

Bachelorarbeit

Interaktive Visualisierung von Software-Requirements mit Augmented Reality: Eine Analyse der Usability und Effektivität

im Studiengang Softwaretechnik und Medieninformatik (SWB)
der Fakultät Informationstechnik
Sommersemester 2024

Kyle Mezger
Matrikelnummer: 765838

Zeitraum: 01.03.2024 bis 31.08.2024
Erstprüfer: Prof. Dr. -Ing. Andreas Rößler
Zweitprüfer: Prof. Dr. rer. nat. Dieter Morgenroth

Firma: IT Designers Gruppe

Betreuer: Stefan Kaufmann

Eidesstattliche Erklärung

Hiermit versichere ich, Kyle Mezger, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Weiterhin erkläre ich, dass ich für die Umformulierung einzelner Textpassagen und die Korrektur von Rechtschreib- und Grammatikfehlern die Hilfe von digitalen Tools, speziell ChatGPT und DeepL Write, in Anspruch genommen habe. Diese Unterstützung betraf ausschließlich die Formulierungshilfe und die Korrektur von Grammatik und Rechtschreibung, ohne dass die inhaltliche Eigenständigkeit und Urheberschaft meiner Arbeit dadurch berührt wurden.

Esslingen, den 3. Juni 2024 _____
Unterschrift

Inhaltsverzeichnis

1 Kurzfassung	5
2 Einleitung	6
2.1 Motivation	6
2.2 Zielsetzung	6
3 Grundlagen	7
3.1 Requirements Engineering	7
3.2 reQlab	9
3.3 Virtuelle Realität	10
3.4 Augmented Reality	10
3.4.1 Head-Mounted Displays	11
3.4.2 Hand-Held-Devices	14
3.4.3 Spatial Displays	14
4 Technologien	16
4.1 Oculus Quest 3	16
4.2 WebGL	17
4.3 WebXR	18
4.3.1 Three.js	19
4.3.2 Babylon.js	19
5 Implementierung	21
5.1 Entwicklungsumgebung für WebXR und Oculus Quest 3	21
5.2 Implementierung der Anwendung	22
5.2.1 Interaktionskonzepte für Requirements	23
5.2.2 Implementierung der Interaktionskonzepte	25
5.3 User Tests	28
6 Zusammenfassung	29
6.1 Ergebnisse	29
6.2 Fazit	29
6.3 Ausblick	29

Abbildungsverzeichnis

1	Realitäts-Virtualitäts-Kontinuum nach Milgram	11
2	Oculus Quest 3 VR-Headset – <i>Quelle: https://unsplash.com/de/fotos/ein-mann-der-auf-einem-stuhl-sitzt-und-eine-virtuelle-brille-tragt-AZ-ND5uJ4S4</i>	13
3	Frontal ausgerichtete Kameras und Tiefenprojektor der Oculus Quest 3 . . .	16
4	Die beiden nach unten gerichteten Kameras der Oculus Quest 3	17
5	Application Flow von WebXR-Anwendungen	18
6	Szenen-Explorer und -Inspektor von Babylon.js	20
7	Screenshots der Immersive Web Emulator Erweiterung in Chrome	22
8	Screenshots des explodierenden Tetris-Blocks mit Anforderungen in AR . .	26
9	Screenshot des in der Anwendung genutzten Porsche Modells	27
10	Bildsequenz der Animation des Porsche Modells	28

1 Kurzfassung

2 Einleitung

2.1 Motivation

2.2 Zielsetzung

Im Laufe der Bachelorarbeit sollen verschiedene Interaktionskonzepte für die Anzeige von Anforderungen in einer AR-Umgebung untersucht und auf ihre Vor- und Nachteile, sowie auf ihre Eignung für den Einsatz in einem realen Projekt untersucht werden.

Dabei sollen möglichst mehrere Konzepte entwickelt und prototypisch umgesetzt werden, um diese anschließend zu evaluieren und zu vergleichen. Der Fokus der Evaluation soll auf der Usability und der Umsetzbarkeit der Konzepte liegen. Vor allem der Mehrwert der Darstellung in AR gegenüber herkömmlichen Methoden soll dabei kritisch betrachtet werden.

Basierend auf dem Prozess der Implementierung soll auch bewertet werden, wie gut sich die Konzepte in bestehende Systeme integrieren lassen und wie aufwändig die Implementierung im Vergleich zum Mehrwert der Darstellung ist. Besonders interessant ist dabei, inwiefern eine Integration mit der reQlab Plattform möglich ist und wie sich die Konzepte in bestehende Workflows einfügen lassen.

3 Grundlagen

Im folgenden Kapitel sollen konzeptionelle Grundlagen erläutert werden, welche für das Verständnis der Bachelorarbeit notwendig sind. Dabei wird auf die Themen Requirements Engineering, Augmented Reality und die Software reQlab eingegangen.

3.1 Requirements Engineering

Die Bachelorarbeit soll sogenannte Requirements, also Anforderungen, visualisieren. Daher ist es für das Verständnis der Arbeit wichtig, die Grundlagen des Requirements Engineering zu kennen.

Requirements

Grundlegend sind Requirements Anforderungen, die an ein System gestellt werden. Das International Requirements Engineering Board (IREEB) definiert sie in ihrem Glossar mit drei Eigenschaften:

- Ein Bedürfnis eines Interesseneigners (Stakeholder).
- Eine Eigenschaft oder Fähigkeit, die ein System haben soll.
- Eine dokumentierte Repräsentation eines Bedürfnisses, einer Fähigkeit oder einer Eigenschaft.

[1, Def. Anforderung]

Sie sollen also die Bedürfnisse der Stakeholder an das System repräsentieren und dokumentieren.

Die Gestaltung von Requirements kann dabei je nach System und Anforderungen unterschiedlich sein. Chris Rupp nennt in ihrem Buch „Requirements-Engineering und -Management“ einige Beispiele für verschiedene Formen für Requirements:

- User-Stories
- Use-Cases
- Stories
- formalisierte natürlichsprachliche Anforderungen

- Anforderungen in Form von Diagrammen (semiformales Modell)

[2, S. 19]

Natürlichsprachliche Anforderungen können sehr einfach selbst formuliert werden. Dadurch sind sie jedoch auch anfällig für Missverständnisse und Unklarheiten. Das Ziel von reQlab ist es, diese Missverständnisse und Unklarheiten in natürlichsprachlichen Anforderungen zu erkennen und so die Qualität der Anforderungen zu verbessern. Daher werden im Unfamg dieser Bachelorarbeit nur natürlichsprachliche Anforderungen genutzt.

Zudem werden Requirements in funktionale und nicht-funktionale Requirements unterteilt. Funktionale Requirements beschreiben „die Funktionen, die das System leisten soll, die Informationen die es verarbeiten soll; das gewünschte Verhalten, welches das System an den Tag legen soll.“ [3, S. 12] Nicht-funktionale Requirements hingegen beschreiben alle Requirements, die nicht funktionaler Natur sind, also beispielsweise Performance, Sicherheit oder Zuverlässigkeit. Der Begriff nicht-funktional ist dabei jedoch etwas irreführend, da auch nicht-funktionale Requirements gewissermaßen Funktionen des Systems beschreiben. Peter Hruschka beschreibt in seinem Buch Funktionale Anforderungen mit der Frage: „Was soll das System/Produkt tun?“. Auch unterteilt er nicht-funktionale Anforderungen in die zwei Kategorien Qualitätsanforderungen („Wie gut? Wie schnell? Wie zuverlässig? ...“) und Randbedingungen („Ressourcen, Wiederverwendung, Zukauf, geforderte Technologie ...“) [3, S. 13]. Diese Unterteilung ist hilfreich zur Strukturierung der Anforderungen und könnte im User-Interface der Visualisierung genutzt werden, um die Anforderungen zu kategorisieren.

Stakeholder

Stakeholder können „Personen oder Organisationen sein, die die Anforderungen eines Systems beeinflussen oder die von dem System beeinflusst werden.“ [1]. Beispielsweise wären die Endnutzer eines Systems Stakeholder, welche durch das System beeinflusst werden. Sie haben also ein Bedürfnis an das System, können dieses jedoch nicht selbst umsetzen. Im Gegensatz dazu stehen die Auftraggeber, beziehungsweise der Produkteigner (Product Owner), welche das System entwickeln und die Anforderungen festlegen.

Viele Stakeholder, wie bspw. der Product Owner, sind dabei nicht direkt in den täglichen Entwicklungsprozess des Systems involviert und haben daher nur wenig Überblick über den aktuellen Stand des Systems. Sie nehmen durch die gestellten Anforderungen jedoch großen Einfluss auf das System. Für eine effiziente Zusammenarbeit zwischen Stakeholdern und Entwicklern ist es also wichtig, dass die Anforderungen sowohl den Stakeholdern als auch den Entwicklern klar und verständlich sind.

Das kann vor allem bei großen und komplexen Systemen schwierig sein, da die Zahl der Anforderungen mit der Komplexität des Systems stark wächst. Durch die große Menge an Anforderungen kann in solchen Projekten schnell die Übersicht verloren gehen, weshalb es wichtig ist, die Anforderungen klar und übersichtlich zu dokumentieren und zu verwalten.

System

Die IREB definiert ein System als „Eine kohärente, abgrenzbare Menge von Elementen, die durch koordiniertes Handeln einen bestimmten Zweck erfüllen.“ [1] Das Wort System ist dabei ein Überbegriff für Produkte, Services, Geräte, Prozeduren und Werkzeuge und kann sowohl physisch als auch virtuell sein. Daher wird auch in dieser Bachelorarbeit das Wort System als Überbegriff für alle Arten von Systemen genutzt.

Requirements Engineering

Requirements-Engineering ist der Prozess, in dem Anforderungen an ein System erhoben, dokumentiert, analysiert, spezifiziert und validiert werden. Laut Chris Rupp besteht Requirements-Engineering dabei aus vier Haupttätigkeiten:

- Wissen vermitteln
- Gute Anforderungen herleiten
- Anforderungen vermitteln
- Anforderungen verwalten

Requirements-Engineering ist der erste Schritt bei der Entwicklung eines Systems und kann daher große Auswirkungen auf die Qualität und den Erfolg des Systems haben. Ein sauberes Requirements-Engineering verhindert Missverständnisse und Unklarheiten und verhindert so Fehler, welche sich durch die Systementwicklung ziehen und dann teuer und aufwendig behoben werden müssen, wenn sie erkannt sind. [2, S.20]

Diese Bachelorarbeit soll versuchen einen neuen Ansatz in der Vermittlung und Verwaltung von Anforderungen zu finden, um so langfristig die Qualität und Nützlichkeit der Anforderungen zu verbessern. Zudem soll die Visualisierung der Anforderungen helfen, bei großen Projekten, welche eine sehr große Zahl an Anforderungen besitzen, eine bessere Übersicht über die Anforderungen zu erhalten, um die Verwaltung der Anforderungen zu erleichtern. In der Vermittlung kann die Visualisierung den Stakeholdern eventuell helfen, die Anforderungen besser zu verstehen und so Missverständnisse und Unklarheiten zu vermeiden. So können teure Fehler in der Systementwicklung schon früher erkannt und vermieden werden.

3.2 reQlab

Die Software reQlab ist ein Requirements-Engineering-Tool, welches von der IT-Designers GmbH entwickelt wird. Es dient dazu, Requirements automatisiert zu analysieren und zu bewerten. Dafür nutzt die Software ein Large-Language-Model (LLM), welches natürlichsprachliche Anforderungen analysiert und bewertet. Daher werden in reQlab alle Anforderungen als natürlichsprachliche Anforderungen verfasst um dann vom LLM verarbeitet werden zu können. Die Software analysiert diese Anforderungen und gibt eine begründete Bewertung aus, ob die Anforderung gut oder schlecht ist und gibt Verbesserungsvorschläge. Das Ziel

von reQlab ist es, die Qualität der Anforderungen zu verbessern und so die Qualität des gesamten Systems zu steigern.

3.3 Virtuelle Realität

Für das Verständnis von Augmented Reality ist es wichtig, die Begriffe der virtuellen Realität (VR) zu kennen und zu verstehen. Dabei wird der Nutzer in eine virtuelle Welt versetzt, die durch Computer generiert wird. In virtueller Realität ist, im Gegensatz zu Augmented Reality, die gesamte Umgebung digital [vgl. 4, S.15].

Der Begriff der virtuellen Realität lässt sich noch genauer in die Begriffe immersive und nicht-immersive virtuelle Realität unterscheiden. Folgende nutzerbezogene Eigenschaften sind dabei Indikatoren für nicht-immersive virtuelle Realität:

- Der Nutzer steht nicht im Mittelpunkt.
- Der Nutzer ist nicht vollständig von digitalen Inhalten umgeben.
- Der Nutzer erfährt die virtuelle Realität als Beobachter anstatt als Teilnehmer.

Im Gegensatz dazu sind bei immersiver virtueller Realität alle äußeren Einflüsse so weit wie möglich reduziert und alle Indikatoren sollten auf immersive virtuelle Realität hinweisen. Der Nutzer sollte bei immersiver VR das Gefühl haben, sich selbst in der virtuellen Umgebung zu befinden [vgl. 5, S.23-24]. Daher geht es bei immersiver virtueller Realität vor allem um den visuellen Sinn, da dieser am meisten zur Immersion in die virtuelle Welt beiträgt. Jedoch wird in den meisten immersiven VR-Anwendungen auch der auditive Sinn über Kopfhörer oder Lautsprecher angesprochen, um die Immersion zu steigern. Auch der Tastsinn spielt heutzutage eine Rolle, da viele VR-Controller, wie bspw. die Meta Quest Touch Plus-Controller, dem Nutzer auch haptisches Feedback für Interaktionen geben.

3.4 Augmented Reality

Augmented Reality (AR) ist eine Technologie, die die reale Welt mit digitalen Informationen erweitert. Dabei wird ein ähnlicher Ansatz wie bei Virtueller Realität verfolgt, jedoch wird die reale Welt nicht komplett ersetzt, sondern nur erweitert. Der Nutzer sieht also weiterhin seine reale Umgebung, diese wird aber durch digitale Informationen ergänzt.

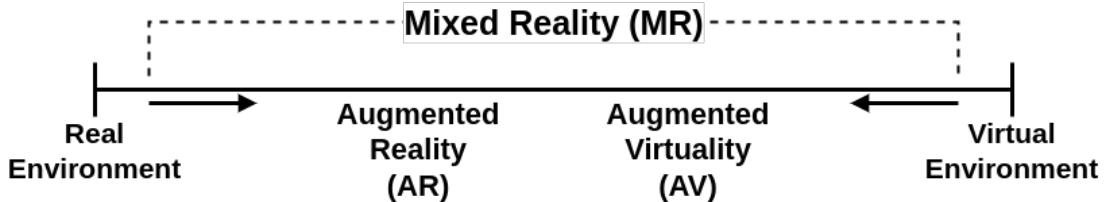


Abbildung 1: Realitäts-Virtualitäts-Kontinuum nach Milgram

Auf dem Realitäts-Virtualitäts-Kontinuum von Milgram, welches, wie in Abbildung 1 dargestellt, einen fließenden Übergang zwischen Realität und Virtualität beschreibt, liegt Augmented Reality zwischen der realen Welt und der virtuellen Welt [vgl. 6, S.9]. Daher fällt Augmented Reality unter den Überbegriff der Mixed Reality, da die reale Welt mit der digitalen Welt gemischt wird. Für diese Erweiterung der Realität müssen die Anzeigegeräte auch Informationen über die echte Umgebung sammeln können. Will man beispielsweise virtuelle Objekte in die reale Welt einfügen, so muss das Anzeigegerät die eigene Position kontinuierlich bestimmen können und die Position und Rotation des virtuellen Objekts anhand der Bewegungen des Nutzers anpassen.

Anzeigegeräte für Augmented Reality haben viele Gemeinsamkeiten mit Anzeigegeräten für Virtuelle Realität. Die Besonderheit von AR-Anzeigegeräten ist jedoch, dass sie die reale Welt mit digitalen Informationen erweitern. Das heißt sie müssen dem Nutzer auch eine Sicht auf die reale Welt ermöglichen und in diese Informationen einblenden. Beispielsweise kann das Display eines Anzeigegeräts transparent sein, sodass der Nutzer durch das Display hindurch sehen kann. Alternativ kann das Gerät eine oder mehrere Kameras besitzen, dessen Aufnahme auf undurchsichtige Bildschirme projiziert wird. Diese Methode nennt sich auch Image Passthrough. Es gibt verschiedene Arten von Anzeigegeräten, die für Augmented Reality genutzt werden können, wobei man allgemein zwischen 3 Hauptkategorien unterscheiden kann: Head-Mounted Displays, Hand-Held-Devices und Spatial Displays [7, S. 346]. Im Folgenden werden diese drei Kategorien genauer erläutert und einige Beispiele genannt.

3.4.1 Head-Mounted Displays

Head-Mounted Displays (HMDs) sind grundsätzlich Bildschirme, die direkt auf dem Kopf des Nutzers getragen werden. Durch ihre Nähe zu den Augen des Nutzers können sie ein großes Sichtfeld abdecken, ohne dabei selbst große Displays zu nutzen. Dadurch sind sie für volle Immersion im Normalfall kosteneffektiver als große Displays, wie bspw. eine Leinwand.

VR Headsets

Klassische VR-Headsets kann man sich vorstellen wie eine Skibrille mit 2 Bildschirmen, die auf dem Kopf getragen wird und üblicherweise 2 Displays hinter 2 Linsen hat, bzw. ein Display virtuell in 2 Displays aufteilt. Durch die 2 Bildschirme wird für jedes Auge ein eigenes Bild erzeugt, wodurch Inhalte in 3D dargestellt werden können. Außerdem können sie das volle Sichtfeld des Nutzers abdecken und so eine immersive Erfahrung schaffen. Jedoch entstehen durch die Nähe des Nutzers zu den Displays auch Probleme, wie bspw. der Screen-Door-Effekt, bei welchem die Zwischenräume zwischen den Pixeln sichtbar sind. Entscheidend dabei ist die Kennzahl der Pixel pro Grad (Pixel per Degree, PPD), welche angibt, wie viele Pixel auf einen Grad des Sichtfelds des Nutzers kommen. Eine Möglichkeit um die PPD zu erhöhen und so den Screen-Door-Effekt zu minimieren, ist das Einsetzen von Displays mit einer hohen Pixeldichte (Pixels per Inch, PPI). Beispielsweise hat die Oculus Quest 3 eine PPI von 1218 und erreicht damit einen PPD Wert von 25 Pixeln pro Grad [8]. Vergleichsweise hat das iPhone 15 Pro eine Pixeldichte von nur 460 PPI [9].

Die Position des Headsets muss dabei kontinuierlich bestimmt werden, um die Bewegungen der Nutzer zu verfolgen und so die virtuelle Welt anpassen zu können. Das Tracking kann beispielsweise durch Basisstationen (bspw. bei der Valve Index) realisiert werden, welche im Raum verteilt werden und mithilfe von Kameras und Sensoren die Position des Headsets bestimmen. Eine andere, modernere Methode des Headset Trackings ist das Tracking durch Kameras im Headset (bspw. bei der Oculus Quest 3). Diese Kameras nehmen die Umgebung des Nutzers auf und können mithilfe dieser Daten kontinuierlich die Position des Headsets bestimmen. Die Interaktion mit der Umgebung wird dann meist mit Controllern realisiert, deren Position und Rotation ebenfalls kontinuierlich durch Tracking Stationen oder eigene Kameras bestimmt werden muss. In VR-Headsets mit eigenen Kameras ist es auch möglich, die Hände des Nutzers zu tracken und als Eingabegeräte zu nutzen. Dabei können dann bestimmte Gesten oder Handbewegungen als Eingaben interpretiert und zur Interaktion genutzt werden.

Ursprünglich mussten VR-Headsets mit einem Computer verbunden werden, um die Rechenleistung für die Darstellung der Inhalte zu haben. Mit der Zeit wurden jedoch auch Standalone-VR-Headsets entwickelt, wie zum Beispiel das im Jahr 2024 auf den Markt gebrachte Apple Vision Pro. Diese verfügen über eine eigene Rechenleistung und können somit unabhängig von einem Computer genutzt werden.



Abbildung 2: Oculus Quest 3 VR-Headset

Jedoch sind nicht alle VR-Headsets für AR geeignet. Die meisten VR-Headsets besitzen keine Kamera, um die reale Welt aufzunehmen und dem Nutzer wiederzugeben. Nur Headsets wie bspw. die Oculus Quest 3, welche in Abbildung 2 dargestellt ist, oder die Apple Vision Pro besitzen eine Kamera nach außen, um die reale Welt aufzunehmen und so AR zu ermöglichen.

Durch all diese Technik, können die neusten VR-Headsets eine sehr hohe Immersion und audiovisuelle Qualität für Anwendungen in AR bieten. Sie sind besonders dafür geeignet hochauflösende dreidimensionale Inhalte in die reale Welt zu projizieren und so eine immersive Erfahrung zu schaffen. Doch für lange Anwendungszeiten und die Verwendung in der Öffentlichkeit sind sie meist zu groß und schwer.

Smart-Glasses

Smart-Glasses sind Brillen, die digitale Informationen in das Sichtfeld des Nutzers einblenden. Sie zeichnen sich durch die Transparenz bzw. Semi-Transparenz der Displays aus, sodass der Nutzer auch durch die Displays hindurch sehen kann. So kann auch ohne digitales Image-Passthrough ein AR-Effekt erzielt werden. Die Durchsicht durch die Displays sind zudem meist schärfer als bei Image-Passthrough in VR-Headsets, da direkt die echte reale Welt gesehen wird und nicht eine Kameraaufnahme.

Smart-Glasses sind außerdem meist leichter und kleiner als VR-Headsets, da sie meist nicht das volle Sichtfeld des Nutzers abdecken müssen und die Technik für Image-Passthrough nicht benötigt wird. Dadurch sind sie besser für den Alltag und für längere Anwendungzeiten geeignet als konventionelle VR- oder AR-Headsets.

3.4.2 Hand-Held-Devices

Hand-Held-Devices sind Geräte, die ein Nutzer in der Hand hält. Diese Kategorie besteht heutzutage hauptsächlich aus Smartphones und Tablets, da die meisten dieser Geräte über eine Kamera und einen Bildschirm verfügen, können sie fast alle für die Darstellung von AR-Inhalten genutzt werden. Das Smartphone oder Tablet ist dabei wie ein Fenster in die digitale Welt, durch das der Nutzer die erweiterte Realität sehen kann.

Die Immersion ist bei diesem Anzeigegerät jedoch relativ gering, da der Nutzer immer die reale Welt sieht und das Smartphone oder Tablet nur ein kleines Fenster in die digitale Welt ist. Allein durch die Entfernung der Geräte vom den Augen der Nutzer können sie nur ein vergleichsweise kleines Sichtfeld abdecken.

Jedoch ist die Nutzung von Smartphones für AR sehr weit verbreitet, da fast jeder ein Smartphone besitzt und so keine zusätzliche Hardware benötigt wird. Beispielsweise hatte das AR-Spiel Pokémon Go, welches 2016 veröffentlicht wurde, bereits Anfang 2019 über 1 Milliarde Downloads [10]. Bei dem Spiel werden Pokémons über die Smartphone-Kamera in die reale Welt projiziert, sodass der Nutzer sie fangen kann. Der Erfolg dieses simplen Konzepts zeigt das Potenzial der riesigen Nutzergruppe von Smartphones für AR-Anwendungen.

Auch können Smartphones als Displays für HMDs genutzt werden. Dabei ist jedoch meist die Kamera des Smartphones nicht mehr nutzbar, wodurch normalerweise die Wiedergabe von AR-Inhalten nicht möglich ist. Zudem ist die Pixeldichte von Smartphones meist geringer als bei speziellen AR-Anzeigegeräten, was die Immersion und Nutzererfahrung verschlechtern kann, da das Display sehr nah an den Augen des Nutzers ist.

3.4.3 Spatial Displays

Die bisher vorgestellten Konzepte für AR-Anzeigegeräte basieren auf Displays die direkt am Nutzer selbst befestigt sind. Im Kontrast dazu ist die meiste Technik bei Spatial Displays in der Umgebung des Nutzers verbaut. Der Nutzer selbst muss dabei keine spezielle Hardware tragen um die AR-Inhalte zu sehen [11]. Jedoch können beispielsweise zur Interaktion mit Inhalten Eingabegeräte genutzt werden, die der Nutzer in der Hand hält.

Da in vielen Fällen keine spezielle Hardware am Nutzer selbst befestigt werden muss, kann die Immersion durch Spatial Displays sehr hoch sein. Auch das ergonomische Nutzererlebnis über lange Zeiträume kann durch Spatial Displays verbessert werden, da der Nutzer nicht durch das Tragen von schwerer Hardware belastet wird.

Ein Nachteil von Spatial Displays ist der Abstand der Displays zum Nutzer. Da sie weiter vom Nutzer entfernt sind als HMDs oder Hand-Held-Devices können sie mit der gleichen Displaygröße nur ein kleineres Sichtfeld abdecken. Daher werden bei Spatial Displays oft sehr große Displays genutzt, um trotzdem ein großes Sichtfeld abzudecken. Dadurch sind Spatial Displays deutlich teurer und aufwendiger in der Installation als HMDs oder Hand-Held-Devices. Sie werden daher meist in aufwendigen Anwendungen genutzt und sind für den privaten Gebrauch eher ungeeignet.

4 Technologien

4.1 Oculus Quest 3

Im Laufe der Bachelorarbeit wird als Anzeigegerät für die AR-Anwendung die Oculus Quest 3 verwendet. Dabei handelt es sich um ein Standalone-AR- und VR-Headset, welches von Meta (ehemals Facebook) entwickelt wurde und Ende 2023 auf den Markt kam. Das HMD ist mit einem eigens für mobile XR-Geräte entwickelten Qualcomm Snapdragon XR2 Gen2 Prozessor ausgestattet und ist dadurch selbst in der Lage auch anspruchsvolle Inhalte zu rendern. Es muss nicht wie viele andere HMDs an einen Computer oder ein Smartphone angeschlossen werden, sondern kann direkt verwendet werden. Über zwei Displays mit jeweils einer Auflösung von 2064 x 2208 Pixeln und einer Bildwiederholrate von 120Hz bietet das Headset eine hohe Bildqualität und eine flüssige Darstellung von Inhalten. Durch die Kombination der geringen Latenz aufgrund des On-Board-Prozessors und der hohen Bildwiederholrate sollte das Gerät eine hohe Immersion und ein angenehmes Nutzungserlebnis bieten.



Abbildung 3: Frontal ausgerichtete Kameras und Tiefenprojektor der Oculus Quest 3

Darüber hinaus verfügt die Oculus Quest 3, wie auf Abbildung 3 zu sehen ist, über zwei RGB-Kameras (untere Kameras), zwei IR-Kameras (obere Kameras) sowie einen Tiefenprojektor (schwarze Fläche in der Mitte) auf der Vorderseite, welche die Umgebung des Nutzers erfassen können und so AR-Anwendungen mittels Image-Passthrough

ermöglichen. Zusätzlich zu den 2 RGB-Kameras sind insgesamt noch 4 Infrarot-Kameras auf dem Headset verbaut, zwei davon auf der Vorderseite, direkt über den RGB-Kameras, und zwei auf der Unterseite. Über diese Kameras wird auch die Position des Nutzers im Raum kontinuierlich getrackt und angepasst, sodass für das Headset keine externen Trackingstationen notwendig sind.



Abbildung 4: Die beiden nach unten gerichteten Kameras der Oculus Quest 3

Die in Abbildung 4 dargestellten nach unten gerichteten Kameras, dienen dazu zu jedem Zeitpunkt die Hände oder Controller des Spielers mit einer Kamera erfassen zu können. Alternativ zu Controllern als Eingabegeräte kann das Headset mithilfe der 4 IR-Kameras und 2 RGB-Kameras die Hände des Nutzers erkennen und tracken, sodass die eigenen Hände in VR- und AR-Anwendungen als Eingabegerät verwendet werden können. [8]

4.2 WebGL

WebGL ist eine JavaScript-API, die das Rendern von 3D-Grafiken in einem Webbrowser ermöglicht. Die WebGL-Spezifikation wird von der Khronos Group entwickelt, einer Non-Profit-Organisation, welche als ein Zusammenschluss aus über 180 Unternehmen aus der Tech-Industrie technologische Standards entwickelt. Darunter sind auch die Unternehmen hinter den größten Browsern der Industrie: Google (Chrome), Mozilla (Firefox), Apple (Safari) und Microsoft (Edge) [12, 13]. Aufgrund dieser Mitarbeit verschiedener Unternehmen und der offenen Entwicklung der Spezifikation kann WebGL in fast allen modernen Webbrowsern verwendet werden.

WebGL basiert dabei auf der OpenGL ES Spezifikation, welche, ebenfalls von der Khronos Group, speziell für Embedded Systems und mobile Systeme, also zum Großteil Systeme ohne eigene Grafikkarte, entwickelt wurde [14]. Aufgrund dieser Ausrichtung auf mobile Geräte, können mit WebGL Anwendungen entwickelt werden, die auf einfachen Geräten wie Smartphones oder aber auch auf VR- und AR-Headsets ohne zusätzliche Rechenleistung laufen [15, S.3].

4.3 WebXR

WebXR ist eine standardisierte API für Webanwendungen, welche es ermöglicht, immersive VR und AR Anwendungen für das Internet zu erstellen. Das World Wide Web Consortium (W3C), welches die WebXR-Standards definiert beschreibt WebXR wie folgt: „Die WebXR Device API bietet die notwendigen Schnittstellen, damit Entwickler ansprechende, komfortable und sichere immersive Anwendungen im Web für eine Vielzahl von Hardware-Formfaktoren erstellen können.“ [aus dem Englischen mit DeepL 16, 1. Introduction] Mit WebXR entwickelte Anwendungen können als Webanwendungen direkt von einem Webbrowser eines AR- oder VR-Geräts aus aufgerufen werden, ohne dass eine zusätzliche Installation der Anwendung notwendig ist.

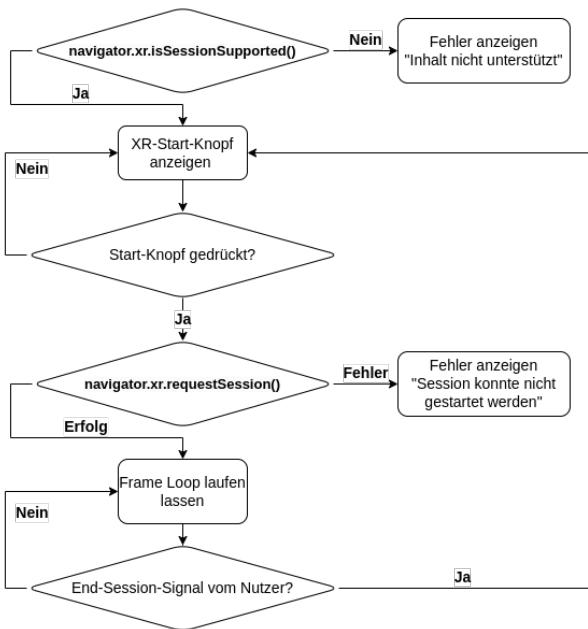


Abbildung 5: Application Flow von WebXR-Anwendungen

Quelle: nach [16, Kap. 1.2. Application Flow]

Der Flow von WebXR-Anwendungen ist in Abbildung 5 dargestellt. Dabei wird zuerst beim Aufrufen der Anwendung geprüft, ob die angegebene Art von XR-Inhalt von der Hardware und dem User Agent (UA) unterstützt werden. Mit dem User Agent (UA) der Software, ist dabei die Software gemeint, mit welcher der Nutzer auf die Anwendung zugreift, also der Webbrowser des Nutzers. Ist die Art des XR-Inhalts von der Technik des Nutzers unterstützt, kann die XR-Session vom Nutzer manuell über einen Knopf gestartet werden. Startet die Session erfolgreich, wird der Frame Loop der Anwendung gestartet. Der Frame Loop ist eine Schleifenfunktion, die kontinuierlich während der XR-Session ausgeführt wird um die einzelnen Frames, bzw. Bilder, für das XR-Display zu rendern. Dieser Frame Loop läuft, bis die Session durch den UA beendet wird. Befindet sich der UA noch auf der Seite

der WebXR-Anwendung, wird wieder der Knopf zum Starten der XR-Session angezeigt, falls der Nutzer direkt wieder eine neue Session starten möchte.

4.3.1 Three.js

Three.js ist eine Open-Source JavaScript-Bibliothek unter der MIT-Lizenz, die auf WebGL basiert und die Erstellung und Darstellung von 3D-Grafiken in Webanwendungen vereinfacht. Die Bibliothek bietet eine Vielzahl von Funktionen, die es Entwicklern ermöglichen, detaillierte 3D-Modelle und -Szenen zu erstellen und zu rendern [17]. Dabei unterstützt Three.js auch die Erstellung von AR- und VR-Anwendungen mithilfe von WebXR.

Es existieren auch einige Frameworks die auf der Three.js API basieren und die Entwicklung von AR- und VR-Anwendungen und 3D-Anwendungen generell weiter vereinfachen. Ein Beispiel dafür ist das Framework A-Frame, welches auf die Entwicklung von AR und VR Anwendungen spezialisiert ist und Features einer fast vollwertigen Game-Engine bietet [18].

Zudem wird Three.js von vielen Entwicklern und Unternehmen verwendet und hat dadurch eine große Community und so viele Ressourcen und Tutorials, die Entwicklern helfen können, ihre Anwendungen zu erstellen und zu debuggen.

Diese Vielfalt an Features und die große Community machen Three.js, vor allem in Kombination mit A-Frame, zu einer interessanten Option für die Entwicklung von AR- und VR-Anwendungen. Jedoch ergaben sich in den ersten Tests der Funktionalität von Three.js und A-Frame im Browser der Oculus Quest 3 Probleme mit der Erkennung der VR-Brille und der Darstellung von AR-Inhalten.

4.3.2 Babylon.js

Babylon.js ist eine Open-Source Rendering- und Game-Engine verpackt in einer JavaScript-Bibliothek die unter Anderem von einem Team von Entwicklern bei Microsoft entwickelt wird. Es bietet Support für WebGL 1.0/2.0 sowie WebGPU und hat eine Vielzahl von Funktionen, die es Entwicklern ermöglichen, detaillierte 3D-Modelle und -Szenen zu erstellen und zu rendern. Außerdem bietet es native Unterstützung und eine Dokumentation für WebXR, wodurch die Entwicklung von VR- und AR-Anwendungen vereinfacht wird [19].

Die Bibliothek bietet dabei auch viele Quality-of-Life-Features, wie beispielsweise einen Szenen-Explorer und -Inspektor, der sich für die Entwicklung und das Debugging von 3D-Szenen eignet und mit einer Zeile Code in die Anwendung eingebunden werden kann. Über den Szenen-Explorer, welcher in Abbildung 6 links dargestellt ist, können aus dem Szenengraph der aktuellen Szene Objekte ausgewählt werden um im Szenen-Inspektor

eine Detailansicht des Objekts und seinen Eigenschaften zu erhalten. Mithilfe des Szenen-Inspektors, welcher in Abbildung 6 rechts dargestellt ist, können dann in Echtzeit die Eigenschaften von Objekten betrachtet und verändert werden.



Abbildung 6: Szenen-Explorer und -Inspektor von Babylon.js

5 Implementierung

5.1 Entwicklungsumgebung für WebXR und Oculus Quest 3

Die Entwicklung von WebXR-Anwendungen für die Oculus Quest 3 erfordert eine spezielle Entwicklungsumgebung für einen effizienten Entwicklungsprozess. Dabei müssen verschiedene Technologien und Tools miteinander kombiniert werden, um eine reibungslose Entwicklung und ein möglichst schnelles Testen der Anwendung zu ermöglichen.

Der erste Schritt ist das Anzeigen der WebXR-Anwendung auf der Oculus Quest 3. Das Problem dabei im Vergleich zu „normalen“ Webanwendungen ist, dass WebXR-Anwendungen nur über HTTPS aufgerufen werden können. Das bedeutet, dass die Anwendung über HTTPS gehostet werden muss, um direkt vom Browser der Oculus Quest 3 aufgerufen werden zu können. Hierfür gibt es verschiedene Möglichkeiten, wie beispielsweise das Erstellen eines eigenen Zertifikats für den lokalen Entwicklungsrechner oder das Hosting der Anwendung auf einem Server mit HTTPS-Unterstützung. Für den Rahmen der Entwicklung dieser Bachelorarbeit wird die Anwendung jedoch, wie auch in der Artikelserie des Taikonautenmagazins [20, Part 0/8] empfohlen, mit LocalTunnel gehostet, um die Anwendung direkt von der Oculus Quest 3 aus testen zu können. LocalTunnel erstellt einen temporären HTTPS-Link, über den die Anwendung aufgerufen werden kann, ohne dass ein eigenes Zertifikat oder ein eigener Server notwendig ist. Dafür muss die Anwendung nur lokal auf dem Entwicklungsrechner laufen und der LocalTunnel-Client gestartet sein, um den temporären Link zu generieren. Als zusätzliche Sicherheit muss dann beim Aufrufen der Seite noch die IP-Adresse des Entwicklungsrechners angegeben werden, um sicherzustellen, dass nur der Entwickler die Anwendung testen kann. Dies muss in der Regel jedoch nur einmal nach jedem Neustart oder Crash gemacht werden, da der Link für die Dauer der Sitzung gespeichert wird.

Der nächste Schritt, wenn Zugriff mit der Oculus Quest 3 auf die WebXR-Anwendung besteht, ist die Anzeige von Entwickler-Tools und Debugging-Informationen der Oculus Quest 3. Da die Oculus Quest 3 auf Android basiert, kann auf dem Entwicklungsrechner Android Debug Bridge (ADB) installiert werden, um über USB eine Verbindung zur Oculus Quest 3 herzustellen. Die Verbindung muss noch in der VR-Brille bestätigt werden, um den Zugriff auf die Entwickleroptionen zu ermöglichen. Ist das geschehen, erscheint die Quest 3 mit ihren geöffneten Websites in der Geräteliste der Chrome DevTools, die über die URL `chrome://inspect/#devices` aufgerufen werden können. Dort können dann die

Entwickler-Tools der Oculus Quest 3 geöffnet werden, um beispielsweise die Performance der Anwendung zu überwachen und Fehlermeldungen zu sehen.

Für viele Aspekte der Entwicklung von XR-Anwendungen, wie beispielsweise einfache Tests von Interaktionen wie einzelnen Klicks können auch über einen WebXR-Emulator direkt am Entwicklungsrechner getestet werden. In dieser Arbeit wird dafür die Chrome-Erweiterung Immersive Web Emulator von Meta verwendet, die es ermöglicht, WebXR-Anwendungen direkt im Browser zu testen. Mit dieser Erweiterung wird für WebXR ein einfacher Raum mit einem Headset und den beiden Meta Quest Controllern simuliert. Das Headset sowie die beiden Controller können unabhängig voneinander über die Erweiterung positioniert und gesteuert werden. In dem in Abbildung 7 links dargestellten Panel können verschiedene Interaktionen, wie beispielsweise Klicks oder das Bewegen der Controller, simuliert werden, um die Anwendung zu testen. Der Emulator generiert dann eine live Vorschau der Anwendung, die direkt im Browser angezeigt wird.

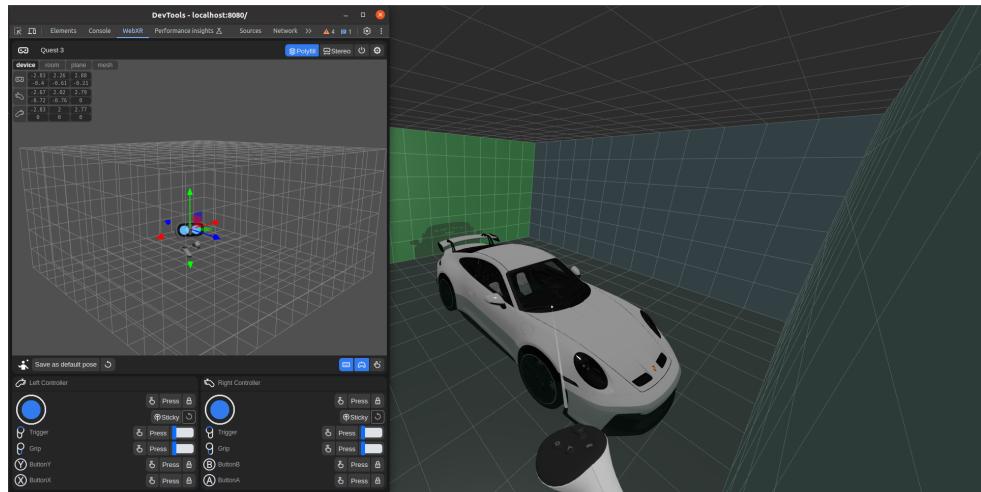


Abbildung 7: Screenshots der Immersive Web Emulator Erweiterung in Chrome

Das Testen der Anwendung mit dieser Erweiterung ist deutlich schneller und einfacher als das Testen auf der Oculus Quest 3, da die Anwendung direkt im Browser getestet werden kann und keine Verbindung zur Oculus Quest 3 notwendig ist. Zudem sind die Ladezeiten beim Neuladen der Anwendung deutlich kürzer, da die Anwendung nicht auf die Übertragung der Daten auf die Oculus Quest 3 warten muss. Dennoch ist es wichtig, die Anwendung auch regelmäßig auf der Oculus Quest 3 zu testen, da die Performance und die Interaktionen auf dem Emulator nicht immer exakt der Realität entsprechen.

5.2 Implementierung der Anwendung

In diesem Kapitel wird der Prozess der Implementierung der Anwendung beschrieben. Dabei wird zunächst die Entwicklung der grundlegenden zu implementierenden Konzepte

beschrieben, bevor auf die tatsächliche Implementierung der ausgearbeiteten Interaktionskonzepte eingegangen wird.

5.2.1 Interaktionskonzepte für Requirements

Die Bachelorarbeit beschäftigt sich mit der Darstellung von Requirements in AR beziehungsweise VR. Dabei soll untersucht werden, wie Anforderungen in einer 3D-Umgebung dargestellt werden könnten. Dazu werden verschiedene Interaktionskonzepte entwickelt, welche als Basis für die Implementierung mit WebXR dienen.

Beispiel 1: Explodierende Bauteile

Das erste untersuchte Interaktionskonzept ist hauptsächlich für die Darstellung von Anforderungen von Produkten gedacht. Die Idee ist, ein Produkt in einer Animation in seine einzelnen Bauteile zu zerlegen und die Anforderungen auf ihren zugehörigen Bauteilen darzustellen. In der Animation werden die Bauteile von einem Punkt in der Mitte des Produkts nach außen bewegt, sodass sie sich um den Ursprungspunkt des Produkts herum anordnen. Beispielsweise könnte ein Auto so zerlegt werden, dass bei der Animation die Räder, die Karosserie, der Motor und die Innenausstattung einzeln als eigene Objekte sichtbar werden. So kann der Nutzer das gesamte Produkt betrachten und sich dann auf Wunsch einzelne Bauteile und deren Anforderungen genauer ansehen.

Die Anforderungen sollen dabei als Text auf UI-Panelen dargestellt werden, die an den zugehörigen Bauteilen angebracht sind.

Am bereits genannten Beispiel des Autos wird klar, dass die Komplexität der Darstellung bei vielen Bauteilen schnell ansteigt und die Übersichtlichkeit verloren gehen kann. Daher ist es bei umfangreichen Produkten eventuell sinnvoll, die Darstellung der Bauteile in mehreren Schritten zu realisieren. Zum Beispiel könnte in einer Übersicht das gesamte Auto in wenigen Bauteilen dargestellt werden, indem beispielsweise ein Rad, das eigentlich aus Reifen, Felge, Radmuttern, Bremsscheibe etc. besteht, als ein einzelnes Objekt dargestellt wird. Will der Nutzer dann die Räder genauer betrachten, kann er in eine Detailansicht wechseln, in der nur ein Rad mit all seinen Bauteilen animiert wird. So lässt sich eine hohe Komplexität der Darstellung erreichen, ohne dass die Übersichtlichkeit verloren geht.

Für diese Darstellung soll der Nutzer zunächst mithilfe eines Controllers einen Ort für die Darstellung im Raum auswählen. Dieser Ort wird als Ursprungspunkt für die Darstellung der Bauteile verwendet. Dann soll der Nutzer frei durch die Explosionsanimation navigieren können, um sich die Bauteile aus verschiedenen Perspektiven anzusehen.

Auch bei Software-Requirements ist es denkbar, diese in ihre Komponenten zu zerlegen und zu diesen Komponenten zugehörige Anforderungen darzustellen. Jedoch bietet die Darstellung in AR bzw. VR hierbei quasi keine Vorteile im Gegensatz zu einer 2D-Darstellung auf

einem Bildschirm. Bei Software-Anwendungen ist die einfache Darstellung auf einem Bildschirm näher an der tatsächlichen Laufumgebung der meisten Softwares als bei physischen Produkten, bei denen durch eine dreidimensionale Darstellung ein Mehrwert entstehen kann.

Bei diesem Konzept soll die Realisierbarkeit solcher Darstellungen für physische Produkte untersucht werden. Dabei muss die Kosten-Nutzen-Relation kritisch betrachtet werden, da die Implementierung einer solchen Darstellung sehr aufwändig sein kann und daher einen hohen Mehrwert gegenüber anderer Darstellungen bieten muss.

Beispiel 2: Wolken von Anforderungen

Der Ansatz der explodierenden Bauteile ist aufgrund der Individualität des Konzepts sehr zeitaufwendig und komplex zu implementieren. Zudem ist dieser Ansatz nur für die Darstellung von Produkten, also physischen Systemen, geeignet. Daher wird ein weiteres Interaktionskonzept entwickelt, welches sich theoretisch auch automatisiert generieren lässt und für alle Arten von Anforderungen geeignet ist.

Die grundlegende Idee ist, Anforderungen in Wolken von Texten darzustellen, also als eine Gruppierung von UI-Elementen im Raum. Hierbei soll eine räumliche Gruppierung der Anforderungen nach verschiedenen Kriterien möglich sein. Beispielsweise könnten Anforderungen, die zu einem bestimmten Feature gehören, in einer Wolke gruppiert werden, während Anforderungen, die zu einem anderen Feature gehören, in einer anderen Wolke gruppiert werden. Durch die räumliche Anordnung der Wolken kann der Nutzer schnell erkennen, welche Anforderungen zusammen gehören und welche nicht.

Dabei soll es auch möglich sein in Wolken hinein- und herauszuzoomen, um die Granularität der angezeigten Anforderungen zu erhöhen. Gehen wir dabei wieder vom Beispiel des Autos aus, könnte es möglich sein in die Wolke der Räder hineinzuzoomen, um die Anforderungen an die Reifen, Felgen, Radmuttern etc. zu sehen. Daraufhin kann wieder herausgezoomt werden, um die Anforderungen an das gesamte Auto zu sehen. Eine weitere Möglichkeit wäre es, auf den Anforderungspanels Filtermöglichkeiten für verschiedene Beziehungen der Anforderung darszustellen, um dann bei einer Auswahl in die neue Anforderungswolke zu zoomen. Diese Interaktion und die räumliche Anordnung der Wolken sollen dem Nutzer helfen, auch bei einer großen Anzahl von Anforderungen einen Überblick zu behalten und schnell die gewünschten Anforderungen zu finden.

Bei diesem Konzept soll vor allem der Vorteil gegenüber einer 2D-Darstellung kritisch untersucht werden. Denn die Darstellung der Anforderungen in Wolken ist prinzipiell auch in 2D möglich, auch mit der Interaktionsmöglichkeit des Hinein- und Herauszoomens. Daher soll untersucht werden, ob die räumliche Anordnung der Anforderungen in AR tatsächlich einen Mehrwert gegenüber einer 2D-Darstellung bietet und ob die Interaktionen intuitiv und effizient sind.

Beispiel 3: Anforderungen als 3D-Objekte

5.2.2 Implementierung der Interaktionskonzepte

In den folgenden Abschnitten wird auf die Implementierung der zuvor beschriebenen Interaktionskonzepte eingegangen. Dabei werden grundlegende Konzepte und Technologien vorgestellt und erklärt, die für die Implementierung notwendig sind. Zudem werden Screenshots der implementierten Konzepte gezeigt und die Funktionsweise der Interaktionen beschrieben. Dabei werden auch die Herausforderungen und Probleme bei der Implementierung aufgezeigt und diskutiert.

Vor der Implementierung des Konzepts muss zunächst eine grundlegende WebXR-Anwendung erstellt werden, die die Interaktionen mit dem Controller ermöglicht. Hierfür wird das Skelett einer WebXR-Anwendung aus einer Artikelserie des Taikonauten-Magazins verwendet, welches als Basis für die Implementierung dient [20]. Die Anwendung nutzt bereits die vom Nutzer gescannten Umgebungen, um einen virtuellen Raum zu erstellen, in dem die Interaktionen stattfinden. Dabei werden Wände und Böden der Umgebung erkannt und als Mesh in die Szene eingefügt, um dem Nutzer eine Interaktion mit der realen Umgebung zu ermöglichen. Zudem wurden schon einige Interaktionen des Controllers, wie das Auswählen von Objekten durch Raycasting, implementiert, die als Basis für die Implementierung der Interaktionskonzepte dienen.

Explodierende Bauteile

Der erste Schritt bei der Implementierung des Interaktionskonzepts der „explodierenden“ Bauteile ist die Erstellung eines 3D-Modells, welches die Bauteile des Produkts enthält. Dabei muss jedes Bauteil als eigenes Objekt im 3D-Modell vorhanden sein, um sie in der Animation separat darstellen und referenzieren zu können. Für die erste Implementierung wird ein einfaches 3D-Modell eines Tetris Blocks verwendet, welcher aus 4 verschiedenfarbigen Bauteilen besteht.

Dieses Modell wurde in Blender erstellt und als 3D-Modell im glTF-Format, welches wie WebXR von der Khronos Group entwickelt wurde, in die Anwendung exportiert. Das glTF-Format ist ein offenes 3D-Dateiformat, welches für die effiziente Übertragung von 3D-Modellen im Web optimiert ist und die Dateigröße möglichst klein hält. Ein weiterer Vorteil des glTF-Formats ist, dass es auch Animationen und Materialien unterstützt, die im 3D-Modell enthalten sind. So kann die Animation der Bauteile auch direkt in der Modellierungssoftware erstellt und in das glTF-Modell mit eingebettet werden.

Der nächste Schritt ist das Platzieren des erstellten 3D-Modells in der WebXR-Umgebung. Dazu wird das Prinzip des Raycastings verwendet, um dem Nutzer zu ermöglichen, mit dem Controller einen Punkt im Raum auszuwählen, an dem das 3D-Modell platziert werden soll. Der Raum, in dem sich der Anwender befindet, muss dafür zu Beginn einmalig in den Meta Quest Einstellungen eingescannt werden. Ist das geschehen, werden durch

das Skelett der Anwendung des Taikonauten-Magazins bereits die Wände und Böden der Umgebung als Meshes erkannt und in die Szene eingefügt. Dann wird vom Controller ein Strahl in die Szene geschossen, und der Punkt, an dem der Strahl ein Objekt trifft, wird als Event zurückgegeben.

An diesem Punkt wird dann ein Ankerpunkt erstellt, an den das 3D-Modell angehaftet wird. Dieser Ankerpunkt ist ein Babylon.js-Objekt, welches einen Punkt im Raum in AR an einer bestimmten Position darstellen kann. Das 3D-Modell wird dann an diesen Ankerpunkt angehängt, sodass es sich relativ zu diesem Punkt bewegt.

Ist das 3D-Modell platziert, kann die Animation des Produkts gestartet werden. Dafür wird ein Knopf des Controllers als Start-Button für die Animation verwendet, mit dem sich die Animation vorwärts und wieder rückwärts abspielen lässt. Für die Animation wird die aus Blender in das glTF-Modell eingebackene Animation verwendet. Dafür wird in Babylon.js eine Animationsfunktion erstellt, die die Animation des Modells über einige Parameter, wie beispielsweise den Start- und Endframe der Animation, steuert. So kann für die Vorwärts- und Rückwärtsanimation die gleiche Funktion verwendet werden, indem die Start- und Endframe-Parameter basierend auf einem globalen Boolean, der den Animationsstatus speichert, gesetzt werden.

Beim Abspielen der Animation wird jedes Bauteil des Produkts in einer Schleife durchgegangen und dessen Animation gestartet. Diese Schleife wird dann auch genutzt, um den einzelnen Bauteilen ihre Anforderungen als Text-UI-Elemente zuzuweisen. Dafür wird in Babylon.js eine Fläche erstellt, auf der der Text dargestellt wird, und diese Fläche an das Bauteil angehängt. Die Fläche kann dabei auch als Knopf funktionieren, um bei einem Klick beispielsweise das Bauteil zu vergrößern. Diese UI-Elemente werden aber, wie in Abbildung 8 zu sehen ist, nur angezeigt, wenn das Produkt gerade „explodiert“ ist und nicht, wenn das Produkt gerade im Normalzustand ist.

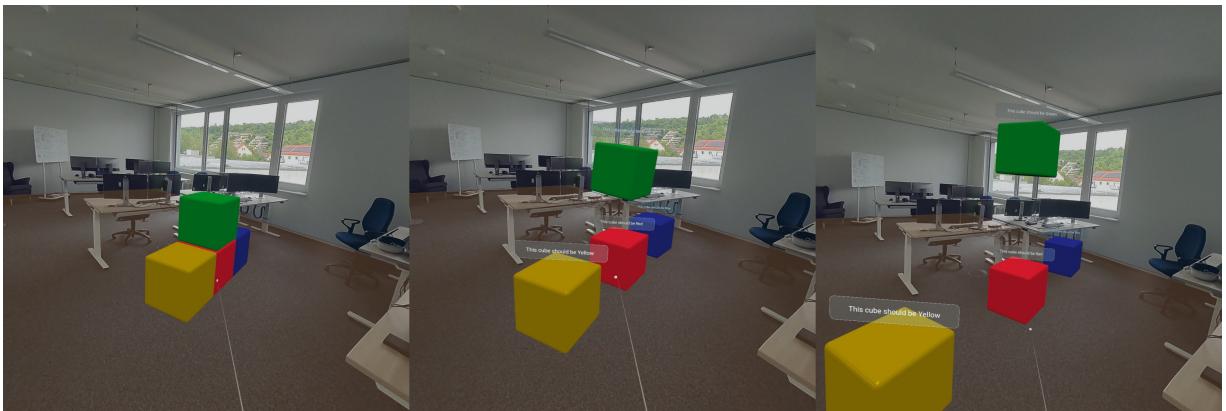


Abbildung 8: Screenshots des explodierenden Tetris-Blocks mit Anforderungen in AR

Implementation an einem komplexeren Modell

Nachdem das Interaktionskonzept funktionell am einfachen Modell eines Tetris-Blocks implementiert wurde, ist die nächste Herausforderung die Implementierung an einem komplexeren Modell. Dafür wird ein relativ detailliertes 3D-Modell eines Porsche 911 von der 3D-Asset-Website Sketchfab verwendet, welches von dem Nutzer Abdul Azim Sharif erstellt und unter der CC BY 4.0 Lizenz veröffentlicht wurde [21]. Das Modell, welches in Abbildung 9 zu sehen ist, wurde ausgewählt, da es viele verschiedene Bauteile enthält, die in der Animation separat dargestellt werden können.



Abbildung 9: Screenshot des in der Anwendung genutzten Porsche Modells

Bei dem in dieser Bachelorarbeit verwendetem Modell ist es jedoch wichtig zu beachten, dass es sich um kein vollständig akkurate und funktionales Modell eines Autos handelt. Beispielsweise existieren keine Achsen an denen die Räder hängen. Für den anschaulichen Zweck dieser Bachelorarbeit ist das Modell jedoch ausreichend, da es genug „reale“ Bauteile enthält um das Interaktionskonzept zu demonstrieren. Würde das Interaktionskonzept in einer echten Anwendung für Kunden verwendet werden, sollte jedoch mit möglichst akkuren CAD-Modellen gearbeitet werden, um die Anforderungen an die Bauteile möglichst genau darzustellen. Auf die Möglichkeit einer solchen professionellen Anwendung wird in der Diskussion eingegangen.

Für die Verwendung des Modells in der Anwendung muss als nächstes eine Animation mit dem Modell erstellt werden, die die Bauteile des Autos in ihre Einzelteile zerlegt. Dafür wird in Blender eine Animation erstellt, welche einige Bauteile, wie beispielsweise die Verkleidung, die Räder und der Spoiler, nach außen weg bewegt. Zum besseren Verständnis ist in Abbildung 10 eine Bildsequenz der Animation des Porsche Modells zu sehen.



Abbildung 10: Bildsequenz der Animation des Porsche Modells

Um den Bauteilen später ihre Anforderungen zuzuweisen, müssen die relevanten Bauteile identifizierbar sein. Dafür wird in Blender jedem animierten Objekt eine ID (bspw. #1_Rad, #2_Lichter) am Anfang des Objektnamens zugewiesen, die später in der Anwendung als Referenz für die Anforderungen dient. Die IDs werden dabei zusätzlich in anzeigenende und nicht-anzeigenende IDs unterteilt, um bei Bauteilen welche aus mehreren kleinen Objekten bestehen in der großen Ansicht nur die Anforderungen des Hauptobjekts anzuzeigen. Beispielsweise besteht ein Rad aus Reifen, Felge, Bremsscheibe etc., wobei nur das Rad als Hauptobjekt die Anforderungen an das Rad anzeigt und die anderen Objekte die Anforderungen an sich selbst in der Detailansicht. Um die Objekte jedoch einfach ihren Elternobjekten zuordnen zu können, wie beispielsweise für die Klick-Abfrage des Rads, werden die IDs der Elternobjekte in den IDs der Kindobjekte gespeichert. Dabei jedoch ohne Unterstrich „_-“ hinter der ID, wodurch sie beim Erstellen der Anforderungspanels ignoriert werden.

Für die Änderung auf die detaillierte Radansicht wird einer Abfrage hinzugefügt, die prüft, ob der Nutzer auf ein Rad klickt. Dafür wird wieder das Raycasting-Prinzip verwendet, um den Punkt zu bestimmen, an dem der Controller auf ein Objekt trifft. Ist das getroffene Objekt Teil vom Rad, wird an der Stelle ein neuer Ankerpunkt erstellt, an dem die Detailansicht des Rads angezeigt wird. Der alte Ankerpunkt und die Bauteile des Autos werden dann ausgeblendet, und die Bauteile des Rads werden an den neuen Ankerpunkt angehängt. So kann der Nutzer die Anforderungen an das Rad in einer Detailansicht betrachten und bei Bedarf wieder zur Gesamtansicht des Autos zurückkehren.

Wolken von Anforderungen

5.3 User Tests

6 Zusammenfassung

6.1 Ergebnisse

6.2 Fazit

6.3 Ausblick

Literatur

1. INTERNATIONAL REQUIREMENTS ENGINEERING BOARD (IREB). *CPRE Glossary* [online]. 2024 [besucht am 2024-03-20]. Abgerufen unter: <https://www.ireb.org/de/cpre/glossary/>. Definition übersetzt von Englisch auf Deutsch.
2. RUPP, Christine; DIE SOPHISTEN. *Requirements-Engineering und -Management*. 7., aktualisierte und erweiterte Auflage. München: Carl Hanser Verlag GmbH & Co. KG, 2020. Abgerufen unter DOI: 10.3139/9783446464308.
3. HRUSCHKA, Peter. *Business Analysis und Requirements Engineering*. 3., updated edition. München: Carl Hanser Verlag GmbH & Co. KG, 2023. Abgerufen unter DOI: 10.3139/9783446478190.
4. DALTON, Jeremy; ACKER, Olaf. *Immersive Unternehmenswelten: Wie Augmented, Mixed und Virtual Reality die Wirtschaft transformieren*. Schäffer-Poeschel Stuttgart, 2023. Abgerufen unter DOI: 10.34156/978-3-7910-5689-0.
5. WÖLFEL, Matthias. *Immersive Virtuelle Realität*. Karlsruhe: Springer Vieweg Berlin, Heidelberg, 2023. Abgerufen unter DOI: 10.1007/978-3-662-66908-2.
6. MILGRAM, Paul; COLQUHOUN, Herman u. a. A taxonomy of real and virtual world display integration. *Mixed reality: Merging real and virtual worlds*. 1999, Jg. 1, Nr. 1999, S. 1–26.
7. CARMIGNIANI, Julie; FURHT, Borko; ANISETTI, Marco; CERAVOLO, Paolo; DAMIANI, Ernesto; IVKOVIC, Misa. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*. 2011, Jg. 51, Nr. 1, S. 341–377. ISSN 1573-7721. Abgerufen unter DOI: 10.1007/s11042-010-0660-6.
8. META PLATFORMS, INC. *Meta Quest 3 Homepage* [online]. 2024 [besucht am 2024-04-03]. Abgerufen unter: <https://www.meta.com/de/quest/quest-3/>.
9. APPLE INC. *iPhone 15 Pro - Technische Daten* [online]. 2023 [besucht am 2024-04-03]. Abgerufen unter: <https://support.apple.com/de-de/111829>.
10. LINDNER, Jannik. *Must-Know Pokemon Go Usage Statistics* [online]. 2023 [besucht am 2024-04-08]. Abgerufen unter: <https://gitnux.org/pokemon-go-usage-statistics/>.
11. BIMBER, Oliver; RASKAR, Ramesh. Modern approaches to augmented reality. In: *Acm siggraph 2006 courses*. 2006, 1–es.
12. KHRONOS GROUP. *WebGL Overview* [online]. 2024 [besucht am 2024-04-08]. Abgerufen unter: <https://www.khronos.org/webgl/>.

13. KHRONOS GROUP. *About the Khronos Group* [online]. 2024 [besucht am 2024-04-08]. Abgerufen unter: <https://www.khronos.org/about>.
14. KHRONOS GROUP. *OpenGL ES Overview* [online]. 2024 [besucht am 2024-04-08]. Abgerufen unter: <https://www.khronos.org/opengles/>.
15. BARUAH, Rakesh. *AR and VR Using the WebXR API: Learn to Create Immersive Content with WebGL, Three.js, and A-Frame*. Entering VR Through WebXR. Berkeley, CA: Apress, 2021. ISBN 978-1-4842-6318-1. Abgerufen unter DOI: 10.1007/978-1-4842-6318-1_6.
16. WORLD WIDE WEB CONSORTIUM (W3C). *WebXR Device API* [online]. 2024 [besucht am 2024-04-02]. Abgerufen unter: <https://www.w3.org/TR/webxr/>.
17. THREE.JS. *three.js Documentation* [online]. 2024 [besucht am 2024-05-13]. Abgerufen unter: <https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>.
18. A-FRAME. *A-Frame Introduction* [online]. 2024 [besucht am 2024-05-13]. Abgerufen unter: <https://aframe.io/docs/1.5.0/introduction/>.
19. BABYLON.JS. *Babylon.js Features* [online]. 2024 [besucht am 2024-04-03]. Abgerufen unter: <https://www.babylonjs.com/specifications/>.
20. TAIKONAUTEN. *WebXR with Babylon.js: Beginner's Guide* [online]. 2024 [besucht am 2024-03-15]. Abgerufen unter: <https://medium.com/taikonauten-magazine-english/welcome-to-the-exciting-world-of-webxr-and-babylon-js-e2dbd406dbcb>.
21. SHARIF, Abdul Azim. *Porsche 911 GT3* [<https://sketchfab.com/3d-models/porsche-911-gt3-very-high-quality-free-4eee314e142f4737872e08b0df0ff7f4>]. 2024. Auch verfügbar unter: <https://sketchfab.com/3d-models/porsche-911-gt3-very-high-quality-free-4eee314e142f4737872e08b0df0ff7f4> Published under the CC BY 4.0 license.