HAW HAMBURG

Bachelorarbeit

Gehui Xu

Aufzeichnung und Streaming von dynamischen 3D Gesichtsmodellen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Gehui Xu

Aufzeichnung und Streaming von dynamischen 3D Gesichtsmodellen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 22. Mai 2020

Gehui Xu

Thema der Arbeit

Aufzeichnung und Streaming von dynamischen 3D Gesichtsmodellen

Stichworte

Gesichtslokalisation, Gesichtsrekonstruktion, Streaming, Augmented reality

Kurzzusammenfassung

Kurze Ausblick in die Zukunft der Videokonferenz ...

Gehui Xu

Title of Thesis

Record and streaming of a dynamic 3D facial models

Keywords

Facial localization, facial reconstruction, streaming, augmented reality

Abstract

Short view into the future of videoconferencing ...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagen

Es gibt natürlich unzählige Menschen, die maßgeblich dazu beitragen haben, dass diese Arbeit in vorliegenden Art realisiert werden konnte. Ganz besonderes möchte ich mich bei meinem Betreuender Prüfer Herrn Prof. Dr. Philipp Jenke bedanken, die Bereitschaft und Betreuung meiner Arbeit zu übernehmen und die sich stets bereit erklärt haben, mir in allen Fragen hilfreich zur Seite zu stehen. Ein ganz besonderer Dank geht auch an Herrn Prof. Dr. Stefan Sarstedt, die Bereitschaft und Prüfung meiner Arbeit zu die übernehmen. Ein großer Dank geht an Mein Vater Wenhua und Meine Schwester Tingyan die das Korrekturlesen übernommen haben. Neben der fachlichen und organisatorischen Unterstützung gibt es auch eine Menge von Helfern, die im Hintergrund zum Gelingen beigetragen haben. All diesen gilt hier mein Dank. Ganz besonders bedanken möchte ich mich bei meiner Mutter Shanhong und meinen Kinder Maximillian und Anton. Sie bildeten ein geschlossenes Team, das immer für die nötige Rückendeckung und die Verfolgung des Fehlerteufels sorgte.

1 Einleitung

In den Zeiten der Globalisierung wird es vermehrt notwendig, Geschäftsreisen zu unternehmen, um mit den Geschäftspartnern in unterschiedlichen Ländern zu kommunizieren. Damit steigert sich auch die Flugverkehr und belastet die Umwelt. Das Reisen ist nicht nur belastend für die Natur, sondern auch für die Menschen, die diese Reisen unternehmen. Die Distanz in der Welt wird durch das immer schneller werdende Internet kleiner. Mit der digitalen Vernetzung ist die Kommunikation über Ländern hinweg mit Hilfe von Telefon- und Videokonferenzen schneller, einfacher und günstiger geworden. Die Unternehmen benutzen diese Möglichkeiten der Kommunikation, soweit es möglich ist. Eine Dienstreise wird dennoch oft unternommen, da dies Kundennähe mit sich bringt.

Durch die neueren Entwicklungen in der Technik, vor allem die Virtuelle Realität(VR), ist eine neue Art der Kommunikation möglich geworden: Die Menschen können in eine virtuelle Welt eintauchen und sich online "persönlich"treffen und miteinander interagieren.

Die Augmented Reality (AR), eine Unterart der VR, ermöglicht die Projektion von Menschen und Objekten in die reale Welt. Auch wenn es nicht den körperlichen Kontakt ersetzen kann, ermöglicht sie tiefer gehende Interaktionsmöglichkeiten. Dreidimensionale Kameras fangen die Realität inklusive Tiefen-Informationen ein, die dann dem digitalen Gegenüber übertragen werden. Entwicklungen wie die der Kinect Kameras, VR-Kameras von Microsoft für verschiedenen XBOX Generationen, sind damit wichtige Grundbausteine für 3D-Videotelefonie.

1.1 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll ein Konzept erstellt werden, das klärt, welche Faktoren erfüllt sein müssen, damit ein dreidimensionales Gesichtsmodell auf Basis von polygonalen Netzen, Bewegungserkennung und VR-Technik realisiert werden kann.

Diese 3D Modelle sollen für die AR-Videoübertragung entwickeln werden. Mit diesen soll es möglich sein, per AR-Video Konferenzen zu schalten. Bei den Teilnehmern soll das Gefühl geweckt werden, mit einer anderen Person am Tisch zu sitzen, ohne dass der gegenüber in der realen Welt anwesend ist.

Im Detail sollen hier für ein Bedienungsschema, das auf Gestensteuerung basiert, graphische Konzepte, die benötigten Systemfunktionen, die zwingend für den Betrieb erforderlich sind, und mögliche Optimierungsmöglichkeiten spezifiziert und entworfen werden.

In dieser Arbeit entwickle ich einen Ansatz der AR-Videoübertragung, um die Grundlagen für Konferenzen mittels AR-Video zu schaffen. Ziel ist es, das Gesicht einer Person in AR live zu streamen und so eine persönliche Nähe vorzutäuschen.

Im Anschluss daran soll geprüft werden, in wie weit sich dieses Konzept mit den momentan verfügbaren technischen Möglichkeiten realisieren lässt. Hierbei soll ein Prototyp implementiert werden, der eine Teilfunktionalität aus dem erstellten Konzept besitzt.

1.2 Aufbau der Bachelorarbeit

Der Kern dieser Arbeit ist die Erkennung und Freistellen sowie das Versenden der Bilddaten von Gesichtern an einem AR-fähigen Endgerät.

Im Kapitel SStand der Technik"werden die die Entwicklungen im Bereich Gesichtslokalisation, 3D-Kameras, Internetprotokollen und AR-Telefonie dargestellt und der Ansatz von dieser Arbeit eingeordnet.

Der Kapitel "Grundlagen"führt in AR ein und stellt die entsprechend benötigten computertomographischen Grundlagen vor.

Im Kapitel Ëntwürfen"wird dann der im Kapitel SStand der Technikërwähnten Ansatz für diese Arbeit näher erläutert.

Im Kapitel "3D Gesicht Rekonstruktion"werden die benötigte Hard- und Software aufgelistet. Hier werden auch die begegneten Problemen und deren Lösungwege sowie geschlossene, notwendige Kompromissen erläutert.

Im Kapitel Ëvaluation"wird dann das im Rahmen dieser Arbeit entwickelte Programm getestet und die Resultate ausgewertet.

Abschließend gibt es im letzten Kapitel Fazit und Schluss eine Zusammenfassung der Arbeit und einen Ausblick in die Zukunft, sowie mögliche Erweiterungen für das Projekt.

2 Der Stand der Technik

In diesem Kapitel wird die Hintergrundinformationen und wissenschaftlichen arbeiten über das Thema Aufzeichnung und Streaming von dynamischen 3D Gesichtsmodellen vorgestellt. Besonderen Focus wird auf Gesichtserkennung und 3D-Modellen gelegt.

2.1 Gesichtslokalisation

Der Begriff Gesichtslokalisation oder Gesichtsfindung ist nicht zu verwechseln mit der Gesichtserkennung. Bei dem zweiteren Begriff geht es darum, Gesichter von einzelnen vorgegebenen Personen zuzuordnen, während es bei dem ersteren darum geht, in einem Bild den Bereich zu finden, welches ein Gesicht darstellt.

Seit der 80er Jahren gibt es Forschungen zur computergestützten Gesichtslokalisation in Bildern[10]. Seit 1990 wurde die Interesse in diesem Bereich mit der Entwicklung der digitalen Bildverarbeitung immer stärker. Bis zur heutiger Zeit wurde eine Reihe von Verfahren entwickelt, um dies zu bewerkstelligen.

Im Rahmen meiner Arbeit verwende ich Gesichtslokalisation in einem laufenden Video-Stream.

Für Menschen ist das Erkennen von Gesichtern eine einfache Angelegenheit. Für computergestützte Gesichtslokalisation ist es jedoch eine Herausforderung. In den als Pixeln vorliegenden Bilddaten das Muster eines Gesichts zu finden, wird noch durch z.B. unterschiedliche Beleuchtungen, Drehungen oder Winkel des aufgezeichneten Gesichts, oder aber auch Einstellungen und Eigenschaften, wie der Winkel des Objektives von dem Kamera erschwert. Es wurden eine Reihe von Gesichtslokalisationsalgorithmen entwickelt, um diese Herausforderungen zu meistern.

Hjelmas und co. unterteilten ihren Arbeit [5] die Gesichtsfindungsmethoden in verschiedenen Kategorien: merkmalsbasiert und bildbasiert. Diese zwei Klassen werden unterschieden dadurch, wie das Wissen übers Aussehen von Gesichtern gespeichert und benutzt wird.

- Merkmalsbasierte Verfahren verwenden die Charakteristika von Gesichtern, um sie in Bildern zu identifizieren. Dabei wird das Wissen über die Merkmale des menschlichen Gesichts durch den Programmierer oder Anwender direkt in das Verfahren eingebracht.
- Bildbasierte Verfahren dagegen behandeln die Gesichtsfindung als allgemeines Mustererkennungsproblem. Dabei lernt das Verfahren selbst, was ein Gesicht ist und was keins ist.

In der Ausarbeitung von Schmalstieg und Co. wurde eine Methode entwickelt, der verschiedenen Muster, features genannt, verwendet um Teile des zuerkennenden Bildes zu verdecken. Es wird vielen Richtigen und Falschen Beispiel benutzt um das Algorithmus zu trainieren. [11] Diese Algorithmus wurde dann von Leinhart und co. um weiteren features erweitert.[6] Die Algorithmus kann für frontalen Gesichter oder Seitenansicht trainiert werden, aber nicht für beides gleichzeitig.

Mit den Entwicklungen der Neuronalen Netzwerken wurde von Liu und co. ein Single Shout MultiBox Detector effizient und Akkurat erkennen[7] entwickelt. Dazu funktioniert sie auch für Gesichter aller Orientierungen und Größen.

Mein Ansatz ist es, ein Videokonferenz in einem Konferenzraum zu halten, wobei nicht alle Teilnehmer unbedingt vor Ort sein müssen. Dazu wurde jedoch noch keine Publikationen und Arbeiten veröffentlicht. Also habe ich meine Arbeit in meheren Teile unterteilt und Literatur dazu untersucht. Die Teile sind jeweils:

- Bearbeitung des Gesichts
- Übertragung von 3D Modelldaten und
- AR-Darstellung.

2.2 Gesichtsrekonstruktion

Mit Gesichtsrekonstruktion wird hier das Erstellen von dreidimensionalen Gesichts-Modellen der realen Gesicht-Objekten bezeichnet. Es können auch ganze köper einschließlich die in Kamera Sichtfeld befundene Objekt und Strukturen rekonstruiert werden. Die 3D-Gesichtsrekonstruktion ist ein Teil der Bildverarbeitung. In den letzten Jahrzehnten Forschungen in diesem Bereichen sehr weiter Entwickelt.

Der wesentliche Unterschied zum üblichen Erstellen von 3D-Modellen wie z.B in CAD1-Anwendungen liegt im erheblich geringeren Arbeitsaufwand. Die Punkte werden von einem Sensor registriert, um so halbautomatisch ein digitales Abbild eines Objekts oder einer Szenerie zu erstellen. Während eine 2D-Fotografie nur eine einzelne Perspektive eines dreidimensionalen Objektes zeigt, ist für eine vollständiges 3D Rekonstruktion das Registrieren von Punkten aus mehreren Perspektiven nötig (siehe Abbildung 2.1). Das Zusammenführen der Daten aus mehreren Bildern ist das Hauptproblem (siehe Kapitel 3.2) der 3D Rekonstruktion, welches Algorithmen wie KinectFusion (siehe Abschnitt 3.1) zu lösen versuchen.

2.3 3D-Kameras

Für die 3D-Rekontstruktion ist eine Kalibrierung der 3D-Kamera erforderlich. 3D-Kameratechniken werden in 5 verschiedenen Arten unterteilt:

- Das Stereokamerasystem ist den menschlichen Augen nach empfunden. Es wird zwei normale Kameras verwendet und täuscht dem Betrachter ein 3D-Bild vor, indem jedem Auge ein Bild einer der beiden Kameras angezeigt wird.
- Das Triangulationssystem benutzt eine Lichtquelle, die ein vordefiniertes Muster wirft, und eine Kamera mit einem versetzten Position nimmt ein Bild mit dem geworfenen Muster auf und berechnet die Verzerrung.
- TOF(Time of flight)-Kamera berechnet die Tiefe über die Laufzeit des Lichtstrahls.
- Interferometrie arbeitet mit dem Interferenz eines Mess- und Objektstrahls, um die Tiefe zu berechnen.

- Bei Lichtfeldkameras werden durch das Verwenden von Mikrolinsenarray die Helligkeit und Lichtrichtung der Strahlen im Bildpunkt aufgenommen. So kann das Fokus eines Bildes nach dem Aufzeichnen noch nachträglich verändert werden.

In meiner Arbeit wird die Triangulationskamera der Kinect Version X-BOX360 benutzt. Dabei ist der Fischaugeneffekt der Linsen zu beachten. Der Fischaugeneffekt wurde schon sehr früh entdeckt und die Fischaugenlinsen kam daher. Im Buch von Thomas Huang und Armin Grün wurden verschieden Algorithmen für das Kalibrierungsproblem in Computervision dargestellt.[4]

2.4 AR in der Telefonie

Es gibt verschiedenen Richtungen in der Entwicklung der AR-Telefonie. Die ursprüngliche Idee war eine Studierstube, in der mehreren Nutzern gleichzeitig eine stereoskopische Grafik gezeigt wird. Diese Nutzer sind hierbei im selben Raum und arbeiten zusammen miteinander an einem virtuellen Objekt. [9]

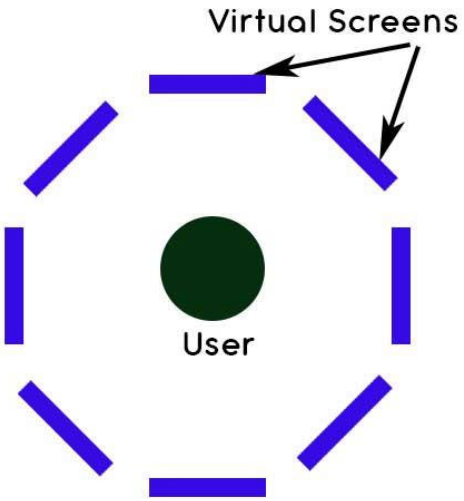

Abbildung 2.1: Aufbaubeispiel AR-Telefonkonferenz

Eine Richtung ist die Nutzung von Smartphonegeräten als AR-Geräte. über die werden die Videoübertragung der Teilnehmer einer Telefonkonferenzen im Kreis um den Nutzer im realen Raum platziert[8]. Abbildung 2.1 zeigt den Aufbau der Platzierung. Diese Aufstellung wurde von Indian Institute of Technology Madras, Chennai, in ihrer Veröffent-

lichung verwendet. Allerdings sind die virtuelle Screens nur schwebende Video-Streams und keine 3D-Dastellung der Personen.

Eine andere Richtung hat Schmalstieg und Co. in ihrer Arbeit untersucht. Sie wollen ein Tool für herkömmlichen Videokonferenzen entwickeln. Dabei sitzen die Teilnehmen immer noch vor ihren Computern, allerdings können sie mit Markern vorher definierte Objekten den Personen am anderen Ende der Videoübertragung zeigen[2].

Eine weitere Richtung wurde in der Veröffentlichung von Mark Billinghurst und Hirokazu Kato eingeschlagen[3]. Sie schlugen vor, mittels Markern die Videokonferenzteilnehmern jeweils in die reale Welt zu projizieren. Dabei werden die Portaits, sofern benötigt, freigestellt. Die Abbildung 2.2 zeigt eine mögliche Aufbau.

Abbildung 2.2: Beispielbild

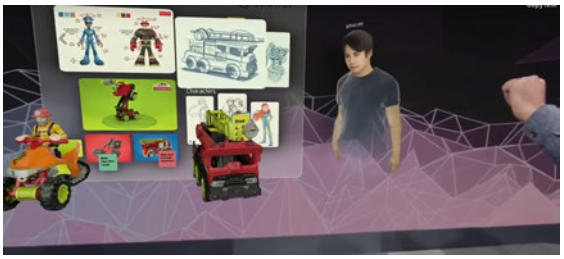

Abbildung 2.3

Eine letzte Richtung hatte die Firma Spatial Systems Inc. eingeschlagen. Sie entwickelten ein Arbeitsumgebung, damit Personen von verschiedenen Orten zusammenarbeiten können.[1] In Abbildung 2.3 wird eine Demo gezeigt.

Unser Ziel dieser Arbeit ist es, dass Personen von verschiedenen Orten aus mit Hilfe der AR verbunden und in einem virtuellen Raum zusammengeführt werden sollen. Damit liegt der Focus hierbei auf die Darstellung der projizierten Avatare in dem Videokonferenz. Die Avatare sollen also möglichst nah an der Realität sein und auch als 3-dimensional dargestellt werden. Dieser Ansatz ähnelt dem von Mark Billinghurst und Hirokau Kato, jedoch mit 3D Darstellungen.

3 Grundlagen

In diesem Kapitel werden die Grundlagen der Einsatzumgebung des Projektes und die grundlegenden technischen Begriffe erklärt. Neben der genutzten Hard- und Software sind die grundlegenden Algorithmen der 3D Rekonstruktion Teil dieser Ausführung.

3.1 Software Komponent

Für die Verarbeitung sowie Darstellung von 3d-Gesicht-Model gibt es diverse Programme. Die für diese Arbeit genutzte Software soll in diesem Abschnitt kurz vorgestellt werden.

Um in der Arbeit Kinect Kamera nutzen zu können, müssen vorher einige Bibliotheken (opencv, pthreads, libusb) installiert werden. Als Treiber für den Kinect Sensor benötigt libfreenect. Für das Installieren der openCV sowie zuvor beschriebenen Komponenten ist zudem CMake nötig, sowie ein Compiler, hier Microsofts Visual Studio. Der Installationsablauf unter Windows 10 System ist im Anhang ausführlich beschrieben.

Für die 3D Gesichter Rekonstruktion mit der Microsoft Kinect V1 in dem Arbeit wird folgenden Software Komponenten verwandet:

1. **opencv** OpenCV (Open Source Computer Vision Library) ist eine Open Source-Bibliothek für Computer Vision. Es wurde für die Zwecke, Computer Vision, Algorithmus, mathematische Operationen, Videoerfassung, Bildverarbeitung in der Arbeiten verwandet. Die Bibliothek enthält mehr als 2500 optimierte Algorithmen, die eine hervorragende Genauigkeit in Bezug auf Leistung und Geschwindigkeit bieten. Diese Algorithmen können verwendet werden, um Gesichter zu erkennen und zu erkennen, Objekte zu identifizieren. Das hat für das Arbeit hervorgehobene Bedeutung.

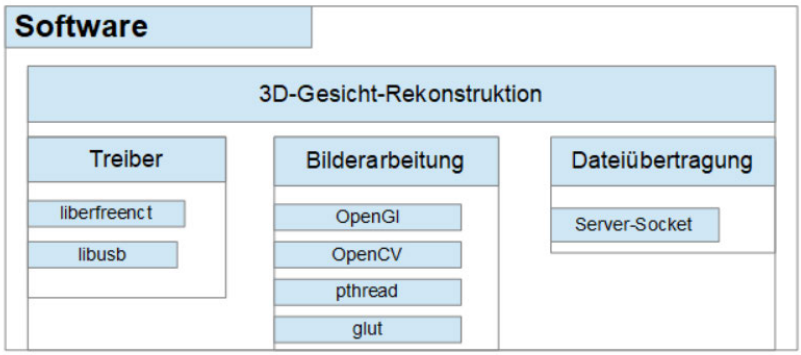

Abbildung 3.1: Software für 3D Gesicht Model

2. **OpenGL** ist eine plattformunabhängige Grafikbibliothek, die inzwischen zu einem weltweiten Standard für 3D-Grafikprogrammierung geworden. Es werden grundlegende geometrische Objekte bereitgestellt wie Rect, ellipse usw. Teilweise wird OpenGl direkt von Graphikkarten unterstützt, dann kann es natürlich schnell Bilden verarbeiten.

3. **Glut(OpenGL Utility Toolkit)** ist eine weitere Zusatzbibliothek von OpenGL. um die Interaktion behandelt. Sie stellt Funktionen zum Aufbau von Fenstern und zum Behandeln von Tastatur- und Mausereignissen zur Verfügung.

3.2 3D Modell

Ein 3D-Modell ist ein virtuelles Modell, das ein Objekt in Planung oder in der realen welt nachstellt. Sie wird mit Hilfe von einem oder mehreren Dreiecksnetzen dargestellt. Die Abblidung 3.2 stellt ein Delphin dar.

Hier wird die Grundlagen von 3D Modell und Begriffe erläutert.

3.2.1 Dreiecksnetze

Ein Dreiecksnetz besteht aus Punkten (engl. vertex), Kanten (engl. edge) und Flächen (engl. face). (Im Folgenden werden jeweils die englischen Begriffe dieser Datenstrukturen genutzt.) Dreiecksnetze können in unterschiedlichen Datenstrukturen gespeichert werden.

Jenach Anforderung können sie die drei vorher genannten Elementen oder noch weitere Strukturen verwenden. Drei Vertices verbunden mit Edges ergeben ein Face, und mehrere Faces verbunden ergeben ein Netz.

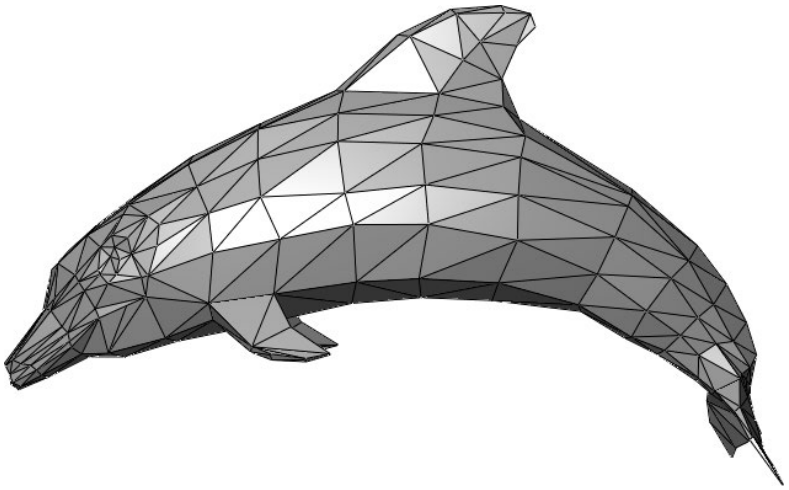

Abbildung 3.2: Ein Beispiel eines Dreiecksnetzes, das einen Delphin darstellt

3.2.2 Texture-Mapping

Man kann bei der Speicherung eines Dreiecknetzes die Farben der Faces mit abspeichern. Dies wird jedoch selten gemacht, da nur eine Farbe pro Face gespeichert werden kann. Falls aber mehr Farben oder Abbildungen benötigt werden, so kann eine Texture Map zum Einsatz kommen. Texture Mapping ist das Aufbringen von 2D-Texturen auf 3D-Oberflächen. Dabei können Tiefen und Höhen vorgetäuscht werden.

3.3 Hardware

In folgenden werden die technischen Grundlagen und aktuellen Voraussetzungen erklärt.

3.3.1 Augemented Reality

Augmented Reality (AR) ist ein Unterbegriff der Mixed Reality (MR). Mixed Reality ist Englisch von vermischte Realität. Hier wird die reale Welt, so wie wir sie ohne (digitale)

Hilfsmittel wahrnehmen, mit alternativen Realitäten vermischt. Alternativen Realitäten existieren z.B. in Büchern, Filme und auch Spiele, in die man für einige Zeit der realen Welt entfliehen kann. Durch die Technik in der Informatik kann man mit Hilfe von Headmounted-Displays (HMD) den User mit einer virtuellen Welt umgeben.

Heutzutage bezeichnet MR meist die Vermischung der reallen Welt mit der virtuellen, digital erzeugten Realität. In Abbildung 3.3 ist ein Graph zu den verschiedenen Realitäten. Unter Mixed Reality findet man in dieser Differenzierung neben der realen Welt die erweiterte Realität (AR), die erweiterten Virtualität (AV, augmented Virtuality) sowie Virtuelle Realität (VR). Die Virtualisierung schreitet von Links nach Rechts fort und man geht von einigen virtuellen Gegenständen in AR zu komplett virtuelle Umgebung in VR. Die Übergang zwischen AR und AV ist fließend und nicht eindeutig definierbar.

Abbildung 3.3: Mixed Reality chart, übersetzt übernommen aus Milgram, Takemure und Co.

3.3.2 AR-Geschichte

Die Entwicklung der AR Geräte beginnt sehr früh. Bereits im Jahr 1968 baute Ivan Sutherland und Thomsd A. Furness III mit dem Massachusetts Institute of Technology(MIT) den ersten HMD mit dem Namen Sword of Damocles, den Urvater der AR-Geräte. Das erste HMD war noch recht sperrig und musste auf einem Gerüst montiert werden (vgl. Abbildung 3.4).

Dieses erste AR-Gerät bestimmte auch in den folgenden Jahren die Entwicklungsrichtung der AR-Geräten. Das erste AR-Game war Quake und wurde von Id-Softwares im Jahr 2000 entwickelt. Dabei wurde ein HMD und ein Rucksack mit einem Laptop benutzt, wie in Abbildung 3.5 gezeigt wird.

Im Jahr 2005 wurde dann der erste AR-App von Nokia entwickelt, es war ein Tennisspiel für zwei Personen. Um Sie zu testen wurde es in einem Museum zum spielen freigegeben, die Resonanz war Es wurde in einem Museum getestet mit einem positiven Resonanz. Im Jahr 2012 wurde dann der erste Cloud-based App enwickelt. Im gleichen Jahr wurde

Abbildung 3.4: First AR-Equipment

das erste AR-Glass von Google entwickelt. Sie war kleiner und leichter als seine Vorgängern. Gleichzeitig hatte Google und Niantic ein AR-GPS-Spiel mit dem Namen Ingress für Smartphones herausgegeben, das große Popularität erlangte. Das Spiel zeigte dem Nutzern über dem Smartphone Spielinfos in einer an der realen Welt angelehnten Karte an. Mit den Erfahrungen von Ingress entwickelten Nintendo und Niantic im Jahr 2016 dann das Spiel Pokemon Go und später in 2019, entwickelten Niantic in Zusammenarbeit mit WB Games San Francisco Harry Potter: Wizards Unite.

Die AR-Endgeräten entwickelten sich in der Zeit in drei Richtungen:

- Bei der ersten Richtung werden Smartphones und Tablets verwendet, ob als Hand held Device oder mit Hilfe von Vorrichtungen am Kopf vor den Augen befestigt.
- Die zweite Richtung ist die der HMD. Inzwischen gibt es auch eine große Varietät bei den HMD. Beispiel wäre der HoloLens von Microsoft, der Google Glass von Google und der Magic Leap.
- Die Dritte Richtung ist eine feste Installation in den Einkaufsläden und an den Messen, die so genannten AR-Kiosks.

Abbildung 3.5

3.3.3 Kinect

Microsoft hatte die Kinect das erste Mal im Jahr 2009 auf einer Spielmesse vorgestellt. Sie war für die damalige Zeit eine technologische Sensation. Die Fähigkeit von Kinect, räumlich zusehen, war sehr interessant und schon bald entdecken anderen Bereichen der Informatik diese für sich. Mehr und mehr wurde der Kinect vom Spielcontroller zum kostengünstigen 3D-Scanner für Informatikern, Technikern und Forschern. Doch die Kinect lieferte nicht was sie versprach und die Verkaufszahlen sanken rapide ab und in Jahr 2017 die Produktion von Microsoft eingestellt und nur noch die Rest bestände wurden verkauft.

Es gab zwei Versionen von Kinects: die Kinect Xbox 360 und Kinect Xbox One. Sie haben grundsätzlich unterschiedlichen Funktionsweise. Währen die Kinect Xbox 360 eine Triangulations-Kamera ist, funktioniert die Kinect Xbox One als TOF Kamera. Die Abb. 3.6 zeigt, wie die Kinect Xbox 360 aufgebaut ist.

Das spezielle Triangulationsverfahren der Kinect 1 wird Structured-Light Prinzip genannt. Diese funktioniert wie folgt: Ein Projektor projiziert ein (bei Kinect 1 infra-rotes) Muster auf den zu erfassenden Bereich, und die Kamera erfasst das veränderte Muster. Dabei muss der Abstand zwischen Projektor und Kamera bekannt sein. Das originalen

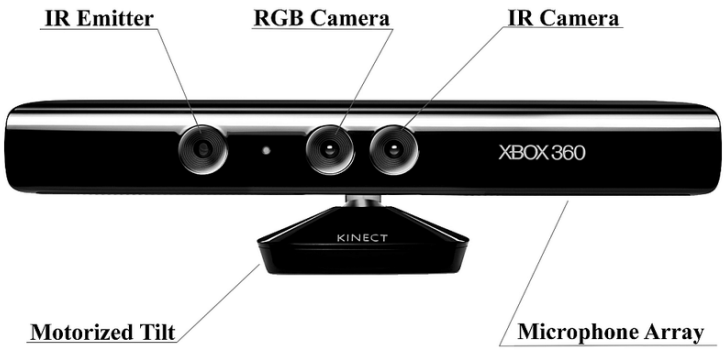

Abbildung 3.6: Die Aufbau der Kinect Xbox 360

Muster vom Projektor ist dem System bekannt. So kann durch das veränderte aufgenommene Muster ein 3D-Bild abgeleitet werden. Das Muster wird in Blöcken abgearbeitet. Dabei werden die Abständen der einzelnen Punkten zu ihren Nachbarn berechnet. Problem hierbei ist, bei zu kleinen Flächen können nicht ausreichend unterschiedliche Punkte berechnet werden. Die Abb. zeigt das Muster eines solchen IR-Patterns.

Um das Kinect Camera effektiv zu arbeiten muss folgende Punkte beachten: Vermeiden Sie es, den Sensor in direktem Sonnenlicht oder näher als 30 cm an Große Gegenstand aufzustellen. Platzieren Sie Kinect so nahe wie möglich an der Kante einer flachen, stabilen Aufstellfläche. Stellen Sie sicher, dass keine anderen Objekte in der Nähe das Sichtfeld blockieren. Dabei sollte der Sensor in einer Höhe von 0,6 m bis 1,8 m über dem Boden positioniert werden. Je höher, desto besser.

Die Kinect ist ein Sensorsysteme, welche abgesehen von einer RGB-Farbkamera noch weitere Depth-Sensoren nutzen um die Tiefeninformationen des aufgenommenen Bildes zu registrieren. In dem Arbeit wird Kinect V1 Kamera benutzt, das folgenden Eigenschaft besetzt.

- Auflösung RGB Kamera 640 x 480
- Auflösung Tiefenkamera 320 x 240
- Framerate 30 fps
- Tiefenerkennung mit Structured Light

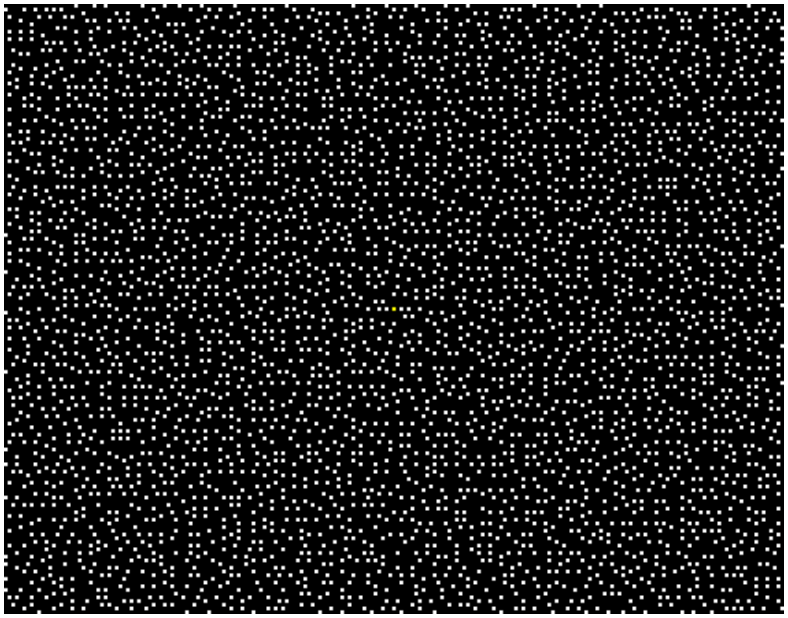

Abbildung 3.7: IR Pattern zu Entfernungsmessung

- Maximale Tiefe 4,5 m
- Minimale Tiefe 40 cm
- Horizontales Sichtfeld 57
- Vertikales Sichtfeld 43
- SDK Kinect für Windows 1.8
- PCL Unterstützung Ja

4 Entwurfen

Vorrangiges Ziel dieses Abschnitts ist, das effiziente Verfahren für 3d-Gesicht-Rekonstruktion aus Depth-Image und RGB-Image zu entwickeln. Die Erzeugung von dreidimensionalen Gesichtsmodel des zu betrachte Person lässt es sich mehrere Herausforderung in der Arbeiten zu leisten. Hier kann man in drei zusammenhängende Teilen zusammen gefasst werden.

4.1 Hardawre Komponent

Als Hardwaren in der Arbeit wird ausser der Computer das XBox 360 Kamerasystem und Endgerate -einen Android Smartphone- benutzt.

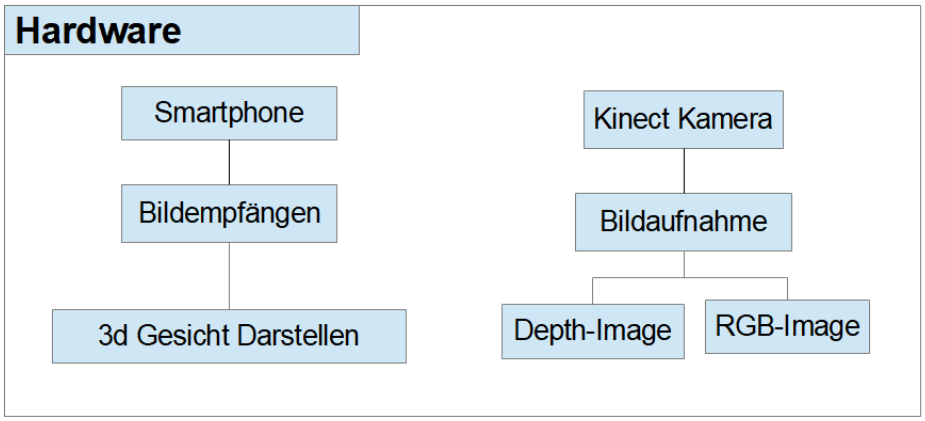

Abbildung 4.1: Hardwarekomponent

1. Die in der Arbeit zur 3D Rekonstruktion verwandte Hardwaren ist Microsoft Kinect V1 - XBox 360-. Das System werden als Farbbildaufnahme und Tieferkennung bedient.

2. Das Smartphone, das in der Arbeit benutzt hat, wird hier als Endgrat verwandet, um die 3D-Gesicht-Model zu darstellen.

4.1.1 Kinect v1 Kamera Positionieren

Der Abstand zwischen Gesichter von dem Testperson und Kamera muss eventuell etwas angepasst wird, damit das Scannen später einfacher abläuft und es seltener zum unvollständige Bild kommt. Wann der Abstand zwischen Sensor und Objekten kleiner oder großer als die Nahe Einstellgrenze der Kamera ist, wird es unschärfer bis unmöglich den Gesichter zu rekonstruieren. Scheint die Sonne, sollten die Fenster verdunkelt werden. Wann man Brille tragt, soll die Fassung heile Farbe und Gläser mögliche entspiegelt sein. Um das Kinect Camera effektiv zu arbeiten muss folgende Punkte beachten:

- Vermeiden Sie es, den Sensor in direktem Sonnenlicht oder näher als 30 cm an Große Gegenstand aufzustellen.
- Platzieren Sie Kinect so nahe wie möglich an der Kante einer flachen, stabilen Aufstellfläche. Stellen Sie sicher, dass keine anderen Objekte in der Nähe das Sichtfeld blockieren.
- Dabei sollte der Sensor in einer Höhe von 0,6 m bis 1,8 m über dem Boden positioniert werden. Je höher, desto besser.
- Raumbeleuchtung darf nicht zu dunkel auch nicht zu Heiler sein.

4.1.2 Treiber Software für Kinect V1 Verbinden

Die reine Hardware Verbindung zwischen Computer und Kamera ist durch USB relative einfach realisiert. Für die Datei Übertragung z.B. Kamerasteuerung und Bilddatei-Übertragung sind in der Arbeit libfreenect und libusb benutzt.

1. **libfreenect** ist ein Treiber Bibliothek für Microsoft Kinect. Es kann auf Linux, OSX und Windows laufen und supports für RGB und Depth Bilden sowie Motors usw..
2. **libusb** ist eine in C geschriebene Programmbibliothek, um Schreiben und Lesen von USB-Geräten (hier XBOX360) zu bedienen..

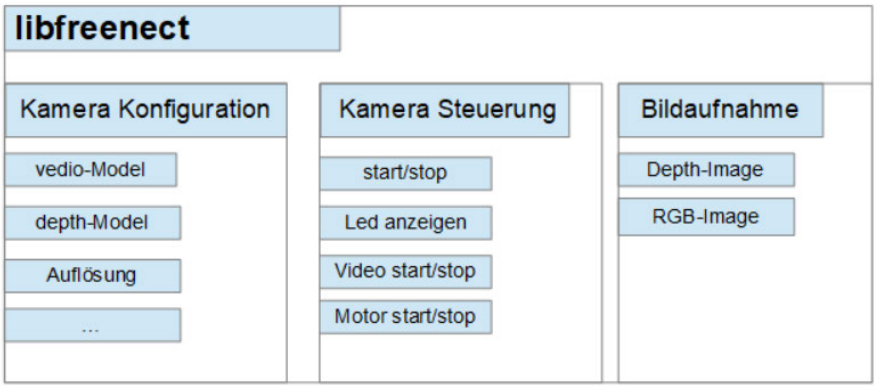

Abbildung 4.2: Software für Kamera Steuerung

4.2 3D Gesicht Rekonstruktion mit der Kinect Kamera

Grundlegend lassen sich in der Arbeit einen Methoden zur Rekonstruktion benutzen,dass es einen 3D-Modell mit dem von Kamera-System erfasste Tiefen- und Farb- Bildinformationen erzeugen kann. Mit3D Gesicht Rekonstruktion wird das Erstellen von dreidimensionalen Gesicht-Modellen aus realer Gesichter bezeichnet. Es können auch ganze Gesicht einschließlich auf ihrer enthaltenen Objekten (Brille usw.) und Strukturen rekonstruiert werden.

4.2.1 Daten Gewinnung

Mit dem Kinect Kamera kann es Bilddatei in zwei unterschiedlich Formate gewonnen werden.

1. In diesem Arbeit wird Tieferkennung von betrachte Gesicht durch Infrarot-Tiefenkamera von Kinect V1 realisiert. Die Tiefen-bilden des kompletten Sichtfeldes wird mit Hilfe Momentaufnahme des Kameras ausgeliefert. In folgenden wird als **Depth-Image** bezeichnet, die zur 3D Rekonstruktion genutzt werden können.

2. **RGB-Image** des Gesichtes wird RGB-Kamera von Kinect V1 auf optischem Wege einen RGB-Farbenbild des Kamerasichtfeldes erfasst. Es wird in späte für Gesicht-localisationsverfahren und 3D-Gesicht Model Kalibrieren benutzt.

4.2.2 Gesicht Erkennung

Mit Hilfe von OpenCV mitgelieferte DNN Module kann die menschliche Gesicht in dem Bilder erkannt werden. Je nach Methode lässt sich in einem oder mehreren Schritten die Lage der sichtbaren Gesichter im Koordinatensystem des Kameras, die Position und Große des Gesichts im Bild sind, ermitteln.

Um die Hintergrund und Vordergrund abzutrennen wird in dem Arbeit die Rang-Filter-Funktion auf dem Tiefenbilder bearbeitet. Mit Hilfe von die Position und Größe des Gesichts in Tiefenbildern in voriger Phase kann die Kopfbilder aus dem Tiefenbildern ausgeschnitten werden.

4.2.3 3D Positionsdatei Berechnen

Das Hauptproblem der 3D-Gesicht-Rekonstruktion in dem Arbeiten ist das Zusammenführen der erfassten Depth- und RGB-Image Informationen. Mit dem in der Arbeit entwickelte Funktion computePoint() kann man dann aus diesem Kopfbilder einen Punktwolke berechnen.

Diese Tiefenbild (Depth image) werden mittels eines infrarot Entfernungsmessgerätes die Depth-Kamerasichtfeldes ermittelt. Jeden Pixel in Depth-Image Zeigt die "Tiefe"des Pixels an.

Abbildung 4.3: View der Depth Image

Ein Tiefenbild kann i.d.R. als Graustufenbild dargestellt werden. Je heller ein Pixel, umso näher das jeweilige Hindernis. Mit einer transformierte Berechnung kann man die Position und Große des Gesichts, die bei Gesichtserkennung ermittelt, in Tiefenbildern wieder zu finden. Mit dem Pixel Wert *depth* in Tiefbildern des es kann man dann alle Abstand z (in Meter) zwischen Kamera und alle in Depth-Kamerasichtfeld befundene Gesichtspunktposition berechnen.

$$z = \frac{1}{((depth \times (-1) \times 0.0030711016) + 3.3309495161)} \tag{4.1}$$

Die Räum x, y Position des Gegenständ in der Kamera Koordinatensystem können dann mit i, j Pixel-Position Depth-Image berechnet werden.

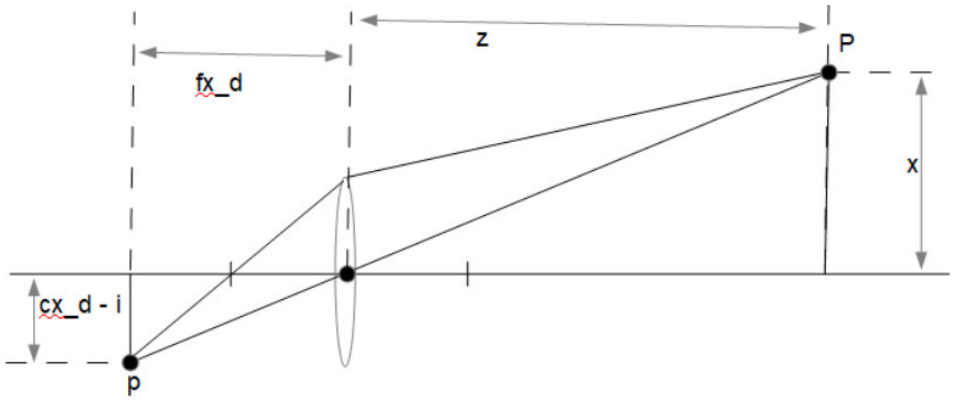

Abbildung 4.4: View Bildgewinnung

Folgend Skizze hat das einfaches physikalisches Modell der Bildgewinnung dargesttlt. Alle von einem Punkt P auf die Linse auftreffende Lichtstrahlen werden in der Hauptebene der Linse so gebrochen, dass sie in einem Punkt p der Bildebene zusammenlaufen, welche sich im Abstand fx_d zur Hauptebene befindet. Der Betrag der Strecke fx_d wird als Bildweite bezeichnet. Der Abstand zwischen P und der Hauptebene ist die Gegenstandsweite z. Der Abstand zwischen der optischen Achse (in der Grafik als o.A. abgekurzt) und P wird als Gegenstandsshöhe x bezeichnet. Als Bildhöhe $i - cx\_d$ bezeichnet man den Abstand zwischen dem Bild des Punktes p und der optischen Achse. Aus dem Strahlensatz ergibt sich die Abbildungsgleichung:

$$\frac{x}{z} = \frac{i - cx\_d}{fx\_d} \tag{4.2}$$

Mit der Abbildungsgleichung kann man dann Umformen wie folgende Gleichung, um P Position x, y in Raum berechnen.

$$x = \frac{((i - cx\_d) \times z}{fx\_d} \tag{4.3}$$

$$y = \frac{((j - cy\_d) \times z}{fy\_d} \tag{4.4}$$

Folgen Werten sind

$$fx\_d = 5.9421434211923247e + 02 \tag{4.5}$$

$$fy\_d = 5.9104053696870778e + 02 \tag{4.6}$$

$$cx\_d = 3.3930780975300314e + 02 \tag{4.7}$$

$$cy\_d = 2.4273913761751615e + 02 \tag{4.8}$$

4.2.4 Farben Daten berechnen

Mit dem Punktinformation kann man aus dem RGB-Bilder entspricht Farbe für denjenigen Punkt berechnen.

$$PunktKoordinate : p3d = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.9}$$

$$R = \begin{pmatrix} 9.9984628826577793e - 01 & 1.2635359098409581e - 03 & -1.7487233004436643e - 02 \\ -1.4779096108364480e - 03 & 9.9992385683542895e - 01 & -1.2251380107679535e - 02 \\ 1.7470421412464927e - 02 & 1.2275341476520762e - 02 & 9.9977202419716948e - 01 \end{pmatrix} \tag{4.10}$$

$$T = \begin{pmatrix} 1.9985242312092553e - 02 \\ 7.4423738761617583e - 04 \\ 1.0916736334336222e - 02 \end{pmatrix} \tag{4.11}$$

Raumposition des Gesicht-Punkts:

$$p = R \times p3d + T \tag{4.12}$$

Mit der räumliche Position kann die Pixelposition i, j berechnet werden-

$$i = \frac{((p\_x \times fx\_rgb)}{z} \times cx\_rgb \tag{4.13}$$

$$j = \frac{((p\_y \times fy\_rgb)}{z} \times cy\_rgb \tag{4.14}$$

Der Inhalt des Vertext-Dateis lässt sich entweder direkt als Punktwolke ausgeben oder zur Laufzeit als Meschen darstellen. Mit dem Datei lassen sich die Normalen des Meshes sehr einfach mitberechnen. Nachdem farbige Punktwolke erstellt wird, kann man durch einen Server Socket an Klient gesendet wird.

5 3D Gesicht Rekonstruktion

In diesem Kapitel wird die Implementierung der Gesicht-Rekonstruktion vorgestellt. Die drei wesentliche Bestandteile der Umsetzung des im vorherigen Kapitel vorgestellten Planungsverfahren sind die Kinect-Steuerung, Bilddatei-Kalibrierung, Klient-Implementation.

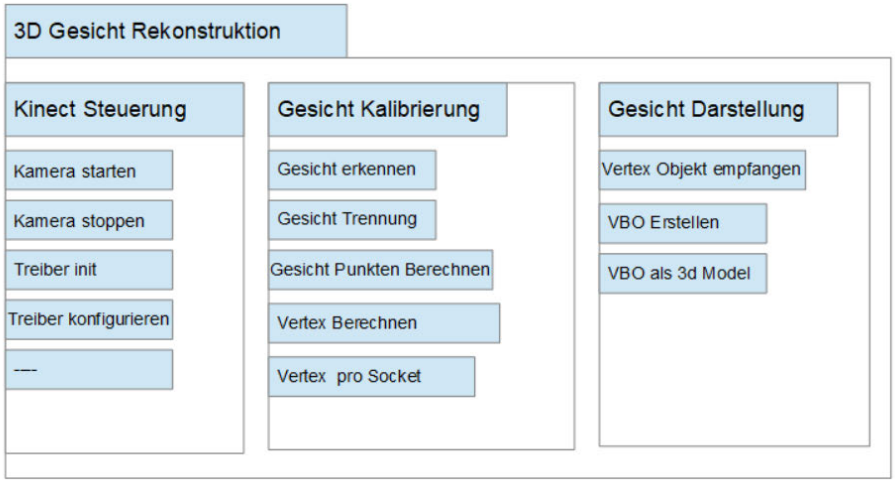

Abbildung 5.1: Software Implementation

5.1 Kinect Steuerung

Das Kamerasystem-Steuerung wird so implementiert, dass es jedes Mal beim Systemstarten die Konfiguration upgedatet und überprüft. Der gesamte Ablauf wird in der folgenden Abbildung schematisch dargestellt.

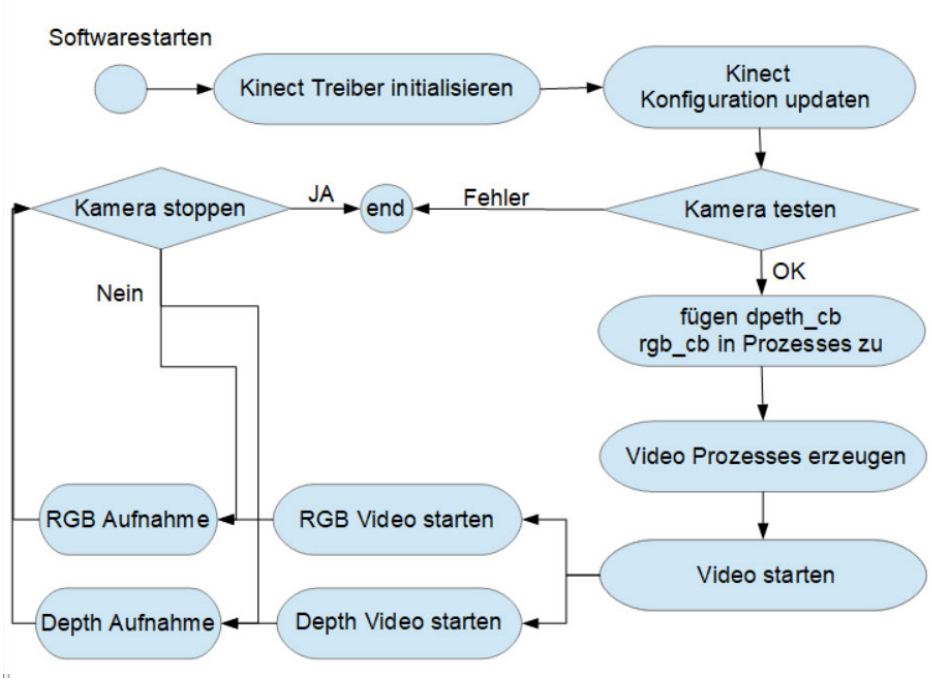

Abbildung 5.2: Ablauf für die Kamerasteuerung der Kinect V1

Die Kamera-Steuerung wird mit *libfreenect.jar* realisiert. z.B.:

- Treiber installieren:

```
freenect_init(&f_ctx, NULL)
```

- Kinect V1 Kamera Treiber starten:

```
freenect_open_device(f_ctx, &f_dev, user_device_number)
```

- Datenaufnahmen Prozess erzeugen und starten:

```
pthread_create(&freenect_thread, NULL, freenect_threadfunc, NULL
    )
```

- Set Depth-Image Aufnahme Methode:

```
freenect_set_depth_callback( getDevice(),depth_callbacck)
```

- Depth-Image Aufnahme starten:

```
freenect_start_depth(getDevice());
```

- Set RGB-Image Aufnahme Methode:

```
freenect_set_video_callback(getDevice(), rgb_callback);
```

- RGB-Image Aufnahme starten:

```
freenect_start_video(getDevice());
```

Nach dem Start des Kinect-Kamerasystems werden die EGB- und Depth-Image Data aufgenommen.

5.2 Bildbearbeiten mit OpenCV

In dieser Implementation wird die Opensourcebibliothek OpenCV benutzt. Folgende Headerklasse werden im Projekt verwendet.

- *opencv2/core.hpp* – Grundlegende Datenstrukturen und Algorithmen
- *opencv2/imgproc.hpp* – Bildverarbeitungsfunktionen (Filtern,Histogramme,...)
- *opencv2/video.hpp* – Tracking in Videos und Bewegungssegmentierung
- *opencv2/videoio.hpp* – Funktionen für das Lesen und Schreiben von Videos in verschiedenen Formaten
- *opencv2/objdetect.hpp* – Funktionen für Objektdetektion
- *opencv2/highgui.hpp* – Einfache GUI-Funktionalität mit Bildanzeige,und Parametereinstellungen
- *opencv2/opencv.hpp* – Inkludiert alles, was es in OpenCV gibt

5.3 Bildaufnahmen

Um die erfassten Bilddaten in der Arbeit mit OpenCV weiter bearbeiten zu können, bietet sich in OpenCV das Speichern des Bildens als Mat an. Mit $cv :: Mat$ - Objekte kann man Bilder in verschiedenen Datentypen speichern.

- Das Depth-Image Data wird in Mat-Objekt als Grauwert gespeichert. $depth\_img = cv :: Mat(introws, intcols, inttype)$ erstellt eine Matrix mit $rows = 480$ Zeilen und $cols = 640$ Spalten mit nicht initialisierten Werten des entsprechenden Typs $type = CV\_8UC1$ (8-bit Grauwert)
- Das RGB-Image Data wird in Mat-Objekt als Farbbild gespeichert. $rgb\_img = cv :: Mat(introws, intcols, inttype)$ erstellt eine Matrix mit $rows = 480$ Zeilen und $cols = 640$ Spalten mit nicht initialisierten Werten des entsprechenden Typs $type = CV\_8UC3$ (3-Kanaliges Farbbild mit 8 Bit pro Kanal)

Das Empfangen der Bilder werden über den Methoden $rgb_callback$ und $depth_callback$ realisiert. Die Bilderdata sind dann im vorherigen Abschnitt definierte Mat-Projekte rgb_img und $depth_img$ gespeichert und werden im nachfolgenden Abschnitt für Gesicht-Kalibrierung genutzt.

5.4 Gesicht-Kalibrierung

Mit den von dem Kinect-Kamerasystem aufgenommenen RGB- und Depth-Image-Data wird dann die Anwendung für Gesicht-Kalibrierung implementiert, dabei kommt eine bereits vorhandene OpenCV-Bibliothek zum Einsatz.

5.4.1 Gesichtserkennung

Zunächst erfasst die Kinect-Kamera auf optischem Wege das RGB-Farbenbild(rgb_img) und Depth-Bild ($depth\_img$) des Kamerasichtfeldes. OpenCV (Open Source Computer Vision) liefert alles, was wir benötigen, um in Bildern menschliche Gesichter zu erkennen. In dieser Arbeit wird für Gesichtserkennung der von OpenCV unterstützte Deep Learning Framework Caffe Model verwendet. Um das OpenCV-Deep-Neural-Network-Modul mit Caffe-Modellen benutzen zu können, benötigen wir zwei Dateien:

- *.prototxt*-Datei als $caffeConfigFile$, die Modellarchitektur definiert
- *.caffemodel*-Datei als $caffeWeightFile$, die Gewichte für tatsächlichen Ebenen enthält

In dieser Arbeit werden zwei bereits trainierte Dateien *deploy.prototxt* und $res10\_300x300\_ssd\_iter\_140000\_fp16.caffemodel$ von diesen Quellen verwendet:

- https://raw.githubusercontent.com/opencv/opencv/master/samples/dnn/face_detector/deploy.prototxt
- https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20180205_fp16/ res10_300x300_ssd_iter_140000_fp16.caffemodel

Die gesamte Ablauf der Gesichtserkennung kann wie folgenden implementieren:

1. Es Wird zuerst einige Parameter definiert:

```
const size_t inWidth = 300;
const size_t inHeight = 300;
const double inScaleFactor = 1.0;
const float confidenceThreshold = 0.666666;
const cv::Scalar meanVal(104.0, 177.0, 123.0);
const Size scaledSize(640, 640 * rgb_img.rows / rgb_img.cols);
```

2. Das Netzwerk initialisieren:

```
Net net = cv::dnn::readNetFromCaffe(caffeConfigFile,
caffeWeightFile);
```

3. Zuerst wird einen 4-dimensionalen Blob (Binary Large Object, darunter werden Bereiche aus einem Bild oder das Bild an sich gefasst) aus dem Bild rgb_img erstellt:

```
cv::Mat inputBlob = blobFromImage(rgb_img,
inScaleFactor,
cv::Size(inWidth, inHeight),
meanVal,
false,
false);
```

4. Einen neuen Eingabewert für das Netzwerk zuweisen:

```
net.setInput(inputBlob, "data");
```

5. Gesichtserkennung in dem Netzwerk durchführen:

```
cv::Mat detection = net.forward("detection_out");
```

6. Alle Gesichtsinformation von dem *rgb_img* in einer Array abspeichern:

```
cv::Mat detectionMat(detection.size[2],
detection.size[3],
CV_32F,
detection.ptr<float>());
```

7. Gesichts Positionen $p1(x1, y1)$ und $p2(x2, y2)$ können dann ausgegeben werden:

```
x1 = static_cast<int>(detectionMat.at<float>(i, 3) * frameWidth)
    ;
y1 = static_cast<int>(detectionMat.at<float>(i, 4) * frameHeight
    );
x2 = static_cast<int>(detectionMat.at<float>(i, 5) * frameWidth)
    ;
y2 = static_cast<int>(detectionMat.at<float>(i, 6) * fameHeight)
    ;
```

5.4.2 Kopf Modell erstellen

Mit den Gesichtspositionen und der Größe kann man dann eine Punktgruppe erstellen, die die gesamte Kopfposition darstellt.

- Mit den im oberen Abschnitt berechneten Position und Größe kann man einen Pixel-Vertex Objekt *verteies* für den Kopf der betrachteten Person im Vordergrund aus *depth_img* erstellen. Die Pixel-Vertex-Folgen werden schlangenförmig in *verteies* eingefügt.

```
std::vector<Point3f> verteies;
bool steigen=true;
for (int x = x1; x < x2; x++){
if(steigen){
steigen=!steigen;
for (int y =y1;y <y2; y++){
float d=(FLOAT) depth_img.at<unsigned_char>(y,x);
if(d>30 && d<220)
verteies.push_back(cv::Point3f(x, y,d));
}
```

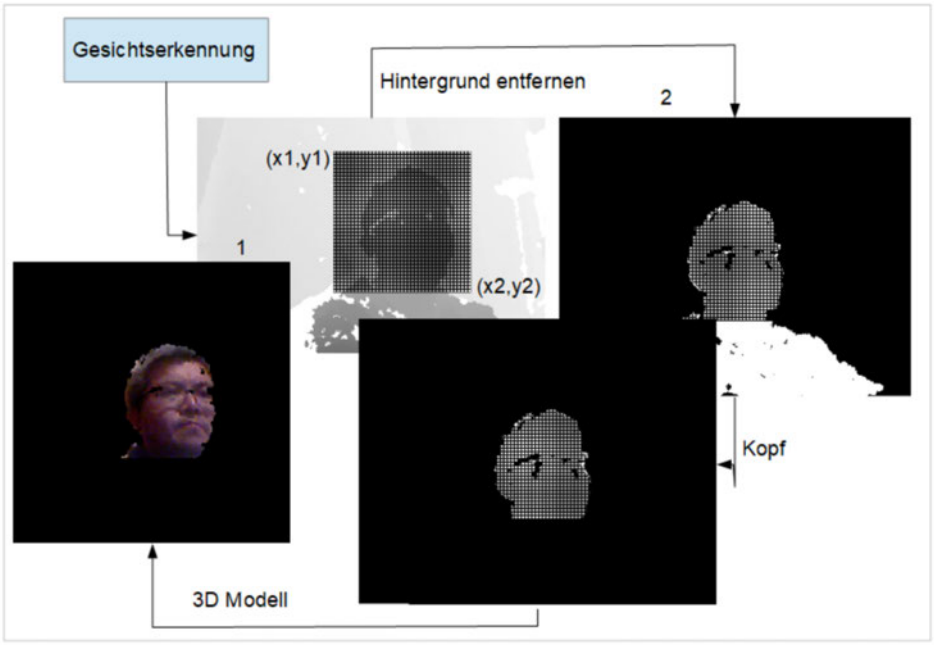

Abbildung 5.3: Erstellung eines 3D-Gesichtsmodelles

```
}else{
steigen=!steigen;
for (int y = y2-1; y > y1-1; y--){
float d=(FLOAT) depth_img.at<unsigned_char>(y,x);
if(d>30 && d<220)
verteies.push_back(cv::Point3f(x, y,d));
}
}
}
```

- Die Raumposition *pointVerteies* und Farben *colorVerteies* der Punkten in *verteies* werden mit in der Projekt implementierte Methoden *computePoint*() und *computeColor*() berechnet.

Die Vertext-Dateien *pointVerteies* und *colorVerteies* können zur Laufzeit als Meshes dargestellt werden. Da die Darstellung der 3D Modelle in dieser Arbeit auf der Endgräte (Smartphone) passiert, müssen hier die Vertext-Dateien *pointVerteies* und *colorVerteies* zuerst in einer *iovec* Struktur konvertiert werden, damit sie sich transpor-

tieren lassen. Der Inhalt sollte nun ausschließlich aus den X-, Y-, und Z-Koordinaten und den dazu gehörigen RGB-Farbwerten bestehen.

Die in der Arbeit benutzt Vertex-Strukturen können wie folgende definiert werden:

```
struct Vertex{
float position[3];
float color[3];
};
```

5.5 Netzwerkkommunikation mit Socket

Die Netzwerkkommunikation für Kommunikation und Bilddatei Übertragung zwischen Servercomputer und Endgeräte in der Arbeit verwendet die Socket-Klasse, die umfangreiche Methoden und Eigenschaften für die Netzwerkkommunikation bereitstellt.

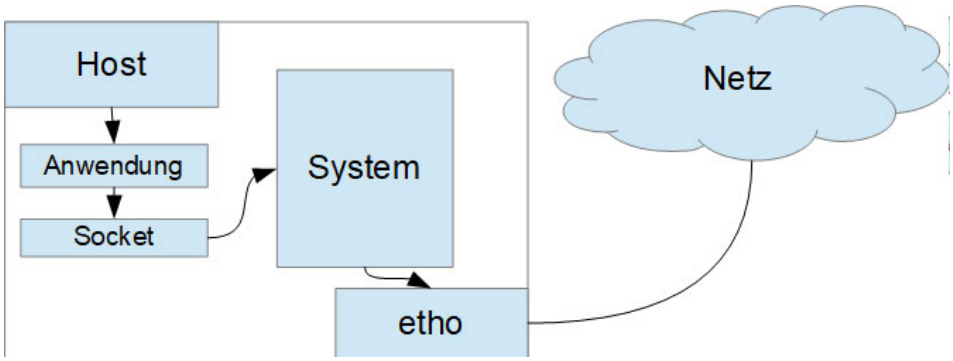

Abbildung 5.4: Netz Kommunikation

Die Socket-Klasse ermöglicht es, sowohl synchrone als auch asynchrone Datenübertragungen mithilfe eines der Kommunikationsprotokolle auszuführen. Die Socket-Schnittstelle ist damit die Grundlage der Programmierung verteilter Anwendungen unter TCP/IP. Bei Sockets geht es nicht um Dateien, sondern um Kommunikationskanäle, über die Daten gesendet und empfangen werden können.

Die in der Arbeit benutzte TCP/IP-Verbindung ist durch eine Client-Server-Architektur geprägt und damit asymmetrisch. Vor der Kommunikation muss die Verbindung hergestellt sein. Das betrifft die Verbindung zwischen den Host-Computer und Client-Gerät.

Die Adressierung der Beiden erfolgt per Hostname, der vom System auf die IP-Nummer umgesetzt wird.

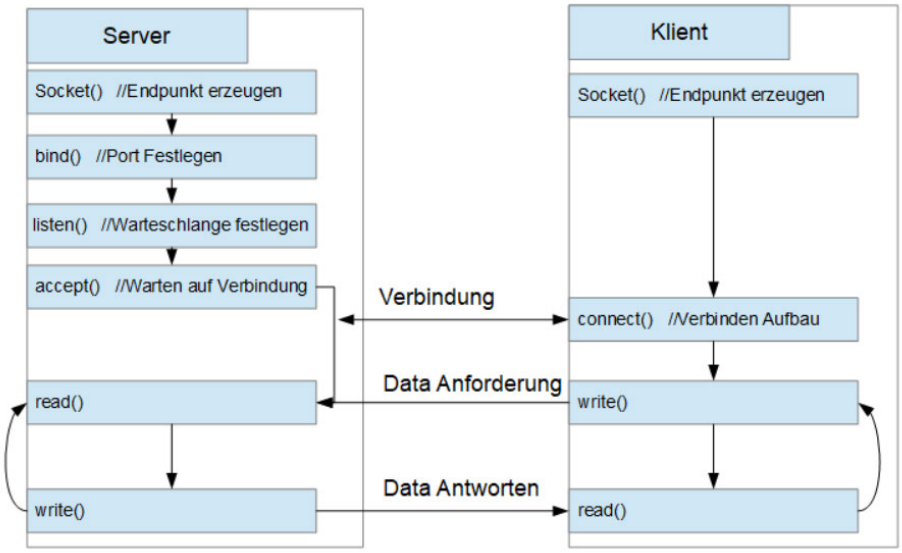

Abbildung 5.5: Client-Server-Kommunikation

5.6 Client und Server Verbindung

In der Arbeit wird das Verbindung orientiertes Protokoll TCP verwendet.

- Um mit Sockets zu arbeiten, muss zuerst eine Verbindung geöffnet werden. Dies geschieht mit dem socket()-Systemaufruf:

```
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

  socket() eröffnet eine Kommunikation im Internet. Die Funktion hat drei Parameter:

  - AF_INET: Internet-Protokolle
  - $SOCK\_STREAM$: Stream-Socket (TCP)
  - $IPPROTO\_TCP$: TCP-Protokoll ($SOCK\_STREAM$)

Der socket()-Systemaufruf liefert einen Integerwert zurück, die *id* des eröffneten Sockets. Dieser Wert wird daher *sockfd* genannt. Es muss noch festgelegt werden, für welchen Port der Socket zuständig sein soll und ob es sich um einen Server- oder Client-Socket handeln soll.

- Um den Serverprozess von außen zu erreiche braucht er einen Port. Die Portnummer muss dem Clientprozess bekannt sein. Um das Socket mit der lokalen Endpunktadresse zu binden, wird der bind-Methode verwendet.

```
bind(sockfd, address, addresslength)
```

- Die *listen*()-Methode gibt an, wieviele Anfragen gepuffert werden können.

```
listen(sockfd,backlog)
```

- Die Accept-Methode verarbeitet alle eingehenden Verbindungsanforderungen und gibt einen Socket zurück, mit dem kann der Client mit dem Remote-Host kommunizieren und Daten austauschen. Der Aufruf von *accept*() liefert als Rückgabewert die Socket-ID des Partners. Zum Verwenden des zurückgegebenen Sockets kann man die *read*() und *write*()-Methode aufzurufen..

```
accept(sockfd, address, addresslength)
```

- Sobald der Server läuft, kann der Client eine Verbindung zum *well_known_port* des Servers über einen virtuellen Kanal aufnehmen. Der entsprechende Aufruf lautet *connect*.

```
connect(sockfd, address, addresslength)
```

5.6.1 Daten Übertragung über Socket

Für die Kommunikation bei TCP-Verbindugen sind die Standard-Systemcalls *read*(), *write*(), *reav*() und *writev*() zum Bechreiben bzw. Lesen des bestehenden virtuellen Kanals einsetzbar.

```
nwrite = write(sockfd, buffer, nbytes)
nread = read(sockfd, buffer, maxbytes)
```

Der Systemaufruf *writev* kann einen Datensammler übertragen, sodass das Anwendung eine Nachricht schreiben kann, wobei diese nicht unbedingt in zusammenhängenden Speicherstellen abgespeichert sein muss. *writev* besetzt die folgende Form:

```
nwritev = writev(sockfd, iovector, vectorlen)
```

Das Argument *iovector* gibt die Adresse eines Arrays von Type *iovec* an als eine Reihe von Zeigern auf die Byteblöcke, welche die Nachricht bildet. Mit jedem Zeiger wird eine Länge übergeben. *vectorlen* spezifiziert die Anzahl der Einträge des *iovector*s.

Der Systemaufruf *readv* kann Daten von *socket* lesen und in eines Arrays von Type *iovec* speichern.

```
nreadv = readv(sockfd, iovector, vectorlen)
```

5.7 3D Gesicht Darstellen

Um das 3D Gesichtsmodell auf den Android-Gerät darzustellen, wird eine Android-Anwendung implementiert. Diese kommuniziert vom Android-Gerät aus über dem Netzwerk mit dem Server (Zentralrechner). Mit dem Socket kann man dann die auf dem Server erstellt *verteies* Datei auf das Gerät übertragen.

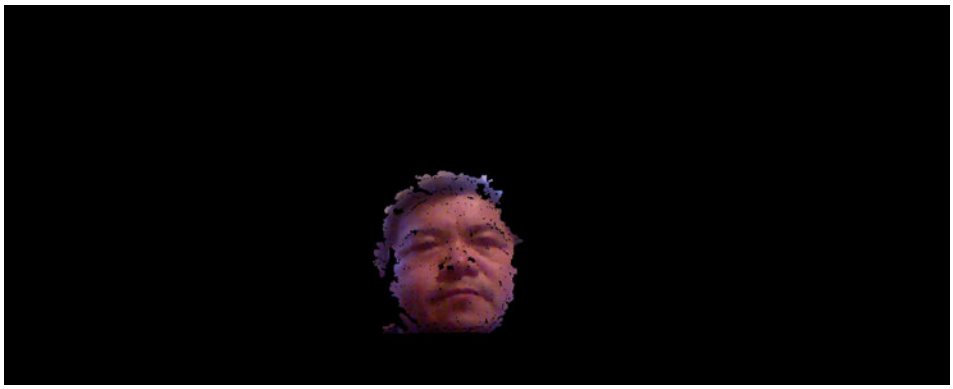

Abbildung 5.6: 3D Modell

Der Inhalt des Vertext-Datei *verteies* kann sich zur Laufzeit als Meshes dargestellt werden. Dabei lassen sich die Normalen des Meshes sehr einfach mitberechnen. Das Erstellen

eines VBOs(Vertex Buffer Objekts) geschieht in der Regel vor dem eigentlichen Rendern, also z.B. in der Funktion *init*(). Das Vorgehen umfasst folgende Schritte:

1. Mit Hilfe der Funktion $glGenBuffers(1, \&bufID)$ kann ein eindeutiger Kennzeichner generiert werden
2. Mit $glBindBuffer(GL_ARRAY_BUFFER, bufID)$ wird ein neues VBO erzeugt.
3. Mit $glBufferData(GL_ARRAY_BUFFER, verteies.size(), verteies, GL_STATIC_DRAW)$ werden die Vertex-Daten übergeben.
4. Aktivieren des VBOs für eine bestimmte Verwendung mit $glEnableClientState(GL_VERTEX_ARRAY$
5. Nach Aktivierung des VBOs wird wird das Rendern aller enthaltenen Daten mit dem Funktionsaufruf $glDrawArrays(GL_POINTS, 0, count)$ angestoßen.
6. Wenn der VBOs nicht mehr verwendet wird, kann der Speicher mit $glDeleteBuffers(1, \&bufID)$ wieder freigegeben werden

6 Evaluation

6.1 Optische Probleme während der Aufzeichnung

Da die Methode der Kinect V1 zur Tiefenerkennung auf der Reflektion von Infrarotlicht basiert, kommt es zu diversen Problemen, wenn die Infrarotstrahlung nicht wie erwartet reflektiert wird. Ist das getroffene Material lichtdurchlässig, wie zum Beispiel Fensterglas oder Folien, werden die Punkte nicht reflektiert. Sie werden auf das dahinter liegende Material projiziert, welches häufig zu weit entfernt ist, um von der Kamera richtig erkannt zu werden. Es wird keine Tiefe aufgezeichnet und somit entsteht ein Loch in der Punktwolke.

6.2 Messgenauigkeit

Die während der Aufzeichnung des 3D Gesicht-Model auftretenden Probleme lassen sich grob in drei Kategorien teilen. Auf der einen Seite entstehen Fehler in der Rekonstruktion während der Berechnung des Punktwolken. Auf der anderen Seite kann es aufgrund von optischen Problemen schon während der Aufnehme von Tiefendaten selbst zu Löchern in der Rekonstruktion kommen. Die auftretenden Probleme werden im folgenden näher beschrieben.

6.2.1 Treiber Einschränkung

Frame verlost beim Bild aufnahmen Inder Dateiübertragung von Kinect v1 zu Host-Computer kann es oft Passieren, dass das Bilddatei während des Übertragung verlostet werden kann.

6.2.2 Hardware Einschränkung

In der Arbeit benutzte Kinect V1 Kamera wird zur Tiefenerkennung auf der Reflektion von Infrarotlicht basiert. Es kommt zu diversen Problemen führen, dass die Infrarotstrahlung nicht wie erwartet reflektiert werden können. Besondere sind die Brille Träger und das dunkel Haariger.

- Durch das dunkel Haar, das zu dunkel Brille Fassung und Gläser werden die Infrarotstrahlung absorbiert. Es kommt zu einem Loch in der Punktwolke, da keine Tiefendaten aufgezeichnet werden können.

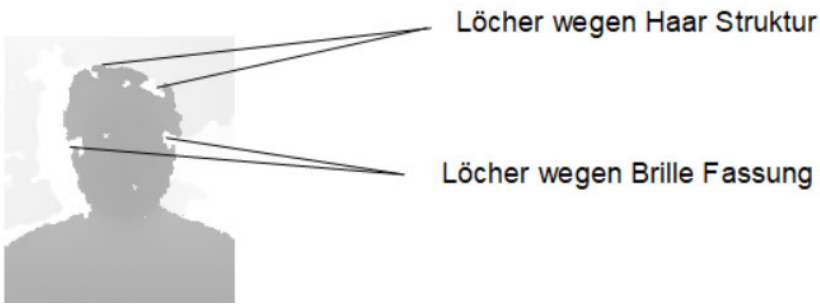

Abbildung 6.1: Kamera Steuerung

- Ist das Fassung-Material sehr glatt und der Winkel der Kamera zu dieser Oberfläche zu gering, wird ein Großteil des Infrarotlichts nicht zur Kamera zurückgeworfen, sondern von der Oberfläche im Einfallswinkel reflektiert. In diesem Fall kommt es ebenfalls zu einem Loch in der Punktwolke, da keine Tiefendaten aufgezeichnet werden. Sehr glatt Haar kann auch die gleiche Effekt führen.

- Das sehr dünner oder schmalen Fassung von Brille und auch das Haar kann die Grenzen der Kinect V1 stoßen, weil die Oberfläche des Gegenstandes zu klein ist. Dann werden die Gegenstande nicht von dem Kamera erkannt.

- Bei starker Sonneneinstrahlung oder der Raum Beleuchtung ist die Leistung des Infrarotlichts der Kinect nicht ausreichend, um sich von dem der Sonne abzuheben. Es können keine Punkte erkannt werden und somit kommt es auch in diesem Fall zu Löchern in der Punktwolke.

- Nicht ausreichte Depth-Kamera Auslösung (240x320) führt zu Ungenauigkeit an der Kopfgrenze.

7 Fazit und Schluss

Es konnte gezeigt werden, dass durch Verwendung von Kinect Kamerasystem die Gewinnung von Tiefenbildern und Farberblidern unter sehr geringem Rechenaufwand möglich ist.

Da aber die Information zur Entfernungsbestimmung direkt aus den gemessenen Intensitäten errechnet werden, ist die Verwendung einer Kamera mit hoher Auflösung und das Vorhandensein kontrastreicher Oberflächen erforderlich.

Die Kinect V1 projiziert ein pseudozufälliges Muster, aus Nah-Infrarot-Laserpunkten, auf den kompletten, für die Kamera sichtbaren Bereich. Die Hardware arbeitet das erfasste Bild in Blöcken ab und sucht in diesen zuerst nach einzelnen Punkten. Diese müssen dann mit benachbarten Punkten in Zusammenhang gesetzt werden. Wird ein bekanntes Muster erkannt, kann mittels Triangulation für einen bekannten Punkt im Sichtfeld der Kamera die Distanz berechnet werden.

Ein Problem dieses Ansatzes ist es, dass auf zu kleinen Oberflächen nicht ausreichend Punkte erfasst werden können, um einen Teil des Musters adäquat zu identifizieren. Bei sehr dünnen oder schmalen Gegenständen wie zum Beispiel Haaren oder Kabeln stößt die Kinect V1 deshalb an ihre Grenzen. Bewegliche Objekte oder auch Schatten von in der Nähe befindlichen Personen oder Körpern können die Messung massiv stören.

Zusammenfassend lässt sich also festhalten, dass das vorgestellte Verfahren das Potenzial besitzt, zur optischen Raumdatenerfassung von Gesicht eingesetzt werden zu können. Die stetige Steigerung der verfügbaren Kameras bezüglich Empfindlichkeit und Auflösungsvermögen wird das Erreichen der angestrebten räumlichen Genauigkeitsgrenzen nach der vorgestellten Methode in naher Zukunft ermöglichen.

Literaturverzeichnis

[1] : *Mainpage*. – URL https://spatial.io/

[2] Barakonyi, Istvan ; Fahmy, Tamer ; Schmalstieg, Dieter: Remote Collaboration Using Augmented Reality Videoconferencing. In: *Proceedings of Graphics Interface 2004*, ., 2004, S. 89–96

[3] Billinghurst, Mark ; Kato, Hirokazu: Collaborative Augmented Reality. In: *Communications of the ACM* 45 (2002), S. 64–70

[4] Gruen, Armin ; Huang, Thomas S.: BOOK REVIEW: Calibration and Orientation of Cameras in Computer Vision, 2001

[5] Hjelmas, Erik ; Low, Boon K.: Face detection: a survey. In: *Computer Vision and Image Understanding* 83 (2001), Nr. 3, S. 236 – 274

[6] Lienhart, Rainer ; Kuranov, Er ; Pisarevsky, Vadim: Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. In: *In DAGM 25th Pattern Recognition Symposium*, 2003, S. 297–304

[7] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott ; Fu, Cheng-Yang ; Berg, Alexander C.: *SSD: Single Shot MultiBox Detector*. 2015. – URL http://arxiv.org/abs/1512.02325. – cite arxiv:1512.02325Comment: ECCV 2016

[8] Mridul, Kumar ; Ramanathan, M. ; Ahirwar, Kunal ; Sharma, Mansi: *Multi-user Augmented Reality Application for Video Communication in Virtual Space*. 2019

[9] Szalavári, Zsolt ; Schmalstieg, Dieter ; Fuhrmann, Anton ; Gervautz, Michael: 'Studierstube' An Environment for Collaboration in Augmented Reality / Institute of Computer Graphics and Algorithms, Vienna University of Technology. Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, Januar 1997 (TR-186-2-97-01). – Forschungsbericht

[10] Toshiyuki, Sakai ; Makoto, Nagao ; Kanade, Takeo: Computer Analysis and Classification of Photographs of Human Faces. In: *Proceedings of Proc. First USA-JAPAN Computer Conference*, January 1972, S. 55–62

[11] Viola, P. ; Jones, M.: Rapid object detection using a boosted cascade of simple features. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* Bd. 1, 2001, S. I–511–I–518 vol.1. – ISSN 1063-6919

Anhang

Faces - Dynamische Rekonstruktion von Gesichtern

Repository Download: ($https://gitlab.informatik.haw-hamburg.de/vr_ar_lab/cg_vr_ar.git$)

Das Projekt hat Abhängigkeiten zu

- libusb (neuste Version)
- libfreenect (Version 0.5.4 Wander)
- Qt 5 (neuste Open Source Version, Registrierung erforderlich, keine Kosten)

Einrichtung (Windows)

Windows-spezifische zusätzliche Abhängigkeiten:

- CMake ([URL](https://cmake.org/download/))
- Visual Studio 2017
- POSIX Threads Win32

Das Qt5-<bin>-Verzeichnis mit <qmake.exe> sollte im Windows-Pfad stehen, damit man es per Kommandozeile verwenden kann.

Visual Studio Projekt erstellen

Die Bibliotheken, von denen das Projekt abhängt liegen bereits vorkompiliert im Ordner <external_libs>. Daher kann direkt das Visual Studio Projekt erstellt werden:

cd build

qmake ../faces.pro && qmake -tp vc -r ../faces.pro

Externe Bibliotheken selber kompilieren

libusb

- TAR-Ball (Source) von [URL](https://github.com/libusb/libusb/releases) herunterladen
- <libusb_2017.sln> in Verzeichnis <msvc> mit Visual Studio öffnen
- Projekt bauen
- Ergebnis
- <libusb.h> nach <external_libs>/<platform>/<include>
- <libusb-1.0.lib> nach <external_libs>/<platform>/<lib>

POSIX Threads

- Vorkompilierte Version herunterladen [URL](https://sourceforge.net/projects/pthreads4w/)
- Ergebnis
- <pthread.h>, <sched.h>, <semaphore.h> nach <external_libs>/<platform>/<include>
- <pthreadVC2.lib> nach <external_libs>/<platform>/<lib>
- <pthreadVC2.dll> nach <bin>

libfreenect

- Download von Freenect: [URL](<https://github.com/OpenKinect/libfreenect/releases>) (neuste Version, 0.57)
- CMake Öffnen
- Source-Dir: <libfreenect>
- Binary-Dir: <libfreenect/build> (neuer Ordner)
- Target: Visual Studio 2017
- für libusb
- Source-Verzeichnis auf <libusb/include>
- Library-Verzeichnis auf <libusb/lib>
- für pThread
- Library-Verzeichnis auf <libusb/lib>
- BUILD_EXAMPLES nicht auswählen
- BUILD_FAKENECT nicht auswählen
- Configure (besser 2x)
- Generate
- Visual Studio Solution aus Verzeichnis build öffnen
- Ergebnis
- <include>-Ordner nach <external_libs>/<platform>/<include>/<libfreenect>
- <freenect.lib> nach <external_libs>/<platform>/<lib>
- <freenect.dll> nach <bin>

Einrichtung (macOS)

macOS-spezifische zusätzliche Abhängigkeiten:

- Homebrew
- XCode

libusb

Bibliothek für den Zugriff auf USB-Geräte. Verwendet von libfreenect. Für die Installation wird hier Homebrew (<https://brew.sh/>) eingesetzt. Ebenso ist Fink (<http://www.finkproject.org/>) oder die manuelle Kompilierung denkbar.

brew install libusb

lbfreenect

Schnittstelle zu einer Kinect 1 (Kinect für Windows) Kamera. Achtung: unter macOS muss die Version 0.5.4 Wander (und nicht etwa die aktuelle Version) verwendet werden. Zumindest auf dem Testgerät liess sich die Kamera mit den neueren Versionen nicht ansteuern.

Download: <https://github.com/OpenKinect/libfreenect/releases>

mkdir build

cd build

ccmake .. (Configure, Generate)

make

sudo make install

Qt 5

Installation von der Qt-Webseite: <https://www.qt.io/>

XCode-Projekt

Repository clonen

cd <Repo-Dir>

cd faces

XCode-Projekt erstellen:

qmake && qmake -spec macx-xcode

oder mit dem bereist vorhandenen Skript

./make_project.sh

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Aufzeichnung und Streaming von dynamischen 3D Gesichtsmodellen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original