Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Raimund Wege

Ein kollaborativer 3D Mixed-Reality Editor für mobile Geräte

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Raimund Wege

Ein kollaborativer 3D Mixed-Reality Editor für mobile Geräte

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Birgit Wendholt
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 8. Dezember 2017

Raimund Wege

Thema der Arbeit
Ein kollaborativer 3D Mixed-Reality Editor für mobile Geräte

Stichworte
Verteilte Kollaboration, Augmented-Reality, Virtual-Reality, Mixed-Reality, mobile Geräte, 3D-Editor, Entity-Component-System

Kurzzusammenfassung
Diese Thesis beschreibt die Konzeption und prototypische Implementation eines kollaborativen 3D-Editors. Eine Server-Instanz hält eine Szene, zu der sich Client-Instanzen verbinden können, um diese zu betrachten und zu manipulieren. Eine Implementation für mobile Geräte ermöglicht größtmögliche Verfügbarkeit. Schwache Geräte können als Werkzeugkasten verwendet werden, während stärkere Geräte unter Nutzung von Augmented- und Virtual-Reality eine intuitivere Visualisierung und Manipulation von 3D-Objekten ermöglichen. Das Ergebnis ist ein einsatzbereiter Prototyp.
Raimund Wege

Title of the paper
A collaborative 3D mixed-reality editor for mobile devices

Keywords
Distributed collaboration, augmented-reality, virtual-reality, mixed-reality, mobile devices, 3D editor, entity-component-system

Abstract
This thesis describes conception and prototypic implementation of a collaborative 3D editor. A server-instance is holding a scene at those client-instances can register to view or manipulate it. The implementation for mobile devices makes it available as much as possible. Weak devices can serve as a toolbox, while stronger ones may use augmented- and virtual-reality to allow intuitively operable visualization and manipulation of 3D objects. The outcome is a ready-to-use prototype.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Nachdem der Erfolg für Project Tango[1] ausblieb, hat Apple in diesem Jahr mit ARKit[2] einen Volltreffer gelandet und Google damit so sehr unter Zugzwang gestellt, dass sie Project Tango kurzerhand eingestellt haben und nun mit ARCore[3] versuchen mit Apple gleichzuziehen [Janssen (2017)]. Der Vorteil von ARKit und ARCore gegenüber dem Project Tango ist, dass das Smartphone, auf dem die AR-Anwendung laufen soll, keine zusätzliche Hardware benötigt, sondern nur die Kamera und den Beschleunigungssensor, die bei den meisten Geräten zur Standardausstattung gehören. Der Nachteil gegenüber dem Project Tango ist jedoch, dass ein Gerät ohne Tiefenwahrnehmung bislang keine Entscheidung darüber treffen kann, ob sich ein virtuelles Objekt vor oder hinter einem realen Objekt befindet. Nichtsdestotrotz wurde mit diesen Frameworks nun der Grundstein für die Zukunft von Augmented-Reality Apps gelegt.

1.2 Vision

Die Vision eines Editors, dessen Realisierung in dieser Arbeit angestrebt wird, soll im Folgenden durch ein konkretes Anwendungsbeispiel beschrieben werden. In dem Beispiel werden alle gewünschten Funktionalitäten und Anforderungen abgedeckt. Der nachfolgende Text wird durch Abbildung 1.1 visualisiert.

Alice und Bob leben in einer kleinen Mietwohnung im Zentrum von Hamburg. Nun haben sie sich dazu entschieden, ein eigenes Haus zu bauen. Dafür haben sie bereits ein Grundstück erworben. Im nächsten Schritt soll das neue Zuhause von beiden geplant werden. Im Editor erstellen sie einfach eine neue 3D-Szene, in welcher das neue Haus entstehen soll. Alice nutzt ein Smartphone als Toolbar und bearbeitet das Haus an ihrem Tablet, während Bob mit seiner AR-Brille die selbe Szene sieht wie Alice, nur dass er direkt in sein zukünftiges Heim eintauchen

[1] http://get.google.com/tango/
[2] https://developer.apple.com/arkit/
[3] https://developers.google.com/ar/

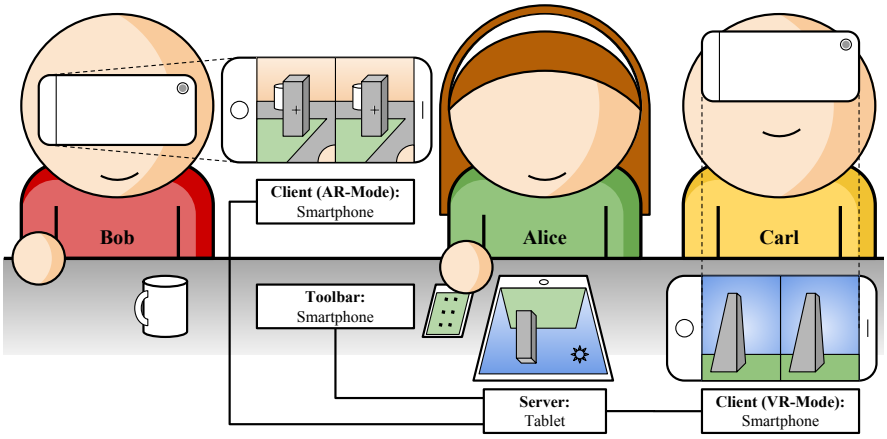


Abbildung 1.1: Alice und Bob entwerfen ihr Traumhaus, Carl macht einen virtuellen Rundgang.

kann. Gemeinsam planen sie den Grundriss, einzelne Zimmer, Treppen und Fenster. Bevor sie mit der virtuellen Einrichtung beginnen, lassen sie ihren Bekannten Carl per Smartphone den Hausentwurf erkunden. Er macht sie durch sein Fachwissen auf einige Konstruktionsfehler aufmerksam, welche die beiden in seiner Anwesenheit und mit seiner Rückmeldung verbessern können. Während Alice schon den Standort der Schränke plant, geht Bob mit dem 3D-Entwurf zu einem Bauherren, um das Projekt Traumhaus zu starten.

1.3 Zielsetzung

Im Rahmen dieser Arbeit wird die Realisierung eines kollaborativen 3D-Editors verfolgt, der den Schwerpunkt auf mobile Geräte legt und der in der Lage sein wird die Vision zu verwirklichen. Es soll keine spezielle Hardware nötig sein, wie z. B. festinstallierte Tiefenbildkameras oder ein spezieller Handschuh, um mit dem Arbeitsbereich interagieren zu können. Der Einsatz von AR und VR soll für die Anwender optional bleiben. Der Funktionsumfang des Editors soll dem entsprechen, was innerhalb des Zeitrahmens der Masterarbeit umgesetzt werden kann.

Der entstehende Editor soll eigenständig nutzbar sein und ohne großen Aufwand eingerichtet werden können. Dies ist vor allem für eine Zielgruppe wichtig, die wenig IT-Erfahrungen aufweist, es soll also keine Spezialisten-Software entstehen, sondern eine für die breite Masse nutzbare Anwendung. Erstellte Szenen sollen unter den Editoren ausgetauscht werden können.

Bei der Implementierung des Systems wird viel Wert auf Erweiterbarkeit gelegt, vor allem da der im Rahmen der Masterarbeit geleistete Umfang nur wenig mehr als die geforderte Grundfunktionalität beinhalten wird. So ist es mehr als eine Machbarkeitsanalyse zu betrachten, aus welcher mit entsprechendem Mehraufwand ein marktfähiges Produkt entstehen kann. Einige

Funktionalitäten, die eine plattformspezifische Implementation erfordern würden, werden aus Zeitgründen nur exemplarisch für eine Plattform zur Verfügung gestellt.

1.4 Struktur der Arbeit

Kapitel 2 geht auf notwendige Grundlagen ein und stellt vergleichbare Arbeiten vor. Kapitel 3 führt eine Anforderungsanalyse an einem Szenario durch, das sich auf die Vision aus diesem Kapitel bezieht. Kapitel 4 beschäftigt sich mit dem Konzept und der Umsetzung eines Prototypen. Kapitel 5 fasst die Arbeit abschließend zusammen.

2 Vergleichbare Arbeiten

Im Folgenden werden einige Arbeiten vorgestellt, die vorrangig mit der Erstellung von 3D-Modellen in Verbindung stehen. Sie behandeln Aspekte und Algorithmen, die für die vorliegende Masterarbeit übernommen wurden oder als Inspiration dienten. Nach der Vorstellung einer Arbeit wird jeweils noch einmal auf den Kernaspekt eingegangen und inwiefern ein Mehrwert für die Masterarbeit daraus gezogen werden konnte.

2.1 Grundlagen

Als Grundlage für das Verständnis der weiteren Arbeit gilt es, zum einen Grundbegriffe der 3D-Modellierung zu klären, sowie das Entwurfsmuster vorzustellen, das für einen Großteil des Editors verwendet wurde.

2.1.1 3D-Objekte und -Techniken

Auf detailliertere Erläuterungen und Erklärungen von 3D-Objekten und 3D-Techniken wird an dieser Stelle verzichtet, als Grundlagenwerk für die 3D-Programmierung für Smartphones wird der "Open GL(R) ES 2.0 Programming Guide" [Munshi u. a. (2008)] und im Speziellen für die Programmierung von Shadern das "Orange Book" [Rost u. a. (2009)] empfohlen.

Nachfolgend werden grundlegende Begriffe eingeführt, die für die Verwendung im Editor und der Handhabung durch den Benutzer wichtig sind. Einige der Begriffe sind geprägt durch das Grafik-Framework "libGDX" [1], das für die Umsetzung der Thesis verwendet wurde.

Vertex plural Vertices, sind Punkte, über welche das 3D-Modell definiert wird. Dies sind besonders bei einfachen Körpern die Eckpunkte, ein Würfel besitzt somit 8 Vertices, die über Edges verbunden werden.

[1] https://libgdx.badlogicgames.com/

Edge Kanten (Edges) verbinden Vertices miteinander und werden durch einfache Linien dargestellt. Drei Edges können dafür verwendet werden um ein Dreieck bzw. eine Fläche zu bilden.

Face Flächen (Faces) definieren das sichtbare Objekt. Für gewöhnlich wird eine Fläche nur dann angezeigt, wenn ihre Normale (die Ausrichtung oder Oberseite der Fläche) in Richtung des Benutzers zeigt.

Mesh Ein Mesh ist ein Geflecht aus Vertices, Edges und Faces, die zusammen ein eigenständiges Konstrukt bilden.

Model Ein Model ist die Blaupause für eine komplexe Konstruktion, die auch aus mehreren Meshes bestehen und Animationen, sowie Material beinhalten kann.

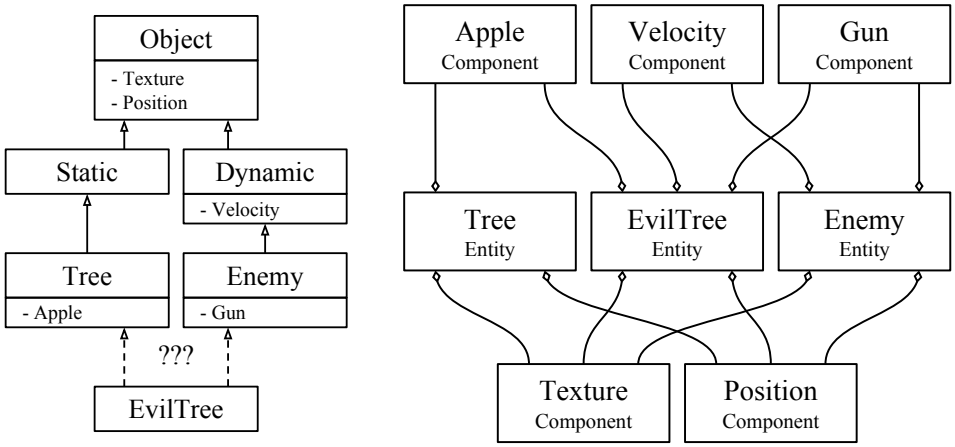
Model-Instance Aus einem Model können eigenständige Instanzen erzeugt werden, die eigene Ausprägungen haben. Ausprägungen sind z. B. die Position, die Größe, die Ausrichtung und das Material.

2.1.2 Entity-Component-System

Bei dem Entity-Component-System (ECS) handelt es sich um ein Design-Pattern, das auf Komposition anstelle von Vererbung setzt. Dieses Pattern findet häufig aufgrund seiner Erweiterbarkeit und Flexibilität bei Computerspielen Verwendung [Hall u. a. (2014)].

Die traditionelle Art und Weise, Spielelemente zu implementieren, bestand lange Zeit darin, objektorientierte Programmierung zu verwenden. Dabei wurde jedes Spielelement als eine eigene Klasse implementiert und durch Polymorphie sollten bestehende Klassen intuitiv erweitert werden können. Dies führte jedoch zu großen, starren Klassenhierarchien. Als die Anzahl der Spielelemente bzw. der Klassen wuchs, wurde es zunehmend schwieriger, ein neues Spielelement in der Hierarchie unterzubringen, insbesondere dann wenn es viele verschiedene Funktionstypen benötigte (siehe Abbildung 2.1a).

Um dieses Problem zu lösen, wurden Spielelemente nicht länger durch Vererbung erzeugt, sondern durch Komposition. Ein Spielelement wird von nun an durch eine Entität repräsentiert und besteht aus einer Aggregation von Komponenten. Eine Entität ist nur noch der Container für eine Ansammlung von Daten, die in der Form von Komponenten festgehalten und dafür verwendet werden eine Entität zu charakterisieren. Dies hat einige entscheidende Vorteile



(a) Vererbung

(b) Komposition

Abbildung 2.1: Bei der Vererbung a) kommt es zum Diamond-Problem, sobald eine neue Klasse die Funktionalität von zwei existierenden Klassen benötigt, die sich aber auf unterschiedlichen Vererbungspfaden befinden. Die Komposition b) umgeht dieses Problem, in dem ein neues Objekt nur durch die Zusammenstellung von Components definiert wird.

gegenüber der objektorientierten Architektur. Zum einen ist es jetzt möglich neue Eigenschaften bzw. Komponenten zu entwickeln ohne bestehenden Code verändern zu müssen und zum anderen können jetzt komplett neue Entitäten nur durch die Auswahl von Komponenten definiert werden (siehe Abbildung 2.1a).

Funktionsweise Alle ECS funktionieren mit einer sog. Engine als Haupteinheit. Bei dieser werden Entitäten und Systeme registriert. Entitäten wiederum sind mit Komponenten versehen, über welche den Entitäten Eigenschaften mitgegeben werden können. Systeme wiederum iterieren über Entitäten mit bestimmten Komponenten. Ein Rendering-System verarbeitet z. B. alle Entitäten mit der Komponente Renderable. Die Aufteilung der Architektur anhand eines ECS erhöht die Übersichtlichkeit und Erweiterbarkeit, da je nach Bedarf auch zur Laufzeit Entitäten, Komponenten und Systeme hinzugefügt oder entfernt werden können. Im Falle des Editors können über Entitäten beliebige, neue 3D-Objekte der Szene hinzugefügt, verändert und entfernt werden. Ein weiterer großer Vorteil ist, dass der gesamte Zustand der Engine, durch die Entitäten und Komponenten beschrieben wird. Somit ist es möglich den Zustand durch deserialisieren aller Entitäten und Komponenten zu persistieren und sogar zu transferieren. Ebenfalls eine Tatsache, die sich der Editor bei der initialen Synchronisation zu Nutzen macht.

Nachfolgend werden grundlegende Begriffe eingeführt, die für die Verwendung im Editor und der Handhabung durch den Benutzer wichtig sind. Einige der Begriffe sind geprägt durch das ECS *ashley* [2], das für die Umsetzung der Thesis verwendet wurde und Bestandteil von libGDX ist (siehe Abbildung 2.2). Es ist allerdings auch möglich *ashley* in anderen Projekten als eigenständiges Framework zu nutzen.

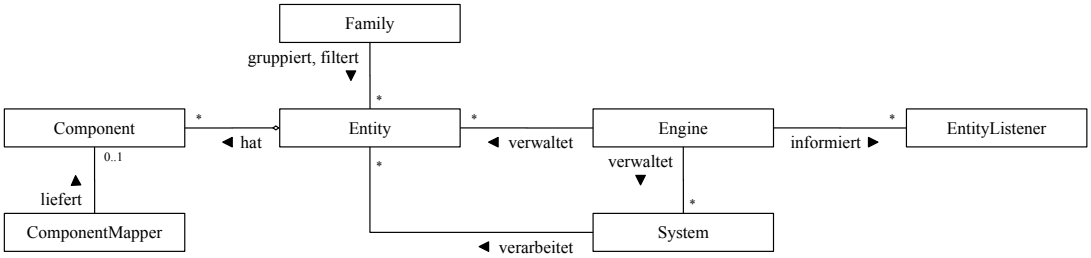


Abbildung 2.2: Das Java Framework *ashley* stellt eine mögliche Implementation des ECS Design-Patterns zur Verfügung [vgl. Saltares (2014)].

Entity Eine Entität (Entity) ist ein Konstrukt, das innerhalb der Engine existiert. Sie definiert sich durch eine Ansammlung von Komponenten. Bei Bedarf können zur Laufzeit Komponenten zur Entität hinzugefügt oder weggenommen werden [vgl. Ehrlich (2013)].

Component Eine Komponente (Component) ist ein Konstrukt, welches ausschließlich als Datenspeicher fungiert. Sie selbst enthält keine eigene Logik. Eine Ausnahme stellen berechnende Methoden dar, die auf den Daten der Komponente basieren. Erst durch die Kombination mehrerer Komponenten zu einer Entität erhält sie ihre Mächtigkeit. Die Instanz einer Komponente kann auch mehreren Entitäten zugeordnet werden. Dies ist z. B. sinnvoll, wenn sich mehrere Entitäten eine Textur teilen sollen. Eine Entität wiederum kann maximal eine Instanz von jeder Komponente besitzen. Sollte eine Entität mehrere Instanzen einer Komponente benötigen, kann eine neue Komponente definiert werden, die eine Liste für die gewünschten Instanzen beinhaltet. Es ist auch möglich eine Komponente zu definieren, die eine Entität beinhaltet. Auf diese Weise kann eine Entität einer anderen Entität zugewiesen werden. Auch leere Komponenten ohne Daten sind hilfreich, um Entitäten zu markieren.

System Ein System ist darauf ausgelegt eine bestimme Aufgabe immer wieder zu wiederholen. Dies kann z. B. das Rendern einer Szene, das Speichern von Daten oder das Lösen von

[2] https://github.com/libgdx/ashley

Gleichungen sein. Der Kreativität sind hier keine Grenzen gesetzt. Über eine Family legt das System fest für welche Entitäten es zuständig ist.

Family Über eine Family können Entities gruppiert bzw. gefiltert werden. Diese Funktion wird unteranderem in den Systemen und den Entity-Listenern benötigt.

Component-Mapper Über den Component-Mapper wird ein schneller Zugriff auf die Komponenten einer Entity gewährleistet.

Entity-Listener Entity-Listener können benutzt werden, um auf das Hinzufügen oder Löschen einer Entity zu reagieren. Auch hier wird über eine Family beschrieben, auf welche Entities reagiert werden soll.

2.2 Editoren

Im Folgenden werden einige bereits existente Editoren zur Manipulation von dreidimensionalen Objekten und Szenen vorgestellt. Teilweise werden bereits Funktionsweisen umgesetzt, die für den zu erarbeitenden Editor wünschenswert sind, es gibt aber auch Nachteile, die im Anschluss an die jeweilige Vorstellung kurz beleuchtet werden.

2.2.1 Clara.io

Bei Clara.io [Houston u. a. (2010)] handelt es sich um einen cloud- und webbasierten Editor, der sich an professionelle 3D-Artists richtet. Die Lösung bietet einen kollaborativen Arbeitsbereich, in dem es möglich ist 3D-Objekte zu modellieren, zu animieren und zu simulieren. Dabei verwendet Clara.io, wie die meisten 3D-Editoren, einen erweiterbaren und generativen Szenen-Graphen. Was Clara.io jedoch von anderen Editoren unterscheidet, ist die Tatsache, dass die Szene nicht in einer lokalen Datei gespeichert wird, sondern über Cloud-Functions in einer richtigen Datenbank. Um Anwendern den Einstieg zu erleichtern, orientiert sich das User-Interface an bereits existierenden Editoren.

Fazit

Clara.io ist ein kollaborativer Editor, der im Browser funktioniert und somit keine spezielle Hardware benötigt. Allerdings ist dieser Editor nicht auf mobile Geräte ausgelegt. Er nutzt nicht die Vorteile von Gesten oder Mixed-Reality, was die Handhabung deutlich vereinfachen könnte.

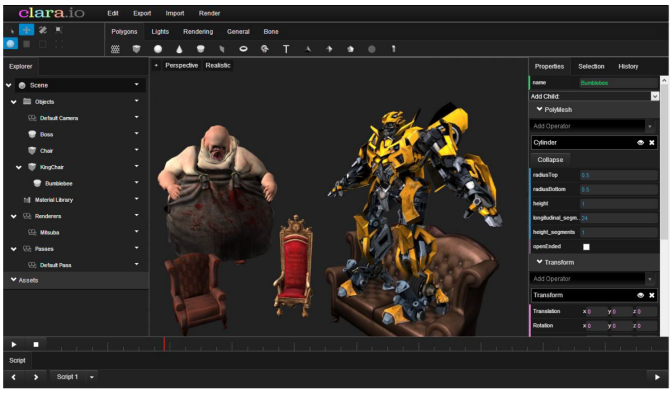


Abbildung 2.3: Clara.io [Quelle: Houston u. a. (2010)].

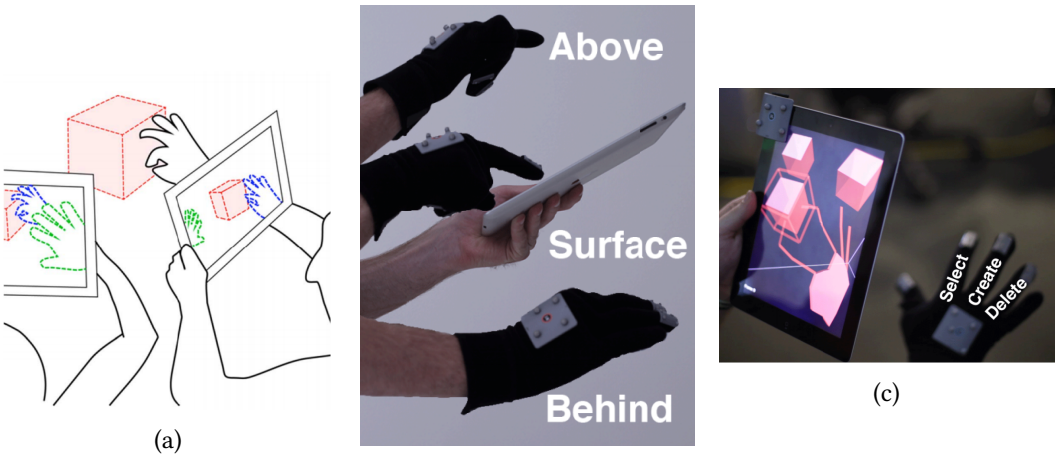
2.2.2 T(ether)

Es handelt sich bei T(ether) [Lakatos u. a. (2014)] um ein räumlich wahrnehmbares Mehrbenutzer-Anzeigesystem, das kollaborative Manipulationen und Animationen von virtuellen 3D-Objekten gestattet. Das Handheld-Display fungiert als Fenster in die virtuelle Realität (siehe Abbildung 2.4) und bietet dem Anwender eine perspektivische Darstellung von 3D-Daten. T(ether) verwendet ein Tracking-System, das die Köpfe, Hände, Finger und Fingerbewegungen der Benutzer verfolgt. Zusätzlich hat jeder Anwender ein Tablet, über das eine Vielzahl von Interaktionen mit der virtuellen Szene ausgeführt werden können.

Es werden Interaktionstechniken vorgestellt, die Tiefenbildinformationen nutzen, um die Benutzeroberfläche des Handheld-Displays auf Grundlage der Handposition anzupassen. Dabei wird unterschieden zwischen: die Hand befindet sich über dem Display, unter dem Display oder auf der Oberfläche des Displays (siehe Abbildung 2.4b). In jedem Interaktionsraum lassen sich unterschiedliche Interaktionen durchführen. Oberhalb des Handheld-Displays lässt sich bspw. die Geschwindigkeit von Animationen durch physikalische Gesten steuern. Unterhalb des Handheld-Displays wird die eigene Hand in Form eines Skelett-Models sichtbar (siehe Abbildung 2.4c) und es lassen sich mit der eigenen Hand Modelle manipulieren. Durch Touch-Gesten auf dem Display lassen sich Figuren in den Raum zeichnen. Diese räumlichen Interaktionen sollen es dem Benutzern vereinfachen, das Modell zu manipulieren und zu animieren.

Fazit

T(ether) ist ein kollaborativer 3D-Editor, der für mobile Geräte implementiert ist und sowohl Gestensteuerung, als auch Mixed-Reality ermöglicht. Ein großer Nachteil ist, dass für T(ether)



(a) (b) (c)

Abbildung 2.4: T(Ether): (a) Skizzierte Darstellung eine Kollaboration von zwei Anwendern, (b) die drei unterschiedlichen Aktionsräume, (c) mit einem Tablet wird in die virtuelle Szene geblickt und mit einem Handschuh lässt sich die Welt manipulieren Lakatos u. a. (2014).

spezielle Hardware benötigt wird (ein Handschuh). Jeder Teilnehmer an der Modellierung der 3D-Szene müsste also einen entsprechenden Handschuh besitzen.

2.2.3 MixFab

Bislang sind Kenntnisse in der 3D-Modellierung erforderlich, um eigene 3D-Objekte zu erzeugen und auszudrucken. Diese Barriere möchte MixFab Weichel u. a. (2014) senken. In MixFab interagieren Benutzer durch Gesten in einer immersiven AR-Umgebung mit virtuellen Objekten und sie können existierende Objekte in ihre Designs einbinden. Hände und andere reale Objekte überdecken virtuelle Objekte in dem sie als ein schwarzer Schatten in der virtuellen Welt dargestellt werden. Mit der Hilfe einer Webcam findet ein Face-Tracking statt, das dafür verwendet wird, um beim Rendern eine Parallaxe zu erzeugen, die wiederum als Tiefeneffekt wahrgenommen wird.

Für die Visualisierung verwendet MixFab eine Strahlenteilerkamera und einen halbdurchlässigen Spiegel, der mit einem Winkel von 45° angebracht ist. Durch den halbdurchlässigen Spiegel sieht der Anwender in den Interaktionsraum. Der Interaktionsraum entspricht in etwa der maximalen Größe von 3D-Druckerzeugnissen aus herkömmlichen 3D-Druckern. Eine Tiefenbildkamera die oberhalb des Setups angebracht ist, dient der Erfassung von realen Objekten und der Erkennung von Benutzergesten. Um reale Objekte mit einem hohen Detailgrad einscannen zu können, befindet sich am Boden der Konstruktion ein Drehteller. Das System

bedarf nur einer einmaligen Kalibrierung und muss nicht für einzelne Benutzer angepasst werden.

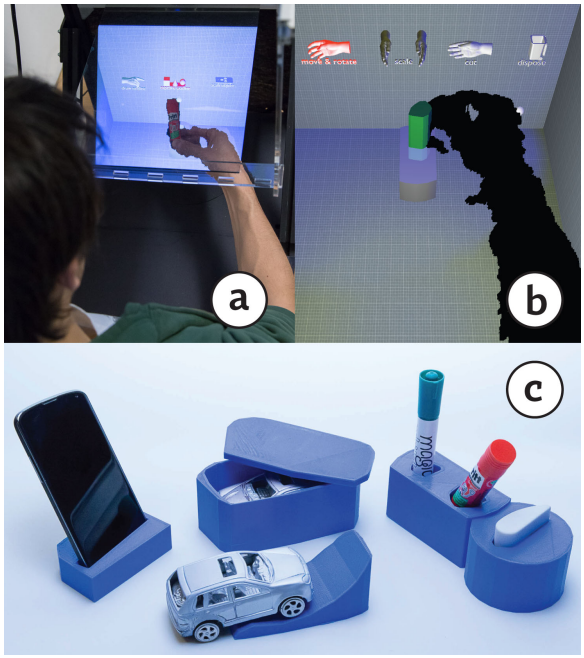


Abbildung 2.5: MixFab: Mixed-Reality Umgebung für die Produktion von 3D-Druckerzeugnissen. (a) Ein Benutzer positioniert ein physisches Objekt im MixFab Prototyp-System, (b) ein Benutzer manipuliert ein virtuelles Objekt, (c) eine Auswahl von 3D-Druckerzeugnissen [Quelle: Weichel u. a. (2014)].

In einer Benutzerstudie wurde in Erfahrung gebracht mit welchen Gesten Benutzer intuitiv mit dem System interagieren würden. Hierzu wurde den insgesamt zwölf Probanden jeweils eine Abbildung mit einer Ausgangssituation und einem gewünschten Resultat vorgelegt. Die Probanden sollten nun eine Geste beschreiben, die sie selbst als intuitiv erachten würden, um von der Ausgangssituation zum gewünschten Result zu gelangen. Anschließend sollten sie bewerten für wie intuitiv sie die eigene Geste halten und wie schwer sie wohl auszuführen wäre. Zum Abschluss mussten sie noch Fragen zu ihrem Alter, ihrem Geschlecht und ihren Erfahrungen mit CAD Anwendungen beantworten. Aus den am häufigsten vorgeschlagenen Gesten wurde das User-Interface hergeleitet.

Das User-Interface sieht vor, dass der Benutzer zu Begin eine der zu dem Augenblick möglichen Gesten auswählt in dem er auf ein entsprechendes “Gesture-Icon” zeigt. Danach erwartet das System, dass der Benutzer die entsprechende Geste ausführt. Objekte, die in MixFab virtuell dargestellt werden, können drei unterschiedliche Zustände annehmen. Es gibt den Ruhezu-

stand, Objekte sind grau und werden nicht durch die Geste beeinflusst, den inaktiven Zustand Objekte sind gelb und können durch die Geste beeinflusst werden, und den aktiven Zustand, Objekte sind grün und werden bereits aktiv manipuliert.

Fazit

MixFab ist ein 3D-Editor, der in einer speziellen Mixed-Reality-Umgebung mit Gesten gesteuert werden kann. Allerdings lässt sich MixFab nicht mobil einsetzen, da sehr große und spezielle Hardware benötigt wird. Außerdem ist MixFab nicht für kollaboratives Arbeiten ausgelegt.

2.2.4 Teddy und Plushie

Bei Teddy [Igarashi (1999)] und der Weiterentwicklung Plushie [Mori und Igarashi (2007)] handelt es sich um ein Interface, das aus einfachen Strichzeichnungen rundliche 3D-Objekte erzeugt, die verwendet werden können um z.B. Stofftiere zu designen. Auf einem Bildschirm zeichnet der Anwender hierbei interaktiv unterschiedliche 2D-Silhouetten. Aus den Silhouetten konstruiert das System anschließend automatisch plausible 3D-Objekte (siehe Abbildung 2.6). Das System verfügt über weitere Operationen, mit denen die 3D-Objekte modelliert werden können. Diese Operationen werden ebenfalls durch einfache Linien und Kurven ausgeführt. Erstanwender sollen bereits nach zehn Minuten alle Operationen beherrschen.

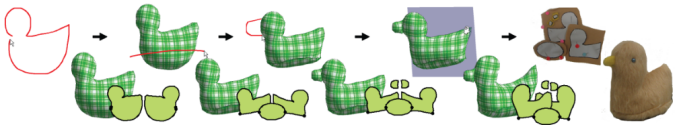


Abbildung 2.6: In Plushie werden einfache 2D-Zeichnungen verwendet, um automatisch 3D-Objekte zu erzeugen [Quelle: Mori und Igarashi (2007)].

Fazit

Teddy bzw. Plushie ist ein 3D-Editor, der durch den Input von 2D-Gesten in der Lage ist, 3D-Modelle zu erstellen. Der Editor ist konzeptionell auch auf mobilen Geräten einsetzbar, benötigt also keine spezielle Hardware. Der Editor funktioniert allerdings nicht kollaborativ und nutzt nicht die Möglichkeiten von Mixed-Reality. Die erzeugten 3D-Modelle haben außerdem den Nachteil, dass sie sehr rundlich werden und es nicht möglich ist gerade Kanten und rechte Winkel zu modellieren. Daher eignet sich dieser Editor nicht für die Modellierung von technischen Bauteilen, sondern eher für die Modellierung von organischen Objekten.

2.3 Zusammenfassung

Clara.io ist bereits ein sehr mächtiger Editor, der es auch erlaubt Szenen in einer kollaborativen Umgebung zu gestalten. Jedoch wird er nur über das Web zur Verfügung gestellt. Dies ermöglicht zwar auch die Ausführung auf einem Smartphone, aber das User-Interface ist so stark überladen, dass kein Platz mehr bleibt, um die Szene zu betrachten.

Der Editor Teddy bzw. Plushie ist zwar nicht geeignet um 3D-Objekte mit harten Kanten und rechten Winkeln zu erzeugen und es könnte auch passieren, dass durch die vielen Rundungen die Anzahl der Vertices schnell ansteigt, aber die hohe Lernkurve und die Verwendung von einfachen 2D-Gesten, die sich ohne Probleme auch auf einen Touch-Screen übertragen ließen, werden bei der Entwicklung des Editors berücksichtigt.

Der Nachteil des MixFab Systems ist, dass sehr viel spezielle Hardware benötigt wird, es ortsgebunden ist, es eine Single-User Anwendung ist und dass man nur einfache 3D-Objekte entwerfen kann. Ein interessanter Aspekt ist jedoch die Möglichkeit das reale Objekte eingescannt werden können und dass durch das Face-Tracking ein Parallax-Effekt erzeugt werden kann.

Der Nachteil von T(ether) ist leider, dass es auch ortsgebunden ist und nicht ohne ein komplexes Tracking-System und einen speziellen Handschuh auskommt.

3 Anforderungsanalyse

In den vorangegangenen Kapiteln wurde die Vision eines immersiven und kollaborativen Arbeitsbereichs vorgestellt, der auf unterschiedlichen mobilen Geräten genutzt werden kann und der keine spezielle Hardware benötigt. Dazu wurde in einer Vision zunächst ein Anwendungsfall näher umrissen und anschließend wurden vergleichbare Arbeiten untersucht und bewertet. Dabei hat sich herausgestellt, dass kein Verfahren bisher dafür geeignet ist, die geschilderte Vision zu realisieren.

Damit eine geeignete Lösung gefunden werden kann, wird im Abschnitt 3.1 zunächst die Vision aus dem ersten Kapitel aufgegriffen und durch ein Szenario weiter ausgearbeitet. Anschließend werden im Abschnitt 3.2 die Rollen und Anwendungsfälle betrachtet, die sich aus dem Szenario ergeben haben. Im Abschnitt 3.3 wird das Storyboard vorgestellt, das beschreibt, welche Ansichten es später im Editor geben muss und wie zwischen ihnen gewechselt werden kann. Zum Schluß werden im Abschnitt 3.4 die Anforderungen präsentiert, die an den Editor gestellt werden, um die Vision im Rahmen dieser Arbeit zu verwirklichen.

3.1 Szenario

In diesem Abschnitt wird ein Szenario vorgestellt, in dem Alice eine neue Szene erstellt, die sie zunächst alleine im Editor bearbeitet. Später möchte sie die Meinung von Bob einholen und lädt ihn aus diesem Grund zu sich in die Szene ein. Sie stellen die Szene gemeinsam fertig und laden Carl ein, um ihm das Ergebnis zu präsentieren.

A) Einen neuen Editor starten

1) Alice startet die Anwendung auf ihrem Tablet. Nach einem kurzen Ladebildschirm befindet sie sich im Startmenü, von wo aus sie drei Optionen hat. Sie kann einen neuen Editor starten, sich mit einem bestehenden Editor verbinden oder eine Einladung scannen. Alice entscheidet sich dafür einen neuen Editor zu starten.

2) Alice sieht eine Liste von Szenen, die bereits auf ihrem Tablet vorhanden sind. Eine Szene lässt sich durch antippen selektieren. Eine selektierte Szene kann umbenannt,

entfernt oder geöffnet werden. Außerdem lassen sich neue Szenen erstellen. Alice entscheidet sich dafür eine neue Szene zu erstellen, woraufhin sich ein Dialog öffnet, der sie auffordert den Namen der Szene einzugeben. Sie gibt ihr den Namen "Unser neues Haus" und bestätigt den Dialog.

3) Alice sieht nun den neuen und bereits selektierten Eintrag "Unser neues Haus" in der Liste und öffnet die Szene.

4) Alice hat durch das Öffnen der Szene einen neuen Editor gestartet und befindet sich nun in einer Default-Szene.

B) Eine Szene bearbeiten

1) Alice hat einen neuen Editor gestartet und befindet sich in einer neuen Default-Szene. Sie möchte einige der bereits existierenden 3D-Objekte in der Szene löschen. Dafür selektiert sie die ungewollten Objekte durch antippen und tippt anschließend auf eine "Löschen"-Schaltfläche. Kurz darauf sind die Objekte verschwunden.

2) Als nächstes möchte Alice ein neues Objekt hinzufügen. Sie tippt auf die "Neu"-Schaltfläche und erhält eine Liste von Modellen und Lichtern, die sie hinzufügen kann. Sie entscheidet sich für einen Würfel, der unmittelbar nach der Auswahl in der Szene erscheint.

3) Der neu hinzugefügt Würfel befindet sich noch an der falschen Position. Alice tippt den Würfel an wodurch er selektiert wird und es erscheint ein Steuerelement, über das sie den Würfel verschieben, rotieren und skalieren kann. Sie verschiebt den Würfel in der Szene an die gewünschte Position.

4) Um zuschauen ob der Würfel die richtige Größe hat, wechselt sie über die Einstellungen in den AR-Modus. Auf ihrem Tablet sieht sie nun das Live-Bild ihrer Gerätekamera und die Szene, die über das Kamerabild gerendert wird. Ihr Würfel ist auch zu sehen und sie ist nun in der Lage um den Würfel herumzugehen. Sie findet, dass er noch zu klein ist und darum vergrößert sie ihn mit dem selben Werkzeug, das sie auch schon vorher für die Positionierung verwendet hat. Anschließend deaktiviert sie den AR-Modus wieder.

C) Einem Editor beitreten

1) Alice befindet sich nach wie vor im Editor. Sie ist sich aber nun bei einem Teil ihrer Konstruktion unsicher. Darum bittet sie Bob um Hilfe. Sie öffnet die Einstellungen von ihrem Editor und wählt die Option "Leute einladen" aus.

2) Es öffnet sich eine Ansicht mit einem QR-Code. Sie wird gebeten ihr Netzwerk-Interface auszuwählen, über das sie erreichbar ist, und sie wird gefragt ob sie einen Kollaborator oder eine Toolbar einladen möchte. Alice wählt das Netzwerk-Interface aus, über das Bob sie erreichen kann und sie entscheidet sich für die Kollaborator-Einladung. Der QR-Code aktualisiert sich nach der Auswahl automatisch.

3) Bob öffnet die selbe Anwendung auf seinem Tablet. Nach dem Ladebildschirm entscheidet sich Bob für die Option eine Einladung zu scannen, woraufhin sich ein QR-Code Scanner öffnet. Bob scannt den QR-Code mit der Kamera seines Tablets. Das Tablet erkennt den QR-Code und Bob wird aufgefordert seinen Namen einzugeben. Anschließend verbindet sich das Tablet von Bob automatisch mit dem Tablet von Alice.

4) Bob ist nun mit dem Editor von Alice verbunden. Er sieht die selbe Szene wie Alice und kann sie auch bearbeiten. Alice wiederum sieht die Präsenz von Bob in ihrem Editor und kann nachvollziehen, welche Objekte er bearbeitet und wo er sich gerade umschaut.

D) Eine Toolbar verbinden

1) Bob ist noch immer mit der Szene von Alice verbunden und möchte den Arbeitsbereich von seinem Tablet erweitern. Er öffnet die Einstellungen und wählt, wie Alice zuvor auch, die Option “Leute einladen” aus. In der Einladungsansicht ist bereits das richtige Netzwerk-Interface ausgewählt. Er entscheidet sich für die Toolbar-Einladung und der QR-Code wird automatisch aktualisiert.

2) Bob startet jetzt die selbe Anwendung auf seinem Smartphone. Nach dem Ladebildschirm entscheidet sich Bob für die Option eine Einladung zu scannen. Der QR-Code Scanner öffnet sich und Bob richtet sein Smartphone auf den Bildschirm von seinem Tablet. Das Smartphone von Bob erkennt die Einladung und baut eine Verbindung auf.

3) Das Smartphone von Bob ist nun als Toolbar mit dem Tablet von Bob verbunden. Es stehen ihm jetzt alle Schaltflächen im Vollbildmodus auf seinem Smartphone zur Verfügung und die Einstellungen sind synchron mit dem Editor auf seinem Tablet.

E) Eine Szene moderieren

1) Alice und Bob sind mit der Szene zufrieden und sie möchten sie nun Carl präsentieren. Dafür startet Carl die selbe Anwendung auf seinem Smartphone. Im

Startmenü entscheidet sich Carl für eine manuelle Verbindung, da die Kamera an seinem Smartphone kaputt ist.

2) Carl wird aufgefordert seinen Namen, sowie die IP-Adresse und die Portnummer des Editors anzugeben, zu dem er sich verbinden möchte. Alice öffnet bei sich die Einladungsansicht und nennt ihm die IP-Adresse und Portnummer. Anschließend verbindet sich Carl mit Alice.

3) Carl ist nun ebenfalls mit dem Editor von Alice verbunden und sieht die selbe Szene wie Alice und Bob.

4) Alice fordert Carl auf in den VR-Modus zu wechseln, damit sie ihn durch die Szene führen kann. Carl wechselt über die Einstellungen in den VR-Modus und legt sein Smartphone in ein Google-Cardboard. Mit dem Cardboard ist Carl jetzt in der Lage sich virtuell im Raum umzuschauen.

5) Alice kann die Position von Carl in ihrem Editor sehen und manipulieren. Auf diese Weise führt sie Carl nun durch die Szene.

3.2 Rollen und Anwendungsfälle

Aus dem Szenario wurde ein Use-Case-Diagramm erarbeitet (siehe Abbildung 3.1), das die wesentlichen Rollen und Anwendungsfälle für die Bereitstellung des Editors beinhaltet. Die Hauptrolle ist dabei die des Initiators. Ohne den Initiator gibt es keine Szene, zu der sich Kollaborateure verbinden könnten. Erst wenn der Initiator, der selbst auch die Rolle des Kollaborateurs inne hat, eine Szene geöffnet hat, kann er andere Kollaborateure einladen. Die Kollaborateure haben unter einander die selben Fähigkeiten. Das bedeutet z. B. auch, dass ein Kollaborateur, der über eine Einladung beigetreten ist, selbst auch Kollaborateure einladen kann.

3.3 Storyboard

Aus den Anforderungen wurde ein Storyboard ausgearbeitet, das alle notwendigen Screens und Transitionen beinhaltet, die am Ende in der Applikation enthalten sein werden (siehe Abbildung 3.2).

Sobald der Editor gestartet wird, erscheint zunächst ein Loading-Screen, gefolgt von einem kurzen Intro-Screen. Anschließend befindet sich der Anwender im Main-Screen. Von hieraus hat er die Option einen neuen Editor zu starten, sich mit einem Editor zu verbinden oder eine

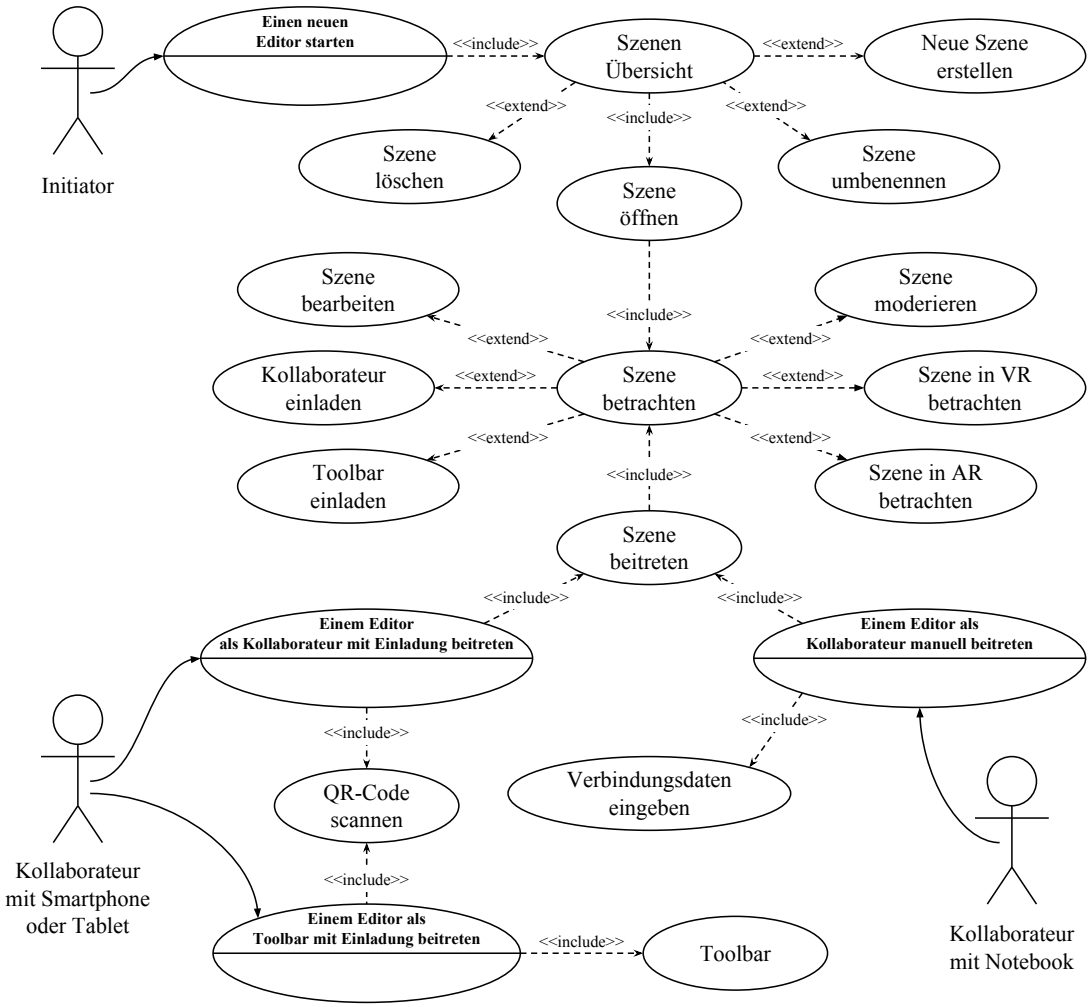


Abbildung 3.1: Use-Case-Diagramm

Einladung in der Form eines QR-Codes zu scannen. Die Möglichkeit einen QR-Code zu scannen, wird allerdings nur auf mobilen Geräten angeboten, sofern das Gerät über eine Kamera verfügt.

Entscheidet sich der Anwender dafür einen neuen Editor zu starten, wird er auf den Scene-Screen geleitet. Dort sieht er eine Liste mit allen Szenen, die sich im Moment auf seinem Gerät befinden. Er kann eine neue Default-Szene erstellen, eine Szene umbenennen oder eine Szene löschen. Wenn er sich für eine Szene entschieden hat, kann er sie öffnen und wird zum Editor-Screen geleitet.

Entscheidet sich der Anwender für die Verbindung zu einem Editor, wird er auf den Connect-Screen geleitet. Dort wird er aufgefordert die IP-Adresse und den Port des Editors anzugeben,

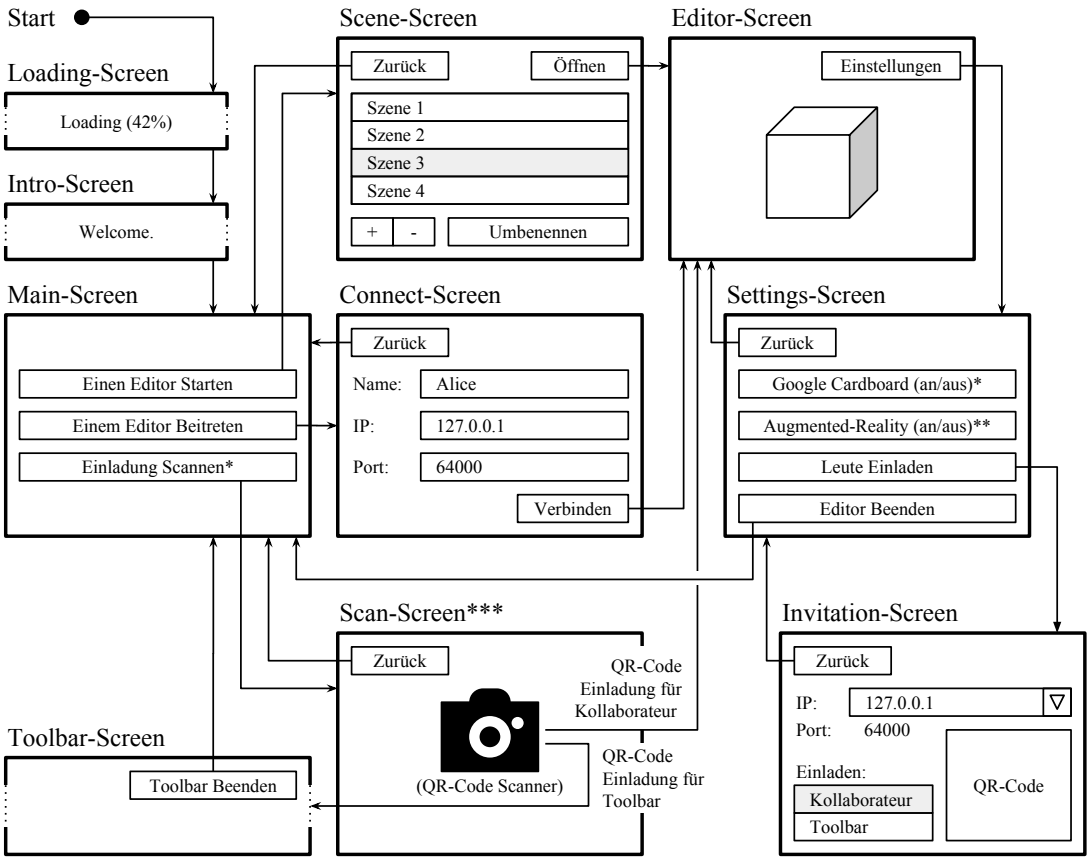


Abbildung 3.2: Das Storyboard

mit dem er sich verbinden möchte. Außerdem muss er einen Namen angeben über den er in der Szene identifiziert werden möchte. Sobald er einen Namen angegeben hat und die Verbindungsdaten korrekt sind, kann er sich mit dem Editor verbinden und gelangt ebenfalls zum Editor-Screen.

Entscheidet sich der Anwender dafür eine Einladung zu scannen, wird er auf den Scan-Screen geleitet. In diesem Screen sieht er das Live-Bild seiner Geräte-Kamera und ist nun dafür verantwortlich den QR-Code seiner Einladung zu scannen. Handelt es sich um eine korrekte Kollaborator-Einladung, wird er automatisch mit dem Editor verbunden und gelangt in den Editor-Screen. Handelt es sich um eine korrekte Toolbar-Einladung, wird er auch automatisch mit dem Editor verbunden, gelangt aber in den Toolbar-Screen. Der Scan-Screen hat noch die Besonderheit, dass er nur unter iOS existieren wird. In Android kann diese Aufgabe nämlich

an eine beliebige QR-Code Scanner App delegiert werden und in der Desktop-Anwendung wird es die Scan-Funktion gar nicht geben.

Sobald sich der Anwender nun im Editor-Screen befindet, hat er die Möglichkeit die Szene zu betrachten und zu bearbeiten. Vom Editor-Screen gelangt der Anwender nur in den Settings-Screen. Dort angekommen hat er die Option das Google-Cardboard und / oder Augmented-Reality ein- oder auszuschalten. Außerdem kann er andere Geräte zu sich in den Editor einladen und er hat auch die Möglichkeit den Editor zu beenden, wodurch er wieder im Main-Screen landet.

Wichtig beim Google-Cardboard und Augmented-Reality Feature ist, dass sich diese beiden Optionen nicht gegenseitig ausschließen. Wenn das Google-Cardboard aktiviert ist, wird die Szene im Editor durch einen Split-Screen einmal für das linke Auge und einmal für das rechte Auge gerendert. Wenn Augmented-Reality aktiviert ist, wird im Hintergrund das Live-Bild der Geräte-Kamera dargestellt und die Szene wird mit den in- und extrinsischen Parametern der Gerätekamera auf das Kamerabild gerendert. Wenn beides aktiviert ist, kann das Smartphone in Kombination mit einem Google-Cardboard als AR-Brille verwendet werden. Das Google-Cardboard steht allerdings nur für Smartphones zur Verfügung und Augmented-Reality ist nur auf mobilen Geräten der neueren Generation verfügbar.

Entscheidet sich der Anwender im Settings-Screen dafür andere Geräte einzuladen, öffnet sich der Invitation-Screen. Hier hat er die Möglichkeit den QR-Code für eine Einladung zu generieren, die von einem oder mehreren Geräten eingescannt werden kann. Er wählt dafür die Art der Einladung aus und das Netzwerk-Interface, über das er für das Gerät erreichbar ist. Die Port-Nummer ist bereits im QR-Code enthalten, aber sie wird für die Möglichkeit der manuellen Verbindung zusätzlich im Klartext dargestellt. Als Einladungsart stehen ihm die Kollaborator-Einladung und die Toolbar-Einladung zur Auswahl.

Wenn sich ein Gerät über die Toolbar-Einladung verbindet, wird es auf den Toolbar-Screen geleitet. Das eingeladene Gerät steht damit als Toolbar für das Gerät zur Verfügung, über das es eingeladen wurde. Die beiden Geräte sind nun gekoppelt und repräsentieren den selben Benutzer. Wenn sich eine Einstellung auf dem einen Gerät ändert, so ändert sie sich auch auf dem anderen Gerät und vice versa. Eine Einstellung wäre z. B. ob das Licht oder der Schatten in der Szene visualisiert werden soll.

3.4 Anforderungen

Aus dem in Abschnitt 3.1 vorgestellten Szenario sollen im folgenden die Anforderungen für die Implementation eines Prototypes abgeleitet werden.

A. 1 Die Objekte in der Szene können selektiert und deselektiert werden.

A. 2 Die Selektion eines Benutzers ist für alle anderen Benutzer sichtbar und kann einem Benutzer zugeordnet werden.

A. 3 Die Blickrichtung eines Benutzers ist für alle Benutzer sichtbar und kann einem Benutzer zugeordnet werden.

A. 4 Es können neue Objekte zur Szene hinzugefügt werden.

A. 5 Die Objekte in der Szene können entfernt werden.

A. 6 Die Objekte in der Szene lassen sich verschieben, rotieren und skalieren.

A. 7 Die Farbe von Objekten kann verändert werden.

A. 8 Der Editor erlaubt die konkurrierende Manipulation eines Objekts.

A. 9 Eine Instanz des Editors kann sich durch das Scannen eines QR-Codes mit einer anderen Editor-Instanz verbinden.

A. 10 Eine Instanz des Editors kann sich manuell mit einer anderen Editor-Instanz verbinden.

A. 11 Alle Änderungen, die an der Szene vorgenommen werden, werden kontinuierlich persistiert.

A. 12 Die Szene wird zwischen den verbundenen Geräten synchronisiert.

A. 13 Die Szene kann mit einer AR-Brille virtuell betreten werden.

A. 14 Die Szene kann mit einer VR-Brille virtuell betreten werden.

A. 15 Die Position einer VR-Brille kann durch alle Benutzer verändert werden.

A. 16 Der Editor unterstützt Licht und Schatten.

3.5 Zusammenfassung

In diesem Kapitel wurde ein detailliertes Szenario vorgestellt, das auf der Vision aus dem ersten Kapitel aufbaut und als Grundlage für die Definition der Rollen und Anwendungsfälle verwendet wurde. Es wurde außerdem ein Storyboard erarbeitet und es wurden die Anforderungen identifiziert, die im nächsten Kapitel an den Prototypen gestellt werden.

4 Konzept

Zu Beginn dieser Arbeit wurde eine Vision vorgestellt, in der unterschiedliche Akteure kollaborativ in einer Mixed-Reality Umgebung eine Szene gestalten. Diese Vision wurde im vorherigen Kapitel durch ein Szenario weitergedacht. Anhand des Szenarios wurden die Rollen und Anwendungsfälle, sowie die Anforderungen, die durch den Editor erfüllt werden müssen, identifiziert. In diesem Kapitel wird nun anhand eines entwickelten Prototypen erläutert, mit welchem Konzept der Prototyp realisiert wurde, um den Anforderungen gerecht zu werden.

Um den Aufbau des Prototypen besser zu verstehen, werden durch eine Modul-Übersicht als erstes die vier wesentlichen Bestandteile des Editors vorgestellt. Die einzelnen Module werden im Anschluss noch näher betrachtet. Abschnitt 4.1.1 widmet sich daher dem User-Interface-Modul, Abschnitt 4.1.3 dem Network-Middleware-Modul, Abschnitt 4.1.2 dem Cross-Platform-Modul und Abschnitt 4.2 dem ECS-Modul.

Nach der Vorstellung aller Module werden in Abschnitt 4.3 die Editor-Entities vorgestellt, die den Zustand beschreiben in dem sich der Editor und die darin enthaltene Szene befinden. Anschließend werden im Abschnitt 4.4 alle Operationen beschrieben, die auf den Editor-Entities durchgeführt werden können.

4.1 Modul Übersicht

Die Editor-Application besteht aus vier Modulen, die alle Zugriff haben auf einen Logger und einen Asset-Manager (Assets). Der Logger wird von den Modulen verwendet, um Debug-Informationen zu protokollieren. Über die Assets werden unteranderem Texturen, Models, Fonts und der Skin für das User-Interface geladen und bereitgestellt.

Die Editor-Application ist das Bindeglied zwischen den Modulen und verantwortlich für den Wechsel der Screens aus dem User-Interface-Modul, sowie für das Starten und Stoppen der Engines aus dem ECS-Modul.

Für die Umsetzung des Editors wurde das libGDX[1] Framework verwendet. Hierbei handelt es sich um ein Java-Game-Development-Framework, das auf OpenGL basiert. Es wurde von Mario

[1] https://libgdx.badlogicgames.com/

Zechner[2] für die plattformunabhängige Spiele-Entwicklung geschaffen und wird als Open Source Projekt auf GitHub[3] zur Verfügung gestellt. Damit die Editor-Application durch libGDX auch auf anderen Plattformen bereitgestellt werden kann, muss sie das Application-Listener Interface des Frameworks implementieren (siehe Abbildung 4.1).

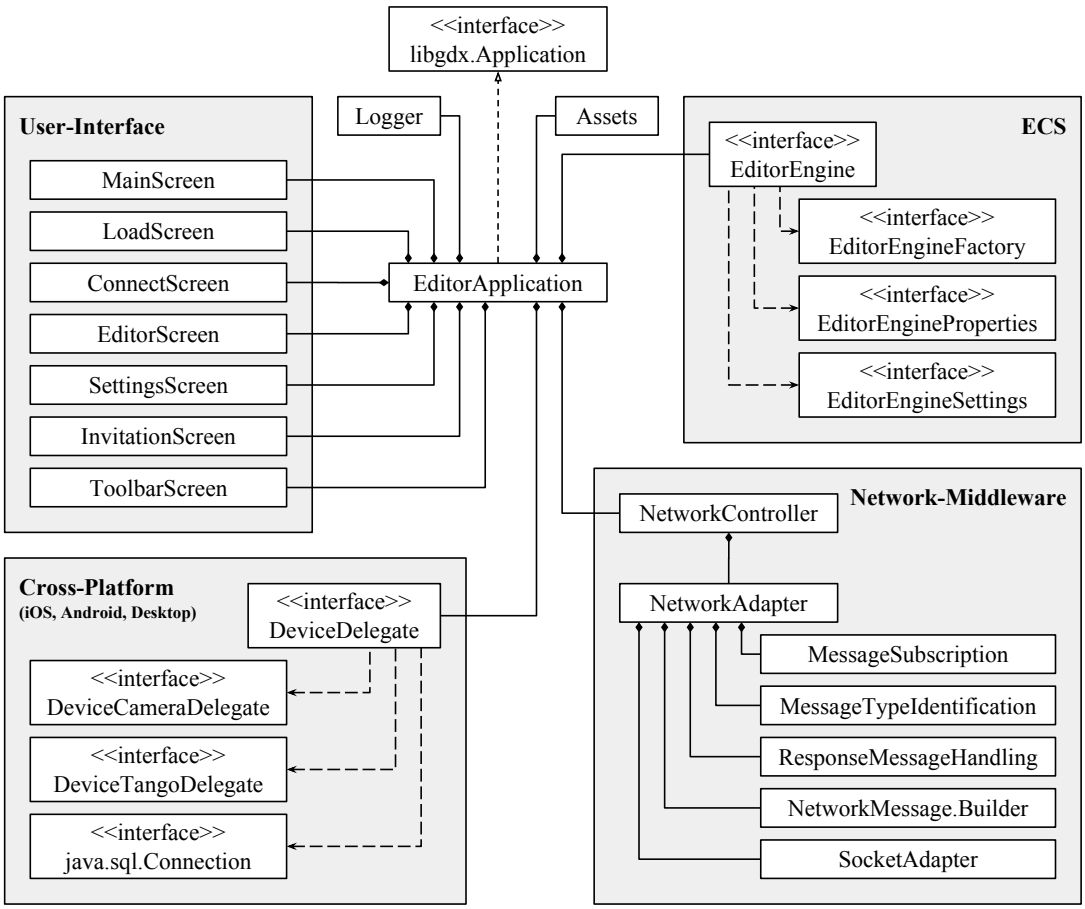


Abbildung 4.1: Modul Übersicht

Das größte Modul ist das ECS-Modul, in dem sich unterschiedliche Engines befinden, die alle das Editor-Engine Interface implementiert haben. Die Aufgaben des ECS-Moduls sind die Kommunikation und Synchronisation von allen an einer Szene beteiligten Editor-Engines, sowie die lokale Visualisierung der aktiven Szene. Die unterschiedlichen Engines werden im Abschnitt 4.2.4 vorgestellt.

[2] https://twitter.com/badlogicgames?lang=de
[3] https://github.com/libgdx/libgdx

Das zweit größte Modul ist die Network-Middleware. Sie beinhaltet das Serialisieren, Deserialisieren, Versenden und Empfangen von Network-Messages, sowie die Verwaltung von TCP-Verbindungen.

Das kleinste Modul ist das Plattform-Modul. Dieses Modul existiert jedoch für jede unterstützte Plattform genau einmal. Unterstützt werden die Plattformen: Android, iOS und Desktop. Es beinhaltet die systemabhängige Implementation, wie bspw. den Datenbank- oder Kamerazugriff.

Das letzte Modul ist das User-Interface-Modul. Es beinhaltet alle Screens, die bereits im Storyboard des vorherigen Kapitels skizziert wurden.

4.1.1 User-Interface

Für die Umsetzung der Screens aus dem Storyboard wurde das Scene2D[4] Package aus dem libGDX Framework verwendet, in dem Standard UI-Elemente enthalten sind, wie z. B. Windows, Dialogs, Labels, Buttons und Listen. Der verwendete UI-Skin stammt von Raymond Buckley[5]. Die Screens wurden mit einem Table-Layout[6] gestaltet, das sich automatisch an die Bildschirmgröße anpasst.

Der Editor-Screen

Der Editor-Screen (siehe Abbildung 4.2) stellt einige UI-Elemente zur Verfügung, über die mit den selektierten Editor-Entities interagiert werden kann. Es ist aber auch möglich neue Objekte hinzuzufügen. Des weiteren kann das Rendering der Szene beeinflusst werden.

Der Invitation-Screen

Über den Invitation-Screen (siehe Abbildung 4.3) können QR-Codes generiert werden, die von anderen Geräten eingescannt werden können. Der QR-Code enthält die IP-Adresse und die Portnummer der Server-Engine, sowie die Art der Einladung. Es können weitere Kollaborateure oder eine Toolbar eingeladen werden.

Der Toolbar-Screen

Der Toolbar-Screen (siehe Abbildung 4.4) ermöglicht die Fernsteuerung einer Client-Engine. Er stellt dabei die selben UI-Elemente bereit, wie sie auch im Editor-Screen zur Verfügung

[4] https://github.com/libgdx/libgdx/wiki/Scene2d
[5] https://ray3k.wordpress.com/artwork/shade-ui-skin-for-libgdx/
[6] https://github.com/libgdx/libgdx/wiki/Table

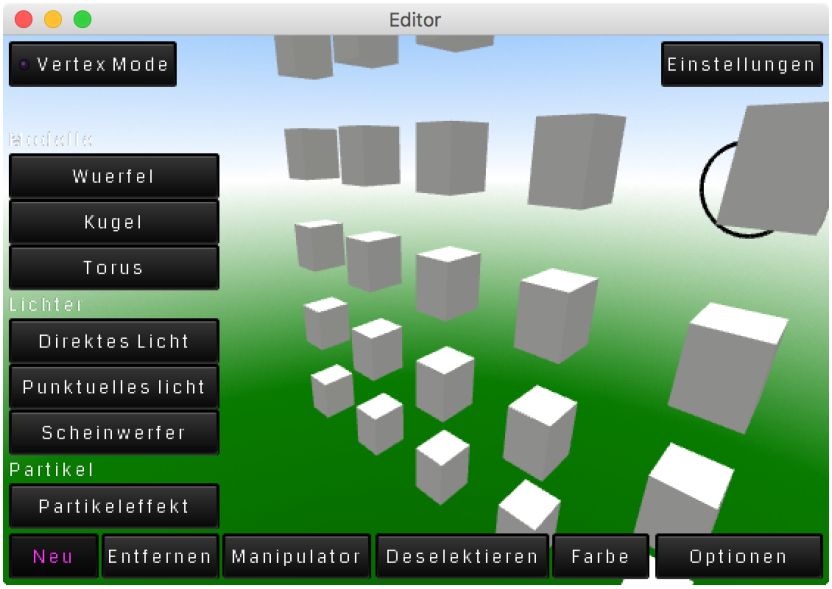


Abbildung 4.2: Der Editor-Screen

Abbildung 4.3: Der Invitation-Screen

stehen, jedoch mit dem Unterschied, dass nicht erst ein Menü geöffnet werden muss und dass der gesamte Screen für die Darstellung genutzt wird.

Zukünftig könnte die Toolbar zu einer AR-Fernbedienung ausgebaut werden. Der Träger einer AR-Brille hätte dann die Möglichkeit über ein Smartphone, das er als Toolbar verwendet, Touch-Gesten an seine AR-Brille zu senden. Die Gesten könnten dann auf die Objekte angewandt werden, die der Benutzer in der AR-Brille fokussiert.

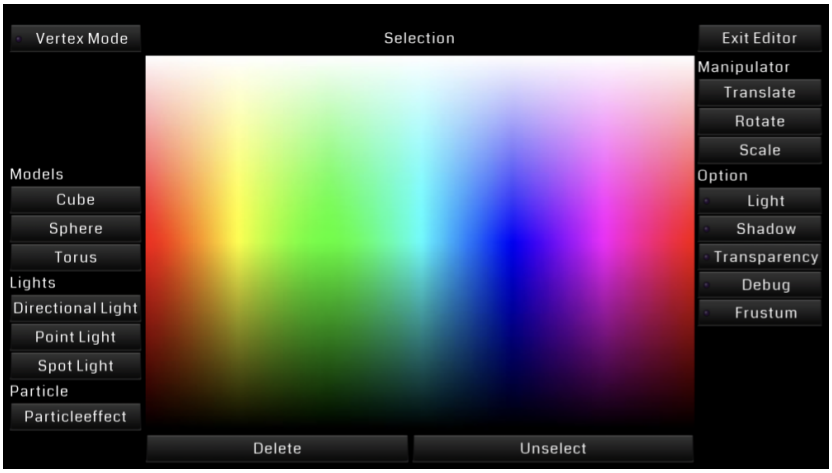


Abbildung 4.4: Der Toolbar-Screen

4.1.2 Cross-Platform

Für die Plattformunabhängigkeit kann ein Großteil des Codes übergeordnet verwendet werden. Jedoch gehen die verschiedenen Geräte und Betriebssysteme mit Besonderheiten einher, für welche spezifische Implementationen benötigt werden.

Das libGDX Framework sieht vor, dass für jede Plattform, die unterstützt werden soll, eine Starter-Klasse implementiert werden muss. Diese Starter-Klasse instanziiert eine plattformspezifische Implementierung einer libGDX Backend-Application. Für die Instanziierung der jeweiligen Backend-Application wird die Instanz einer plattformspezifischen Backend-Configuration und die Instanz einer plattformunabhängigen Anwendung, die Editor-Application, benötigt.

Die Editor-Application wiederum benötigt für die Instanziierung selbst noch eine Instanz, die das Device-Delegate Interface implementiert. Dieses Interface wurde für den Prototypen definiert, um innerhalb der Editor-Application Zugriff auf die Gerätekamera, Project Tango und eine Datenbank zu erhalten.

Für die Bereitstellung auf iOS Geräten wird zusätzlich ein Ahead-of-time-Compiler benötigt, der den Java Code in nativen Objective-C Code konvertiert. Bei dieser Arbeit wurde für diese Aufgabe die Multi-OS Engine[7] von Intel verwendet.

Die Multi-OS Engine stellt außerdem einen Großteil der iOS Frameworks bereit, unteranderem auch das ARKit, das ohne all zu große Probleme in den Editor integriert werden konnte. Lediglich der Zugriff auf die intrinsischen und extrinsischen Parameter des ARCamera-Objekts war ein Problem, das nur durch einen unsauberen Workaround[8] behoben werden konnte. Es fehlten zum Zeitpunkt der Implementation schlicht und ergreifend die notwendigen Properties im ARCamera-Objekt. Glücklicherweise enthielt die To-String-Methode des ARCamera-Objekts alle notwendigen Parameter und so konnten sie von dort extrahiert werden. Diese Lösung ist allerdings nicht befriedigend, da die Parameter pro Frame extrahiert werden müssen.

4.1.3 Network-Middleware

Die Kommunikation zwischen den Geräten erfolgt über eine Network-Middleware, die binärcodierte Messages verschickt. Das Konzept der Network-Middleware beruht auf der Arbeit von Malte Eckhoff [Eckhoff (2017)] und wurde im Rahmen dieser Arbeit und während eines Gemeinschaftsprojekts der i^2e Arbeitsgruppe [Blank u. a. (2015)] an der HAW in Java implementiert.

Alle Nachrichten bzw. Messages, die über die Network-Middleware ausgetauscht werden können, sind in Form von automatisch generierten Klassen zum Zeitpunkt der Kompilierung bekannt. Für die Message Generierung wurde Googles Protocol Buffer[9] (Protobuf) verwendet.

Die offizielle Protobuf Implementation stellte sich für mobile Geräte mit begrenzter Heap-Size leider als ungünstig heraus. Da sie nur mit unveränderbaren Nachrichten arbeitet, konnte kein Nachrichten-Pool realisiert werden[10]. Dies führt durch zu große Nachrichten zu einem schnell fragmentierten Heap und zwingt den Garbage-Collector zum regelmässigen Abräumen der verbrauchten Messages. Eine mögliche Lösung wäre die Verwendung einer anderen Protobuf Implementation[11].

Durch die zusätzliche Verwendung des Ahead-of-time-Compilers der Multi-OS Engine konnten die generierten Klassen und die gesamte Java Implementation der Network-Middleware auch zu einer äquivalenten Objective-C Implementation überführt werden. Hierdurch ist es nun möglich Messages zwischen iOS, Android und Desktop Geräten auszutauschen.

[7] https://software.intel.com/en-us/multi-os-engine
[8] https://discuss.multi-os-engine.org/t/arcamera-is-missing-properties/1268/4
[9] https://developers.google.com/protocol-buffers/
[10] http://stackoverflow.com/questions/25138361/pooling-protobuf-pojos
[11] https://github.com/google/protobuf/wiki/Third-Party-Add-ons

Messages

Nach dem Aufbau einer TCP-Connection kommunizieren die unterschiedlichen Editor-Instanzen ausschließlich über Protobuf-Messages. Dabei handelt es sich um eine Plattform- und Sprach-neutrale Serialisierung von strukturierten Daten. Die benötigten Messages werden mit einer DSL beschrieben (siehe Listing 4.1). Aus den so definierten Messages kann anschließend der Source-Code generiert werden, der benötigt wird, um die beschriebene Nachricht zu serialisieren bzw. wieder zu deserialisieren. Der Vorteil gegenüber dem JSON- oder XML-Format ist der geringe Overhead in den Nachrichten, da nur die Werte und nicht die Beschreibungen der Parameter verschickt werden. Dafür sind Protobuf-Messages weniger flexibel, da bei jeder Änderung die Message-Definition geändert und der Source-Code neu generiert werden muss. Außerdem kommt es eher zu einer Inkompatibilität, wenn eine neue Version der Applikation mit einer älteren Version kommuniziert.

Network-Message Alle Messages, die verschickt und empfangen werden, sind als Payload in einer Network-Message eingebettet (siehe Listing 4.1). Neben dem Payload enthält die Network-Message außerdem ein Response-Flag, eine Transaction-ID und den Message-Type der Message, die sich im Payload befindet. Das Response-Flag wird bei Nachrichten gesetzt, bei denen der Sender vom Empfänger eine Antwort erwartet. Die Antwort des Empfängers muss die selbe eindeutige Transaction-ID enthalten, damit sie als Antwort zugeordnet werden kann. Der Message-Type wird beim Empfänger für die Message-Deserialisierung des Payloads benötigt.

```
1  syntax = "proto2";
2
3  option java_package = "de.raimundwege.editor.network.messages";
4  option java_outer_classname = "NetworkMessageOuter";
5
6  message NetworkMessage {
7     optional bool isAnswerRequired = 1 [default = false];
8     optional bytes transactionId = 2;
9     required int32 messageTypeId = 3;
10    required bytes payloadMessage = 4;
11 }
```

Listing 4.1: Die Network-Message dient als Container für die Übertragung von anderen Messages.

Response-Message Hierbei handelt es sich um eine Message, die von vielen unterschiedlichen Requests als Response verwendet wird. Sie enthält ein Success-Flag, das angibt ob der jeweilige Request erfolgreich gewesen ist und eine optionale Text-Nachricht, die innerhalb der Anwendung häufig für Dialoge Verwendung findet, insbesondere dann, wenn der Request abgelehnt wurde.

Client-Register- / Client-Unregister-Message Die Client-Register-Message enthält genau eine Client-Message mit der ID, dem Namen, der Startposition in der Szene, der Selektionsfarbe und weiteren Editor-Einstellungen des Clients, der sich verbinden möchte. Die Client-Unregister-Message enthält nur die ID des Clients und gibt damit an, welcher Client sich abmelden möchte.

Client-Synchronize-Request- / Client-Synchronize-Response-Message Mit einer Client-Synchronize-Request-Message signalisiert die Client-Engine der Server-Engine, dass er bereit ist für eine Synchronisation. Die Client-Engine wartet dann auf die Client-Synchronize-Response-Message von der Server-Engine. In der Response-Message befindet sich der aktuelle Stand des Editors in Form einer serialisierten Datenbank und die Sequenznummer des letzten Updates, das in die Datenbank geschrieben wurde. Nachdem die Client-Engine die Datenbank geladen hat, synchronisiert sich die Client-Engine nur noch durch den Empfang von Update-Request-Messages (siehe Abschnitt 4.1.3).

Client-Relation-Message Mit der Client-Relation-Message kann eine Beziehung zwischen zwei Client-Engines hergestellt werden. Die Message enthält eine Parent-Client-ID und eine Child-Client-ID, sowie einen Relation-Type, der die Beziehung zwischen den beiden Client-Engines beschreibt. Aktuell gibt es jedoch nur den *Toolbar* Relation-Type über den eine Client-Engine mit einer Toolbar-Engine verbunden werden kann.

Select- / Unselect-Message Die Select- bzw. Unselect-Messages enthalten die ID der Client-Engine, die ihre Selektion updaten möchte und eine Liste von Editor-Entity-IDs, die selektiert bzw. deren Selektion aufgehoben werden soll.

Add-Message Die Add-Message wird verwendet, um neue Editor-Entities hinzuzufügen. Sie enthält die ID der Client-Engine, die etwas hinzufügen möchte und eine Anzahl von Listen mit weiteren Messages, die festlegen welche Entities hinzugefügt werden sollen. Jede Liste steht für eine bestimmte Entity-Family und eine leere Liste bedeutet, dass keine Entity dieser Family

hinzugefügt werden soll. Es gibt eine Liste für Model-, Particle-Effect-, Directional-Light-, Point-Light- und Spot-Light-Messages.

Delete-Message Die Delete-Message wird verwendet, um existierende Editor-Entities zu entfernen. Sie enthält die ID der Client-Engine, die etwas entfernen möchte und eine Liste der Editor-Entity-IDs, die entfernt werden sollen.

Manipulate-Message Die Manipulate-Message wird verwendet, um existierende Editor-Entities zu manipulieren bzw. zu transformieren. Sie enthält die ID der Client-Engine, die etwas manipulieren möchte, eine Liste der Editor-Entity-IDs, die manipuliert werden sollen, sowie eine Translation, Rotation und Skalierung.

Properties-Message Mit Hilfe der Properties-Message können Eigenschaften bzw. Property-Components von Editor-Entities angepasst werden. Sie enthält die ID der Client-Engine, die Anpassungen vornehmen möchte, eine Liste der Editor-Entity-IDs, deren Property-Components angepasst werden sollen, und eine Reihe von optionalen Eigenschaften die aktualisiert werden sollen. Eine Property-Component ist z. B. die Color-Component, die von der Client-, Vertex-, Directional-Light-, Point-Light- und Spot-Light-Entity verwendet wird.

Update-Request- / Update-Response-Message Die Update-Request-Messages werden von der Server-Engine generiert und mit einer aufsteigenden Sequenznummer versehen. Der Request enthält alle Messages, die während einer System-Iteration von Server-Engine akzeptiert wurden. Die Client-Engines starten mit der Sequenznummer, die sie bei der Synchronisation erhalten haben. Mit der Update-Response-Message können die Client-Engines der Server-Engine signalisieren, dass sie bereit sind ein neues Update zu empfangen und welche Sequenznummer sie als nächstes erwarten.

Camera- / Camera-Summary-Message Jede Client-Engine sendet pro Sekunde eine Camera-Message an die Server-Engine. In der Camera-Message ist die aktuelle Model-View- und die Inverse-Projection-View-Matrix der virtuellen Kamera der Client-Engine enthalten. Mit diesen Informationen können bspw. die Blickrichtung und der sichtbare Bereich einer Client-Engine visualisiert werden. Damit alle Client-Engines diese Informationen erhalten, sendet die Server-Engine pro Sekunde eine Camera-Summary-Message an alle Client-Engines. Die Camera-Summary-Message enthält die aktuellste Camera-Message von jeder Client-Engine. Noch werden diese Nachrichten über die TCP-Connection übertragen, doch zukünftig sollte

die Client- und Server-Engine auf UDP umgestellt werden, da diese Informationen unkritisch sind und die Konstruktion der Szene nicht beeinflussen.

4.2 Entity-Component-System

Als Architekturpattern wurde das Entity-Component-System (ECS) verwendet, welches oft bei Computerspielen Verwendung findet Hall u. a. (2014). Dieses Pattern setzt auf Komposition an Stelle von Vererbung. Eine Szene im Editor besteht bspw. aus Modellen, die vom Anwender hinzugefügt, verändert und entfernt werden können. Ein Modell wiederum besteht aus Punkten (Vertices), Linien bzw. Kanten (Edges) und Flächen (Faces). Jedes dieser Objekte wird selbst durch eine Entity repräsentiert und eine Entity setzt sich zusammen aus Components. Wichtige Components sind bspw. die Position-, Rotation-, Scale-, Color- und Editor-Component. Neben den Components und Entities gibt es noch Systeme. Ein System iteriert pro Interval in der Regel über eine bestimmte Gruppe von Entities. Eine solche Gruppe wird auch als Entity-Family bezeichnet. Das Rendering-System erwartet bspw. Entities, die sowohl eine Position- und mindestens eine Model-, Vertex-, Edge- oder Face-Component besitzen. In der jetzigen Implementation gibt es für jede Aufgabe ein eigenes System, das je nach Anforderung aktiviert und deaktiviert werden kann.

4.2.1 Kommunikation

Für die Übermittlung von Nachrichten wurden unterschiedliche Request-Systeme implementiert, die mit einem entsprechenden Response-System einer anderen Editor-Instanz kommunizieren. Jedes Request- und Response-System ist auf genau eine Request- und Response-Message spezialisiert. Das Request-System verschickt Request-Messages und erwartet entsprechende Response-Messages. Das Response-System wiederum empfängt, validiert und beantwortet Request-Messages mit einer Response-Message. Das Select-Request-System eines Clients verschickt bspw. ausschließlich Select-Request-Messages an die Server-Engine. Das Select-Response-System der Server-Engine wiederum verarbeitet alle Select-Request-Messages und informiert das Select-Request-System der jeweiligen Client-Engine durch Select-Response-Messages über den Erfolg oder den Misserfolg der Selektion. Sollte ein Request keinen Response erhalten oder der Response zu spät eintreffen, wird durch einen Response-Timeout ein negativer Ausgang der Aktion angenommen und die Aktion wird bei der Client-Engine verworfen. Tritt der Fall ein, dass bspw. der Select-Request erfolgreich gewesen ist, der Select-Response aber nicht angekommen ist, wird bei der Client-Engine die Änderung zwar verworfen, aber

durch den Empfang der Update-Request-Messages wird die Client-Engine über den Erfolg informiert.

Das Versenden von Request- und Response-Messages erfolgt im jeweiligen System direkt über den Network-Controller. Die Messages, die durch den Network-Controller empfangen werden, werden hingegen als eigenständige Entities zur Engine hinzugefügt und durch eine entsprechende Message-Component ihres Nachrichten-Types gekennzeichnet (siehe Listing 4.2). Dadurch lässt sich jede Art von Request- und Response-Message einer bestimmten Family bzw. einem bestimmten System zuordnen und bei der nächsten System-Iteration abarbeiten.

```
public Entity createSelectMessageEntity(
   SelectMessageOuter.SelectMessage message,
   ByteString transactionId
) {
   Entity entity = new Entity();
   entity.add(new SelectMessageComponent(message,
      transactionId));
   return entity;
}
```

Listing 4.2: Fabrikmethode für die Erzeugung einer Select-Message-Entity.

```
public class SelectMessageComponent implements Component {

   public final SelectMessageOuter.SelectMessage message;
   public final ByteString transactionId;

   public SelectMessageComponent(
         SelectMessageOuter.SelectMessage message,
         ByteString transactionId
   ) {
      this.message = message;
      this.transactionId = transactionId;
   }
}
```

Listing 4.3: Die Select-Message-Component enthält die Protobuf Message und die Transaction-ID.

Die Request-Systeme verarbeiten in erster Linie den Input des Benutzers und generieren daraus Request-Messages, die sie anschließend verschicken. Die Response-Systeme validieren

die empfangenen Request-Messages, die ihrer Family zugeordnet sind, und informieren den Absender durch eine Response-Message über den Erfolg oder den Misserfolg der Aktion. Invalide Request-Messages werden mit einem negativen Response beantwortet und anschließend einfach verworfen. Valide Request-Messages werden mit einem positiven Response beantwortet und mit einer *ACCEPT*-Marker-Component aus der entsprechenden Family markiert. Auf diese Weise werden während einer System-Iteration alle akzeptierten Requests gesammelt.

4.2.2 Synchronisation

Die initiale Synchronisation wird durch die Client-Engine über die Client-Synchronize-Request-Message angefordert, worauf die Server-Engine mit einer Client-Synchronize-Response-Message reagiert (siehe Abschnitt 4.1.3). In der Response-Message ist der aktuelle Stand der Szene in der Form einer serialisierten SQLite[12]-Datenbank enthalten (siehe Abbildung 4.5). Außerdem enthält sie die Sequenznummer des letzten Updates, das in die übermittelte Datenbank geschrieben wurde. Nach dem Deserialisieren der Datenbank, lädt die Client-Engine alle Entities aus der Datenbank und merkt sich die Sequenznummer. Die Client-Engine ist damit initial synchronisiert und wartet auf das Update mit der Sequenznummer: $Sequenznummer + 1$.

Der Austausch einer ganzen Datenbank erfolgt nur bei der initialen Synchronisation. Jede weitere Veränderung an der Szene wird durch die Server-Engine mittels Update-Request-Messages (siehe Abschnitt 4.1.3) an die Client-Engines propagiert und die Client-Engines müssen den Erhalt bestätigen. Diese Messages haben eine eindeutige Sequenznummer und sie beinhalten alle Änderungen, die während einer System-Iteration der Server-Engine akzeptiert wurden. Die Sequenznummer garantiert, dass die Client-Engines die Änderungen in der selben Reihenfolge ausführen, wie die Server-Engine. Da Transformationen wie z. B. die Translation und die Rotation nicht kommutativ sind, befinden sich auch innerhalb eines Updates alle Änderungen in der selben Reihenfolge, wie sie von der Server-Engine während einer System-Iteration akzeptiert wurden. Wenn keine Änderungen akzeptiert wurden, bereitet die Server-Engine auch keine Update-Message vor. Und wenn eine Update-Message nicht die Sequenznummer enthält, die eine Client-Engine erwartet, hat die Client-Engine in der Update-Response-Message die Möglichkeit die Server-Engine auf die erwartete Sequenznummer hinzuweisen.

Die Update-Request-Messages werden im Update-Request-System nach allen zustandsverändernden Systemen, aber noch vor dem Save-System, generiert (siehe Abbildung 4.9). Die zustandsverändernden Systeme markieren zuvor alle Message-Entities, die akzeptiert wurden, mit einer *Accept*-Marker-Component. Das Update-Request-System kann dann über

[12] https://www.sqlite.org/

alle akzeptierten Message-Entities iterieren und sie der Update-Request-Message hinzufügen. Die akzeptierten Protobuf-Messages, die sich in den Message-Entities befinden, werden auf diese Weise recycled. Jede Update-Request-Message erhält außerdem eine aufsteigende und eindeutige Sequenznummer, auf deren Funktion bereits eingegangen wurde.

Nach der Generierung wird die Update-Request-Message zu einer Warteschlange hinzugefügt und es werden alle akzeptierten Message-Entities gelöscht. Das Update-Request-System iteriert zum Schluss über alle Client-Entities und versorgt die entsprechenden Client-Engines individuell mit den nächsten Update-Request-Messages aus der Warteschlange. Reagiert eine Client-Engine über einen längeren Zeitraum nicht mit einer Update-Response-Message, wird die Client-Engine zwangsgetrennt.

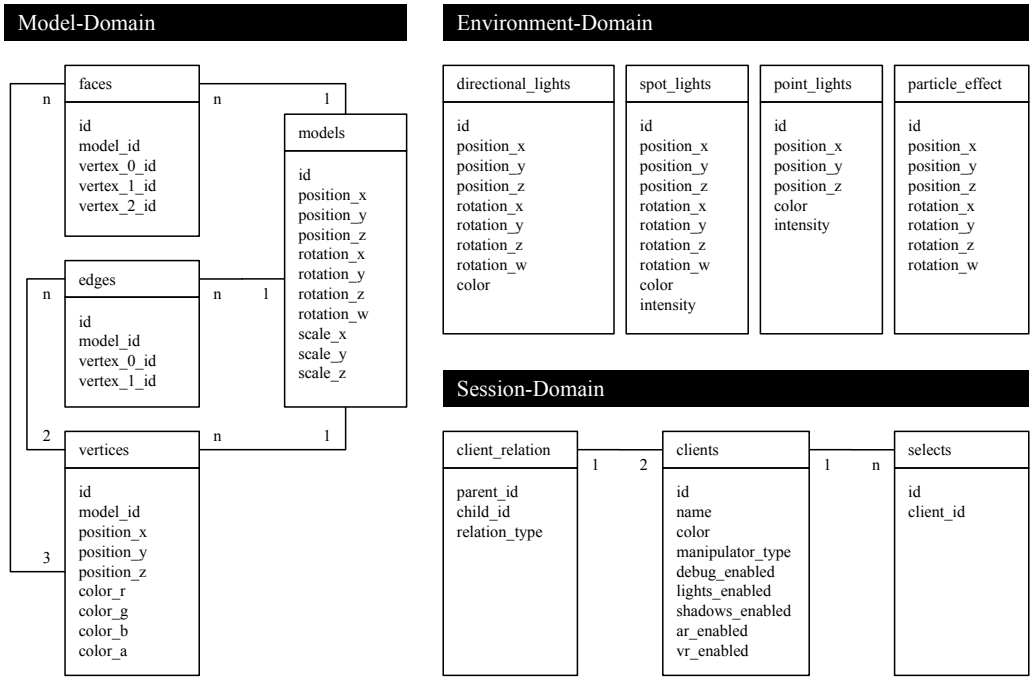


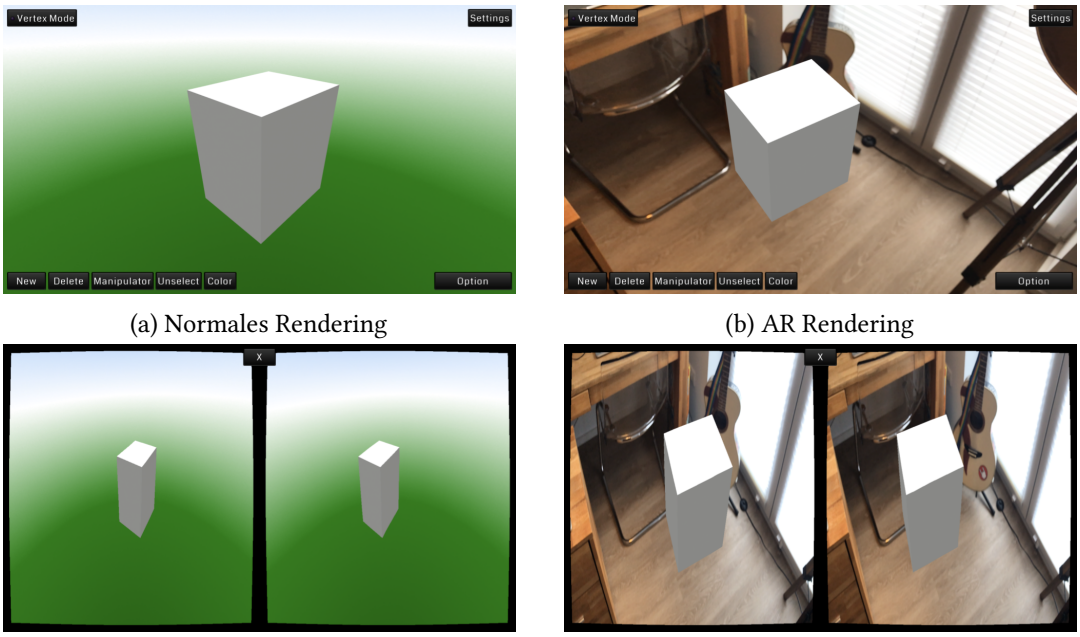
Abbildung 4.5: Das Datenbank-Schema, das zum Persistieren und für die initiale Synchronisation eines Clients genutzt wird.

4.2.3 Rendering

Für das Rendern der Szene wurde ein eigener Renderer entwickelt, der durch ein Rendering-System gesteuert wird. Der Renderer stellt dem Rendering-System Listen zur Verfügung, über die bestimmt werden kann, welche Editor-Entities pro Frame gerendert werden sollen. Es gibt

für jede Art von Editor-Entity eine eigene Liste. Das Rendering-System ist darauf ausgelegt, dass in den Listen nur die Editor-Entities enthalten sind, die im Frustum der virtuellen Kamera liegen.

Der Renderer stellt eine Methode bereit, um die Szene in den gerade aktiven Framebuffer zu rendern. Neben dem Rendering der gesamten Szene, unterstützt der Renderer auch das Rendern der Shadow-Map (siehe Abbildung 4.7c) und der Selection-Map (siehe Abbildung 4.12a). Welcher Framebuffer dabei gerade aktiv ist, entscheidet das Rendering-System. Auf diese Weise kann auch direkt in eine Textur gerendert werden, die in einem weiteren Rendering verwendet werden kann. Dies ist z. B. erforderlich für das Shadow-Map Rendering, das Selection-Map Rendering oder das Google-Cardboard Rendering (siehe Abbildung 4.6c), bei dem die Szene unter Berücksichtigung des Pupillenabstands einmal für das linke Auge und einmal für das rechte Auge gerendert wird.



(a) Normales Rendering

(b) AR Rendering

(c) VR Rendering

(d) MR Rendering

Abbildung 4.6: Die Renderings

Camera-Controller

Der Camera-Controller verwaltet eine orthografische und eine perspektivische Kamera, sowie eine Pinhole-Kamera. Durch den Camera-Controller wird innerhalb der Applikation festgelegt,

welche Kamera gerade aktiv für das Rendering der Szene zuständig ist. Die orthografische und die perspektivische Kamera sind bereits im libGDX Framework enthalten und werden für alle rein virtuellen Szenen verwendet. Bei der Pinhole-Kamera handelt es sich jedoch um eine neue Kameraklasse, die die Geräte-Kamera in der Virtualität abbildet, damit virtuelle Objekte auf das reale Kamerabild gerendert werden können. Die Projection-Matrix der Pinhole-Kamera wird durch die intrinsischen Parameter bestimmt und die View-Matrix durch die extrinsischen Parameter der Gerätekamera. Die Parameter erhält der Camera-Controller hierfür von der Device-Camera-Delegate der Device-Delegate.

Grafik-Shader

Während dieser Thesis wurden diverse Grafik-Shader benötigt. Für die Darstellung der VR-Glasses wurde ein Barrel-Shader entwickelt um die Linsenkrümmung des Google-Cardboards auszugleichen und ein Blur-Shader um einen Unschärfe-Effekt im äußeren Sichtfeld zu erzeugen. Der Blur-Shader wird außerdem für Überblend-Effekte verwendet.

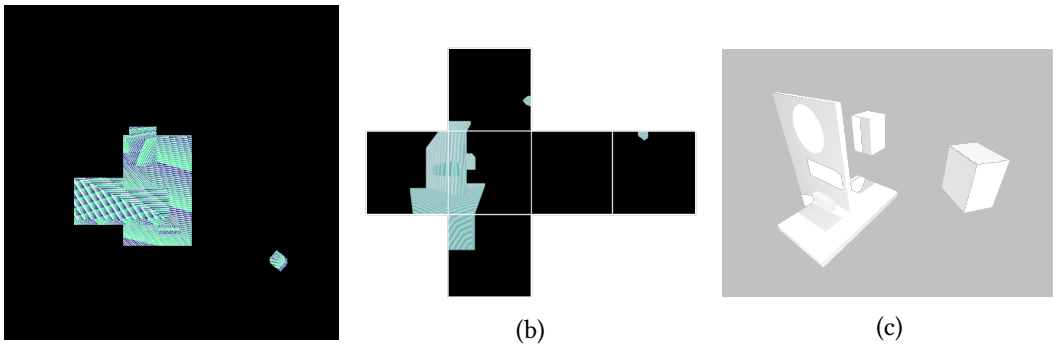
Ein sehr einfacher Shader wurde für Android benötigt, um das Bild der Geräte-Kamera zu rendern. Das Kamerabild konnte in Android direkt über eine OpenGL Extension[13] an eine Textur gebunden werden, ohne dass das Kamerabild erst noch über das YUV-Farbmodell[14] in den RGB-Farbraum[15] konvertiert werden musste. In iOS hingegen gibt es diese Extension nicht und die Umrechnung musste innerhalb eines Shaders durchgeführt werden.

Damit in der Szene Schatten angezeigt werden kann, mussten drei weitere Shader implementiert werden. Der erste Shader rendert ein Tiefenbild aus der Sicht der jeweiligen Lichtquelle. Für ein direktes Licht reicht ein Tiefenbild aus (siehe Abbildung 4.7a). Für ein Punktlicht hingegen, das in alle Richtungen strahlt, werden sechs Tiefenbilder in der Form einer Cube-Map benötigt (siehe Abbildung 4.7b). Der zweite Shader fast alle Schatten zu einer Shadow-Map zusammen (siehe Abbildung 4.7c) und der dritte Shader verwendet die Shadow-Map für das finale Rendering der Szene (siehe Abbildung 4.10).

4.2.4 Engine Übersicht

Es gibt insgesamt vier verschiedene Editor-Engines: die Default-Engine, die Server-Engine, die Client-Engine und die Toolbar-Engine. Die Engines können zur Laufzeit gewechselt werden, je nachdem welche Anforderungen gerade erfüllt werden müssen. Es existiert aber immer

[13] https://www.khronos.org/registry/OpenGL/extensions/OES/OES_EGL_image_external.txt
[14] https://de.wikipedia.org/wiki/YUV-Farbmodell
[15] https://de.wikipedia.org/wiki/RGB-Farbraum



(a) (b) (c)

Abbildung 4.7: Die Shadow-Map

nur eine Engine zur Zeit. Eine Ausnahme ist die Server-Engine, denn sie beinhaltet auch eine Client-Engine.

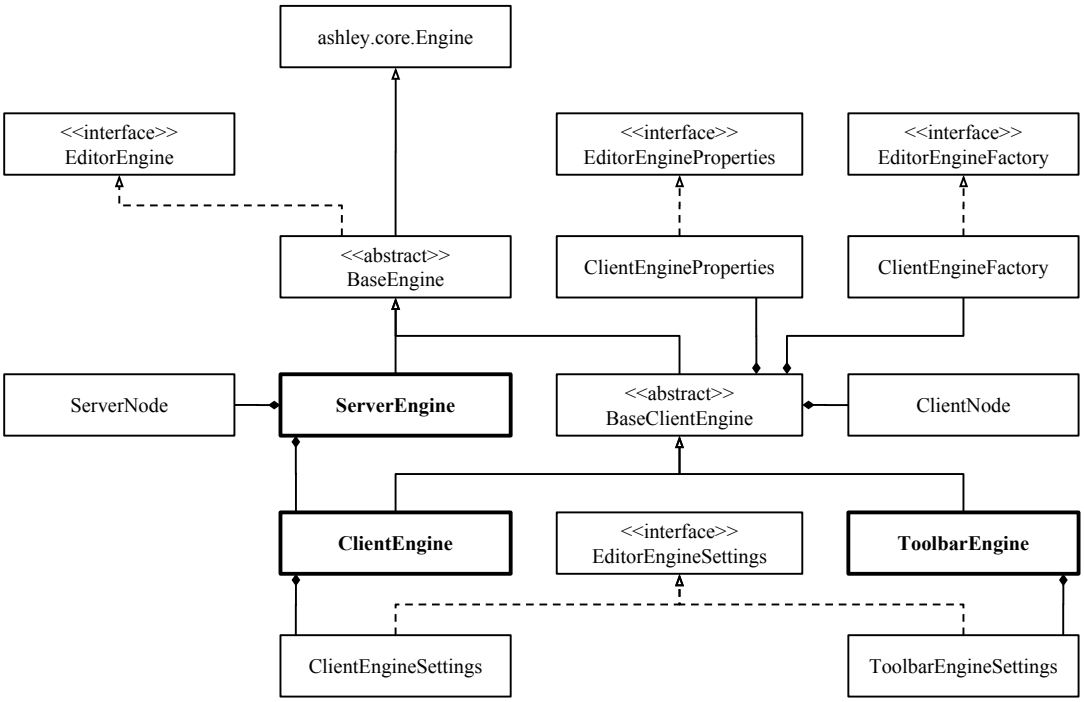


Abbildung 4.8: Engine Übersicht

Die Default-Engine

Die Default-Engine wird in der Zeit verwendet, in der noch kein Editor gestartet wurde. Dies ist z. B. im Main-Screen, dem Scene-Screen oder dem Connect-Screen der Fall. In der Default-Engine ist nur eine Entity enthalten und es sind nur zwei Systeme registriert. Bei der Entity handelt es sich um eine Sky-Entity mit einer Model-Instance-Component und einer Transform-3-Component. In diesem Fall wird die Model-Instance-Component mit der Model-Instance einer Kugel initialisiert, auf die ein Kugelpanorama als Textur gelegt wurde. Die Normalen der Kugel zeigen ins Zentrum der Kugel, wo sich auch der Ursprung des Models befindet. Auf diese Weise wird eine Sky-Sphere erzeugt, wie sie oft in Computerspielen genutzt wird, um einen nie zu erreichenden Horizont darzustellen. Damit der Horizont auch in dieser Engine unerreichbar bleibt, wurde der Engine das Sky-System hinzugefügt. Dieses System sorgt dafür, dass der Horizont unerreichbar bleibt, in dem es die Position in der Transform-3-Component immer an die Position der virtuellen Kamera setzt. Das Rendering-System greift auf diese Position zurück, wenn es die Model-Instance aus der Model-Instance-Component der Sky-Entity rendert. Das Rendering-System ist unteranderem darauf ausgelegt alle Entities zu rendern, die sich im Frustum der virtuellen Kamera befinden und eine Model-Instance-Component, sowie eine Transform-3-Component beinhalten, daher ist es auch für diese Engine geeignet.

Die Server-Engine

Die Server-Engine ist für die An- und Abmeldung von Client-Engines, die Synchronisation von Client-Engines und den Zustand der Szene verantwortlich. Sie entscheidet autoritär welche Änderungen an der Szene unternommen werden dürfen und welche nicht. Client-Engines senden nur ihre Änderungswünsche an die Server-Engine. Die Server-Engine validiert diese Wünsche und informiert die entsprechende Client-Engine über den Erfolg oder Misserfolg. Die Änderungen die von der Server-Engine akzeptiert wurden, werden in der Server-Engine direkt auf die Editor-Entities übertragen und persistiert. Alle Client-Engines werden nach einer Änderung durch die Server-Engine mit einem Update versorgt.

Während der Implementation hat sich gezeigt, dass die Server-Logik und die Client-Logik keine Schnittmenge haben und es vermehrt zu Ausnahmesituationen in den Client-Systemen gekommen ist, sobald die Client-Systeme in der selben Engine liefen wie die Server-Systeme. Der Grund hierfür ist, dass die Server-Systeme bereits die akzeptierten Änderungen an den Editor-Entities vorgenommen haben und die Client-Systeme in diesem besonderen Fall nur noch einen Teil ihrer Logik ausführen durften. Um diese Ausnahmefälle zu umgehen und um Inkonsistenzen vorzubeugen, falls doch einmal eine Ausnahme bei der Implementation

übersehen wird, wurde entschieden, dass auf dem Server zwei Engines laufen. Eine für den Server und eine für den Client. Nach außen fällt dies gar nicht auf, da die Client-Engine in der Server-Engine gekapselt ist und alle Methoden von der Server-Engine an die Client-Engine delegiert werden. Diese Lösung macht die System-Modellierung zwar einfacher, jedoch musste in Kauf genommen werden, dass bei der Server-Engine alle Entities doppelt existieren und somit mehr Ressourcen verbraucht werden. Denn eine Editor-Entity existiert nun einmal in der Server-Engine und einmal in der Client-Engine. Die Kommunikation zwischen den beiden Engines erfolgt wie bei allen anderen Client-Engines, die mit der Server-Engine verbunden sind, über die Network-Middleware. Die Client-Engine wird initialisiert, sobald die Server-Engine erfolgreich gestartet wurde. Anschließend verbindet sich die Client-Engine über die "localhost"-Domain mit der Server-Engine, um sich zu registrieren und zu synchronisieren.

Die Client-Engine

Bevor die Client-Engine mit der System-Iteration beginnt, registriert und synchronisiert sie sich über die Server-Engine. Die Client-Engine ist für die Visualisierung der Szene verantwortlich und hat dementsprechend Systeme, um die Editor-Entities für das Rendering vorzubereiten. Darüber hinaus stellt es Systeme zur Verfügung, um Änderungs-Anfragen an die Server-Engine zu verschicken.

Die Toolbar-Engine

Die Toolbar-Engine ist im Grunde auch eine Client-Engine, nur enthält sie deutlich weniger Systeme, da sie keine Szene rendern muss.

4.2.5 System Übersicht

Jedes System in der Editor-Engine hat eine bestimmte Aufgabe, auf die es spezialisiert ist und es gibt keine Editor-Engine, die alle Aufgaben erfüllt. In Abbildung 4.9 sind die System-Iterationen der unterschiedlichen Editor-Engines illustriert.

Die verfügbaren Systeme lassen sich unterscheiden in Client-Systeme und Server-Systeme.

Client-Systeme

Camera-System Das Camera-System ist für die Ausrichtung der virtuellen Kamera, die für das Rendering der Szene verwendet wird, verantwortlich. Dabei spielt der aktive Modus eine Rolle. Beim VR-Modus wird bspw. die Ausrichtung des Geräts verwendet, um die Kamera zu

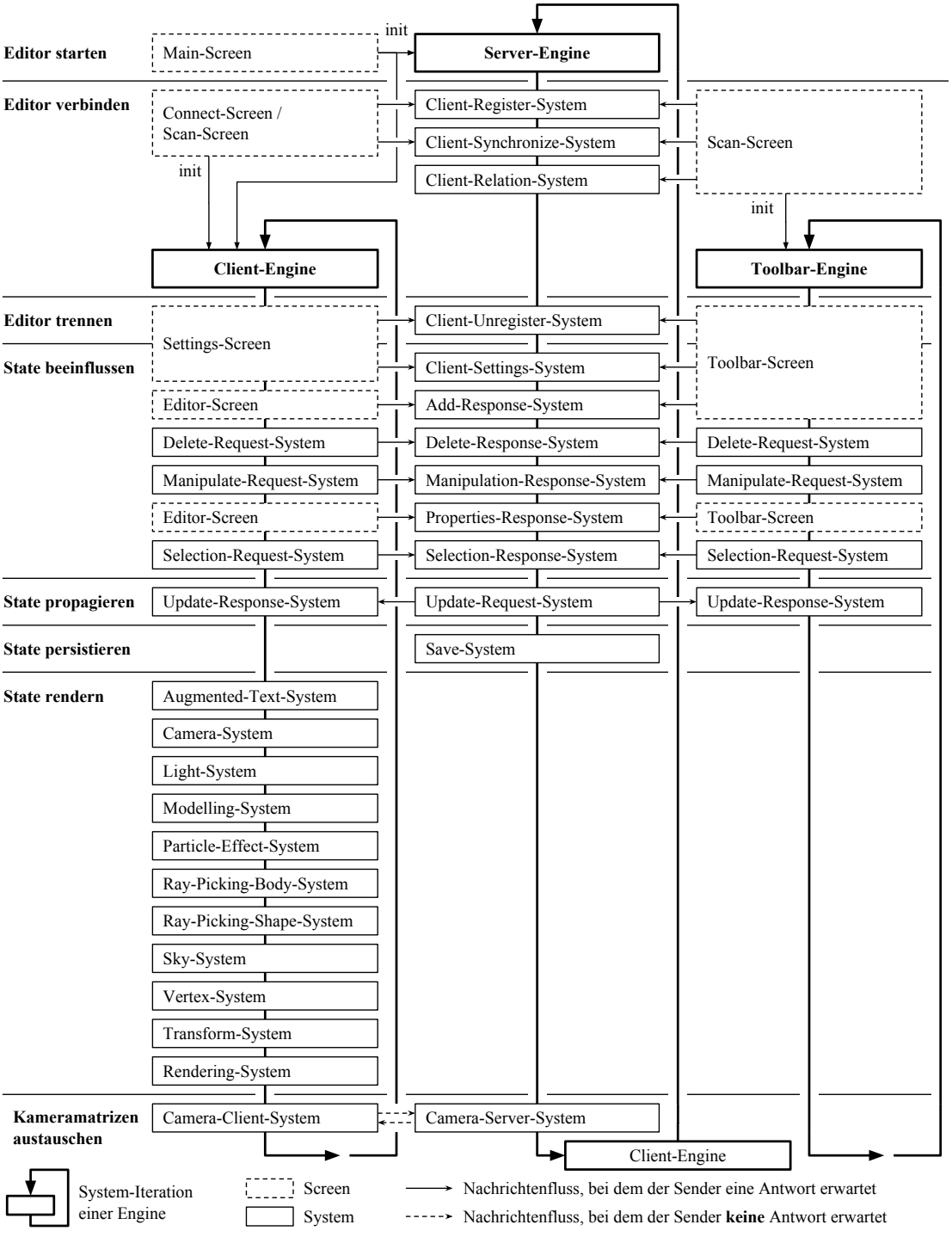


Abbildung 4.9: System Übersicht

drehen. Im AR-Modus werden die intrinsischen und extrinsischen Parameter der Gerätekamera verwendet und im Editor-Modus werden Touch-Gesten verwendet, um die Kamera zu steuern.

Modeling-System Dieses System erzeugt Models aus den Model-, Vertex-, Edge- und Face-Entities. Die Models können vom Rendering-System zum Rendern der Szene verwendet werden. Ein Model enthält neben den Mesh-Informationen auch Materialien, wie z. B. eine Texture, eine Farbe und eine Transformation. Damit Modelle in der Szene angetippt bzw. angeklickt werden können, wird die Ray-Picking Methode aus der Bullet[16] Physik-Engine verwendet. Diese Methode setzt voraus, dass jedes Objekt in der Szene einen Collision-Body besitzt und zu einer Collision-World hinzugefügt wurde. Daher erzeugt das Modelling-System auch die Collision-Bodies mit Hilfe der erstellten Models und fügt sie der Collision-World hinzu. Der Collision-Body wird der Model-Entity als Ray-Picking-Body-Component hinzugefügt und wird von nun an beim Ray-Picking berücksichtigt.

Ray-Picking-Body-System Damit die Szene und die darin enthaltenen Objekte mit der Collision-World übereinstimmen, werden alle Collision-Bodies in diesem System mit der Transformation aus der Szene aktualisiert. Nach außen stellt das System eine Methode zur Verfügung, um einen Ray-Test durchzuführen. Wenn der Test positiv ist, wird die entsprechende Editor-Entity zurückgegeben.

Ray-Picking-Shape-System Nicht für alle Anwendungsfälle ist die Verwendung einer Collision-World notwendig, besonders dann nicht wenn die Objekte beim Rein- und Rauszoomen immer die selbe Größe auf dem Bildschirm behalten sollen. Dies ist z. B. notwendig für die Interaktion mit Vertex-Entities, die durch ein immer gleich großes Quadrat dargestellt werden (siehe Abbildung 4.3). Den Collision-Body in Bullet zu skalieren, wäre viel zu aufwändig. Aus diesem Grund gibt es das Ray-Picking-Shape-System, dass ein Ray-Picking auf einfachen Geometrien bzw. Shapes durchführen kann und dabei das Zoom-Level berücksichtigt.

Transform-System Dieses System bestimmt die tatsächliche Transformation einer Entity in der Szene. Die tatsächliche Transformation wird in einer Transform-Component gespeichert und berechnet sich aus den vorhandenen Components einer Entity. Die folgenden Components nehmen Einfluß auf die Transformation: Position-, Rotation-, Scale-, Manipulate- und Manipulate-Pending-Component.

[16] http://www.bulletphysics.org/wordpress/

Rendering-System Dieses System sammelt alle 2D und 3D Informationen, die sich im Frustum der aktiven Kamera befinden und lässt sie durch einen Renderer, der zum System gehört, darstellen. Das System übernimmt außerdem die Split-Screen Darstellung für AR- und VR-Brillen, sowie das Überblenden zwischen der Modi *Normal*, *Google-Cardboard* und *Augmented-Reality*.

Light-System Dieses System aktualisiert alle Lichter, die sich in der Szene befinden.

Vertex-System Vertices sollen in der Szene immer die selbe Größe haben und das unabhängig davon, wie nah sie sich an der Kamera befinden. Dieses System berechnet die notwendigen Transformationen für diesen Effekt.

Delete-Request-System Sobald ein Anwender auf den *"Entfernen"*-Button tippt, werden alle selektierten Entities mit einer Delete-Request-Component markiert. Dadurch werden sie von diesem System erkannt. Das System sendet daraufhin einen Delete-Request an die Server-Engine. Nachdem die Nachricht erfolgreich verschickt wurde, wird die Delete-Request-Component durch eine Delete-Pending-Component ersetzt.

Manipulate-Request-System Sobald eine Manipulation beendet wurde, wird ein Manipulate-Request an die Server-Engine gesendet und die Manipulate-Component wird durch eine Manipulate-Pending-Component ersetzt (siehe Abschnitt 4.4.5).

Selection-Request-System Wenn ein Anwender ein Objekt im Editor antippt, wird es mit einer Select-Request- bzw. Unselect-Request-Component markiert. Dieses System verschickt die entsprechenden Requests an den Server.

Es gibt kein Add-Request-System, da direkt eine Add-Message mit den gewünschten Objekten erzeugt und verschickt wird. Sollte beim Versenden ein Fehler auftreten oder der Server die Anfrage ablehnen, wird dem Anwender ein Fehler-Dialog angezeigt. Der Text für den Fehler-Dialog wird der Response-Message entnommen.

Serverseitige Systeme

Client-Register- / Client-Unregister-System Das Client-Register-System nimmt Anmeldungen entgegen und das Client-Unregister-System führt Abmeldungen durch.

Client-Synchronize-System Das Client-Synchronize-System nimmt Client-Synchronize-Request-Messages entgegen, auf die es mit Client-Synchronize-Response-Messages reagiert.

Selection-Response-System Das Selection-Response-System validiert Select- und Unselect-Messages.

Add-Response-System Dieses System validiert alle Add-Request-Messages und fügt der Szene nach einer erfolgreichen Validierung die angefragten Objekte hinzu.

Delete-Response-Message Dieses System validiert alle Delete-Request-Messages und entfernt nach einer erfolgreichen Validierung die angefragten Objekte aus der Szene.

Manipulate-Response-Message Dieses System validiert alle Manipulate-Messages und manipuliert bei einer erfolgreichen Validierung die entsprechenden Objekte in der Szene.

Update-Request-System Dieses System sammelt alle akzeptierten Messages aus einem Durchlauf und erstellt daraus eine Update-Request-Message mit aufsteigender Sequenznummer, die einer Warteschlange hinzugefügt wird. Anschließend werden die Update-Request-Messages an die Client-Engines verschickt. Dabei achtet das System darauf, dass jeder Client immer nur ein Update erhält. Ein Client erhält erst dann ein neues Update, wenn er das vorherige mit einer Update-Response-Message bestätigt hat. Dadurch wird sichergestellt, dass sämtliche Änderungen bei der Client-Engine in der selben Sequenz wie in der Server-Engine ausgeführt werden. Sollte eine Update-Request-Message nicht bestätigt werden, wird sie nach einem Timeout erneut zugestellt.

Save-System Dieses System persistiert alle akzeptierten Veränderungen in einer SQLite Datenbank. Jede Client-Engine die sich mit dem Server verbindet, erhält eine Kopie dieser Datenbank und eine Sequenznummer. Die Sequenznummer bestimmt die letzte Update-Request-Message, die in der Kopie der Datenbank persistiert wurde. Dadurch weiß die Client-Engine welche Update-Request-Message sie als nächstes erwarten muss.

4.3 Die Editor-Entities

Alle Entities mit denen im Editor interagiert werden kann und die Bestandteil der Szene sind (siehe Abbildung 4.10), besitzen eine Editor-Component über die sie eindeutig identifiziert werden können, denn jede Editor-Component besitzt einen Universally Unique Identifier (UUID). Mit Hilfe eines Editor-Entity-Listeners wird die Base-Editor-Engine informiert, sobald eine neue Editor-Entity hinzugefügt oder eine alte Editor-Entity entfernt wird. Diesen Mechanismus nutzt die Base-Editor-Engie, um eine Hash-Map zu pflegen, in der sie alle Editor-Entities mit

ihrem Identifier referenziert. Diese Hash-Map stellt die Base-Editor-Engine ihren Kind-Klassen und deren Systemen zur Verfügung, denn der Identifier wird auch in den Request-Messages und in der Datenbank zur Identifikation einer Editor-Entity verwendet.

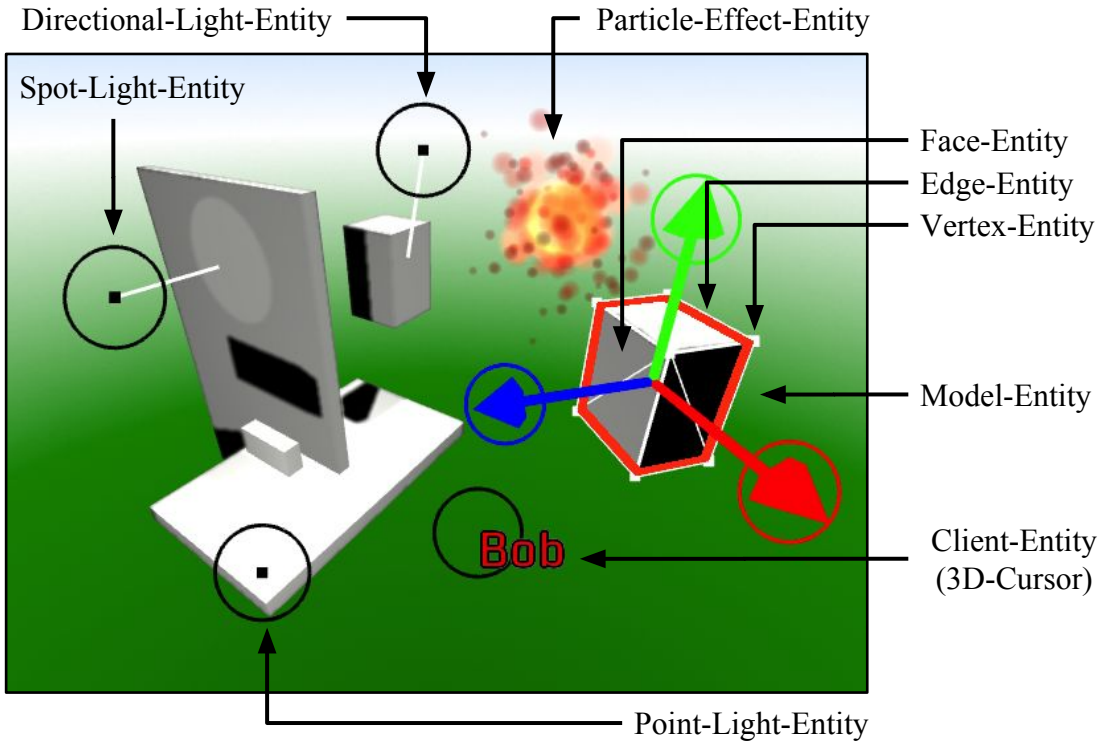


Abbildung 4.10: Editor-Entities

Neben der Editor-Component erhalten alle Editor-Entities außerdem eine Position-Component, wodurch sein eine Position in der Szene erhalten, eine Selectable-Marker-Component, wodurch sie selektierbar gemacht werden und eine Selected-By-Component in der alle Clients aufgelistet sind, die diese Entity im Moment selektiert haben (siehe 4.4). In Abschnitt 4.4.2 wird näher auf die Thematik der Selektion eingegangen.

```
1 public Entity createEditorEntity(ByteString id, Vector3
    position) {
2   Entity entity = new Entity();
3   entity.add(new EditorComponent(id));
4   entity.add(new PositionComponent(position))
5   entity.add(SelectableComponent.MARKER);
6   entity.add(new SelectedByComponent());
7   return entity;
8 }
```

Listing 4.4: Fabrikmethode für die Erzeugung einer Editor-Entity

4.3.1 Client-Entity

Diese Entity repräsentiert einen Kollaborateur bzw. eine Client-Engine. Der Identifier der Editor-Component, ist auch gleichzeitig der Identifier der Client-Engine. Die Position dieser Editor-Entity wird durch einen 3D-Cursor dargestellt (siehe Abbildung 4.10). Neben den Components der Editor-Entity enthält diese Entity außerdem eine Augmented-Text-, Color-, Client- und Transform-3-Component.

Durch die Position-Component erhält der Client eine Position in der Szene. Die Position wird durch einen 3D-Cursor dargestellt und durch die Augmented-Text-Component mit dem Namen des Clients bzw. des Kollaborateurs angereichert.

Der 3D-Cursor bestimmt an welcher Stelle sich der Client im VR-Modus befindet und wo die Objekte erscheinen, die der Client neu hinzufügt. Außerdem können alle Clients die 3D-Cursor der anderen Clients in ihrem Editor sehen und sie können sie sogar durch den Manipulator verschieben. Wenn sich ein Client im VR-Modus befindet, während sich seine Position verändert, wird er durch eine sanfte Interpolation von seiner alten Position zu seiner neuen Position bewegt. Dies ermöglicht eine moderierte Führung durch die Szene (siehe Abbildung 4.11).

Die Color-Component bestimmt in welcher Farbe die Selektion des Clients in der Szene hervorgehoben werden soll. Die Farbe bestimmt der Client während der Registrierung per Zufall. Ein Kollaborateur kann die Farbe seines Clients nachträglich ändern, in dem er seinen 3D-Cursor selektiert und über die Farbpalette eine neue Farbe auswählt.

Die Client-Component enthält neben dem Namen des Kollaborateurs auch die aktuellen Einstellungen seiner Editor-Engine.

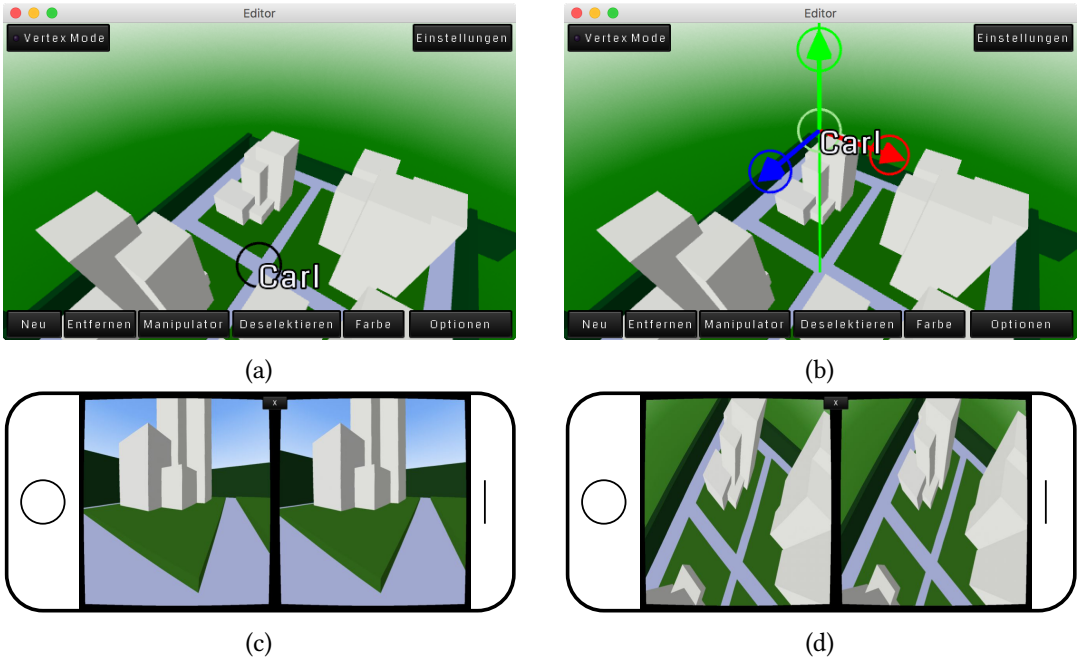


Abbildung 4.11: Die Client-Entity von Carl ist im Editor für alle Kollaborateure sichtbar 4.11a. Eine Client-Entity kann durch andere Kollaborateure manipuliert werden 4.11b. Carl schaut im VR-Modus von der Position aus in die Szene, an der sich seine Client-Entity befindet 4.11c. Wenn die Client-Entity von Carl verschoben wird, während er sich im VR-Modus befindet, wird er durch eine sanfte Interpolation an die neue Position bewegt 4.11d.

Tabelle 4.1: Entity Component Matrix

Component \ Entity	Model	Vertex	Edge	Face	PointLight	SpotLight	DirectionalLight	ParticleEffect	Client	ClientRelation	AugmentedText	Sky
Editor	●	●	●	●	●	●	●	●	●	–	–	–
Selectable (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Selected (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Selected-By	●	●	●	●	●	●	●	●	●	–	–	–
Select-Request	○	○	○	○	○	○	○	○	○	–	–	–
Select-Pending	○	○	○	○	○	○	○	○	○	–	–	–
Unselectable (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Unselect-Request	○	○	○	○	○	○	○	○	○	–	–	–
Unselect-Pending	○	○	○	○	○	○	○	○	○	–	–	–
Save-Add (Marker)	○	○	○	○	○	○	○	○	○	○	–	–
Save-Delete (Marker)	○	○	○	○	○	○	○	○	○	○	–	–
Save-Update (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Save-Select (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Save-Unselect (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Delete (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Delete-Pending (Marker)	○	○	○	○	○	○	○	○	○	–	–	–
Manipulate (Singleton)	○	○	○	○	○	○	○	○	○	–	–	–
Manipulate-Pending	○	○	○	○	○	○	○	○	○	–	–	–
Ray-Picking-Body	◐	–	–	–	–	–	–	–	–	–	–	–
Ray-Picking-Shape	–	●	–	–	●	●	●	●	●	–	–	–
Sphere-Shape	–	●	–	–	●	●	●	●	●	–	–	–
Augmented-Text	–	–	–	–	–	–	–	–	●	–	●	–
Augmented-Text-Entity	–	–	–	–	–	–	–	–	○	–	–	–
Augmented-Text-Body	–	–	–	–	–	–	–	–	–	–	●	–
Bitmap-Font	–	–	–	–	–	–	–	–	–	–	●	–
Color	–	●	–	–	●	●	●	–	●	–	–	–
Decal	–	●	–	–	●	●	●	–	–	–	–	–
Light-Intensity	–	–	–	–	●	●	–	–	–	–	–	–
Model-Instance	◐	–	–	–	–	–	–	–	–	–	–	●
Particle-Controller	–	–	–	–	–	–	–	●	–	–	–	–
Position	●	●	–	–	●	●	●	●	●	–	–	–
Rotation	●	–	–	–	–	●	●	●	–	–	–	–
Scale	●	–	–	–	–	–	–	–	–	–	–	–
Transform2	–	●	–	–	–	–	–	–	–	–	●	–
Transform3	●	–	–	–	●	●	●	●	●	–	–	●

● = immer vorhanden; ◐ = werden erzeugt, wenn sie nicht vorhanden sind; ○ = zeitweise vorhanden; – = niemals vorhanden

4.4 Editor-Entity Operationen

4.4.1 Persistieren

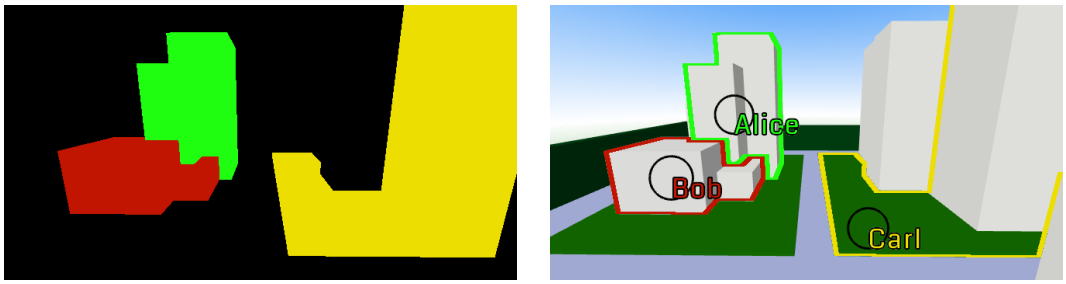
Das Persistieren erfolgt in der Server-Engine am Ende einer System-Iteration (siehe Abbildung 4.9), durch das Save-System, das sämtliche Szenenänderungen in einer Datenbank speichert (siehe Abbildung 4.5). Zuvor hat jedes System, das während der System-Iteration eine Änderung an einer Editor-Entity vorgenommen hat, die betroffene Entity mit einer entsprechenden Save-Marker-Component gekennzeichnet. Entities die neu hinzugefügt wurden, haben eine Save-Add-Component erhalten, Entities die gelöscht werden sollen, haben eine Save-Delete-Component erhalten und Entities die verändert wurden, haben eine Save-Update-Component erhalten. Außerdem erhalten Entities, die während der System-Iteration selektiert wurden, eine Save-Select-Component und Entities, die deselektiert wurden, eine Save-Unselect-Component. Die Save-Marker-Components ermöglichen es dem Save-System über alle neuen, gelöschten, veränderten, selektierten und deselektierten Editor-Entities einer bestimmten Family zu iterieren. Dies wiederum ermöglicht den Einsatz von Stapelverarbeitung bei SQL-Befehlen. Wenn z.B. neue Vertices während einer System-Iteration hinzugefügt wurden, iteriert das Save-System über alle Entities der Insert-Vertex-Family und erstellt dabei einen einzigen SQL-Befehl für alle neuen Vertices. Zur Insert-Vertex-Family gehören in diesem Fall alle Entities, die eine Vertex- und eine Save-Add-Component besitzen.

Nachdem eine Editor-Entity vom Save-System abgearbeitet wurde, wird die entsprechende Save-Marker-Component von der Entity entfernt. Handelt es sich bei der zu entfernenden Component jedoch um die Save-Delete-Component, wird die gesamte Entity gelöscht, denn das Save-System ist die letzte Instanz, durch die eine akzeptierte Szenenänderung läuft.

4.4.2 Selektieren

Um Objekte in der Szene verändern zu können, müssen sie zunächst vom Benutzer selektiert werden. Objekte werden durch anklicken bzw. antippen selektiert und können auf die gleiche Weise auch wieder deselektiert werden. Außerdem befindet sich im Editor-Screen eine Schaltfläche, die bei Betätigung die gesamte Selektion aufhebt.

Damit es leichter fällt die selektierten Objekte in der Szene zuerkennen und einem Benutzer zuzuordnen, erhalten sie eine Umrandung in der Farbe des jeweiligen Benutzers. Hierfür sind zwei zusätzliche Renderings notwendig. Beim ersten Rendering werden nur die selektierten Objekte in der jeweiligen Benutzerfarbe gerendert (siehe Abbildung 4.12a). Beim zweiten Rendering wird das Ergebnis des ersten Renderings mit einem Shader, der Kanten erkennt, auf das Ergebnis des Szenen Renderings gerendert (siehe Abbildung 4.12b).



(a) (b)

Abbildung 4.12: Die Selektion eines Kollaborateurs wird durch seine Benutzerfarbe verdeutlicht.

4.4.3 Erstellen

Neue Objekte können der Szene über Schaltflächen im Editor-Screen und dem Toolbar-Screen hinzugefügt werden. Momentan kann ein Würfel, eine Kugel oder ein Torus als Model hinzugefügt werden. Außerdem ist es möglich ein direktes Licht, ein Scheinwerfer, ein punktuelles Licht oder ein Partikel-Effekt hinzuzufügen. Das Hinzufügen von einzelnen Vertices, Edges oder Faces wird noch nicht unterstützt.

4.4.4 Entfernen

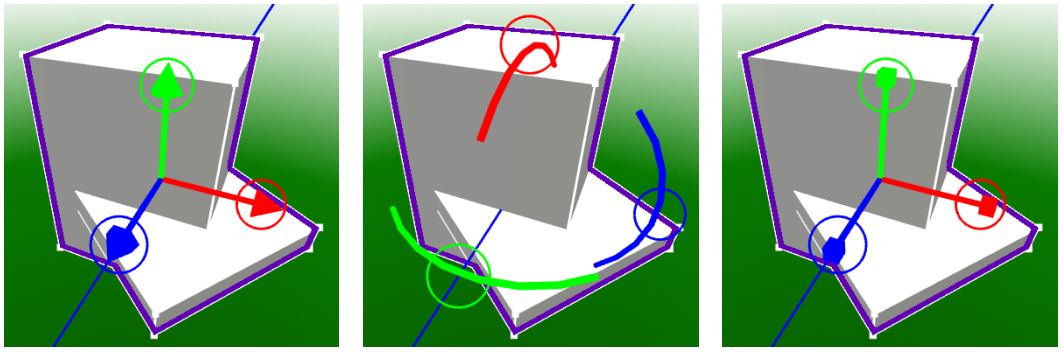
Das Löschen von Editor-Entities ist sehr einfach gehalten. Es gibt eine Schaltfläche im Editor-Screen und im Toolbar-Screen, über die alle Editor-Entities, die aktuell durch die Client-Engine selektiert sind, gelöscht werden können.

4.4.5 Manipulieren

Damit Objekte in der Szene verschoben, rotiert und skaliert werden können, wurde ein Manipulator-Controller entwickelt. Der Manipulator-Controller verwaltet einen Translate-Manipulator, mit dem sich Objekte verschieben lassen (siehe Abbildung 4.13a), einen Rotate-Manipulator, mit dem sich Objekte rotieren lassen (siehe Abbildung 4.13b), und einen Scale-Manipulator, mit dem sich Objekte skalieren lassen (siehe Abbildung 4.13c).

Der Manipulator-Controller sammelt alle vom Benutzer durchgeführten Veränderungen in einer einzigen Manipulate-Component. Diese Manipulate-Component wird vor Beginn einer Manipulation allen selektierten Editor-Entities hinzugefügt und während des Renderings auf die originale Transformation addiert bzw. im Falle einer Rotation multipliziert. Am Ende der Manipulation wird die Manipulate-Component entfernt und durch eine Manipulate-Pending-

Component ersetzt, die die gleichen Manipulationen enthält. Sollte der Manipulate-Reqeust des Clients fehlschlagen, wird die Manipulate-Pending-Component entfernt und damit wird die getätigte Manipulation verworfen. Sollte der Manipulate-Request jedoch erfolgreich sein, bleibt die Manipulate-Pending-Component solange erhalten, bis die Client-Engine den Update-Request mit der entsprechenden Manipulation von der Server-Engine empfängt. Nach dem Empfangen des Update-Requests, wird die akzeptierte Manipulation bei allen Client-Engines in die originale Transformation übernommen.



(a) Translation (b) Rotation (c) Skalierung

Abbildung 4.13: Der Manipulator

4.4.6 Anpassen

Es gibt bislang nur eine Eigenschaft der Editor-Entities, die angepasst werden kann und das ist die Farbe bzw. die Color-Component. Um die Farbe einer Editor-Entity zu verändern, ist es notwendig die jeweilige Editor-Entity zu selektieren und ihr über das Farbfeld eine neue Farbe zuzuweisen.

4.4.7 Zusammenfassung

In diesem Kapitel wurde das Konzept und ein Prototyp des kollaborativen 3D Mixed-Reality Editors präsentiert. Zuerst wurden die Module vorgestellt aus denen sich der Editor zusammensetzt. Anschließend wurde das Entity-Component-System vorgestellt, in dem die unterschiedlichen Engines, Systeme und Entities samt ihrer Components näher erläutert wurden. Zum Schluss wurden die Operationen beschrieben, die auf den Editor-Entities innerhalb einer Szene ausgeführt werden können.

5 Fazit

Die Realisierung eines kollaborativen 3D Mixed-Reality Editors für mobile Geräte wurde im Rahmen des Masterstudiums erfolgreich bearbeitet. Es existiert ein Prototyp, der sowohl auf Android und iOS als auch auf dem Desktop verwendet werden kann. Der Prototyp bietet eine einfache Szenenverwaltung und unterstützt das Erstellen, Entfernen und Manipulieren von 3D-Objekten innerhalb einer Szene. Um gemeinsam an einer Szene zu arbeiten, können mehrere Instanzen des Prototyps über eine Network-Middleware, die auf binärcodierten Nachrichten basiert, miteinander verbunden werden. Die Verbindung erfolgt entweder manuell oder über eine QR-Code Einladung. Die Benutzer des Editors sehen sich untereinander und erkennen welche Objekte gerade von wem bearbeitet werden. Außerdem können sie sich gegenseitig in der Szene herumführen und verwenden ihr Smartphone dabei als VR-Brille. Auf mobilen Geräten macht der Prototyp Gebrauch von der neuesten Augmented-Reality Technologie, um eine immersive Bearbeitung der Szene zu ermöglichen. Der aktuell vorliegende Prototyp bietet zahlreiche Erweiterungsmöglichkeiten in alle erdenklichen Richtungen.

5.1 Nächste Schritte

Network-Middleware Die Network-Middleware benötigt noch die Unterstützung von UDP-Paketen, damit unkritische Informationen gestreamt werden können, wie z. B. die Kamerainformationen eines Clients oder die Daten der Beschleunigungssensoren, die in den meisten Smartphones verbaut sind. Ein Stream der Kamerainformationen könnte dafür verwendet werden, um durch die Augen eines anderen in die Szene zu blicken. Die Daten eines Beschleunigungssensors ließen sich wiederum auf ein virtuelles Objekt übertragen. Ein weiterer offener Punkt ist die Verschlüsselung von Nachrichten und die eindeutige Identifizierung von Clients. Ohne die Verschlüsselung und Identifizierung reicht es bislang aus die Client-ID zu kennen, um unter falschem Namen Änderungen vornehmen zu können.

Editor-Entities Bislang werden noch zu viele Editor-Entities bzw. Objekte allokiert, da selbst jeder Vertex ein Entity-Objekt ist, das sich zusätzlich aus mehreren Component-Objekten zusammensetzt. Es müsste ein Weg gefunden werden, bei dem die Erzeugung von Vertex-, Edge-

und Face-Entities auf ein Minimum reduziert wird. Eine Möglichkeit wäre die Beschränkung der Vertex-, Edge und Face-Manipulation auf eine einzige Model-Entity. Die notwendigen Editor-Entities müssten dann erst beim Aktivieren der Model-Manipulation erzeugt werden und sie könnten anschließend beim Verlassen der Model-Manipulation auch wieder entfernt werden.

Ein anderer Nachteil der sich bei zu vielen Editor-Entities bemerkbar gemacht hat, ist die Verwendung von UUIDs. Sie verursachen besonders in den Network-Messages die größte Last. Eine Möglichkeit diese Last zu reduzieren, wäre die Verwendung des Kompressionsverfahrens gzip[1]. Dieses Verfahren wurde auch bereits in einem vorherigen Projekt [Blank u. a. (2015)] erfolgreich verwendet, um gerenderte Bilder über die Network-Middleware an eine AR-Brille zu streamen. Durch die Kompression entsteht allerdings zusätzlicher Rechenaufwand. Eine ganz andere Lösung wäre die Verwendung eines kleineren Identifiers, der wohlmöglich nur durch die Server-Engine gepflegt und vergeben wird.

Während einer kleinen Benutzerstudie mit 20 Probanden, kam mehrfach die Frage auf, wie eine Aktion rückgängig gemacht werden kann. Hierbei handelt es sich jedoch um kein triviales Problem, da sich die eigenen Änderungen auf dem Server in eine Kette mit den Veränderungen der anderen Kollaborateure einreihen. Das bedeutet, dass zwischen den Änderungen, die ein Benutzer durchgeführt hat, viele andere Änderungen liegen können, die Einfluss auf die Änderungen des Benutzers genommen haben könnten. Eine globale Lösung wäre die inverse Durchführung des letzten Schritts auf dem Server, den ein Client durch eine Undo-Message auslösen könnte. Andere Benutzer könnte dies jedoch irritieren, da womöglich sie den letzten Schritt auf dem Server ausgelöst haben.

Weitere Ergänzungen wären der Ausbau der Vertex-, Edge- und Face-Manipulation, sowie die Verwendung von Texturen und Materialien. Auch die Erzeugung von neuen Objekten durch Touch-Gesten wäre ein wünschenswertes Feature oder die Möglichkeit Objekte zu kopieren und zu gruppieren.

Augmented-Reality Es bleibt zu erwarten, dass die meisten mobilen Geräte zukünftig in der Lage sein werden Tiefe wahrzunehmen. Dann könnten Tiefenbilder verwendet werden, um mit der eigenen Hand Objekte zu selektieren. Die ersten Geräte wie z. B. das Project Tango oder das iPhone X haben bewiesen, dass es möglich ist. Die Schnittstellen sind jedoch noch nicht ausreichend geöffnet worden, so unterstützt das iPhone X die Tiefenwahrnehmung bislang nur bei der Frontkamera, wo es in erster Linie für die Gesichtserkennung verwendet wird. Geräte

[1] http://www.gzip.org

wie das iPhone 7 Plus verfügen auch bereits über die Hardware, um Tiefenbilder zu erzeugen, allerdings steht diese Funktionalität bislang nur für die Bearbeitung von Fotos zur Verfügung.

5.2 Ausblick

Ein mögliches Einsatzgebiet wäre die Planung und Besprechung von Bauvorhaben, das gemeinsame Betreten einer Szene, die einen historischen Schauplatz repräsentiert und im Geschichtsunterricht als Diskussionsgrundlage dienen könnte, oder die anschauliche Planung einer neuen Küche. Ein ähnliches Szenario wäre auch an einem Messestand denkbar, bei dem ein Interessent einen räumlichen Sachverhalt erklärt bekommt. Zukünftig könnte der Editor auch mit Live-Daten von außen gespeist werden, um bspw. den Kurs von Segelboten bei einer Regatta zu visualisieren. Oder er könnte durch eine Erweiterung für die Outdoor- und Indoor-Navigation verwendet werden, wobei Benutzer dann in der Lage wären über den Editor Wegpunkte in der Form von 3D-Objekten zu platzieren.

Literaturverzeichnis

[Blank u. a. 2015] Blank, Christian ; Eckhoff, Malte ; Petersen, Iwer ; Wege, Raimund ; Wendholt, Birgit: Distributed Collaborative Construction in Mixed Reality. (2015), Nr. c, S. 198–202. ISBN 9781612083827

[Eckhoff 2017] Eckhoff, Malte: *Konsistenz in verteilten kollaborativen Konstruktions MR-Anwendungen*, HAW Hamburg, Masterarbeit, 2017

[Ehrlich 2013] Ehrlich, Justin: The Component Entity System for Virtual Environments. In: *Proceedings of the International Conference on Computer Graphics and Virtual Reality (CGVR)* The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (Veranst.), 2013, S. 32

[Hall u. a. 2014] Hall, Daniel ; Slo, Cal P. ; Wood, Zoe ; Slo, Cal P.: ECS Game Engine Design. (2014), Nr. June, S. 1–9

[Houston u. a. 2010] Houston, Ben ; Larsen, Wayne ; Larsen, Bryan ; Caron, Jack ; Nikfetrat, Nima ; Leung, Catherine ; Silver, Jesse ; deen, Hasan Kamal-al ; Callaghan, Peter ; Chen, Roy ; Mckenna, Tim: Clara . io : Full-Featured 3D Content Creation for the Web and Cloud Era. (2010), S. 2010. ISBN 9781450322614

[Igarashi 1999] Igarashi, Takeo: Teddy : A Sketching Interface for 3D Freeform Design. (1999), S. 409–416. ISBN 0201485605

[Janssen 2017] Janssen, Jan-Keno: *Googles Augmented Reality: Tango ist tot, es lebe ARCore*. 2017. – URL https://www.heise.de/newsticker/meldung/Googles-Augmented-Reality-Tango-ist-tot-es-lebe-ARCore-3817226.html. – Zugriffsdatum: 2017-12-06

[Lakatos u. a. 2014] Lakatos, David ; Blackshaw, Matthew ; Olwal, Alex ; Barryte, Zachary ; Perlin, Ken ; Ishii, Hiroshi: T(Ether): Spatially-aware Handhelds, Gestures and Proprioception for Multi-user 3D Modeling and Animation. In: *Proceedings of the 2Nd ACM Symposium on Spatial User Interaction*. New York, NY, USA : ACM, 2014 (SUI '14),

S. 90–93. – URL http://doi.acm.org/10.1145/2659766.2659785. – ISBN 978-1-4503-2820-3

[Mori und Igarashi 2007] Mori, Yuki ; Igarashi, Takeo: Plushie. In: *ACM Transactions on Graphics* 26 (2007), Nr. 99, S. 45. – URL http://portal.acm.org/citation.cfm?doid=1276377.1276433. – ISSN 07300301

[Munshi u. a. 2008] Munshi, Aaftab ; Ginsburg, Dan ; Shreiner, Dave: *OpenGL(R) ES 2.0 Programming Guide*. 1. Addison-Wesley Professional, 2008. – ISBN 0321502795, 9780321502797

[Rost u. a. 2009] Rost, Randi J. ; Licea-Kane, Bill ; Ginsburg, Dan ; Kessenich, John M. ; Lichtenbelt, Barthold ; Malan, Hugh ; Weiblen, Mike: *OpenGL Shading Language*. 3rd. Addison-Wesley Professional, 2009. – ISBN 0321637631, 9780321637635

[Saltares 2014] Saltares, David: *Ashley entity framework*. 2014. – URL http://saltares.com/blog/games/ashley-entity-framework/. – Zugriffsdatum: 2017-11-26

[Weichel u. a. 2014] Weichel, Christian ; Lau, Manfred ; Kim, David ; Villar, Nicolas ; Gellersen, Hans W.: MixFab: A Mixed-Reality Environment for Personal Fabrication. In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14* (2014), S. 3855–3864. – URL http://dl.acm.org/citation.cfm?doid=2556288.2557090. ISBN 9781450324731

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 8. Dezember 2017 Raimund Wege