3D Rekonstruktion eines Raumes, mittels Microsoft Kinect und Point Cloud Library

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Sean Milbach

2063941

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Roland Greule

Zweitprüfer: Dipl. Ing. Matthias Kuhr

Hamburg, 29. 02. 2016

Inhaltsverzeichnis

Abstract

3D reconstruction of rooms has been very costly for a long time. Thanks to the Microsoft Kinect and the open-source library PCL, a room can now be reconstructed using a system which will cost under 1000€. The resulting 3D model can be used for demonstrating purpose in CAD software. All needed steps, as well as the used soft- and hardware and its underlying algorithms, will be described in this paper.

Zusammenfassung

3D Rekonstruktion von Räumen war lange Zeit mit hohen Kosten verbunden. Dank der Microsoft Kinect und der open-source Bibliothek PCL ist es nun jedoch möglich, schon mit Systemen für unter 1000€ Räume so zu rekonstruieren, dass die 3D-Modelle sich zu Anschauungszwecken in beispielsweise CAD-Software importiert lassen. Diese Arbeit beschreibt alle nötigen Arbeitsschritte und stellt die genutzte Soft- und Hardware sowie die Algorithmen, auf denen das System basiert, vor.

1 Einleitung

3D Rekonstruktion klang lange Zeit nach weit entfernter Zukunftsmusik. Kleinere Objekte lassen sich mittlerweile mit 3D Scannern erfassen. Die Rekonstruktion von größeren Objekten oder Oberflächen war langezeit mit großem Aufwand und vor allem sehr hohen Kosten verbunden. Die meist hierzu genutzten LiDAR-[1] und TOF Kamera Systeme[2], welche Laserstrahlen zur Entfernungsmessung nutzen, sind für Heimanwender nicht bezahlbar.

Mit der Microsoft Kinect[3] ist nun jedoch eine erstaunlich günstige RGB-D[4] Kamera am Markt, die zu einem, an ihrer Hardware gemessen, extrem niedrigen Preis verfügbar ist. Zusammen mit dem Kinect SDK[5] und öffentlichen Bibliotheken bieten sich viele Einsatzmöglichkeiten.

Die Möglichkeit, mit der Kinect und der open-source Library PCL[6], kostengünstig einen ganzen Raum zu rekonstruieren, soll in in dieser Arbeit analysiert werden. Außerdem werden die damit verbundenen Einschränkungen in Bezug auf andere Systeme eingehend beleuchtet.

1.1 Ziele der Arbeit

Die Hauptziele der Arbeit sind:

1. das Erfassen der Geometrie eines Raumes unter Verwendung einer Microsoft Kinect und der PCL Implementierung von KinectFusion „KinFu“

2. das Aufbereiten der Daten um die weitere Nutzung der erstellten 3D Modelle zu ermöglichen

3. das Einbinden der 3D Modelle in Unity sowie ReCap zur Veranschaulichung sowie zur Demonstration der Nutzbarkeit der erstellten 3D Modelle

4. die Analyse des genutzten Systems sowie der Vergleich mit ähnlichen Systemen

[1] engl. Light detection and ranging, Siehe Abschnitt 2.2.1
[2] engl. Time of Flight, siehe Abschnitt 2.3.2
[3] Kamerasystem, siehe Abschnitt 2.2.3
[4] engl. Red Green Blue - Depth, siehe Kapitel 2.2.2
[5] Software API und Sammlung von Programmen für die Kinect, siehe Abschnitt2.7.4
[6] engl. Point Cloud Library siehe Abschnitt 2.5

2 Grundlagen

In diesem Kapitel werden die theoretischen sowie technischen Grundlagen dieser Arbeit beschrieben. Neben der genutzten Hard- und Software sind die grundlegenden Algorithmen der 3D Rekonstruktion Teil dieser Ausführung.

2.1 3D Rekonstruktion

Mit 3D Rekonstruktion wird das Erstellen von dreidimensionalen Modellen von reallen Objekten bezeichnet. Es können auch ganze Szenerien einschließlich ihrer enthaltenen Objekten und Strukturen rekonstruiert werden. Der wesentliche Unterschied zum üblichen Erstellen von 3D-Modellen in CAD[1]-Anwendungen liegt im erheblich geringeren Arbeitsaufwand. Punkte werden nicht manuell erfasst um daraus Modelle zu erzeugen, sondern 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.

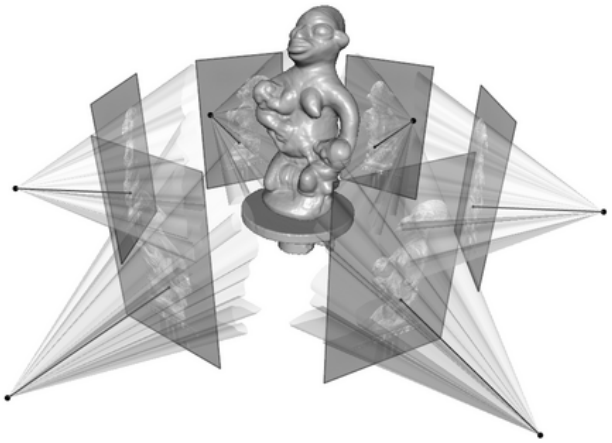

Abbildung 2.1: 3D Rekonstruktion

Grundlegend lassen sich aktive sowie passive Methoden zur Rekonstruktion unterscheiden. Passiven Methoden erzeugen aus mehreren gewöhnlichen digitalen Bildern oder Videos, mittels Bildverständnis, ein 3D Modell. Die aktiven Methoden hingegen berechnen

[1] engl. Computer Aided Design

ein 3D-Modell aus Tiefeninformationen, welche das System erfasst. Diese werden mittels eines mechanischen oder radiometrischen Entfernungsmessgerätes ermittelt. Im folgenden werden ausschließlich Systeme betrachtet, die aktive Methoden nutzen.

2.2 Hardware

In diesem Abschnitt werden unterschiedliche Kamerasysteme, die zur 3D Rekonstruktion genutzt werden können, vorgestellt.

2.2.1 LiDAR

Ähnlich wie bei einem Radar, messen LiDAR[2] Systeme die Entfernung, mit einem richtbaren Laserstrahl (siehe Abbildung 2.2). Dafür kann etwa der Laser beweglich sein, oder der Strahl wird über einen Spiegel ausgerichtet. Über die Lichtlaufzeit lässt sich die Entfernung zum getroffenen Objekt berechnen. So wird die Tiefenkarte Punkt für Punkt erweitert (siehe Abbildung 2.4).

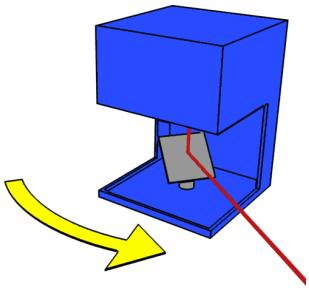

Abbildung 2.2: Scanner

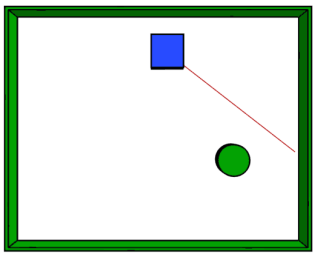

Abbildung 2.3: Umgebung

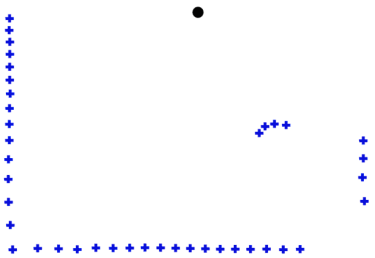

Abbildung 2.4: Aufgezeichnete Daten

2.2.2 RGB-D Kameras

RGB-D[3] Kameras sind Sensorsysteme, welche abgesehen von einer Farbkamera noch weitere Sensoren nutzen um die Tiefeninformationen des aufgenommenen Bildes zu registrieren. Um die Entfernung zu messen, lassen sich TOF-Systeme (siehe Abschnitt 2.3.2), Projektions-Systeme (siehe Abschnitt 2.3.1) oder auch Stereokamera-Systeme nutzen.

[2] Abkürzung Englisch für: Light Detection and Ranging
[3] Kurzform für Red Green Blue - Depth

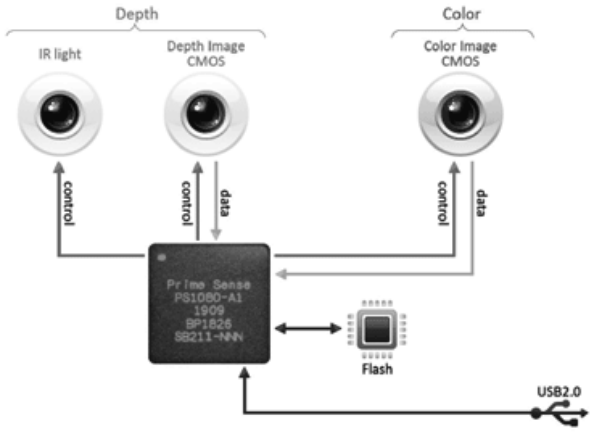

Abbildung 2.5: PrimeSense Aufbau

Abbildung 2.6: PrimeSense Kamera

2.2.3 Die Kinect Kamera

Die Kinect ist eine Hardware von Microsoft, die eigentlich zur Steuerung der Spielekonsole Xbox konzipiert wurde. Der Entwicklungsansatz war es, eine Eingabemöglichkeit zu schaffen, die ohne einen bisher üblichen Controller auskommt. Allein durch die Bewegung des Körpers sollte die Interaktion mit der Spielkonsole ermöglicht werden. Im Februar 2012 wurde die erste Version der Kinect für Windows veröffentlicht und somit bot sich Entwicklern die Möglichkeit, die Hardware und Software für ihre Ideen zu nutzen und zu verändern.

Abbildung 2.7: Kinect V1

Abbildung 2.8: Kinect V2

Feature	Kinect V1	Kinect V2
Auflösung RGB Kamera	640 x 480	1920 x 1080
Auflösung Tiefenkamera	320 x 240	512 x 424
Framerate	30 fps	30 fps
Tiefenerkennung	Structured Light	Time Of Flight
Maximale Tiefe	4,5 m	4,5 m
Minimale Tiefe	40 cm	50 cm
Horizontales Sichtfeld	57°	70°
Vertikales Sichtfeld	43°	60°
SDK	Kinect for Windows 1.8	Kinect for Windows 2.0
PCL Unterstützung	Ja	Nein
Preis Veröffentlichung	299€	199€
Preis Aktuell	150€	130€

2.3 Tiefensensoren der Kinect Systeme

Im Gegensatz zu Lidar Systemen, bei denen jeder Punkt nacheinander erfasst wird, bieten RGB-D Kameras den Vorteil einer kompletten Momentaufnahme des Sichtfeldes. Während die Kinect V1 mithilfe eines Structure Light Verfahrens (Weiterentwicklung der Streifenprojektion, siehe Kapitel 2.3.1), ihr Tiefenbild berechnete, wird bei der Kinect V2, mit TOF[4] der gleiche Ansatz wie bei den meisten Laserscannern genutzt. Diese nutzen einen Laser zur Entfernungsermittlung mittels Time of Flight. Für jeden Punkt, der abgetastet wird, emittiert ein Laser einen Lichtimpuls. Da die Lichtgeschwindigkeit bekannt ist, kann aus der Zeit, die der Impuls benötigt, um wieder zum Scanner zurück zu kehren, die Distanz zum getroffenen Objekt berechnet werden.
Im folgenden werden die unterschiedlichen Methoden zur Entfernungsmessung der Kinect V1 sowie der Kinect V2 erläutert.

2.3.1 Kinect V1 - Structured-Light

Um Tiefeninformationen zu erfassen nutzt die Kinect V1 das Structured-Light Prinzip. Dafür sind mindestens eine Kamera und ein Projektor nötig, welcher ein Muster auf den zu erfassenden Bereich projiziert. Zwischen Projektor und Kamera muss ein bekannter Abstand eingehalten werden, damit mittels Triangulation die Entfernung berechnet werden kann. Die Muster des Lichts können parallel verlaufende Linien, Schachbrettmuster oder auch Punktmuster sein. Das Muster ist dem System bekannt. Wird es auf eine nicht

[4] engl. Time Of Flight

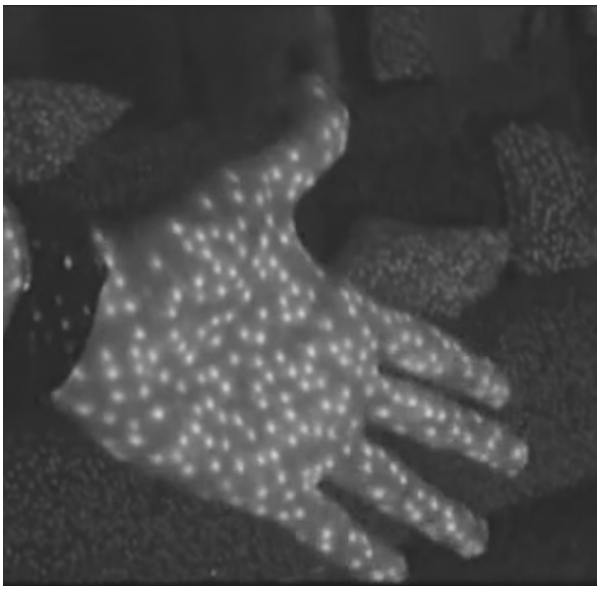

Abbildung 2.9: Strukturiertes Infrarotlicht

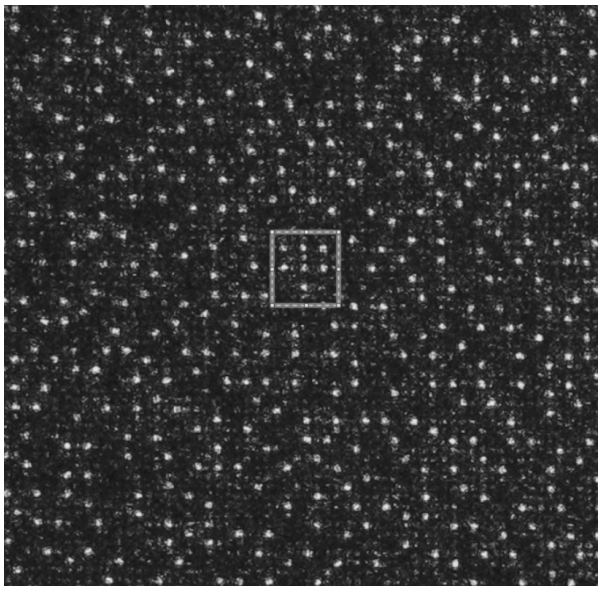

Abbildung 2.10: Gruppe von Punkten

absolut plane Ebene[5] projiziert, kommt es zu Verformungen des Musters, welche von der Kamera erfasst werden. Je näher ein getroffenes Objekt dem System kommt, desto größer wird das Muster auf ihm abgebildet.

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.

2.3.2 Kinect V2 - Time Of Flight / Laufzeitmessung

Das Erfassen von Tiefeninformationen wird bei der Kinect V2 mit einem gänzlich anderen Verfahren ermöglicht. Der Sensor misst indirekt die vergangene Zeit zwischen dem Aussenden eines Laserimpulses und dem Aufzeichnen desselben. Während eines 1 GHz

[5] parallel zum Projektor

Taktzyklus (1 ns) breitet sich Licht um 29,9 cm aus. Also lässt sich die Entfernung mit einer 10 GHz Stoppuhr schon auf 3 cm genau berechnen. Wird diese Lasermessung nun mit hoher Frequenz wiederholt, erhöht dies die Präzision des Systems noch weiter.

Im Sensor der Kinect V2 sind alle Pixel zweigeteilt. Zwischen den beiden Hälften, wird im selben Takt hin und her gewechselt, in dem auch der Laser ein- und ausgeschaltet wird. Ist der Laser an, so erwartet die erste Pixelhälfte Informationen, ist er aus, registriert die zweite Pixelhälfte einfallendes Licht. Wird der Laser direkt, also ohne Entfernung, auf den Sensor gerichtet, so ist die vergangene Zeit beinahe null und das gesamte Licht wird von den ersten Pixelhälften aufgezeichnet. Wird der Abstand zwischen Sensor und Laser nun langsam vergrößert, so verzeichnet die zweite Pixelhälfte einen immer größeren Anteil des gesamt aufgezeichneten Lichtes. Ausgehend von einem 1 GHz Taktzyklus wird bei einem Abstand von 29,9 cm zwischen Quelle und Empfänger die gesamte Lichtmenge auf das zweite Pixel entfallen. Jede Distanz dazwischen lässt sich nun aus den anteiligen Lichtmengen der einzelnen Pixelhälften von der Gesamtlichtmenge errechnen, die aufgezeichnet wurde. Wäre nur die Lichtmenge zu einer Phase aufgezeichnet worden, so wäre die Berechnung des Anteils an der Gesamtlichtmenge nicht möglich, denn die Menge an reflektiertem Licht variiert, zum Beispiel ist die Farbe des getroffenen Gegenstandes relevant, oder auch Verunreinigungen in der Luft. Es kann also nicht von einer fest eingestellten Lichtmenge ausgegangen werden.

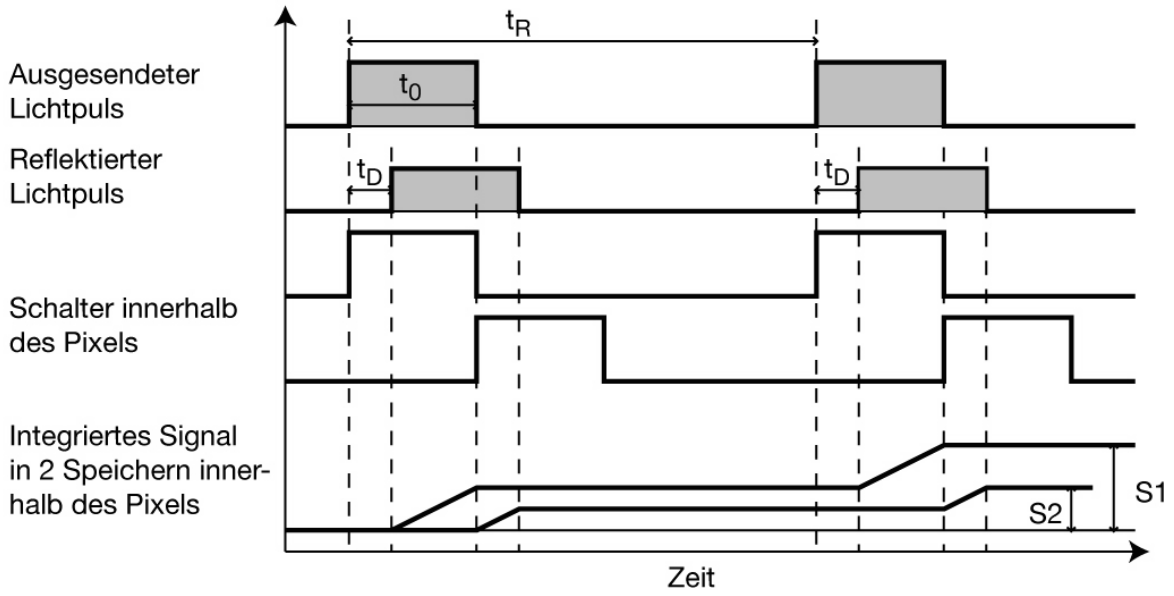

Abbildung 2.11: Time of Flight Kamera Prinzip

Wird die Zeit eines Zyklus überschritten, wird das Licht des letzten Zyklus von der ersten Pixelhälfte aufgezeichnet. Das Ergebnis der Messung ist nun nicht mehr eindeutig, da die Daten der Entfernung auch jedem eigenen Vielfachen entspricht. Um dies zu vermeiden, muss die Länge des Zyklus vergrößert werden, damit das Licht ausreichend Zeit hat, um nicht nach der zweiten Pixelhälfte anzukommen. Durch das Vergrößern der Aufzeichnungszeit vermehren sich jedoch auch aufgezeichnete Fehler, zum Beispiel verursacht durch freie Elektronen, die vom Sensor erfasst werden. Dies vermindert die Exaktheit der Aufnahme. Um diese beiden Probleme zu umgehen macht die Kinect V2 immer zwei Aufnahmen. Die Erste mit einer langen Laufzeit, um ein ungefähres Abbild der Tiefe des danach folgenden Bildes zu bekommen. Mit den Informationen des ersten Scans lässt sich nun bestimmen, ob und um wie viele Zyklen verzögert ein Signal aufgezeichnet wurde.

Durch dieses in der Kinect V2 implementierte System ist es im Gegensatz zur V1 nun möglich, die Tiefeninformationen für jedes einzelne Pixel zu berechnen, ohne auf die benachbarten Werte angewiesen zu sein. Hinzu kommt eine Funktion, die ermöglicht das Umgebungslicht zu vernachlässigen. Dies ermöglicht auch Aufnahmen bei stark unterschiedlichen Lichtverhältnissen, was mit die Kinect V1 problematisch ist.

2.4 Point Clouds

Eine Point Cloud (Punktwolke oder Punkthaufen) ist eine Menge von Punkten in einem Koordinatensystem. Punktwolken sind ein übliches Ausgabeformat für 3D-Scanner. Hier entspricht jeder aufgezeichnete Wert einem Punkt der Wolke in einem dreidimensionalen Koordinatensystem. Die Punktwolke beschreibt die Oberflächenstruktur der eingelesenen Geometrie. Zusätzlich können zu jedem Punkt weitere Informationen gespeichert werden, wie zum Beispiel Farbwerte.

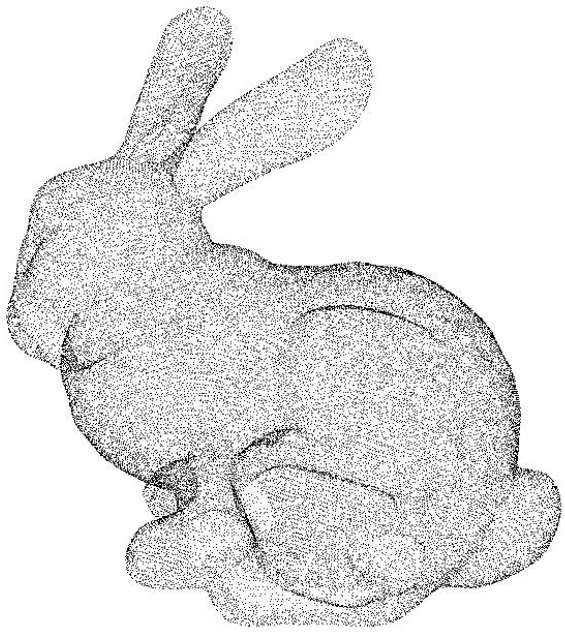

Abbildung 2.12: Point Cloud Modell eines Hasen

2.5 Point Cloud Library

Die Point Cloud Library ist eine open-source Bibliothek von Algorithmen zur Verarbeitung von Punktwolken im zwei- sowie dreidimensionalen Raum. Sie enthält mehrere Bereiche, die den unterschiedlichen Einsatzmöglichkeiten von Punktwolken gerecht werden, wie zum Beispiel Filterung, Modellschätzung, Oberflächenrekonstruktion, Registrierung, Segmentierung und Visualisierung. Die Bibliothek wurde unter der BSD-Lizenz veröffentlicht. Somit ist das Kopieren, Verändern und Verbreiten zur kommerziellen Nutzung sowie zu Forschungszwecken erlaubt und frei von Kosten. Die Point Cloud Library ist plattformunabhängig und ist in C++ programmiert. Sie enthält eine Reihe kleiner Bibliotheken, die sich auch einzeln verwenden lassen um die Anwendungen nicht unnötig zu vergößern. Die Entwicklung wird stetig vorangetrieben. Neben zahlreichen weltweit vertretenen Universitäten wird das Projekt auch von diversen Stiftungen und Firmen unterstützt, unter

ihnen sind auch Intel und Nvidia.

2.6 SLAM

In der Robotik wird zur Lokalisierung das SLAM-Verfahren[6] genutzt. Für den Roboter ist es wichtig seine eigenen Position in der eigens erstellten Karte berechnen zu können. Zur Rekonstruktion eines Raumes mit einer in der Hand gehaltenen Kamera, ist es ebenfalls elementar, die Position dieser im Bezug zur erfassten Geometrie bestimmen zu können. In den KinectFusion Implementierungen wird dieses Problem mit dem ICP Algorithmus (siehe Abschnitt 3.2.1) gelöst. Mithilfe des ICP lässt sich nach dem Zusammenfügen von zwei Meshes oder Punktwolken die Bewegung der Kamera zwischen den beiden Perspektiven ermitteln. Aus diesen Bewegungen lässt sich die Position der Kamera im Aufzeichnungsvolumen berechnen.

[6]Simultaneous Localization and Mapping; deutsch Simultane Lokalisierung und Kartenerstellung

2.7 Software

Für die Verarbeitung sowie Darstellung von Punktwolken und Meshes gibt es diverse Programme. Die für diese Arbeit genutzte Software soll in diesem Abschnitt kurz vorgestellt werden.

2.7.1 MeshLab

MeshLab ist eine open-source 3D-Meshbearbeitungs Software. Sie lässt sich sowohl als alleinstehende Software, aber auch als library einsetzen. Die Software wurde gezielt für die oftmals sehr großen Modelle von 3D Scans konzipiert. Für die üblichen Bearbeitungen bietet MeshLab Werkzeuge für das Bearbeiten, Säubern, Ergänzen sowie das Rendern und Konvertieren. Unterstützt werden eine Vielzahl von Formaten, darunter PLY, OBJ, 3DS, Collada, STL sowie die Möglichkeit, Pointclouds zu bearbeiten.

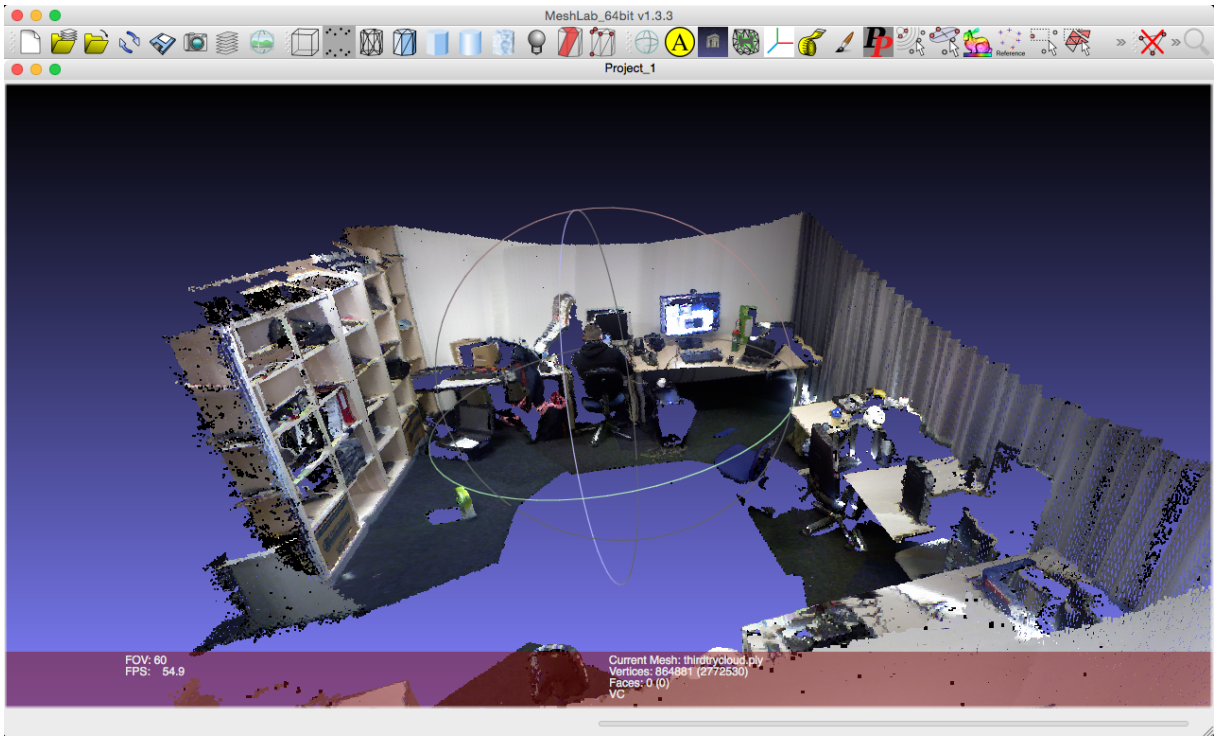

Abbildung 2.13: Benutzeroberfläche von MeshLab

2.7.2 ReCap

ReCap ist eine Erweiterung der AutoCAD Software. Mittels ReCap lassen sich Punktwolken, zum Beispiel von Laserscans, importieren und so konvertieren, dass sie sich in anderen Autodesk-Produkten nutzen lassen. ReCap ermöglicht außerdem das Bearbeiten der Punkwolke. Neben dem Korrigieren und Säubern stehen dem Anwender diverse Werkzeuge zur Vermessung sowie Visualisierung bestimmter Merkmale der Punktwolken zur Verfügung.

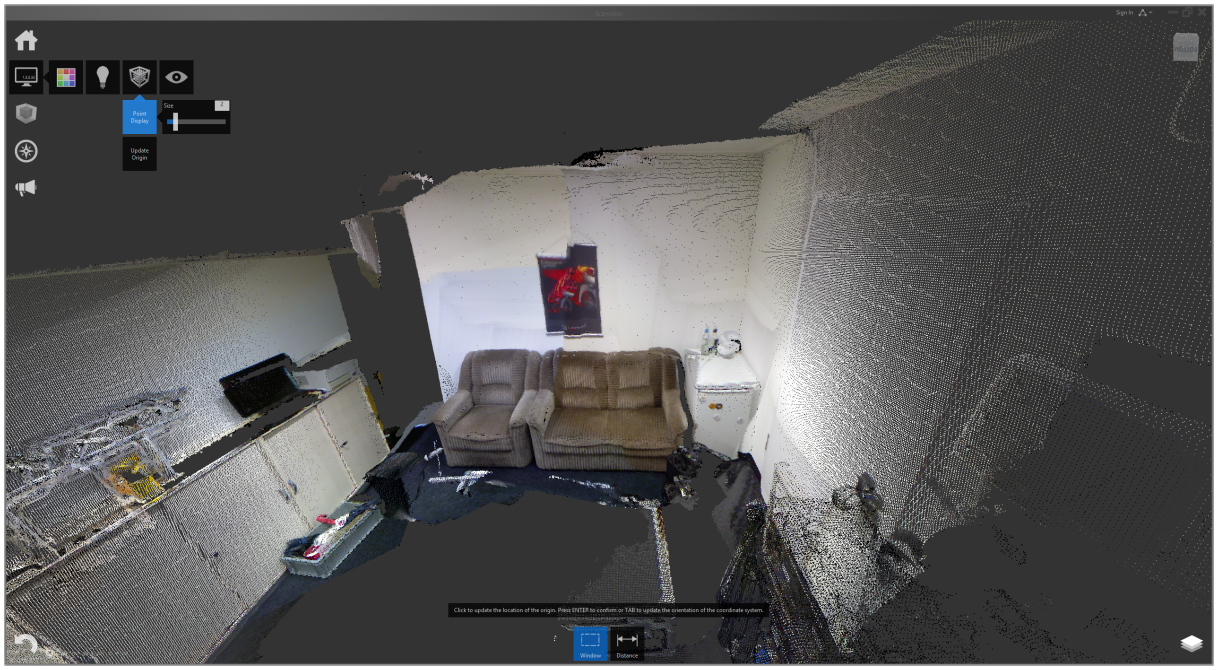

Abbildung 2.14: Benutzeroberfläche von ReCap

2.7.3 Unity

Unity ist eine plattformübergreifende Spiel-Engine sowie Entwicklungsumgebung. Neben den großen bekannten Betriebsystemen wie Windows und Mac OS X werden auch viele mobile und auch Konsolen-Systeme unterstützt. Zudem gibt es bereits Bibliotheken für mehrere Head-Mounted Displays. Es lassen sich sehr einfach und schnell 3D Modelle so einbauen, dass sie mit einem Head-Mounted Display, wie zum Beispiel der Oculus Rift, angesehen werden können. Die Bewegung einer Spielfigur in einem 3D Modell eines Raumes lässt sich in kurzer Zeit zu Anschauungszwecken ermöglichen.

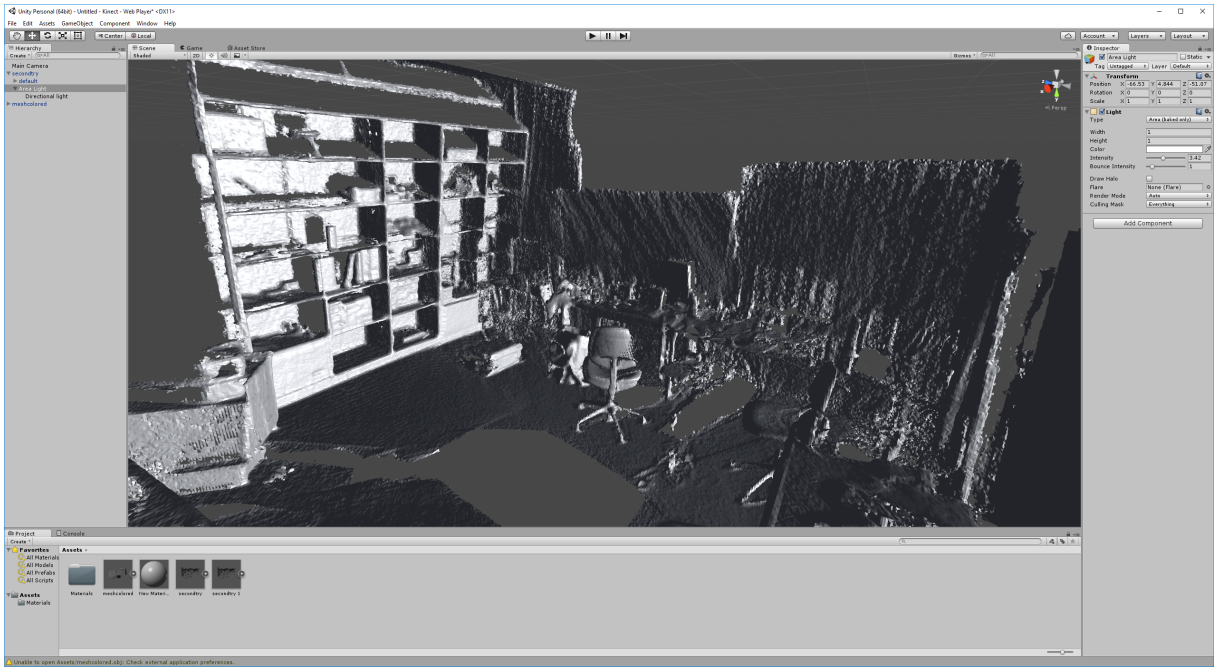

Abbildung 2.15: Benutzeroberfläche von Unity

2.7.4 Kinect for Windows SDK

Das Software Development Kit der Kinect von Microsoft ist eine API sowie eine Sammlung von Programmen, mit denen die Möglichkeiten der Kinect zugänglich gemacht und veranschaulicht werden. Es wurde für diese Arbeit verwandt, um die Vorteile der Kinect V2 unter KinectFusion zu untersuchen, da diese zur Zeit noch nicht von KinFu unterstützt wird.

3 Methode

Dieses Kapitel befasst sich mit der Funktionsweise von Kinect Fusion und den späteren Neuimplementierungen. Außerdem werden die grundlegenden Algorithmen betrachtet.

3.1 KinectFusion

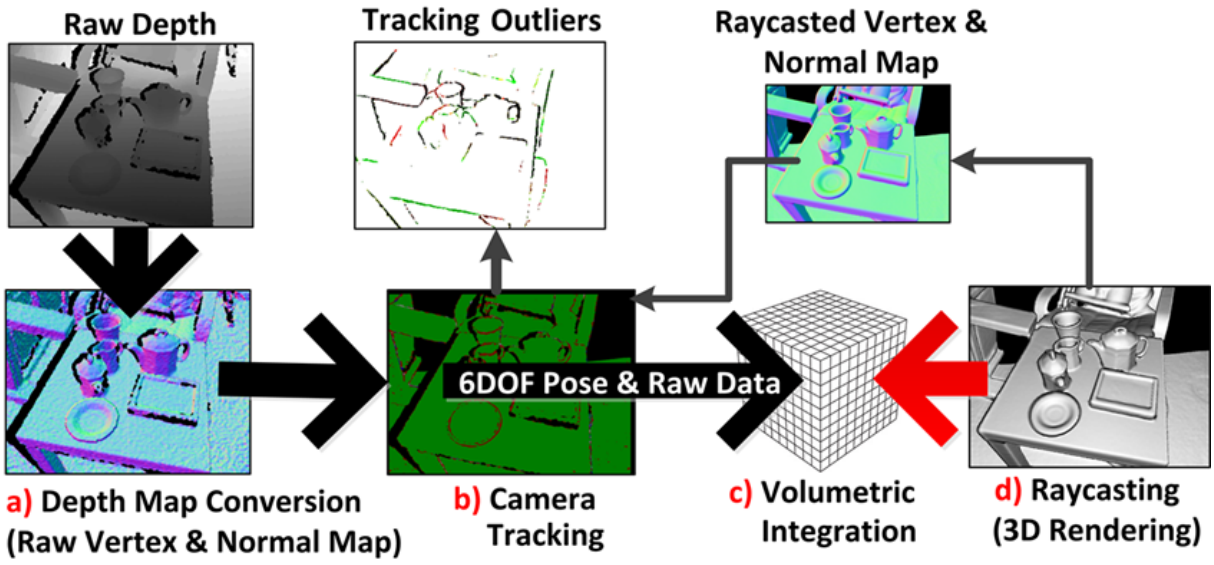

Abbildung 3.1: Pipeline von KinectFusion: beginnend beim Tiefenbild, welches die Kinect liefert (links oben), bis zur Ausgabe für den User (rechts unten)

KinectFusion ist ein System zur 3D Rekonstruktion von Objekten und Szenerien in Echtzeit. Mithilfe der Kinect-Tiefenkamera wird ein 3D Scan erstellt, der zeitgleich visualisiert wird. Hauptaugenmerk war das Nutzen der Prozessoren auf der Grafikkarte, um viele Berechnungen parallel durchführen zu können. Der Hauptteil der Pipeline verläuft über vier Schritte, die auch in Abbildung 3.1 veranschaulicht sind:

1. **Konvertierung des Tiefenbilds:** Das Tiefenbild der Kamera wird abgegriffen und dessen Werte werden in Fließkommazahlen umgerechnet. Diese Werte beschreiben die Tiefe in Metern und werden in einer gerichteten Punktwolke gespeichert.

2. **Erfassen der Kamera-Position:** Mithilfe des ICP Algorithmus wird die Position der Kamera und ihre Orientierung berechnet. Diese 6DOF[1] Position wird gespeichert und lässt sich später zum Beispiel für das Mappen der Farbbilder auf die Point

[1] Six Degrees Of Freedom; deutsch: sechs Freiheitsgrade

Cloud nutzen. Die Position der Kamera wird über die Translation und Rotation des aktuellen Frames im Bezug auf das bisher erfasste Volumen berechnet.

3. **Volumetrische Integration:** Mit der nun bekannten Kameraposition lassen sich die Daten des Tiefenbildes in globale Koordinaten umrechnen. Diese werden genutzt um das Voxelgitter des Scans zu ergänzen oder gegebenenfalls zu verbessern. Kinect Fusion speichert die Werte im Voxelgitter als gestutzte vorzeichenbehaftete Abstandsfunktion (TSDF[2]). Die Werte geben den Abstand zur nächsten Oberfläche an. Null beschreibt in diesem Fall die Oberfläche. Positive Werte sind näher, negative ferner. Da nur die Werte um null interessant sind, werden Werte, die sich nicht in unmittelbarer Nähe einer Oberfläche befinden, während der Berechnungen vernachlässigt.

4. **Raycasting:** Mittels Raycasting wird aus den Werten des Voxelgitters ein Bild für den Nutzer erzeugt. Jeder Strahl wird von einem GPU-Thread abgelaufen. Gibt es einen Vorzeichenwechsel, wird mit der entsprechenden Position und der dort anliegenden Beleuchtung ein Pixel für das Ausgabebild berechnet. Dieses synthetische Tiefenbild wird außerdem in den ICP Algorithmus eingespielt.

Alle vier Prozesse sind für das parallele Abarbeiten durch die zahlreichen Prozessoren der GPU optimiert. Dieser Ansatz ermöglicht erst das Nutzen des ICP. Dieser ist davon abhängig, dass sich die Punktmengen möglichst nah sind. Um dies zu erreichen ist es essenziell, dass der Algorithmus mehrfach pro Sekunde abläuft. Im besten Fall so oft, wie die Kamera ihre Werte liefert, also 30 mal pro Sekunde.

Die voreingestellte Größe des TSDF Voxelgitters, in dem Informationen gespeichert werden können, sowie dessen Auflösung sind die beiden Faktoren, welche den Speicherbedarf auf der Grafikkarte bestimmen. Ein Voxelgitter mit einer Auflösung von 512 x 512 x 512 benötigt 512 MB Grafikspeicher[3]. Da nicht unendlich Speicher zur Verfügung steht, muss der Nutzer sich für eine hohe Auflösung oder für ein großes Capturing Volumen entscheiden. Der Verbund von mehreren Grafikkarten über SLI[4] erlaubt keine größeren Voxelgitter. Jedoch ist es möglich parallel mehrere Volumina, also je eins pro Grafikkarte, zu erstellen.

Es gibt jedoch mehrere Projekte, die nach der Veröffentlichung des Papers (Richard (2014)) entstanden sind. Diese greifen die Konzepte von KinectFusion auf und erreichen

[2]engl. Truncated Signed Distance Function
[3]Bei einer Kantenlänge von 1024 liegt die benötigte GPU-Speichermenge bereits bei 4GB
[4]Scalable Link Interface: Multi GPU Technologie von NVIDIA

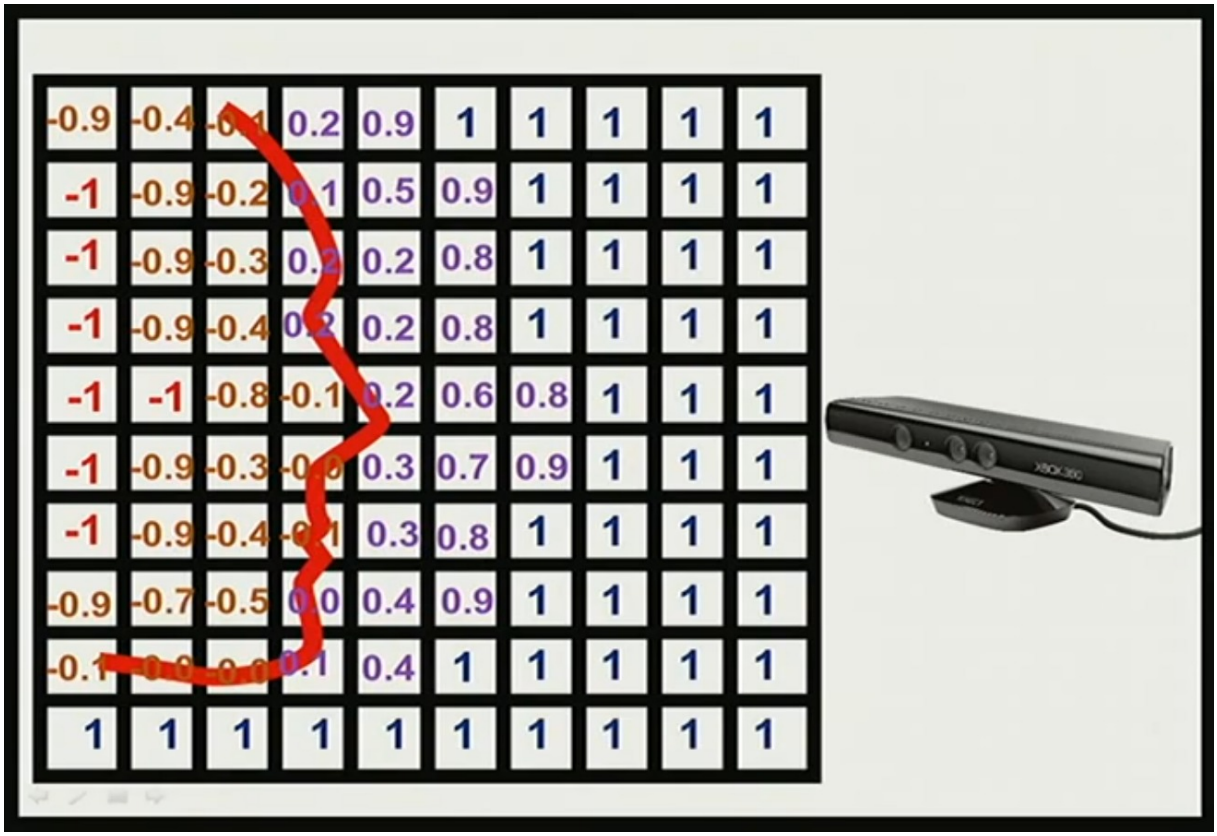

Abbildung 3.2: Truncated Signed Distance Function, zur Veranschaulichung im zweidimensionalen Raum

ähnliche, teilweise sogar bessere Ergebnisse.

3.1.1 KinFu und Erweiterung KinFu Large Scale

KinFu ist Teil der Point Cloud Library und die Erste von wenigen open-source Implementierungen des KinectFusion Systems. KinFu[5] Large Scale ist eine Erweiterung der PCL open-source Implementierung von KinectFusion. Der Rekonstruktionsraum ist in einem umfassenden Raum bewegbar, was es ermöglicht, größere Objekte oder Szenerien zu erfassen. Sobald die Kameraposition den derzeitigen Rekonstrukstionsraum verlässt, wird ein neuer erzeugt und das Tracking in diesem fortgesetzt. Die Informationen des alten Raumes werden, weil sie nicht mehr für die laufende Rekonstruktion zu berücksichtigen sind, vom GPU-Speicher zum Hauptspeicher verlegt. Dies ermöglicht KinFu Large Scale die Rekonstruktion von größeren Szenerien und Objekten ohne die Einschränkung durch das maximale Volumen des Voxelgitters.

3.1.2 KinFu Remake

Das KinFu Remake ist eine verbesserte Version des KinFu Systems der PCL. Die Reimplementierung ist 1,6 mal schneller als die PCL-eigene KinFu Version. Außerdem wurde die Lesbarkeit erheblich verbessert sowie die Menge an Code deutlich reduziert. Das Remake lässt sich zudem unabhängig von der PCL Bibliothek nutzen. Um das KinFu Remake nutzen zu können ist die Installation der Bibliothek OpenCV nötig.

3.2 Mesh Ausrichtung

Das Hauptproblem der 3D Rekonstruktion ist das Zusammenführen der erfassten Informationen. Hierzu müssen die einzelnen Punktmengen aneinandergefügt werden. Die Überschneidungen der Punktmengen werden genutzt um die einzelnen Perspektiven zu einer Menge zu vereinigen. So wird Stück für Stück das Gesamtmodell durch jede Perspektive mit neuen Punkten erweitert.

Zudem ist es möglich, über den Vergleich von zwei nacheinander aufgenommenen Punktwolken auf die Bewegung der Kamera zu schließen. Dies ist vor allem für SLAM nützlich.

[5] Siehe Kapitel 3.1.1

3.2.1 Iterative Closest Point Algorithmus (ICP)

Seit seiner Vorstellung 1992 durch Besl und McKay (Besl (1992)) ist der Itertative Closest Point Algorithmus die meist genutzte Möglichkeit zum Angleichen von dreidimensionalen Formen. Der ICP Algorithmus ermöglicht es, zwei Punktwolken einander anzunähern, sofern diese sich in ausreichend vielen Punkten überschneiden. Es ist notwendig, dass sich die Punktwolken zu Beginn des Algorithmus bereits möglichst nah sind. Der ICP Algorithmus wird hauptsächlich genutzt um eine Oberfläche aus mehreren Scans zu rekonstruieren. Der Algorithmus sucht nach der bestmöglichen Koordinatentransformation, damit die Zielmenge um die neuen Werte der aktuellen Quelle erweitert werden kann. Die aktuelle Punktwolke wird so transformiert (translation und rotation), dass für jeden gemeinsamen Punkt der Abstand zur Referenzmenge so gering wie möglichst wird. Iterativ werden die Transfomationsparameter so lange angepasst, bis sich der Abstand zwischen den Mengen nicht weiter verringert.

Es gibt eine Vielzahl von Implementierungen des Algorithmus, die vor allem im Bezug auf ihre Geschwindigkeit von Rusinkiewicz und Levoy miteinander verglichen worden. Sie gliederten den Ablauf in ihrer Arbeit (Levoy (2001)) in die folgende sechs Schritte:

1. **Auswahl** von Punktmengen aus einem oder beiden Meshes.
2. **Zuordnen** der Punktmengen zu Auswahlen aus dem anderen Mesh.
3. **Angemessene Gewichtung** der übereinstimmenden Paare.
4. **Verwerfen** von bestimmen Paaren nach individueller Begutachtung jedes einzelnen Paares sowie Begutachtung der Menge aller Paare.
5. Zuweisen einer **Fehlerfunktion**, basierend auf den Paaren.
6. **Minimieren** der Fehlerfunktion.

Alle 3D Rekonstruktionslösungen mit der Kinect nutzen eine Version des ICP. Die Microsoft KinectFusion Implementierung durchläuft die folgenden Schrittes.

1. Als aktuelle Quelle wird die Punktwolke aus dem aktuellen Tiefenbild der Kinect genutzt. Für die Zielmenge werden die im Frustum[6] des Sensors liegenden Punkte des Voxelvolumens herangezogen.

[6]Frustum einer Kamera: Pyramidenstumpf, dreidimensionaler Raum des Sichtbereichs einer Kamera

2. Die Punkte werden einander in KinectFusion nicht einfach anhand der Nähe[7] zugewiesen. Kinect Fusion nutzt „projective data association“ (Richard (2011)) um die Übereinstimmungen zu finden. Es ist davon auszugehen, dass sich die beiden Punktwolken sehr nahe sind, weil die Bewegung der Kamera zwischen zwei Frames sehr gering ist. Die aktuelle Punktmenge wird so transformiert, als entspräche die Perspektive der des vorherigen Frame. Die erste Annahme für die aktuelle Position der Kamera ist dabei die letzte bekannte Position.
   Beide Perspektiven werden nun auf eine zweidimensionale Bildebene projiziert. Die Punkte die in dieser Projektion auf dem gleichen Pixel dargestellt werden, lassen sich einander zuordnen. Diese Berechnung lässt sich für alle Pixel mit der GPU parallel durchführen.
   Diese Methode funktioniert jedoch nur, wenn die Punktmengen sich zu Beginn schon sehr nah sind. Daher darf die Kinect während des Scans nicht zu schnell bewegt werden. Außerdem muss die GPU leistungsstark sein, damit die Berechnung so oft abläuft wie die Framerate der Kamera neue Daten liefert.[8]

3. Alle gefundenen Paare werden als gleichbedeutend angesehen.

4. Verwerfen bestimmter Paare unter Berücksichtigung mehrerer Thresholds.

5. Punkt-zu-Ebene Fehlerfunktion durchlaufen (Siehe Kapitel 3.2.2).

6. Optimieren der Fehlerfunktion mittels linearer Näherung, beschrieben von Low in (Low (2004)). Nur möglich, weil die aufeinander folgenden Frames sich sehr nah sind.

Für die Geschwindigkeit des ICP Algorithmus ist maßgeblich die Fehlerfunktion verantwortlich. Nach Levoy und Rusinkiewicz (Levoy (2001)) erreicht die Punkt-zu-Ebene-Fehlerfunktion die gewünschte Näherung am schnellsten. Jedoch können glatte Oberflächen einander nicht richtig zugeordnet werden, weil sie sich gegeneinander verschieben lassen, ohne das sich die Punkt-zu-Ebene-Fehlerfunktion verändert. Dies führt zum Verlust der Kameraposition während des Scans und ist das größte Problem von KinectFusion.

3.2.2 Punkt-zu-Ebene-Fehlerfunktion

Für jede Fehlerfunktion müssen vorher Punkt-Paare zwischen der aktuellen Quelle und der Zielmenge bestehen. Im vorherigen Abschnitt 3.2.1 wurde dargelegt, wie diese Paarung in der KinectFusion Implementierung erreicht wird.

[7] Wie üblicherweise bei einer Punkt-zu-Punkt-Fehlerfunktion
[8] 30 Frames pro Sekunde

Das Ziel des ICP ist es, eine Transformationsmatrix **M** zu erhalten, welche die Punkte der Quelle so rotiert und bewegt, dass die beiden Punktwolken so gut wie möglich aneinander gefügt werden können. Speziell für die Punkt-zu-Ebene-Fehlerfunktion wird nach der Transformation gesucht, welche die Summe der quadrierten Abstände zwischen den Punkten aus der Quelle und der Tangente der dazugehörigen Punkte der Zielmenge minimiert.
In der Abbildung 3.3 sind die Punkte der Quelle $s_i = (s_{ix}, s_{iy}, s_{iz}, 1)^T$ die zugehörigen Punkte der Zielmenge $d_i = (d_{ix}, d_{iy}, d_{iz}, 1)^T$ und deren Normalenvektoren $n_i = (n_{ix}, n_{iy}, n_{iz}, 1)^T$. Dann ist es das Ziel des ICP, die optimale Matrix zu finden, für die gilt:

$$M_{opt} = arg\,min_M \sum_i ((M \cdot s_i - d_i) \cdot n_i)^2$$

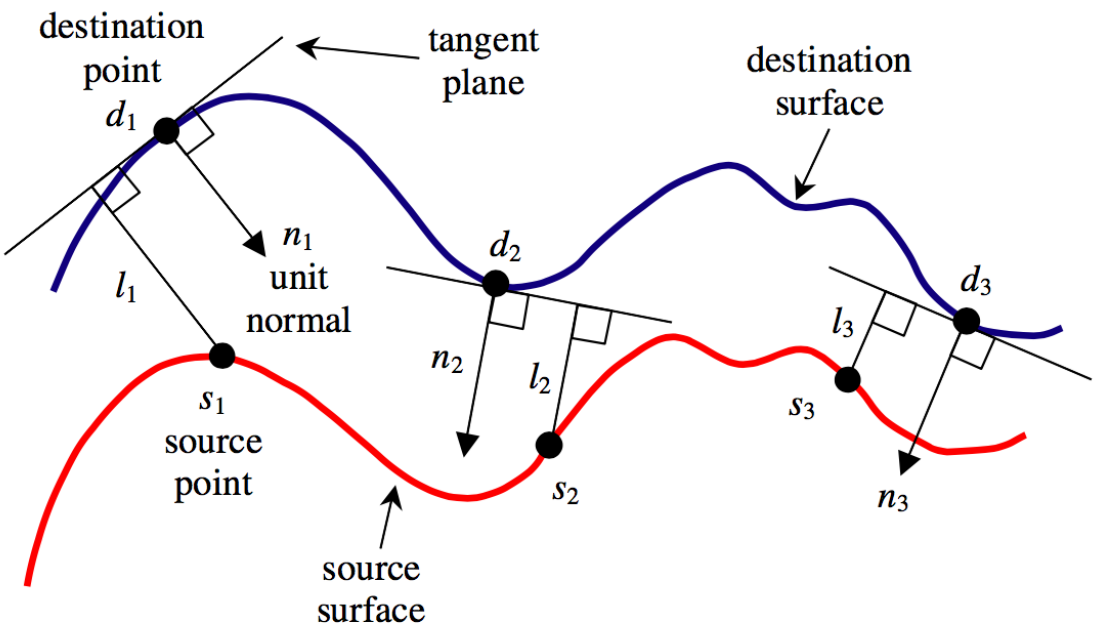

Abbildung 3.3: Punkt-zu-Ebene Abstand zweier Oberflächen

Dabei sind die Matrizen M sowie M_{opt} 4x4 Tansformationsmatrizen, welche sich zusammen setzen aus einer Rotationsmatrix $R(\alpha, \beta, \ă\gamma)$ sowie einer Translationsmatrix $T(t_x, t_y, t_z)$. Also gilt:

$$M = T(t_x, t_y, t_z) \cdot R(\alpha, \beta, \gamma)$$

mit der Translationsmatrix:

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

und der Rotationsmatrix:

$$R(\alpha, \beta, \gamma) = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$$

$$= \begin{pmatrix} \cos\gamma\cos\beta & -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha & \sin\gamma\sin\alpha + \cos\alpha\sin\beta\cos\alpha & 0 \\ \sin\gamma\cos\beta & \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha & -\cos\gamma\sin\alpha + \sin\alpha\sin\beta\cos\alpha & 0 \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In seiner Arbeit (Low (2004)) zeigte Low wie sich das Problem so umformen lässt, dass alle unbekannten Werte $\alpha, \beta, \gamma, t_x, t_y, t_z$ sich in einem Vektor x zusammenfassen lassen. So gilt:

$$\min_{\hat{M}} \sum_i ((\hat{M} \cdot s_i - d_i) \cdot n_i)^2 = \min_x |Ax - b|^2$$

wobei A eine N x 6 Matrix von linearen Kombinationen zwischen n_i und s_i ist, sowie b ein N Vektor der linearen Kombinationen von s_i, d_i und n_i.
Also kann $\hat{M}_{opt}$ gefunden werden wenn nach x aufgelöst wird:

$$x_{opt} = arg\,min_x |Ax - b|^2$$

Während Low mittels Singulärwertzerlegung dieses Standard Problem der kleinsten Quadrate löst indem x über das Pseudoinverse von A ermittelt wird[9], geht die KinectFusion Implementierung einen anderen Weg, welcher von der Parallelisierung auf der GPU profitiert (Richard (2014)).

[9] $x_{opt} = A^+ b$

4 3D Rekonstruktion mit der Kinect

Im folgenden Abschnitt wird der Ablauf des Scans und die Aufbereitung der erfassten Daten eingehend beschrieben. Zudem wird auf Einschränkungen sowie Vorteile der verschiedenen Systeme eingegangen.

4.1 Vorbereitung und Installation

Um KinFu nutzen zu können, müssen vorher einige Bibliotheken installiert werden, von denen die PCL abhängig ist. Als Treiber für den Kinect Sensor benötigt KinFu OpenNi sowie das Sensor Modul. Für das Installieren der PCL sowie der zuvor beschriebenen Komponenten ist zudem CMake nötig, sowie ein Compiler, zum Beispiel eine Version von Microsofts Visual Studio. Der Installationsablauf für ein Windows 10 System ist im Anhang ausführlich beschrieben.

Der Raum, welcher rekonstruiert werden soll, muss vorher eventuell etwas angepasst werden, damit das Scannen später einfacher abläuft und es seltener zum Abbruch kommt. Sehr kleine wie auch sehr große Räume sind mit KinFu nur schwer oder gar nicht zu rekonstruieren. Ist der Raum so klein, dass der Abstand zwischen Sensor und Objekten kleiner als die Naheinstellgrenze der Kamera ist, wird es sehr schwer bis unmöglich den Raum zu rekonstruieren. Scheint die Sonne, sollten die Fenster verdunkelt werden. Sind wenige Objekte im Raum vorhanden, sollten umbedingt zusätzliche Gegenstände im Raum positioniert werden. Der Standpunkt des Rechners und Bildschirms, sowie deren Kabel und die der Kinect, sollten so gewählt werden, dass die Bewegung im ganzen Raum möglich ist. Außerdem sollte der Bildschirm für den Nutzer immer sichtbar sein. Dieser muss während des Scans eventuell gedreht oder optimalerweise von einer Hilfsperson gehalten und mit durch den Raum bewegt werden.

Abbildung 4.1: Für die Scans verwendeter System-Aufbau auf Schubwagen

Die Position der Kamera im Voxelvolumen ist gegebenenfalls vorher so anzupassen, dass es den Scan erleichtert. Für die meisten Räume empfiehlt sich der Mittelpunkt des Raumes. Die Auflösung des Volumens, sowie dessen Größe müssen an die verfügbare Speichermenge der Grafikkarte sowie die Größe des Raumes in Metern angepasst werden.

4.2 Datenakquise

Für die 3D Rekonstruktion mit der Kinect gibt es diverse unterschiedliche Systeme. Im folgenden werden

- Kinect V1 und PCL KinFu
- Kinect V1 und PCL KinFu Large Scale
- Kinect V2 und Kinect for Windows SDK

vorgestellt und ihre Eigenschaften während des Scans miteinander verglichen.

4.2.1 Kinect V1 und KinFu

Die Auflösung und die Größe des Voxelvolumens müssen vorab an die Größe des Raumes angepasst werden, welcher rekonstruiert werden soll. Es empfiehlt sich die, Startposition des Sensors in die Mitte des Voxelvolumens zu legen und mit der Aufzeichnung ebenfalls in der Mitte des Raumes zu beginnen. Während des Scans muss darauf geachtet werden, dass die Kinect langsam bewegt wird, damit sich der aktuelle und der letzte Frame möglichst nahe sind. Wird der Raum in horizontalen Streifen aufgezeichnet, kommt es zu geringeren Fehlern. Dazu wird die Kamera langsam von oben nach unten geführt. Danach dreht sich dann der Anwender, um den nächsten Streifen zu erfassen. Der Ansatz des mehrfachen Rotierens um die eigene Achse, um weiter oben und unten liegende Eigenschaften des Raumes zu erfassen, funktioniert schlechter. Es kommt zu einer größeren Menge an Fehlern, vor allem im Bezug auf die Winkel der Wände zueinander.
Für die menschliche Wahrnehmung vollkommen eindeutige Strukturen sind für den Algorithmus nicht ähnlich leicht zu erkennen. Es ist wichtig, während des Scans zu berücksichtigen, dass der Algorithmus ausschließlich die Tiefeninformationen zur Berechnung der Kameraposition nutzt. Farben der Oberflächen sowie Muster werden nicht berücksichtigt. Sind zu wenige Informationen vorhanden, kann die Position der Kamera nicht richtig erfasst werden. Dies kann meistens, jedoch nicht immer, durch die Visualisierung auf dem Display erkannt werden. Typischerweise lässt sich ein Verschieben der Rekonstruktion um einen Punkt erkennen, welcher nicht der Kamera entspricht. Hat sich der Algorithmus einmal verloren, muss komplett von vorn begonnen werden. Denn die bereits richtig erfasste Werte würden mit neuen überschrieben werden. Das Verlieren der Kameraposition des ICP tritt vor allem bei großen glatten Oberflächen auf. Es sollte generell vermieden werden, schlecht zu erfassende Bereiche zu scannen.
Schlecht zu erfassen sind somit Räume mit wenigen Gegenständen, oder beispielsweise nur glatte Tische und Schränke. Kahle Wände lassen sich nur sehr schwer erfassen. Für

Übergängen von Decken und Böden zur Wand ist es unabdingbar, dass sich andere Gegenstände im Vordergrund befinden. Ausgenommen hiervon sind Ecken mit drei Flächen, diese reichen dem Algorithmus, um alle Freiheitsgrade zu ermitteln. Das Aufzeichnen von Decken ist nur sehr selten möglich, da für gewöhnlich wenige Objekte an Decken befestigt sind und sich zur Aufzeichnung auch nur mit viel Aufwand dort anbringen lassen.

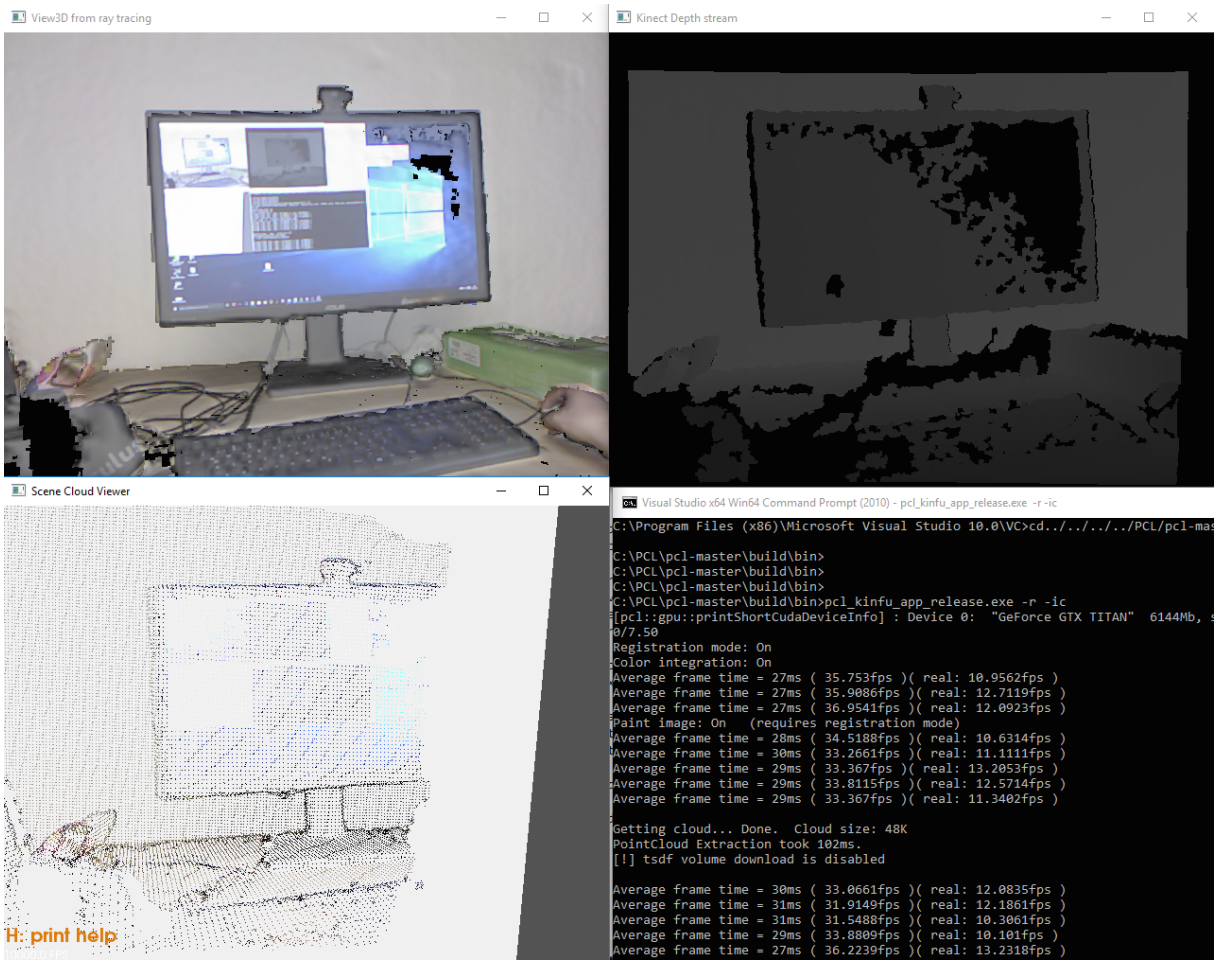

Abbildung 4.2: KinFu Bildschirmausgabe: links oben Raytracing, rechts oben Tiefenkarte, links unten Darstellung der Rekonstruktion als Punktwolke, rechts unten Kommandozeile

Um das Aufzeichnen für den Nutzer etwas zu erleichtern, wird das Tiefenbild während des Scans ausgegeben. Dies visualisiert dem Nutzer, welche Tiefeninformationen dem Algorithmus für die Berechnung zur Verfügung stehen. Bereiche mit sehr wenigen Merkmalen können so erkannt und gegebenenfalls sehr vorsichtig gescannt werden. Besonders schlecht zu erfassenden Bereiche sollte eventuell komplett ausgelassen werden, da nach Verlust der

Kameraposition ein neuer Scan begonnen werden muss.

Allein für das Erfassen der Konturen ist es nicht nötig, den Raum zusätzlich zu beleuchten. Die Kinect nutzt für das Erfassen der Tiefeninformation das Licht des sich im System befindenden Infrarotprojektors. Sind die Farben jedoch von Interesse, macht es Sinn, den Raum möglichst gleichmäßig auszuleuchten.
Da Sonnenlicht Infrarotlicht enthält, behindert dieses das Tracking erheblich. Bei Sonnenschein müssen Fenster verdunkelt werden. Eine Aufzeichnung im Freien ist somit, zumindest am Tag, nicht möglich.
Nicht alle Oberflächen reflektieren das Licht des Projektors so wie erwartet. In diesem Fall wird keine Oberfläche erkannt und es kommt zu einem Loch in der Aufzeichnung. Dies gilt besonders für sehr dunkle, glatte, oder lichtdurchlässige Oberflächen, wie zum Beispiel PC-Gehäuse oder Fenster.

Ist der Anwender mit den aufgezeichneten Daten zufrieden, lassen sich diese mithilfe von Tastatureingaben speichern. Eine Ausgabe ist als Pointcloud, Mesh oder auch direkt als TSDF möglich. Wurde die Anwendung im „Registrierungsmodus“ gestartet, lässt sich die Pointcloud inklusive Farbwerte ausgeben. Für die Ausgabe als Mesh gibt es diese Berechnung jedoch nicht.

4.2.2 Kinect V1 und KinFu Large Scale

Der Ablauf während des Erstellens einer Rekonstruktion mit KinFu Large Scale ähnelt dem Ablauf mit KinFu sehr. Weil das Rekonstruktionsvolumen immer wieder bewegt und neu initialsiert werden kann, muss die größe des Voxelgitters nicht an den Raum angepasst werden. Der Anwender muss während des Scans sehr genau die Ausgabe beachten, damit während des Volumenshifts die Kinect nicht bewegt wird. Das Auslagern des Volumens vom GPU Speicher auf den allgemeinen Arbeitsspeicher dauert mehrere Sekunden. Schon eine Bewegung der Kinect um wenige Zentimeter reicht aus, um die Funktion des ICP soweit zu stören, dass ein neuer Scan begonnen werden muss. Während einer Vielzahl von versuchen, ist es nicht gelungen, einen 6m x 6m Raum mit ähnlich vielen Merkmalen wie in der Rekonstruktion mit KinFu zu erfassen. Die Position der Kinect mit der visualisierten Position im Volumen wiederzufinden wirkte nahezu unmöglich.

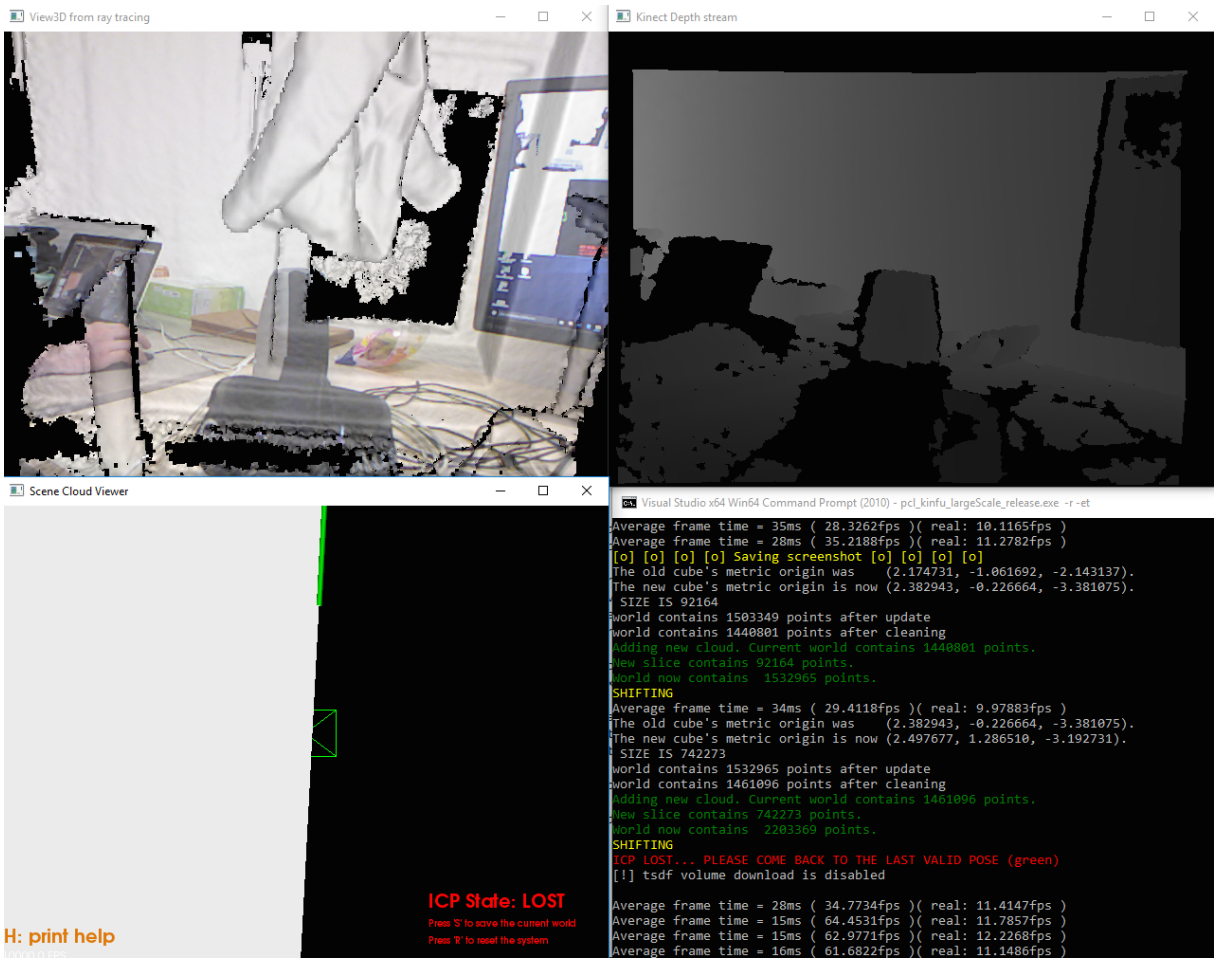

Abbildung 4.3: KinFu Large Scale Bildschirmausgabe nach Verlust der Kameraposition

4.2.3 Kinect V2 und Kinect for Windows SDK 2.0

Das Erstellen eines Scans mit dem Windows SDK gibt einen Ausblick auf die zu erwartenden Verbesserungen, die mit der verbesserten Hardware einer Kinect V2 unter KinFu möglich wären. Zur Zeit der Erstellung dieser Arbeit[1], ist die Unterstützung für die Kinect V2 Hardware noch nicht in der PCL implementiert. Zur Nutzung des KinectFusion Beispiel-Projekts muss nur das Kinect SDK 2 installiert und die Kinect V2 angeschlossen werden. In der grafischen Benutzeroberfläche hat der Anwender die Möglichkeit, die Größe des Capturing Volumens einzustellen. Hierzu müssen die Auflösung von Voxel pro Meter sowie die Voxelanzahl pro Dimension (X,Y,Z) eingestellt werden. Die Ausgabe der 3D Rekonstruktion ist als Mesh möglich, ob die Rekonstruktion entsprechend der erfassten Bilder eingefärbt werden soll, lässt sich ebenfalls einstellen.

[1] bis Februar 2016

4.3 Konvertieren

Der Inhalt des Voxelvolumens lässt sich aus KinFu entweder direkt als Punktwolke ausgeben oder zur Laufzeit vermeshen. Dabei lassen sich die Normalen des Meshes durch die Eigenschaften der TSDF sehr einfach mitberechnen.

4.3.1 Autodesk

Um die erfassten Daten mit Autodesk-Produkten weiter bearbeiten zu können, bietet sich in KinFu das Speichern des Volumens als Punktwolke an. Dafür wählt man bei Punktwolken mit Farbwerten ein ASCII-basiertes Format, zum Beispiel .PLY. Die Datei muss vorher zum Beispiel in das .XYZ Format gebracht werden, damit sie sich importieren lässt. Dazu reicht es aus, die Datei in einem Texteditor von ihrem Head zu befreien sowie die Dateiendung anzupassen. Der Inhalt der Datei sollte nun ausschließlich aus den X-, Y-, und Z-Koordinaten und den dazu gehörigen RGB-Farbwerten bestehen.
Das Programm ReCap wurde von Autodesk als Importschnittstelle entwickelt. Die Formate von vielen Laserscannern lassen sich direkt importieren. Bevor die Daten zur Nutzung in anderen Autodesk Programmen als ReCap Projekt exportiert werden, lassen sich die Punktwolken säubern und bearbeiten.

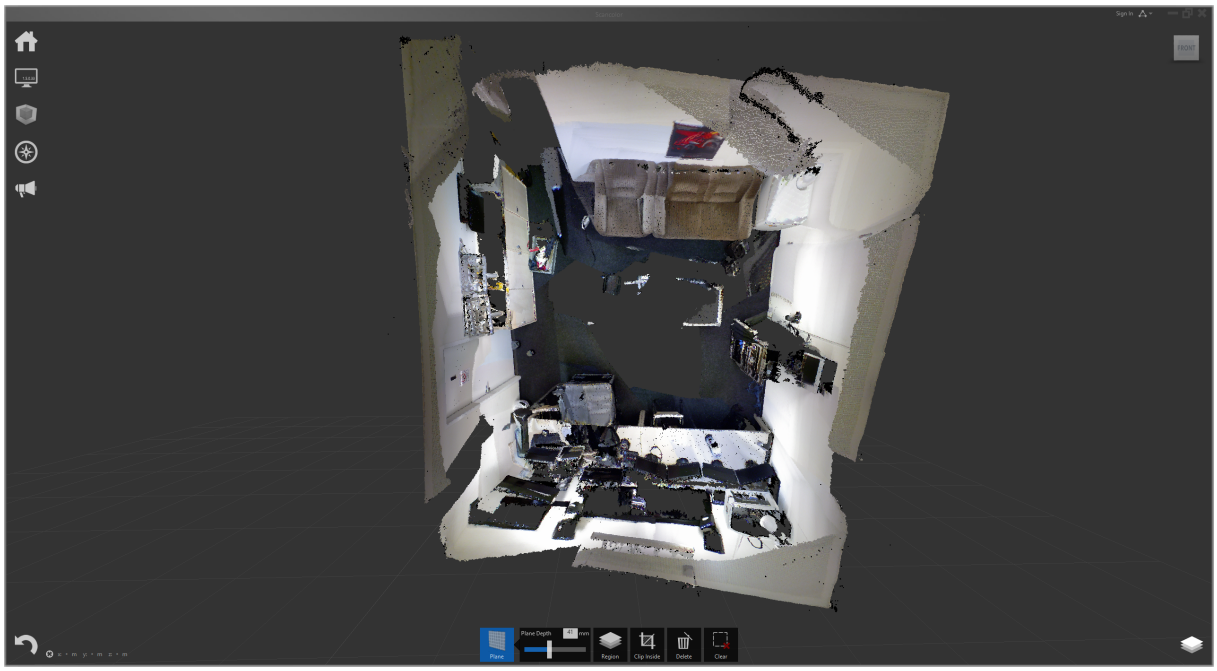

Abbildung 4.4: Farbige Punktwolke nach Import in Recap

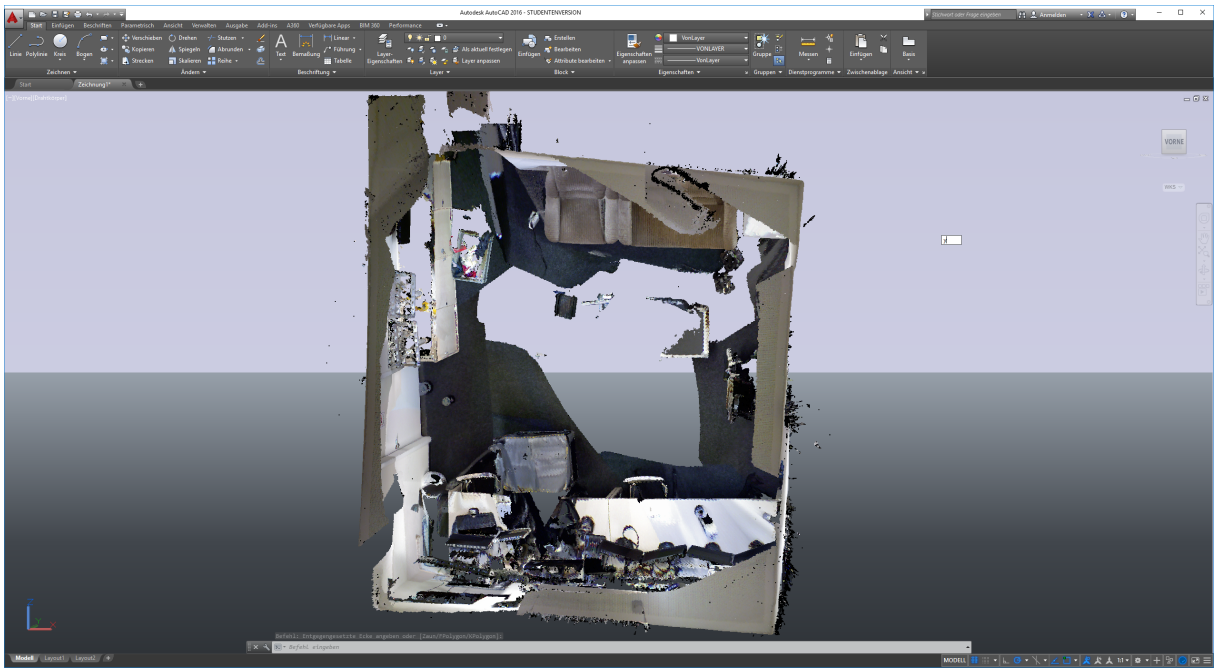

Abbildung 4.5: Farbige Punktwolke nach Import in AutoCAD

4.3.2 Unity

Um die erstellte 3D Rekonstruktion in Unity nutzen zu können, empfiehlt sich das Speichern des Voxelvolumens als Mesh in einer .PLY Datei. Beim Speichern als Mesh werden leider die Farbwerte nicht mit berücksichtigt. Die .PLY kann nun zum Beispiel mittels Meshlab zu einer .OBJ konvertiert werden. Als .OBJ Datei lässt sich das Mesh als Asset in Unity importieren und ohne weitere Anpassungen nutzen. Mittels Unity lässt sich die 3D Rekonstruktion in Head Mounted Displays, wie etwa der Oculus Rift, darstellen.

5 Ergebnis

Das Projekt hat verdeutlicht, dass die Rekonstruktion von Räumen auch mit dem, eigentlich als Unterhaltungselektronik entwickelten, Windows Kinect System möglich ist.
Das Nutzen der KinectFusion Implementierungen ermöglicht die Ausgabe einer nutzbaren 3D Rekonstruktion von Räumen, obwohl der Preis der Kinect weit unter den Preisen der für diesen Anwendungszweck üblichen Geräten liegt.

Auch wenn der Aufwand für den Anwender größer ist, lässt sich mit erheblich geringeren Kosten, als mit einem professionellen Laser 3D Scanner, ein brauchbares 3D Modell erstellen. Dieses kann zu diversen Veranschaulichungszwecken genutzt werden.

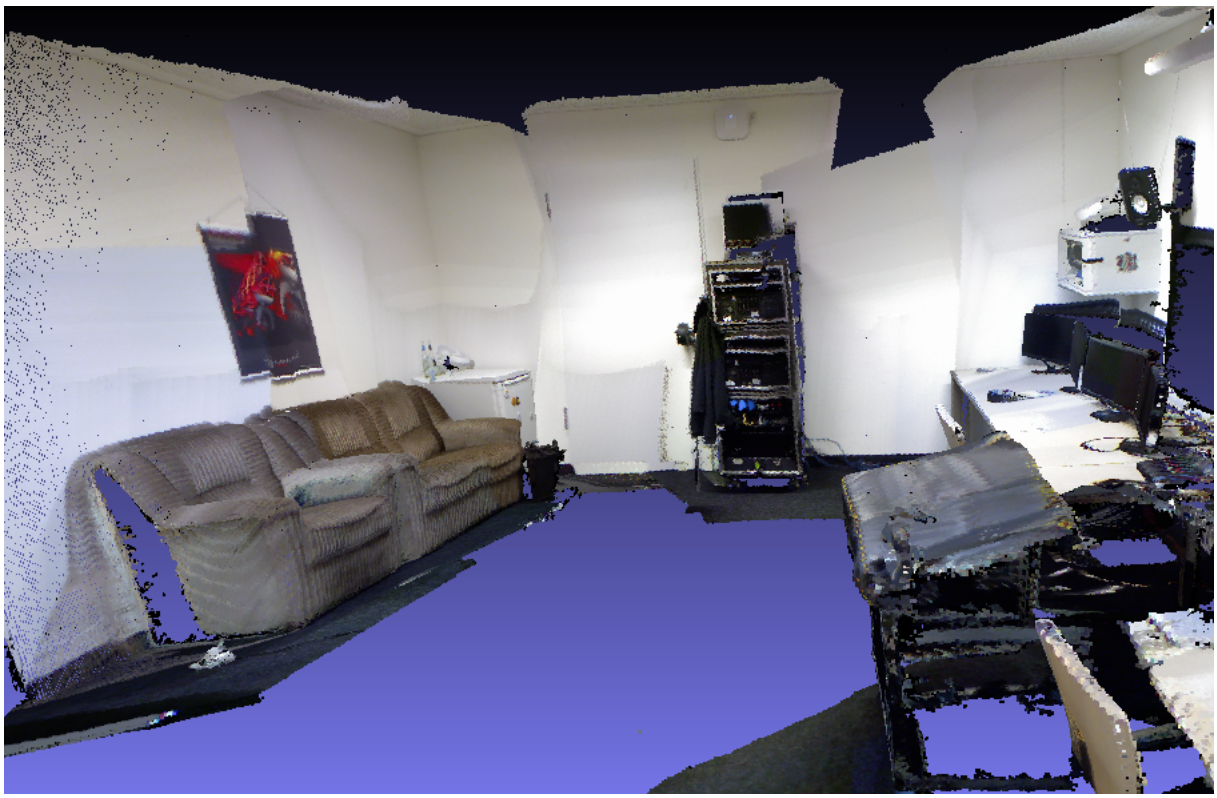

Abbildung 5.1: Pointcloud der 3D Rekonstruktion des Presetting Raumes des Lichtlabors am Mediencampus der HAW

5.1 Messgenauigkeit

Die Messgenauigkeit des Systems im Bezug auf die Größe des Raumes ist besser als erwartet. Während die Verzerrungen natürlich zu diversen Ungenauigkeiten führen, sind die Abstände zwischen Wänden und Decken sehr nah an der Realität. Wenn an Stellen gemessen wird, an denen es nicht zu Verschiebungen oder anderen Fehlern während des Scans gekommen ist, liegen die Werte bei +/- 10 cm.

Die folgende Tabelle vergleicht die Abstände zwischen Punkten an den Wänden sowie Decke und Boden der Punktwolke mit den nachgemessenen Werten. In der Punktwolke wurde für jede Distanz jeweils an fünf Stellen gemessen und danach der Mittelwert berechnet.

In Zentimetern	Punktwolke	Realität
Breite	∅ 449	454
Tiefe	∅ 491	484
Höhe	∅ 260	260

5.2 Einschränkungen

Die während der Aufzeichnung des Raumes auftretenden Probleme lassen sich grob in zwei Kategorien teilen. Auf der einen Seite entstehen Fehler in der Rekonstruktion durch Fehler während der Berechnung des ICP. Auf der anderen Seite kann es aufgrund von optischen Problemen schon während der Aufzeichnung der Tiefendaten selbst zu Löchern in der Rekonstruktion kommen. Dies vermindert zusätzlich die Stabilität des ICP in schlecht zu erfassenden Bereichen. Die auftretenden Probleme werden im folgenden näher beschrieben.

5.2.1 Optische Probleme während der Aufzeichnung

m 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.

- Ist das getroffene Material sehr dunkel, wird die Infrarotstrahlung absorbiert. Auch in diesem Fall kommt es zu einem Loch in der Punktwolke, da keine Tiefendaten aufgezeichnet werden können.
- Ist das getroffene 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.

Bei starker Sonneneinstrahlung ist die Intensität 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.

Abbildung 5.2: Löcher in der Punktwolke, durch Sonnenlicht und Fenster

Aufgrund der automatischen Rekalibrierung der Belichtung, sowie des Weißabgleichs, kommt es während des Scans oft zu falsch aufgezeichneten Farben. Dies ist in Abbildung 5.1 gut zu sehen.

5.2.2 Berechnungsprobleme während der Aufzeichnung

Das größte Problem aller genutzten Implementierungen von KinectFusion ist das Verlieren der Kameraposition während des Scans. Für den Verlust der Kameraposition gibt es zwei Ursachen. Zum Einen tritt dies auf, wenn in der Szenerie beziehungsweise im aktuellen Kamerafrustum zu wenige Objekte erkennbar sind. Zum anderen verliert sich der Algorithmus, wenn die Kamera zu schnell bewegt wurde, weil sich das aktuelle Tiefenbild zu stark vom Vorherigen unterscheidet.

Die aufgrund ihrer Geschwindigkeit genutzte Punkt-zu-Ebene-Fehlerfunktion trägt leider oft zum Verlust der Kameraposition bei. Liegen hauptsächlich glatte Flächen im sichtbaren Bereich, kann eine parallele Verschiebung zu diesen nicht ausgemacht werden. Dieses Problem entsteht aufgrund der Zuordnung von Quellpunkten zur Tangente der Zielpunkte (siehe Abschnitt 3.2.2). Lassen sich die Tiefenstrukturen gegeneinander verschieben, können die Freiheitsgrade der Kamera nicht mehr zureichend eingeschränkt werden. Es kommt zum Verlust der Kameraposition. In den meisten Fällen muss nun der Scan erneut begonnen werden.

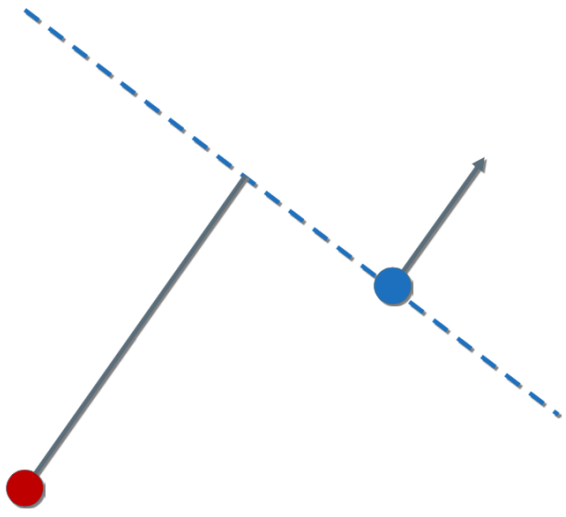

Abbildung 5.3: Seitliche Verschiebung aufgrund der Zuordnung von Punkt zu Tangente

Aufgrund der fehlenden Loop Closure[1] in den KinectFusion Implementierungen kommt es zu Fehlern in der Punktwolke. Nach der Rotation um 360° werden die zu Beginn aufgezeichneten Werte überschrieben. Aufgrund der nicht optimalen Bestimmung der Kameraposition werden die Werte durch falsche Werte überschrieben. Durch die Ungenauigkeit kommt es zu einer Verzerrung der Flächen, die sich vor allem auf die Bodenhöhe auswirkt. Das Anhäufen der Messfehler macht sich in nicht parallel verlaufenden Wänden bemerkbar.

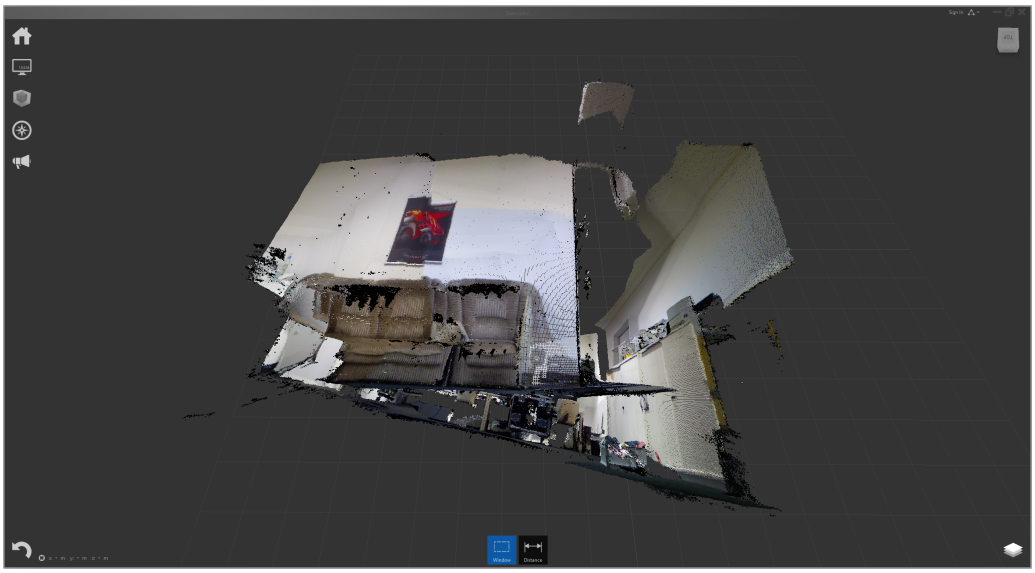

Abbildung 5.4: Verzerrung Bodenhöhe

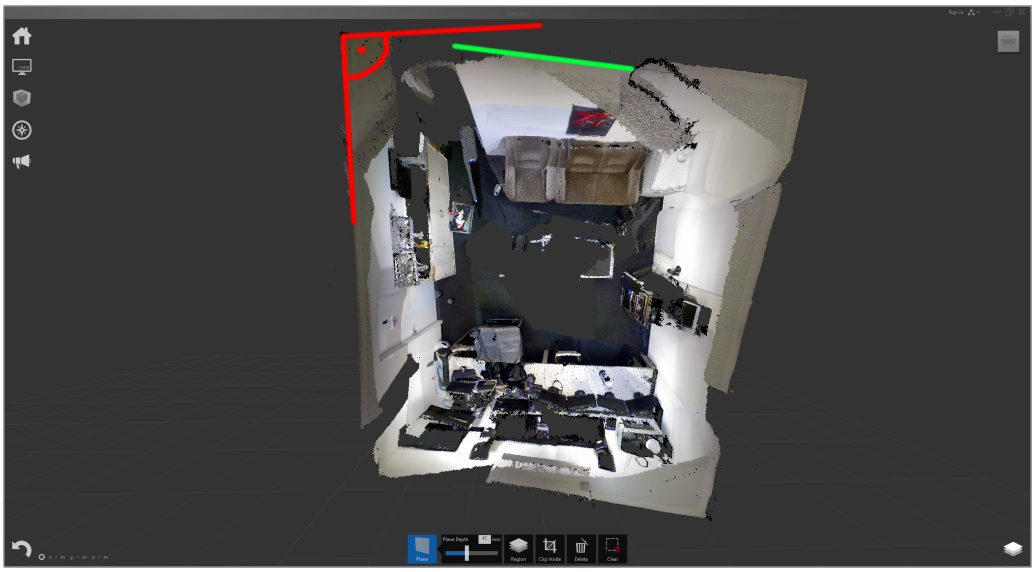

Abbildung 5.5: Verzerrung Winkel Wände

[1] Wiedererkennen bereits aufgezeichneter Werte

6 Ausblick

In seiner Diplomarbeit (Schneider (2014)) hat Schneider ein System zur Rekonstruktion von beliebig großen Innenräumen mit der Kinect V2 entwickelt. Als Grundlage diente ihm dazu die PCL KinectFusion Implementierung KinFu Large Scale. Die Performance wurde so weit verbessert, das es nicht mehr zu Abbrüchen durch den Verlust der Kameraposition kommt, wenn die Kamera in ein neues Volumen bewegt wird. Zusätzlich werden die erfassten Daten nach dem Scan, durch eigens erstellte Algorithmen nachbearbeitet, um Fehler und Überschreibungen zu verringern. Die Messgenauigkeit nimmt durch die Nachbearbeitung erheblich zu. Durch das Nutzen einer Vielzahl von kleinen Volumen kann die Auflösung während des Scans erheblich höher sein, weil nur die Daten des aktuellen Volumes im Speicher der GPU liegen müssen. Die vorher erfassten Volumina lassen sich auslagern und werden erst in der Nachbearbeitung wieder benötigt. Zudem kann durch das kleine Volumen ein System mit geringerer Rechenleistung genutzt werden. Es kann auch ein leistungsstarkes Notebook ausreichen. Ein mobiler Computer verbessert die Beweglichkeit des Anwenders und somit auch die Qualität der Rekonstruktion. Das Nutzen der Kinect V2 ermöglicht zudem ein Erkennen von kleineren Strukturen und die höhere Auflösung verbessert die Qualität der Tiefenkarte. Schneider veröffentlichte den Code nicht, sondern nutzte seine Ergebnisse, um die Software Roomplan[1] zu entwickeln.

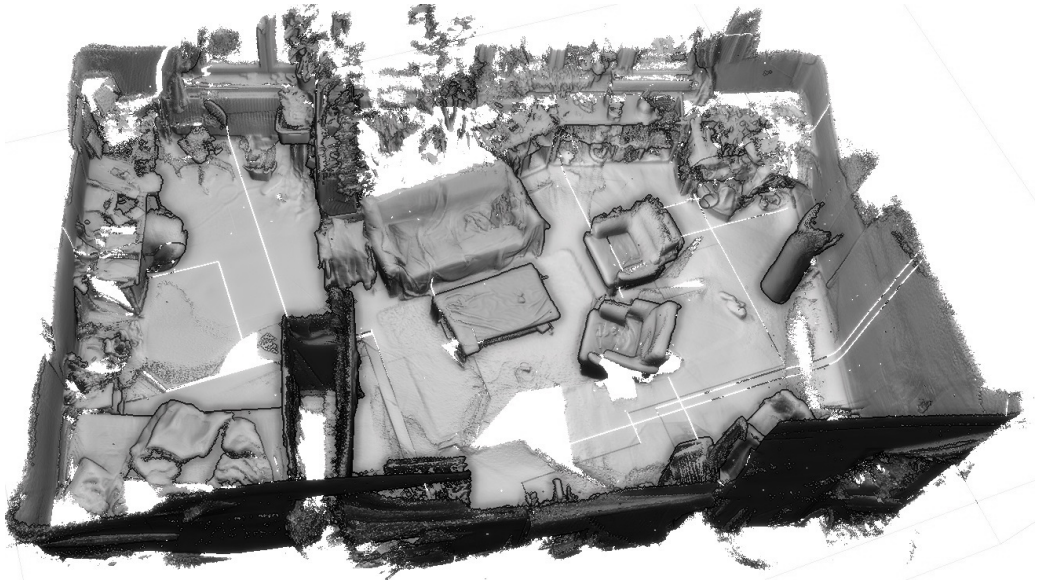

Abbildung 6.1: Rekonstruktion eines Raumes, Max Schneider

[1] Grundriss- und CAD-Modell-Erstellung in Echtzeit mittels Microsoft Kinect oder Projekt Tango Mobilgerät

Abbildungsverzeichnis

Literaturverzeichnis

Unbekannt: *PCL KinFu GitHub*, `https://github.com/PointCloudLibrary/pcl/tree/master/gpu/kinfu`, 2015, letzter Zugriff: 22. 02. 2016

Schneider, Max: *Erstellung von Innneraum-3D-Modellen mit dem Kinect for Windows v2 Sensor*, 2014

Hambüchen, Niklas: *HouseScan: Building-scale interior 3D reconstruction with KinectFusion*, 2014

Rusinkiewicz, Szymon & Levoy, Marc: „Efficient Variants of the ICP Algorithm“, *Presented at the Third International Conference on 3D Digital Imaging and Modeling*, 2001

Low, Kok-Lim: „Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration“, *Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill, February*, 2004

Besl, Paul J. & McKay, Neil D.: „A Method for Registering 3-D Shapes“, *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 2 Feb.*, 1992

Richard, A. Newcombe & Shahram, Izadi & Otmar, Hilliges & David, Molyneaux & David, Kim & Andrew, Davison & Pushmeet, Kohli & Jamie, Shotton & Steve, Hodges & Andrew, Fitzgibbon: „KinectFusion: Real-Time Dense Surface Mapping and Tracking“, *ACM Symposiom on User Interface Software and Technology*, Oktober 2011

Richard, A. Newcombe & Shahram, Izadi & Otmar, Hilliges & David, Molyneaux & Andrew, Davison & Dustin, Freeman & Andrew Davison & Pushmeet, Kohli & Steve, Hodges & Andrew, Fitzgibbon: „KinectFusion: Real-Time Dense Surface Mapping and Tracking“, *IEEE ISMAR*, Oktober 2011

Whelan T. & Kaess M. & Fallon M. & Johannsson H. & Leonard J. & McDonald J.; *Kintinuous: Spatially extended kinectfusion*, 2012

Lau, Daniel: *The Science Behind Kinects or Kinect 1.0 versus 2.0*, `http://http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php`,2013, letzter Zugriff: 12. 01. 2016

Heredia, Francisco & Favier, Raphael: *KinFu Large Scale*, `http://www.pointclouds.org/blog/srcs/fheredia`,2013, letzter Zugriff: 12. 02. 2016

Heredia, Francisco & Favier, Raphael: *Using Kinfu Large Scale to generate a textured mesh*, `http://pointclouds.org/documentation/tutorials/using_kinfu_large_scale.php`,2013, letzter Zugriff: 24. 02. 2016

Riisgaard, Søren& Rufus Blas, Morten: *SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping*, 2005

Li, Larry: *Time-of-Flight Camera, an Introduction*, 2014

Unbekannt: *Point Cloud Library, PCL*, `http://pointclouds.org/`, letzter Zugriff: 22.02.2016

Unbekannt: *Point Cloud Library Documentation, PCL*, `http://pointclouds.org/documentation/`, letzter Zugriff: 22.02.2016

Unbekannt: *PCL User Mailing List, PCL*, `http://www.pcl-users.org/`, letzter Zugriff: 22.02.2016

Wu, Han: *PCL Build Guide*, `https://mediaic.hackpad.com/ep/pad/static/e2Rbx4nSCyi`, letzter Zugriff: 22.02.2016

Unbekannt: *Autodesk Knowledge Network*, `https://knowledge.autodesk.com/`, letzter Zugriff: 22.02.2016

Unbekannt: *MeshLab*, `http://meshlab.sourceforge.net/`, letzter Zugriff: 27.02.2016

Pirovano, Michele: *Kinfu ? an open source Implementierung of Kinect Fusionää + case study: implementing a 3D scanner with PCL*, 2012

Unbekannt: *Microsoft Developer Network*, `https://msdn.microsoft.com/en-us/library/dn188670.aspx`, letzter Zugriff: 15.02.2016

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Sean Milbach

A Install Guide

Die Installation der PCL hat erheblich länger gedauert als erwartet. Der All-in-One-Installer wurde nicht genutzt, da die Interessanten Projekt, wie KinFu und KinFu Large Scale, nicht in der Version 1.6.0 der PCL beinhaltet sind. Da es vom Aktuellen stand der PCL keinen offiziellen All-in-One-Installer gibt, muss die Library vom Anwender selbst kompiliert werden. Um diesen Vorgang zu vereinfachen wurde dieser Install Guide verfasst.

Das System auf dessen Grundlage diese Anleitung erstellt wurde beinhaltete:

Betriebssystem	Windows 10 x64
Grafikkarte	Nvidia GTX Titan
Compiler	Microsoft Visual Studio 2010 x64
RGB-D Kamera	Microsoft Kinect V1 (Xbox 360)

A.1 Dependencies

Die PCL selbst nutzt diverse Bibliotheken von denen sie abhängig ist. Diese müssen vorab Installier werden. Zudem wird das Tool CMake für die Installation benötigt. Es empfiehlt sich Microsoft Visual Studio 2010 zur Installation zu nutzen. Die ermöglicht es für die Dependencies auf die Installer auf der PCL Download Seite zurück zu greifen. Das selbst Bauen der einzelnen Dependencies dauert teilweise Stunden. Die Reihenfolge der Auflistung ist nicht die einzige mögliche Installationsreihenfolge, jedoch kann es durch andere Installationsabläufen zu Problemen kommen.

Alle Dependencies sollten für die gleiche Architektur gebaut werden um Probleme zu vermeiden. Ist das System 64 Bit sollten auch alle Dependencies für 64 Bit sein.

CMake Wird für die Installation benötigt. Download und Installation genügt.
https://cmake.org/download/

CUDA Wird mit Visual Studio 2010 gearbeitet ist die Version von CUDA nicht relevant. Es empfiehlt sich die aktuellste Version zu installieren.Download und Installation im Defaultverzeichnis genügt.
https://developer.nvidia.com/cuda-downloads

Boost 1.5.0 Installer ist auf der Downloadseite der PCL verfügbar. Download und im

Defaultverzeichnis Installation genügt.
http://www.pointclouds.org/downloads/windows.html

Eigen 3.0.5 Installer ist auf der Downloadseite der PCL verfügbar. Download und im Defaultverzeichnis Installation genügt.
http://www.pointclouds.org/downloads/windows.html

Flann 1.7.1 Installer ist auf der Downloadseite der PCL verfügbar. Download und im Defaultverzeichnis Installation genügt.
http://www.pointclouds.org/downloads/windows.html

Perl Zur Konfiguration von Qt wird Perl benötigt. Download und Installation im Defaultverzeichnis genügt.
http://strawberryperl.com/

Qt 4.8.0 Vor der Installation von Qt sollte Perl Installiert worden sein. Die Archivdatei von Qt 4.8.0 kann von der Qt Downloadseite heruntergeladen werden.
https://download.qt.io/archive/qt/4.8/4.8.0/

1. Verzeichnis erstellen unter C:/Qt/4.8.0 (für Qt 4.8.0)
2. Archivdatei in diesem Verzeichnis entpacken
3. Visual Studio 2010 x64 Command Promt mit Administratorrechten öffnen (Rechtsklick)
4. In der Kommandozeile mit cd ins Verzeichnis wechseln in dem Qt entpackt wurde. In diesem fall
   prompt> cd C:/Qt/4.8.0
5. Für einen minimalen build. Wenn ein Custom Build benötigt wird müssen diese Optionen angepasst werden
   prompt> configure -opensource -confirm-license -fast -debug-and-release -nomake examples -nomake demos -no-qt3support -no-xmlpatterns -no-multimedia -no-phonon -no-accessibility -no-openvg -no-webkit -no-script -no-scripttools -no-dbus -no-declarative
6. Zum bauen von Qt
   prompt> nmake
7. Zum bereinigen des Verzeichnisses
   prompt> nmake clean

8. Bevor VTK installiert wird sollte noch die Systemvariable für Qt Gesetzt werden.
   QtDir = C:/Qt/4.8.0

VTK 5.8.0 with Qt support
Vor der Installation MUSS Qt installiert worden sein!!!
Installer ist auf der Downloadseite der PCL verfügbar. Download und Installation im Defaultverzeichnis genügt.
http://www.pointclouds.org/downloads/windows.html

QHull 2011.1 (6.2.0.1385) Installer ist auf der Downloadseite der PCL verfügbar. Download und Installation im Defaultverzeichnis genügt.
http://www.pointclouds.org/downloads/windows.html

OpenNI 1.5.4 (patched) Installer ist auf der Downloadseite der PCL verfügbar. Download und Installation im Defaultverzeichnis genügt.
http://www.pointclouds.org/downloads/windows.html

Sensor 5.1.0 (patched) Installer ist auf der Downloadseite der PCL verfügbar. Download und Installation im Defaultverzeichnis genügt.
http://www.pointclouds.org/downloads/windows.html

Es sollte nun überprüft werden ob alle Verzeichnisse an der richtigen Stelle im System erzeugt wurden.

C: / Qt / 4.8.0
C: / Program Files / Boost
C: / Program Files (x86) / Eigen
C: / Program Files / VTK 5.8.0
C: / Program Files / qhull
C: / Program Files / flann
C: / Program Files / OpenNI
C: / Program Files / NVIDIA GPU Computing Toolkit / CUDA / v7.x

Danach sollte überprüft werden ob die Systemvariablen richtig gesetzt worden. Gegebenenfalls sind diese zu verbessern oder zu hinzuzufügen.

BOOST_ROOT	C: / Program Files / Boost
CUDA_PATH	C: / Program Files / NVIDIA GPU Computing Toolkit / CUDA / v7.x
EIGEN_ROOT	C: / Program Files (x86) / Eigen
FLANN_ROOT	C: / Program Files / flann
OPEN_NI_BIN64	C: / Program Files / OpenNI/ bin64
OPEN_NI_LIB64	C: / Program Files / OpenNI/ lib64
QHULL_ROOT	C: / Program Files / qhull
QTDIR	C: / Qt / 4.8.0
VTK_DIR	C: / Program Files / VTK 5.8.0 / lib / VTK-5.8

Außerdem sollte der Path aktualisiert werden. Ans Ende anfügen:

% BOOST_ROOT% \lib;% EIGEN_ROOT% \bin;% FLANN_ROOT% \bin; C: \Qt \4.8.0 \bin; C: \Program Files \VTK 5.8.0 \bin;% QHULL_ROOT% \bin;% OPEN_NI_BIN64 %;

Die Installation der Dependencies ist damit abgeschlossen.

A.2 PCL

Nachdem die Dependencies installiert und überprüft worden müssen noch zwei Einstellungen in den CMake files der PCL 1.8.0 vorgenommen werden.//

1. Öffne die Datei ..\pcl-master \CMakeLists.txt
   Suche die Zeile 346: set (PCL_QT_VERSION 5 CACHE STRING „Which QT version to use“)
   Ersetze sie durch: set (PCL_QT_VERSION 4 CACHE STRING "Which QT version to use")
   Datei speichern und schließen.
   Diese Änderung wird vorgenommen weil Qt 4.8.0 Installiert wurde.

2. Öffne die Datei ..\pcl-master \cmake \pcl_find_boost.cmake
   Suche die Zeile 36: set (BOOST_REQUIRED_MODULES system filesystem thread date_time iostreams)
   Ersetze sie durch: set (BOOST_REQUIRED_MODULES system filesystem thread date_time iostreams chrono)
   Datei speichern und schließen.
   Diese Änderung ist wichtig damit die Chrono Datein gefunden werden, da es sonst zu Problemen mit dem Linker kommt.

Nun zur Installation der PCL. Die Archivdatei kann von Git heruntergeladen werden. Entweder ein bestimmter Zweig oder aber wie in diesem fall der Master branch. Neues Verzeichnis anlegen zum Beispiel unter C:/ PCL und das Archiv hier entpacken.
https://github.com/PointCloudLibrary/pcl

1. Öffne die CMake GUI.
2. Unter „Where is the source code:“ den Pfad zur PCL angeben: „C:/PCL“
3. Unter „Where to build the binaries:“ zum Beispiel „C:/PCL/build“ eingeben
4. Checkbox „Grouped“ und „Advanced“ anwählen und auf „Configure“ gehen
5. Im Dialog den Compiler VS 2010 64 bit auswählen
6. Im nun roten oberen Teil den Ordner „Build“ ausklappen
7. Haken setzen bei den Optionen „BUILD_CUDA“ und „BUILD_GPU“
8. Haken entfernen bei „OPENNI2“
9. Haken entfernen bei „BUILD_apps“
10. Erneut auf „Configure“ gehen
11. Wenn keine Probleme aufgetreten sind auf den Button „Generate“ klicken

Sollten während der Konfiguration Probleme im Log angezeigt werden sollten sie behoben werden. Die Bibliothek libusb wird NICHT benötigt auch wenn es die Ausgabe „NOT FOUND“ gibt. OpenNi und Sensor wurden als Treiber für die Kinect installiert.

Die nun generierten Projekte befinden sich am vorher eingestellten Pfad. Visual Studio 2010 mit Administratorrechten ausführen und die .sln Datei öffnen.

Nun kann entweder ein einzelnes Projekt, zum Beispiel die KinFu gebaut werden oder aber über BUID_ALL alles. Es muss einmal im Debug modus und einmal im Release gebaut werden. Danach wieder für Debug und Release auf INSTALL den Build ausführen. Hierbei sollten keine Fehler auftreten und es sollte ungefähr 140 Projekte gebaut werden.

Unter C:/PCL/build/bin sollten sich nun die Apps befinden und können ausgeführt werden

Über die Konsole lässt sich die KinFu exe, mit dem Flag „-r“ hinter der exe, zudem im „registration Mode“ starten. Dann lässt sich während des Scans mit der Eingabe „*“ den aufgezeichneten Punkten die entsprechende erkannte Farbe zuweisen. Wird die Pointcloud ausgegeben, werden die Farbwerte für jeden Punkt ebenfalls gespeichert.