umgewandelt wird. Jede Null in der Binärzahl repräsentiert ein weißes Feld und jede Eins ein schwarzes Feld im Barcodepattern. Damit das Pattern am Ende jedoch Rotationsinvariant ist sind insgesamt drei Felder des Musters bereits eine feste Farbe zugeordnet(in Abbildung 6.2 durch Kreuze in der entsprechenden Farbe dargestellt). Für diese vordefinierten Felder werden deshalb entweder eine Null oder eine Eins an den entsprechenden Stellen des Strings, der die Binärzahl repräsentiert , eingefügt. Im nächsten Schritt kann dann lediglich die Bitmap durchlaufen werden und der Wert des Strings an der Stelle  $i * n + j$ , wobei i die aktuelle Zeile und j die aktuelle Spalte repräsentiert, eingefügt werden.

# Kapitel 7

## Evaluation

### 7.1. Tests

Während der Entwicklung wurde mit Hilfe von ausführlichen Tests die Funktionalität neuer Features sichergestellt und eine ausführliche Testdokumentation angefertigt. Dadurch sollten Fehler, Probleme und mögliche Verbesserungen entdeckt und festgehalten werden.

Das Ziel war es für jedes neue Feature einen Test durchzuführen.

#### 7.1.1. Testdurchführung

Bei der Testdurchführung wurde darauf Wert gelegt, dass die einzelnen Durchläufe in der selben Testumgebung und unter den gleichen Umständen stattfinden, um eine Vergleichbarkeit zwischen den verschiedenen Versionen herzustellen. So konnte nach jedem Test zusätzlich festgestellt werden, ob sich eventuell die bestehende Eigenschaften im Vergleich zur Vorgängerversion verschlechtert oder verbessert hatten. Zu diesem Zweck wurde für jeden Durchlauf der in Abbildung 7.1 gezeigte Versuchsaufbau gewählt.

Da der Prototyp in Android Studio entwickelt wurde, war es zudem möglich die Anwendung direkt auf einem Androidendgerät zu testen. Dieses bot die Möglichkeit die Tests in einer realistischen, praxisnahen Umgebung durchzuführen und demzufolge noch bessere Erkenntnisse über die Alltagstauglichkeit der neuen Features zu erhalten.

Dafür wurde bei jedem Durchlauf ein Huawei P30 Pro genutzt und das Tracking wurde anhand eines dafür angefertigten Dokumentes getestet. Dieses Dokument wurde im Laufe der Entwicklung an die neuen Features angepasst und optimiert. Grundlegend bildet das Dokument einen oder mehrere Marker ab. Gegebenenfalls sind die Marker in Textpassagen eingebunden, um den Schwierigkeitsgrad für die Markererkennung zu erhöhen.

Verweis auf  
Testdoku-  
ment(Anhang)

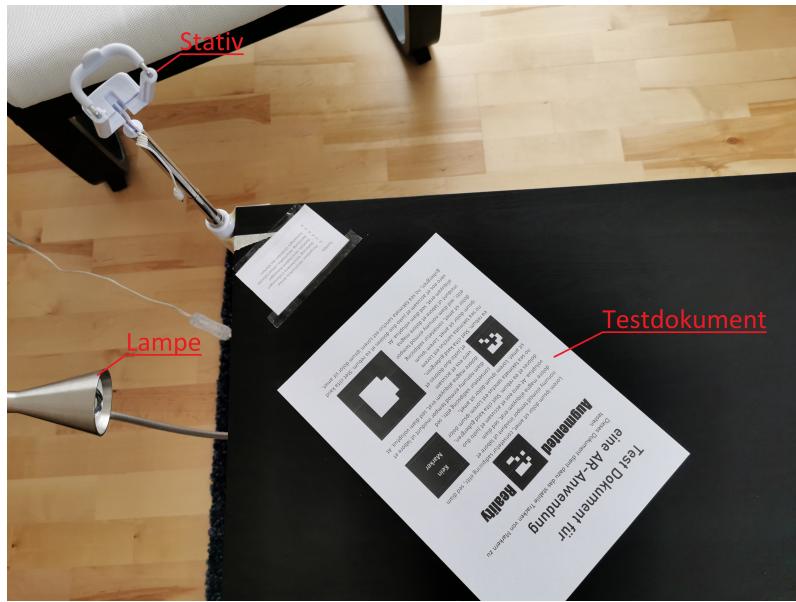


Abbildung 7.1.: Der Testaufbau für jeden Durchlauf. (Quelle: Eigene Darstellung)

Während des Testlaufs wurden jedes mal vier Testfälle durchlaufen, die in Abschnitt 7.1.2 beschrieben werden. Dabei wurden sowohl die Neuerungen getestet, als auch Veränderungen in den bereits bestehenden Features festgehalten.

Jeder Testfall wurde dabei mit Hilfe der in Android enthaltenen Funktion „Bildschirmrekorder“ aufgezeichnet und in dem entsprechenden Testbericht dokumentiert.

### 7.1.2. Testfälle

Die Testfälle beruhen auf den Eigenschaften des SIFT-Algorithmus, welcher Bildmerkmale extrahiert, die invariant gegenüber Rotation, Translation, Skalierung und partiell invariant gegenüber Helligkeitsveränderungen sind (Nischwitz u. a., 2011, S. 345). Mithilfe des Algorithmus können dieselben Objekte in zwei verschiedenen Aufnahmen wiedererkannt werden.

Da auch beim Marker Tracking ein ähnliches Verfahren angewandt werden muss, sind auch hierbei die genannten Eigenschaften relevant. Deshalb wurden die folgenden Testfälle gewählt, um die Funktionen der Anwendung zu testen:

- Perspektivische Invarianz: Dieser Testfall diente dazu das Tracking aus verschiedenen Perspektiven zu testen. Dazu wurde die Kamera auf das Testdokument gerichtet und anschließend der Winkel zum Dokument so verändert das verschiedene, perspektivische Verzerrungen der Marker erzeugt wurden.

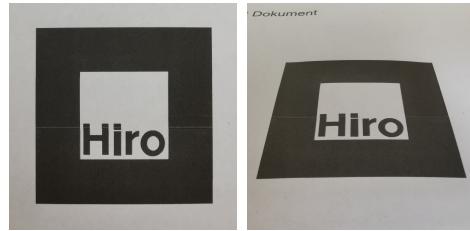


Abbildung 7.2.: Perspektivische Verzerrung eines Markers. (Quelle: Eigene Darstellung)

- Skalierungsinvarianz: Dieser Testfall diente dazu das Tracking aus verschiedenen Entfernungen zu testen. Dazu wurde die Kamera langsam auf das Testdokument zu- und weg bewegt, um verschiedenen Markergrößen bzw. -auflösungen zu erhalten.

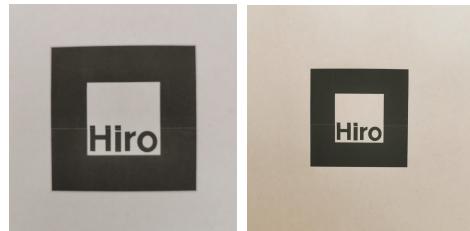


Abbildung 7.3.: Verschiedene Skalierungen eines Markers. (Quelle: Eigene Darstellung)

- Rotationsinvarianz: Dieser Testfall diente dazu das Tracking von Markern mit unterschiedlichen Rotationen zu testen, dazu wurde die Kamera auf das Dokument gerichtet und anschließend wurde letzteres langsam rotiert, um zu evaluieren, ob die Marker auch mit unterschiedlichen Rotationen korrekt erkannt werden.

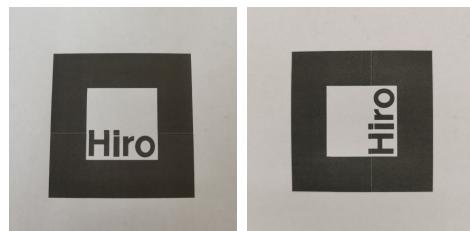


Abbildung 7.4.: Verschiedene Rotationen eines Markers. (Quelle: Eigene Darstellung)

- Belichtungsinvarianz: Dieser Testfall diente dazu das Tracking von Markern gegenüber unterschiedlichen Belichtungen zu testen. Dazu wurde mit Hilfe einer Lampe die Belichtung des Testdokumentes verändert.



Abbildung 7.5.: Verschiedene Belichtungen eines Markers. (Quelle: Eigene Darstellung)

- Trackinggeschwindigkeit: Dieser Testfall sollte die Geschwindigkeit des Trackings, sowie die Robustheit testen. Dazu wurden ein oder mehrere Marker kurzzeitig mit der Hand verdeckt, um zu testen, ob anschließend alle Marker wieder erfolgreich getrackt wurden und wie schnell dieses erfolgte.

Besserer Name

All diese Testfälle beziehen sich auf das Marker Tracking, während dessen konnten jedoch auch das Rendern des Modells und weitere Features die nicht direkt mit dem Tracking zusammenhängen getestet werden.

### 7.1.3. Zusammenfassung

- Belichtung nicht einfach zu testen - starke Belichtungswechsel teilweise kurze Aussetzer

Am Ende schreiben

# **Kapitel 8**

## **Fazit**

**8.1. Fazit**

**8.2. Ausblick**

# Literaturverzeichnis

- [Android a] ANDROID: *Application Fundamentals*. Webquelle. – URL <https://developer.android.com/guide/components/fundamentals?authuser=1>. – letzter Abruf: 16.08.2020
- [Android b] ANDROID: *Introduction to Activities*. Webquelle. – URL <https://developer.android.com/guide/components/activities/intro-activities>. – letzter Abruf: 16.08.2020
- [Android c] ANDROID: *Platform Architecture*. Webquelle. – URL <https://developer.android.com/guide/platform>. – letzter Abruf: 16.08.2020
- [Android d] ANDROID: *Understand the Activity Lifecycle*. Webquelle. – URL <https://developer.android.com/guide/components/activities/activity-lifecycle>. – letzter Abruf: 16.08.2020
- [Balzert 2011] BALZERT, H.: *Lehrbücher der Informatik*. Bd. 1: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Spektrum Akademischer Verlag, 2011. – ISBN 978-3-8274-1706-0
- [Böhm und Fuchs 2002] BÖHM, R. ; FUCHS, E.: *Wirtschaftsinformatik*. Bd. 1: *System-Entwicklung in der Wirtschaftsinformatik*. 5. vdf Hochschulverlag, 2002. – ISBN 978-3-7281-2762-4
- [de Vries, J. ] DE VRIES, J.: *Learn OpenGL - Hello Triangle*. Webquelle. – URL <https://learnopengl.com/Getting-started>Hello-Triangle>. – letzter Abruf: 20.08.2020
- [Gargenta 2011] GARGENTA, M.: *Learning Android - Building Applications for the Android Market*. 1. O'Reilly, 2011. – ISBN 978-1-449-39050-1
- [Hedberg u. a. 2018] HEDBERG, H. ; NOURI, J. ; HANSEN, R.: A Systematic Review of Learning Through Mobile Augmented Reality. In: *International Journal of Interactive Mobile Technologies* 12 (2018), Nr. 3, S. 75–85. – URL <https://www.online-journals.org/index.php/i-jim/article/view/8404>

- [Khronos Group a] KHRONOS GROUP: *Geometry Shader*. Webquelle. – URL [https://www.khronos.org/opengl/wiki/Geometry\\_Shader](https://www.khronos.org/opengl/wiki/Geometry_Shader). – letzter Abruf: 02.09.2020
- [Khronos Group b] KHRONOS GROUP: *OpenGL Overview*. Webquelle. – URL <https://www.opengl.org/about/>. – letzter Abruf: 09.08.2020
- [Khronos Group c] KHRONOS GROUP: *Rendering Pipeline Overview*. Webquelle. – URL [https://www.khronos.org/opengl/wiki/Rendering\\_Pipeline\\_Overview](https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview). – letzter Abruf: 20.08.2020
- [Khronos Group d] KHRONOS GROUP: *Tessellation*. Webquelle. – URL <https://www.khronos.org/opengl/wiki/Tessellation>. – letzter Abruf: 02.09.2020
- [Kind u. a. 2019] KIND, H. ; FERDINAND, J. ; JETZKE, T. ; RICHTER, S. ; WEIDE, S.: Virtual und Augmented Reality: Status quo, Herausforderungen und zukünftige Entwicklungen / TAB - Büro für Technikfolgen-Abschätzung beim Deutschen Bundestag. 2019. – TAB-Arbeitsbericht
- [Klein 2006] KLEIN, G.: *Visual Tracking for Augmented Reality*, University of Cambridge, Dissertation, 2006
- [Mehler-Bicher und Steiger 2014] MEHLER-BICHER, A. ; STEIGER, L.: *Augmented Reality - Theorie und Praxis*. Bd. 1. 2. De Gruyter Oldenbourg, 2014. – ISBN 978-3-11-035385-3
- [Milgram u. a. 1994] MILGRAM, P. ; TAKEMURA, H. ; UTSUMI, A. ; KISHINO, F.: Augmented Reality: A class of displays on the reality-virtuality continuum. In: *Telemanipulator and Telepresence Technologies* Bd. 2351, SPIE, 1994, S. 282–292
- [Murphy 2009] MURPHY, M. L.: *Beginning Android - Master Android from first principles and begin the journey toward your own successful Android applications!* 1. Apress, 2009. – ISBN 978-1-4302-2420-4
- [Nischwitz u. a. 2011] NISCHWITZ, A. ; FISCHER, M. ; HABERÄCKER, P. ; SOCHER, G.: *Computergrafik und Bildverarbeitung - Band II: Bildverarbeitung*. Bd. 2. 3. Vieweg+Teubner Verlag, 2011. – ISBN 978-3-8348-1712-9
- [Owen u. a. 2002] OWEN, C. ; XIAO, F. ; MIDDLIN, P.: What is the best fiducial?, 2002, S. 8 pp.. – ISBN 0-7803-7680-3
- [Robertson und Robertson 2012] ROBERTSON, S. ; ROBERTSON, R.: *Mastering the Requirements Process: Getting Requirements Right*. Bd. 1. 3. Pearson Education, 2012. – ISBN 978-0-321-81574-3

[Shreiner u. a. 2006] SHREINER, D. ; WOO, M. ; NEIDER, J. ; DAVIS, T.: *OpenGL programming guide - the official guide to learning OpenGL, version 2. 5.* Addison-Wesley, 2006. – ISBN 0-321-33573-2

[Ćuković u. a. 2015] ĆUKOVIĆ, S. ; GATTULLO, M. ; PANKRATZ, F. ; DEVEDZIC, G. ; CARRABBA, E. ; BAIZID, K.: Marker Based vs. Natural Feature Tracking Augmented Reality Visualization of the 3D Foot Phantom, 2015

[Winter 2018] WINTER, Andreas: *SRS Anforderungen.* Vorlesung Softwaretechnik I. 2018

# **Anhang A**

## **Beispielanhang**

Beispiel für einen Anhang!

# **Erklärung**

Hiermit versichere ich, Johannes Scheibe, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

---

Johannes Scheibe