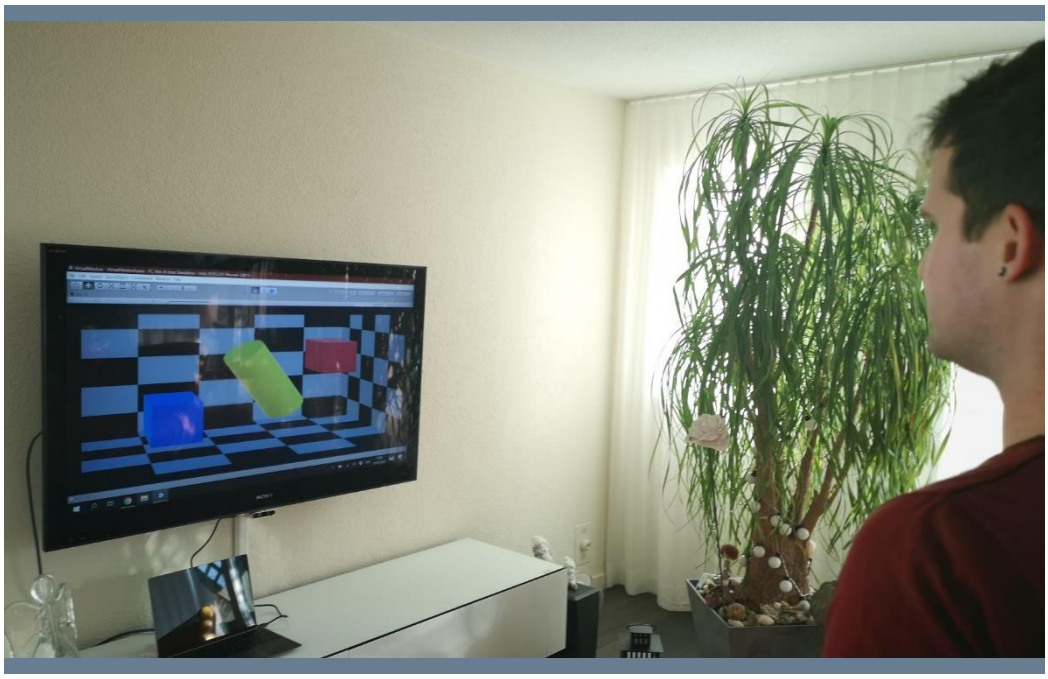


Virtual Window TV

Eine perspektivische Projektion

Bachelorthesis

Studiengang:	BSc Informatik
Autor:	Carosella Dario
Betreuer:	Hudritsch Marcus
Experten:	Dr. Studer Harald
Datum:	16.01.2020

Versionen

Version	Datum	Status	Bemerkung
0.1	01.10.2019	Initial	
0.2	04.01.2020	Draft	Erster Entwurf
0.25	06.01.2020	Draft	Korrektur 1. Entwurf
0.3	10.01.2020	Draft	Zweiter Entwurf
0.35	13.01.2020	Draft	Korrektur 2. Entwurf
1.0	16.01.2020	Final	Finale Version

Management Summary

Vision

Der Fernseher soll zu einem Kunstobjekt werden und auch etwas bieten, wenn er gerade nicht «eingeschalten» ist. Fernseher, die an der Wand aufgehängt sind, wirken meist etwas öde. Samsung hat dies mit dem **THE FRAME** Fernseher geändert. Der Fernseher kann mit unterschiedlichsten Rahmen ausgestattet werden. Zusammen mit dem Art Mode verwandelt sich der Fernseher schliesslich zu einem Gemälde und man nimmt den Fernseher gar nicht mehr als Fernseher war. Mit meinem Ansatz will ich Samsung toppen noch ein realeres Raumerlebnis schaffen.

Idee

Auf oder unter einem Fernseher, der an einer Wand angebracht ist, soll eine Tiefenkamera installiert werden, die dem Computer die genaue Position des Betrachters mitteilt. Die Software generiert daraufhin ein Bild auf dem Bildschirm, welches sich auf die Position des Betrachters bezieht, ähnlich wie bei einem Trompe-l'oeil. Damit kann dem Betrachter ein virtuelles Fenster in der Wand mit einem virtuellen Ausblick generiert werden. Der generierte Inhalt ist dabei dynamisch und ändert sich mit der Position des Betrachters. Der Betrachter soll dabei in den Bann gezogen werden und neugierig reagieren. Was passiert, wenn ich näher an den Fernseher gehe? Was geschieht, wenn ich aufspringe? Der Betrachter soll beginnen zu spielen und experimentieren

Strategie

Für die Realisierung gab es drei Hauptschritte, die umgesetzt wurden:

1. Mittels einer Tiefenkamera, der Intel RealSense D435, wird die Position des Betrachters getrackt. Hierbei wurde auf das SDK von Nuitrack zurückgegriffen welches als Middleware zwischen der Tiefenkamera und des Computers dient.
2. Anschliessend wurde mit Blender eine Szene modelliert und in Unity, eine Spiel-Engine, importiert.
3. Schliesslich wird die Perspektive relativ zum Betrachter berechnet und Unity generiert auf dem Fernseher das der perspektiven entsprechenden Bild.

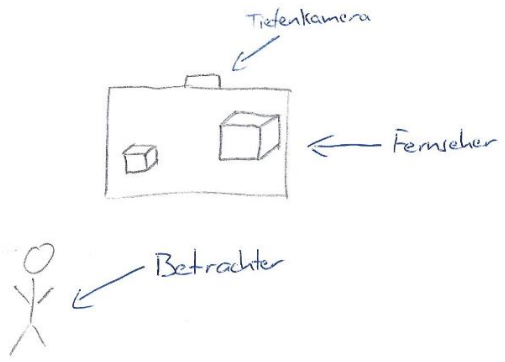


Abbildung 1: Skizze der Idee

Inhaltsverzeichnis

1 Einleitung

Im Rahmen meines Bachelorstudiums an der Berner Fachhochschule, Studiengang Informatik, schrieb ich im neunten und letzten Semester diese Bachelorarbeit. Da ich als Vertiefung *Computer Perception and Virtual Reality* gewählt habe, entschied ich mich auch meine Bachelorarbeit in diesem Themengebiet zu absolvieren. Zusammen mit meinem Betreuer, Marcus Hudritsch, entstand nach diversen Gesprächen die Idee zur Bachelorarbeit mit dem Arbeitstitel **Virtual Window TV**. Was kann man sich darunter vorstellen? Grundsätzlich geht es darum, einen Fernseher etwas aufzupeppen. Meistens hängen die Fernseher in der Wohnstube an der Wand und sehen einfach nur öde aus, wenn sie nicht eingeschaltet sind. Das soll sich durch das System, das ich entwickelt habe nun verändern!

Die Dokumentation ist in folgende Themenblöcke gegliedert:

1. **Ideen und Stakeholder(s)**
    a. Was sind die Projektziele?
    b. Wer sind die Stakeholder(s)?
    c. Was gibt es bereits auf dem Markt?
2. **Anforderungen und theoretische Grundsätze**
    a. Welche (funktionalen und nicht-funktionalen) Anforderungen gibt es?
    b. Welche technischen Voraussetzungen braucht es?
    c. Grundlegende Theorie
3. **Projektvorgehen / Umsetzung und Organisation**
    a. Arbeitsorganisation
    b. Umsetzung / Realisierung
4. **System Review und Fazit**
    a. Funktioniert das System?
    b. Sind die Projektziele erreicht?
    c. Was gibt es für Verbesserungspotenzial?
    d. Projektbezogenes sowie persönliches Fazit

2 Idee

2.1 Projektziel

Auf oder unter einem Fernseher, der an einer Wand angebracht ist, soll eine Tiefenkamera installiert werden. Die dem Computer die genaue Position des Betrachters mitteilt. Die Software generiert daraufhin ein Bild auf dem Bildschirm, welches sich auf die Position des Betrachters bezieht, ähnlich wie bei einem Trompe-l'oeil [1]. Damit kann dem Betrachter ein virtuelles Fenster in der Mauer mit einem virtuellen Ausblick generiert werden. Der generierte Inhalt ist dabei **dynamisch** und ändert sich mit der **Position des Betrachters**. Eine Steigerung dieser optischen Täuschung kann durch Berücksichtigung der realen Beleuchtungsverhältnisse erreicht werden.

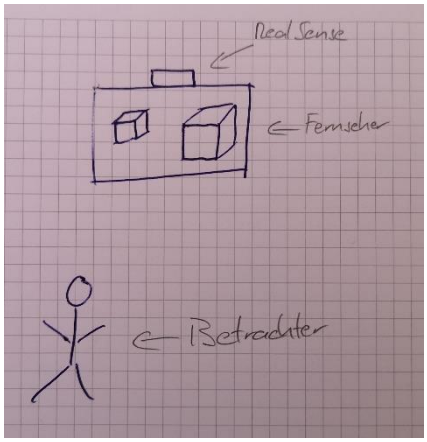


Abbildung 3: Skizze der Idee

Abbildung 2: Trompe l'oeil

Das ganze Zusammenspiel soll ähnlich funktionieren wie bei einem ***Samsung The Frame*** [2] Fernseher. Sprich der Betrachter sollte nicht das Gefühl haben, das er vor einem TV steht. Samsung erreicht dies, indem der Fernseher einen Holzrahmen besitzt. Zusammen mit einem Standbild, das nun dargestellt wird, denkt der Betrachter, dass er sich ein Gemälde ansieht. Der Fernseher verschwindet also und genau diesen Effekt möchte ich mit meinem System erreichen. Der Betrachter soll in den Bann gezogen werden und neugierig reagieren. Was passiert, wenn ich näher an den Fernseher gehe? Was geschieht, wenn ich aufspringe? Der Betrachter soll beginnen zu «spielen» und experimentieren. Das Hauptziel ist es den Fernseher als Objekt verschwinden zu lassen und die (vorhandene) Umgebung zu integrieren.

2.2 Stakeholders

Zielgruppen sind Lifestyle affine Personen wie z.B. Kunstliebhaber und Hobby Innenarchitekten sowie Technikfreaks. Der Fernseher soll auch etwas bieten, wenn er nicht eingeschalten ist. Meistens hängen die schwarzen «Glotzen» einfach nur an der Wand oder stehen auf einem stylischen Sideboard. Zusammen mit der hübsch eingerichteten und dekorierten Wohnung sieht der Fernseher dann etwas öde und verloren aus.

2.3 Marktanalyse

Wie bereits im vorherigen *Kapitel 2.1 Projektziele* erwähnt hat Samsung mit dem The Frame aktuell ziemlich vorgelegt. In der unterstehenden Abbildung ist der Fernseher nur schwer zu erkennen. Er verwandelt sich zusammen mit dem Holzrahmen und der Software der sogenannte Ar Mode zu einem einzigartigen Gemälde. Der Art Mode[3] ermöglicht es auch zwischen diversen Bildern zu wechseln und somit für Abwechslung in der Wohnstube zu sorgen.

Einen ähnlichen Effekt, also der des «Verschwindens», soll auch beim Virtual Window TV erreicht werden. Allerdings soll bei meinem Ergebnis noch eine spielerische Komponente berücksichtig werden und der Betrachter soll sich im Raum bewegen und neugierig reagieren.

Abbildung 4: Samsung The Frame [3]

2.4 Prototyp

Als Vorläufer der Bachelorarbeit wurde im 8. Semester während des Moduls BTI7302 – Projekt 2 die Grundlagen für die Thesis gelegt. Einerseits war es das Ziel die Tiefenkamera und deren SDK zu verstehen und herumzuexperimentieren, was diese alles zu bieten hat und andererseits einen Prototyp zu erstellen, welcher die Position der Person im Raum erkennt (und darstellt) sowie den TV zusammen mit der Tiefenkamera den Betrachtungswinkel simuliert. Dieser Prototyp wurde mittels Unity und dem Nuitrack SDK[4] erstellt:

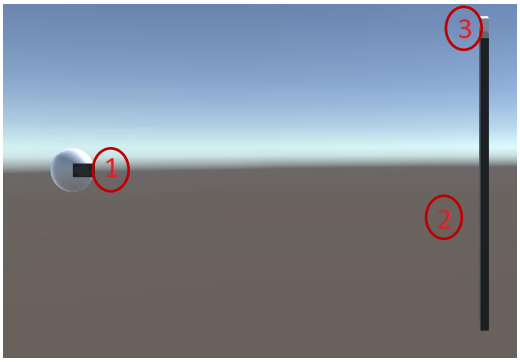


Abbildung 6: Seitenansicht des Prototyps

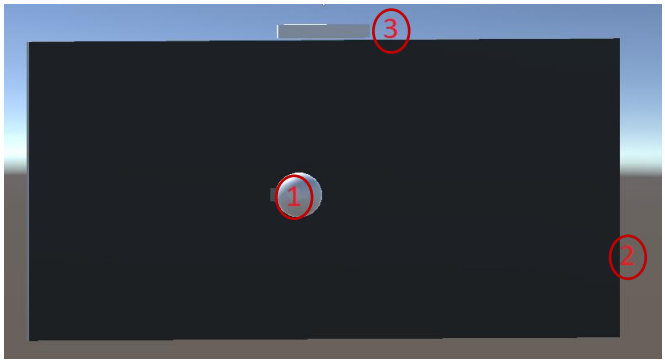


Abbildung 5: Frontansicht des Prototyps

Bildlegende (gültig für beide Abbildungen):

1. Der Kopf des Betrachters (Eine Kugel mit Brille)
2. Der Fernseher
3. Die Tiefenkamera

Bewegt sich die Person im Raum, so ändert sich auch die Position der Kugel (Nummer eins). Schaut man sich das Ganze im Game-Modus von Unity[5] an, so sieht man den TV direkt vor sich. Bewegt man sich nun vor der Kamera hin und her oder auch nach vorne oder zurück, so ändert sich auch die Perspektive auf den Fernseher. Es wird also simuliert, wie der Fernseher aufgrund der Perspektive wahrgenommen wird. Ist der Fernseher genau auf Augenhöhe und man schaut von leicht links auf den Fernseher, so sieht das wie folgt aus:

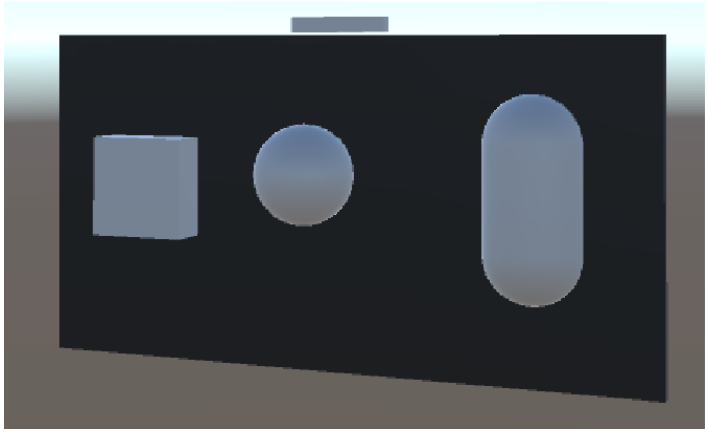


Abbildung 7: Prototyp Ansicht von Links im Gamemodus

Mit diesen Erkenntnissen wurden die Grundlagen geschaffen um mit der Bachelorarbeit zu starten.

3 Anforderungen & theoretische Grundsätze

In diesem Kapitel sind die funktionalen sowie nicht funktionalen Anforderungen an das System beschrieben. Zudem werden hier die theoretischen Grundsätze und Grundlagen für die Umsetzungen von zwei Lösungsansätzen, die im Verlauf der Bachelorarbeit entstanden sind, erläutert.

3.1 Funktionale Anforderungen

Aus den definierten Zielen ergeben sich folgende Detailanforderungen an das System.

Nr.	Titel	Beschreibung	Muss/kann
1	Betrachter/Person erkennen	Die genaue Position des Betrachters muss erkannt werden.	**Muss**
2	Szene modellieren	Es soll eine Szene in Unity/Blender modelliert/erstellt werden, welche auf dem Fernseher und somit für den Betrachter zu sehen ist.	**Muss**
3	Dynamisches Bild generieren	Bewegt sich der Betrachter im Raum, so muss anhand seiner Position die Szene perspektivisch korrekt dargestellt werden.	**Muss**
4	Helligkeit berücksichtigen	Die Raumhelligkeit soll erkannt werden und die Helligkeit in der Szene/Bild dementsprechend angepasst werden.	Kann

Tabelle 1: Funktionale Anforderungen

3.2 Nicht funktionale Anforderungen

Nr.	Titel	Beschreibung	Muss/kann
1	Plattform unabhängig	Das Endergebnis soll auf allen gängigen Plattformen lauffähig sein.	**Muss**
2	Tiefenkamera unabhängig	Die Position des Betrachters soll mit den meisten gängigen Tiefenkameras ermittelt werden können. Somit muss das SDK Plattformunabhängig sein	**Muss**
3	Applikation für Smart-TV entwickeln	Die Applikation soll auf einem Smart-TV laufen, somit braucht es keinen zusätzlichen Computer.	Kann

Tabelle 2: Nicht funktionale Anforderungen

3.3 Technische Voraussetzungen

In diesem Kapitel sollen die technischen Voraussetzungen genauer beschrieben werden:

- Was für ein Tracking-System wird eingesetzt?
- Wie funktioniert eine Tiefenkamera?
- Was ist stereoskopisches Sehen?
- Warum habe ich mich für dieses System entschieden?

All diese Fragen werden im nachfolgenden Kapitel behandelt und beantwortet.

3.3.1 Tracking-System

3.3.1.1 RealSense – Tiefenkamera

Die Tiefenkamera *RealSense D435* von Intel ist eine Stereo-Tracking-Lösung [6]. Sie besitzt ein weites Sichtfeld, welches sich perfekt für Anwendungen wie Robotik, Augmented und Virtual Reality anbietet. In all den genannten Bereichen ist es von entscheidender Bedeutung, so viel wie möglich von der Szene zu sehen. Die Kamera besitzt eine Reichweite von bis zu 10 Metern. Nebst dem grossen Sichtfeld besitzt die Kamera noch einen Global-Shutter-Sensor. Dieser ist ideal für die Anwendung wie Roboternavigation oder Objekterkennung. Da die Global-Shutter-Sensoren eine sehr geringe Lichtempfindlichkeit besitzen, ermöglicht dies z.B. Robotern bei ausgeschaltetem Licht sich ohne weitere Probleme durch Räume zu navigieren.

Zusammen mit dem mitgelieferten SDK bietet die RealSense eine kostengünstige Lösung für die Realisierung verschiedenster Projekte an. Ein weiterer Vorteil ist die geringe Grösse, durch diese lässt sie sich problemlos in ein Gesamtsystem integrieren, ohne gross aufzufallen. Dies ist ein entscheidender Punkt gewesen bei meiner Bachelorarbeit.

Abbildung 8: Intel RealSense D435 Tiefenkamera

3.3.1.2 Technische Daten

Features	**Use Enviroment:**	Indoor/Outdoor
	Image Sensor Technology:	Global Shutter, 3µm * 3µm pixel size
	Maximum Range:	Approx. 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
Depth	**Depth Technology:**	Active IR Stereo
	Depth Field of View (FOV):	87˚±3˚ x 58˚±1˚ x 95˚±3˚
	Minimum Depth Distance (Min-Z):	0.105 m
	Depth Output Resolution & Frame Rate:	Up to 1280x720 active stereo depth resolution. Up to 90 fps.
RGB	**RGB Sensor Resolution & Frame Rate:**	1920x1080
	RGB Frame Rate:	30 fps
	RGB Sensor FOV (H x V x D):	69.4˚ x 42.5˚ x 77˚ (+/- 3˚)
Major Components	**Camera Module:**	Intel RealSense Module D430 + RGB Camera
	Vision Processor Board:	Intel RealSense Vision Processor D4
Physical	**Form Factor:**	Camera Peripheral
	Length x Depth x Height:	99 mm x 25 mm x 25 mm
	Connectors:	USB-C* 3.1 Gen 1*
	Mounting Mechanism:	One 1/4-20 UNC thread mounting point, Two M3 thread mounting points.

Tabelle 3: RealSense D435 - Spezifikationen [6]

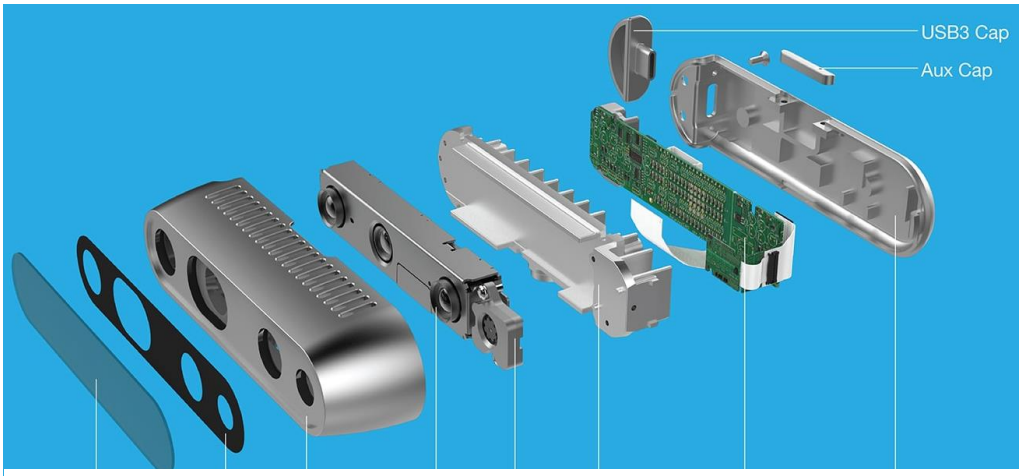


Abbildung 9: RealSense D435 – Aufbau [6]

3.3.2 Stereoskopisches Sehen

Wir Menschen können dank unseren Augen und dem Gehirn räumlich sehen. Bei der Tiefenkamera funktioniert das auf ähnliche Art und Weise. So wie wir Menschen besitz die Tiefenkamera zwei Augen, respektive Kameras (Right & Left Imager). Zusammen mit dem Gehirn / Software wird nun das Räumliche (3D) sehen möglich. Im Fachjargon spricht man hier vom stereoskopischen Sehen [7]. Schauen nun beide Augen auf denselben Gegenstand, so tun sie dies in einem leicht unterschiedlichen Winkel. Auf der Netzhaut des linken Auges fällt somit ein etwas anderes Bild als auf der Netzhaut des rechten Auges. Das kann man ganz einfach selbst an einem Experiment überprüfen. Dazu streckt man den Arm aus und fixiert mit dem Daumen einen Punkt z.b die Ecke des Monitors. Kneift man nun abwechselnd die beiden Augen zu, so scheint es, als würde der Daumen hin- und herspringen -in Relation zum Fixationspunkt (Fixation -> das gezielte Betrachten eines Objekts). Der Gegenstand wird auf der Netzhaut des linken und der des rechten Auges auf dem entsprechenden Stellen abgebildet. Die Bilder von näher gelegenen Objekten, wie der Daumen in unserem Selbstversuch, sind gegenüber diesen korrespondierenden Netzhautstellen versetzt. Aus der sogenannten retinalen Disparität, das heisst dem seitlichen Versatz und dem Unterschied der Abbilder eines Objekts auf den beiden Netzhäuten, bildet das Gehirn die räumliche Tiefe und somit das 3D-Bild. So ähnlich funktioniert auch die Tiefenkamera. Die RealSense D435 besitzt zwei Kameras, die einen leicht unterschiedlichen Winkel haben. Zusammen mit der Prozessor-Unit wird nun das 3D-Bild und somit das räumliche Sehen produziert. Der Infrarot-Projektor, welcher sich zwischen den beiden Kameras befindet, projiziert ein statisches IR-Muster zur Verbesserung der Tiefengenauigkeit in Szenen mit geringer Textur. [8]

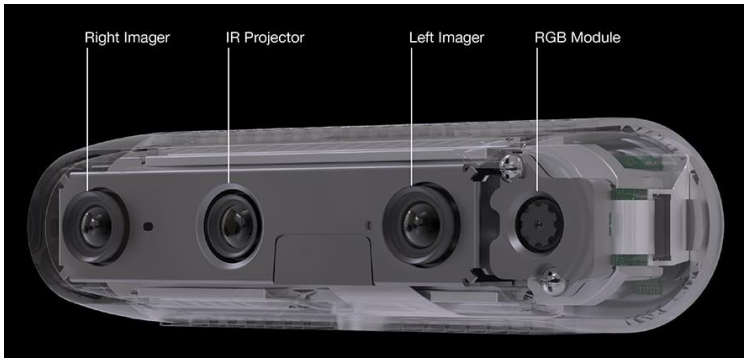


Abbildung 10: RealSense D435 - Aufbau

3.3.3 Nuitrack – Tracking SDK

3.3.3.1 Was ist das Nuitrack SDK?

Nuitrack ist ein Framework für Skelettverfolgung und Gestenerkennung. Das 3D-Tracking ist eine Middleware-Lösung, die von 3DiVi Inc. entwickelt wird. Das Softwareentwicklungsunternehmen wurde im Jahr 2011 gegründet und ist im Bereich Computer Vision, Künstliche Intelligenz und Deep Learning tätig. Das von ihnen entwickelte Nuitrack SDK ist mehrsprachig und plattformübergreifend. [10]

Features:

- Ganzkörper-Skelett Tracking (19 Gelenke)
- 3D Punktewolke
- Benutzermasken
- Gestenerkennung
- Plattformübergreifendes SDK für Android, Windows und Linux
- Unabhängig vom 3D-Sensor (Unterstützt nebst der Intel RealSense auch Kinect v1, Asus Xtion, Orbbec Astra, Orbbec Persee)
- Unitiy und Unreal Engine Plugins
- OpenNI 1.5-kompatibel

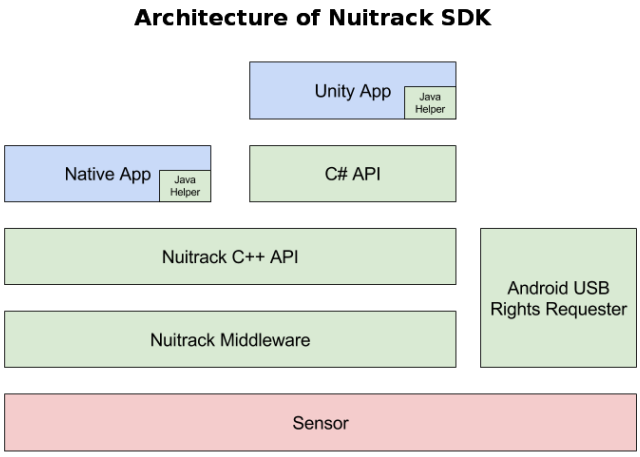


Abbildung 11: Architektur des Nuitrack SDK [9]

Anwendungsbereiche des Frameworks:

- NUI (Natural User Interface) für Windwos/Linux/Android-Plattformen
- Spiele und Training (Fitness, Tanzstunden)
- Medizinische Rehabilitation
- Positions- und Ganzkörpertracking für VR
- Zielgruppenanalyse
- Robot Vision

Abbildung 12: Nuitrack Logo

3.3.3.2 Nuitrack-Skelettsystem

Mit Nuitrack können bis zu sechs Personen, also Skelette, gleichzeitig verfolgt werden. Allerdings werden standardmässig nur 2 getrackt. Sollen sechs Personen getrackt werden, so muss dies konfigurativ geändert werden. Jedes Nuitrack-Skelett hat 19 Gelenke. Jedes Gelenk hat eine Position und eine Ausrichtung. Auf der unterstehenden Abbildung ist das Grundgerüst zu sehen. Wichtig für meine Arbeit ist der *Joint Head*.

Abbildung 13: Nuitrack Skelett [10]

3.3.3.3 Warum Nuitrack?

Die Intel RealSense bietet ein eigenes SDK an. Warum habe ich mich für das Nuitrack SDK entschieden? Das Nuitrack SDK ist sehr einfach zu implementieren und liefert schnell und unkompliziert die gewünschten Ergebnisse. Ein weiterer Vorteil welches das SDK bietet, ist, dass es plattformunabhängig einsetzbar ist und auch die gängigsten Tiefenkameras unterstützt. So ist es ohne Probleme möglich, statt die Intel RealSense D435, die ich einsetze, mit einer anderen Kamera (z.B. die Kinect) das Projekt zu realisieren. Ein weiterer Pluspunkt für das Nuitrack SDK ist die gute Online-Dokumentation, die angeboten wird. Auch können zahlreiche Beispielprojekte anhand von Tutorials nachgebaut werden. Ein weiterer positiver Aspekt des SDK ist die Anbindung an Unity. Diesen Pluspunkten steht der Nachteil gegenüber, dass es ohne Lizenz «nur» 3 Minuten zu Verfügung steht und danach die Personen/Skeletts nicht mehr getrackt werden.

3.4 Geometrische Operationen

Bei Geometrischen Operationen werden die Pixel Positionen innerhalb eines Bildes verändert. Bekannte geometrische Operationen sind das Skalieren, Verschieben, Rotieren und Scheren von Bildern. Diese Operationen gehören in die Kategorie der affinen Transformationen. Zudem gibt es noch die projektiven Abbildungen. Diese werden verwendet, um perspektivische Verzerrungen zu korrigieren. [11]

3.4.1 Projektive Abbildung

Mittels der projektiven Abbildung lässt sich zum Beispiel ein hohes Gebäude, das von unten fotografiert wurde, korrekt entzerren. Die Fassade wirkt auf dem Foto eher trapezförmig statt rechteckig oder sogar quadratisch, wie sie in echt ist. Um die projektive Abbildung nun anzuwenden, braucht man 4 Punktpaare auf dem Ausgangsbild und 4 Punktpaare auf dem neuen Zielbild. [12]

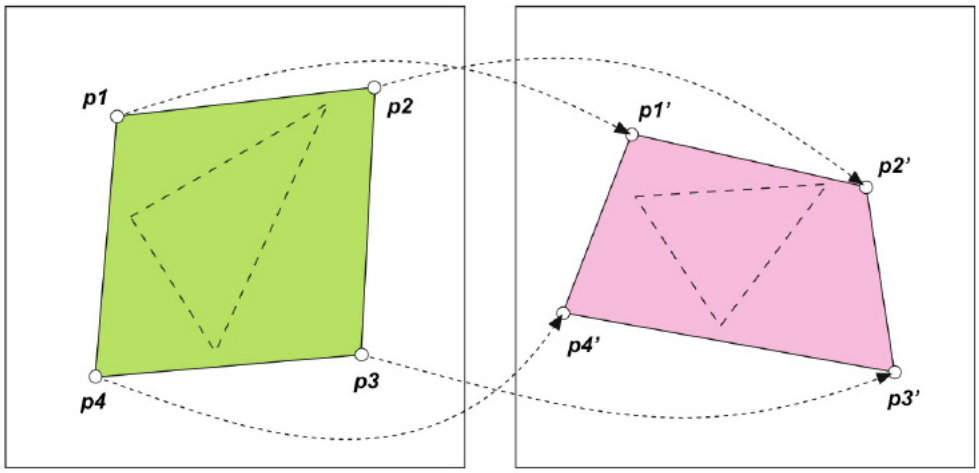


Abbildung 14: projektive Abbildung [12]

3.5 Perspektivische Projektion

Da in 3D-Computergrafik die virtuelle Kamera meist der Position des Kopfes des Benutzers folgt, ist eine **On-Axis Perspektive** nicht ideal, bei einer **Off-Axis Perspektive** hingegen werden die beliebige Kamerapositionen berücksichtig. Aber zuerst mal zu den Erklärungen der beiden Perspektivprojektionen.

3.5.1 On-Axis-Perspektive

Bei einer sogenannten «On-Axis-Perspektive» schauen wir mittig in die Projektionsebene. Das Auge, respektive der Betrachter P_e, schaut wie in der Abbildung 14 zu sehen zentriert auf die Fläche (die gestrichelte Linie vom Punkt P_e aus). Es entsteht ein pyramidenförmiges Volumen, welches perfekt symmetrisch ist. Der Schnittpunkt, der aus der gestrichelten Linie und der Projektionsfläche entsteht bezeichnet man als Koordinatenursprung (relativ gesehen zum lokalen Koordinatensystem).[13]

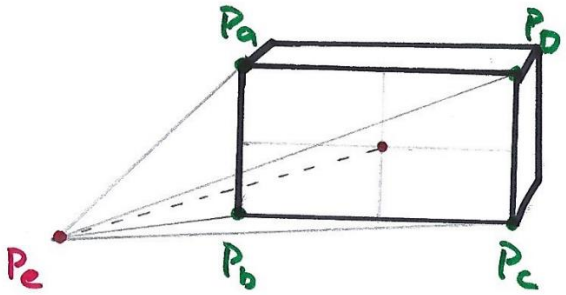


Abbildung 15: Skizze On-Axis-Perspektive

3.5.2 Off-Axis-Perspektive

Bei der «Off-Axis-Perspektive» befinden wir uns nicht mehr auf der Achse, sondern bewegen uns von der Mitte weg, somit schaut das Auge, respektive der Betrachter P_e nun nicht mehr mittig auf die Projektionsfläche:

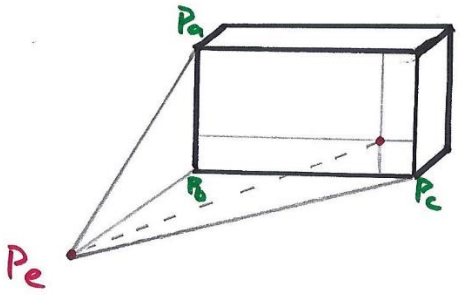


Abbildung 16: Skizze Off-Axis-Perspektive

Wir sehen also, dass die gestrichelte Linie nicht mehr mittig auf die Projektionsfläche trifft. Auch ist die Pyramide nicht mehr perfekt symmetrisch. Der Koordinatenursprung bleibt aber weiterhin der Schnittpunkt. Somit sieht man das sich dieser mitbewegt, wenn sich der Betrachter bewegt. [14]

3.5.3 Projektionsfläche bestimmen

In einen ersten Schritt werden die Punkte P_a, P_b sowie P_c auf der Projektionsfläche bestimmt. Wie in der unterstehenden Abbildung zu erkennen ist, sind dies die Ecken oben links, unten links sowie die Ecke unten rechts des grünen Gitters (Projektionsebene). Zusammen ergeben diese Punkte die Grösse, das Seitenverhältnis sowie die Position und die Ausrichtung der Projektionsfläche.

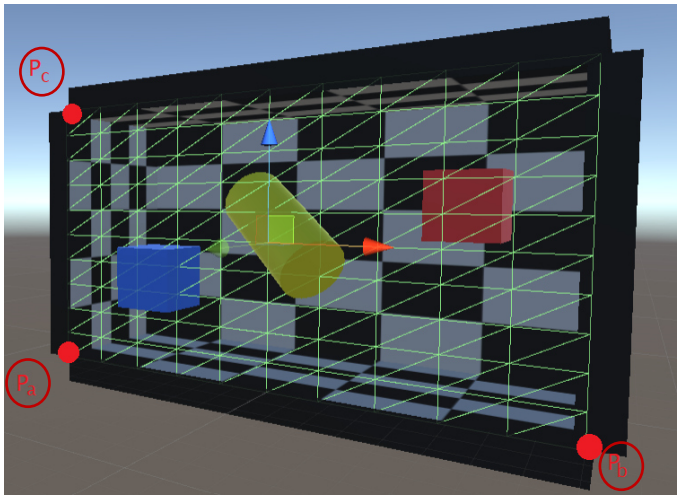


Abbildung 17: Eckpunkte auf der Projektionsfläche

Mittels diesen drei Punkten können wir nun die Orthonormalbasis der Projektionsfläche berechnen. Doch was war nochmal die Orthonormalbasis [16]? In einem 3D-Raum sind dies 3 Vektoren, die alle senkrecht zueinanderstehen und alle die Länge eins haben. Im lokalen Raum zeigt nun der Vektor V_u nach oben, der Vektor V_r nach rechts und schliesslich der Vektor V_n nach vorne respektive aus der Projektionsfläche hinaus. Ähnlich wie bei einem kartesischen 3D Koordinatensystem, wo man mittels den Achsen X, Y und Z Punkte in einem 3D-Raum beschreiben kann, können wir mit den lokalen Achsen V_u, V_r und V_n Punkte beschreiben, die in unserem lokalen Koordinatensystem vorhanden sind. Also Punkte, die relativ zu unserer Projektionsfläche stehen.

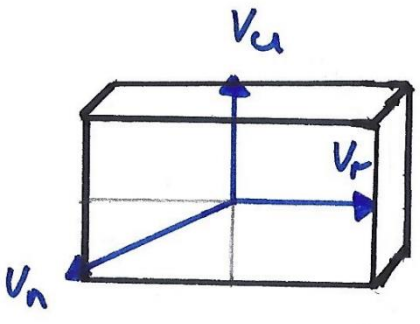


Abbildung 18: Skizze Orthonormale

$$V_r = \frac{P_b - P_a}{||P_b - P_a||} \qquad V_u = \frac{P_c - P_a}{||P_c - P_a||} \qquad V_n = \frac{V_r \times V_u}{||V_r \times V_u||}$$

3.5.4 Was ist das View Frustum?

Das View Frustum oder auch Kegelstumpf genannt entsteht, wenn man bei einer Pyramide der obere Teil abschneidet. In der Computergrafik ist dies die mathematische Abbildung eines 3D-Universums auf einen Bildschirm. Dadurch das in der Fluchtpunktperspektive jedes Objekt in der Tiefe kleiner wird, erhält man den Eindruck das man auf das Ende (den Stumpf) eines Kegels schaut. Der Pyramidenstumpf stellt in der Computergrafik das Volumen dar, welches vom Betrachter gesehen und von der perspektivischen Kamera gerendert werden kann. Das Ganze kann mit einem kleinen Experiment selber getestet werden. Hält man z.B. ein Kugelschreiber mittig und senkrecht vor einer Kamera, so sieht man auf dem Bild nur einen runden Punkt. Bewegt man den Kugelschreiber nun langsam nach oben, so sieht man den Kugelschreiber von unten (vorausgesetzt er ist immer noch senkrecht). Neigt man den Kugelschreiber langsam nach oben, so sieht man wieder nur einen Punkt. Sobald man den Kugelschreiber nun aber weiter nach oben bewegt, so verschwindet er am oberen Bildrand und ist schliesslich für die Kamera nicht mehr sichtbar. Dieses kleine Experiment soll aufzeigen, dass jeder Punkt in einem Kamerabild auch tatsächlich zu einer Linie in Weltkoordinatensystem gehört.

Alles was sich ausserhalb der gestrichelten Linien befindet (siehe Abbildung 18), wird von der Unity Kamera nicht erfasst und somit auch nicht gerendert. Auch nicht sichtbar und somit nicht gerendert wird der Bereich der nach der «Far clipping plane» kommt sowie auch der Bereich der sich vor der «Near clipping plane» befindet. Würde man die Linien bis zur Kamera zurückverfolgen, bis die Punkte schliesslich zu einem einzigen Punkt zusammenlaufen, so erreicht man den Mittelpunkt der Perspektive. Das View Frustum wird durch die Parameter **t** = top, **l** = left, **r** = right, **b** = bottom, **n** = near und **f** = far definiert: [17]

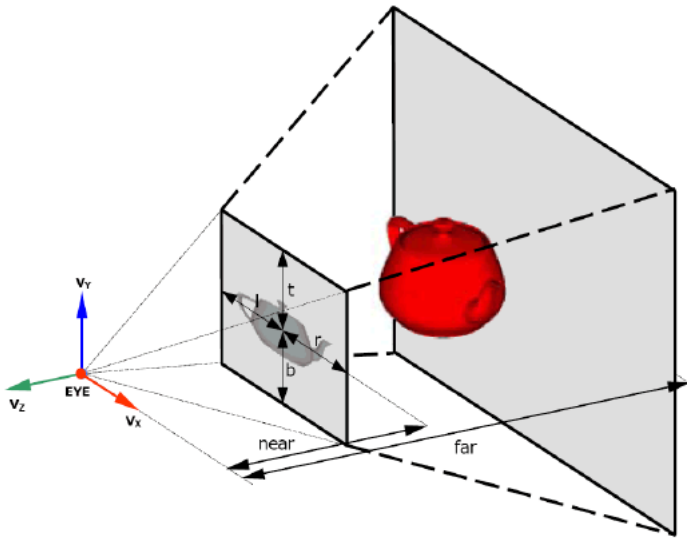


Abbildung 19: far and near clippling plane [18]

3.5.5 Koordinatensysteme

Um ein 3D modelliertes Objekt auf dem Bildschirm (also 2D) anzeigen zu können, sind nachfolgende Matrixtransformationen nötig. Wir gehen davon aus das wir in Blender [19] ein Objekt erstellt haben (z. B. ein Würfel).

1. Im **Model Space** lassen sich die Koordinaten dieses Objektes durch das lokale Koordinatensystem bestimmen. Somit befinden sich alle Eckpunkte des Modells im Model Space: Sie sind alle lokal für das Objekt.
2. Im nächsten Schritt werden die lokalen Koordinaten in Weltkoordinaten umgewandelt, dies erfolgt mit der Modelmatrix. Nach der Umwandlung befinden sich das Objekt im **World Space**. Somit kann man das Objekt nun «in der Welt» zu der es gehört, platzieren und auch ausrichten.
3. Mittels der ViewMatrix wird der World Space in die **Camera Space** umgewandelt. Somit werden die Koordinaten aus der Sicht der Kamera, respektive des Betrachters betrachtet.

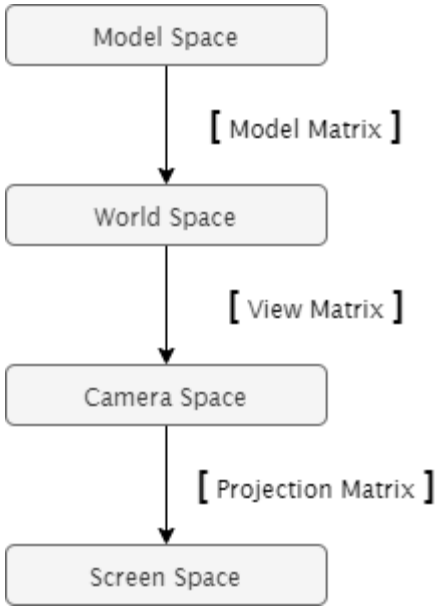


Abbildung 20: Matrixtransformationen

4. Durch die Projektionsmatrix (Projection Matrix) werden die Koordinaten in den **Screen Space** umgewandelt. Nun befinden wir uns im Projektionsraum. Die perspektivische Projektionsmatrix hilft uns dabei den Pyramidenstumpf / View Frustum zu bestimmen:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Dadurch wird der Bereich definiert, welcher wir sehen können. Alle Koordinaten, die sich ausserhalb des Stumpfes befinden, werden somit abgeschnitten und nicht dargestellt. Zudem befinden wir uns nun im 2D-Raum. [20]

4 Projektvorgehen

Im nachfolgenden Kapitel wird das Projektvorgehen beschrieben.

- Nach welchem Projektvorgehen ging ich vor?
- Wie organisierte ich meine Arbeit?
- Wie wurden die Arbeiten zeitlich organisiert?
- Was wurde realisiert?

4.1 Projektvorgehensmodell

Die Bachelorarbeit wurde in mehrere Arbeitspakete aufgeteilt. Die Erarbeitung der Lösung und damit auch die Erstellung des Softwarecodes, wurde in Iterationen vorgenommen. So konnte der Code kontinuierlich verbessert werden. Bei den Iterationen ging ich Scrum [21] ähnlich vor. So hatte eine Iteration eine Länge von 3 Wochen und jeden Dienstag gab es zusammen mit Herrn Hudritsch ein Weekly. In diesen wurde der aktuelle Stand besprochen und wo es Probleme und oder Schwierigkeiten gab. Im Idealfall wurden diese bereits während des Gesprächs geklärt oder es gab einen kleinen «schubs» in die richtige Richtung. Wurde eine zu Beginn der Iteration definierte Arbeit nicht fertig, so wurde diese in die nächste Iteration mitgenommen.

4.2 Versionskontrolle

Für den Sourcecode wurde auf GitHub [22] ein Repository erstellt. Für jedes neue Feature wurde ein neuer Branch erstellt. Somit wurde ein sauberes Tracking der Arbeit gewährleistet und da Git ein verteiltes Versionierungssystem ist, diente GitHub auch noch als Backup für meine Arbeiten.

4.3 Arbeitsplatz

Während meiner ganzen Bachelorarbeit hatte ich jeweils dienstags einen Arbeitsplatz im CPVR-Lab. Dort hatte ich auch einen grossen Fernseher zu Verfügung, wo ich die Umsetzung testen konnte. Auch erleichtere dies die Kommunikation mit Herrn Hudritsch, da er nur ein paar Meter entfernt vor mir Sass.

An den anderen Tagen arbeitete ich meistens von zu Hause aus. Da ich das Studium berufsbegleitend absolviere, arbeitete ich vor allem an den Wochenenden an der Bachelorarbeit. Gegen Ende der Arbeit traf ich mich mit anderen Mitstudenten im Gründerzentrum in Bern. Dort fand jeweils ein interessanter Austausch statt über die Arbeit und vor allem über die Dokumentation, das Poster oder den Film.

4.4 Zeitplanung

Gemäss dem Bologna-System wird für die Bachelorarbeit mit einem gesamten Zeitaufwand von 360h gerechnet [23]. Ich habe mir diese wie folgt aufgeteilt:

Arbeitspakete	Zeitaufwand
Umgebung aufsetzen	10h
1. Iteation	65h
2. Iteration	65h
3. Iteration	65h
4. Iteration	65h
Projektabschluss	90h

Tabelle 4: Zeiteinteilung der Bachelorarbeit

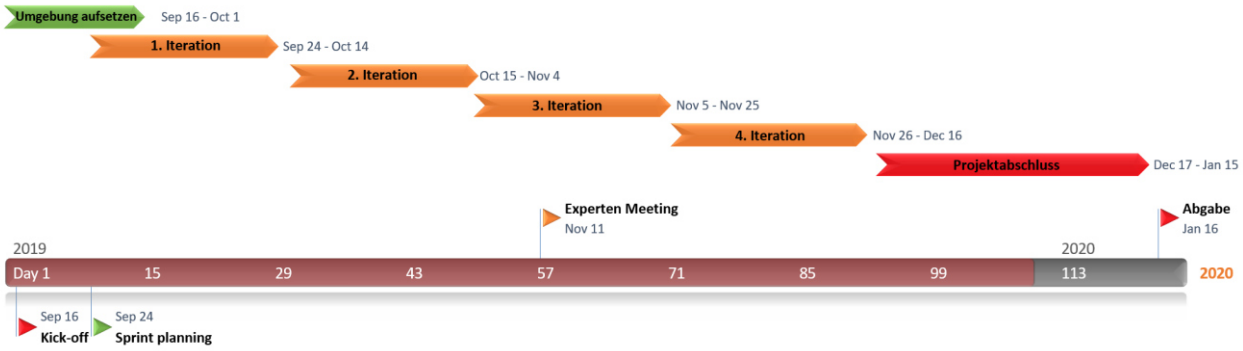


Abbildung 21: Zeitstrahl der Bachelorarbeit

4.5 Umgebung aufsetzen

4.5.1 Ziele

Ziel des ersten Arbeitspaketes *Umgebung aufsetzten* war es die Grundlagen zu schaffen, damit an einer erfolgreichen Bachelorarbeit nichts mehr im Wege stand. So wurde der Arbeitsplatz eingerichtet, eine grobe Planung für die Thesis erstellt und zusammen mit Herrn Hudritsch nochmals die Projektziele besprochen und abgestimmt.

4.5.2 Umsetzung

Als Erstes wurden die Entwicklungsumgebungen, also Unity sowie Visual Studio auf die neuste Softwareversion aktualisiert. Den Arbeitsplatz, den ich im CPVR-Lab erhielt, habe ich für meine Bedürfnisse eingerichtet, damit ich jeden Dienstag dort ungestört arbeiten konnte. Auch zum Arbeitsplatz einrichten gehörte das Anlegen eines Github Repository, damit ich meine Arbeit stets sichern konnte, falls mein Notebook aus welchen Gründen auch immer seinen Geist aufgeben sollte oder ich auch mit meinem zweiten Notebook auf meine Arbeiten zurückgreifen könnte. Weiter wurde mit Herrn Hudritsch das Projektziel als solches nochmals angeschaut und daraus Etappen definiert:

- Homographie Matrix mittels MatLab (Iteration 1)
- Homographie Matrix mittels C# (Iteration 2)
- Shader programmierung (Iteration 3)
- Zusammenführen der ganzen Arbeiten (Iteration 4)
- Projektabschluss (Dokumentieren, Präsentation vorbereiten, Film, Poster und was noch alles zu einer Bachelorarbeit dazu gehört…)

Nach all diesen Arbeiten konnte also nun die Bachelorarbeit inhaltlich in Angriff genommen werden.

4.5.3 Ergebnis

Die Umgebung konnte funktionsfähig und termingerecht aufgesetzt werden. Die Etappen und der Vorgehensplan wurden gemeinsam definiert und abgestimmt.

4.6 Erste Iteration

Die Suche nach der idealen Lösung.

4.6.1 Ziele

In der ersten Entwicklungsiteration war das Hauptziel die Homographie-Matrix und deren Funktionsweise zu verstehen. Hierfür wurde vom Prototyp ein Printscreen gemacht als ein Betrachter vor dem Fernseher stand. Mittels der Homographie Matrix sollte das verzerrte Bild entzerrt werden.

4.6.2 Umsetzung

Der Prototyp simuliert einen Fernseher, auf dem sich auf der oberen Kante eine Tiefenkamera befindet, wie im Kapitel 2.4 Prototyp beschrieben. Startet man das Unity-Projekt, erkennt die RealSense D435 den Betrachter im Raum und ermittelt seine Position. Schaut der Betrachter von unten links auf den Fernseher, so entsteht das nachfolgende Bild.

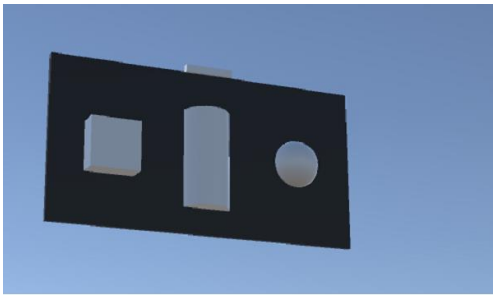


Abbildung 22: Ausgangsbild für Matlab

Nun galt es dieses mittels der Homographie Matrix (sieh Kapitel 3.4.1 Projektive Abbildung) zu entzerren. Nachfolgend zuerst die Erklärung der Schritte zum Matlab Script (Listening 1):

1. In einem ersten Schritt wird der Printscreen geladen und angzeigt
2. Als zweitens wurden die «movingPoints» definiert. Die X und Y Werte, von jeder Ecke des simulierten Fernsehers, erhielt ich mit Hilfe von Windows Paint
3. In einem dritten Schritt wurden die «fixedPoints» definiert. Hierfür wurden die vier Ecken des Bildes gewählt
4. Die Methode *fitgeotrans* führt schliesslich die projektive Abbildung durch (Die «movingPoints» werden zu den «fixedPoints» gezogen, hierbei werden für die restlichen Pixel mittels Homographie Matrix der neue Pixelwert berechnet.)
5. Da das Zielbild genau gleich gross sein soll wie das eingangs Bild, werden dessen X und Y Werte der vier Bildecken in die Variable *RA* abgespeichert
6. Am Ende wird das neue Bild generiert und ausgegeben

```
clear all;
close all;
clc;

img = imread('TV_Screen.png');
imshow(img);

movingPoints = [62 59; 420 112; 425 300; 50 265];
fixedPoints = [0 0; size(img,2) 0; size(img,2) size(img,1); 0 size(img,1)];

tform = fitgeotrans(movingPoints, fixedPoints, 'Projective');

RA = imref2d([size(img,1) size(img,2)], [1 size(img,2)], [1 size(img,1)]);

[out,r] = imwarp(img, tform, 'OutputView', RA);
figure, imshow(out,r);
```

Listing 1: Matlab script - FindHomography

4.6.3 Ergebnis

Die Erstellung des doch relativ kurzen Codes stellte sich als grosse Herausforderung dar. So hatte ich zu Beginn Mühe, welche Punkte ich als die «fixedPoints» und welche ich als die «movingPoints» definieren musste. Vertauscht man beide, so entstehen ganz lustige Bilder. Auch hatte ich zu Beginn Mühe, den Koordinatenursprung zu finden. Doch mit der Hilfe von Windwos Paint klappte dies danach ohne Probleme. Schliesslich folgte die dritte Schwierigkeit. In welcher Reihenfolge müssen die Punkte definiert sein? Von oben links und danach im Uhrzeigersinn oder gegen den Uhrzeigersinn…

Schlussendlich konnten alle Probleme und Schwierigkeiten behoben werden und das MatLab Skript lieferte das gewünschte Ergebnis:

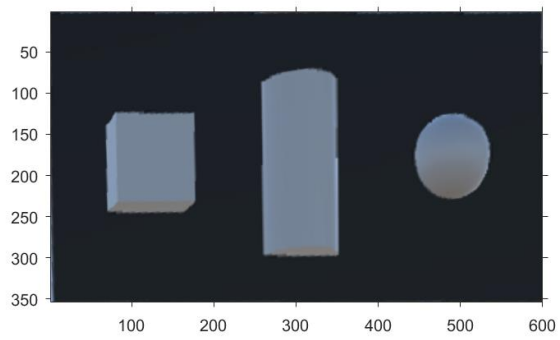


Abbildung 23: Zielbild

So sollte also das Bild aussehen, welches der Betrachter sieht, wenn er von unten links auf den Fernseher schaut.

4.7 Zweite Iteration

Aus lauter Bäumen den Wald nicht mehr sehen.

4.7.1 Ziele

Ziel der zweiten Iteration war es das Beispiel von Matlab in C# nach zu programmieren, da in Unity C# als Programmiersprache genutzt wird.

4.7.2 Umsetzung

Für die Implementation der Homographie Matrix in C# verwendete ich dasselbe Ausgangsbild wie in der ersten Iteration bei der Implementation mittels MatLab. Die Teilschritte sind eigentlich identisch wie beim Matlab Skript:

1. Zuerst wird das Bild geladen
2. Danach werden die «movingPoints», respektive heisst die variable hier source, definiert. Damit OpenCV das Array aus Point2f weiterverarbeiten kann, müssen diese mit der Methode MathOfPoint2f.FromArray(...) umgewandelt werden. Es entsteht dabei die neue Variable src.
3. Anschliessend werden die «fixedPoints», respektive hier die Variable destination, definiert. Auch diese müssen zur Weiterverarbeitung umgewandelt werden
4. Anschliessend berechnet die Methode FindHomography(...), welche von OpenCV [24] zur Verfügung gestellt wird, die Homographie Matrix der 8 Punktpaaren aus.
5. Mittels der Methode WarpPerspective(...) wird die Homographie Matrix auf das eingangs Bild angewendet und es werden die neuen Pixelpositionen für das Zielbild berechnet

```
        public static void TransformTv()
        {
            Mat img =
Cv2.ImRead(@"C:\prog\consoleHomography\ConsoleApp1\ConsoleApp1\images\input\in.jpg");
            Mat imgOut = new Mat();

            Point2f[] source =
            {
                new Point2f(62f,59f),
                new Point2f(420f,112f),
                new Point2f(425f,300f),
                new Point2f(50f,265f)
            };
            MatOfPoint2f src = MatOfPoint2f.FromArray(source);

            Point2f[] destination =
            {
                new Point2f(0f,0f),
                new Point2f(img.Width,0f),
                new Point2f(img.Width,img.Height),
                new Point2f(0f,img.Height)
            };
            MatOfPoint2f dest = MatOfPoint2f.FromArray(destination);

            Mat m = Cv2.FindHomography(src, dest);
            Cv2.WarpPerspective(img, imgOut, m, img.Size());

            string imgName = $"{DateTime.Now:dd_MM_yyyy_HH_mm_ss}_out.jpg";

Cv2.ImWrite(@"C:\prog\consoleHomography\ConsoleApp1\ConsoleApp1\images\output\" +
imgName, imgOut);
        }
```

Listing 2: C# Script - FindHomography

4.7.3 Ergebnis

Wie man aus der Abbildung 25 sehen kann, ist das Endergebnis bei der C# und OpenCV Lösung identisch zum Zielbild in der Abbildung 23, wo die Homographie Matrix mittels MatLab realisiert wurde.

Abbildung 24: Zielbild (C#)

Die Schwierigkeit hier allerdings war die fehlende Dokumentation von OpenCV in C#. Da OpenCV für die Programmiersprachen C, C++ und Python geschrieben wurde, gibt es zwar für diese Sprachen sehr viele und gute Dokumentation oder Tutorials aber für die Wrapper Libraries in C# nur sehr wenige. So musste in dieser Iteration viel Recherche betrieben werden was für Parameter die Methode FindHomography(…) benötigt. Da wie anfangs angenommen, ein Array bestehend aus Point2f nicht reichte, musste ich hier ein bisschen ausprobieren und herumexperimentieren, bis ich die richtige Klasse fand.

Auch hatte ich zu Beginn die ImageStream Klasse von C# selber verwendet. Allerdings funktioniert diese nur zum Laden des Bildes. Doch leider kann ich die Homographie Matrix, welche ich mittels des OpenCV Wrapper gefunden habe, auf das Bild nicht anwenden. Ich muss das Bild mit derselben Library öffnen und bearbeiten, mit welcher ich die Homographie Matrix definiert hatte…

Schlussendlich konnte ich die zweite Iteration gleichwohl erfolgreich abschliessen.

4.8 Dritte Iteration

Mehrere Wege führen nach Rom.

4.8.1 Ziele

Ursprünglich war das Ziel der dritten Iteration das Bild, das aus der zweiten Iteration entstand, mittels Shader Programmierung zu skalieren, damit dieses der Gesamtgrösse des Fernsehers entspricht. Doch schliesslich kam alles ein bisschen anders. Da ich grosse Mühe hatte, den Lösungsansatz, der zusammen mit Herrn Hudritsch besprochen wurde, umzusetzen suchte ich einen anderen Weg.

4.8.2 Umsetzung

Nach ein paar Recherchen im Internet über ähnliche Arbeiten tauchte dort immer wieder der Begriff Projektionsmatrix auf. Ich schaute mir die Projektionsmatrix einmal genauer an und kam so schrittweise näher zum Ziel:

1. Zuerst muss wie bereits im Kapitel *3.5.3 Projektionsfläche bestimmen* beschrieben die 3 Eckpunkte bestummen werden (oben und unten links, sowie der unten rechts):

```
Vector3 pa = ProjectionScreen.transform.TransformPoint(new Vector3(-5.0f, 0.0f, -5.0f));
Vector3 pb = ProjectionScreen.transform.TransformPoint(new Vector3(5.0f, 0.0f, -5.0f));
Vector3 pc = ProjectionScreen.transform.TransformPoint(new Vector3(-5.0f, 0.0f, 5.0f));
Vector3 pe = transform.position;
```

Listing 3: transform the edges from the projectionScreen in world coordinates

   Die Punkte / Koordinaten P_a, P_b und P_c transformieren wir hierbei vom lokalen Koordinatensystem ins Weltkoordinatensystem. Der Punkt P_e ergibt sich aus der Position der Kamera, also des Betrachters (Die Kamera übernimmt in Unity die Position des Betrachters).
2. Nun berechne ich ebenfalls, wie im Kapitel *3.5.3 Projektionsfläche bestimmen* beschrieben ist, die Orthonormalen für die 3 Eckpunkte

```
Vector3 vr = (pb - pa).normalized;
Vector3 vu = (pc - pa).normalized;
Vector3 vn = -Vector3.Cross(vr, vu).normalized;
```

Listing 4: projectionScreen local axes

3. Weiter geht es mit der Berechnung des Kegelstumpfes respektive das sogenannte View-Frustum (Kapitel *3.5.4 Was ist das View Frustum?*) Hierfür müssen wir zuerst die Vektoren, welche die 3 Eckpunkten aus der Projektionsfläche bestimmen, berechnen:

```
Vector3 va = pa - pe;
Vector3 vb = pb - pe;
Vector3 vc = pc - pe;
```

Listing 5: the vectors of the edges from the projectionScreen

Zusätzlich brauchen wir noch die Near, respektive Far clipping plane:

```
float n = _cam.nearClipPlane;
float f = _cam.farClipPlane;
```

Listing 6: define near and far clipping plane

Nachdem wir noch die Distanz vom Betrachter bis hin zur Projektionsfläche bestimmt haben, könne wir die Parameter berechnen, welche das View Frustum beschreiben:

```
float d = -Vector3.Dot(va, vn);
```

Listing 7: distance eye to projectionsscreen

```
float l = Vector3.Dot(vr, va) * n / d;
float r = Vector3.Dot(vr, vb) * n / d;
float b = Vector3.Dot(vu, va) * n / d;
float t = Vector3.Dot(vu, vc) * n / d;
```

Listing 8: calculate the parameters for the projectionMatrix

4. Die letzten 3 schritte, die wir noch umsetzten müssen, sind die Transformationen. Dadurch wird einerseits bestimmt, was die Unity-Kamera, also der Betrachter, sehen kann und anderseits werden die projizierten Punkte auf ihrer endgültigen Position auf dem Bildschirm abgebildet. Hierfür brauchen wir die Matrixtransformationen, welche im *Kapitel 3.5.5 Koordinatensysteme* beschrieben sind.

   Zuerst erstellen wir mittels der Projektionsmatrix das Anzeigevolumen, also der Kegelstumpf der im *Kapitel 3.5.4 Was ist das View Frustum* beschrieben:

```
Matrix4x4 projectionMatrix = new Matrix4x4();
projectionMatrix.m00 = 2.0f * n / (r - l);
projectionMatrix.m01 = 0.0f;
projectionMatrix.m02 = (r + l) / (r - l);
projectionMatrix.m03 = 0.0f;

projectionMatrix.m10 = 0.0f;
projectionMatrix.m11 = 2.0f * n / (t - b);
projectionMatrix.m12 = (t + b) / (t - b);
projectionMatrix.m13 = 0.0f;

projectionMatrix.m20 = 0.0f;
projectionMatrix.m21 = 0.0f;
projectionMatrix.m22 = -((f + n) / (f - n));
projectionMatrix.m23 = -(2.0f * f * n / (f - n));

projectionMatrix.m30 = 0.0f;
projectionMatrix.m31 = 0.0f;
projectionMatrix.m32 = -1.0f;
projectionMatrix.m33 = 0.0f;
```

Listing 9: set the projectionMatrix

5. Anschliessend müssen wir noch eine Rotation machen, da das View Frustum aktuell von der Projektionsfläche wegschaut:

```
Matrix4x4 rotationMatrix = new Matrix4x4();
rotationMatrix.m00 = vr.x;
rotationMatrix.m01 = vr.y;
rotationMatrix.m02 = vr.z;
rotationMatrix.m03 = 0.0f;

rotationMatrix.m10 = vu.x;
rotationMatrix.m11 = vu.y;
rotationMatrix.m12 = vu.z;
rotationMatrix.m13 = 0.0f;

rotationMatrix.m20 = vn.x;
rotationMatrix.m21 = vn.y;
rotationMatrix.m22 = vn.z;
rotationMatrix.m23 = 0.0f;

rotationMatrix.m30 = 0.0f;
rotationMatrix.m31 = 0.0f;
rotationMatrix.m32 = 0.0f;
rotationMatrix.m33 = 1.0f;
```

Listing 10: World to Camera space (View Matrix)

6. In einem letzten Schritt müssen wir die Spitze der Pyramide zum Auge führen, dies erreichen wir mit folgender translationMatrix:

```
Matrix4x4 translationMatrix = new Matrix4x4();
translationMatrix.m00 = 1.0f;
translationMatrix.m01 = 0.0f;
translationMatrix.m02 = 0.0f;
translationMatrix.m03 = -pe.x;

translationMatrix.m10 = 0.0f;
translationMatrix.m11 = 1.0f;
translationMatrix.m12 = 0.0f;
translationMatrix.m13 = -pe.y;

translationMatrix.m20 = 0.0f;
translationMatrix.m21 = 0.0f;
translationMatrix.m22 = 1.0f;
translationMatrix.m23 = -pe.z;

translationMatrix.m30 = 0.0f;
translationMatrix.m31 = 0.0f;
translationMatrix.m32 = 0.0f;
translationMatrix.m33 = 1.0f;
```

Listing 11: Model to World space (ModelMatrix)

7. Im allerletzten Schritt müssen wir die ganze Matrixtransformationen noch der Unity-Kamera anhängen:

```
_cam.worldToCameraMatrix = rotationMatrix * translationMatrix;
_cam.projectionMatrix = projectionMatrix;
```

Listing 12: add transformation to unity cam

4.8.3 Ergebnis

Et voilà, wenn wir uns nun im Raum bewegen, so stimmt die Perspektive des Bildes, das auf dem Fernseher angezeigt wird, relativ zu unserer Position. Allerdings ist zu erwähnen, dass sich der Betrachter zwar im Raum bewegt, aber die Mathematik der perspektivischen Projektion uns dies verbietet. Die Kamera ist am Ursprung gefangen. Das heisst, wenn sich der Betrachter um zwei Schritte nach links bewegt, so bewegt in der 3D-Computergrafik die Kamera sich nicht mit. Stattdessen wird die ganze Welt um zwei Einheiten nach Links verschoben.

Leider konnte ich in dieser Iteration nicht den Lösungsansatz Umsetzen, der eigentlich eingeplant war. Aber trotzdem war ich ziemlich froh, dass ich eine Lösung gefunden und erarbeiten konnte, die mir das gewünschte Ergebnis lieferte. Da ich schon während der ersten beiden Iterationen mit der Vorstellung des mit Herrn Hudritsch besprochene und abgestimmte Lösungsansatzes Mühe hatte, war ich ziemlich froh, als ich auf das Dokument von Robert Kooima – Generalized Perspective Projection [25] gestossen bin. Dieses Dokument half mir sehr, einen Einstieg zu finden und schlussendlich einen anderen Lösungsansatz zu erarbeiten.

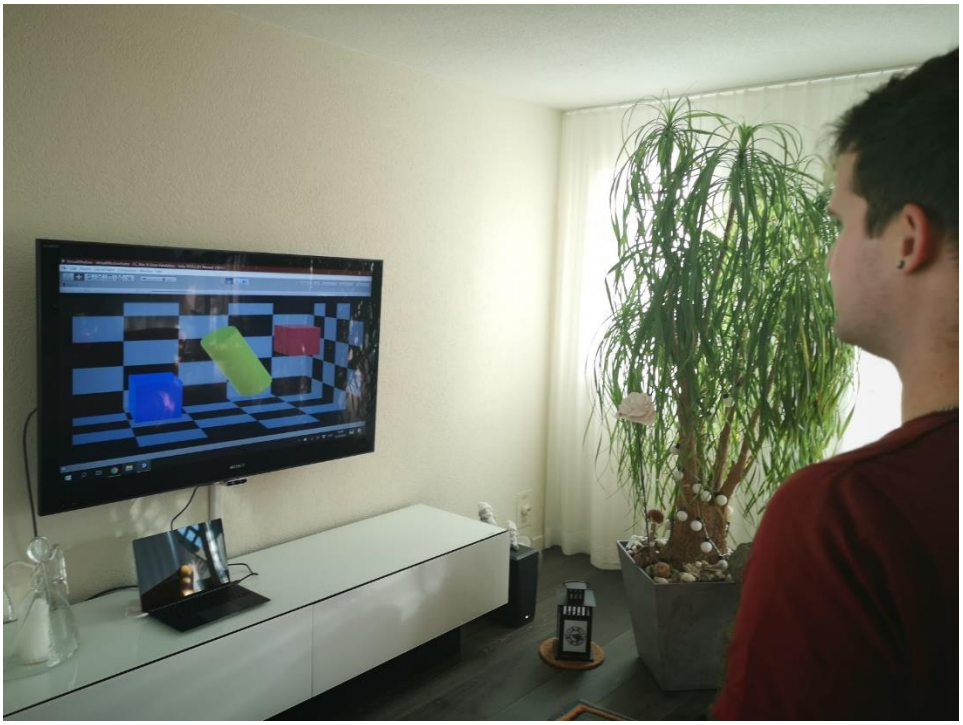


Abbildung 25: Betrachter in Action

4.9 Vierte Iteration

Erstens kommt es anders, zweitens als man denkt!

4.9.1 Ziele

Hätte ich den ersten Lösungsansatz verstanden und implementiert, so hätte ich in der vierten und letzten Iteration alle erarbeite Komponenten zusammengeführt und noch die letzten Anpassungen vorgenommen. Da aber alles ein bisschen anders kam als geplant und ich mich nun für einen anderen Lösungsansatz entschieden habe, erfolgte hier das erweitern der Szene, die der Betrachter anschauen soll.

4.9.2 Umsetzung

Die gewählte Szene für das Endprodukt war eine Box in einem Schachbrettmuster, in die der Betrachter schauen soll. In dieser Box gibt es diverse geometrische Formen. Genauer gesagt einen gelben Zylinder, der sich auch um die eigene Achse drehen kann, eine blaues sowie ein rotes Quadrat. Alle geometrischen Formen sind leicht versetzt. Die ganze Box kann man auch ausblenden und es erscheinen zwei Schneemänner. Der schwierigste Part dieser Iteration war es, die Box so zu platzieren damit, wenn sich der Betrachter auf der Höhe der linken oder rechte Kante des Fernsehers befindet, er auch die Kanten der linken oder eben rechten Seite der Box sieht und nicht noch eine schraffierte Wand. Zudem wurde in dieser Iteration das gesamte System getestet. Funktioniert alles wie gewünscht? Was meint der Tester dazu?

4.9.3 Ergebnis

In der vierten und letzten Iteration entstand zusammen mit Blender und Unity die Szene, die der Betrachter schlussendlich anschauen kann. Ein kurzer Feldtest bestätige mir, dass mit dem zweiten Lösungsansatz die Projektziele erreicht wurden. Sicherlich wäre in dieser Zeit noch eine grössere und vielleicht interessantere Szenen Modellierung möglich gewesen. Jedoch investierte ich nochmals ein bisschen Zeit in den ersten Lösungsansatz. So schaute ich mir diverse Tutorials an und versuchte ein, zwei Beispiele von Shader-Programming nach zu schreiben, um das Konzept zu verstehen. Nach 3-4 Tagen habe ich alles wieder verworfen. Da ich nach wie vor grosse Mühe hatte das Gesamtbild zu sehen und zudem auch die Zeit nun ein bisschen knapp wurde.

4.10 Projektabschluss

Ohne Fleiss kein Preis!

4.10.1 Ziele

Zum Projektabschluss gehört eine gut gegliederte und verständliche Dokumentation der geleisteten Arbeit. Für die Bachelorthesis müssen zudem noch ein Poster, Film, ein Book-Eintrag sowie eine, respektive zwei Präsentationen gehalten werden. All dies zu erstellen, braucht seine Zeit, daher wurde für den Projektabschluss am meisten Zeit von allen Arbeitspaketen eingeplant.

4.10.2 Umsetzung

Im letzten Arbeitspaket kamen noch ganz viele kleinere sowie grösser Arbeiten auf mich zu. So musste noch eine Dokumentation über das Gesamtwerk geschrieben werden. Der Start war etwas harzig. Man legt sich zwar zu Recht, über was man schreiben möchte und erstellt zu Beginn einmal die Kapitel. Aber ein Schreibfluss entwickelt sich erst nach ein paar hundert Wörter. Zwischendurch musste auch noch ein Poster sowie ein Book-Eintrag geschrieben werden, da hier die Deadline etwas früher war. Zu guter Letzt musste noch ein Film gemacht werden. Da ich noch nie ein Film selber gedreht und auch geschnitten habe, war dies für mich noch recht aufwendig.

4.10.3 Ergebnis

Als Endergebnis in diesem Arbeitspaket entstand, aus meiner Sicht, eine gut gegliederte und verständliche Dokumentation über meine geleistete Arbeit. Auch die für die Bachelorthesis dazu gehörigen Abgaben (Poster, Film, Book-Eintrag) wurden fristgerecht eingereicht. Für das letzte Arbeitspaket habe ich am meisten Zeit eingeplant. Dies war auch gut so, da ich die Zeit wirklich gebraucht habe, um einen guten Projektabschluss zu erreichen.

5 System-Review

In diesem Kapitel wird der Review des entwickelten Systems beschrieben.

- Wurden alle definierte Ziele erreicht?
- Funktioniert das ganze System so wie es muss?
- Was gibt es für Verbesserungspotential(e)?

5.1 Funktionstest

Der Funktionstest wurde mittels eines Feldtests durchgeführt. So kamen gegen Ende der Arbeit Familie, Freunde und auch Mitstudent in den Genuss das System zu testen. Teilweise erinnerte alles ein bisschen an eine Turnübung. Man sprang hoch, duckte sich, rannte schnell nach vor und zurück oder auch nach links und rechts um zu schauen, was auf dem Bildschirm passierte. Das Lachen im Gesicht meiner Tester bestätigte mir das, dass ganze System so funktionierte wie geplant. Probleme tauchten meist nur dann auf, wenn sich mehr als ein Betrachter vor der Tiefenkamera/Fernseher befand und diese nicht mehr genau wusste, welches Skelett sie nun tracken muss/soll. Dieses Problem lässt sich leider nur schlecht beheben. Es wäre zwar technisch möglich mehrere Personen gleichzeitig mit der Intel RealSense D435 zu tracken, allerdings ist es praktisch unmöglich, dass das Bild auf dem Bildschirm für 6 unterschiedliche Personen immer noch die korrekte Perspektive hat, ausser die Personen befinden sich alle an der exakten selben Position…

Aktuell ist die Szenenwahl, die angezeigt werden kann mengenmässig beschränkt. Man kann zwischen einem blauen, roten Würfel oder einem gelben Zylinder in einer Box wählen oder die Box ganz ausblenden und zwei Schneemänner anzeigen lassen. Der Zylinder kann man zudem noch über die eigene Achse drehen lassen.

5.2 Vergleich zum Samsung The Frame

In den ersten beiden Kapiteln hatte ich als vergleichbares Objekt den Samsung The Frame Fernseher erwähnt. Das Ziel beim Frame ist es den Fernseher so «verschwinden» zu lassen das es aussieht, als würde man auf ein Gemälde oder Bild schauen. Das Ganze wird beim Frame so perfektioniert das der Rahmen des Monitors, demjenigen eines Bilderrahmens ähnelt. Also indem eine Holzumrandung hinzugefügt wird. Wenn der Fernseher nun an einer Wand hängt und den Art Mode startet, so verwandelt sich der Fernseher in ein Gemälde.

Das Verschwinden des Fernsehers ist mit meiner Lösung leider nicht ganz möglich. Da zusätzlich zum Fernseher noch eine Tiefenkamera installiert werden muss. Zwar sind die neusten Kameras, welche auf dem Markt sind ziemlich klein, aber dennoch erkennt man nebst dem Fernseher noch eine weitere Komponente. Auch ist meiner Meinung nach die Illusion, dass man in eine Box schaut, noch nicht ganz perfektioniert.

5.3 Zielerfüllung

5.3.1 Funktionale Anforderungen

Nr.	Titel	Beschreibung	Muss/kann	Erreicht?
1	Betrachter/Person erkennen	Die genaue Position des Betrachters soll erkannt werden.	**Muss**	**Ja**
2	Szene modellieren	Es soll eine Szene in Unity/Blender modelliert/erstellt werden, welche auf dem Fernseher und somit für den Betrachter zu sehen ist.	**Muss**	**Ja**
3	Dynamisches Bild generieren	Bewegt sich der Betrachter im Raum, so wird anhand seiner Position die richtige perspektive der Szene/ des Bilds angezeigt.	**Muss**	**Ja**
4	Helligkeit berücksichtigen	Die Raumhelligkeit soll erkannt werden und die Helligkeit in der Szene/Bild dementsprechend angepasst werden.	Kann	Nein

Tabelle 5: Zielerfüllung

Das Hauptziel, das sich das gerenderte Bild auf den Bildschirm relativ zur Position des Betrachters verändert, wurde erfüllt. Allerdings ist das Erlebnis für den Betrachter nur teilweise erfolgreich. Die Illusion, dass er durch die Wand blickt und nicht einfach einen normalen Fernseher betrachtet, konnte nicht zu 100% umgesetzt werden. Auch wird die Helligkeit des Raums nicht berücksichtigt.

5.3.2 Nicht funktionale Anforderungen

Nr.	Titel	Beschreibung	Muss/kann	Erreicht?
1	Plattform unabhängig	Das Endergebnis soll auf allen gängigen Plattformen entwickelt und auch kompiliert werden können.	**Muss**	**Ja**
2	Tiefenkamera unabhängig	Die Position des Betrachters soll mit den meisten gängigen Tiefenkameras ermittelt werden können. Somit muss das SDK Plattform unabhängig sein.	**Muss**	**Ja**
3	Applikation für Smart-TV entwickeln	Die Applikation soll auf einem Smart-TV laufen, somit braucht es die Komponente Computer / Notebook nicht.	Kann	Nein

Bei den nicht funktionalen Anforderungen konnten alle Muss-Anforderungen umgesetzt werden.

5.4 Verbesserungspotenzial / Ausbaumöglichkeiten

Aktuell besteht das ganze System aus 3 Komponenten. Einen Fernseher, eine Tiefenkamera und einem Computer/Notebook. Der Fernseher und auch die Tiefenkamera braucht es immer zwingend. Aber den Computer/Notebook könnte man ersetzen, indem man die Software so umschreiben würde, dass diese auf einem SmartTV als eigenständig Applikation ausgeführt werden könnte. Somit könnte eine Hardwarekomponente eingespart werden. Da ich für das Tracking des Betrachters, das Nuitrack SDK gewählt habe, kann man auch ganz einfach die Tiefenkamera austauschen und es muss nicht zwingend die RealSense D435 sein.

Ein weiteres Verbesserungspotential wäre, wenn man zwischen mehrere Szenen auswählen könnte. Ähnlich wie beim Samsung The Frame. Dort besitzt man die Möglichkeit zwischen verschiedenen Bildern zu wechseln und sorgt damit für Abwechslung.

Weiter wäre auch eine Möglichkeit das Ganze spielerischer aufzuziehen. Das Nuitrack SDK liefert das ganze Skelett des Betrachters zurück. Daher wäre es möglich, nebst dem Kopf, auch andere Körperteile zu tracken. Würde man nun zum Beispiel zusätzlich noch die Hände tracken, so könnte man den Betrachter noch ein Spiel, wie zum Beispiel Tetris, spielen lassen.

Die Illusion, dass man durch eine Wand schaut und nicht in einen Fernseher könnte mit baulichen Massnahmen verstärkt werden. So könnte, wenn das System z.B. in einem Ausstellungsraum eingerichtet wird, der Fernseher in einem schwarzen Holz- oder Kartonkasten versteckt werden. Dabei dürften natürlich keine Kanten vom Fernseher mehr sichtbar sein. Wird das Programm gestartet, so hat man nicht mehr das Gefühl, einen Fernseher anzuschauen.

6 Fazit

6.1 Projektbezogenes Fazit

Die modellierte Szene ist sicherlich eher spartanisch und könnte noch etwas ausgefallener sein. Schön wäre gewesen, wenn man zum Beispiel zwischen einer winterlichen Berglandschaft und einem malerischen Palmenstrand hätte auswählen können. Aufgrund des zeitlichen Aufwandes für die Modellierung war dies leider nicht mehr möglich.

Für die Arbeitsorganisation, respektive Vorgehensweise wurde Scrum als Basis genommen und Teile davon adaptiert. Dank dem agilen Vorgehen konnten dann auch rasch die Schwierigkeiten mit dem ersten Lösungsansatz erkannt und in den nächsten Iterationen ein anderer Lösungsansatz entworfen und auch umgesetzt werden.

Das System-Review hat mir gezeigt, dass die Anforderungen umgesetzt werden konnten. So war es sicherlich eine grosse Genugtuung zu sehen, wie meine Tester auf das System reagiert haben. Allerdings ist daraus auch zu erkennen, dass das Ganze noch nicht ganz perfekt ist. So ist mir das «verschwinden» des Fernsehers nicht zu 100% geglückt.

6.2 Persönliches Fazit

Die ganze Thematik rund um Virtual Reality und auch Computer Perception ist für mich nach wie vor ein sehr spannendes Thema, jedoch war oder ist es immer noch relativ weit weg von meiner beruflichen Tätigkeit als Softwareentwickler im Bereich des Performancemanagement von mobilen Datennetzen. Dennoch wollte ich mich dieser Herausforderung stellen! Entgegen kam mir das in Unity mit der Programmiersprache C# entwickelt wird. Da wir in unsere Firma diese auch einsetzen, musste ich hier nicht noch einen zusätzlichen Zeitaufwand betreiben, um diese zu erlernen.

Da es mir aber oft an allen Ecken und Kanten am Verständnis fehlte, konnte der erste Lösungsansatz, der zusammen mit Herrn Hudritsch diskutiert und auch ausgearbeitet wurde, leider nicht wie ursprünglich vereinbart und umgesetzt werden. Dennoch wurde eine Lösung gefunden, um die Projektziele zu erfüllen. Dies erfüllt mich mit Stolz. Auch konnte ich trotz der erwähnten Schwierigkeiten extrem viel in einem wirklich spannenden Umfeld lernen.

Ich möchte es nicht unterlassen mich bei Herrn Hudritsch für die lehrreiche Zeit und seine Geduld herzlich bedanken.

Selbständigkeitserklärung

Ich bestätige, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Textstellen, die nicht von mir stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Biel, 16.01.2020
Name Vorname: Carosella Dario

Unterschrift: ……..

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] Wikipedia - Trompe-l'oeil. URL: https://en.wikipedia.org/wiki/Trompe-l%27%C5%93il

[2] Samsung The Frame. URL: https://www.samsung.com/ch/tvs/the-frame/highlights/

[3] Samsung The Frame. URL : https://www.samsung.com/ch/tvs/the-frame/art-mode/

[4] Nuitrack SDK – 3 Body (skeletall) tracking middleware. URL: https://nuitrack.com/

[5] Unity Echtzeit-Entwicklungsplattform. URL: https://unity.com/de

[6] Intel RealSense D435. URL: https://www.intelrealsense.com/depth-camera-d435/

[7] Stereoskopisches Sehen (Stereosehen). URL: https://www.brillen-sehhilfen.ch/auge/stereoskopisches-sehen.php

[8] Intel RealSense Tiefenkamera der D400-Baureihe. URL: https://www.mouser.ch/new/Intel/intel-realsense-camera-400/

[9] Nuitrack SDK Architecture. URL: http://download.3divi.com/Nuitrack/doc/Architecture_page.html

[10] Nuitrack Skeleton System. URL: http://download.3divi.com/Nuitrack/doc/Overview_page.html

[11] Skript Grundlagen der Bildverarbeitung – Geometrische Operatoren von Marcs Hudritsch, 4. Februar 2019

[12] Skript Grundlagen der Bildverarbeitung – Projektive 2D-Koordinatentransformation von Marcus Hudritsch, 4. Februar 2019

[13] Generalized Perspective Projection – On-axis perspective von Robert Kooima, August 2008 (revised June 2009)

[14] Generalized Perspective Projection – Off-axis perspective von Robert Kooima, August 2008 (revised June 2009)

[16] Mathepedia – Orthonormalbasis. URL: https://mathepedia.de/Orthonormalbasis.html

[17] Unity – Understanding the View Frustum. URL: https://docs.unity3d.com/550/Documentation/Manual/UnderstandingFrustum.html

[18] Skript Realtime Rendering & Global Illumination – Zentralprojektion von Marcus Hudritsch, 12. August 2019

[19] Blender – Home ot the Blender project. URL: https://www.blender.org/

[20] OpenGL - Coordinate System. URL: https://learnopengl.com/Getting-started/Coordinate-Systems

[21] Das Scrum Team - In aller Kürze: Scrum erklärt in 100 Wörtern. URL: https://www.dasscrumteam.com/de/scrum

[22] Wikipedia – GitHub. URL: https://de.wikipedia.org/wiki/GitHub

[23] Rahmenreglement für Kompetenznachweise an der Berner Fachhochschule (KNR). 2005

[24] OpenCV URL: https://opencv.org/

[25] Generalized Perspective Projection von Robert Kooima, August 2008 (revised June 2009)

Anhang

A. Benutzerhandbuch

Dieses Handbuch soll dabei behilflich sein die Software zu bedienen oder den eigenen Bedürfnissen anzupassen. Möchte man die Software anpassen, so wird Unity als Entwicklungsumgebung gebraucht. Zudem muss für die Bearbeitung des Codes noch das Nuitrack SDK installiert werden

Vorbereitungen

Möchte man den Softwarecode bearbeiten, so sollte zuerst die Installationsanleitung gelesen und ausgeführt werden. Danach sollten folgende Vorbereitungen getroffen werden:

1. Es wird empfohlen das das Sichtfeld der RealSense nicht direktem Sonnenlicht ausgesetzt werden sollte
2. Die optimale Höhe für die RealSense beträgt ca. 120 cm. Zudem sollte die Tiefenkamera mittig vom Fernseher platziert werden und parallel zum Boden ausgerichtet sein.
3. Die «Spielfläche» sollte gross genug sein, damit man sich gut im Raum bewegen kann.
4. Dunkle Kleidung sollten vermieden werden.

Unityprojekt starten / konfigurieren

Im Unityprojekt Virtual Window TV sind folgende Scripts vorhanden

- ⇨ HeadTracker.cs
  - Bestimmt mittels der RealSense Tiefenkamera die Position des Betrachters.
  - Das Script ist dem GameObject **Viewer** anghängt
  - Als *Tracked Joint Type* ist der Kopf bestimmt
- ⇨ ObjectController.cs
  - Im ObjectController werden die geometrischen Formen, die in der Szene vorhanden sind, einer Tastatur zugeordnet und können ein oder ausgeblendet werden
- ⇨ ProjectionMatrix.cs
  - In diesem Script findet die ganze Berechnung für das generierte Bild statt.
- ⇨ Rotate.cs
  - Bringt ein GameObject zum rotieren

Das wichtigste GameObject

Die Projection_Plane ist das wichtigste GameObject in meinem Unityprojekt. Alles was sich hinter ihr befindet wird schlussendlich auch gerendert und hat somit die perspektivische korrekte Darstellung. Möchte man also die ganze Szene auswechseln, so muss man die Szene die man darstellen möchte hinter ihr platzieren.

Unityprojekt ausführen / exe ausführen

Startet man die Software, ohne eigene Anpassungen vorgenommen zu haben, ist *ObjectController* wie folgt konfiguriert:

Taste	Funktion
C	Die ganze Box wird ausgeblendet, beim erneuten tippen der Taste «C» wird die Box wieder eingeblendet.
B	Der blaue Würfel ausgeblendet, beim erneuten tippen der Taste «C» wird der Würfel wieder eingeblendet.
R	Der rote Würfel ausgeblendet, beim erneuten tippen der Taste «C» wird der Würfel wieder eingeblendet.
Y	Der gelbe Zylinder ausgeblendet, beim erneuten tippen der Taste «C» wird der Würfel wieder eingeblendet.
S	Der linke Schneemann (ist nur sichtbar wenn die Box ausgeblendet ist) ausgeblendet, beim erneuten tippen der Taste «C» wird der Würfel wieder eingeblendet.
O	Der rechte Schneemann (ist nur sichtbar wenn die Box ausgeblendet ist) ausgeblendet, beim erneuten tippen der Taste «C» wird der Würfel wieder eingeblendet.
Space	Der Zylinder beginnt sich zu rotieren, betätigt man nochmals die «Space-Taste» so stopp die Rotation

B. Installationsanleitung

Diese Installationsanleitung bezieht sich auf die Installation von Unity und des Nuitrack SDK auf einem Windows-Rechner. Für alle anderen Plattformen verweise ich gerne auf die Installationsanleitungen der jeweiligen Programme:
Unity: https://docs.unity3d.com/Manual/GettingStartedInstallingUnity.html
Nuitrack: http://download.3divi.com/Nuitrack/doc/Installation_page.html

Unity installieren

Unity kann entweder als Unity oder Unity Hub installiert wählen. Mittels Unity Hub lassen sich mehrere Unity-Versionen verwalten. Zudem liefert Unity Hub eine Übersicht über alle Projekte und die jeweiligen Unity-Versionen, die für diese gebraucht wurden.
Die Installation von Unity ist relativ einfach und benötig nur ein paar wenige Schritte:

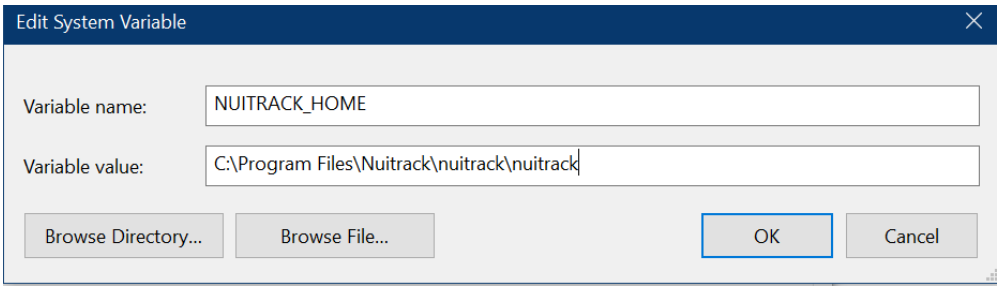
1. Installationsdatei herunterladen
    a. URL: https://unity3d.com/de/get-unity/download
2. Unity installieren
3. Starten!

Nuitrack SDK installieren

Das Nuitrack SDK ist etwas aufwändiger zum Installieren:

1. Nuitrack herunterladen
    a. Windows 32-Bit: http://download.3divi.com/Nuitrack/platforms/nuitrack-windows-x86.exe
    b. Windows 64-Bit: http://download.3divi.com/Nuitrack/platforms/nuitrack-windows-x64.exe
2. .exe ausführen und den Anweisungen folgen
3. Stellen Sie sicher das Sie Microsoft Visual C++ Redistributable für Visual Studio auf ihrem Computer installiert haben. Wenn nicht installieren Sie dieses Paket abhängig von ihrer Visual Studio version:
    a. Visual C ++ Redistributable 2015 (x86)
    b. Visual C ++ Redistributable 2015 (x64)
    c. Visual C ++ Redistributable 2017 (x86)
    d. Visual C ++ Redistributable 2017 (x64)

4. Falls bei der automatischen Einrichtung Probleme aufgetreten sind, müssen die Umgebungsvariablen manuell eingerichtet werden:
    a. Zum Control Panel navigieren
    b. System auswählen
    c. Advanced system settings auswählen
    d. Auf den Button «Enviroment Variables» auswählen
    e. Unter System variables auf den Button «New…» klicken
    f. Als Variable name **NUITRACK_HOME** und aus Value den Installationspfad definieren



    g. Fertig

Nuitrack Lizenzaktivierung

In einem letzten Schritt muss die Lizenz noch aktiviert werden. Folgende Lizenzen sind für Nuitrack verfügbar: https://nuitrack.com/#pricing

1. Zum Aktivierungstool navigieren (C:\Program Files\Nuitrack\nuitrack\nuitrack\activation_tool)
2. Nuitrack.exe starten
3. RealSense anschliessen und auf den Button «Compatibility test» klicken
4. Lizenz Key eintippen
5. Auf den Button «Activate» klicken

Quelle: http://download.3divi.com/Nuitrack/doc/Installation_page.html

C. Inhalt des USB-Stick

Auf dem beigelegten USB-Stick befindet sich folgender Inhalt:

- Source
    - Virtual Window TV
    - FindHomography MatLab
    - FindHomography C#
- Dokumentation
- Film
- Poster