

Rheinische  
Friedrich-Wilhelms-Universität Bonn

BACHELORARBEIT

**3D-Segmentierung in  
Mixed-Reality-Umgebungen für Microsoft  
HoloLens**

*Autor:*

Peer SCHÜTT

*Erstprüfer:*

Prof. Dr. Sven BEHNKE

*Zweitprüfer:*

Dr. Michael WEINMANN

*Betreuer:*

Max SCHWARZ

Datum: 08.04.2019



# **Eidesstattliche Erklärung**

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen verwendet habe. Die Stellen der Arbeit sowie evtl. beigelegte Abbildungen, Zeichnungen oder Grafiken, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, habe ich unter Angabe der Quelle kenntlich gemacht.

---

Ort, Datum

---

Unterschrift



# Zusammenfassung

Die Microsoft HoloLens ist eine Augmented-Reality-Brille. Sie ist als unabhängiges Head-Mounted Display konzipiert und erlaubt dem Nutzer durch eine Vielzahl von Sensoren, wie bspw. einer RGB-Kamera und einer Tiefenkamera, mit der Umgebung zu interagieren und diese virtuell zu verändern.

Diese Bewegungsfreiheit wird in dieser Arbeit genutzt, um die Umgebung durch Semantik zu erweitern und darüber Interaktionsmöglichkeiten zu schaffen. Dazu soll das von der HoloLens aufgenommene Mesh mithilfe von Semantic Segmentation in 3D annotiert werden, um Anhaltspunkte für Interaktionen zu erkennen. Wir greifen auf ein aktuelles Deep-Learning-basiertes 2D-Segmentierungsverfahren zurück und fusionieren die Ergebnisse auf dem aufgenommenen Umgebungsmesh. Die Bewegung des Nutzers ergibt so mit der Zeit eine Segmentierung der gesamten Umgebung.

Wir evaluieren unseren Ansatz in einem Testszenario mit Bürostühlen und Lampen. Es werden qualitative Ergebnisse und Laufzeitanalysen präsentiert und diskutiert.



# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>1</b>
1.1 Einleitung . . . . .	1
1.2 Zielsetzung . . . . .	2
<b>2 Related Work</b>	<b>5</b>
<b>3 Grundlagen</b>	<b>9</b>
3.1 Microsoft HoloLens . . . . .	9
3.1.1 Hardware . . . . .	9
3.1.2 Nutzbare Daten . . . . .	10
3.1.3 Unity . . . . .	11
3.1.4 Interaktion . . . . .	13
3.1.5 Spatial Mapping . . . . .	14
3.2 RefineNet . . . . .	16
3.2.1 Multi-Path Refinement . . . . .	16
3.2.2 RefineNet Block . . . . .	18
3.3 Wavefront OBJ-Dateiformat . . . . .	19
<b>4 Systemarchitektur</b>	<b>21</b>
4.1 Aufbau . . . . .	21
4.2 Ablauf . . . . .	22
4.3 Funktionsweise der HoloLens-App . . . . .	24
4.4 Projektion . . . . .	29
4.4.1 Funktionsweise . . . . .	29
4.4.2 Splatting/Rendering . . . . .	32
4.5 Semantic Fusion . . . . .	33
4.6 Netzwerkarchitektur . . . . .	34
<b>5 Auswertung</b>	<b>37</b>
5.1 Hardware . . . . .	37
5.2 RefineNet-Training . . . . .	37
5.3 Laufzeitanalysen . . . . .	39
5.4 Qualitative Ergebnisse . . . . .	41

*Inhaltsverzeichnis*

<b>6 Diskussion</b>	<b>51</b>
6.1 Ausblick . . . . .	52

# 1 Einführung

## 1.1 Einleitung

Head-Mounted Displays (HMD) wie die HTC Vive, die Oculus Rift, Immersive Headsets oder die Microsoft HoloLens sind ein Technologiezweig, dessen Anwendungsbereiche und Nutzungszahlen seit Jahren stetig steigen. Durch Fortschritte in der Technik sind solchen Virtual-Reality(VR)- und Augmented-Reality(AR)-Geräten immer neue Einsatzmöglichkeiten geschaffen worden und die Verbreitung in der Wissenschaft und Industrie als Hilfsmittel sowie Forschungsobjekt nimmt zu. Aus diesem Grund soll sich in dieser Arbeit genauer mit der Microsoft HoloLens auseinandersetzen werden, um die Umsetzbarkeit ausgewählter Konzepte zu überprüfen und Aufschluss über die Nutzbarkeit in zukünftigen Forschungsprojekten zu geben.

Während immer bessere Kameras, Lokalisierungsverfahren und Prozessoren verbaut werden, um die Nutzerbewegungen und die Umgebung zu detektieren, wodurch diese als Eingabe für Anwendungen genutzt werden können, wurde der Bereich der Semantik bisher nicht betrachtet. Semantik wird jedoch als Grundlage benötigt, um Interaktionen mit der Umgebung des Nutzers zu erlauben. Dem Fehlen dieser Funktionalität soll sich in dieser Arbeit gewidmet werden, um Möglichkeiten zu erforschen, diesen wichtigen Faktor in die HoloLens zu integrieren. State-of-the-art-Anwendungen benötigen hierfür die Platzierung durch den Nutzer oder feste räumliche Marker, an denen die Hologramme angezeigt werden. Eine konzeptuelle Erweiterung der Umgebung durch Semantik ist in Abb. 1.1 zu sehen. Sie stammt aus einem Werbevideo von Microsoft zum Release der HoloLens 2015.

Die Nutzung von Segmentierungsverfahren ist eine Möglichkeit, um Anwendungen die dynamisch anpassbare Semantik zur Verfügung zu stellen.

Das Ziel von Semantic Segmentation ist es, jedem Pixel des Bildes die passende Klasse zuzuordnen. Da jedem Pixel eine Klasse zugeordnet wird, spricht man von einer dichten Vorhersage, die als substantieller Pre-Processing-Schritt für weitergehende Anwendungen wie Scene Parsing oder Scene Understanding wichtig ist. Diese Technik wird bereits vielfach angewendet, bspw. in der Robotik, bei Autonomen Fahrzeugen und diversen Vision-Anwendungen. Die Einsatzgebiete werden



Abbildung 1.1: Idee für eine semantische Erweiterung der Umgebung: Das Motorrad wird erkannt und über dieses wird das korrespondierende 3D-Modell gelegt (Microsoft 2019b).

jedoch immer vielfältiger, da das Erkennen und Integrieren von Semantik immer essentieller wird und neue Anwendungsgebiete erschließt.

In Verbindung mit Fusionsverfahren auf Meshes und geeigneten Darstellungs-ideen in der HoloLens wäre es möglich, dem Nutzer die Interaktion mit seinem nahen Umfeld zu erlauben. Wie diese Interaktion am Ende aussieht, ist individuell anpassbar.

## 1.2 Zielsetzung

In dieser Arbeit sollen RGB-Bilder semantisch segmentiert werden, um diese anschließend mithilfe einer Fusion auf eine 3D-Szene zu erweitern. Auf Basis dieser Segmentierung in 3D soll in Verbindung mit den Augmented-Reality-Fähigkeiten der HoloLens die Interaktion mit erkannten Objekten ermöglicht werden. Diese Methode entspricht dem momentanen Forschungsstand, da auch dort die Segmentierung von 3D-Szenen meist über den Rückschritt der Segmentierung in 2D, gefolgt von einer Rückprojektion mit einer Fusion in 3D, erfolgt.

Als Ziele wurden daher formuliert:

1. Segmentierung von RGB-Bildern
2. Fusionierung der Segmentierung in eine 3D-Szene

## *1.2 Zielsetzung*

3. Darstellung der Segmentierungsergebnisse als Hologramme in der Microsoft HoloLens
4. Integration von Interaktionsmöglichkeiten

Im Rahmen dieser Arbeit wurde sich auf die Segmentierung von Lampen und Stühlen beschränkt. Eine Interaktionsmöglichkeit soll mit einer Lampe hergestellt werden und die Steuerung dieser soll über einen Server laufen. Die Anwendung ließe sich zukünftig auf Smart-Home-Anwendungen erweitern, indem die bereits vorhandene Verknüpfung untereinander genutzt wird.

In dieser Arbeit wird von einer statischen Umgebung ausgegangen, um die Komplexität des Problems zu reduzieren. Eine Erweiterung auf dynamische Änderungen in der Welt könnte ein Ziel für zukünftige Forschung darstellen.

Nach bestem Wissen handelt es sich bei der in dieser Arbeit umgesetzten Kombination aus verschiedenen Methoden um eine neuartige Idee, die bisher noch nicht behandelt wurde.



## 2 Related Work

Mixed Reality für HMD ist ein Technologiezweig, der erst in den letzten Jahren an Relevanz gewonnen hat, weil der technische Fortschritt es nun erlaubt, Geräte herzustellen, die die Hardwarevorraussetzungen erfüllen und gleichzeitig handlich sind. Es wird vermutet, dass Microsoft in den zwei Jahren seit ihrem Release ca. 50000 HoloLens verkauft hat (Strange 2018). Dies schränkt die Basis möglicher Veröffentlichungen ein. Außerdem sind die Anwendungsbereiche sehr vielfältig; momentan findet sie vor allem in der Medizin Anwendung.

In dieser Disziplin wird die HoloLens als visuelle Erweiterung für Ärzte genutzt, um bspw. die Position eines Tumors darzustellen (Poothicottu Jacob 2018), CT-Scans über das zu operierende Körperteil als Hilfe zu projizieren (Pratt et al. 2018, Abb. 2.1) oder um komplexe Verletzungen wie eine Myokardnarbe räumlich besser betrachten zu können (Jang et al. 2018). Die Platzierung von Hologrammen im Raum wird in den meisten Anwendungen per Hand erledigt und benötigt einige Zeit. Eine Erweiterung um eine automatische Platzierung wäre nützlich und die HoloLens bietet mit ihren Spatial-Mapping-Fähigkeiten auch die Grundlage dafür.

In der Arbeit von Evans et al. 2017 wurde zur Evaluierung der HoloLens eine Mixed-Reality-App gestaltet, die den Benutzer durch eine Montageanleitung führen soll. Wie bei einer Anleitung aus Papier sollte es die Möglichkeit geben, durch die einzelnen Schritte durchzublättern. Bei jedem Schritt wird das benötigte Bauteil angegeben und wenn sich das Bauteil im Sichtfeld befindet, wird es auch optisch markiert. Zur Erkennung der verschiedenen Bauelemente haben Evans et al. 2017 zuerst das Spatial Mapping Mesh der HoloLens ausprobiert. Jedoch liefert es nicht die Auflösung um kleine Objekte exakt als Mesh darzustellen und somit der Anwendung zu ermöglichen, diese zu erkennen. Sie entschieden sich deshalb für ein Marker Based Tracking.

Die von uns vorgestellte Methode umgeht dieses Problem, da das Mesh durch das Update im April 2018 bei gleichem Rechenaufwand genauer und feiner ist und wir zusätzlich Semantic Segmentation zur Erkennung nutzen. Durch die Verbindung beider ist es möglich, die Schwächen des Meshes auszugleichen.

Durch die technischen Einschränkungen der HoloLens sind hochaufgelöste Tie-

## 2 Related Work

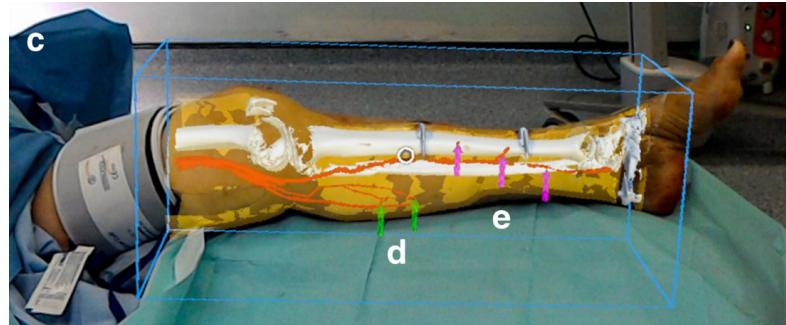


Abbildung 2.1: Ein Bild der Anwendung von Pratt et al. 2018 aus einer Operation zur Rekonstruktion von Extremitäten mithilfe der Microsoft HoloLens.

fenbilder nicht in Real-Time möglich. Um dieses Problem zu lösen, haben Garon et al. 2016 eine IntelRealSense auf der HoloLens montiert und die Tiefenbilder durch einen Stick-PC auf die HoloLens streamen lassen. Durch eine passende Kalibrierung ist es möglich, die Daten beider Geräte und die dadurch entstehenden detaillierten Tiefenbilder zusammen einzusetzen. Garon et al. 2016 nutzen dies, um Small Object Detection zu betreiben und über die HoloLens anzuzeigen.

Für die hier präsentierte Arbeit ist diese Verbesserung jedoch nicht nötig, da die zu detektierenden Objekte wesentlich größere Ausmaße besitzen und eine schlechtere Auflösung durch die zusätzliche Nutzung von semantischer Segmentierung ausgeglichen werden kann.

Orsini et al. 2017 haben in ihrer Arbeit eine Anwendung vorgestellt, um mit der HoloLens ein besseres Kocherlebnis zu erzeugen. Beim Start der Anwendung wird die Umgebung des Benutzers mithilfe von neuronalen Netzen nach Lebensmitteln abgesucht. Basierend auf den Detektionen werden Rezeptideen vorgeschlagen. Sobald der Nutzer ein Rezept ausgewählt hat, werden ihm Anweisungen erteilt, wie das Gericht zu kochen ist. Diese Anweisungen werden durch animierte Hologramme unterstützt. Auf diese Art soll dem Nutzer eine möglichst natürliche Hilfestellung geleistet werden. Die Steuerung erfolgt über Voice Commands.

Nützlich ist die Erkenntnis, dass ein Zusammenwirken zwischen einem Server zur Klassifizierung und der HoloLens möglich ist. Zudem entspricht die Idee der Erweiterung der Möglichkeiten des Benutzers durch Augmented-Reality auch in der Arbeit von Orsini et al. 2017 einem Hauptziel. Die Verbindung von Mesh und Semantik wie sie von uns umgesetzt wird, fand in der Arbeit von Orsini et al. 2017 keine Verwendung.

Chen et al. 2018 haben ein Framework vorgestellt, mit dem sie der realen Umgebung semantische Eigenschaften zuweisen. Um die Möglichkeiten von Mixed-

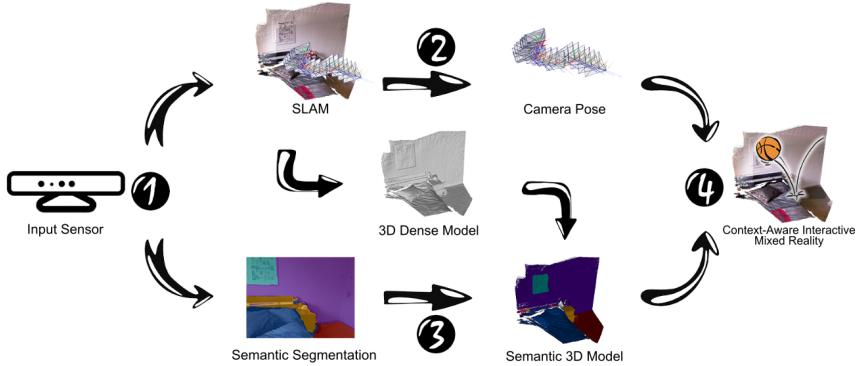


Abbildung 2.2: Der Pre-Processing Workflow der Mixed-Reality-Anwendung von Chen et al. 2018.

Reality in Verbindung mit diesen Eigenschaften zu zeigen, haben sie eine First-Person-Shooter-App programmiert, die materialabhängig unterschiedliche Einschusslöcher zeigt. Um die Semantik der Umgebung zu detektieren, nutzen sie Dense Scene Reconstruction und Deep Image Understanding. Segmentierung der RGB-Bilder erfolgt durch ein neuronales Netz basierend auf CRF-RNN (Conditional Random Fields as Recurrent Neural Network). Zur Fusionierung der semantischen Labels mit dem Mesh wird eine Bayesian Fusion, gefolgt von einem Refinement mithilfe von CRFs eingesetzt. Die Aufnahmen der RGB-Bilder und der Tiefenbilder erfolgt vorher mit einem Kinect V2 Sensor, weil dieser höher aufgelöste Tiefenbilder als die HoloLens liefert. Alle benötigten Daten werden vor Start der Anwendung preprocessed, um später nur noch auf die Informationen zugreifen zu müssen.

Unsere Vorgehensweise weist viele Übereinstimmungen mit der Arbeit von Chen et al. 2018 auf. Auch hier soll der realen Umgebung semantische Eigenschaften zugewiesen werden und dadurch spezielle Interaktionen ermöglicht werden. Jedoch nutzen wir RefineNet zur Segmentierung der 2D-Bildern. Der Vorteil unserer Methode ist, dass sie online funktioniert und nicht auf einen externen Sensor zum vorherigen Scannnen und Pre-Processing angewiesen ist. Dadurch sind wesentlich flexiblere Einsatzmöglichkeiten vorhanden.



# 3 Grundlagen

## 3.1 Microsoft HoloLens

Die Microsoft HoloLens ist eine Augmented-Reality-Brille, die von der Microsoft Corporation hergestellt wird. Sie ist als Head-Mounted Display mit eigenständiger Recheneinheit konzipiert, was eine Verwendung ohne stationäre Bindung ermöglicht. Im Gegensatz zu vergleichbaren AR-Brillen besitzt die Microsoft HoloLens eine eingebaute Tiefenkamera, was eine realistischere Interaktion der Apps mit der Nutzerumgebung erlaubt. Daher bezeichnet Microsoft sie als Mixed-Reality(MR)-Brille.

Sie ist seit dem 30. März 2016 als Development Edition für Forschungszwecke und als kommerzielle Variante erhältlich. Das Nachfolgemodell HoloLens 2 wurde am 24.02.2019 während dem Mobile World Congress in Barcelona vorgestellt.



Abbildung 3.1: Die Microsoft HoloLens (Microsoft 2018d)

### 3.1.1 Hardware

Betrieben wird die Microsoft HoloLens von vier Intel Atom x5-Z8100-Prozessoren mit 1,04 GHz, einer von Microsoft entwickelten Holographic Processing Unit (HPU) und 2 GB RAM. Als Betriebssystem ist Windows 10 vorinstalliert. Insgesamt sind 64 GB Speicherplatz vorhanden, pro App aber maximal 900MB. Der eingebaute

### 3 Grundlagen



Abbildung 3.2: Die holographischen Linsen (links) und die Sensoren (rechts) der HoloLens (Microsoft 2018d)

Sensor	Anzahl
Inertial Measurement Unit (IMU)	1
Environment Understanding Cameras	4
Tiefenkamera	1
RGB-Kamera	1
Mikrofon	4
Ambient Light Sensor	1

Tabelle 3.1: Die Sensoren der Microsoft Hololens (Microsoft 2018d)

Akku mit 16,5 Wh reicht für 2-3 Stunden aktiver Nutzung und bis zu zwei Wochen im Standby.

Die Anzeige erfolgt mithilfe von zwei durchsichtigen holographischen Linsen (Abb. 3.2), die auf Augenhöhe angebracht sind. Durch zwei HD 16:9 light engines sind für die Auflösung der Hologramme insgesamt 2,3 Millionen Lichtpunkte mit >2.500 Lichtpunkten pro Radian verfügbar. Eine Kalibrierung bezogen auf die Entfernung der Pupille erfolgt automatisch (Microsoft 2018d).

Zur Aufnahme der Umgebungsinformationen sind eine Vielzahl an Sensoren eingebaut:

Bei den „Environment Understanding Cameras“ handelt es sich um Graustufenkameras. Die Tiefenkamera ist eine IR-Time of Flight Kamera, ihre Reichweite beträgt maximal 4 m und das Sichtfeld ist  $120^\circ \times 120^\circ$  groß. Die RGB-Kamera hat bei der Aufnahme von Fotos eine maximale Auflösung von  $2048 \times 1152$  Pixel und bei Videoaufzeichnung eine Auflösung von  $1408 \times 792$  Pixel bei 30 FPS.

#### 3.1.2 Nutzbare Daten

Die Software der HoloLens und das MixedRealityToolkit (Microsoft 2017b) beinhaltet diverse APIs. In dieser Arbeit werden insbesondere die Photocapture- und die Spatial Mapping-API verwendet. Diese werden im Folgenden erklärt:

Die PhotoCapture-API stellt fünf verschiedene Auflösungen zur Aufnahme von Bildern zur Verfügung:

2048×1152, 1408×792, 1344×756, 896×504, 1280×720

Die Aufnahme der Bilder erfolgt als BGRA-Bild und die Bildinformationen stehen entweder als Byte-Array zur Verfügung oder es kann in der Galerie abgespeichert werden. Bei der Aufnahme von Fotos liefert die API außerdem die benötigten Transformationen, um die Aufnahmepose zu lokalisieren. Es werden die CameraToWorld- und die Projektionsmatrizen automatisch berechnet und ausgegeben. Die CameraToWorld-Matrix beinhaltet die extrinsischen und die Projektionsmatrix die intrinsischen Kameraparameter. Weitere Informationen hierzu sind in Kapitel 4.4 zu finden.

Die API für das Spatial Mapping erlaubt es, die Auflösung und die Zeit zwischen Updates zu verändern. Es gibt die Möglichkeit das Mesh als .obj-Datei abzuspeichern und mehrere Methoden, um die Liste aller Submeshes der Umgebung abzufragen und diese dann weiter zu verwenden. Genauere Informationen über die API sind in Abschnitt 3.1.5 vorhanden.

Seit dem im Mai 2018 veröffentlichtem Update (Microsoft 2018e) ist es möglich im sog. „Research Mode“ der HoloLens die rohen Daten der vier „Environment Understanding Cameras“, zwei Versionen der Daten der Tiefenkamera (ShortThrowToFDepth, LongThrowToFDepth) und zwei Versionen der Infrarot-Reflektivität (ShortThrowToFReflectivity, LongThrowToFReflectivity) Daten zu verwenden. In dieser Arbeit wurde sich jedoch für das bereits vorverarbeitete Spatial Mapping Mesh als Datenquelle entschieden, da die Annahme getätigt wurde, dass dieses Mesh bereits für die Verwendung im Zusammenhang mit der HoloLens optimiert wurde und für eine permanente Übertragung der rohen Tiefeninformationen nicht genug Bandbreite bei einer drahtlosen Verbindung zur Verfügung gestellt werden kann.

### 3.1.3 Unity

Die Entwicklungsumgebung für die Universal Windows Platform (UWP) ist Unity und die Programmiersprache C#. Ein Unity-Projekt besteht aus mindestens einer Szene, die von mehreren GameObjects bevölkert wird. Ein GameObject funktioniert als Basisklasse für alle Entities in einer Szene und Verbindungsstück mit den anderen GameObjects. GameObjects dienen als Container für ihre Components, die die Funktionalitäten implementieren. Components können bereits von Unity

### 3 Grundlagen

integrierte Funktionalitäten sein (bspw. Transform, Mesh Collider, Mesh Renderer, Mesh Filter) oder selbst geschriebene Skripte umfassen (Unity 2017).

Viele bereits implementierte Komponenten bietet das kostenlose MixedRealityToolkit von Microsoft (Microsoft 2017b). Nach eigener Aussage stellt es „eine Reihe von grundlegenden Komponenten und Funktionen, um die Entwicklung von MR-Apps in Unity zu beschleunigen“ zur Verfügung. Dieses Toolkit bietet mehrere Funktionalitäten wie den Spatial Mapping Manager (vgl. 3.1.5), der somit als bereits fertig programmiertes GameObject verwendet werden kann. Weitere Komponenten, die in dieser Arbeit verwendet wurden, sind Skripte zur Kommunikation mit einem Server und zur Verwaltung von Gesten sowie Shader zur Anzeige in der HoloLens.

GameObjects können zusätzlich über Skripte zur Laufzeit initialisiert werden. Eine Methode dafür ist die Initialisierung mithilfe eines Prefabs. Ein Prefab ist ein abgespeichertes GameObject, dessen gesamte Components, Variablenwerte und child GameObjects erhalten bleiben. Dieses Prefab muss einem Skript als Variable übergeben werden und kann über die Methode `Instantiate(Object original, Vector3 position, Quaternion rotation)` initialisiert werden (Unity 2018b).

Eine Anwendung für die UWP läuft auf per-Frame Basis ab. Da die durchschnittliche Framerate 60 FPS ist, müssen Rechenoperationen, die länger als  $\frac{1}{60}$ s benötigen, asynchron ausgeführt werden, um nicht die Anwendung einzufrieren. Asynchrone Methoden werden parallel durchgeführt und behindern dadurch nicht die Ausführung anderer Methoden, benötigen jedoch Handler, um die Rückgabe abzufangen.

Um selbstständig Funktionen auszuführen, definieren die meisten Unity-Skripte folgende Basis-Funktionen:

1. **Awake:** Bei der Initialisierung eines GameObjects werden die `Awake()`-Funktionen jedes angefügten Skripts einmal ausgeführt.
2. **Start:** Bei der Initialisierung eines GameObjects werden die `Start()`-Funktionen jedes angefügten Skripts einmal ausgeführt. Insbesondere werden keine `Update()`-Funktionen aufgerufen, bis alle `Start()`-Funktionen abgeschlossen sind. Im Gegensatz zur `Awake()`-Funktion wird `Start()` nur aufgerufen, wenn das GameObject auch enabled ist.
3. **Update:** Die `Update()`-Funktion ist hauptverantwortlich für die Funktionalitäten der Skripte. Sie wird einmal pro Frame aufgerufen.
4. **OnDestroy:** `OnDestroy()` wird nach dem letzten Frame-Update aufgerufen, wenn das GameObject gelöscht wird.

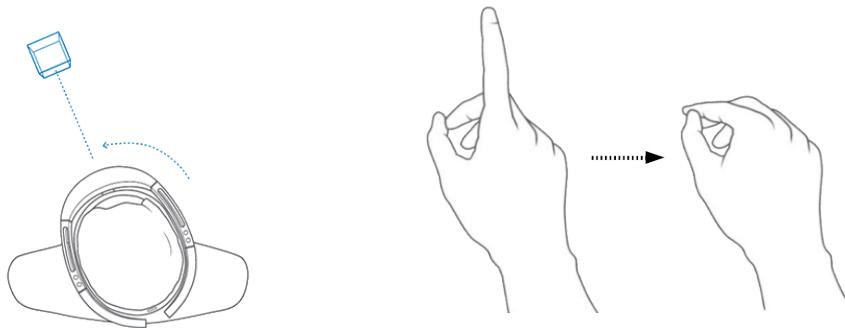


Abbildung 3.3: Gaze Tracking (links) und Air Tap (rechts) als Interaktionsmöglichkeiten (Microsoft 2018a,b)

Eine ausführliche Erklärung der Funktionsweise der Applikation, die mit Unity für die HoloLens programmiert wurde, findet sich in Kapitel 4.3.

### 3.1.4 Interaktion

Um die Kommunikation mit dem Nutzer zu ermöglichen sind vier Key-Features eingebaut: Spatial Sound, Gaze Tracking, Gesture Input und Voice Support. Letztere sind hierbei als Eingabemöglichkeiten des Nutzers und Spatial Sound zur Erweiterung des Feedbacks vorgesehen.

Gaze Tracking ist die primäre Art, um in der Mixed Reality Umgebung der HoloLens Ziele anzuvisieren. Die HoloLens erkennt dabei, in welche Richtung der Kopf zeigt (Abb. 3.3) und führt in diese Richtung einen Raycast aus. Am ersten Hindernis, sei es ein Hologramm oder das Spatial Mapping Mesh, wird dann ein Cursor angezeigt, um dem Nutzer zu signalisieren, was momentan fokussiert wird. Dadurch ergeben sich verschiedene Interaktionsmöglichkeiten wie die Platzierung von Hologrammen, die Veränderung dieser oder auch das Lenken des Blickfeldes des Nutzers. Es wird von Microsoft empfohlen, Gaze zur Auswahl zu verwenden und nicht zu versuchen, die Hände zur Auswahl zu verwenden, weil der damit einhergehende Rechenaufwand zu groß sei (Microsoft 2018a).

Um Aktionen auszuführen, benutzt die HoloLens ein "gaze-commit"-Verfahren. Durch Gaze Tracking wird dabei das Interaktionsziel ausgewählt und mithilfe von vordefinierten Gesten wird die Art der Interaktion mitgeteilt. Gesten können von jeweils einer Hand oder beiden Händen zusammen ausgeführt werden. Die beiden primären Gesten sind Air Tap und Bloom. Air Tap (Abb. 3.3) ist vergleichbar mit einem Mausklick. Bloom ist wie der "Home"-Button auf Smartphones konzipiert und lässt sich somit in der App nicht für andere Funktionen verwenden. Die Air-Tap-Geste lässt sich zu drei weiteren Gesten zusammensetzen: tap and hold, Manipulation und Navigation.

### 3 Grundlagen

Tap and hold ist die Air-Tap-Geste, nur mit der Änderung, dass der Zeigefinger nicht wieder angehoben wird, sondern in der Tap-Geste bleibt. Dadurch lassen sich komplexere "drag-and-drop"-Gesten durchführen. Die Manipulation Gestures erlauben die Bewegung, die Größenänderung und die Rotation von Hologrammen. Dazu reagiert das Hologramm nach der tap-and-hold-Geste 1:1 auf die Handbewegungen der Nutzer. Nach der tap-and-hold-Geste erscheint ein Würfel, in dem man das Hologramm durch Handbewegungen in die drei möglichen Richtungen (X,Y,Z) verschiebt.

Die Hände müssen bei dieser Nutzung im Gesture Frame bleiben. Dies ist ein Bereich, der sich ca. von der Nase zur Hüfte und zwischen den Schultern befindet. Nur dort kann die Tiefenkamera die Gesten der Hand erkennen (Microsoft 2018c).

Als Alternative oder Erweiterung zu Gesten ist der Voice Support vorhanden. Zuerst muss das zu bearbeitende Hologramm fokussiert werden, um dann den für das Wort vorgesehene Befehl auszuführen. Dies ermöglicht es, einfacher komplizierte Befehle auszuführen, die mit Gesten aufwendig zu aktivieren wären. Realisiert wird die Spracherkennung durch die KeywordRecognizer-API, der man die benötigten Schlüsselwörter übergibt (Microsoft 2018h).

Die Aufgabe des Spatial Sound liegt darin, dem Nutzer Feedback von Hologrammen in seiner Umgebung zu geben oder auch seine Aufmerksamkeit auf einen bestimmten Bereich zu lenken. Durch die Anordnung der vier Lautsprecher um die HoloLens herum entsteht die Illusion räumlichen Hörens. Durch eine Head Related Transfer Function kann somit die Lautstärke basierend auf Entfernung und Position vom Nutzer angepasst werden (Microsoft 2018g).

#### 3.1.5 Spatial Mapping

Benötigt eine Anwendung Spatial-Mapping-Daten der Umgebung, können Bereiche im Raum definiert werden, für die die HoloLens Tiefeninformationen zu einem Mesh zusammenfasst. Ein Mesh wird dabei in Unity über seine Vertices und Triangles definiert. Die Vertices werden als Array von 3D-Vektoren und die Triangles als Integer-Array abgespeichert. Die Einträge des Triangle-Arrays sind Indizes in die Liste der Vertices; drei Indizes für jedes Dreieck. Es können noch zusätzliche Eigenschaften wie Farbe oder Normalen definiert werden (Unity 2018a).

Die gewählten Bereiche im Raum können stationär in der Umgebung verankert sein, aber sich auch mit der HoloLens mitbewegen. Das Mesh hat standardmäßig eine Auflösung von 500 Dreiecken pro  $m^3$  und updatet alle 3.5 Sekunden. Die Auflösung und die Zeit zwischen den Updates lassen sich für jede App individuell einstellen, maximal auf 2000 Dreiecke pro  $m^3$  und auf minimal eine Sekunde.

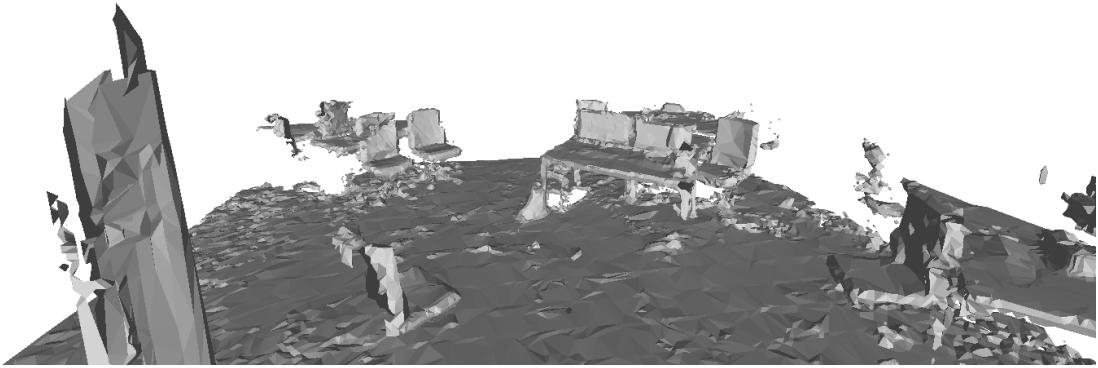


Abbildung 3.4: Ausschnitt des Spatial Mapping Meshes der Robotikhalle des Institut für Informatik der Universität Bonn. Große Objekte, wie Stühle, Tische oder Bildschirme sind gut zu erkennen.

Für das Mesh werden nur Punkte verwendet, die einen Abstand von mindestens 0,85 m zur HoloLens haben, um eine Verfälschung des Mesh durch das Einbeziehen der Hände des Nutzers zu verhindern. Der in dieser Anwendung standardmäßig verwendete Scan-Bereich ist  $4 \times 4 \times 4 \text{ m}^3$  groß. Aufgrund der Hardwaregrenzen der HoloLens wird das Spatial Mapping Mesh für große Objekte verwendet und scannt kleine Objekte gar nicht oder nur grob.

Der Spatial Mapping Manager nutzt den Spatial Surface Observer zur Erstellung des Meshes. Diesem werden anfangs die Bereiche übergeben, in denen die Anwendung Spatial-Mapping-Daten benötigt. Der Koordinatenursprung des Meshes wird in die Pose der HoloLens beim Start der App gelegt. Die Anwendung kann entweder selbstständig Änderungen zu jeder Zeit vom Observer anfordern, oder auf ein `surface changed`-Event warten. Abgeänderte Surfaces sind dabei in die drei Klassen `added`, `changed` und `removed` eingeteilt. Für jedes lässt sich eine geeignete Strategie zum Umgang festlegen. Das Spatial Mapping Mesh als Ganzes ist in viele kleine SurfaceObjects eingeteilt. Sollten Änderungen eintreten, wird das betroffene SurfaceObject durch die oben beschriebenen Funktionen ausgewählt und abgeändert oder gelöscht (Microsoft 2018f).

Die Anwendung kommt ohne weitere Lokalisationsverfahren wie bspw. SLAM aus, da die HoloLens über ein integriertes Inside-Out-Tracking verfügt. Dieses nutzt die vier „Environment Understanding Cameras“ und die IMU, um markante Punkte der Umgebung zu beobachten und die Veränderung dieser mit den Daten der IMU zu fusionieren. Auf diese Weise wird eine Lokalisierung ermöglicht. Die Ergebnisse dieses Verfahrens weisen Schwächen bei zu wenig Licht und zu starker Bewegung auf. Jedoch handelt es sich um Extrema, die bei der normalen Nutzung durch einen Nutzer nicht auftreteten sollten (Microsoft 2017a).

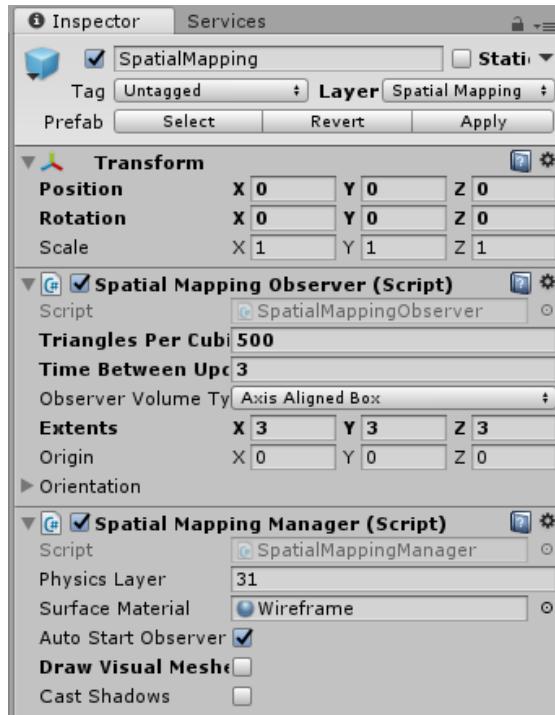


Abbildung 3.5: Drei Komponenten des Spatial Mapping - GameObjects mit den modifizierbaren Parametern.

## 3.2 RefineNet

Für diese Arbeit wird eine Semantic Segmentation von RGB-Bildern benötigt. Als neuronales Netz wird RefineNet (Lin et al. 2017) verwendet. Es handelt sich dabei um ein Multi-Path Refinement Network, dessen Struktur in Abbildung 3.6 dargestellt ist. Als Eingabe dient ein RGB-Bild beliebiger Größe und die Ausgabe ist eine dense score map, die für jeden Pixel die Klassenwahrscheinlichkeiten enthält.

Die Idee der RefineNet-Pipeline ist es, grobe semantische High Level Features mit feinkörnigen Low Level Features, die aus den Zwischenergebnissen der CNNs extrahiert werden, zu fusionieren und auf diese Weise hochauflösende semantische Feature Maps zu erzeugen. Das Verfahren hat auf dem Cityscapes Datensatz State-of-the-Art-Ergebnisse geliefert und wurde bereits mehrfach erfolgreich eingesetzt (Periyasamy, Schwarz und Behnke 2018; Schwarz et al. 2018).

### 3.2.1 Multi-Path Refinement

ResNet (He et al. 2016) wurde in der Arbeit von Lin et al. 2017 ausgewählt und die Ergebnisse der vier ResNet-Blöcke als Eingabe der RefineNet-Blöcke genutzt (vgl. Abb.3.6). ResNet kann allerdings auch durch ein anderes Netzwerk ersetzt werden,

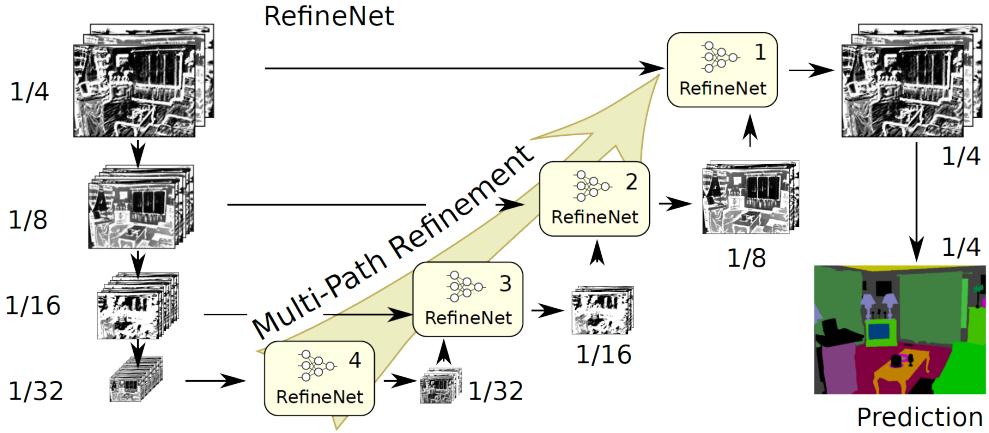


Abbildung 3.6: Eine Beispiel-Architektur der RefineNet-Pipeline. Die vier Zwischenergebnisse von ResNet (linke Seite) werden als Eingaben für die RefineNet-Blöcke genutzt. Grobe semantische High-Level-Features werden durch die Fusionierung zu höher aufgelösten Feature Maps. Die engültige Prediction erfolgt mit einem Soft-max Layer (Lin et al. 2017)

wie bspw. ResNeXt-101 (Xie et al. 2017), das von Schwarz et al. 2018 genutzt wird.

Jeder ResNet-Block wird mit einem RefineNet-Block verbunden (vgl. Abb. 3.6). In jeder Stufe werden die Ergebnisse von ResNet-Block-x und RefineNet-Block-(x+1) als Eingaben für RefineNet-Block-x genutzt. Dies wird stufenweise fortgesetzt, bis ResNet-Block-1 erreicht ist. Die hochauflösten Feature Maps von ResNet-Block-x sollen genutzt werden um die grob aufgelösten aber semantisch High-Level-Features von RefineNet-Block-(x+1) zu verfeinern und auf die gleiche Auflösung, wie ResNet-Block-x, zu bringen. Im letzten Schritt werden die Feature Maps von einem soft-max layer angepasst, um die finale Vorhersage in Form einer dense score map zu generieren. Es findet noch ein Up-Sampling statt, damit die score map die gleiche Auflösung wie das Originalbild besitzt. Wichtig ist, dass die Parameter der Blöcke nicht miteinander verbunden sind, sondern jeder Block seine eigenen Parameter lernt.

Zwischen den RefineNet- und ResNet-Blöcken, sowie innerhalb der RefineNet-Blöcke, werden long range residual connections eingeführt. Durch diese Verbindungen ist es möglich eine direkte Backpropagation zu Operatoren verschiedener Stufen durchzuführen, was ein effektives und effizientes end-to-end Training ermöglicht.

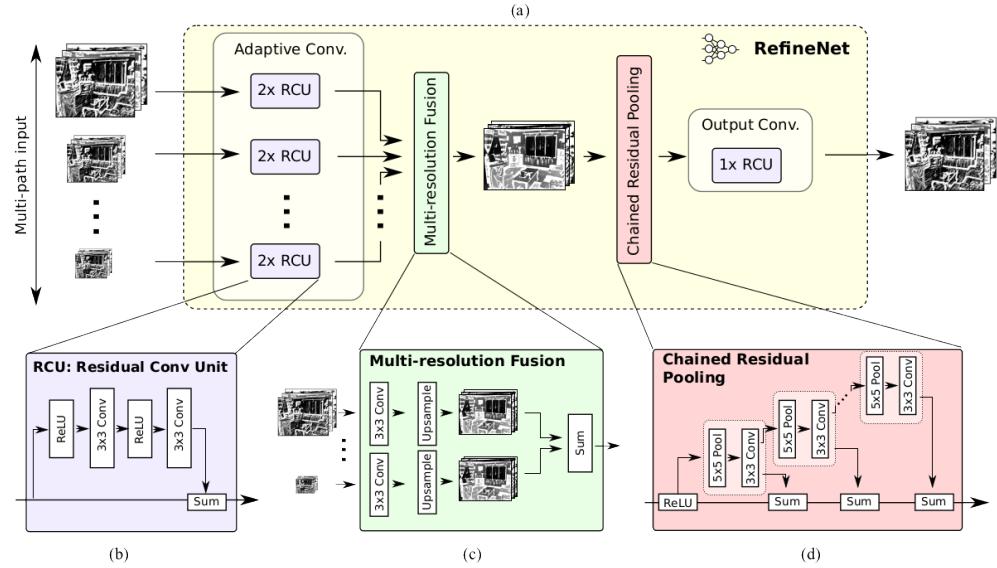


Abbildung 3.7: Die interne Struktur eines RefineNet-Blocks mit seinen vier Teilen(Lin et al. 2017).

### 3.2.2 RefineNet Block

RefineNet-Blöcke sind in vier Teile aufgeteilt: Adaptive Convolutions, Multi-resolution Fusion, Chained Residual Pooling und Output Convolutions. In der hier gezeigten Beispielarchitektur (Abb. 3.6) bekommt jeder RefineNet-Block maximal zwei Eingaben, jedoch ist die Anzahl an Eingaben variabel, wie aus Abb. 3.7(a) ersichtlich ist. Sollte es nur eine Eingabe für einen RefineNet-Block geben, dient dieser Block dem Fine-Tuning der von ResNet gelernten Gewichte.

In der Adaptive Convolution wird eine Eingabe nacheinander durch zwei Residual Convolution Units (RCUs) geschickt, um die bereits vorher gelernten Gewichte von ResNet zu verfeinern und eine Nichtlinearität einzuführen. Jede RCU besteht aus zwei rectified linear units (ReLUs) und zwei 3x3 Convolutions (vgl. Abb. 3.7(b)). Im Aufbau der RCUs ist das Konzept der long range residual connections gut ersichtlich: Die Eingabe einer RCU wird am Ende mit dem Ergebnis der RCU summiert.

Der nächste Block ist der Multi-Resolution Fusion Block. Dort werden die eingegebenen Feature Maps zuerst durch eine Konvolution auf die gleiche Feature Dimension gebracht (die kleinste Dimension der Eingaben), um anschließend auf die höchste Auflösung der Eingaben upgesampled zu werden. Die Fusion der Feature Maps erfolgt dann durch Summation.

Chained Residual Pooling nutzt mehrere  $5 \times 5$  Pooling Operationen, die jeweils von einer  $3 \times 3$  Konvolution gefolgt werden, um Kontextinformationen aus großen Bildbereichen einzufangen (Abb. 3.7(d)). Die Wiederverwendung der Ergebnisse des vorherigen Pooling-Blocks als Eingabe ermöglicht dem nachfolgenden Pooling-Block einen noch größeren Bereich in der ursprünglichen Eingabe zu betrachten, ohne den Pooling-Operator zu vergrößern. Durch die Konvolution und die Summation nach jedem Pooling wird das effiziente Lernen einer gewichteten Fusion ermöglicht. Wie oft Pooling stattfinden soll und welcher Stride gewählt wird ist variabel.

Die Output Convolutions sind der letzte Schritt eines RefineNet-Blocks. Die Ergebnisse des Chained Residual Poolings werden durch eine Residual Convolution Unit geschickt. Dadurch werden weitere Nichtlinearitäten auf die Feature Maps angewendet. Die Feature Dimension bleibt identisch.

### 3.3 Wavefront OBJ-Dateiformat

Das Wavefront OBJ-Dateiformat wird genutzt, um Polygon-Meshes in ASCII darzustellen. Es ist ein standardisiertes und weit verbreitetes Format. Es wird von Unity direkt unterstützt, weshalb wir es in dieser Arbeit verwenden. Einige Aspekte der Struktur werden im Folgenden erklärt:

**Vertex: v x y z** Vertices werden mit ihren kartesischen Koordinaten abgespeichert. Sie werden zudem intern mit einem Index versehen, der basierend auf ihrer Position in der Datei festgelegt wird (Erster Vertex hat den Index 1, zweiter Index 2 etc.). Farbinformationen werden für jeden Vertex einzeln definiert. Dazu werden die drei RGB-Werte, durch jeweils ein Leerzeichen getrennt, hinter die Koordinaten geschrieben.

**Vertex Normal: vn x y z** Sie definieren die Normalen von Vertices. Die Normalen werden genau wie die Vertices durchnummeriert. Sie finden erst Verwendung, wenn sie bei der Definition von Faces mit den Vertices, deren Normale sie darstellen sollen, in Verbindung gebracht werden.

**Face Data: f v1[//vn1] v2[//vn2] v3[//vn3] [...]** Ein Polygon wird über die Vertices, die dieses bilden, definiert. Dabei werden die Vertices über ihre Indices referenziert (Bspw. "f 1 3 4" ist ein Dreieck mit den Vertices mit internen Indizes

### 3 Grundlagen

1,3 und 4 als Eckpunkten). Die Orientierung wird bei Übergabe von Vertex Normalen durch diese definiert. Sollten keine übergeben werden, wird die Orientierung anhand der Ordnung der Vertices errechnet. Die Ordnung wird als entgegen dem Uhrzeigersinn angesehen.

**Object Name: o name** Durch Objects lassen sich mehrere Meshes in einer Datei definieren und über den Namen referenzieren.

```
o cube
v 0.00 2.00 2.00
v 0.00 0.00 2.00
v 2.00 0.00 2.00
v 2.00 2.00 2.00
v 0.00 2.00 0.00
v 0.00 0.00 0.00
v 2.00 0.00 0.00
v 2.00 2.00 0.00
f 1 2 3 4
f 8 7 6 5
f 4 3 7 8
f 5 1 4 8
f 5 6 2 1
f 2 6 7 3
```

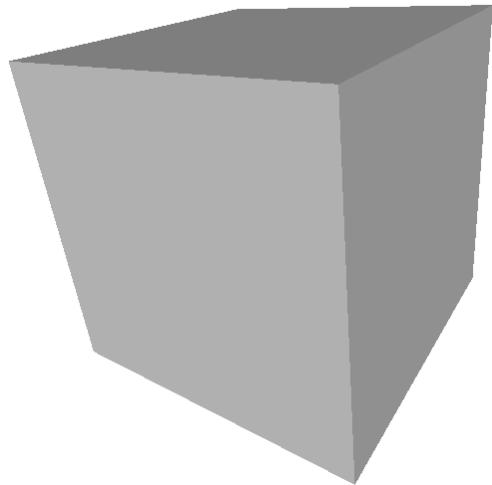


Abbildung 3.8: Die Struktur von OBJ-Dateien anhand eines Beispiels erklärt. Links die Codierung und rechts der Würfel als Ergebnis.

Die Informationen aus diesem Abschnitt wurden Bourke 2012 entnommen.

# 4 Systemarchitektur

In diesem Kapitel wird die in dieser Arbeit genutzte Architektur vorgestellt. Dabei werden zuerst der Aufbau (4.1) und der Ablauf (4.2) beschrieben. In den darauf folgenden Abschnitten werden die einzelnen Module im Detail erklärt.

## 4.1 Aufbau

Bei der hier genutzten Architektur handelt es sich um eine Client-Server-Struktur. Die Microsoft HoloLens dient als Client und ein Linux-Computer als Server (vgl. Abb. 4.1). Diese Aufteilung wird benötigt, um die Rechenleistung der HoloLens zu erhöhen, da sie für Segmentierungsoperationen nicht ausgelegt ist und die Verarbeitung des Mesh rechenintensiv ist. Zudem bietet sie nicht genug Speicherplatz für die RefineNet-Gewichte.

Die UWP-App wurde unter Unity 2017.4.1f1 programmiert. Die Serveranwendungen wurden mit Python 3.7.1 implementiert.

Verbunden sind Client und Server mithilfe einer WLAN-Verbindung. Weil die Hololens als unabhängiges HMD konzipiert wurde, wird die kabellose Verbindung mit einer geringeren Bandbreite und Stabilität bevorzugt, um die Bewegungsfreiheit des Nutzers nicht einzuschränken. Für die WLAN-Verbindung werden an beiden Endpunkten TCP-Sockets verwendet, um eine möglichst einfache Kommunikation zu ermöglichen und den Datenstrom über das Netzwerk zu optimieren (vgl. Abs. 4.6).

Die HoloLens dient der Aufnahme der benötigten Daten, der Interaktion mit dem Nutzer und der Anzeige der Semantic Annotations. Um diese Funktionalitäten zu gewährleisten, werden das Spatial Mapping Mesh, Kamerabilder und die Transformationsmatrizen an den Server übertragen. Serverseitig werden dann die Bilder durch RefineNet segmentiert, auf das Mesh projiziert und mit der bisherigen Segmentierung fusioniert. Das Ergebnis wird schließlich auf der HoloLens angezeigt, um dem Nutzer Interaktionsmöglichkeiten zu geben. Genauere Angaben zum Ablauf folgen im nächsten Abschnitt.

Die Lampe wird vom Server mithilfe einer ROS-Schnittstelle über einen Micro-

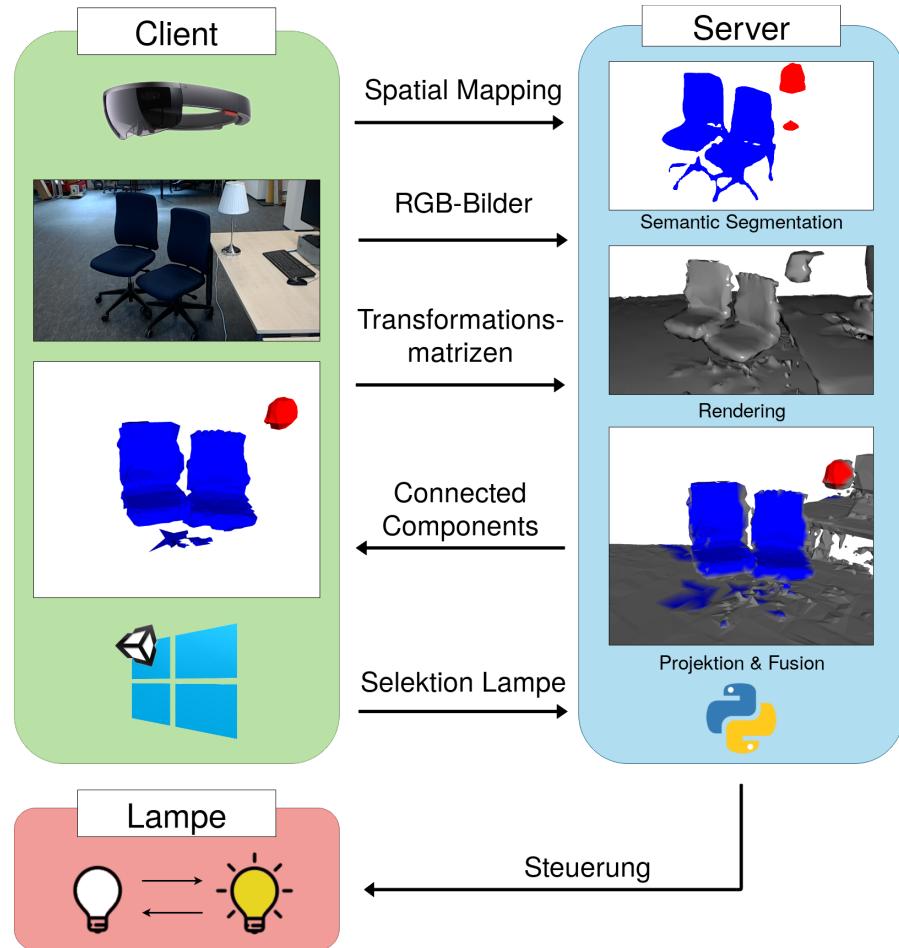


Abbildung 4.1: Die hier genutzte Architektur mit ihren zwei Hauptkomponenten.

controller gesteuert.

## 4.2 Ablauf

Um der Umgebung des Nutzers mithilfe von Segmentierungen Interaktionsmöglichkeiten zu verschaffen, wird ein stetiger Strom an Daten von und zur HoloLens benötigt. Dieser ist zyklisch gestaltet. In dieser Arbeit wird ein Ablauf, wie er in Abb. 4.2 aufgezeigt ist, vorgeschlagen:

Beim Start der App wird angefangen, die Umgebung zu scannen und zu einem Mesh zusammenzufassen. Das so erhaltene Spatial Mapping Mesh wird nach 1000 Frames zum Server gesendet. Serverseitig wird das Mesh als Obj-Datei gespeichert und durch das python-Paket Trimesh verwaltet (Dawson-Haggerty 2017). Da die Umgebung als statisch angesehen wird und das Mesh somit nur einmal übertragen

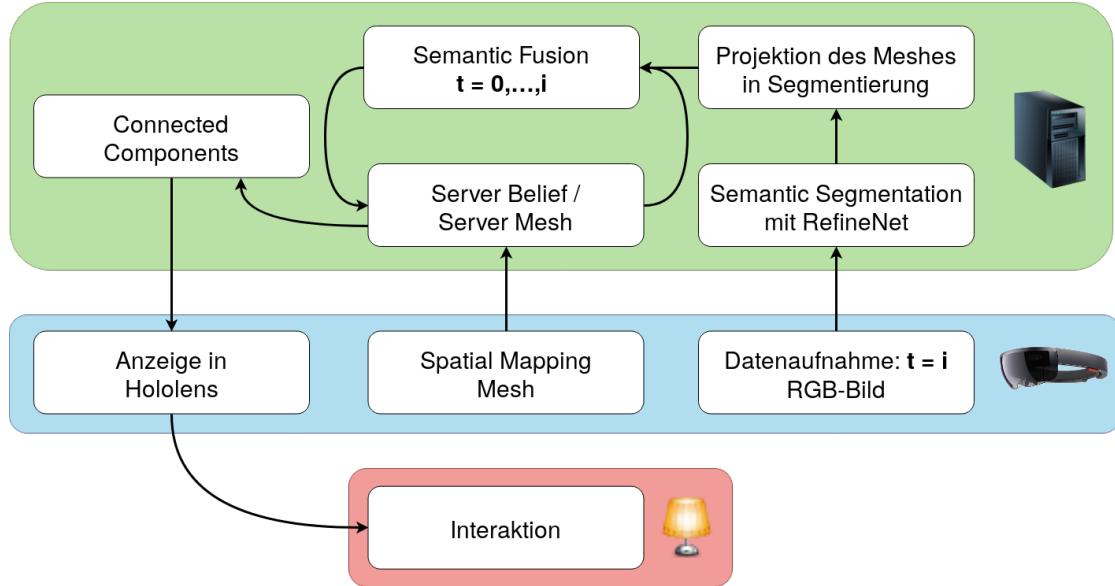


Abbildung 4.2: Ablauf der hier verwendeten Anwendung und die Kommunikation zwischen den Modulen.

wird, muss der Nutzer vor Start der App die Umgebung ausführlich scannen, um der Anwendung eine möglichst genaue Grundlage zu liefern.

Jeden Zeitschritt nach Übertragung des Meshes wird von der HoloLens ein neues Bild mit seinen Transformationsmatrizen aufgenommen. Beides wird an den Server gesendet. Das Bild wird vom Server mithilfe der vortrainierten RefineNet-Backbone segmentiert. Das Mesh wird dann auf die berechneten Klassenwahrscheinlichkeiten per Pixel projiziert (vgl. Abs. 4.4). Damit die Ergebnisse der vorherigen Segmentierungen berücksichtigt werden, findet eine Fusion mit allen vorherigen Zeitschritten statt (vgl. Abs. 4.5). Auf diese Art wird für das Mesh der Überblick über die Wahrscheinlichkeit der Klassen für jeden Vertex gehalten.

Um nun aus diesen Wahrscheinlichkeiten eine Segmentierung zu erzeugen, die auch auf die HoloLens übertragen werden kann, muss das Maximum über die Klassenwahrscheinlichkeiten der Vertices gebildet werden. Im Ergebnis wird jedem Vertex ein Label („Lamp“, „Chair“ oder „Unknown“) zugewiesen.

Vertices deren Klasse „Unknown“ ist, interessieren uns für die Darstellung der Segmentierung in der HoloLens nicht. Wenn die als „Unknown“ segmentierten Vertices herausgefiltert sind, wird der Rest des Meshes mithilfe des trimesh-Pakets in seine Connected Components aufgeteilt. Somit sollen die einzelnen Stühle und Lampen aus dem Gesamtmesh gelöst werden. Die Meshes der Connected Components werden anschließend anhand der Anzahl an Dreiecken gefiltert, um Artefakte und false positives nach Möglichkeit bei der Anzeige zu verringern und zu kleine

## 4 Systemarchitektur

und somit irrelevante Meshteile zu eliminieren. Die Anzahl an Dreiecken, ab der eine Component gesendet wird, variiert je Klasse und wird empirisch festgelegt. Standardmäßig muss ein Stuhl aus mehr als 30 und eine Lampe aus mehr als fünf Dreiecken bestehen.

Die so erhaltenen Connected Components werden nach einer gewissen Anzahl an Bildern an die HoloLens zur Darstellung gesendet. Standardmäßig nach fünf Bildern. Zusätzlich zum Mesh bekommt jede Component auch noch die Klasse mitgesendet, zu der sie gehört, um die Segmentierungsergebnisse in die HoloLens zu übertragen. In der HoloLens werden die Meshes der Components als unabhängige Hologramme (bzw. GameObjects) erstellt und angezeigt. Die Segmentierungsergebnisse werden nicht nach jedem Bild an die HoloLens gesendet, weil die Annahme getätigt wurde, dass ein einzelnes Bild noch keinen massiven Einfluss auf das Segmentierungsergebnis hat. Erst mehrere Bilder ergeben diesen.

Bei der Selektion eines der Hologramme wird die Information über die Position des Raycastendpunkts und den Namen des Hologramms an den Server gesendet, damit dieser die Informationen verarbeitet und eine mögliche Interaktion einleiten kann: Das Ein- bzw. Ausschalten einer Lampe.

### 4.3 Funktionsweise der HoloLens-App

Dieser Abschnitt beschäftigt sich mit den wichtigen Skripten der UWP-App auf der HoloLens und deren Funktionsweise:

- MeshAndPictureDeliverer
- ChangeMesh
- AsynchronousClientSocket
- GazeGestureManager
- GazeGestureBehaviour

Die zyklische Ausführung der Funktionen ist nicht durch Zeitabstände definiert, sondern wie in Unity üblich über die Anzahl an Frames seit der letzten Ausführung.

**MeshAndPictureDeliverer:** Nach 1000 Frames wird das Mesh der Umgebung an den Server übertragen, zusammen mit dem ersten Bild. Es wird 1000 Frames gewartet, weil der Spatial Mapping Manager anfangs einige Zeit benötigt, um das Mesh der Umgebung richtig zu initialisieren und die Funktionalitäten zur Verfügung zu stellen. In den Tests kam es bei einer Übertragung bei früheren Frames

#### 4.3 Funktionsweise der HoloLens-App

sehr häufig vor, dass nur ein Teil des Meshes gesendet wurde, weil manche Teile noch nicht initialisiert waren. Nach den 1000 Frames werden Bilder alle 100 Frames gemacht und an den Server gesendet.

Das Auslesen der Meshdaten zur Übertragung auf den Server läuft über eine Funktionalität des Spatial Mapping Managers. Die genutzte Funktion heißt **GetMeshFilters**. Sie iteriert über alle SurfaceObjects des Spatial Mapping Managers und gibt deren MeshFilter zurück. Von Interesse ist die `sharedMesh`-Variable des MeshFilters, da sie das Mesh referenziert. Anschließend kann mit einer for-Schleife über alle `sharedMeshes` der MeshFilter iteriert werden, um diese in einen Byte-Array zu verarbeiten, das an den Server gesendet werden soll. Als Codierung des Meshes wurde in dieser Arbeit die Codierung von Obj-Dateien verwendet (vgl. Abs. 3.3). Dazu wird das Mesh in einen String geschrieben, der mithilfe der BitConverter-Klasse in ein Byte-Array umgewandelt wird. Das Ergebnis des Serialisierens der MeshFilter ist somit ein Byte-Array, dass einen String decodiert, in dem das gesamte Mesh der Umgebung abgespeichert wird. Dieses Array kann anschließend an den Server gesendet werden.

```
public void SendMeshToServer(List<MeshFilter> MeshFilters) {
    MeshFilter mf = MeshFilters[0];
    byte[] buffer = Serialize(mf.sharedMesh, 0, mf.transform);

    for (int index = 1; index < MeshFilters.Count; index++)
    {
        mf = MeshFilters[index];
        byte[] data = Serialize(mf.sharedMesh, index, mf.transform);
        int len = buffer.Length;
        Array.Resize(ref buffer, buffer.Length + data.Length);
        data.CopyTo(buffer, len);
    }

    byte[] meshBuffer = clientSocket.GetWholeMeshAsByteData(buffer);
    clientSocket.SendData(meshBuffer);
}
```

Die Aufnahme von Fotos erfolgt durch die PhotoCapture-Klasse. Dieser wird bei Initialisierung die gewünschte Auflösung, die Undurchsichtigkeit (Opacity) der Hologramme und das Aufnahmeformat (BGR) übergeben. Zur Fotoaufnahme wird die asynchrone Methode `TakePhotoAsync(OnCapturedPhotoToMemory)` aufgerufen. Sobald das Bild aufgenommen wurde, wird die Methode `OnCapturedPhotoToMemory` aufgerufen, um die Daten zu verarbeiten:

## 4 Systemarchitektur

```
private void OnCapturedPhotoToMemory(PhotoCapture.PhotoCaptureResult
    result, PhotoCaptureFrame photoCaptureFrame) {
    List<byte> byteBuffer = new List<byte>();
    photoCaptureFrame.CopyRawImageDataIntoBuffer(byteBuffer);

    Matrix4x4 CamToWorldMatrix;
    Matrix4x4 ProjectionMatrix;
    photoCaptureFrame.TryGetCameraToWorldMatrix(out CamToWorldMatrix);
    photoCaptureFrame.TryGetProjectionMatrix(out ProjectionMatrix);

    //codiert die Informationen als Byte-Array, das versendet wird
    byte[] imageBuffer = clientSocket.GetImageAsByteData(byteBuffer,
        pictureWidth, pictureHeight, CamToWorldMatrix, ProjectionMatrix);
    clientSocket.SendData(imageBuffer);
}
```

**AsynchronousClientSocket:** Dieses Skript beinhaltet den Socket, die Netzwerklogik und Funktionen zur Codierung von verschiedenen Datentypen als Byte-Array. Die IP-Adresse des Hosts und die Portnummer werden vor Start der App übergeben.

Beim Start der App wird versucht, die Verbindung zum Server herzustellen. Sollte die Verbindung nicht hergestellt werden, wird der Verbindungsauftbau alle zwei Sekunden erneut probiert. Ein Socket besitzt in der Universal Windows Platform (UWP) einen Input- und einen Outputstream, von denen die eintreffenden Daten abgelesen bzw. in den zu versendenden Nachrichten geschrieben werden müssen.

```
Stream outputStream = streamSocket.OutputStream.AsStreamForWrite();
Stream inputStream = streamSocket.InputStream.AsStreamForRead();
```

Die Verbindung zwischen Client und Server ist asynchron, um ein permanentes Senden und Empfangen von Daten zu ermöglichen. Daher rufen andere Skripte nicht direkt die Funktion zum Senden an den Server auf, sondern reihen ihre Daten mit der Funktion `SendData(byte[] dataBufferToSend)` in eine Liste ein.

In der Update-Methode wird bei jedem Frame geprüft, ob momentan noch gesendet wird (mithilfe eines Sending-Flags) und wenn dies nicht der Fall ist, wird das nächste Byte-Array aus der Liste entnommen und an den Server gesendet. Wichtig ist hierbei, dass die Liste nicht zu lang wird, also der Client es nicht schafft, die Daten schnell genug an den Server zu senden. Sollte es dazu kommen, wird ein immer größerer Backlog aufgebaut und die zu sendenen Daten entsprechen nicht mehr dem momentanen Zustand der Umgebung.

Eintreffende Nachrichten werden über die async Methode `WaitForData()` abgefangen und bearbeitet. Diese Methode versucht auf dem InputStream einen Byte einzulesen. Sollte dies möglich sein, entspricht dieser Byte dem Typ der eintreffenden Nachricht. Bei Type 2 wird dem ChangeMesh-Skript mitgeteilt, dass neue Connected Components gesendet werden und die alten disabled werden sollen. Bei Type 1 handelt es sich um ein neues Connected Component. Das angekommene Byte-Array wird in einen Int32 für die Klasse des Connected Component, ein Vector3-Array (für die Vertices) und ein int-Array (für die Faces) umgewandelt. Diese drei Variablen werden dann dem MeshAndPictureDeliverer übergeben, sodass dieser sie in ein Mesh umwandeln kann, um sie durch das ChangeMesh-Skript zu instanziieren. Genaue Informationen über die Codierung und die Arten von Nachrichten werden in Abschnitt 4.6 gegeben.

```
public void Update() {

    if (connectionEstablished && !Sending && dataSendQueue.Count > 0)
    {
        byte[] nextPacket = dataSendQueue[0];
        dataSendQueue.RemoveAt(0);
        SendDataOverNetwork(nextPacket);
    }

    if(!WaitingForData) {
        WaitForData();
    }
    [...]
}
```

**GazeGestureManager:** In Unity gibt es die Möglichkeit, einen Raycast aus einer bestimmten Pose durchzuführen. Dieser Raycast hat als Ergebnis verschiedene Informationen über den Endpunkt, bspw. die Weltkoordinaten des Endpunktes oder das GameObject des getroffenen Hologramms. Zusätzlich lässt sich spezifisch definieren, welche Layer durch diesen Raycast getroffen werden sollen. Hierzu ist wichtig anzumerken, dass für jedes GameObject definiert werden muss in welchem Layer dieses liegt. Der Raycast in der hier vorgestellten Anwendung wird nur auf das Layer der Hologramme ausgeführt, die Stühle bzw. Lampen darstellen. Begründet wird diese Implementierung dadurch, dass uns die Ergebnisse des Raycasts auf andere Hologramme oder das Mesh der Umgebung für die App nicht interessieren und möglicherweise die Interaktion auch bei keiner physischen Sichtbarkeit ermöglicht werden soll.

Gesten werden über einen GestureRecognizer erkannt. Mit dem GestureReco-

## 4 Systemarchitektur

gnizer ist es möglich, für alle benötigten Gesten Verhaltensweisen zu hinterlegen, die beim Erkennen dieser Geste ausgeführt werden. In diesem Fall wird eine OnSelect-Nachricht an das anvisierte Hologramm gesendet, damit dieses die Funktion OnSelect mit den übergebenen Argumenten ausführen kann.

```
void Awake() {
    recognizer = new GestureRecognizer();

    // Tap gesture wurde erkannt
    recognizer.Tapped += (args) =>
    {
        // Send an OnSelect message to the focused object.
        if (FocusedObject != null)
        {
            FocusedObject.SendMessageUpwards("OnSelect",
                raycastHitPoint, SendMessageOptions.DontRequireReceiver);
        }
    };
    recognizer.StartCapturingGestures();
}
```

Um die Selektionsmöglichkeiten besser zu visualisieren, wird der Endpunkt des Raycasts beim Treffen eines Hologramms farblich hervorgehoben, sodass der Nutzer nicht nur mit einem Cursor das visuelle Feedback bekommt. Hierbei handelt es sich um einen gelben Kreis auf dem Hologramm.

**GazeGestureBehaviour:** Wenn ein Hologramm vom Nutzer selektiert wird, bekommt das GazeGestureBehaviour-Skript einen Aufruf die OnSelect Methode auszuführen. Diese übermittelt dem Server mithilfe des Sockets den Namen des Hologramms „Lamp\_X“ bzw. „Chair\_X“ und den Endpunkt des Raycasts, der dieses Hologramm getroffen hat. Der Server soll basierend auf diesen Informationen eine Fallunterscheidung durchführen und die Lampe gegebenenfalls an- bzw. ausschalten. Die Information über die Selektion wird als wichtigste Nachricht an den Server angesehen, weshalb sie mit der SendData-Methode an den Anfang der dataSendQueue eingefügt wird.

Um dem Nutzer ein visuelles Feedback über die Selektion zu geben, wird die Farbe des selektierten Hologramms geändert. Ein „selektiertes“ Hologramm bekommt die Farbe grün. Sollte das Hologramm nochmal ausgewählt, also deselektiert, werden, wird die Farbe wieder auf ihren ursprünglichen Wert zurückgesetzt.

**ChangeMesh:** Das Ergebnis der serverseitigen Segmentierung wird als mehrere Meshes, die Connected Components entsprechen, auf die HoloLens übertragen. Um diese dem Nutzer zu visualisieren und einen Raycast zu erlauben, muss für jeden Connected Component ein neues GameObject initialisiert werden. Dies wird mithilfe eines Prefabs gemacht. Die GameObjects der Connected Components besitzen einen Mesh Filter, einen Mesh Renderer zur Anzeige und einen Mesh Collider, damit der Raycast das Mesh treffen kann. Die Farbe wird über die Material-Komponente des Mesh Renderers festgelegt und lässt sich beliebig anpassen. In dieser Arbeit wurde sich dafür entschieden, Stühle in blauer und Lampen in roter Farbe darzustellen. Diese Farben wurden gewählt, da sie in den Tests am besten in verschiedenen Lichtverhältnissen sichtbar waren und sehr gut voneinander und vom Hintergrund unterschieden werden können. Die GameObjects der Lampen bzw. Stühle bekommen zusätzlich Namen basierend auf ihrer Klasse („Lamp“, „Chair“) und einen Laufindex, um die verschiedenen Objekte einer Klasse voneinander unterschieden zu können. Die so initialisierten GameObjects werden in einer Liste referenziert.

Wenn die nächste Generation an Connected Components vom Server gesendet werden, sind die momentan angezeigten Connected Components nicht mehr aktuell und müssen ersetzt werden. Daher wird ihr Renderer ausgeschaltet. Sie werden jedoch nicht zerstört, sondern nur als inaktiv gespeichert, um dann mit den Daten der neuen Connected Components überschrieben zu werden. Mit dieser Methode soll ein unnötiger Rechenaufwand durch das Löschen und Neuerstellen von GameObjects erspart werden.

## 4.4 Projektion

In diesem Abschnitt wird erklärt wie die Projektion des Spatial Mapping Mesh auf das segmentierte RGB-Bild funktioniert. Dazu werden die mathematischen Grundlagen und der Renderer erläutert.

### 4.4.1 Funktionsweise

Bei der Projektion soll das Spatial Mapping Mesh auf ein RGB-Bild projiziert werden. Dazu muss zwischen verschiedenen Koordinatensystemen umgeformt werden. In dieser Arbeit sind dies das Weltkoordinatensystem, das Kamerakoordinatensystem, das Bildkoordinatensystem und die Bildmatrix (siehe Abb. 4.3).

Die Photocapture-API der HoloLens liefert bei der Aufnahme von Fotos zwei Transformationsmatrizen mit; Die CameraToWorld- und die Projektionsmatrix.

## 4 Systemarchitektur

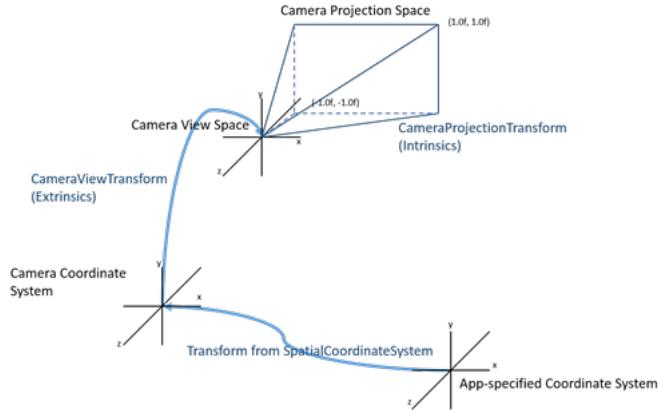


Abbildung 4.3: Die Verfahrensweise zur Lokalisierung eines Pixels im Bildkoordinatensystem in der Microsoft HoloLens (Microsoft 2019c).

Sie berechnen sich aus den extrinsischen bzw. den intrinsischen Kameraparametern. Die extrinsischen Kameraparameter sind drei Parameter der Translation und drei Parameter der Rotation. Verzerrung, Brennweite und die beiden Koordinaten des Hauptpunktes bilden die intrinsischen Parameter. Für beide Operationen werden homogene Koordinaten verwendet. Sie werden zur Umrechnung vom Kamerakoordinatensystem in das Weltkoordinatensystem und zur Umrechnung vom Kamerakoordinatensystem in das homogene Bildkoordinatensystem genutzt:

Sei  $C$  die  $4 \times 4$  CameraToWorld-Matrix

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4.1)$$

$P$  die  $4 \times 4$  Projektions-Matrix (Microsoft 2019c)

$$P = \begin{bmatrix} fx & 0 & 0 & 0 \\ skew & fy & 0 & 0 \\ cx & cy & -1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^T, \quad (4.2)$$

$v_{world}$  ein homogener  $4 \times 1$  Vertex im Weltkoordinatensystem

$$v_{world} = [x \ y \ z \ 1]^T, \quad (4.3)$$

$v_{camera}$  ein  $4 \times 1$  Vektor im Kamerakoordinatensystem

$$v_{camera} = [x_c \ y_c \ z_c \ w_c]^T, \quad (4.4)$$

$v_{image}$  ein  $2 \times 1$  Vektor im Bildkoordinatensystem

$$v_{image} = [x_i \ y_i]^T \quad (4.5)$$

und  $u$  ein  $2 \times 1$  Vektor in der Bildmatrix

$$u = [x_p \ y_p]^T. \quad (4.6)$$

$v_{tmp}(w)$  bezeichnet hierbei die w-Komponente des Vektors. Um einen Vertex  $v_{world}$ , der in Weltkoordinaten vorliegt, auf den korrespondierenden Bildpixel zu projizieren, werden folgende Operationen durchgeführt:

$$v_{camera} = C^{-1} \cdot v_{world}, \quad (4.7)$$

$$v_{tmp} = P \cdot v_{camera}. \quad (4.8)$$

Umwandlung homogener zu inhomogenen Koordinaten:

$$v_{image} = \frac{1}{v_{tmp}(w)} \cdot \begin{bmatrix} v_{tmp}(x) \\ v_{tmp}(y) \end{bmatrix}. \quad (4.9)$$

Berechnung der Pixelposition  $u$  in der Bildmatrix:

$$p_x = \frac{(x_i + 1)}{2} \cdot width, \quad (4.10)$$

$$p_y = \frac{(-y_i + 1)}{2} \cdot height. \quad (4.11)$$

Microsoft definiert das Bildkoordinatensystem anders als es meistens in der Literatur der Fall ist. Während normalerweise das Bildkoordinatensystem von (0,0)



Abbildung 4.4: Bild einer Szene (links) und das korrespondierende Rendering (rechts) basierend auf dem gegebenen Mesh. Gut zu erkennen sind Bereiche der Umgebung, die nicht gescannt wurden, bzw. nicht nach den letzten Veränderungen gescannt wurden, da das Mesh sie nicht berücksichtigt.

bis  $(1,1)$  definiert ist, ist es bei Microsoft von  $(-1,-1)$  bis  $(1,1)$  definiert. Dies wird durch eine Anpassung bei der Berechnung der Bildpixel berücksichtigt.

#### 4.4.2 Splatting/Rendering

Bei gegebenem Mesh und RGB-Bildern muss eine Sichtbarkeitsprüfung für die Vertices des Meshs durchgeführt werden. Bei Eingabe eines RGB-Bildes soll die Ausgabe ein Array der Bildgröße sein, in dem zu jedem Pixel der Abstand zur Kamera steht, den ein Vertex maximal haben darf, um sichtbar zu sein. Die Ausgabe hat also die Dimension  $H \times W$ . Im Folgenden werden zwei mögliche Rendertechniken, das Splatting und das Rendering, analysiert. Am Ende treffen wir eine begründete Wahl:

**Splatting** bezeichnet ein Renderingverfahren, bei dem die Projektion eines einzelnen Vertex mehrere Bildpixel beeinflusst. Dazu wird eine statische Maske gewählt, die Auskunft darüber gibt, dass die Projektion eine bestimmte Anzahl an Pixeln um den von der Projektion getroffenen Pixel mit beeinflusst. Für das Berechnen einer Tiefenkarte bedeutet dies, dass der Abstand des Pixels von der Kamerapose auf alle Pixel in der Nähe übertragen wird. Die Größe der Maske wird vorher statisch festgelegt und ist für alle Pixel identisch. Das Ergebnis dieser Methode soll gute Projektionsergebnisse liefern, da „Löcher“ im Mesh bei der Projektion künstlich gestopft werden. Problematisch ist allerdings die feste Größe und das Vernachlässigen der Vertexzusammenhänge im Mesh. Zudem muss ein Mittelweg zwischen Löchern in der Projektion und der Größe der Splattingmaske gefunden werden, da eine zu große Maske eine ungenaue Projektion verursacht. Das Ergebnis dieser Methode ist eine Tiefenkarte, die für jeden Pixel einen Abstandswert von der Kamera enthält, den ein projizierter Vertex maximal haben darf, um im sichtbaren Bereich zu liegen.

Für eine gegebene Projektion eines Vertex  $v_{world}$  auf die Koordinaten  $pixel_x$  und  $pixel_y$  im Bild, wird folgende Formel beim Splatting verwendet:

$$\begin{aligned} if(z < \text{splatting}[x_p, y_p]) : \\ \text{splatting}[x_p \pm offset, y_p \pm offset] = z \end{aligned}$$

Wichtig hervorzuheben ist, dass für diese Technik mindestens zweimal über die Menge aller Vertices iteriert werden muss. Das erste Mal, um das „Splatting“-Array mit den Tiefeninformationen zu füllen und einmal um die Entfernung der Vertices mit den Einträgen des Arrays zu vergleichen.

Beim **Rendering** wird mithilfe des Meshes, des aktuellen Bildes, der Projektions- und der CameraToWorld-Matrix ein Tiefenbild aus Sicht der Kamerapose berechnet. Dieses Tiefenbild enthält für jeden Pixel des Bildes den berechneten Abstand des Meshs zur Kamerapose. Es werden nur die Klassenwahrscheinlichkeiten der Vertices geupdatet, deren Abstand zur Kamera eine Differenz von maximal 0.01 m zur vom Render berechneten Tiefe besitzt. Die restlichen Vertices werden nicht berücksichtigt, da sie nicht im sichtbaren Bereich liegen. Dies löst die Outlier-Problematik und kann auch mögliche Fehler im Spatial Mapping auffangen. Durch die Verwendung des Meshes als Eingabe gibt es bei dieser Methode keinen Verlust an Informationen (siehe Abb. 4.4). Der Renderer wird anfangs mit dem Mesh initialisiert und bekommt für jedes Bild nur noch die Kamerapose übergeben.

Aufgrund der Vorteile von Rendering gegenüber Splatting wie einer schnelleren Laufzeit durch Nutzung der GPU, Verwendung des gesamten Meshes als Informationsquelle und dem Fehlen des selbstständigen Ausbalancierens zwischen Filtergröße und Ungenauigkeiten, wurde sich für das Rendering entschieden.

## 4.5 Semantic Fusion

Die Projektion des Meshes auf ein segmentiertes RGB-Bild ergibt für sichtbare Vertices eine Wahrscheinlichkeitsverteilung der Klassen. Um diese Zuordnung nicht auf den Sichtbereich eines einzelnen Bildes zu beschränken, sondern im gesamten Mesh durchzuführen, ist eine semantische Fusion notwendig. Als Input dienen hierbei eine beliebige Anzahl an segmentierten Bildern und deren Projektionen. Als Ausgabe soll für jeden Vertex des Meshs eine Klassenzuordnung möglich sein.

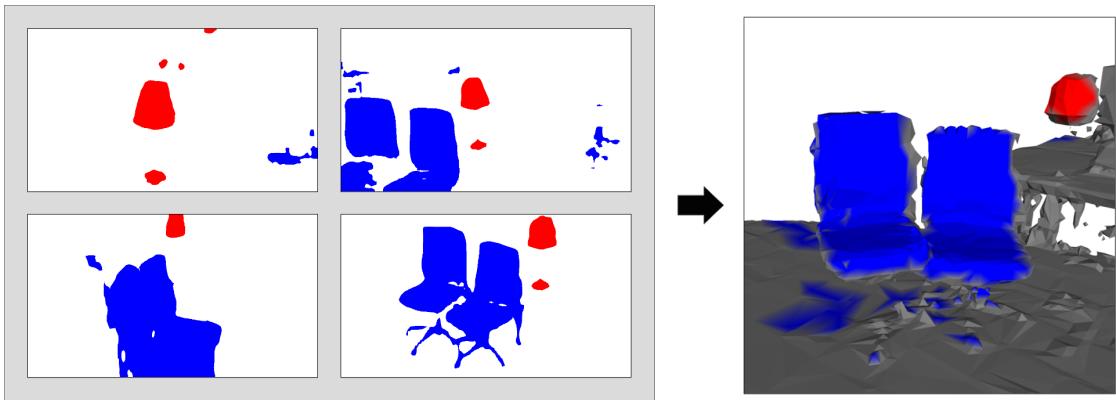


Abbildung 4.5: Semantic Segmentations von vier Frames (links) und das Ergebnis der Semantic Fusion (rechts).

Jeder Vertex  $v$  in unserem Mesh speichert, zusätzlich zu seiner Position, eine diskrete Wahrscheinlichkeitsverteilung  $P(L_v = l_i)$  über die Menge aller Klassen  $l_i \in L$ . Dabei gilt  $L = \{Lamp, Chair, Unknown\}$ . Jeder Vertex wird mit einer Gleichverteilung über die Klassen initialisiert, weil uns lediglich die Bilder Informationen liefern und kein *a priori* Wissen vorliegt.

Der Output von RefineNet ist eine per-pixel Wahrscheinlichkeitsverteilung über die Klassenlabel  $P(O_u = l_i | I_k)$ , wobei  $I_k$  dem  $k$ -ten Bild und  $u$  der Pixelkoordinate entspricht. Die sichtbaren Vertices wurden mithilfe des Renderings ermittelt. Über die bereits diskutierte Projektion wird nun jeder sichtbare Vertex auf einen Bildpixel projiziert. Die Wahrscheinlichkeitsverteilung dieser Vertices wird mit folgender Regel angepasst, die einem rekursiven Bayes-Filter entspricht:

$$P(l_i | I_0, \dots, I_k) = \eta \cdot P(O_{u_{v,k}} = l_i | I_k) \cdot P(l_i | I_0, \dots, I_{k-1}) \quad (4.12)$$

Hierbei ist  $\eta$  der Normalisierungsfaktor der Wahrscheinlichkeitsverteilung.

Die Methodik folgt McCormac, Handa, Davison et al. 2017 und Hermans, Floros und Leibe 2014.

## 4.6 Netzwerkarchitektur

Die Verbindung zwischen Client und Server wird über eine TCP-Verbindung erhalten. Client und Server sind jeweils als TCP Socket implementiert, um eine möglichst große Kontrolle über den Datenstrom zu besitzen.

Ein Problem ist jedoch der große Aufwand beim Versenden der Nachrichten, da jede manuell als Byte-Array codiert werden muss. Zusätzlich muss das Decodieren

auf der Serverseite implementiert werden. Die dadurch gegebene Fehleranfälligkeit und der Aufwand wurden mit einer besseren Netzauslastung und Kontrolle gerechtfertigt.

Die HoloLens kann drei verschiedene Nachrichtenarten an den Server senden:

1. Meshinformationen
2. Bildinformationen
3. Selektionsinformationen

**Meshinformationen** beinhalten das Mesh codiert in einem Byte-Array. Dieses hat folgende Struktur: Länge des Meshstrings (Int32) und im UTF-8-Format codierter String des Meshes. Hierbei ist zu beachten, dass die App das Mesh als String für eine .obj-Datei zusammenfasst und diesen String codiert sendet. Dieser wird auf dem Server in eine Datei geschrieben und als .obj-Datei gespeichert, um von trimesh (Dawson-Haggerty 2017) geöffnet und verwaltet zu werden.

Das Byte-Array für **Bildinformationen** hat folgende Struktur: Bildbreite (Int32), Bildhöhe (Int32), Größe des Byte-Arrays des Bildes (Int32), CameraToWorld-Matrix ( $16 \times \text{Int32}$ ), Projektionsmatrix ( $16 \times \text{Int32}$ ) und die Bildinformationen als Bytes. Das Bild muss serverseitig von BGRA in RGB umgewandelt werden.

**Selektionsinformationen** enthalten den Endpunkt des Raycasts in die Blickrichtung des Nutzer und die Informationen über den Namen des selektierten Hologramms. Der Raycast wird mit drei Floats (je 4 Byte) codiert, da er als Vector3 vorliegt. Mithilfe dieses Punktes ist es möglich, zwischen verschiedenen Gegenständen mit Interaktionsmöglichkeiten in der Umgebung zu unterscheiden. Die Hologramminformationen enthalten den String mit dem Namen des Hologramms und die Länge dieses Strings (Int32). Über den Namen wird dann auf dem Server entschieden, ob die angeschlossene Lampe aktiviert, deaktiviert oder nicht verändert werden soll.

An die HoloLens werden zwei verschiedene Nachrichtenarten gesendet:

1. Rückmeldebenachrichtigungen
2. Informationen über die Connected Components

**Rückmeldebenachrichtigungen** sind lediglich optional und zu Debugzwecken eingebaut. Bei dieser Art von Nachricht wird ein String an die HoloLens gesendet,

## 4 Systemarchitektur

in dem eine Eingangsbestätigung der Informationen codiert ist. Diese Nachricht besteht aus der Länge des Strings (UInt32) und dem String als Byte-Array. Es wird jedoch meistens auf diese Benachrichtigungen verzichtet, da sie Rechenleistung der HoloLens beanspruchen, die Netzwerkauslastung erhöhen und eine Eingangsbestätigung für die App nicht von Relevanz ist.

Das Senden der **Connected Components** an die HoloLens besteht aus zwei Schritten. Im ersten Teil wird der HoloLens mitgeteilt, dass sie neue Connected Components gesendet bekommt. Im zweiten Schritt wird für jeden Component eine Zahl als Maskierung der Klasse (0 für Lampe, 1 für Stuhl)(UInt32) und ein Byte-Array, in dem die Anzahl an Vertices und Faces zusammen mit den Koordinaten der Vertices und der Informationen über die Faces codiert sind, gesendet.

# 5 Auswertung

In diesem Kapitel geht es um die Evaluierung und Auswertung der vorher vorstellten Anwendung. Dazu wird die Hardware präsentiert, Laufzeitanalysen bei unterschiedlicher Parameterwahl miteinander verglichen und qualitative Ergebnisse präsentiert:

## 5.1 Hardware

Die Hardware der Microsoft HoloLens wurde bereits in Kapitel 3.1.1 erläutert. Bei dem genutzten Linux-Server handelt es sich um einen Desktop-PC mit einem Intel(R) Core(TM) i9-9900K ( $8 \times 3,6$  GHz), 64 GB Arbeitsspeicher und einer GeForce RTX 2080. Zur drahtlosen Kommunikation mit der HoloLens wird ein ALFA Networks AWUS036NHR USB-Netzwerkadapter mit einer maximalen Übertragungsrate von 150 MBit verwendet. Das WLAN-Netz, in dem die HoloLens und der Server kommunizieren, wird von einem Windows 10 Rechner mit einem weiteren ALFA Networks AWUS036NHR gehostet. Bei der hier zu segmentierenden Lampe handelt es sich um die IKEA ÅRSTID Tischleuchte mit einem Schirmdurchmesser von 22 cm und einer Schirmhöhe von ca. 22 cm. Sie ist in Abb. 5.1 zu sehen.

## 5.2 RefineNet-Training

Die Parameter für RefineNet wurden wie im Originalpaper von Lin et al. 2017 gewählt. Die Learning Rate wurde auf  $10^{-5}$  festgelegt und die Updates der Gewichte wurden mit dem Adam Optimizer (Kingma und Ba 2014) berechnet.

Als Trainingsdatensatz für RefineNet wurde in dieser Arbeit SceneNet verwendet (McCormac, Handa, Leutenegger et al. 2017). Es handelt sich dabei um einen Datensatz mit mehr als 5 Millionen synthetischen Bildern aus mehr als 15000 Trajektorien in verschiedenen synthetischen Umgebungen. Für jede Aufnahme steht das Tiefenbild, das RGB-Bild und eine Instanzsegmentierung bereit. Die Bilder haben eine Auflösung von  $320 \times 240$  Pixel. Die Trajektorien wurden durch synthetische Räume gelegt, die durch Objekte aus ShapeNet bevölkert wurden. Die

## 5 Auswertung



Abbildung 5.1: IKEA ÅRSTID Tischleuchte (IKEA 2019) und zwei Arten von Bürostühlen. Diese drei Objekte sollen segmentiert und in der HoloLens dargestellt werden.

erstellten Szenen sollen Situationen wie sie in einem Wohnzimmer, Arbeitszimmer oder Schlafzimmer vorkommen könnten, nachstellen. Von Interesse waren Frames, in denen Stühle oder Lampen zu finden sind. Insgesamt gibt es 9985 Frames mit Lampen und 87393 Frames mit Stühlen.

Die Lampen sind weiterhin unterteilt in die Kategorien lamp, floor\_lamp, table\_lamp und Stühle in die Kategorien armchair, chair, deck\_chair, easy\_chair, folding\_chair, lawn\_chair, morris\_chair, rocking\_chair, straight\_chair, swivel\_chair, windsor\_chair, wing\_chair. Es wurde keine Unterkategorie ausgeschlossen, um eine möglichst robuste Segmentierung zu ermöglichen. Aufgrund der großen Anzahl an Trainingsframes wurde mithilfe von Slicing nur jedes dritte Bild in den Trainingsdatensatz übernommen. Dieser Datensatz wurde abgespeichert und beim Training übergeben. Es wurde für 20 Epochen trainiert und die dann erhaltenen Modellparameter verwendet.

## 5.3 Laufzeitanalysen

Die Ergebnisse der Anwendung werden durch die Variation der Parameter miteinander verglichen. Variable Parameter sind:

- Dreiecke pro  $m^3$  beim Spatial Mapping (Default: 800)
- Wartezeit in Frames zwischen der Aufnahme von Bildern (Default: 100)

Andere Parameter wie die Wartezeit bis zum Senden des Meshs wurden nicht abgeändert, weil sie keine Auswirkungen auf die Performance haben. Durch eine erhöhte Auflösung des Meshes wird ein höherer Aufwand für die Projektion und somit möglicherweise eine größere Verzögerung zwischen Selektion und Aktion vermutet.

Es werden die Netzwerkauslastung, die FPS und der Backlog der ungesendeten Bilder aufgezeichnet. Mithilfe der FPS lassen sich Rückschlüsse über die visuelle Darstellung der HoloLens ziehen und die Netzwerkauslastung sowie der Backlog geben Auskunft darüber, wie gut die Kommunikation optimiert ist.

Die Bildgröße wurde nicht geändert, weil die Auflösung mit  $896 \times 504$  px für Bildverarbeitungen bereits hoch genug ist. Zudem ist der Datenaufwand bei jeder Erhöhung der Auflösung nicht im Verhältnis zum Nutzen. Die Nutzung der nächst höheren Auflösung  $1280 \times 720$  würde bereits die Datenmenge mehr als verdoppeln (von 1.806.336 zu 3.686.400 Bytes).

Die Messungen wurden bei einem statischen Versuchsaufbau und der Aufnahme von insgesamt 110 Bildern durchgeführt, um vergleichbare Ergebnisse zu bekommen. Der Backlog gibt die max. Anzahl an ungesendeten Bildern auf der Hololens an.

Die Frame Rate der Anwendung war bei allen Anwendungen bei durchschnittlich 61 FPS, womit gezeigt wurde, dass die Anwendung für den Nutzer kein massives Ruckeln zeigt und flüssig läuft. Minimal waren 57 FPS. Die Übertragungsrate über die drahtlose Verbindung zwischen HoloLens und Server lag bei  $\approx 2 \frac{\text{MB}}{\text{s}}$ , wenn sie voll ausgelastet wurde.

Durch die Veränderung der Auflösung konnte kein signifikanter Unterschied in den hier ausgeführten Laufzeittests erkannt werden. Die Wartezeit zwischen der Bildaufnahme hat, im Gegensatz zur Auflösung, einen großen Einfluss auf die Performance der Anwendung. Bei der Bildaufnahme alle 30 Frames, also ca. zweimal die Sekunde, entstand ein großer Backlog von  $\approx 37$  bis 50 Bildern. Durch die Zeitverzögerung zwischen Versenden und Aufnahme der Bilder verlieren die Informationen den Zusammenhang zwischen Sichtfeld des Nutzers und internem Zustand,

## 5 Auswertung

Auflösung [Dreiecke/ $m^3$ ]	Wartezeit Fotoaufnahme [Frames]	Netzwerk- auslastung [MB/s]	Max. Backlog	Laufzeit [s]
800	30	1,92	50	105
800	60	1,80	3	112
800	100	1,09	0	186
1500	30	2,15	37	94
1500	60	1,76	1	115
1500	100	1,09	0	186

Tabelle 5.1: Messdaten zur Laufzeitanalyse der präsentierten Anwendung. Die hier angegebene Laufzeit entspricht der Anwendungslaufzeit ab der Übertragung des Meshes an den Server.

weshalb diese Variante als nicht praktikabel verworfen werden muss. Durch den Backlog liegt die Laufzeit in Sekunden nur etwas niedriger, als bei einem Bild pro Sekunde. Weiterhin zu vermerken ist, dass der Backlog im Laufe der Anwendung nur größer wurde.

Die Aufnahme eines Bildes alle 60 Frames funktioniert in den meisten Fällen ohne Backlog. Lediglich bei 5% der Bilder kam es zu einem Backlog von maximal drei Bildern. Es lässt sich somit festhalten, dass die Anwendung in der Lage ist ein Bild pro Sekunde zu verarbeiten und an den Server zu versenden.

Bei der Aufnahme eines Bildes alle 100 Frames bildete sich nie ein Backlog, jedoch war auch die Netzwerkauslastung nicht so hoch, wie bei den anderen Testfällen. Da es der Anwendung auch möglich ist ein Bild pro Sekunde zu verarbeiten, sollte diese Variante bevorzugt werden.

Die Verzögerung zwischen Selektion der Lampe und Reaktion lag in den Tests bei allen Anwendungsvarianten bei  $\approx 1,25$  Sekunden. Dieser Versatz kann teilweise durch die Implementierung des Sockets erklärt werden: Der Socket versendet erst die Selektionsinformation, wenn die Verbindung zum Server frei ist. Der Server sendet allerdings jede Sekunde ein Bild und die Selektionsnachricht muss somit warten bis das Versenden des Bildes abgeschlossen wurde, um an den Server übertragen zu werden. In Zukunft sollte sich mit diesem Problem genauer befasst werden. Eine Lösungsidee wären möglicherweise zwei Verbindungen zwischen HoloLens und Server zu erhalten, wobei die eine lediglich die Selektionsinformationen sendet.

## 5.4 Qualitative Ergebnisse

Aufnahmen aus Sicht des Nutzers für die HoloLens-App liegen in mehreren Parametervarianten als Video vor. Die Aufnahmen wurden mithilfe der Mixed Reality Capture gemacht. Die Frame Rate einer Anwendung wird durch diese auf maximal 30 FPS begrenzt. Ohne Videoaufnahme liegt sie bei durchschnittlich 60, weshalb die Anwendung in Videos langsamer Ergebnisse präsentiert.

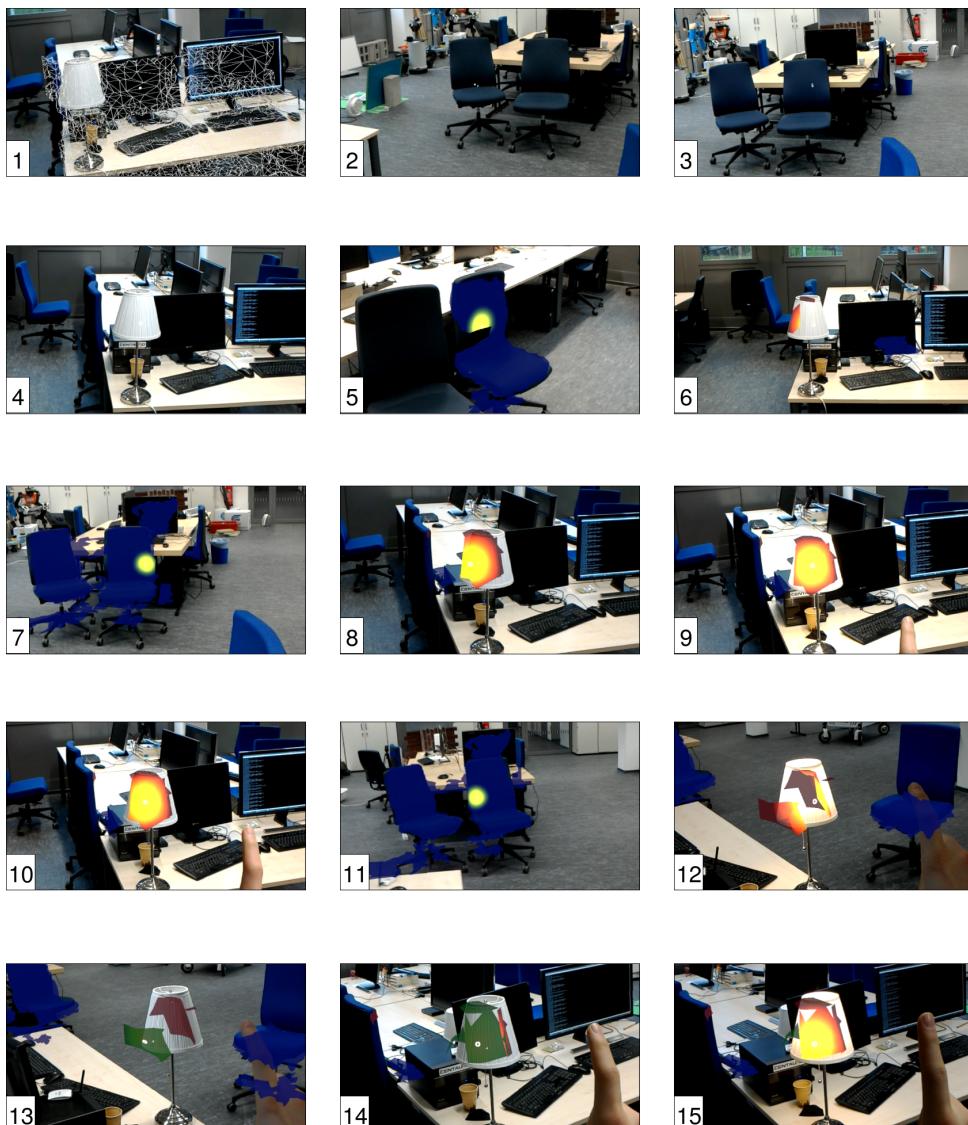


Abbildung 5.2: Screenshots aus den ersten 2.700 Frames eines Anwendungsdurchlaufs.  
Die Auflösung des Mesh war auf 800 Dreiecke pro  $m^3$  gesetzt und Bilder wurden ab Frame 1000 alle 100 Frames aufgenommen. Siehe Text für ausführlichere Erklärung der Bilder.

## 5 Auswertung

In Abb. 5.2 sind mehrere Frames aus einem Durchlauf der App entnommen worden. Die Ergebnisse der Segmentierung am Ende basieren auf insgesamt 15 Bildern der Umgebung. Im Folgenden wird eine detaillierte Erklärung der abgebildeten Frames geliefert:

- 1: Die Umgebung des Nutzers wird 1000 Frames lang gescannt. Um dies zu verdeutlichen und dem Nutzer eine Rückmeldung über das Aussehen des Meshs zu geben, wird das Mesh als Wireframe visualisiert.
- 2-4: Sobald der Scan der Umgebung abgeschlossen wurde, wird die Visualisierung des Mesh abgeschaltet, um den Nutzer visuell nicht zu verwirren. Die ersten Bilder der Umgebung werden in diesen Frames aufgenommen und gesendet.
- 5-7: Die ersten segmentierten Connected Components werden dargestellt. Die Segmentierung hat bisher nur geringe Teile der Lampe erkannt. Gut zu sehen ist in Bild 7 das Hervorheben der Cursorposition durch einen gelben Kreis.
- 8: Die Lampe wird anvisiert und selektiert.
- 9: Aufgrund der Selektion in Bild 8 ist die Lampe eingeschaltet. Mit einer erneuten Selektion soll die Lampe nun ausgeschaltet werden.
- 10: Die Lampe ist ausgeschaltet.
- 13-14: Aufgrund der Selektion in Bild 12 sind Teile des Mesh von rot (nicht selektiert) auf grün (selektiert) gewechselt.
- 15: Erneutes Selektieren ändert die Farbe wieder auf rot und schaltet die Lampe ein.

Besonders bei dem Mesh der Lampe ist zu erkennen, dass sich dieses mit dem Nutzer leicht mitbewegt und nicht an der exakten Position im Raum bleibt (vgl. Abb 5.2 Bilder 12 und 15). Dies hängt einerseits mit einer nicht korrekten Lokalisierung der HoloLens zusammen, als auch mit der Verankerung des Hologramms. Alle Meshes, die in der HoloLens angezeigt werden, wurden in dieser Arbeit mit dem Ankerpunkt (0,0,0) versehen. Microsoft empfiehlt ab einer Entfernung von 5 Metern vom Ursprung die Nutzung eines näheren Ankerpunkts. In dieser Anwendung wird die Distanz von 5 Metern zwar nie überschritten, aber möglicherweise könnte eine neue Art der Verankerung trotzdem stabilere Hologrammpositionen liefern.

#### 5.4 Qualitative Ergebnisse

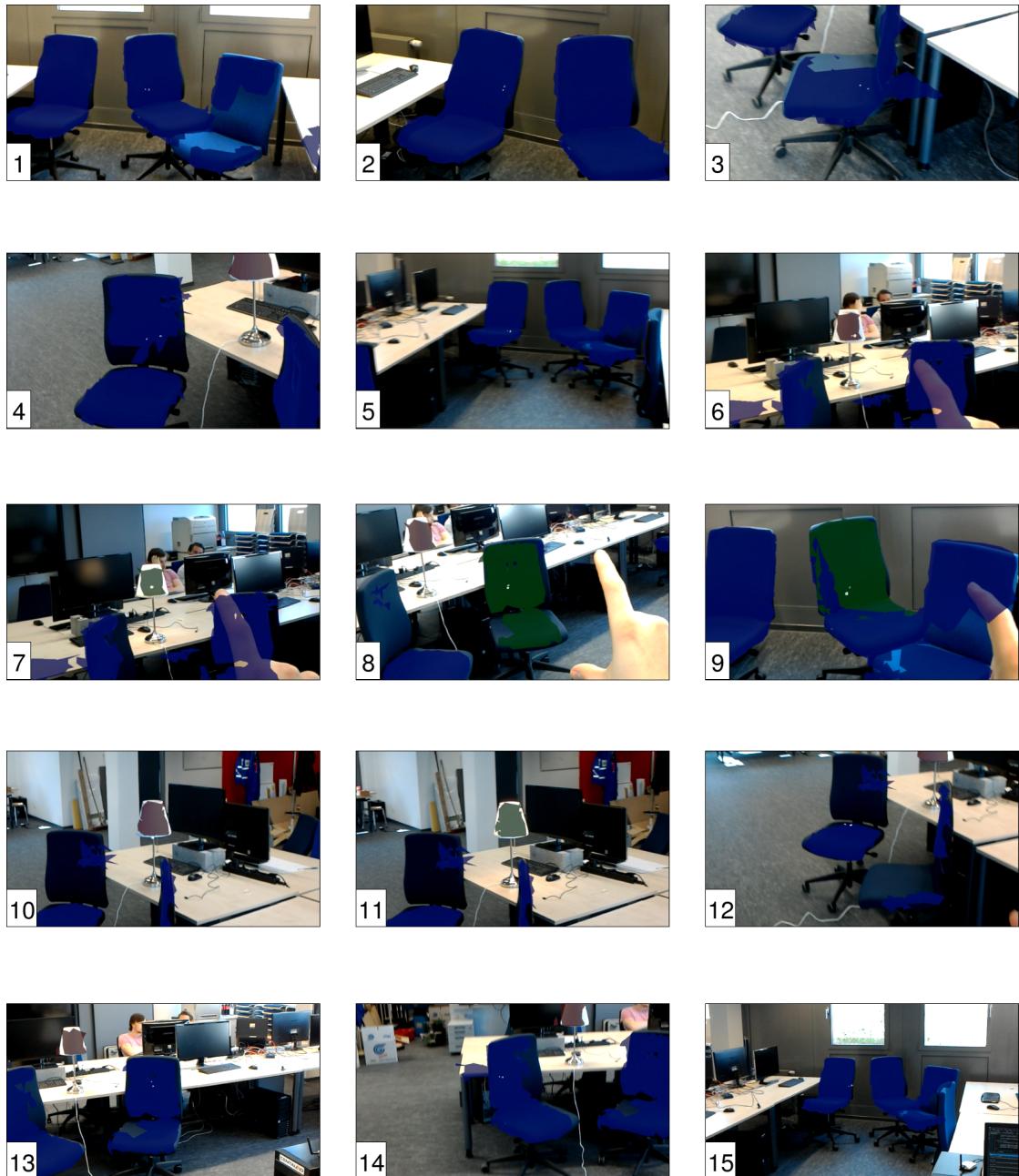


Abbildung 5.3: Aufnahmen aus einem 12000 Frames langem Anwendungsdurchlauf. Die Auflösung des Mesh war auf 800 Dreiecke pro  $m^3$  gesetzt und Bilder wurden ab Frame 1000 alle 100 Frames aufgenommen. Das Hervorheben des Cursors durch Farbe wurde für die Aufnahme abgeschaltet, um das Leuchten der Lampe auf Bildern besser erkennen zu können. In Bild 8 wurde der Stuhl selektiert (grünes Hologramm) und die Lampe ist wie erwartet nicht angeschaltet worden.

## 5 Auswertung

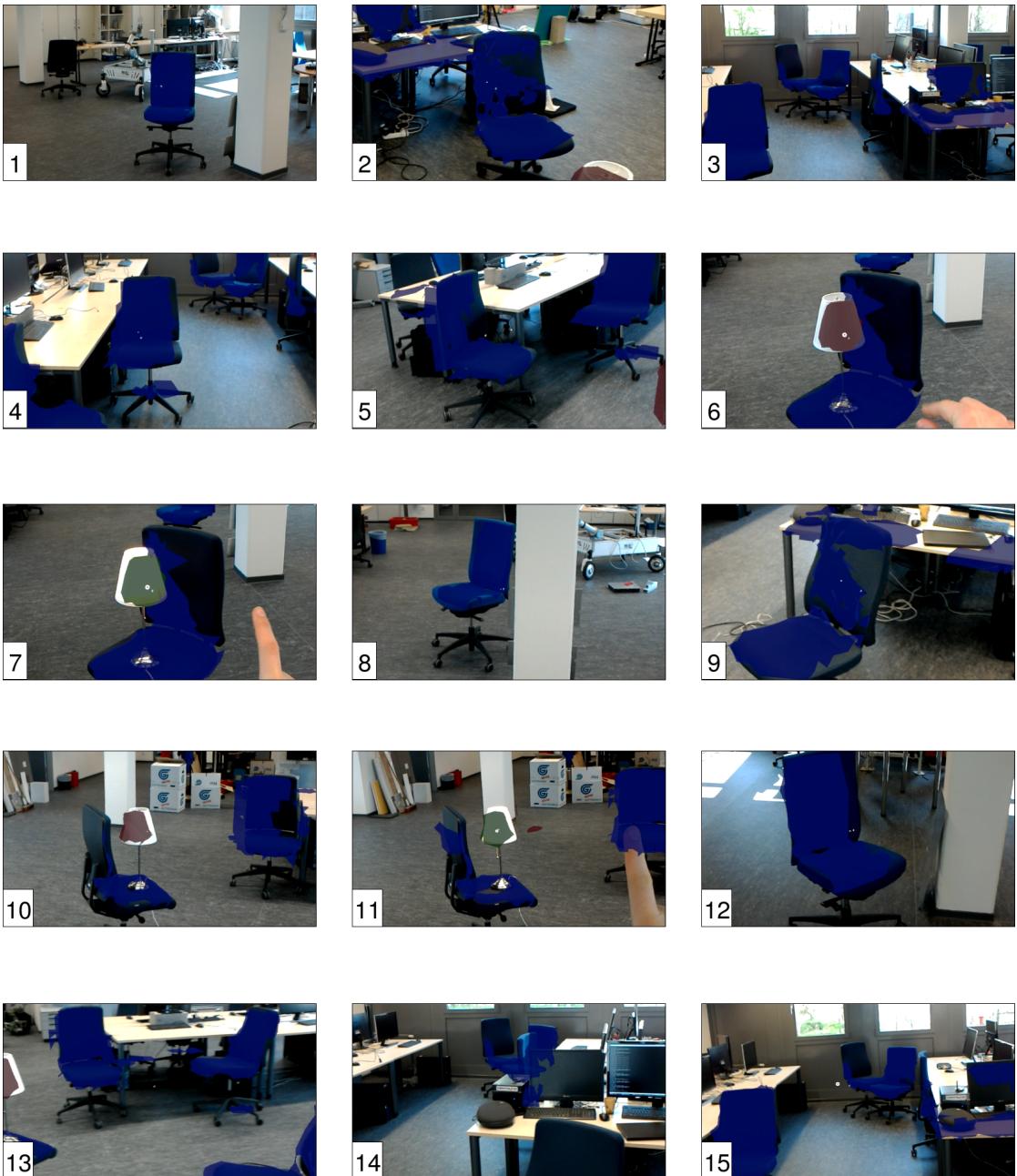


Abbildung 5.4: Auf den Bildern 2,3 und 9 erkennt man, dass das ausführliche Scannen der Umgebung wichtig ist, da dieser Stuhl nicht komplett ins Mesh übertragen wurde und dadurch die Projektionen den Tisch und Bildschirm hinter dem Stuhl treffen. Die Auflösung des Mesh war auf 800 Dreiecke pro  $m^3$  gesetzt und Bilder wurden ab Frame 1000 alle 100 Frames aufgenommen.

Ein wichtiger Faktor der Umsetzung war die Visualisierung der Segmentierungsergebnisse, deren Aussagekraft und die robuste Interaktion mit ihnen:

Die Designentscheidung, das Austauschen der Connected Components lediglich im 5-Bilder-Takt durchzuführen, hat sich in Tests durch eine gute Nutzbarkeit ausgezeichnet. In der momentanen Implementierung ist es dem Nutzer nicht möglich, mit dem Hologramm zu interagieren, während dieses durch ein neues Hologramm ausgetauscht wird. Dieser Prozess benötigt ca. eine halbe Sekunde und würde, wenn er nach jedem Bild auftritt, massiv die Interaktionsmöglichkeiten einschränken. Eine Empfehlung wäre die Darstellung aufwendiger zu implementieren, um die Hologramme nicht immer austauschen zu müssen, sondern sie zu erweitern oder einen Platzhalter an ihre Position zu setzen, der die Interaktion registriert und für eine spätere Bearbeitung abspeichert.

Die geringe Auflösung des Meshes wird durch seine Nutzung als Visualisierung auch hier ein Problem. Die Lampe wird durch ihre relativ geringe Größe und besondere Form meistens nur teilweise gescannt und besitzt dort auch nicht die konische Form wie in der Realität. Bei Stühlen fällt diese Problematik nicht derart ins Gewicht, weil ihre Größe eine realistische Darstellung im Mesh erlaubt. Allerdings ist einer guten und eindeutigen Darstellung des Interaktionsobjekts Priorität einzuordnen. Wir würden empfehlen, in zukünftigen Arbeiten abstrakte Icons einzublenden, um die Semantik zu visualisieren. Dort wo eine Lampe vermutet wird, würde ein Hologramm oder ein Bild einer Lampe platziert werden können. Durch passende Abstandsmetriken müssten zusätzlich unvollständige Segmentierungen erweitert werden, um möglichst den gesamten Bereich der Lampe zu segmentieren. Die Interaktionsmöglichkeit bliebe erhalten und es wäre für den Nutzer ästhetisch ansprechender und verdeckt einen geringeren Teil des Sichtfeldes.

In den ersten Frames der App sind vermehrt false positive Segmentierungen zu sehen, die im Laufe der Nutzung verschwinden. Lediglich durch schlechte Segmentierungsergebnisse sind diese nicht zu erklären, weshalb ein unvollständiges Rendering als weitere Ursache identifiziert wurde. Das Mesh ist meistens nicht in der Lage die Umgebung exakt widerzuspiegeln und somit trifft die Projektion eines Stuhl- oder Lampenpixels nicht den erwarteten Vertex. Dies ist insbesondere bei Stuhlkanten ersichtlich, da die gerade Linie der Kante nicht immer gut nachgebildet wird. Durch die Fusion sinkt die Anzahl der false positives mit steigender Laufzeit, weil die Szene aus anderen Blickwinkeln betrachtet wird und das unvollständige Rendering geringere Auswirkungen hat.

## 5 Auswertung

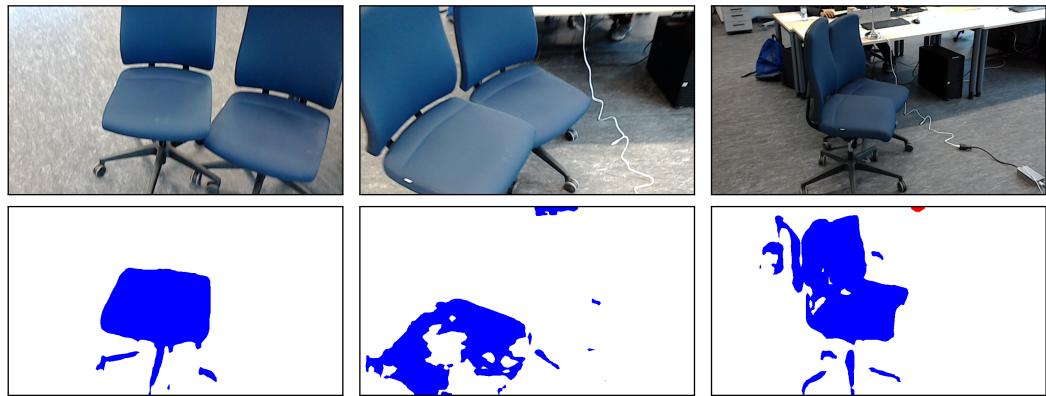


Abbildung 5.5: Drei Bilder und ihre korrespondierenden Segmentierungen. In den Bildern links und in der Mitte ist ersichtlich, dass die Segmentierung bei Teilansichten von Bürostühlen versagt, allerdings sehr gute Ergebnisse liefert, sobald der gesamte Stuhl wieder sichtbar ist (rechts).

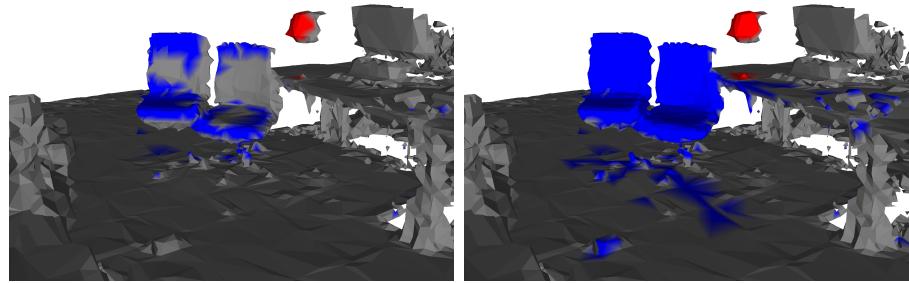


Abbildung 5.6: Das Ergebnis der Fusion mit gleichen Confidences bei jedem Abstand weißt bei Stühlen einige Segmentierungsprobleme auf (links). Durch die Anpassung des Updates basierend auf der Entfernung sinkt die Anzahl an false negatives, aber die Anzahl an false positives nimmt zu (rechts).

In den meisten Appdurchläufen war zu erkennen, dass die korrekte Segmentation der Stühle über viele Frames hinweg ein großes Problem bedeutet hat. Daraufhin ausgeführte Tests haben ergeben, dass das trainierte RefineNet Problem damit hat, alle Bereiche eines Bürostuhls richtig zu segmentieren, wenn dieser nur teilweise sichtbar sind. Dies ist in Abb. 5.5 beispielhaft dargestellt. Durch diese Problematik sind die Ergebnisse der Semantic Fusion für Stühle sehr schlecht, wenn viele Bilder in der näheren Umgebung dieser aufgenommen wurden. Die Lampe zeigt diese schlechten Segmentierungsergebnisse nicht, weil ihre geringe Größe und die Positionierung es erlaubt, sie in den meisten Fällen immer ganz im Bild aufzunehmen.

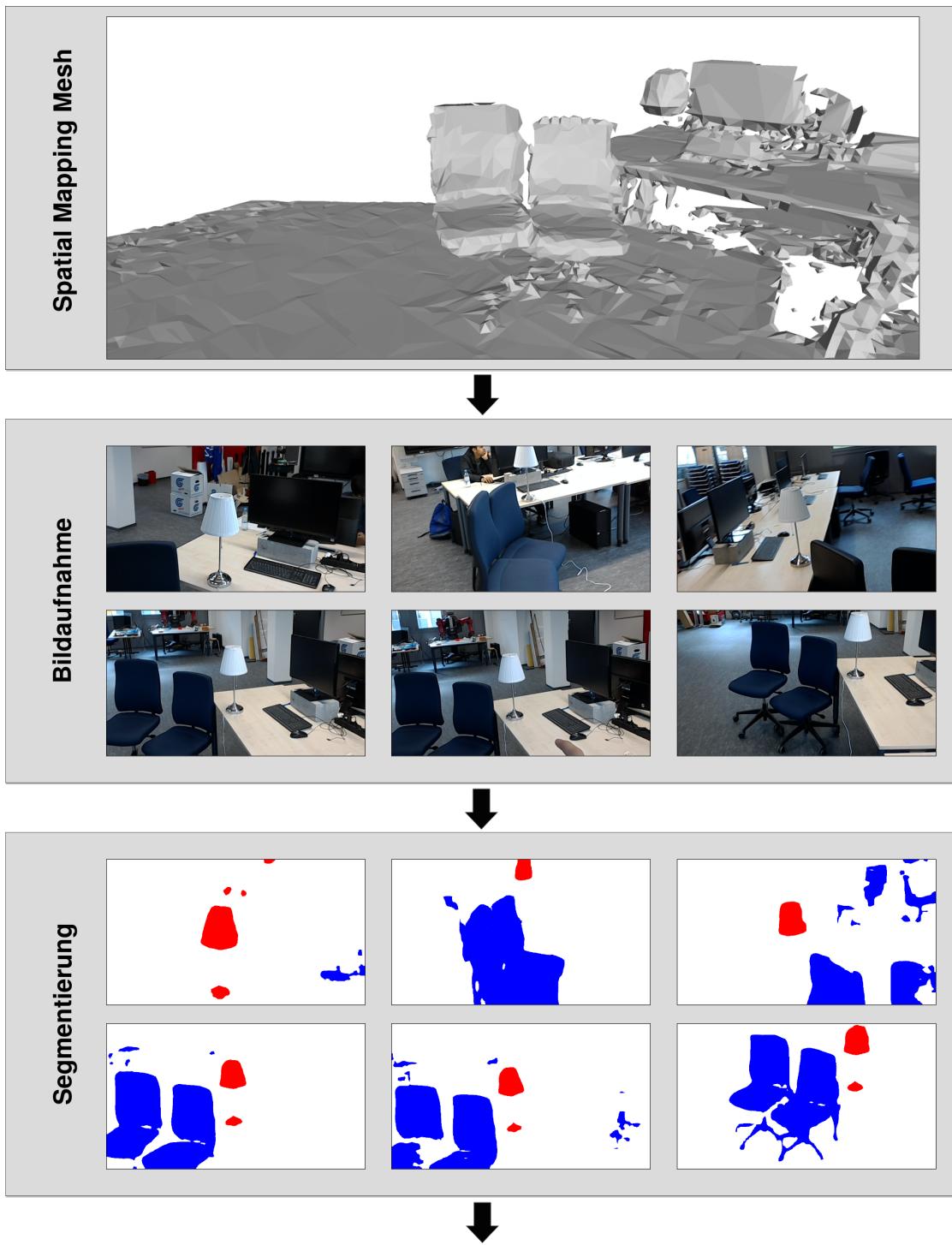
Die Idee unseres Lösungsansatzes ist die Ergebnisse der Segmentierung, die „Unknown“ als wahrscheinlichste Klasse haben und weniger als zwei Meter von der Ka-

#### *5.4 Qualitative Ergebnisse*

meraposition entfernt sind, nicht im Update zu berücksichtigen. Auf diese Weise wird das Ergebnis der Semantic Fusion nicht durch false negatives der schlechten Segmentierung von Stühlen beeinflusst. Problematisch ist eine daraus entstehende zu hohe Confidence der Klassen „Lamp“ und „Chair“ und vermehrte false positives durch ein fehlendes Update der Confidence dieser Vertices (vgl. Abb. 5.6). Es wird daher empfohlen, viele Bilder aus einer Entfernung von mehr als zwei Metern zu machen, um somit die false positive klassifizierten Vertices wieder zu updaten und anzupassen.

Dieser Lösungsansatz sollte in Zukunft überarbeitet werden, um eine ausgeklügeltere Idee mit weniger Nachteilen umzusetzen. Ein erster Schritt wäre die Analyse des Trainingsdatensatzes und eine Anpassung dessen, damit auch Stuhlteile sicher segmentiert werden.

## 5 Auswertung



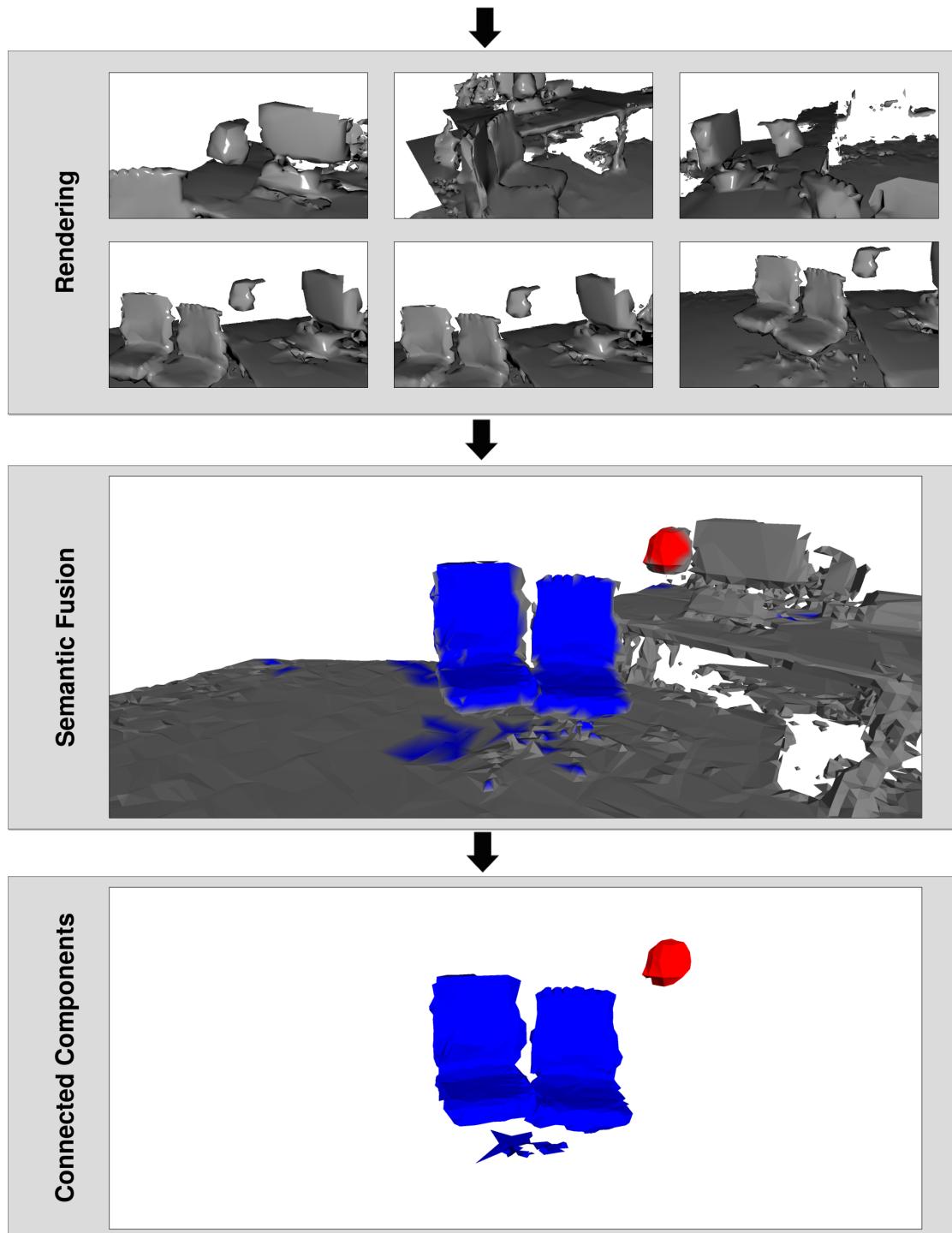


Abbildung 5.6: Die Zwischenschritte der Pipeline auf dem Server an 6 Beispielbildern dargestellt. Das Ergebnis der Fusion basiert lediglich auf den in der Abbildung gezeigten Bildern. Durch das Herausfiltern von zu kleinen Connected Components bezieht das Ergebnis weniger false positives ein. Das Mesh hat eine Auflösung von 800 Dreiecken pro  $m^3$  und die Bilder  $896 \times 504$  px.

## 5 Auswertung

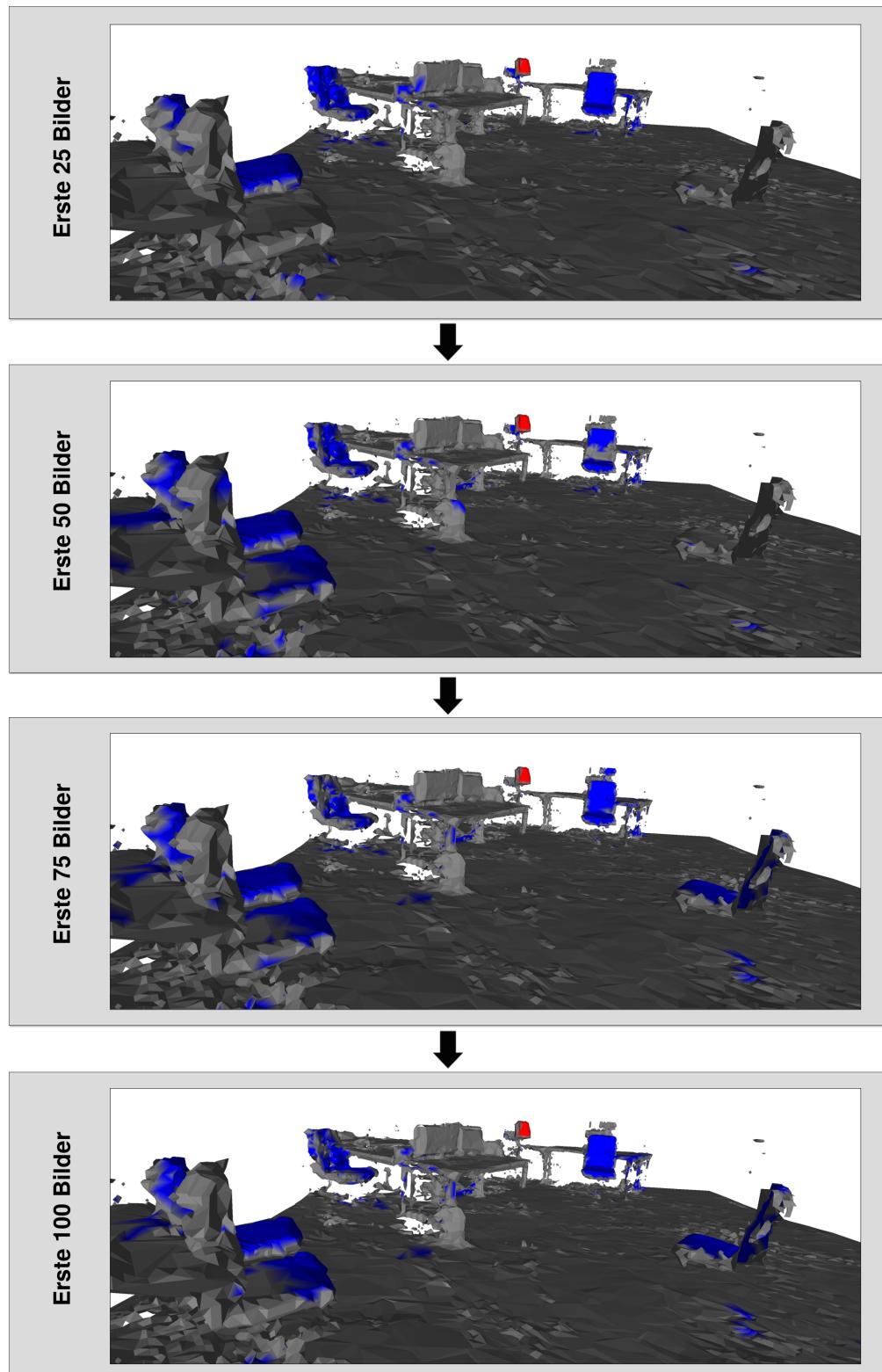


Abbildung 5.7: Das Ergebnis der Semantic Fusion nach jeweils 25 Bildern. Gut erkennbar ist, dass die Fusion true positives durch falsche Segmentierungsergebnisse wieder fallen lässt (Stuhl rechts hinten erstes Bild).

# 6 Diskussion

In dieser Arbeit wurde eine Anwendung präsentiert, die es dem Nutzer einer HoloLens erlaubt, eine dynamisch erkannte Lampe in einer statischen Umgebung anzubzw. auszuschalten. Die Umgebung wurde dabei mithilfe der Spatial-Mapping-Fähigkeiten der HoloLens gescannt, die Kommunikation erfolgte über eine drahtlose Verbindung, die Segmentierung der aufgenommenen Bilder wurde mit RefineNet durchgeführt und die Erweiterung der Segmentierung auf 3D mit einer Semantic Fusion umgesetzt.

Es wurden qualitative Ergebnisse präsentiert und diese miteinander verglichen. Die gesetzten Ziele wurden unter Berücksichtigung der Annahmen erreicht und die Architektur wurde erfolgreich umgesetzt. Dem Nutzer einer HoloLens wurde ermöglicht, seine Umgebung zu scannen und dieser mithilfe von Segmentation und Fusion Semantik zu verschaffen. Die Bildaufnahme lief bei einer Bildaufnahme pro Sekunde ohne Probleme. Die Semantik kann zum Ein- und Ausschalten einer Lampe genutzt werden. Die Fusion wurde mithilfe einer Methode, die einem rekursiven Bayes-Filter entspricht, umgesetzt, die Darstellung der Ergebnisse mithilfe von Connected Components und die Interaktion über die Selektionsgeste und einen Mikrocontroller. Aufgrund des Fehlens einer Metrik und der Einzigartigkeit dieser Arbeit ist es jedoch schwer umsetzbar, einen Vergleich zu ähnlichen Arbeiten zu ziehen.

Die HoloLens hat zudem einige Schwachstellen, die die Programmierung für sie einschränken. Das Spatial Mapping Mesh hat eine zu geringe Auflösung, um eine große Detailfülle zu erlauben und eignet sich vor allem für große Objekte und Oberflächen. Zusätzlich ist das Erkennen von sehr dunklen Oberflächen nur durch mehrfache Scans möglich. Der Bereich in dem Hologramme sichtbar sind ist mit ca.  $34^\circ \times 19.5^\circ$  sehr klein und deckt bei Weitem nicht den Sichtbereich des Nutzers ab, sodass das immersive Gefühl verloren geht.

## 6.1 Ausblick

In Zukunft sollten einige der hier getätigten Annahmen verallgemeinert werden, um die Auswirkungen auf die Segmentierungsergebnisse zu evaluieren.

Beispiele hierfür sind die Annahme der statischen Welt und die Verwendung der vorverarbeiteten Meshdaten als Informationen. Durch die Integration der Surface-Changed-Events des Spatial Mapping Managers wäre es umsetzbar, die Veränderungen der Umgebung zu erfassen und so auch die Änderungen in kurzen Zeitabständen dem Server zur Verfügung zu stellen. Die Nutzung der rohen Daten zur Erstellung des Meshes auf dem Server könnte eine Chance sein, das serverseitige Mesh wesentlich detaillierter zu gestalten und somit die Segmentierungsergebnisse genauer anzeigen zu lassen. Zudem muss sich genauer mit dem Netzwerk zur Semantic Segmentation auseinandersetzt werden, um eine robuste und genaue Segmentierung aus jedem Blickwinkel zu erlauben. Zudem sollte die hier vorgestellte Architektur wesentlich ausführlicher getestet werden, um Schwachstellen und Bottlenecks zu identifizieren, um die beste Parameterwahl für die Bildaufnahme, Auflösung des Meshs etc. zu finden.

Die Integration mehrerer Interaktionsmöglichkeiten ist ein weiterer interessanter Aspekt für zukünftige Arbeiten. Seien es lediglich mehrere Lampen, die räumlich voneinander unterschieden werden müssen, oder ein Roboterarm, der durch Gestensteuerung Aktionen mit Objekten ausführt.



Abbildung 6.1: Die HoloLens 2 in der Seitenansicht (Microsoft 2019a). Durch die Verbesserungen im Vergleich zur ersten Version werden neue Anwendungsbereiche eröffnet.

Durch die Nutzung der HoloLens 2, die Mitte 2019 erhältlich sein soll, können einige dieser Ideen besser umgesetzt werden. Sie bietet im Vergleich zur HoloLens wesentlich verbesserte Hard- und Software und erhöht vermutlich auch das Interesse an der Verwendung. Vorgestellte Neuerungen sind unter anderem eine Erweiterung und Verbesserung der Auflösung des Sichtfeldes und Eye-Tracking als Zielverfolgung, die eine intuitivere Steuerung ermöglichen.

# Abbildungsverzeichnis

1.1	Semantische Erweiterung der Realität anhand eines Motorrads. . . . .	2
2.1	Anwendungsbeispiel der HoloLens bei Operationen. . . . .	6
2.2	Der Pre-Processing Workflow der Mixed-Reality-Anwendung von Chen et al. 2018. . . . .	7
3.1	Die Microsoft HoloLens (Microsoft 2018d) . . . . .	9
3.2	Sensoren und holographische Linsen der HoloLens. . . . .	10
3.3	Gaze Tracking und Air Tap. . . . .	13
3.4	Ausschnitt aus einem Spatial Mapping Mesh. . . . .	15
3.5	Das Spatial Mapping GameObject. . . . .	16
3.6	Eine Beispiel-Architektur der RefineNet-Pipeline. . . . .	17
3.7	Die interne Struktur eines RefineNet-Blocks. . . . .	18
3.8	Die Struktur von OBJ-Dateien anhand eines Beispiels. . . . .	20
4.1	Die hier genutzte Architektur mit ihren zwei Hauptkomponenten. . . . .	22
4.2	Ablauf der präsentierten Architektur. . . . .	23
4.3	Lokalisierungsverfahren eines Pixels im Bildkoordinatensystem der HoloLens. . . . .	30
4.4	RGB-Bild und das korrespondierende Rendering. . . . .	32
4.5	Ergebnis der Semantic Fusion basierend auf vier Segmentierungen. . . . .	34
5.1	Die Lampe und die beiden Stuhlarten, die segmentiert werden sollen. . . . .	38
5.2	Screenshots aus einem Anwendungsdurchlauf. . . . .	41
5.3	Screenshots vom Ende eines Anwendungsdurchlaufs. . . . .	43
5.4	Anwendungsdurchlauf mit fehlerhaftem Scan eines Stuhls. . . . .	44
5.5	Probleme der Segmentierung, wenn Stühle nur teilweise sichtbar sind. . . . .	46
5.6	Der von uns angewendete Lösungsansatz im Vergleich zum Original. . . . .	46
5.6	Die Zwischenschritte der Pipeline auf dem Server an 6 Beispielbildern dargestellt. . . . .	49
5.7	Das Ergebnis der Semantic Fusion nach jeweils 25 Bildern. . . . .	50
6.1	Die HoloLens 2 in der Seitenansicht. . . . .	52



# **Tabellenverzeichnis**

3.1	Die Sensoren der Microsoft Hololens (Microsoft 2018d)	10
5.1	Messdaten zur Laufzeitanalyse der präsentierten Anwendung.	40



# Literatur

- Bourke, Paul (2012). *OBJ-Dataformat*. URL: <http://paulbourke.net/dataformats/obj/> (besucht am 18. 03. 2019).
- Chen, Long, Wen Tang, Nigel W. John, Tao Ruan Wan und Jian Jun Zhang (2018). „Context-aware mixed reality: a framework for ubiquitous interaction“. In: *Corr abs/1803.05541*.
- Dawson-Haggerty, Michael (2017). *Trimesh*. URL: <https://github.com/mikedh/trimesh> (besucht am 17. 03. 2019).
- Evans, Gabriel, Jack Miller, Mariangely Iglesias Pena, Anastacia MacAllister und Eliot Winer (2017). „Evaluating the Microsoft HoloLens through an augmented reality assembly application“. In: *Degraded environments: sensing, processing, and display 2017*. Bd. 10197. International Society for Optics und Photonics, S. 101970V.
- Garon, M., P. Boulet, J. Doironz, L. Beaulieu und J. Lalonde (Sep. 2016). „Real-time high resolution 3D data on the HoloLens“. In: *2016 IEEE international symposium on mixed and augmented reality (ISMAR-Adjunct)*.
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren und Jian Sun (2016). „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 770–778.
- Hermans, Alexander, Georgios Floros und Bastian Leibe (2014). „Dense 3D semantic mapping of indoor scenes from RGB-D images“. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, S. 2631–2638.
- IKEA (2019). *IKEA ARSTID Tischleuchte*. URL: <https://www.ikea.com/de/de/p/arstid-tischleuchte-vernickelt-weiss-70280634/> (besucht am 27. 03. 2019).
- Jang, Jihye, Cory M Tschabrunn, Michael Barkagan, Elad Anter, Bjoern Menze und Reza Nezafat (2018). „Three-dimensional holographic visualization of high-resolution myocardial scar on HoloLens“. In: *Plos one* 13.10, e0205188.
- Kingma, Diederik P und Jimmy Ba (2014). „Adam: a method for stochastic optimization“. In: *Arxiv preprint arxiv:1412.6980*.
- Lin, Guosheng, Anton Milan, Chunhua Shen und Ian Reid (2017). „RefineNet: multi-path refinement networks for high-resolution semantic segmentation“. In: *IEEE conference on computer vision and pattern recognition (CVPR)*.
- McCormac, John, Ankur Handa, Andrew Davison und Stefan Leutenegger (2017). „SemanticFusion: dense 3D semantic mapping with convolutional neural networks“. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, S. 4628–4635.

## LITERATUR

- McCormac, John, Ankur Handa, Stefan Leutenegger und Andrew J Davison (2017). „SceneNet RGB-D: can 5M synthetic images beat generic imagenet pre-training on indoor segmentation?“ In: *Proceedings of the IEEE international conference on computer vision*, S. 2678–2687.
- Microsoft (2017a). *HoloLens tracking system*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/tracking-system> (besucht am 17.09.2018).
- (2017b). *MixedRealityToolkit for Unity*. URL: <https://github.com/Microsoft.MixedRealityToolkit-Unity> (besucht am 17.09.2018).
- (2018a). *HoloLens gaze*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/gaze> (besucht am 17.09.2018).
- (2018b). *HoloLens gestures*. URL: <https://support.microsoft.com/en-us/help/12644/hololens-use-gestures> (besucht am 03.09.2018).
- (2018c). *HoloLens gestures*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/gestures> (besucht am 17.09.2018).
- (2018d). *HoloLens hardware details*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/hololens-hardware-details> (besucht am 31.08.2018).
- (2018e). *HoloLens release notes*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/release-notes> (besucht am 17.09.2018).
- (2018f). *HoloLens spatial mapping*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-mapping> (besucht am 17.09.2018).
- (2018g). *HoloLens spatial sound*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-sound> (besucht am 17.09.2018).
- (2018h). *HoloLens voice input*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/voice-input> (besucht am 17.09.2018).
- (2019a). *HoloLens 2 device hardware*. URL: <https://www.microsoft.com/en-us/hololens/hardware> (besucht am 28.03.2019).
- (2019b). *HoloLens introduction videos*. URL: <https://www.youtube.com/watch?v=fZpWHNou7TI&t=24s> (besucht am 28.03.2019).
- (2019c). *Locatable camera*. URL: <https://docs.microsoft.com/en-us/windows/mixed-reality/locatable-camera> (besucht am 20.03.2019).
- Orsini, Alessandro, Gautam Venkatesan, Glen Huang, Grisam Shah und Niral Shah (2017). „Augmented reality enhanced cooking with Microsoft HoloLens“. In: *Rutgers, state university of new jersey*.
- Periyasamy, Arul Selvam, Max Schwarz und Sven Behnke (2018). „Robust 6D object pose estimation in cluttered scenes using semantic segmentation and pose regression networks“. In:
- Poothicottu Jacob, George (2018). „A HoloLens framework for augmented reality applications in breast cancer surgery“. Magisterarb. University of Waterloo.
- Pratt, Philip, Matthew Ives, Graham Lawton, Jonathan Simmons, Nasko Radev, Liana Spyropoulou und Dimitri Amirnas (2018). „Through the HoloLens™ looking glass: augmented reality for extremity reconstruction surgery using 3D

- vascular models with perforating vessels“. In: *European radiology experimental* 2.1, S. 2.
- Schwarz, Max, Christian Lenz, Germán Martín García, Seongyong Koo, Arul Selvam Periyasamy, Michael Schreiber und Sven Behnke (2018). „Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing“. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, S. 3347–3354.
- Strange, Adario (2018). *Verkaufszahlen HoloLens*. URL: <https://hololens.reality.news/news/we-may-finally-know-many-hololens-devices-microsoft-sold-its-revealing-peek-future-ar-0184481/> (besucht am 21.09.2018).
- Unity (2017). *Unity manual: gameobjects*. URL: <https://docs.unity3d.com/Manual/GameObjects.html> (besucht am 14.03.2019).
- (2018a). *Unity documentation: mesh*. URL: <https://docs.unity3d.com/ScriptReference/Mesh.html> (besucht am 23.03.2019).
  - (2018b). *Unity manual: prefabs*. URL: <https://docs.unity3d.com/Manual/Prefabs.html> (besucht am 14.03.2019).
- Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu und Kaiming He (2017). „Aggregated residual transformations for deep neural networks“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 1492–1500.