FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-Univerität Magdeburg

Fakultät für Informatik

Analyse und Simulation von Time-of-Flight Sensoren mittels GPU Path Tracing

Masterarbeit

Autor:

Artur Schütz

Erstprüfer und Betreuer:

Jun.-Prof. Dr. Christian Lessig

Zweitprüfer:

Hon.-Prof. Dr.-Ing. Klaus Richter

Betreuer:

Martin Hünermund

Thomas Depner

Magdeburg, 16.07.2019

Inhaltsverzeichnis

Kurzzusammenfassung

Bei der Simulation von Time-of-Flight Sensoren handelt es sich um ein komplexes Problem, das hohe Anforderungen an die physikalische Plausibilität stellt. Bei der physikalsich plausiblen Berechnung der Lichtausbreitung handelt es sich um ein eingehend erforschtes Feld, das bisher nur selten in der Simulation von Time-of-Flight Sensoren Berücksichtigung fand. Die akkurate Berechnung direkter und indirekter Beleuchtung wurde zum Standard nicht-interaktiver Anwendungen, die beispielsweise beim Rendering von Filmen eingesetzt werden, während Time-of-Flight Sensor Simulationen bisheriger Arbeiten zugunsten der Reduktion der Berechnungszeit auf die korrekte Berechnung der Tiefenwerte verzichteten. Die Implementierung von Algorithmen auf der GPU bietet die Möglichkeit, Verfahren, die bisher in nicht-interaktiven Anwendungen verwendet werden, parallel auf tausenden von Kernen ausführen zu lassen, wobei die Berechnungszeit verkürzt wird, sodass diese Methoden auch in interaktiven Anwendungen genutzt werden können. Dies schlägt die Brücke zwischen physikalisch plausibler Berechnung der Lichtausbreitung und Echtzeitanwendungen, die bisher auf die korrekte Berechnung indirekter Beleuchtung verzichteten.

In dieser Arbeit wird ein Time-of-Flight Sensor präsentiert, der mittels OptiX auf der GPU implementiert wurde und die Lichtausbreitung physikalisch plausibel simuliert. Dabei wird ein Path Tracing Algorithmus genutzt, der auf der GPU parallelisiert wurde und somit deutlich kürzere Berechnungszeiten ermöglicht, während die Berechnung von indirekter Beleuchtung und eine akkurate Simulation der Oberflächen wie in nicht-interaktiven Anwendungen durchgeführt wird. Dazu ist keine Vorberechnung notwendig, weshalb die Parameter der Kamera und der simulierten Umgebung zur Laufzeit der Applikation angepasst werden können. Dazu wird die Berechnung der Simulation progrssiv durchgeführt, sodass der Nutzer ein Feedback über die Zwischenergebnisse erhält und bereits vor der abschließenden Simulation der Szene das mögliche Ergebnis beurteilen und Anpassungen vornehmen kann.

Diese Arbeit führt den Leser in die Theorie der globalen Beleuchtung und die interne Funktionsweise von Time-of-Flight Sensoren ein und stellt eine ausführliche Analyse der Fehler verschiedener Sensoren vor, die im Anschluss in einer Simulation nachgebildet werden. Die abschließende Evaluation vergleicht die Simulation mit der Kinect v2 und zeigt Übereinstimmungen der simulierten Tiefenwerte mit denen der echten Time-of-Flight Kamera Systeme. Verbesserungen können besonders in der Reduktion der Berechnungszeit der Oberflächensimulation durchgeführt werden, da diese Arbeit den Fokus auf eine möglichst korrekte Simulation der Beleuchtung legt.

1 Einführung

(a) Farbbild **(b)** Infrarotbild **(c)** Tiefenbild

Abbildung 1.1: Die einzelnen Kamera Streams, die von der Microsoft Kinect v2 geliefert werden.

1.1 Einleitung

Time-of-Flight Kameras haben in der Wissenschaft besonders nach der Veröffentlichung der Kinect v2 deutlich an Bedeutung gewonnen, da diese den bisherigen Technologien in Geschwindigkeit und Genauigkeit auch auf große Entfernungen überlegen waren und zudem durch den niedrigen Preis für Endnutzer und die Forschung interessant wurden. So fanden die Tiefensensoren, die ursprünglich für Unterhaltungsbereich konzipiert wurden, auf Gebieten wie der Robotik und Medizin Anwendung [MNK13][HF14][VMFGAL+17]. Auch in der Logistik gewinnen preiswerte und robuste Kamera Systeme an Bedeutung für den Endnutzer, da diese zur flächendeckenden Überwachung eingesetzt werden und zuverlässig korrekte Werte liefern sollten, wobei sie um anderen günstig in der Anschaffung sind. Bei Time-of-Flight Kameras handelt es sich um Tiefensensoren, die mittels kurzer Lichtpulse die Entfernung von Objekten messen, indem die Flugzeit des Lichts geschätzt wird. Dabei kommt es nicht nur zu direkten Reflexionen, sondern auch zu indirekten Reflexionen an Oberflächen. Daher ist es wichtig für die Simulation eben diese indirekte Beleuchtung zu berücksichtigen, da sich diese auf die geschätzten Distanzen im Tiefenbild auswirkt. In Abbildung 1.1 wird veranschaulicht, welche Bilder das Microsoft Kinect v2 Kamerasystem aufnimmt. Dabei handelt es sich um ein RGB-D Kamerasystem, das sowohl das Abgreifen von Farbbildern, als auch das von Tiefenbildern ermöglicht.

Aus der Literatur bekannte Simulationen von Time-of-Flight Sensoren lassen sich in zwei Kategorien einteilen: die interaktiven Simulationen, die es dem Nutzer ermöglichen Eingaben zu tätigen, die direkte Auswirkungen auf die Szene haben und durch die so beispielsweise Anpassungen an der Ausrichtung der Kamera durchgeführt werden können und die nicht-interaktiven Simulationen, in denen die Einstellungen im Vorfeld vorgenommen werden und bei denen die Berechnung einen Zeitraum von einigen Minuten in Anspruch

nimmt. Um als interaktiv bezeichnet werden zu können, sollte eine Anwendung das Bild innerhalb von 50 ms generiert haben oder in dem Zeitraum ein Zwischenergebnis liefern können, das dem Nutzer ein Feedback über das mögliche Endresultat gibt [CCD06]. Die Time-of-Flight Kameras mit der höchsten Anzahl an Bildern pro Sekunde, die in dieser Arbeit genutzt werden, liefern die Bilder innerhalb von 33 ms, weshalb diese Zeitspanne als Grenze für eine Echtzeitsimulation betrachtet wird.

1.2 Motivation und Herausforderungen

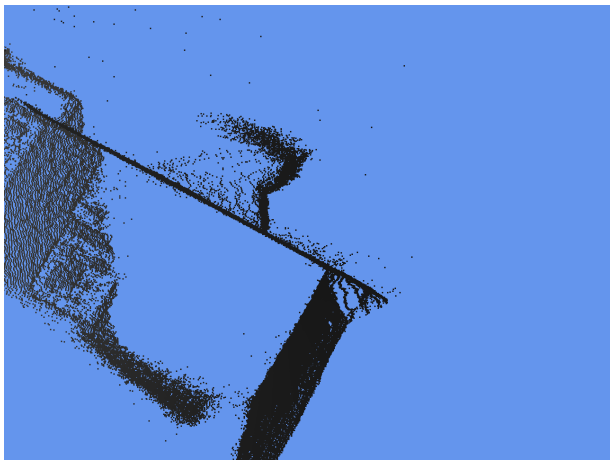


Abbildung 1.2: Die Punktewolke, die aus dem vorher gezeigten Tiefenbild erstellt wurde, auf dem der Einfluss durch die indirekte Beleuchtung auf den Kubus zu sehen ist.

Wie bereits erwähnt sind die Tiefenbilder von Time-of-Flight Kameras Effekten durch indirekte Beleuchtung ausgesetzt, was in Abbildung 1.2 veranschaulicht wird. Diese zeigt dabei dieselbe Szene wie Abbildung 1.1 aus einer anderen Perspektive, von der aus die Verzerrungen am Kubus zu erkennen sind, die durch die indirekte Beleuchtung der Oberfläche verursacht werden. Dies führt besonders in jenen Anwendungen zu Problemen, die diese Tiefenwerte zur Weiterverarbeitung verwenden, um Informationen über die aufgenommene Szene zu erhalten. Da diese Fehler stark Blickwinkelabhängig sind, führen sie dazu, dass Daten fehlerhaft ausgewertet werden und z. B. autonome Maschinen falsch auf den gelieferten Input reagieren. Während der Entwicklung von autonomen Systemen und in der Robotik wird häufig in simulierten Umgebungen gearbeitet, da fehlerhafte Implementierungen die Zerstörung von Geräten wie Drohnen oder autonomen Fahrzeugen zur Folge haben könnten, weshalb eine akkurate Simulation der genutzten Time-of-Flight Kameras wichtig ist [MNK13]. Da Time-of-Flight Kameras beispielsweise auch in der Mensch-Maschine Interaktion verwendet werden, könnte eine Reaktion auf einen fehlerhaften Input folgenschwere Konsequenzen nach sich ziehen, falls diese Artefakte nicht bereits während der Entwicklung berücksichtigt wurden. Einige wissenschaftliche Arbeiten beschäftigen sich außerdem mit der Korrektur dieser Artefakte, die durch Time-of-Flight Kamerasystemen verursacht werden [SHWH18][CCMDH07][CCMDH09][FSK+14]. Die Simulation bietet auch hier Vorteile, da Referenzwerte zur Verfügung stehen, die beispielswiese zum Trainieren von neuronalen Netzen verwendet werden können [SHWH18]. Zusätzlich bietet eine Simulation im Gegensatz zu einer echten Aufnahme die Möglichkeit

Artefakte isoliert zu betrachten und andere Störfaktoren zu entfernen, während bei echten Aufnahmen entweder keine oder ungenaue Referenzwerte zur Verfügung stehen und Artefakte nicht ausgeblendet werden können [NML+13].

Bei der Simulation von Time-of-Flight Kameras wird man vor Herausforderungen gestellt, die bisweilen nicht ohne zeitintensive Berechnungen bewältigt werden können. So erfordert die korrekte Simulation der Lichtausbreitung des Infrarotimpulses die Berechnung mehrerer Reflexionen an Oberflächen, bevor das Licht zum Sensor gelangt. Außerdem führen Reflexionen innerhalb der Linse und Beugung an der Blende zu weiteren Artefakten, die sich auf die Korrektheit der Tiefenwerte auswirkt, die in einer Simulation berücksichtig werden müssen. Darüber hinaus führt die Bewegung der Kamera und die Bewegung von Objekten im Bild während der Aufnahme zu Motion Blur Artefakten, die sich je nach Chip-Architektur des Sensors anders gestalten können. Mit diesem Problem haben sich bereits Forschungsarbeiten beschäftigt [LHK15].

Durch die zeitintensive Berechnung in den eben genannten Fällen greifen themenverwandte Untersuchungen entweder auf vorberechnete Lichtsimulationen zurück, die ausschließlich den Einsatz in statischen Szenen erlauben, oder sie verzichten auf die Simulation der Artefakte und konzentrieren sich auf einzelne Teilaspekte, wie die Simulation der Bewegungsunschärfe, während der Einfluss indirekter Beleuchtung ignoriert wird [Kel15][LHK15].

1.3 Ziele der Arbeit

Diese Arbeit verfolgt das Ziel einer phyikalisch plausiblen interaktiven Simulation von Time-of-Flight Sensoren unter Berücksichtigung möglichst vieler Artefakte. Zu diesem Zweck werden Algorithmen eingesetzt, die auf der Grafikkarte parallelisiert werden können.

Da es sich hierbei um eine zeitlich begrenzte Arbeit handelt, unterliegt diese einigen Limitationen, weshalb nicht alle Effekte simuliert werden können. Beispielsweise wird auf die Simulation der Bewegungsunschärfe verzichtet, da diese eine umfangreiche Untersuchung voraussetzt, was den Rahmen der Arbeit sprengen würde. Allerdings wird die Korrektheit der Berechnung der Reflexionen an Oberflächen priorisiert, sodass die Reduktion der Berechnungszeit nur zweitrangige Priorität hat. Dabei wird vollständig auf Vorberechnungen innerhalb der Simulation verzichtet und auf die dynamische Simulation der Szene Wert gelegt, damit auf jede mögliche Konfigurations- und Positionsänderung reagiert werden kann. Die Simulation soll dabei größtenteils auf der GPU stattfinden um eine serielle Abarbeitung auf der CPU zu vermeiden, die auf der GPU parallel durchgeführt werden könnte.

Außerdem soll die Simulation auch unabhängig von der gewählten Szene funktionieren und sowohl für enge Räume als auch für große Hallen und offene Gelände unter dem Einfluss der Sonneneinstrahlung oder künstlicher Beleuchtung ensprechend plausible Tiefenwerte liefern können.

1.4 Grober Umriss der Arbeit

In den folgenden Kapiteln werden zunächst theoretische Grundlagen besprochen, die den Untersuchungsgegenstand der Arbeit nachvollziehbar machen. Im darauf folgenden Kapitel 4 werden die Time-of-Flight Kameras, die zur Referenz genutzt werden, im Detail analysiert. Aus der Literatur bekannte Artefakte werden reproduziert und isoliert betrachtet.

Anschließend wird in Kapitel 5 die Implementierung der Simulation erläutert und die Ergebnisse in Kapitel 6 evaluiert und mit der Kinect v2 verglichen. Kapitel 7 fasst die Arbeit abschließend zusammen und gibt einen Ausblick auf mögliche zukünftige Arbeiten, die auf den Ergebnissen dieser Arbeit aufbauen könnten.

2 Grundlagen

Dieses Kapitel befasst sich mit den zugrundeliegenden Prinzipien, die dafür benötigt werden, die folgenden Kapitel der Arbeit zu verstehen.

2.1 Physikalisch basiertes Rendering

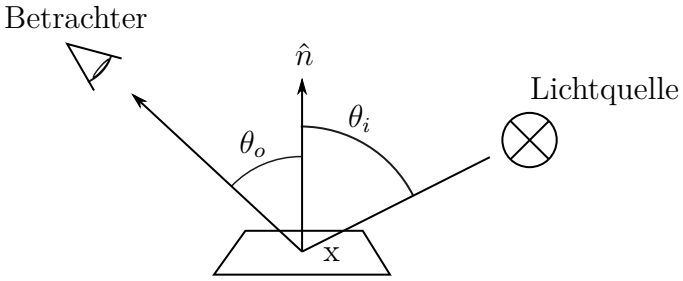


Abbildung 2.1: Licht wird zu einem Anteil an einer Oberfläche reflektiert und erreicht den Betrachter.

Menschen nehmen Oberflächen als sichtbar wahr, wenn sie von einer Lichtquelle bestrahlt werden. Dieses Licht wird an der Oberfläche reflektiert und im Auge von Zapfen oder Stäbchen absorbiert (siehe Abbildung 2.1). Licht ist dabei eine Form von elektromagnetischer Strahlung, die sowohl Welleneigenschaften als auch Teilcheneigenschaften besitzt. Physikalisch basiertes Rendering hat das Ziel das Verhalten von Licht unter Berücksichtigung physikalsicher Eigenschaften der Oberflächen möglichst physikalisch plausibel zu simulieren.

Das folgende Unterkapitel erläutert einige Grundlagen des physikalisch basierten Renderings und bezieht sich dabei größtenteils auf die Arbeit von Pharr et al. [PHJ16] und konzentriert sich auf Themen, die für den Rest der Arbeit wichtig sind. Dafür werden zunächst die Strahlungsquellen erläutert und anschließend die verwendeten Modelle zur Berechnung der Reflexionseigenschaften von Oberflächen beschrieben.

2.1.1 Radiometrische Größen

Zunächst werden die wichtigsten physikalischen Größen und ihre Relation zueinander erläutert. Dabei ist anzumerken, dass es für jede Größe sowohl radiometrische als auch photometrische Bezeichnungen gibt. In der Radiometrie werden elektromagnetische Strahlungen gemessen, während die Photometrie ihren Fokus auf die Betrachtung des sichtbaren Lichts legt. Im Rahmen dieser Arbeit werden die radiometrischen Bezeichnungen verwendet.

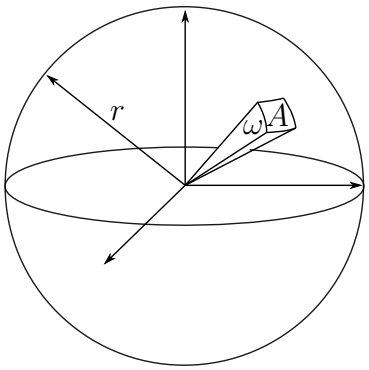


Abbildung 2.2: Der Steradiant ist eine Maßeinheit für den Raumwinkel. Auf einer Kugel mit einem Radius von r umschließt ein Steradiant eine Fläche von 1 r^2.

Einige der Größen werden über einen *Raumwinkel* oder *solid angle* ω, dem 3D Äquivalent zum Bogenwinkel im 2D Raum, gemessen. Der Raumwinkel beschreibt den Flächeninhalt A im Verhältnis zum Quadrat des Radius der Kugel und wird in *Steradiant sr* angegeben:

$$\omega = \frac{A}{r^2}[sr]. \tag{2.1}$$

Der volle Raumwinkel entspricht der Oberfläche der gesamten Einheitskugel, also $4\pi sr$. Wenn über eingehende oder ausgehende Strahlung auf einer Fläche gesprochen wird, dann wird die Strahlung in Relation zur Fläche auf der Sphäre gemessen, die von der Strahlung durchstoßen wird. Da eine einzige Richtung eine unendlich kleine Fläche auf der Sphäre durchstößt, verwendet man den *differentialen Raumwinkel* oder *differential solid angle* $\mathrm{d}\omega$, um einen infestimalen Bereich von Richtungen zu beschreiben. Wie auch in anderen Arbeiten wird das Symbol ω für die normalisierte Richtung und $\mathrm{d}\omega$ für den differentialen Raumwinkel verwendet, wenn er über eine Kugeloberfläche integriert wird [Wyn00]. Abschließend sei angemerkt, dass alle Richtungen ω in normalisierten sphärischen Koordinaten angegeben werden können. So lässt sich eine Richtung ω auch als Kombination von Polarwinkel ϕ und Azimutwinkel θ beschreiben.

Im Falle eines Time-of-Flight Sensors wird Infrarotstrahlung mittels LED's in einem bestimmten Wellenlängenspektrum erzeugt. Die *Strahlungsmenge* oder *Radiant Energy* Q bezeichnet den gesamten Energieverlust, den die Quelle durch die Strahlung erleidet. Eine Strahlungsquelle emittiert permanent Photonen, die jeweils eine Energie von

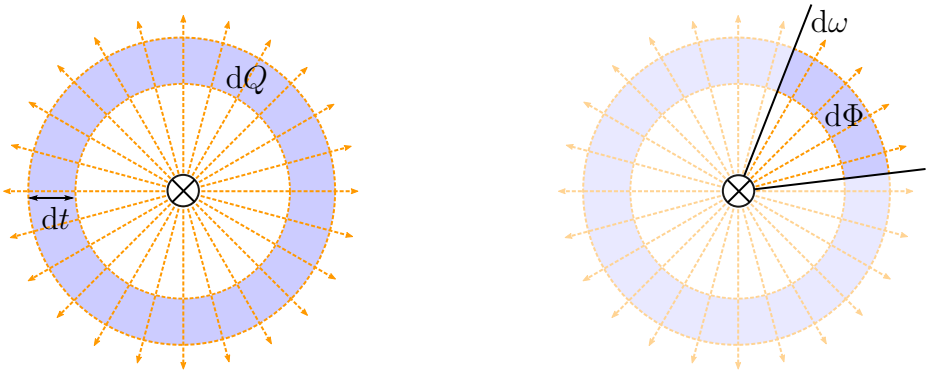
$$E_p = \frac{h \cdot c}{\lambda} \tag{2.2}$$

enthalten. Die Strahlungsmenge ist entsprechend die Summe der Energie aller Photonen, die durch den Emitter ausgestrahlt werden. h ist hierbei die Planck Konstante, c die Lichtgeschwindigkeit und λ die Wellenlänge des emittierten Photons.

Der *Strahlungsfluss* oder *Radiant Flux* Φ bezeichnet die Strahlungsmenge, die pro Zeit von einer Punktquelle emittiert wird

$$\Phi = \frac{\mathrm{d}Q}{\mathrm{d}t} \tag{2.3}$$

und in $Watt$ angegeben wird (siehe Abbildung 2.3a).



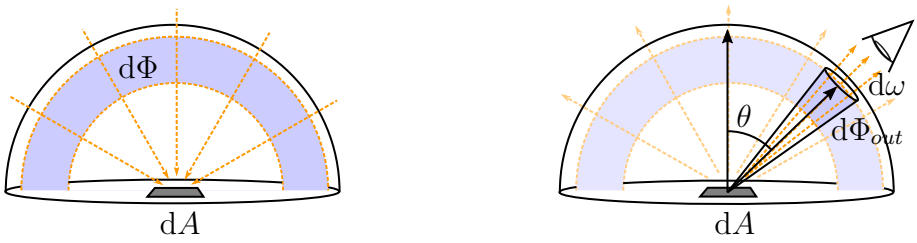
(a) Strahlungsfluß Φ **(b)** Strahlungsintensität I

Abbildung 2.3: Schematische Darstellung des Strahlungsflusses Φ und der Strahlungsintensität I.

Die *Strahlungsintensität* oder *Radiant Intensity* I gibt an, wie viel Strahlung pro Raumwinkel vom Emitter ausgeht (siehe Abbildung 2.3b)

$$I = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega}. \tag{2.4}$$

Das ist besonders nützlich, um zu beschreiben, in welche Richtung die Quelle mehr oder weniger Strahlung emittiert. Dies wird in $\frac{Watt}{sr}$ angegeben.



(a) Bestrahlungsstärke E **(b)** Strahlungsdichte L

Abbildung 2.4: Schematische Darstellung des Bestrahlungsstärke E und der Strahlungsdichte L.

Die *Bestrahlungsstärke* oder *Irradiance* E gibt an, wie viel Strahlung beim Empfänger pro infestimaler Fläche $\mathrm{d}A$ ankommt

$$E = \frac{\mathrm{d}\Phi_{in}}{\mathrm{d}A} \tag{2.5}$$

(siehe Abbildung 2.4a). Dem gegenüber steht die *Radiosität* oder *Radiosity* B, die mit derselben Formel aussagt, wie viel Strahlung einen Emitter pro Fläche $\mathrm{d}A$ verlässt:

$$B = \frac{\mathrm{d}\Phi_{out}}{\mathrm{d}A}. \tag{2.6}$$

Beide Größen werden in $\frac{Watt}{m^2}$ angegeben (siehe Abbildung 2.4b).

Die letzte und wichtigste radiometrische Größe ist die *Strahlungsdichte* oder *Radiance* L. Die Strahlungsdichte gibt an, wie viel Strahlung von einem gegebenen Punkt

der Strahlungsquelle in eine Richtung ausgesendet wird. Sie wird definiert als der versendete Strahlungsfluss $\mathrm{d}\Phi$ pro sichtbarer Empfängerfläche $\mathrm{d}A\cos\theta$ und Raumwinkel $\mathrm{d}\omega$

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}A \cdot \cos\theta \cdot \mathrm{d}\omega} \tag{2.7}$$

und in $\frac{Watt}{m^2 sr}$ angegeben.

2.1.2 Definition der verwendeten Lichtquellen

Bei einem *Punktlicht* handelt es sich um eine Strahlungsquelle ohne räumliche Ausdehnung. Die Quelle ist also unendlich klein, hat keine Oberfläche und emittiert in alle Richtungen gleichmäßig viel Strahlung. Die Bestrahlungsstärke nimmt quadratisch mit dem Abstand zur Strahlungsquelle ab. Außerdem ist die Bestrahlungsstärke zusätzlich vom Winkel θ zwischen der Normale der bestrahlten Oberfläche und der Bestrahlungsrichtung abhängig, da die ‚gesehene' Fläche mit einem größeren Winkel zwischen Flächennormale und Strahlungsrichtung an Ausdehnung abnimmt. Somit lässt sich die Bestrahlungsstärke durch ein Punktlicht entsprechend des photometrischen Entfernungsgesetzes folgendermaßen bestimmen:

$$E = \frac{I \cdot \cos\theta}{d^2}. \tag{2.8}$$

Darüber hinaus werden im Rahmen dieser Arbeit außerdem *Lambert Emitter* als Strahlungsquellen verwendet. Bei Lambert Emittern handelt es sich um eine Strahlungsquelle, die von jedem Blickwinkel aus gleich hell erscheint. Dies resultiert in einer cosinusgewichteten Intensitätsverteilung, was darauf zurückzuführen ist, dass sich bei einem größeren Betrachtungswinkel θ ebenfalls die ‚gesehene' Fläche verringert und die Strahlungsdichte deshalb konstant bleibt:

$$I(\theta) = \cos\theta \cdot I_0. \tag{2.9}$$

2.1.3 Bidirektionale Reflexionsverteilungsfunktion

Eine *bidirektionale Reflexionsverteilungsfunktion* oder *Bidirectional Reflectance Distribution Function* definiert die Reflexionseigenschaften von Oberflächen unterschiedlicher Materialien. Mit Hilfe der oben beschriebenen Größen ist es möglich zu sagen, dass eine bidirektionale Reflexionsverteilungsfunktion ein Material beschreibt, indem es das Verhältnis von eingehender Bestrahlungsstärke $\mathrm{d}E$ aus Richtung ω_i zur ausgehenden Strahlungsdichte $\mathrm{d}L$ in eine Richtung ω_o an der Oberfläche beschreibt

$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) = \frac{\mathrm{d}L_o(\mathrm{x}, \omega_o, \lambda)}{\mathrm{d}E(\mathrm{x}, \omega_i, \lambda)} = \frac{\mathrm{d}L_o(\mathrm{x}, \omega_o, \lambda)}{\mathrm{d}L_i(\mathrm{x}, \omega_i, \lambda) \cdot \cos\theta_i \cdot \mathrm{d}\omega_i}. \tag{2.10}$$

Bidirektionale Reflexionsverteilungsfunktionen, im Folgenden mit BRDF abgekürzt, sind wellenlängenabhängig und beschreiben für jede Wellenlänge ein anderes Verhältnis. Im Rahmen dieser Arbeit wird die BRDF allerdings nicht für das gesamte Spektrum definiert, sondern es wird nur mit RGB Werten gearbeitet und die Funktion wird nur für die Farben Rot, Grün und Blau und ausgewählte Infrarotwellenlängen ausgewertet.

Theoretisch lässt sich mit einer BRDF jedes beliebige Verhalten eines Materials beschreiben. Da in der vorliegenden Untersuchung auf ein physikalisch plausibles Verhalten der Oberflächen Wert gelegt wird, unterliegen die BRDF einigen Einschränkungen. Man spricht von einer physikalisch plausiblen BRDF, wenn sie die folgenden Eigenschaften erfüllt:

1. Sie ist nicht negativ:
$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) \geq 0. \tag{2.11}$$

2. Sie erfüllt den Energieerhaltungssatz. Das bedeutet, dass die gesamte reflektierte Strahlung der Oberfläche nicht die Bestrahlungsstärke, die auf die Oberfläche einwirkt, überschreiten darf [Wyn00]:
$$\forall \omega_i : \int_{2\pi\, sr} f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) \cos\theta_o \, \mathrm{d}\omega \leq 1. \tag{2.12}$$

3. Sie erfüllt die Helmholtz Reziprozität, die besagt, dass der Strahlungsfluss sowohl vorwärts als auch rückwärts identische Ergebnisse liefert. Das bedeutet, dass die BRDF unverändert bleibt, auch wenn die Strahlungsquelle und der Betrachter vertauscht werden:
$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) = f_r(\mathrm{x}, \omega_o, \omega_i, \lambda). \tag{2.13}$$

2.1.4 BRDF Modelle

Es gibt grundsätzlich zwei Möglichkeiten, um eine BRDF zu repräsentieren. Zum einen besteht die Möglichkeit des Einmessens einer Probe mittels Gonioreflektometer und Speicherns der Messergebnisse in einer Matrix in Abhängigkeit zum Einfalls- und Ausfallswinkel der Strahlung. Diese eingemessenen BRDFs sind in Datenbanken erfasst und können zum Rendern von Objekten verwendet werden, z. B. [MPBM03]. Zum anderen können BRDFs mittels analytischer Funktionen approximiert werden. Letztere Vorgehensweise ist die gängigste, da das Einmessen zu einem nicht unerheblichen Datenvolumen führt.

Bei der Lambert BRDF handelt es sich um eine der einfachsten und grundlegendsten Beschreibungen einer Oberflächenreflexion, die ausschließlich diffuse Reflexionen beschreibt. Sie wurde von Lambert [Lam60] vor mehr als 200 Jahren entwickelt und ist das meist genutzte BRDF Modell in der Computergrafik. Dabei wird die Oberfläche durch Bestrahlung selbst zu einem Lambert Emitter, erscheint von allen betrachteten Richtungen aus gleich hell und wird *Lambertscher Reflektor* genannt:
$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) = \frac{\rho(\lambda)_d}{\pi}. \tag{2.14}$$

Bei $\rho(\lambda)_d$ handelt es sich um den Reflexionsgrad, der für jede Wellenlänge jeweils einen Wert zwischen 0 und 1 annimmt und als konstantes Verhältnis zwischen dem gesamten eingehenden und dem gesamten ausgehenden Strahlungsfluss für diese Wellenlänge definiert wird
$$\rho(\lambda)_d = \frac{\Phi_{out}}{\Phi_{in}} = \frac{B}{E}. \tag{2.15}$$

BRDFs können als gewichtete Summe kombiniert werden. Daher lassen sie sich in Komponenten aufteilen, mit denen jeder Aspekt einzeln beschrieben werden kann [TS67]. Eine Lambert BRDF wird daher häufig mit anderen BRDFs kombiniert, die den spiegelnden Anteil beschreiben. Dabei ist darauf zu achten, dass die Kombination der BRDFs die oben genannten Kriterien zur physikalischen Plausibilität erfüllt. So bestehen die BRDFs, die in dieser Arbeit verwendet werden, aus einer diffusen f_d und einer spekularen Komponente f_s, deren gewichtete Summe die resultierende BRDF ergibt:

$$f_r = sf_s + kf_d \mid s + k = 1. \tag{2.16}$$

Ein Beispiel für eine Definition eines Glanzlichtanteils wäre die BRDF für eine perfekt spiegelnde Oberfläche, die mittels der Dirac-Delta-Funktion beschrieben wird und überall, mit Ausnahme des reflektierten Strahls, den Wert 0 annimmt,

$$f_r(\omega_i, \omega_o) = f_r(\theta_i, \phi_i, \theta_o, \phi_o) = \frac{\delta(\theta_i - \theta_o) \cdot \delta(\phi_i + \pi - \phi_o)}{\sin\theta_i \cos\theta_i} \tag{2.17}$$

$$\delta(\mathrm{x}) = \begin{cases} \infty, x = 0 \\ 0, x \neq 0 \end{cases}$$

und folgende Bedingungen erfüllt:

$$\int_{-\infty}^{\infty} \delta(\mathrm{x})\mathrm{dx} = 1. \tag{2.18}$$

Das Verhältnis zwischen diffuser und spiegelnder Reflexion kann mittels des Fresnel Reflexionsgrades $F(\theta)$ bestimmt werden. Der Fresnel Reflexionsgrad wird häufig mit transparenten Oberflächen in Verbindung gebracht, um den Anteil zwischen Reflexion und Transmission zu bestimmen. Dies ist ebenfalls auf opake Materialien anwendbar, da sich wie im Falle von Plastik Pigmente im Material befinden, die den Transmissionsanteil wiederum diffus reflektieren [CT81]. Das Verhältnis von Reflexion und Transmission ist abhängig von dem Brechungsindex des jeweiligen Mediums und dem Einfallswinkel der Strahlung. Außerdem beeinflusst die Polarisation der Strahlung dessen Reflexionsverhalten, weshalb die Berechnung der Fresnel Reflexion mit einem parallelen und einem orthogonalen Anteil vorgenommen wird.

Der Reflexionsgrad besteht aus dem parallelen Anteil

$$r_\parallel = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2} \tag{2.19}$$

und dem orthogonalen Anteil,

$$r_\perp = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2}, \tag{2.20}$$

während dessen Summe den Reflexionsgrad für unpolarisierte Strahlung ergibt

$$F_r(\theta_i) = \frac{1}{2}(r_\parallel^2 + r_\perp^2). \tag{2.21}$$

Hierbei ist η_1 die optische Dichte des Mediums, in der sich die Strahlung vor der Transmission befindet, während η_2 die optische Dichte des Mediums angibt, in die die Strahlung eindringt. Während dielektrische Materialien die Strahlung entsprechend des Fresnel Reflexionsgrades entweder diffus oder vollständig spiegelnd reflektieren, weisen Konduktoren ein besonderes Reflexionsverhalten auf, da sie zum einen keinerlei diffuse Reflexionseigenschaften haben und zum anderen ein Teil der Strahlung, abhängig von der Wellenlänge, absorbiert wird. Der Reflexionsgrad für Konduktoren wird aus dem parallelen Anteil

$$r_{\parallel}^2 = \frac{(\eta^2 + k^2)\cos^2\theta_i - 2\eta\cos\theta_i + 1}{(\eta^2 + k^2)\cos^2\theta_i + 2\eta\cos\theta_i + 1} \tag{2.22}$$

und einem orthogonalen Anteil

$$r_{\perp}^2 = \frac{(\eta^2 + k^2) - 2\eta\cos\theta_i + \cos^2\theta_i}{(\eta^2 + k^2) + 2\eta\cos\theta_i + \cos^2\theta_i} \tag{2.23}$$

berechnet. Hierbei ist η der Brechungsindex des Materials, während es sich bei k um den Absorptionskoeffizienten handelt. Der Reflexionsgrad unpolarisierter Strahlung lässt sich ebenfalls mittels Gleichung 2.21 bestimmen.

Die oben beschriebene BRDF ist geeignet, um glatte Oberflächen zu approximieren. Das Ergebnis weicht allerdings stark von den Reflexionsverhalten rauer Oberflächen ab, weshalb im Rahmen dieser Arbeit sowohl für den diffusen als auch für den spekularen Anteil *Mikrofacetten Modelle* zur Simulation rauer Oberflächenstrukturen verwendet werden. Bei Mikrofacetten BRDFs handelt es sich um Beleuchtungsmodelle, die auf physikalisch basierten Modellen aufbauen. Dabei wird die Oberfläche durch viele kleinere zufällig orientierte Facetten modelliert, was die Simulation der Rauheit und der Abschattung ermöglicht.

Cook-Torrance BRDF/Torrance-Sparrow BRDF

Torrance und Sparrow [TS67] präsentierten 1967 erstmals ein Model zur Berechnung des spiegelnden Anteils rauer Oberflächen, woraufhin Cook und Torrance [CT81] 1981 darauf aufbauend ein allgemeines lokales Beleuchtungsmodell entwickelten

$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) = \frac{F_r(\omega_h)D(\omega_h)G(\omega_o, \omega_i)}{4\cos\theta_i\cos\theta_o}. \tag{2.24}$$

Es sei anzumerken, dass es sich bei Gleichung 2.24 um die Variante von Pharr et al. [PHJ16] bzw. die von Torrance und Sparrow [TS67] handelt, da die ursprüngliche Gleichung der Arbeit von Cook und Torrance gegen den Energieerhaltungssatz verstößt. Die Gleichung 2.24 besteht aus dem in Unterabschnitt 2.1.4 beschriebenen Fresnelterm F_r, einer *Dichtefunktion* D und einem *geometrischen Abschwächungsfaktor* G, die im Folgenden im Detail erläutert werden.

Das Mikrofacetten Modell von Torrance und Sparrow geht davon aus, dass die Oberfläche von beispielsweise rauen Metallen oder Plastik aus einer Anzahl vieler kleiner spiegelnder Facetten besteht, dessen Reflexionen jeweils mittels Gleichung 2.17 berechnet werden können. Die Abbildung 2.5 stellt schematisch eine spiegelnde Reflexion an einer rauen

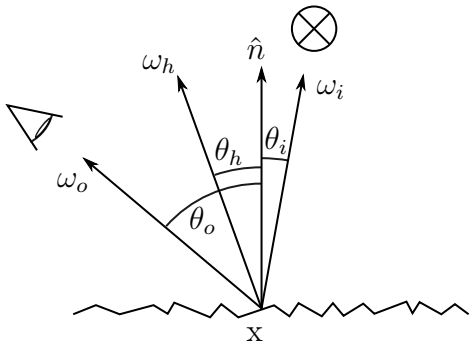


Abbildung 2.5: Die Oberfläche einer Mikrofacetten BRDF wird als eine Sammlung von vielen kleinen Facetten dargestellt.

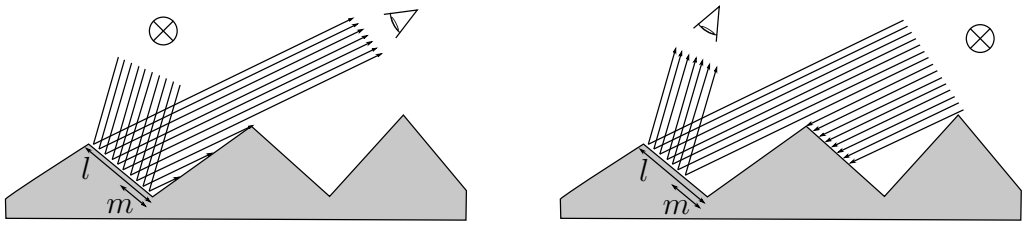
Oberfläche dar, die mittels Mikrofacetten modelliert wird. Für die Berechnung der direkten Reflexion der rauen Oberfläche sind also nur die Mikrofacetten interessant, dessen Normale mit dem *Halbvektor* ω_h übereinstimmt, also so orientiert ist, dass die eingehende Strahlung aus Richtung ω_i in Richtung ω_o reflektiert wird

$$\omega_h = \frac{\omega_o + \omega_i}{|\ \omega_o + \omega_i\ |}. \tag{2.25}$$

Die Dichtefunktion D gibt dabei den Anteil der Mikrofacetten an, dessen Normale in Richtung des Halbvektors ω_h zeigt und frei gewählt werden kann. Torrance und Sparrow [TS67] schlagen in ihrer Arbeit eine Gaußverteilung vor, während in der Arbeit von Cook und Torrance [CT81] eine Beckmannverteilung empfohlen wird, die auch in dieser Arbeit verwendet wird

$$D(\omega_h) = \frac{e^{\frac{-tan^2\theta_h}{\alpha^2}}}{\pi\alpha^2 \cos^4\theta_h}, \alpha = \sqrt{2}\sigma. \tag{2.26}$$

Die *Rauheit* der Oberfläche wird dabei durch die Standardabweichung σ der Orientierung der Mikrofacetten definiert [PHJ16] und wird im Bogenmaß angegeben.



(a) Maskierung **(b)** Abschattung

Abbildung 2.6: Die Selbstverschattung und die Maskierung der Oberfläche, die durch Nachbarfacetten verursacht wird und mit dem geometrischen Abschwächungsfaktor approximiert wird.

Der geometrische Abschwächungsfaktor G berücksichtigt die Maskierung (siehe Abbildung 2.6a) und Abschattung (siehe Abbildung 2.6b) einer Facette durch eine Nachbarfacette. Dafür wird angenommen, dass jede spiegelnde Facette ein symmetrisches Gegenstück hat und diese zusammen V-förmige Kerben bilden (*V-cavities*). Der Abschwä-

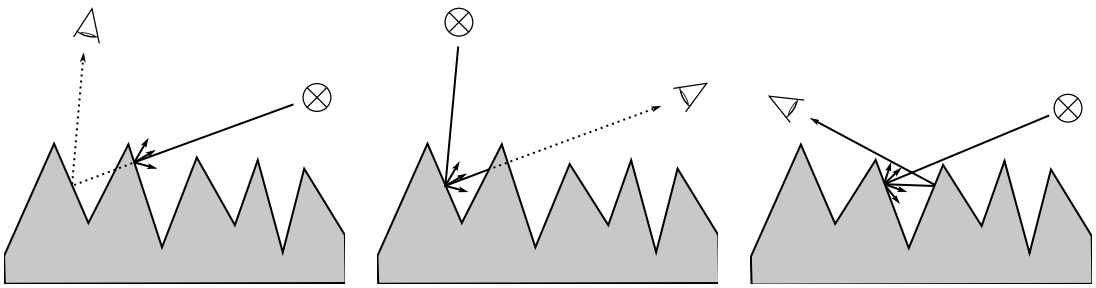
chungsfaktor wird durch das Verhältnis des Teils der Facette $l - m$, der an der Reflexion beteiligt ist, zu der gesamten Oberfläche l, definiert, wobei nur einfache spekulare Reflexionen berücksichtigt und Interreflexionen als perfekt diffus angenommen werden:

$$G(\omega_h) = \min\left[1, \underbrace{\frac{2(\hat{n}\cdot\omega_h)(\hat{n}\cdot\omega_o)}{\omega_o\cdot\omega_h}}_{Maskierung}, \underbrace{\frac{2(\hat{n}\cdot\omega_h)(\hat{n}\cdot\omega_i)}{\omega_o\cdot\omega_h}}_{Abschattung}\right]. \tag{2.27}$$

Das Cook-Torrance Model ist geeignet um Metalle und nichtmetallische Oberflächen ohne diffusen Anteil wie schwarzes Plastik, zu simulieren und die Ergebnisse der Simulation weisen eine starke Übereinstimmung mit realen Messungen auf [TS67].

Oren-Nayar BRDF

1994 stellten Oren und Nayar [ON94] ein lokales Beleuchtungsmodell zur Simulation von rauen diffusen Oberflächen vor, das auf dem Mikrofacetten Modell von Torrance und Sparrow basiert. Oren und Nayar beobachteten durch Experimente, dass sich die wahrgenommene Helligkeit einer rauen Oberfläche abhängig von der Blickrichtung ω_o erhöht, je mehr sich diese der Beleuchtungsrichtung ω_i annähert, und damit vom Lambert Modell abweicht, welches Oberflächen unabhängig vom betrachteten Blickwinkel gleich hell erscheinen lässt.



(a) Maskierung **(b)** Abschattung **(c)** Interreflexion

Abbildung 2.7: Das Mikrofacetten Modell nach Oren und Nayar [ON94], dessen Blickrichtungsabhängigkeit durch Selbstverschattung, Maskierung und Interreflexionen verursacht wird.

Analog zum Mikrofacetten Modell von Torrance und Sparrow basiert das Modell von Oren und Nayar auf vielen kleinen Lambert Facetten und simuliert damit ebenfalls Abschattung und Maskierung durch Nachbarfacetten, erweitert das Modell aber zusätzlich um Interreflexionen zwischen den V-Kerben (siehe Abbildung 2.7). Im Gegensatz zu der Cook-Torrance BRDF ist hier die Dichtefunktion nicht frei wählbar und es wird von einer Gaußverteilung der Facettenoberflächen ausgegangen. Da die Auswertung der Verteilung das Integrieren über die Hemisphäre erfordern würde, wurde eine funktionale Approximation angestrebt, die die Eigenschaften

$$\begin{aligned} f_{direct}(\mathrm{x}, \omega_i, \omega_o, \lambda) = &\frac{\rho(\lambda)_d}{\pi}\Big[C_1(\sigma) + \cos(\phi_o - \phi_i)C_2(\alpha, \beta, \phi_o - \phi_i, \sigma)\tan\beta + \\ &\Big(1 - \mid \cos(\phi_o - \phi_i) \mid\Big)C_3(\alpha, \beta, \sigma)\tan\left(\frac{\alpha+\beta}{2}\right)\Big] \end{aligned} \tag{2.28}$$

erfüllt und die Abschattung und Maskierung berechnet, während

$$f_{interreflection}(\mathrm{x}, \omega_i, \omega_o, \lambda) = 0.17\frac{\rho(\lambda)_d^2}{\pi}\frac{\sigma^2}{\sigma^2 + 0.13}\left[1 - \cos(\phi_o - \phi_i)\left(\frac{2\beta}{\pi}\right)^2\right] \tag{2.29}$$

mit den Koeffizienten

$$C_1 = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33}$$

$$C_2 = \begin{cases} 0.45\frac{\sigma^2}{\sigma^2 + 0.09}\sin\alpha & \text{wenn } \cos(\phi_o - \phi_i) \geq 0 \\ 0.45\frac{\sigma^2}{\sigma^2 + 0.09}\left(\sin\alpha - \left(\frac{2\beta}{\pi}\right)^3\right) & \text{sonst} \end{cases}$$

$$C_3 = 0.125\left(\frac{\sigma^2}{\sigma^2 + 0.09}\right)\left(\frac{4\alpha\beta}{\pi^2}\right)^2$$

$$\alpha = \max(\theta_i, \theta_o)$$

$$\beta = \min(\theta_i, \theta_o)$$

die Interreflexionen berechnet. Die BRDF wird mit der Summe beider Terme berechnet:

$$f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) = f_{direct}(\mathrm{x}, \omega_i, \omega_o, \lambda) + f_{interreflection}(\mathrm{x}, \omega_i, \omega_o, \lambda). \tag{2.30}$$

Es sei zu beachten, dass für glatte Oberflächen die Koeffizienten C_2, C_3 und der Interreflexionsanteil einen Wert von 0 annehmen, während C_1 einen Wert von 1 annimmt, was dazu führt, dass die BRDF auf das Lambert Modell reduziert wird.

Die Approximation von Oren und Nayar zeigt, wie auch das Modell von Cook und Torrance, eine hohe Übereinstimmung mit den gemessenen Daten für raue Oberflächen wie Sandpapier, Stoff und Gips. Zusammen mit dem in Abschnitt 2.1.4 vorgestellten lokalen Beleuchtungsmodell für spiegelnde Oberflächen bilden die vorgestellten Mikrofacetten Modelle jeweils eine der Komponenten der Gleichung 2.16 für eine gemeinsame BRDF.

2.1.5 Rendergleichung

Kajiya stellte 1986 [Kaj86] die oft zitierte *Rendergleichung* oder auch *Rendering Equation* vor, welche in einer allgemeinen Form die Strahlung an einem Punkt x in eine Richtung ω_o beschreibt. Statt der ursprünglichen Formel aus seiner Arbeit wird hier die Rendergleichung in einer äquivalenten Darstellungsform verwendet, da sie den Anwendungsfall in dieser Arbeit anschaulicher beschreibt

$$L_o(\mathrm{x}, \omega_o, \lambda) = L_e(\mathrm{x}, \omega_o, \lambda) + \int_{2\pi\, sr} f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) \cdot L_i(\mathrm{x}, \omega_i, \lambda) \cdot \cos\theta_i \, \mathrm{d}\omega_i. \tag{2.31}$$

Der Emissionsterm L_e beschreibt die Eigenemission der Oberfläche an Punkt x in Richtung ω_o, falls es sich bei der Oberfläche um eine Strahlungsquelle handelt, während der Term L_i die eingehende Strahlung an dem Punkt x aus Richtung ω_i beschreibt. Bei dem Term f_r handelt es sich um die BRDF, wie sie in dem vorangegangenen Unterkapitel beschrieben

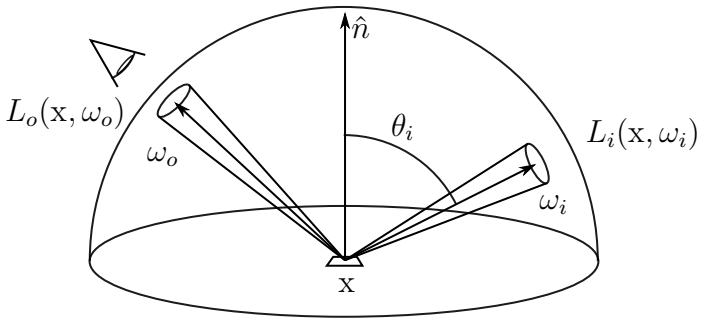


Abbildung 2.8: Schematische Darstellung der Rendergleichung. Die ausgehende Strahlung L_o in Richtung ω_o wird berechnet durch das Integrieren der eingehenden Strahlung L_i aus allen Richtungen ω_i über der Hemisphäre.

wurde. Wie Abbildung 2.8 schematisch darstellt, wird zur Berechnung der Strahlung, die in Richtung ω_o reflektiert und/oder emittiert wird, die eingehende Strahlung aus allen Richtungen ω_i über die gesamte Hemisphäre integriert. Es ist anzumerken, dass zur Berechnung des Terms L_i wiederum eine Integration über die gesamte Hemisphäre an dem Punkt notwendig ist, von dem aus die Strahlung emittiert/reflektiert wird, die an Punkt x aus Richtung ω_i auftrifft. Es handelt sich dabei also um eine Rekursion und die Beleuchtung an dem Punkt x ist von der Beleuchtung aller anderen Punkte x′ abhängig, was durch die ursprüngliche Formulierung der Rendergleichung ausgedrückt wird:

$$L_r^o(\mathrm{x}, \mathrm{x}') = g(\mathrm{x}, \mathrm{x}') \cdot \left(L_e^o(\mathrm{x}, \mathrm{x}') + \int_S b(\mathrm{x}, \mathrm{x}', \mathrm{x}'') L_i^o(\mathrm{x}', \mathrm{x}'') \mathrm{dx}' \right). \tag{2.32}$$

Hierbei gibt L_r^o an, wie viel Strahlung den Punkt x aus x′ erreicht, während es sich bei b um eine BRDF handelt, die den Anteil der Strahlung angibt, der den Punkt x′ von x″ erreicht und in Richtung x reflektiert wird. Bei g handelt es sich um den geometrischen Term, der den Strahlungsabfall durch die gegenseitige Lage der Punkte x und x′ beschreibt und dem Entfernungsgesetz entspricht. L_e^o beschreibt analog zu L_e, wie viel Strahlung, die von x′ emittiert wird, den Punkt x erreicht. S ist dabei die Vereinigung aller Oberflächen, über die integriert wird.

Im Kontrast zu lokalen Beleuchtungsmodellen, die sich auf die direkte Beleuchtung der Oberfläche durch sichtbare Strahlungsquellen konzentrieren, werden Verfahren, die versuchen die Rendergleichung zu lösen oder zu approximieren, als *globale Beleuchtungsmodelle* bezeichnet.

2.1.6 Path Tracing

In derselben Arbeit, in der Kajiya die Rendergleichung vorstellte, präsentierte er ein Verfahren, um diese mittels der *Monte Carlo Integration* zu lösen. Bei der Monte Carlo Integration handelt es sich um ein Verfahren zur näherungsweisen Lösung eines Integrals

durch zufällige Stichproben der Funktion:

$$\int_{2\pi\, sr} f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) \cdot L_i(\mathrm{x}, \omega_i, \lambda) \cdot \cos\theta_i \, \mathrm{d}\omega_i \approx \frac{1}{N} \sum_{j=1}^{n} \frac{f_r(\mathrm{x}, \omega_i, \omega_o, \lambda) \cdot L_i(\mathrm{x}, \omega_i, \lambda) \cdot \cos\theta_i}{p(\omega_i)}. \tag{2.33}$$

Die Schätzung des Integrals erhält man durch das Auswerten der Funktion mit einer Dichte von $p(\omega_i)$ und der Mittelung dieser Werte. Die Dichte $p(\omega_i)$ kann zwar beliebig gewählt werden, ähnelt allerdings im Optimalfall der auszuwertenden Funktion, denn je näher $p(\omega_i)$ der auszuwertenden Funktion ähnelt, desto schneller konvergiert das arithmetische Mittel gegen das tatsächliche Integral der Funktion, während, auch bei schlecht gewählten Dichtefunktionen, mit zunehmender Anzahl an Stichproben die Schätzung des Integrals genauer wird. Falls der Verlauf der Funktion unbekannt ist, kann daher auch eine konstante Dichte $p(\omega_i) = 1$ bei gleichzeitig hoher Anzahl an Stichproben gewählt werden.

Die Implementierung des *Path Tracing* Algorithmus simuliert den Weg der Photonen von der Strahlungsquelle zum Betrachter, indem der Pfad von dem Sensor zur Quelle zurückverfolgt wird. Statt also Strahlen zu berechnen, die ihren Ursprung in der Strahlungsquelle haben und entweder direkt oder durch Reflexionen über Oberflächen auf den Sensor treffen, werden für jeden Pixel des zu berechnenden Bildes mehrere Strahlen vom Betrachter aus in die Szene geschickt und der Weg zur Strahlungsquelle zurückverfolgt. Dafür wird bei jedem Schnittpunkt eines Strahls anhand einer Oberfläche getestet, ob diese von einer oder mehreren Strahlungsquellen bestrahlt wird, woraufhin deren anteilige Reflexion in die Richtung des Strahlenursprungs berechnet wird. Für jeden Schnittpunkt des Strahls mit einer Oberfläche wird anschließend ein neuer Strahl in eine zufällige Richtung erzeugt, dessen neuer Ursprung der berechnete Schnittpunkt ist. Dies wird rekursiv so lange wiederholt, bis eine Terminierungsbedingung, z. B. eine maximale Pfadlänge, erfüllt wird. Abhängig von der Szene, der BRDF und der gewählten Dichte $p(\omega_i)$ kann die Berechnung des Ergebnisses eine hohe Anzahl an Strahlen erfordern, was in einer langen Berechnungszeit resultiert.

Die Implementierung des Algorithmus wird im Detail in Kapitel 5 erläutert, sodass in diesem Unterkapitel nicht weiter darauf eingegangen wird.

2.2 Tiefenwerte durch Time-of-Flight

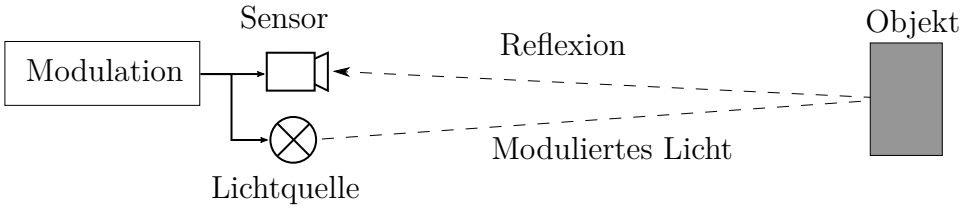


Abbildung 2.9: Grundlegendes Konzept eines Time-of-Flight Tiefensensors.

Das Prinzip der Tiefenmessung mittels Time-of-Flight Sensoren baut auf dem Messen der Zeit auf, die die Strahlung benötigt, um von einer Strahlungsquelle ausgestrahlt, an

einem Objekt reflektiert zu werden und schließlich auf einen Sensor zu treffen. Wie in Abbildung 2.9 veranschaulicht, besteht ein Time-of-Flight System im Kern aus einer Strahlungsquelle, die ein amplitudenmoduliertes Signal aussendet und einem Sensor, der das Signal erfasst, das von Objekten reflektiert wird. Typischerweise wird dabei Infrarotlicht (IR) oder Nahes Infrarotlicht (NIR) verwendet, das auf einen schmalen Frequenzbereich beschränkt ist. So sollen Effekte durch Umgebungslicht unterdrückt werden. Im einfachsten Fall besteht der Sensor aus einem einzelnen Pixel, das nur eine Tiefeninformation liefert. Die Tiefeninformation wird mit der Flugdauer Δt, die das Licht benötigt hat, um von einem Objekt reflektiert zu werden und auf den Sensor zu treffen, und der Ausbreitungsgeschwindigkeit des Lichts c berechnet. Das Produkt wird dabei halbiert, da das Licht die zweifache Strecke zurückgelegt hat

$$d = \frac{c \cdot \Delta t}{2}. \tag{2.34}$$

Die Flugdauer umfasst bei Lichtgeschwindigkeit einen Zeitraum von wenigen Nanosekunden. Für Laserscanner werden dazu Einzelphoton Lawinenphotodioden bzw. Single-Photon Avalanche Diodes (SPADs), auch bekannt als Geigermode-APD (GAPDs), verwendet. Diese können die Photonen mit einer Genauigkeit von wenigen Picosekunden detektieren, was eine Messung der Distanz mit einer Genauigkeit von wenigen Millimetern ermöglicht [GVS18].

Da das direkte Messen der Zeit hohe Anforderungen an die Präzision der Bauteile stellt, wird alternativ das modulierte Lichtsignal über einen Zeitraum integriert und die Distanz entweder direkt mittels Plusdauermodulation aus dem Verhältnis zweier Intensitäten oder mittels der Continuous-Wave Modulation durch Berechnung der Phasenverschiebung einer kontinuierlichen Wellenfunktion berechnet. Die beiden Verfahren werden jeweils in Unterabschnitt 2.2.1 und Unterabschnitt 2.2.2 im Detail erläutert.

Beiden Methoden ist gemeinsam, dass sie ein amplitudenmoduliertes Signal zur direkten Messung der Distanz oder der Phasenverschiebung verwenden, aus der sich die Distanz berechnen lässt. Bei der Amplitudenmodulation wird ein hochfrequentes Trägersignal durch ein niederfrequentes Nutzsignal verändert. Bei dem Trägersignal handelt es sich bei Time-of-Flight Kamerasystemen um das emittierte Infrarotlicht, dessen Amplitude verändert wird. Die Frequenz des Trägersignals spielt bei der Berechnung der Distanz keine Rolle und wird nur zum Übertragen des Nutzsignals verwendet.

2.2.1 Pulsdauermodulation

Time-of-Flight Kameras, die mit der Pulsdauermodulation arbeiten, senden in bestimmten Abständen Lichtpulse mit einer definierten Dauer t_0 aus und ermitteln die Zeit Δt, die das Licht benötigt, um an einer Objektoberfläche reflektiert zu werden und zum Sensor zurückzukehren. Dafür wird das Licht über eine Dauer t_0 durch eine Infrarot LED oder einen Laser emittiert und das zurückstrahlende Licht von Sensoren aufgenommen. Abhängig von der Distanz trifft das Licht mit einer Verzögerung Δt am Sensor auf.

Li [Li14] präsentiert eine Lösung zur Messung der Verzögerung, indem pro Pixel mehrere sogenannte Buckets das reflektierende Licht zeitversetzt über einen Zeitraum von t_0

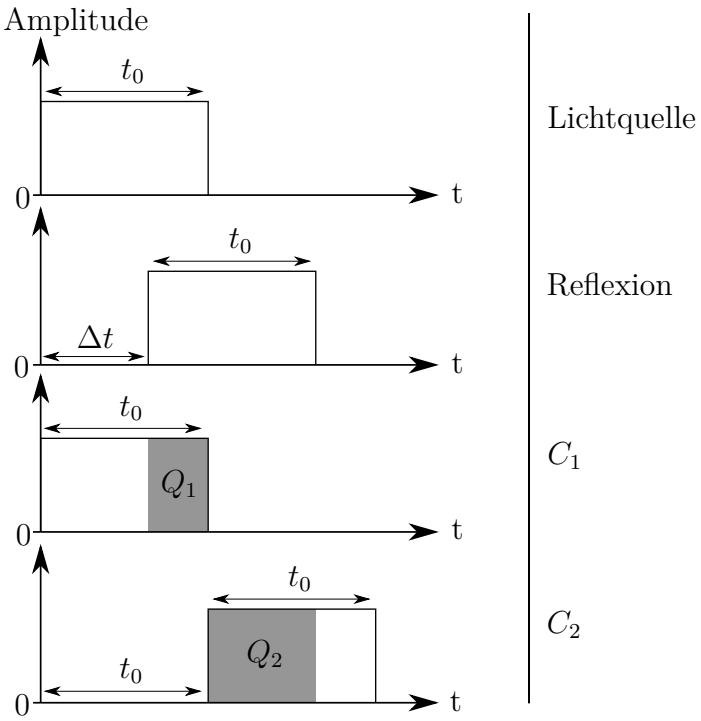


Abbildung 2.10: Bestimmung der Distanz eines reflektieren Lichtpulses mit Hilfe von zwei Buckets C_1 und C_2.

integrieren und der Zeitversatz Δt mit Hilfe des Verhältnisses der Intensitäten approximiert wird. In der einfachsten Form enthält das System zwei Buckets C_1 und C_2. Ein Bucket nimmt das Licht während des Zeitraums t_0 auf, während ein zweites Bucket zeitversetzt beginnend ab t_0 über dieselbe Periodendauer t_0 das reflektierte Licht aufnimmt. Abbildung 2.10 veranschaulicht das Konzept. Die elektrischen Ladungen Q_1 und Q_2 der jeweiligen Buckets C_1 und C_2 werden genutzt, um den Zeitversatz Δt aus dem Verhältnis zu berechnen

$$\Delta t = t_0 \cdot \frac{Q_2}{Q_1 + Q_2}. \tag{2.35}$$

Durch Einsetzen von Δt in Gleichung 2.34 lässt sich die Distanz bestimmen

$$d = \frac{c \cdot t_0}{2} \cdot \frac{Q_2}{Q_1 + Q_2}. \tag{2.36}$$

Die maximale Distanz d_{max}, die gemessen werden kann, ergibt sich aus der Zeit, die das Licht braucht, um während der Periodendauer t_0 von einem Objekt reflektiert zu werden und wieder zurück zum Sensor zu gelangen

$$d_{max} = \frac{c \cdot t_0}{2}. \tag{2.37}$$

Da Licht, das länger als die maximale Zeit von t_0 benötigt, um zurückzukehren, nicht mehr von C_1 erfasst wird, führt eine größere Distanz dazu, dass die Ladung von Q_1 einen Wert von 0 hat und somit die Distanz nicht mehr korrekt berechnet werden kann, da das Verhältnis unabhängig von der Distanz einen Wert von 1 annimmt, solange Q_2 eine

Ladung ungleich 0 hat.

2.2.2 Continuous-Wave Modulation

Im Kontrast zur einfachen Pulsdauermodulation werden bei einer Continuous-Wave Modulation entweder rechteckförmige oder sinusförmige Wellenfunktionen emittiert. Üblicherweise werden Rechteckfunktionen verwendet, da diese einfacher in elektronischen Schaltkreisen zu realisieren sind [Li14].

Wenn die Frequenz f eines Signals bekannt ist, lässt sich daraus die Phasenverschiebung des reflektierten Signals berechnen, womit sich die Flugzeit Δt bestimmen lässt. Das Ermitteln der Phasenverschiebung unterscheidet sich abhängig von der gewählten Modulation. Hierbei sei angemerkt, dass es sich bei der Frequenz f nicht um die Frequenz des Infrarotlichtes handelt, sondern um die Frequenz des Nutzsignals.

Creath [Cre88] verglich mehrere Phasen Messmethoden, wobei laut Giancola et al. [GVS18] die Vier Bucket Variante am weitesten vertreten ist und auch von Meister et al. [MNK13] in ihrer Arbeit zur Simulation einer Time-of-Flight Kamera verwendet wird. Hierbei wird die Anzahl an auftreffenden Photonen in Form von elektrischen Ladungen Q_1, Q_2, Q_3 und Q_4 der vier Buckets C_1, C_2, C_3 und C_4 gemessen und die Phasenverschiebung ϕ geschätzt

$$\phi = atan\left(\frac{Q_3 - Q_4}{Q_1 - Q_2}\right). \tag{2.38}$$

Der Zeitversatz Δt lässt sich mit Hilfe der Phasenverschiebung ϕ berechnen

$$\Delta t = \frac{\phi}{2\pi f}. \tag{2.39}$$

Die daraus resultierende Distanz d lässt sich schlussendlich durch Einsetzen des Zeitversatzes Δt in die Gleichung 2.34 bestimmen.

Zusätzlich kann nach Li [Li14] mit den Ladungen Q_1, Q_2, Q_3 und Q_4 die Intensität A des Infrarotsignals und der Versatz B, der durch die ambiente Beleuchtung verursacht wird, berechnet werden

$$A = \frac{\sqrt{(Q_1 - Q_2)^2 + (Q_3 - Q_4)^2}}{2} \tag{2.40}$$

$$B = \frac{Q_1 + Q_2 + Q_3 + Q_4}{4}. \tag{2.41}$$

Die maximale Distanz, die mittels der Continuous-Wave Modulation gemessen werden kann, wird durch die Wellenlänge des Nutzsignals bestimmt. Da das Licht einen Hin- und Rückweg hat, berechnet sich die maximale Distanz aus der Hälfte der Wellenlänge

$$d_{amb} = \frac{c}{2f}. \tag{2.42}$$

Da sich das Signal bei einer Phasenverschiebung von 2π wiederholt, führt die Schätzung des Tiefenwerts mittels Phasenverschiebung zu einer Mehrdeutigkeit (Ambiguität) ab einem maximal messbaren Tiefenwert d_{amb}. Eine Möglichkeit zur Erhöhung der maximalen

Distanz ist die, die Frequenz zu reduzieren. Dies führt jedoch zu einer Reduktion der Genauigkeit, da die Stärke des Rauschens σ zunimmt. Das Rauschverhalten kann nach Li [Li14] mit folgender Gleichung approximiert werden:

$$\sigma = \frac{c}{4\sqrt{2}\pi f} \cdot \frac{\sqrt{A+B}}{c_d A} \tag{2.43}$$

Der Modulationskontrast c_d beziffert dabei, wie gut der Time-of-Flight Sensor Photoelektronen sammelt und separiert. Der Gleichung 2.43 kann entnommen werden, dass eine hohe Amplitude und eine hohe Modulationsfrequenz die Genauigkeit erhöht, eine hohe Modulationsfrequenz allerdings auch zu einer geringeren Reichweite führt.

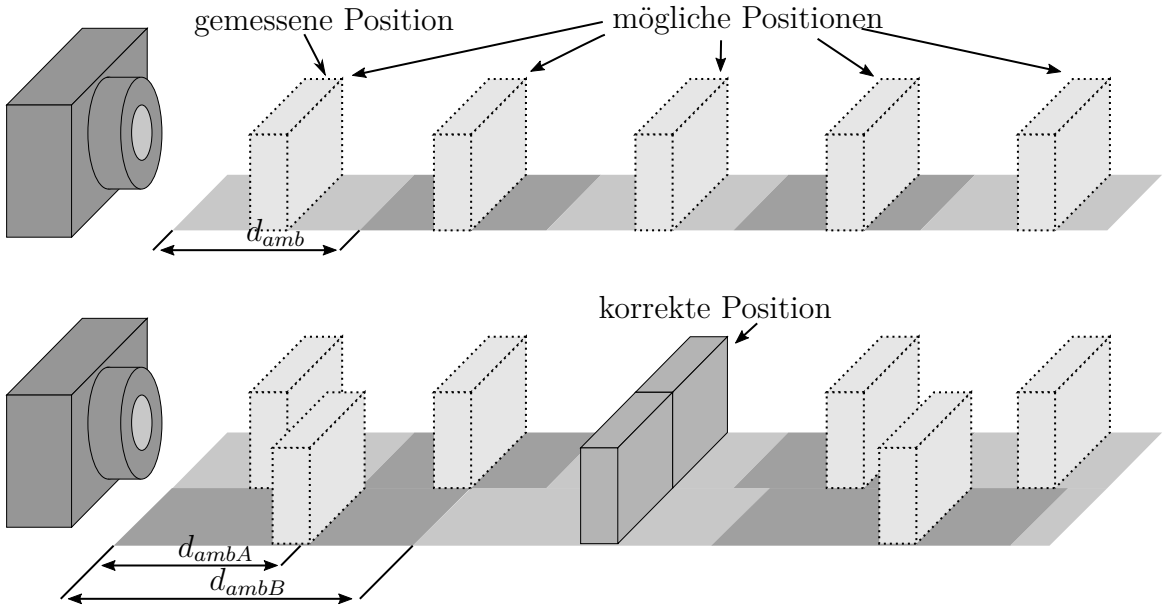


Abbildung 2.11: Konzept eines Time-of-Flight Sensors, der mit mehreren Frequenzen arbeitet.

Einige Time-of-Flight Systeme, wie die Kinect v2, kombinieren daher, wie in Abbildung 2.11 veranschaulicht, mehrere Aufnahmen mit unterschiedlichen Frequenzen, um die Genauigkeit der geschätzten Tiefenwerte bei gleichzeitig großen Distanzen zu erhöhen. Der echte Tiefenwert wird bestimmt, indem die Tiefenwerte aller Frequenzen miteinander verglichen werden und ein Tiefenwert ermittelt wird, der in allen Frequenzen miteinander übereinstimmt. Giancola et al. [GVS18] untersuchte in diesem Zusammenhang im Detail den Aufbau und die Funktionsweise der Kinect v2. Dabei wurde festgestellt, dass die Kinect v2 drei unterschiedliche Frequenzen verwendet. Für ein Tiefenbild wird zunächst eine Aufnahme bei 80 MHz für 8 ms erstellt, gefolgt von einer Aufnahme mit einer Frequenz von 16 MHz für 4 ms und einer abschließenden Aufnahme mit einer Frequenz von 120 MHz für 8 ms. Jede Frequenz hat dabei eine andere Mehrdeutigkeitsdistanz d_{amb}. Die Frequenz, bei der die Mehrdeutigkeit aller Frequenzen übereinstimmen, wird als *Schlagfrequenz* oder *Beat Frequency* bezeichnet. Diese Frequenz ist meist niedriger und weist eine höhere maximale Distanz auf [Li14]. Die Schlagfrequenz der Kinect v2 beträgt 8 Mhz, was eine maximale mehrdeutigkeitsfreie Distanz von 18,73 Metern ermöglicht.

Amplitudenmodulation mit rechteckförmigen Signal

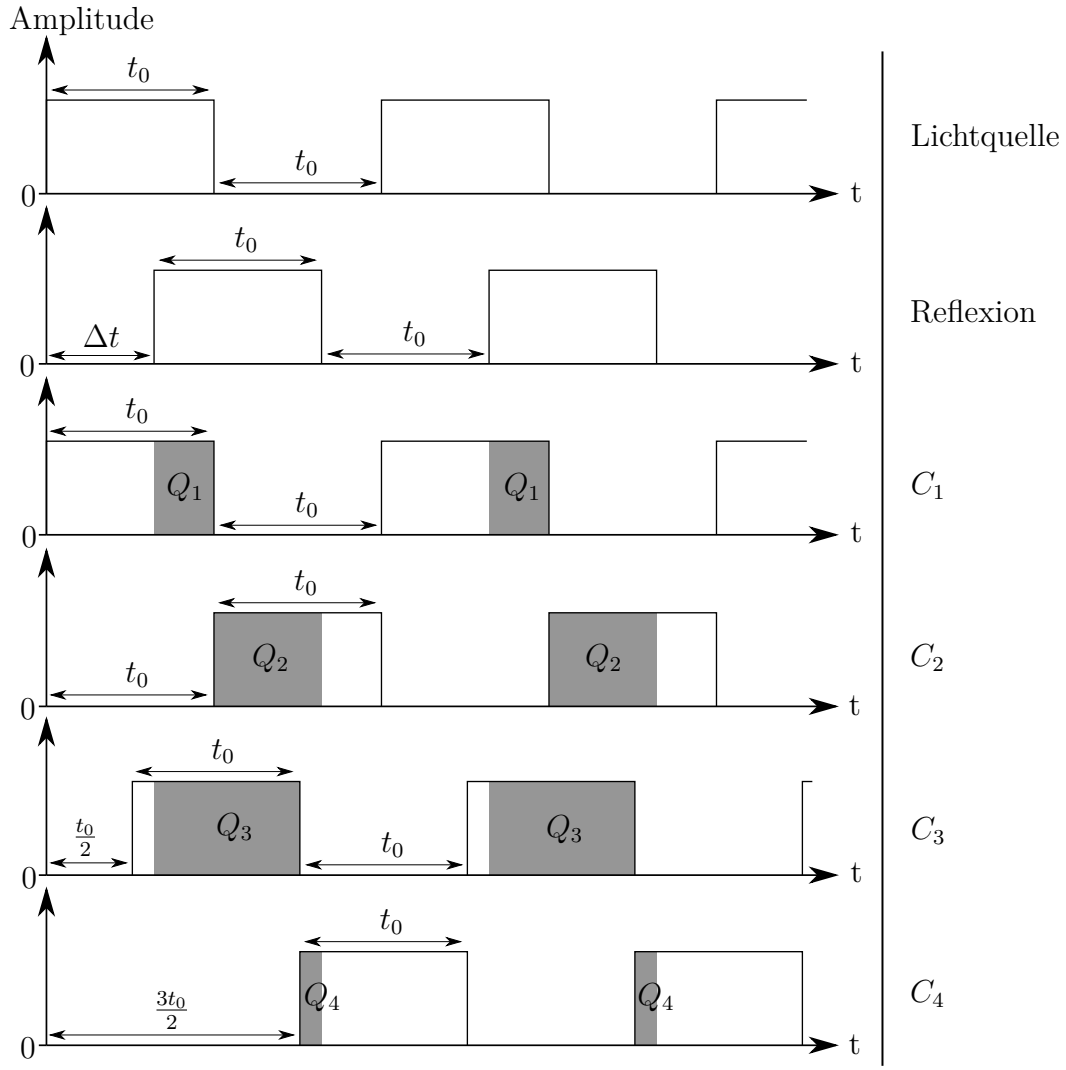


Abbildung 2.12: Approximation der Phasenverschiebung eines rechteckförmigen Signals mit vier Buckets C_1, C_2, C_3 und C_4 nach Wyant [Wya82].

Bei einem rechteckförmigen Signal werden wie bei der Pulsdauermodulation in regelmäßigen Abständen Lichtpulse emittiert. Im Unterschied zur Pulsmethode stimmt die Dauer der Lichtpulse und die Abstände zwischen den Lichtpulsen überein, wodurch ein kontinuierliches Signal entsteht, dessen Phasenverschiebung durch Gleichung 2.39 bestimmt werden kann. Wie Abbildung 2.12 verdeutlicht, geschieht dies mit Hilfe der oben genannten vier, jeweils um 90° versetzten, Buckets. Mittels der Ladungen Q_1, Q_2, Q_3 und Q_4 kann die Phasenverschiebung des reflektierten Signals bestimmt werden. Die Pulsdauer t_0 eines Rechteckpulses eines Signals mit einer Frequenz f lässt sich anhand der halben Wellenlänge berechnen

$$t_0 = \frac{1}{2f}. \tag{2.44}$$

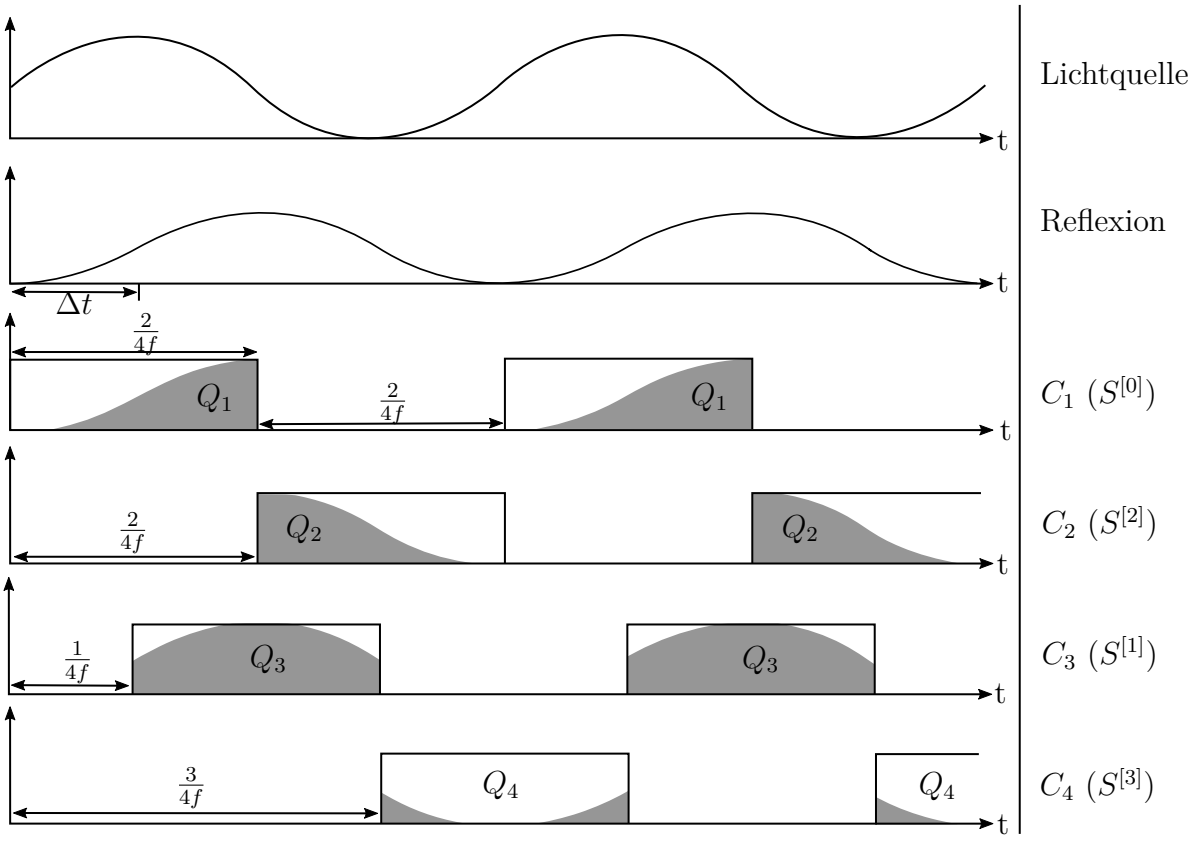


Abbildung 2.13: Approximation der Phasenverschiebung eines sinusförmigen Signals mit vier Buckets C_1, C_2, C_3 und C_4 nach Hertzberg et al. [HF14].

Amplitudenmodulation mit sinusförmigem Signal

Im Folgenden wird das Ermitteln der Flugzeit Δt des Lichts mittels Approximation der Phasenverschiebung ϕ einer Sinuswelle nach Hertzberg et al. [HF14] erläutert. Im Falle einer Sinusmodulation entfallen die hohen Anforderungen an die Flankensteilheit des Rechtecksignals, die besonders bei hohen Frequenzen entscheidend ist. Die Verwendung von Sinuswellen erlaubt in der Praxis daher eine robustere und genauere Schätzung der Phasenverschiebung ϕ, wodurch der Zeitversatz Δt genauer bestimmt werden kann [GVS18].

Bei dem generierten Signal handelt es sich um eine Sinuswelle $\psi(t)$ mit einer bekannten Amplitude I, Frequenz f und Versatz c_o

$$\psi(t) = I \cdot \sin(2\pi f t) + c_o. \tag{2.45}$$

Der Versatz ist dabei so gewählt, dass er größer oder gleich I ist, da kein Licht mit negativer Intensität emittiert werden kann. Ein Anteil α des emittierten Lichtes kehrt zum Sensor zurück, wobei die Phase des Signals in Abhängigkeit von der Zeit Δt, die das Licht unterwegs war, verschoben ist. Zusätzlich kommt ein konstanter Anteil an ambienter Beleuchtung c_B hinzu. Das gemessene Licht kann also wie folgt beschrieben werden:

$$z(t) = \alpha \cdot \psi(t - \Delta t) + c_B. \tag{2.46}$$

Zur Bestimmung der Phasenverschiebung wird das Signal jeweils zeitversetzt über eine halbe Wellenlänge in den einzelnen Buckets integriert

$$s^{[k]} = \int_{\frac{k}{4f}}^{\frac{k+2}{4f}} z(t)dt = \left[c_1 t - \frac{2\alpha a}{4\pi f}\cos(2\pi f(t - \Delta t))\right]_{\frac{k}{4f}}^{\frac{k+2}{4f}} \tag{2.47}$$

$$= c_2 + \frac{2\alpha a}{2\pi f}\cos\left(\frac{\pi}{2}k - 2\pi f \Delta t\right). \tag{2.48}$$

Die Phasenverschiebung wird analog zur Gleichung 2.38 aus den Ladungen der Buckets C_1, C_2, C_3 und C_4 bestimmt

$$\phi = atan\left(\frac{s^{[1]} - s^{[3]}}{s^{[0]} - s^{[2]}}\right). \tag{2.49}$$

Die Flugzeit wird anschließend aus Gleichung 2.39 und die daraus resultierende Distanz schließlich durch Einsetzen in Gleichung 2.34 berechnet.

3 Verwandte Arbeiten

Dieses Kapitel beschäftigt sich mit Forschungsarbeiten, die ein ähnliches oder das selbe Ziel verfolgen. Deren Ergebnisse werden im Kontext der Zielsetzung dieser Arbeit bewertet.

3.1 Photon Mapping based Simulation of Multi-Path Reflection Artifacts in Time-of-Flight Sensors

Die Arbeit von Meister et al. [Mei12] ist die erste Arbeit, die den Einfluss von indirekter Beleuchtung auf die Tiefendaten der Time-of-Flight Sensoren durch ein globales Beleuchtungsmodell simuliert. Dabei konzentriert sie sich auf die Simulation des durch indirekte Beleuchtung verursachten Multipath Fehlers, der in Unterabschnitt 4.0.5 genauer untersucht wird. Zu diesem Zweck wurde die *Photon Mapping* Implementierung des LuxRenders modifiziert und die Beleuchtungsberechnung um eine zeitliche Komponente erweitert. Photon Mapping ist ein Verfahren, das 1996 von Jensen [Jen96] vorgestellt wurde und die Rendergleichung löst, indem in zwei Phasen zunächst Photonen durch Lichtquellen in die Szene geschossen werden und diese anschließend in der zweiten Phase beim Berechnen des Bildes zur Beleuchtung verwendet werden.

In der ersten Phase werden von jeder Lichtquelle aus virtuelle Photonen emittiert. Beim der Kollision des Photons mit der Oberfläche der Geometrie werden dessen Position, Intensität und Einfallswinkel in einer Punktewolke gespeichert. Meister et al. [Mei12] erweitern den Datensatz zusätzlich um die zurückgelegte Distanz des Photons. Anschließend wird das Photon entweder entsprechend der Eigenschaften der definierten BRDF reflektiert oder vollständig absorbiert. Dies wird fortgeführt, bis eine vordefinierte Anzahl an Photonen in die Szene emittiert worden sind.

In der zweiten Phase wird die Szene mittels Ray Tracing gerendert. Dazu werden pro Pixel mehrere Strahlen aus der Richtung des Betrachters in die Szene geschossen. Bei jeder Kollision eines Strahls mit der Szene wird am Schnittpunkt die Strahlungsdichte in Richtung des Betrachters berechnet, indem für die direkte Beleuchtung für jede Lichtquelle getestet wird, ob diese vom Schnittpunkt aus sichtbar ist. Für die indirekte Beleuchtung werden die Photonen in einem Radius zur Berechnung der Beleuchtungsdichte herangezogen und die Reflexion wird in Richung des Betrachters entsprechend der BRDF der Oberfläche bestimmt. Die zurückgelegte Distanz des Photons ist sowohl für die direkte als auch für die indirekte Beleuchtung bekannt, was die Berechnung der einzelnen Phasenbilder betrifft.

Meister et al. [Mei12] simulieren in ihrer Arbeit ausschließlich sinusförmige Signale, wie sie in Abschnitt 2.2.2 beschrieben sind. Der Algorithmus läuft komplett auf der CPU und braucht für die Generierung eines 200x200 Tiefenbildes laut eigenen Angaben mit einem

2,4 GHz Xeon Prozessor ungefähr 2 Stunden. Dazu wird jedes der vier Phasenbilder, welche die Ladungen Q_1, Q_2, Q_3 und Q_4 enthalten, nacheinander generiert und das Tiefenbild anschließend mittels Gleichung 2.49 berechnet.

Die Ergebnisse der Simulation zeigen, dass sich Fehler, die durch Interreflexionen verursacht werden, mittels Photon Mapping simulieren lassen. Allerdings konnten nicht alle beobachteten Artefakte korrekt simuliert werden, die im Testaufbau beobachtet wurden, die besonders durch flache Einfallswinkel des Lichts auf die Oberfläche verursacht werden. Die hohe Berechnungsdauer schränkt den Einsatz zusätzlich auf statische Szenen ein, da die Berechnung dynamischer Szenen mit einem Umfang weniger Sekunden mehrere Tage oder Wochen benötigen würde.

3.2 Simulation of Time-of-Flight Sensors using Global Illumination

Die Arbeit von Meister et al. [MNK13] kommt dem Ansatz dieser Arbeit am nächsten. Sie baut auf den Erfahrungen ihrer vorangegangenen Arbeit [Mei12] auf und konzentriert sich darauf Multiple Path Reflexionen mittels bidirektionalem Path Tracing zu simulieren. Die Arbeit konzentriert sich dabei ebenfalls auf die Simulation des Multipath Problems und erweitert dafür den Path Tracing Algorithmus des LuxRender um eine zeitliche Komponente. Das Grundprinzip eines Path Tracing Algorithmus wurde bereits in Unterabschnitt 2.1.6 beschrieben und wird in Kapitel 5 im Detail erläutert. Bei bidirektionalem Path Tracing handelt es sich um eine Erweiterung dieses Prinzips. Dabei werden zusätzlich zu den Strahlen aus der Position des Betrachters, Strahlen von der Lichtquelle in die Szene geschossen und die Schnittpunkte der Strahlen mit der Geometrie jeweils verbunden. Durch diese Optimierung konvergieren besonders Szenen, die ausschließlich indirekt beleuchtet werden, deutlich schneller zum Ergebnis.

Meister et al. [MNK13] erweiterten diesen bidirektionalen Path Tracer um eine zeitliche Komponente, indem jedem Sicht- und Lichtstrahl bei der Berechnung eines Schnittpunktes mit der Geometrie die zurückgelegte Strecke mitgegeben wurde. Anschließend wird jeder Lichtbeitrag zum Ergebnis gewichtet aufsummiert. Jedes Phasenbild wird dabei wiederum unabhängig von den anderen gerendert. Aus allen wird anschließend das Tiefenbild berechnet, während bei der Generierung der Phasenbilder ausschließlich sinusförmige Signale betrachtet werden.

Die Ergebnisse der Simulation variierten in Abhängigkeit zu der gewählten maximalen Tiefe der verfolgten Pfade. So gab es bei einer Pfadlänge von 1 keinerlei Interreflexionen und die Tiefenwerte entsprachen dem Ground Truth Datensatz, während mit jeder weiteren Erhöhung der Pfadlänge ein stärker Einfluss durch Interreflexionen gemessen werden konnte. Allerdings traten ebenfalls Artefakte auf, die durch den Path Tracer verursacht wurden. Diese Artefakte verringerten sich bei höheren Pfadlängen und ab einer Pfadlänge von 8 waren keine Unterschiede zu höheren Pfadlängen zu erkennen. Allerdings konnten Effekte durch starke Interreflexionen an Kanten, die sich in ‚Ausbeulungen' im Tiefenbild äußern, durch dieses Verfahren nicht simuliert werden.

Meister et al. [MNK13] sind zu dem Schluss gekommen, dass Photon Mapping weniger als Path Tracing Ansätze dazu geeignet ist, um Time-of-Flight Sensoren zu simulieren.

Durch Modifikation des bidirektionalen Path Tracing Algorithmus konnten die Auswirkungen indirekter Beleuchtung auf das Tiefenbild physikalisch plausibel simuliert werden, auch wenn nicht alle Probleme adressiert werden konnten. So wurden ausschließlich ideale sinusförmige Wellen angenommen und die Auswirkungen der Non-Linearität der Sensoren nicht berücksichtigt. Allerdings wird der Algorithmus auf der CPU ausgeführt, was eine hohe Berechnungsdauer mit sich zieht und daher nicht für Simulationen mit einer hohen Anzahl an Bildern geeignet ist.

3.3 Detailed Modelling and Calibration of a Time-of-Flight Camera

Die Arbeit von Hertzberg und Frese [HF14] konzentriert sich auf eine detaillierte physikalisch motivierte Kalibrierung der PMD CamBoard Kamera zur Korrektur von typischen Artefakten, die durch Time-of-Flight Kameras erzeugt werden, die mit sinusförmig moduliertem Licht arbeiten. Hertzberg und Frese [HF14] führen dazu ein idealistisches Sensor Modell ein und ergänzen dieses Modell um typische Abweichungen, die in der Realität existieren. Dabei konzentriert sich die Arbeit auf die detaillierte Kalibrierung des Sensors und ignoriert Effekte, die durch die Lichtausbreitung verursacht werden. Zu diesem Zweck wird die erwähnte Kamera mittels eines Schachbrettes kalibriert, um die Linsenverzeichnung zu ermitteln. Zusätzlich wird die Abweichung und die Non-Linearität einzelner Pixel korrigiert und die Vignettierung der Infrarot LED berücksichtigt.

Ein besonderer Fokus liegt auf der Korrektur des *Lens Scattering* Effekts, der in Unterabschnitt 4.0.6 im Detail untersucht wird. Dazu wird ein Retroreflektor vor einen wenig reflektierenden Hintergrund positioniert und der Einfluss gemessen, den das retroreflektierte Licht auf die umliegenden Pixel hat. Aus den Ergebnissen der Messung wurde eine Punktspreizfunktion berechnet, die sich aus einer Summe von Gaußfunktionen zusammensetzt und den Einfluss eines Pixels auf seine Nachbarpixel angibt. Diese Funktion wird genutzt um dein Einfluss des Lens Scattering zu kompensieren.

Abschließend wurde gezeigt, wie mit Hilfe von 64 Einzelbildern einer statischen Szene mit unterschiedlichen Belichtungszeiten sowohl ein HDR, als auch ein Tiefenbild erzeugt werden konnte. Dabei wurden allerdings Effekte durch Interreflexionen des Lichts ignoriert, beispielsweise der Einfluss von Umgebungslicht und der Einfluss der Temperatur auf den Sensor.

3.4 Real-time Simulation of Time-of-Flight Sensors and Accumulation of Range Camera Data

Das Kapitel 3 der Dissertation von Keller [Kel15] beschäftigt sich mit der Echtzeit Simulation von Time-of-Flight Kameras und typischen Artefakten wie *Bewegungsunschärfe*, die verursacht werden, wenn sich Objekte während der Aufnahme im Bild bewegen, *Mixed Pixels*, die Thema in Unterabschnitt 4.0.7 sind, und systematische Abweichungen, die in Unterabschnitt 4.0.4 untersucht werden. Das Hauptziel der Simulation von Keller ist die

Generierung von synthetischen Tiefensensordaten und die Möglichkeit Kameraparameter zur Laufzeit zu manipulieren.

Zur Simulation des Tiefensensors wird die Rasterisierungspipeline der GPU verwendet, indem für OpenGL Vertex und Pixelschader entwickelt wurden, die Phasenbilder generieren. Dazu wird die Tiefeninformation, die die OpenGL Render Pipeline bereits liefert, genutzt, um die Phasenverschiebung der reflektierten Wellenfunktion zu bestimmen. Da jeder Pixel nur einen Sichtstrahl simulieren würde, wird zur Simulation des Mixed Pixels Fehlers das Bild in doppelter Auflösung gerendert und jeder Pixel des Endbildes aus dem Durchschnitt von vier Pixeln des Bildes in höherer Auflösung gebildet. Die Bewegungsunschärfe wird simuliert, indem das Bild vier Mal in unterschiedlichen Zeitschritten gerendert wird und anschließend die Phasenbilder aus dem Durchschnitt aller vier Einzelbilder berechnet werden.

Der systematische Fehler wird simuliert, indem dem die Abweichung von Ground Truth Tiefenwerten mit einem echten Sensor gemessen und diese in einer Textur gespeichert werden. Diese Textur dient im Anschluss als Look-up Table, um den tiefenwertabhängigen systematischen Fehler auf das Tiefenbild aufzurechnen. Da die Messungen der echten Tiefenwerte durch ein statisches Rauschen beeinflusst werden, wird die Fehlerfunktion mit Hilfe eines Sinussignals approximiert, das durch die Fourier-Transformation der Messwerte berechnet wird.

Der zufällige Fehler wird für jedes Phasenbild einzeln berechnet, indem intensitätsabhängiges Rauschverhalten angenommen wird und durch eine Gaußverteilung approximiert wird. Die Intensität des zufälligen Fehlers wird durch zwei Koeffizienten $\alpha, \beta \geq 0$ gesteuert und das Phasenbild wird folgendermaßen modifiziert:

$$A_{noisy,i} = A_i + \alpha \cdot n_{rand} + A_i \cdot \beta \cdot n_{rand}, \tag{3.1}$$

wobei A_i für ein Phasenbild steht und $n_{rand} \in [-1, 1]$ eine normalverteilte Zufallszahl darstellt.

Die in [Kel15] vorgestellte Simulation zeigt hohe Übereinstimmungen mit dem zur Kalibrierung genutzten Sensor in Bezug auf den systematischen Fehler und das Mixed Pixels Verhalten, basiert allerdings nicht auf einer physikalischen Simulation, sondern auf der Nachbildung des eingemessenen Verhaltens des Kamerasystems. Dabei werden Fehler, die durch die Lichtausbreitung verursacht werden wie Fehler durch Lens Scattering oder Interreflexionen nicht adressiert.

3.5 Zusammenfassung

Nur wenige Simulationen erfüllen die Zielsetzung dieser Arbeit. Vor allem zu erwähnen sind dabei die Arbeiten von Meister et al. [Mei12][MNK13], die durch die Simulation der Lichtausbreitung erstmals Artefakte simulieren konnten, die durch Interreflexionen erzeugt werden. Allerdings werden dazu Beleuchtungsmodelle verwendet, die stark von gemessenen Daten abweichen, weshalb nicht alle durch Interreflexionen verursachten Artefakte simuliert werden konnten. Darüber hinaus wurde angenommen, dass sich die Lichtquelle und die Kamerablende an derselben Position befinden, was von der Realität abweicht.

Häufig wird angemerkt, dass aufgrund der Komplexität der Berechnung der Einsatz in interaktiven Anwendungsgebieten nicht geeignet ist [HF14][LHK15]. Keller [Kel15] und Lamberts et al. [LHK15] stellten in ihren Arbeiten daher Simulationen vor, die auf der GPU ausgeführt werden können und mit Hilfe der Rendering Pipeline durch Verwendung von Vertex- und Pixelshadern entwickelt wurden. Durch Verwenduung der OpenGL Rasterisierungspipeline ergeben sich Einschränkungen, weshalb die Arbeiten auf physikalisch motivierte Simulationen verzichten und stattdessen das Tiefenbild oder die einzelnen Phasenbilder nach zuvor eingemessenen Abweichungen echter Time-of-Flight Kamerasysteme modifizieren.

4 Analyse der Kameras

(a) Microsoft Kinect v2

(b) ASUS Xtion 2

(c) IFM O3D303

Dieses Kapitel beschäftigt sich mit der Analyse der einzelnen Time-of-Flight Kamerasysteme, die im Rahmen der Arbeit genutzt werden. Dafür werden die Eigenschaften der IFM O3D303 (Abbildung 4.1c), der Microsoft Kinect v2 (Abbildung 4.1a) und der Asus Xtion 2 (Abbildung 4.1b) untersucht. In Tabelle 4.1 sind die vom Hersteller veröffentlichten technischen Daten der Kamerasysteme aufgelistet. Dabei ist anzumerken, dass es sich bei der Kinect v2 und der Xtion 2 um RGB-D Kamerasysteme handelt, in denen zusätzlich zum Time-of-Flight Sensor eine RGB Kamera verbaut wurde, während die O3D303 ausschließlich über einen Time-of-Flight Sensor verfügt. Die Kinect v2 liefert ein RGB Farbbild, ein Tiefenbild und ein Infrarotbild, während die Xtion 2 nur ein Farbbild und ein Tiefenbild und die O3D303 ein Tiefenbild und ein Infarotbild liefert. Die Kinect v2 und die Xtion 2 werden über USB 3.0 betrieben und die O3D303 wird über einen Netzwerkanschluss angesprochen, über den die Daten gesendet werden. Da, wie in Unterabschnitt 2.2.2 erläutert, die Mehrdeutigkeitsdistanz in direkter Relation zur genutzten Frequenz steht, wurde anhand der gemessenen Maximaldistanz des Tiefenbildes für die O3D303 eine Frequenz von 30 Mhz und für die Xtion 2 eine Frequenz von 20 Mhz für die genutzte Wellenfunktion festgestellt. Keine der Kameras verfügt über einen einstellbaren

Tabelle 4.1: Technische Spezifikationen der Time-of-Flight Kamerasysteme

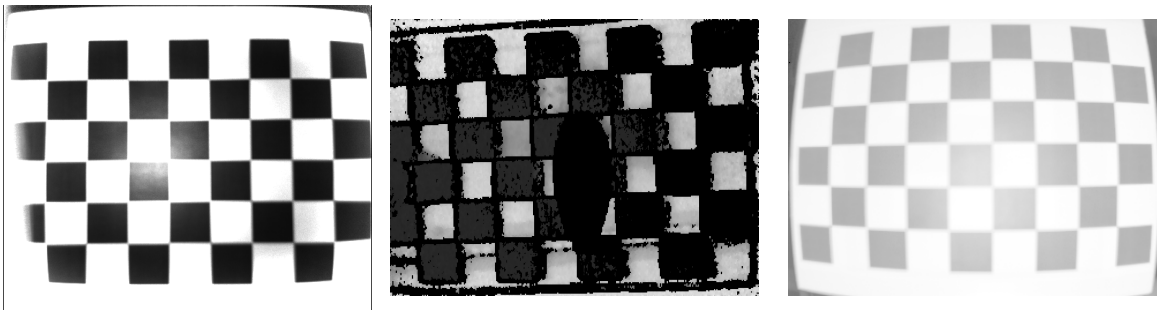
Gerät	Kinect v2	Xtion 2	O3D303
Auflösung Farbbild	1920 x 1080	2592 x 1944	-
Auflösung Tiefenbild	512 x 424	640 x 480	352 x 264
Framerate Tiefenbild	30 fps	30 fps	4 - 25 fps
Zugriff auf IR Aufnahmen	Ja	Nein	Ja
Field of View Tiefenbild	70° x 60°	74° x 52°	60° x 45°
Reichweite	0,5 - 4,5 m	0,8 - 3,5 m	0,3 - 30 m

Fokus oder eine einstellbare Blendenöffnung. Die O3D303 erlaubt allerdings eine feinere Konfiguration der Belichtungszeiten und weitere Konfigurationsmöglichkeiten, wie das Nutzen von wahlweise bis zu drei verschiedenen Frequenzen und drei Belichtungsmodi, in denen mehrere Bilder mit unterschiedlichen Belichtungszeiten erstellt werden, um eine hohe Belichtung und damit verbunden geringes Rauschen zu ermöglichen, ohne die Sensoren dabei überzubelichten. In der vorliegenden Arbeit wird für die Evaluation der O3D303 nur

eine Frequenz von 30 Mhz und der einfachste Belichtungsmodus verwendet, in dem nur eine Aufnahme mit variabler Belichtungszeit gemacht wird.

In den folgenden Unterkapiteln werden die einzelnen Kamerasysteme auf ihre Eigenschaften und auf die Beeinflussung des Tiefenbildes durch die Lichtausbreitung untersucht. Es sei angemerkt, dass zur besseren Unterscheidung die Graphen einem festen Schema folgen: Die Datensätze der Kinect v2 werden durch blaue Quadrate dargestellt, während für die Xtion 2 rote Kreise und für O3D303 schwarze Dreiecke verwendet werden.

4.0.1 Radiale Verzeichnung



(a) Microsoft Kinect v2 **(b)** ASUS Xtion 2 **(c)** IFM O3D303

Abbildung 4.2: Radiale Verzeichnung der Infrarotkameras, die mittels Schachbrettmustern veranschaulicht werden.

Die monokulare Kamera Kalibrierung kann als bereits ‚gelöstes Problem' betrachtet werden und unter Verwendung der durch OpenCV angebotenen Algorithmen durchgeführt werden [HF14]. Diese Arbeit verwendet Schachbrettmuster zur Kalibrierung der RGB und Infrarotkameras, um die *radiale Verzeichnung* oder *radial distortion* zu ermitteln. Als radiale Verzeichnung wird der geometrische Abbildungsfehler optischer Systeme bezeichnet, der duch die Linsenkrümmung verursacht wird. Die Abweichung des Kamerabildes vom linearen Kameramodell kann mit Hilfe eines bekannten Schachbrettmusters ermittelt werden. Durch die Kalibrierung werden außerdem die Brennweite und das optische Zentrum der Kamera bestimmt, die zur Schätzung der Position und Orientierung von ArUco Markern benötigt werden, die in den nächsten Unterkapiteln Verwendung thematisiert werden.

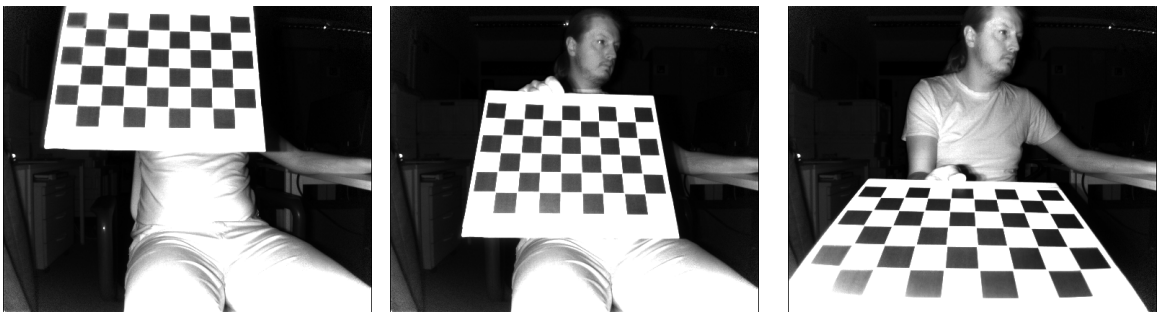


Abbildung 4.3: Auszug der Schachbrettaufnahmen, die zur Kalibrierung der Kinect v2 verwendet wurden.

Bei dem Schachbrettmuster handelt es sich um ein 8 mal 5 Schachbrett mit exakt quadratischen Feldern mit einer Kantenlänge von jeweils 32mm. Zur Kalibrierung wurden für jede Infrarotkamera zwischen 60 bis 80 Aufnahmen in unterschiedlichen Orientierungen und Positionierungen des Schachbretts erstellt (siehe Abbildung 4.3). Für die Xtion 2 musste eine transparente Folie mit einem Schachbrettmuster bedruckt werden, welche an einer Plexiglasscheibe angebracht wurde, da die Xtion 2 keinen Zugriff auf das Infarotbild liefert (siehe Abbildung 4.2b). Daher sind in der Abbildung Blendflecke zu sehen, die durch die Spiegelungen an der Plexiglasscheibe entstanden sind. Statt des Infarotbildes wurde deshalb das Tiefenbild zum Kalibrieren der Tiefenbildkamera verwendet. Das Ergebnis der Kalibrierung wird in Form einer Kameramatrix und Verzeichnungskoeffizienten ausgegeben, mit deren Hilfe sich die Verzeichnung beheben und das Kamerabild in ein lineares Kameramodell überführen lässt.

Die Kameramatrix wird folgendermaßen definiert:

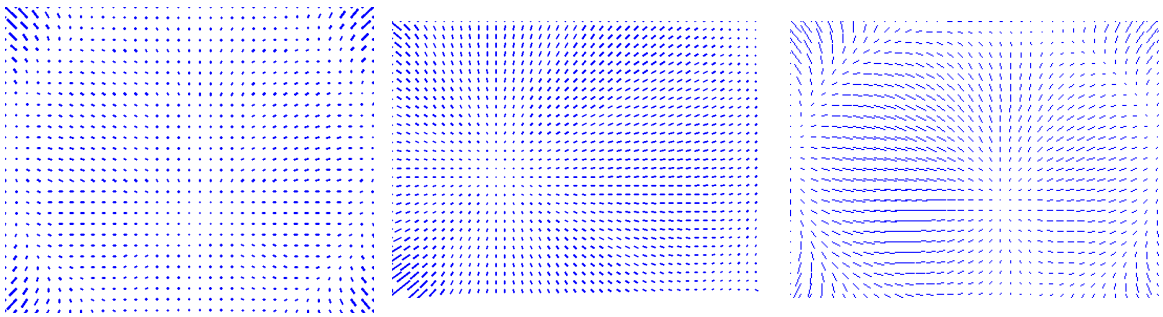
$$C_{matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

wobei f_x und f_y die Brennweite und c_x und c_y das optische Zentrum in Pixelkorrdinaten darstellen. Die Verzeichnungskoeffizienten werden als 4-, 5-, oder 8-Tupel übergeben:

$$D_{coefficients} = (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]]), \tag{4.2}$$

wobei in dieser Untersuchung ausschließlich 5-Tupel verwendet werden. Mit Hilfe der Kameramatrix und den Verzeichnungskoeffizienten kann nun für jede Pixelkoordinate (u, v) eine entzerrte Pixelkoordinate (u', v') bestimmt werden. Hier wird dafür auf die Implementierung von OpenCV zurückgegriffen:

$$\begin{aligned} x^{'} &\leftarrow (u - c_x)/f_x \\ y^{'} &\leftarrow (v - c_y)/f_y \\ (u', v') &= undistort(x^{'}, y^{'}, D_{coefficients}). \end{aligned} \tag{4.3}$$



(a) Microsoft Kinect v2 **(b)** ASUS Xtion 2 **(c)** IFM O3D303

Abbildung 4.4: Ergebnis der Kalibrierung und Darstellung der radialen Verzeichnung der Infrarotkameras.

Abbildung 4.4 veranschaulicht die Korrektur der genutzten Infrarotkameras. Dafür wurde in einem regelmäßigen Gitter eine Linie von (u, v) zu (u', v') gezeichnet, was den Grad der Verzeichnung darstellen soll. Für alle Kameras ist die Linsenform anhand der Verzeichnung

zu erkennen und es ist zu sehen, dass die Stärke der Verzeichnung zum Rand des Bildes zunimmt und in den Ecken besonders stark ausgeprägt ist.

4.0.2 Temperatur

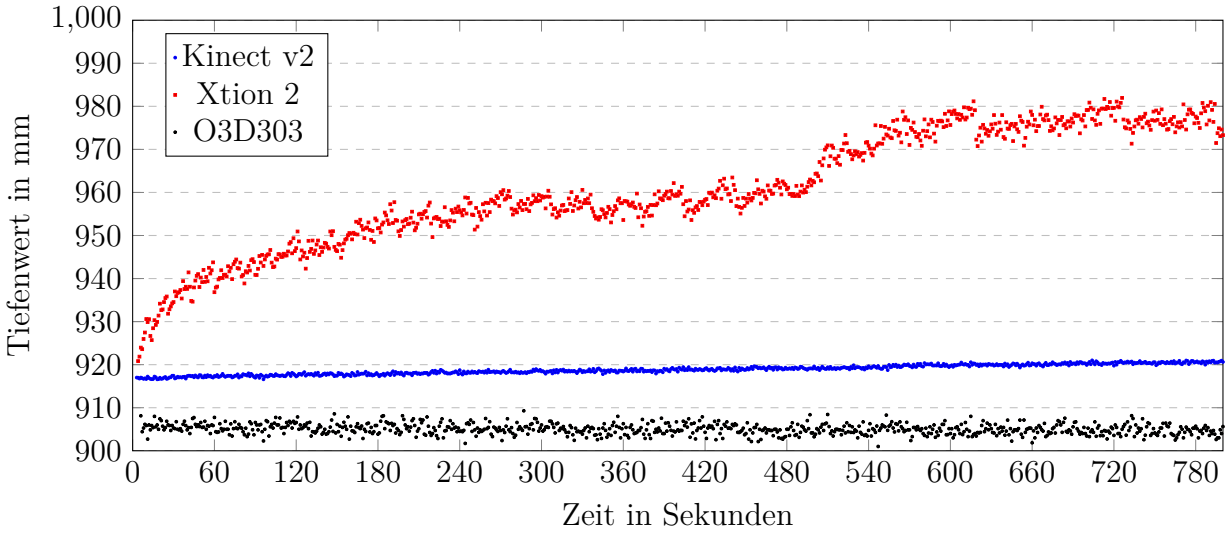
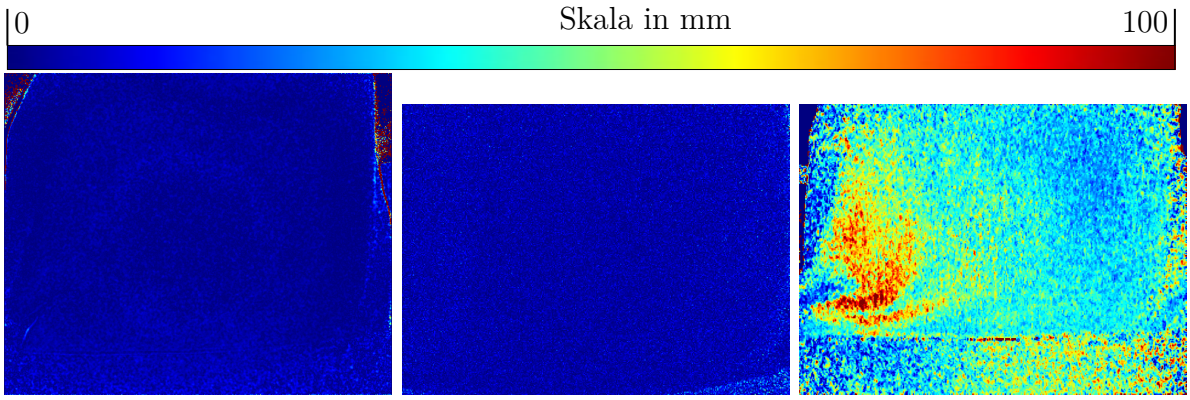


Abbildung 4.5: Die Veränderung der Tiefenwerte aller Sensoren über die Zeit.

Mehrere Quellen belegen, dass die gemessenen Tiefenwerte von Time-of-Flight Sensoren durch die Temperatur der Sensoren selbst beeinflusst werden, da diese sich im Laufe des Betriebs erwärmen [GVS18][HF14]. Während Hertzberg und Frese [HF14] in ihrer Arbeit lediglich darauf hinweisen, dass sich die Betriebstemperatur auf die Sensoren auswirken kann, untersuchten Giancola et al. [GVS18] den Einfluss der Temperatur der Kinect v2 auf die Tiefenwerte. Dabei wurden die Sensoren kalt gestartet und eine statische Szene wurde aufgenommen, während über die Zeit der Einfluss der Betriebstemperatur auf die gemessenen Tiefenwerte beobachtet wurde. Außerdem untersuchten Giancola et al. [GVS18] durch Zuhilfenahme zusätzlicher Lüfter den Einfluss einer besseren Kühlung. Dabei sind sie zum Ergebnis gekommen, dass die Temperatur die gemessenen Tiefenwerte des Gerätes beeinflusst und eine zusätzliche Kühlung dem entgegenwirkt. Auch in der vorliegenden Untersuchung werden daher die genutzten Geräte auf ihre Betriebstemperatur und ihren Einfluss untersucht. Dazu werden die Sensoren kalt gestartet und die optische Achse orthogonal zu einer ebenen Fläche ausgerichtet. Dabei wird der mittlere Pixel der Aufnahme beobachtet und die Veränderung über einen Zeitraum von 13 Minuten festgehalten. Die Abbildung 4.5 veranschaulicht die Veränderung der Tiefenwerte mit zunehmender Betriebszeit. Die Arbeit konzentriert sich nur auf die Temperatur der Sensoren selbst, weshalb der Einfluss der Umgebungstemperatur nicht berücksichtigt wurde. Da die Sensoren von der Umgebungstemperatur beeinflusst werden, wurden die Aufnahmen in einem klimatisierten Raum bei konstanten 23 Grad Raumtemperatur durchgeführt. Da Giancola et al. [GVS18] den Verlauf der Abweichung über einen Zeitraum von 24 Stunden beobachtet haben und dabei feststellten, dass sich die Abweichung nach ungefähr 10 Minuten stabilisiert, wurde auf eine Langzeituntersuchung verzichtet.

Da die Messung dabei auf den zentralen Pixel des Tiefenbildes begrenzt wurde, wurde zusätzlich untersucht, ob die Abweichung der Tiefenwerte, die durch die Temperatur ver-



(a) Microsoft Kinect v2 **(b)** IFM O3D303 **(c)** ASUS Xtion 2

Abbildung 4.6: Systematischer Fehler im Tiefenwert, der nach einer Betriebsdauer von 30 Minuten durch die Temperatur verursacht wird.

ursacht wird, für jeden Pixel des Bildes gleich stark ausfällt. In Abbildung 4.6 ist der Grad der Abweichung in Relation zum Bild zu sehen. Dafür wurde für jeden Pixel der Tiefenwert, der in der ersten Minute gemessen wurde, mit dem Tiefenwert verglichen, der nach 30 Minuten Betriebsdauer gemessen wurde. Dabei sind bei der O3D303 kaum Veränderungen auszumachen, während bei Kinect v2 leichte Varianzen, in Form eines Ringes im Bildzentrum zu sehen sind. Die Abweichung des Tiefenwerts der Xtion 2 ist allerdings stark von der betrachteten Szene abhängig. Da die detaillierte Untersuchung des Einflusses der Temperatur den Rahmen der Arbeit sprengen würde, wird dieser in der Simulation nicht berücksichtigt. Es wird aber festgehalten, dass die Temperatur einen Einfluss auf die Tiefenwerte hat und diese bei der Xtion 2 stark ausgeprägt sind, was in den folgenden Messungen einberechnet wurde.

4.0.3 Zufällige Fehler

Vorangegangene Arbeiten haben einen Zusammenhang zwischen dem Rauschverhalten und der gemessenen Distanz untersucht [GVS18][Kel15][But14]. Giancola et al. [GVS18] untersuchten in ihrer Arbeit speziell die Kinect v2 im Detail. Dabei wurde eine lineare Abhängigkeit zwischen Distanz und Rauschen festgestellt. Giancola et al. [GVS18] haben zur Messung einen auf die Kamera extrinsisch kalibrierten Roboterarm verwendet, weshalb die Distanz zwischen Kamera und Messfläche bekannt war. Sie untersuchten zufällige Fehler ab 1000 mm bis zu einer Distanz von 4000 mm und stellten dabei fest, dass das Rauschen bei einer Distanz zwischen 1000 mm und 1500 mm abnahm, bevor es wieder linear anstieg. Butkiewicz [But14] kam zu ähnlichen Ergebnissen.

Im Kontrast dazu wurde in dieser Arbeit kein kalibrierter Roboterarm verwendet und das Rauschverhalten in Abhängigkeit zur gemessenen Distanz des Time-of-Flight Sensors betrachtet. Dazu wurden zwei unabhängige Aufnahmen angefertigt. Die Kamera wurde dabei fixiert, während der Abstand einer ebenen Fläche, die orthogonal zur optischen Achse ausgerichtet ist, erhöht wurde. Die Fläche wurde in kleinen Schritten immer weiter von dem Sensor entfernt und es wurden jedes Mal jeweils 1000 Tiefenwerte aufgenommen, mit deren Hilfe die Standardabweichung σ für jede Distanz berechnet wurde. Dieser

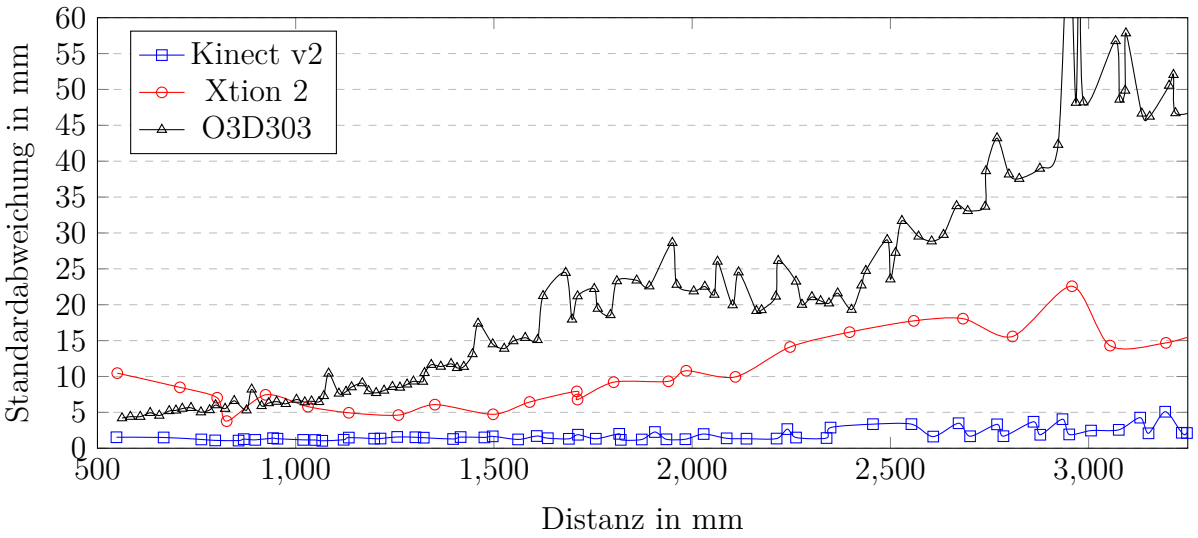


Abbildung 4.7: Zufälliger Fehler bei der Messung der Tiefenwerte in Abhängigkeit zur Distanz.

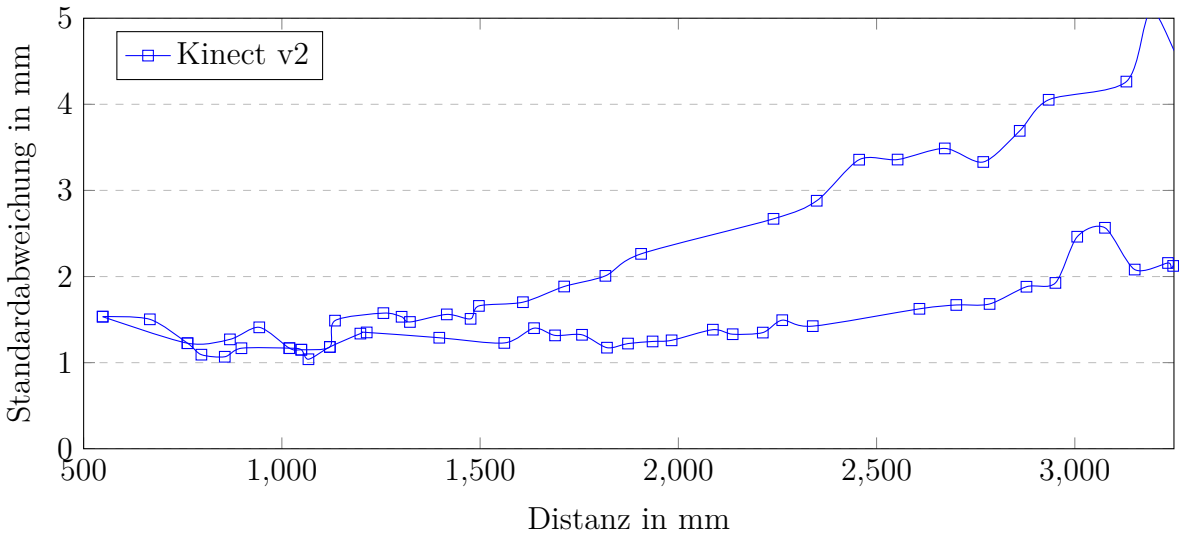


Abbildung 4.8: Zufälliger Fehler der Kinect v2 bei der Messung der Tiefenwerte in Abhängigkeit zur Distanz. Die zwei Verläufe repräsentieren Messwerte der hellen und der dunklen Fläche.

Vorgang wurde mit schwarzem und mit weißem Tonpapier durchgeführt. Abbildung 4.7 stellt die Standardabweichung der Sensoren in Abhängigkeit zur Distanz dar. Dabei sei anzumerken, dass die O3D303 mit einer konstanten Belichtungsdauer betrieben wurde. Die O3D303 ist in der Lage durch komplexere Belichtungsmodi mehrere Phasenbilder mit unterschiedlichen Belichtungszeiten zu erstellen und so das Rauschverhalten zu reduzieren. Zugunsten der Vergleichbarkeit wurde darauf verzichtet. Es ist zu erkennen, dass das Verhalten aller untersuchten Time-of-Flight Sensoren voneinander abweicht. Für die Kinect wurde der Verlauf der Standardabweichung zusätzlich in Abbildung 4.8 in einer anderen Skalierung dargestellt. Bei den zwei Verläufen handelt es sich um die Messwerte der hellen und der dunklen Flächen. Die dunkle Fläche weist dabei eine höhere Standardabweichung in Relation zur Distanz auf. Es ist wie in der Arbeit von Giancola et al.

[GVS18] ein leichter Abfall der Abweichung bis zu einer Distanz von 1000 mm und ein anschließender Anstieg zu erkennen. Die Xion 2 zeigt ein ähnliches Verhalten, wobei das Minimum eher im Bereich von 1500 mm zu finden ist, während das anfängliche Minimum bei der O3D303 nicht zu beobachten ist. Das lokale Minimum der Messwerte der O3D303 im Bereich von 2400 mm steht vermutlich in Korrelation zum systematischen Fehler, der in Unterabschnitt 4.0.4 untersucht wurde.

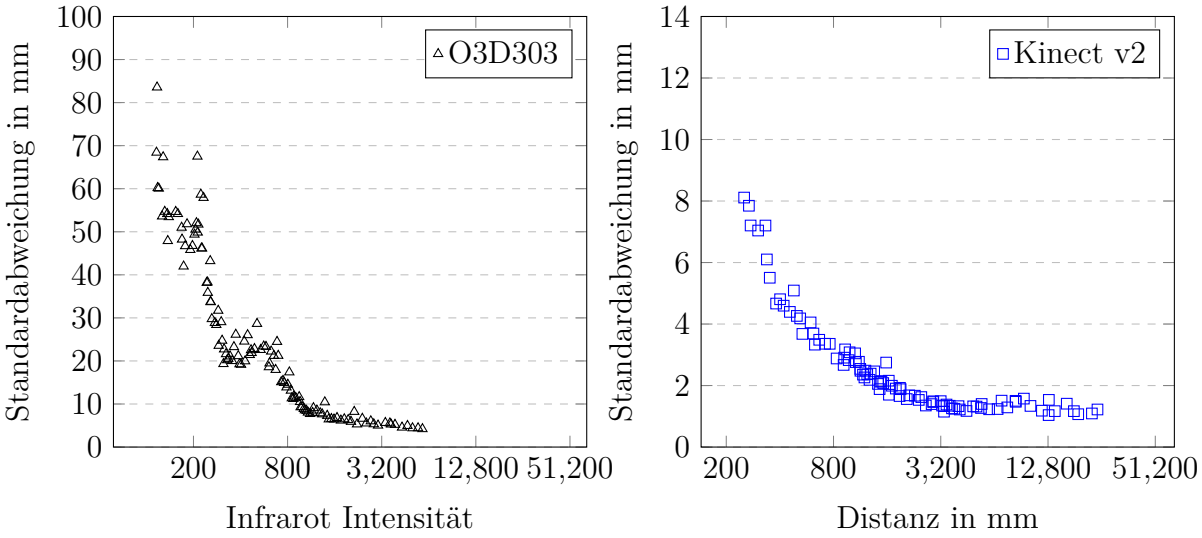
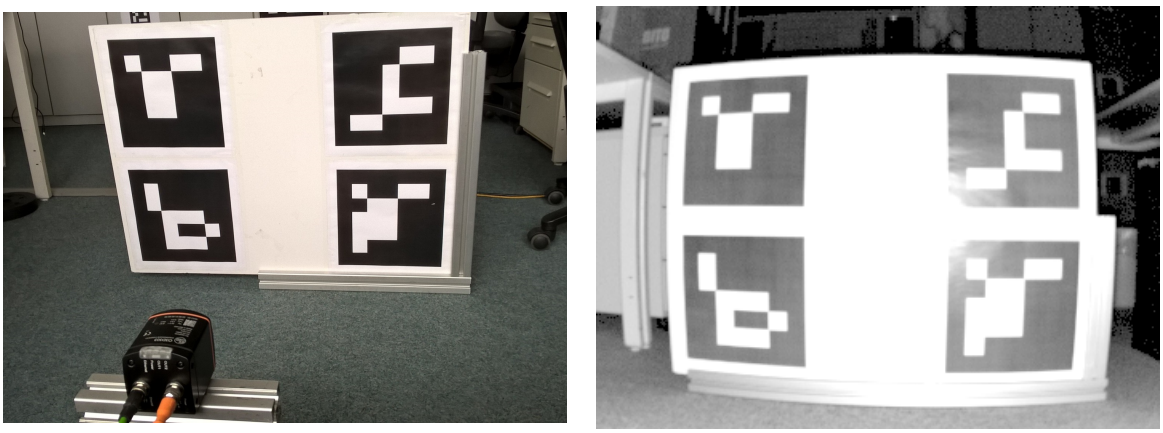


Abbildung 4.9: Zufälliger Fehler bei der Messung der Tiefenwerte in Abhängigkeit zur Intensität des Infrarotbildes.

Da nach Li [Li14] das Rauschverhalten von der Intensität des reflektierten Infrarotsignals abhängig ist, wurde in dieser Arbeit die Untersuchung des Rauschverhaltens um eine Messung in Relation zur Intensität des Signals erweitert. Dabei wurden die selben Aufnahmen genutzt, die zur Erstellung von Abbildung 4.7 verwendet wurden. Abbildung 4.9 zeigt für die Kinect v2 und die O3D303 die Standardabweichung in Abhängigkeit zur Intensität der reflektierten Infrarotstrahlung. Der Verlauf der Abweichung enspricht dem erwarteten Verlauf der in der Arbeit von Li [Li14] vorgestellten Gleichung 2.43.

4.0.4 Systematische Fehler

Der *systematische Fehler*, in anderen Arbeiten oft auch als *Wiggling Error* bezeichnet, wird dadurch verursacht, dass das emittierte amplitudenmodulierte Signal von einer perfekten Sinuswelle abweicht und die Schätzung der Phasenverschiebung daher einen Fehler in Abhängigkeit zur Distanz aufweist [GVS18]. Zur Messung des distanzabhängigen Fehlers wird eine Ground Truth Distanz benötigt, mit der die gemessene Distanz des Sensors verglichen werden kann. Giancola et al. [GVS18] verwendeten in ihrer Arbeit einen Roboterarm, der extrinsisch auf die Kamera kalibriert wurde, wodurch die Distanz zwischen Roboterarm und Sensor bekannt war. Diese Arbeit verwendet ArUco Marker zur Ermittlung der Ground Truth Distanz zwischen der Messfläche und dem Sensor. Bei ArUco handelt es sich um eine Bibliothek für *Augmented Reality* Anwendungen, die an der Córdoba Universität entwickelt wurde [RRMSMC18][SGJMSMCMC15]. Mittels ArUco ist es möglich Marker im Bild zu erkennen und die Position und Orientierung relativ zur Kamera



(a) Foto des Versuchsaufbaus

(b) Infrarotbild der O3D303

Abbildung 4.10: Versuchsaufbau zur Generierung von Ground Truth Tiefenwerten und resultierendes Infrarotbild der O3D303 während der Messung.

zu bestimmen. Wenn die ArUco Marker auf einer planaren Oberfläche angebracht werden, ist es möglich die Entfernung der Messfläche, die sich zwischen den vier ArUco Markern befindet, zur Kamera zu approximieren. Für die Aufnahme wurde die Kamera fixiert und die Messfläche orthogonal zur optischen Achse ausgerichtet, wobei die Entfernung der Messfläche zur Kamera langsam erhöht wurde. Abbildung 4.10a zeigt den Versuchsaufbau, mit dem die Daten für die O3D303 erhoben wurden.

Da die Auflösung des Bildes zur genauen Bestimmung der Position der ArUco Marker eine wichtige Rolle spielt, wurden für die Xtion 2 und die Kinect v2 die Farbbildkamera zur Bestimmung der Markerpositionen verwendet, da diese über eine höhere Auflösung verfügen. Die Tiefenbildkamera und die Farbbildkamera wurden mit Hilfe eines Schachbrettmusters extrinsisch aufeinander kalibriert und die berechneten Tiefeninformationen aus dem Farbbild wurden anschließend in den Raum des Tiefenbildes transformiert. Allerdings führt die Bestimmung der Markerposition einen eigenen Fehler mit sich, der mit zunehmender Enfernung zum Marker größer wird und sich durch einem Rauschverhalten in den Aufnahmen äußert, da die Bestimmung der Ecken der Marker ungenauer wird.

Die Abbildung 4.11 zeigt das Ergebnis der Messungen. Es ist dazu anzumerken, dass die O3D303 über keine Farbbildkamera verfügt und die Auflösung der Infrarotbildkamera mit 352 x 264 zu gering ausfällt, um eine genaue Bestimmung der ArUco Markerpositionen im Bild zu garantieren. Daher wurden drei Aufnahmen mit unterschiedlichen Markergrößen überlagert, um eine angemessene Genauigkeit zu ermöglichen, wodurch aber auch Fehler in die Messung eingeführt wurden. Die Ergebnisse der Kinect v2 decken sich mit den Messergebnissen von Giancola et al. [GVS18], die eine Überlagerung mehrerer Sinuswellen feststellten. Giancola et al. [GVS18] und Keller [Kel15] bestimmten mit Hilfe einer Fourier-Transformation zusätzlich die Phase und Amplitude der Sinuswellen, welche von Keller [Kel15] zur Simulation des Fehlers genutzt wurden, worauf im Rahmen dieser Arbeit verzichtet wird. Die Messergebnisse der O3D303 lassen darauf schließen, dass es sich bei dem Verlauf der Abweichung, ähnlich zu den Ergebnissen der Arbeit von Keller [Kel15], um nur eine Sinuswelle handeln könnte, da die O3D303 im Gegensatz zur Kinect v2 zur Generierung des Tiefenbildes nur eine Wellenfunktion mit einer Frequenz von 30 Mhz verwendet. Lediglich die Messwerte der Xtion 2 weichen von den Erwartungen ab, da keine Sinuswelle

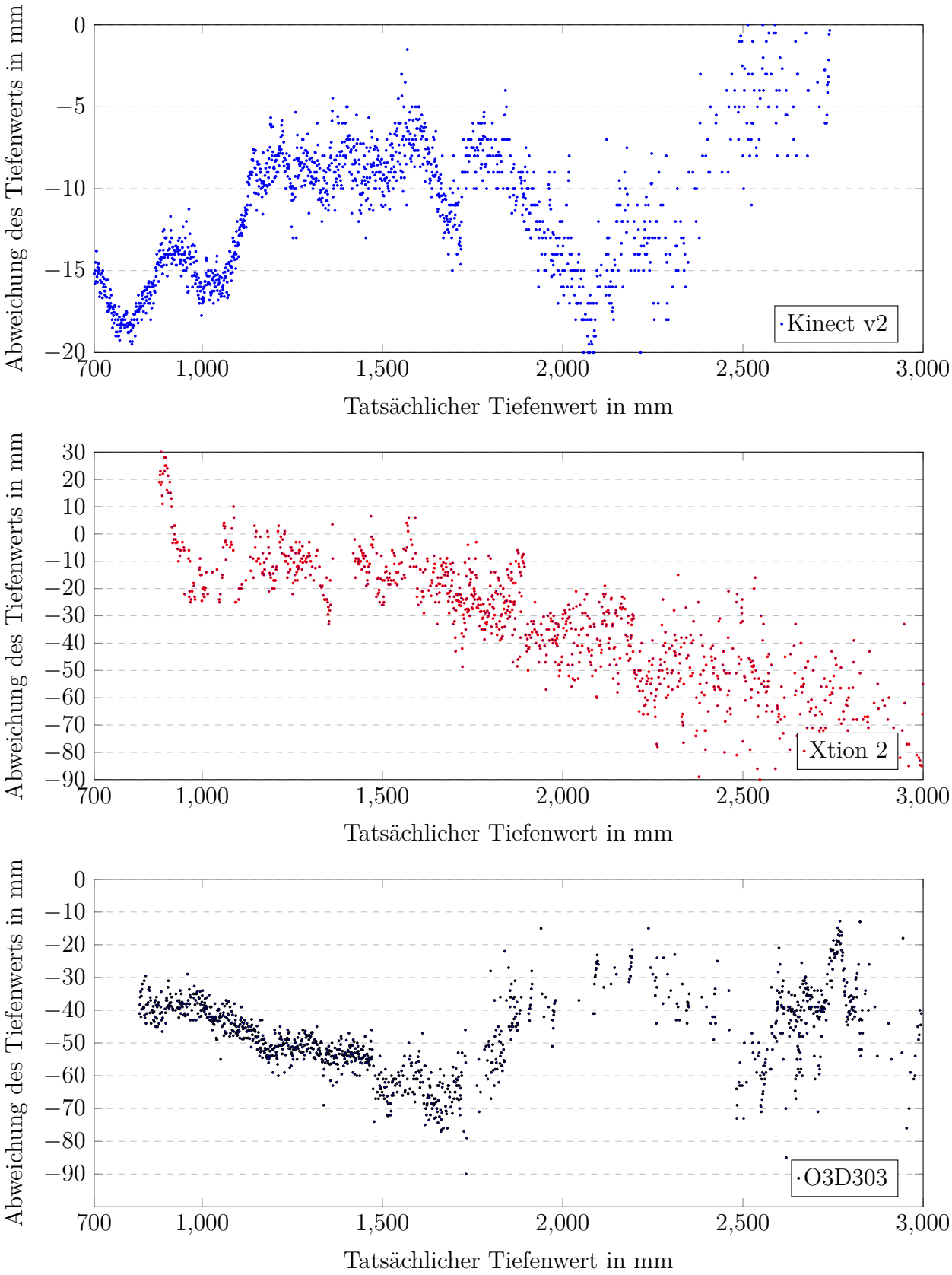
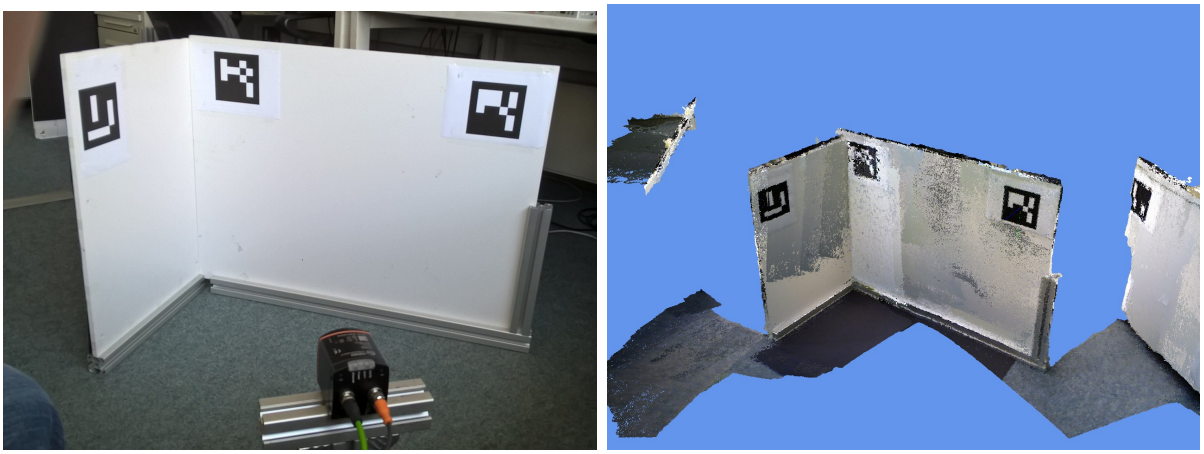


Abbildung 4.11: Systematischer Fehler bei der Messung der Tiefenwerte in Abgängigkeit zur tatsächlichen Distanz.

in den Messergebnissen ausgemacht werden kann.

4.0.5 Multiple Path Fehler durch Indirekte Beleuchtung



(a) Versuchsaufbau zur Multiple Path Fehler Messung

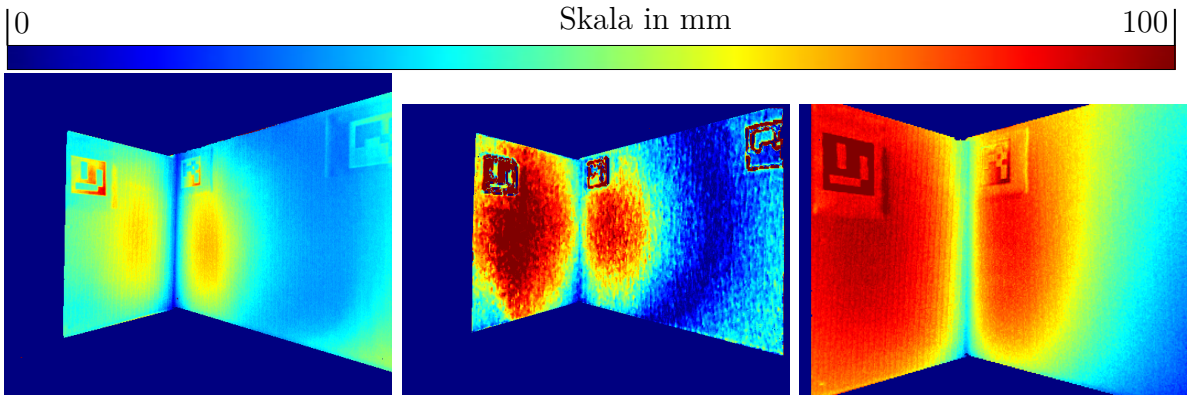
(b) Punktewolke des Versuchsaufbaus zur Bestimmung der Ground Truth Tiefenwerte

Abbildung 4.12: Versuchsaufbau mit orthogonal angeordneten Flächen zur Messung des Multipath Fehlers und die Punktewolke des Versuchsaufbaus, die als Referenz verwendet wurde.

Bei dem *Multipath Fehler* handelt es sich um einen Fehler bei der Berechnung der Tiefenwerte, der durch den Einfluss von indirekter Beleuchtung verursacht wird. Durch Reflexionen an der Oberfläche wird das emittierte Licht über Umwege zum Sensor reflektiert. Dies führt zu einer höheren Schätzung des Tiefenwerts, da indirekte Reflexionen einen längeren Weg zurückgelegt haben als direkte Reflexionen, bevor sie zum Sensor gelangen. Für die Untersuchung des Einflusses der indirekten Beleuchtung wurden für den Versuchsaufbau zwei helle Platten orthogonal zueinander vor der Kamera aufgebaut an denen jeweils ArUco Marker angebracht sind, um die Kameraposition in Relation zu den Flächen bestimmen zu können.

Um die Tiefenwerte, die vom Tiefensensor geliefert werden mit Ground Truth Tiefenwerten vergleichen zu können, wurde eine Punktewolke des Versuchsaufbaus mit einer *Structured Light* Kamera generiert. Bei einer Structured Light Kamera handelt es sich um ein Kamerasystem, das mit einem Infrarotprojektor ein bekanntes Punktmuster projiziert, welches mit einer Infrarotkamera erfasst wird, woraus Tiefenwerte generiert werden. Bei dem Verfahren handelt es sich um ein von PrimeSense entwickeltes proprietäres Verfahren, weshalb nicht im Detail auf die Funktionsweise eingegangen werden kann.

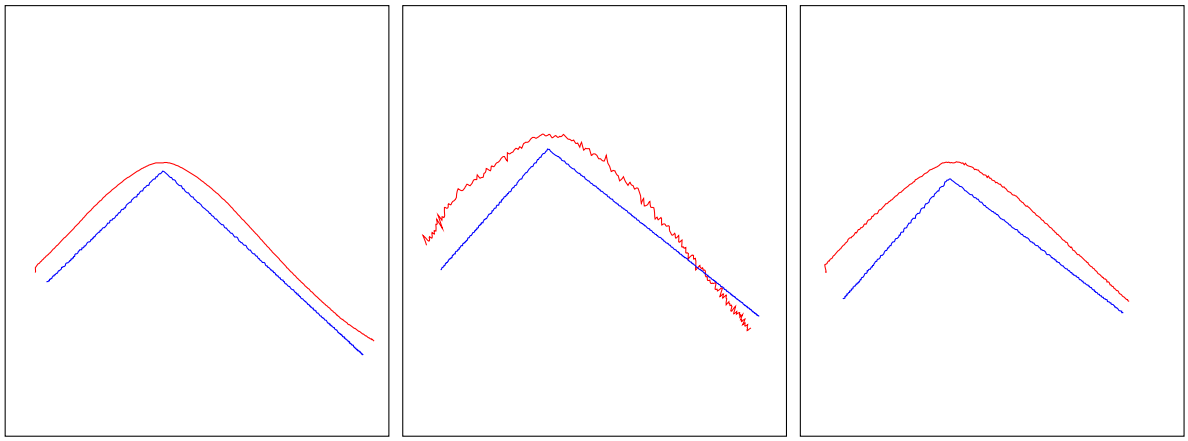
Die verwendete Xtion 1 Kamera weist keinen Multipath Fehler auf, weshalb sie zur Generierung von Ground Truth Tiefenwerten für dieses Experiment geeignet ist [WS17]. Die Xtion 1 weist zwar ebenfalls einen systematischen Fehler auf, dieser nimmt allerdings linear mit der gemessenen Entfernung zu und ist daher auf kurze Distanzen von bis zu 1000 mm ausreichend genau [GVS18][TMT13]. Für die Generierung der Punktewolke fand außerdem das von Muñoz-Salinas et al. [MSMJYBMC17] entwickelte *Marker Mapping* Verwendung. Mittels des Marker Mappings ließen sich die relativen Markerpositionen



(a) Microsoft Kinect v2 **(b)** ASUS Xtion 2 **(c)** IFM O3D303

Abbildung 4.13: Multiple Path Fehler, der durch indirekte Beleuchtung der Flächen verursacht wird.

zueinander berechnen und in Form einer Marker Map nutzen, wodurch es möglich war bei der Generierung der Punktewolke für jeden ArUco Marker eine eindeutige Position im Raum zu bestimmen. Die Xtion 1 verfügt wie die Xtion 2 und die Kinect v2 über eine Farbbild- und eine Tiefenkamera, wodurch es möglich ist das Farbbild in Kombination mit der ArUco Marker Map zu nutzen, um die Orientierung der Kamera im Raum zu bestimmen und die gemessenen Tiefenwerte der Kamera entsprechend in den Raum zu transformieren. Die Abbildung 4.12 zeigt den Versuchsaufbau mit der O3D303 und die generierte Punktewolke, die zur Referenz verwendet wurde.



(a) Kinect v2 **(b)** Xtion 2 **(c)** O3D303

Abbildung 4.14: Top-Down Darstellung der Tiefenwerte aus dem mittleren horizontalen Schnitt des Bildes. Die blauen Tiefenwerte stehen für die Ground Truth Distanz und die roten Tiefenwerte repräsentieren die Messwerte der einzelnen Kameras.

Für jede Kamera wurden 10000 Tiefenbilder der statischen Szene aufgenommen und für jeden Pixel wurde der durchschnittliche Tiefenwert gebildet, um den zufälligen Fehler, der in Unterabschnitt 4.0.3 erläutert wurde, zu reduzieren. Mit Hilfe der ArUco Marker wurde die Punktewolke so transformiert, dass sie mit den realen Positionen des Versuchsaufbaus relativ zur Kamera übereinstimmt. Dadurch konnte eine Abweichung der gemessenen

Tiefenwerte zu den erwarteten Tiefenwerten festgestellt werden. Abbildung 4.13 veranschaulicht das Ergebnis des Versuchs in Form einer Color Map. Dabei ist zu erkennen, dass sich die Einflüsse des Multipath Fehlers für jede Kamera im Detail unterscheiden, aber gemeinsam haben, dass Ecken im Bild durch den Einfluss von Interreflexionen abgerundet werden. Abbildung 4.14 stellt dieses Verhalten noch deutlicher dar, indem die Tiefenwerte der Punktewolke und der Time-of-Flight Kameras in der Mitte des Bildes entlang einer horizontalen Scanlinie verglichen werden.

4.0.6 Lens Scattering Fehler

(a) Versuchsaufbau zur Messung des Lens Scattering Fehlers

(b) Punktewolke des Versuchsaufbaus mit resultierendem Lens Scattering Fehler

Abbildung 4.15: Versuchsaufbau zur Evaluierung des Lens Scattering Fehlers. Zu erkennen ist der Fehler an den hell eingefärbten Pixeln im Hintergrund, der direkt am Übergang stark ausgeprägt ist und mit größerer Entfernung zum Vordergrund abnimmt.

Der *Lens Scattering Fehler* beschreibt einen aus der Literatur bekannten Fehler im Tiefenbild, der dadurch verursacht wird, dass das einfallende Licht einen Einfluss auf benachbarte Pixel ausübt [HF14][JL10][CCMDH07][CCMDH09]. Zusätzlich zur Beugung durch die Blende werden Reflexionen und Streuungseffekte innerhalb des Linsensystems als Grund für verfälschte Tiefenwerte benachbarter Pixel genannt. Dieser auftretende Effekt verursacht, dass stark beleuchtete Objekte im Vordergrund die Tiefenwerte der Objekte beeinflussen, die sich im Hintergrund befinden und schwächer beleuchtet werden. Hertzberg und Frese [HF14] untersuchten den Lens Scattering Effekt mit Hilfe eines runden Retroreflektors der vor einen schwach reflektierenden Hintergrund positioniert wurde. Dazu wurde zunächst eine Aufnahme des Hintergrundes ohne Reflektor erstellt und anschließend die selbe Szene mit Reflektor aufgenommen. Dadurch konnte der Einfluss des Retroreflektors auf den Hintergrund bestimmt werden. In dieser Arbeit wird von der Verwendung eines Retroreflektors abgesehen, da dieser neben dem Lens Scattering Effekt bei allen Sensoren zusätzlich *Blendenflecke* oder *Linsenreflexionen* verursacht hat. Zusätzlich führte die Verwendung des Retroreflektors dazu, dass die Intensität des Infrarotbildes des Reflektors gesättigt war und so die ursprüngliche Intensität der reflektierten Strahlung nicht mehr ermittelt werden konnte, was die Aufnahme für die Evaluation des Lens Scattering Effekts unbrauchbar macht.

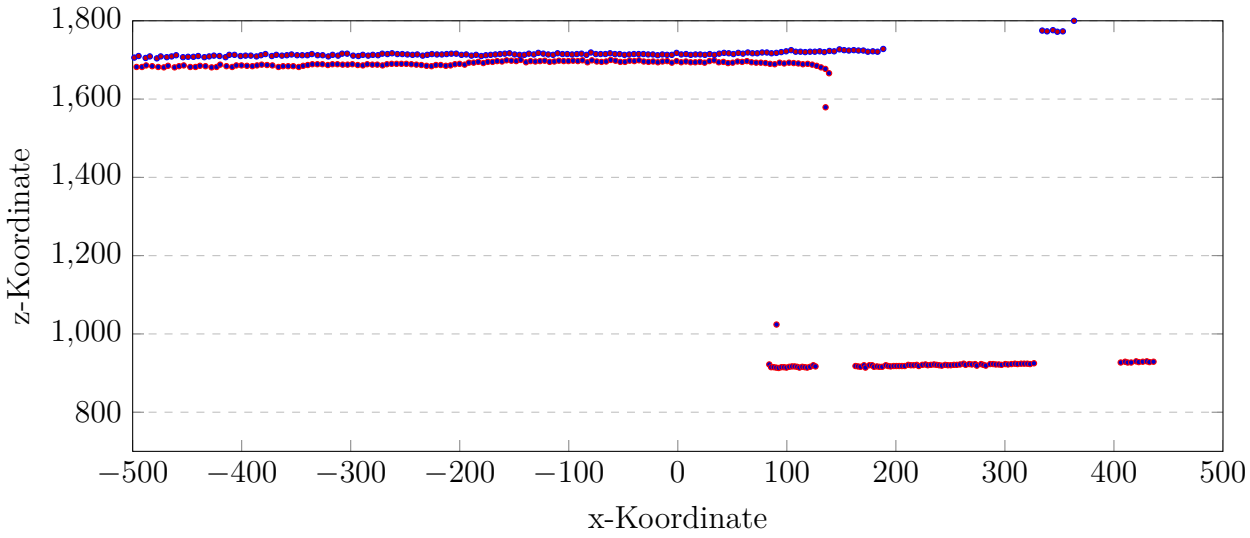
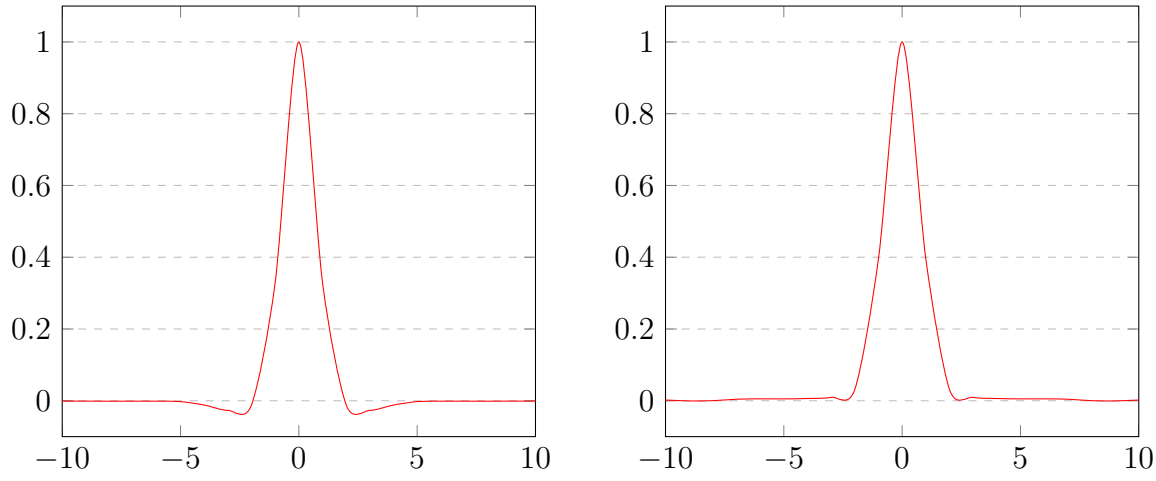


Abbildung 4.16: Top-Down Darstellung der Tiefenwerte aus dem mittleren horizontalen Schnitt des Bildes.

Jamtsho und Lichti [JL10] verwendeten zur Evaluation hingegen zwei stark reflektierende Flächen in unterschiedlichen Distanzen, die orthogonal zur optischen Achse des Sensors positioniert wurden. Dabei wurde der Einfluss der Tiefenwerte der Fläche, die sich näher am Sensor befindet, auf die Tiefenwerte der Fläche im Hintergrund beobachtet. Diese Arbeit verfolgt einen ähnlichen Ansatz und verwendet eine schwach reflektierende Fläche aus schwarzem Tonpapier im Hintergrund und eine stark reflektierende Fläche im Vordergrund. Dabei wird die Distanz der stark reflektierenden Fläche zur Kamera variiert und der Einfluss auf den Hintergrund beobachtet. Abbildung 4.15 veranschaulicht den Versuchsaufbau und die resultierende Punktewolke der Messung, die im Folgenden im Detail untersucht wird.



(a) IFM O3D303 **(b)** Microsoft Kinect v2

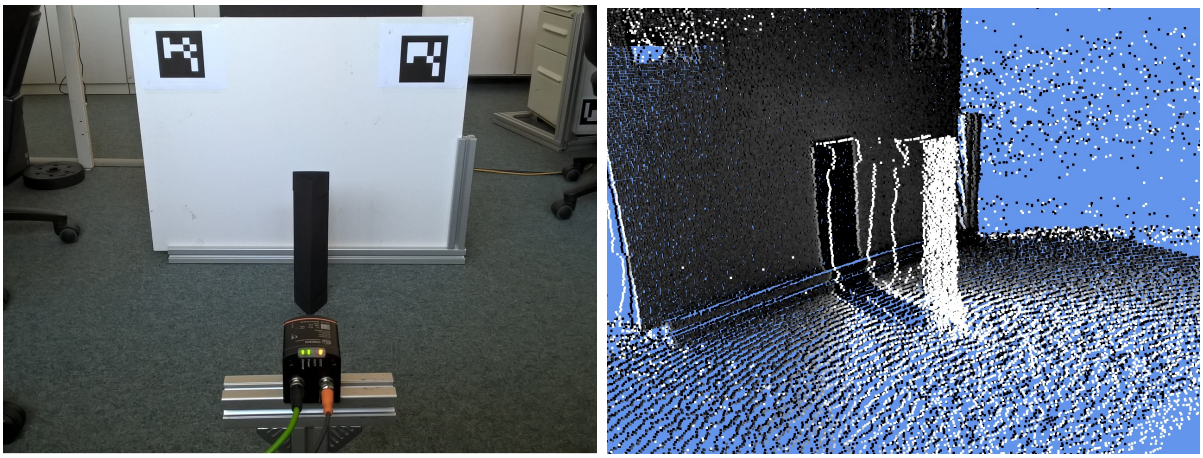
Abbildung 4.17: Lens Scattering Fehler in Abhängigkeit zur Intensität des Infrarotbildes.

In Abbildung 4.16 werden die Tiefenwerte aus der horizontalen Scanlinie dargestellt. Bei den blauen Tiefenwerten handelt es sich um die Referenzaufnahme des Hintergrundes. Die

roten Tiefenwerte wurden einer Aufnahme entnommen, in der ein zusätzliches Objekt im Vordergrund des Bildes positioniert wurde. Der stark reflektierende Vordergrund beeinflusst dabei die Tiefenwerte der gesamten restlichen Aufnahme. An dem Übergang vom Vordergrund zum Hintergrund wirkt sich der Lens Scattering Fehler besonders auf die Tiefenwerte des Hintergrundes aus. Diese Beobachtungen decken sich ebenfalls mit denen von Hertzberg und Frese [HF14] und Jamtsho und Lichti [JL10].

Abbildung 4.17 zeigt das Ergebnis der Untersuchung des Lens Scattering Effekts. Die Graphen zeigen den Einfluss des Pixels auf seine Nachbarpixel in Abhängigkeit zur Intensität in Form einer Punktspreizfunktion. Diese gibt an, wie stark sich die Intensität eines Pixels auf seine Nachbarpixel verteilt. Die Ergebnisse weichen von denen der Untersuchungen durch Hertzberg und Frese [HF14] und Jamtsho und Lichti [JL10] ab. Während Hertzberg und Frese einen ausschließlich positiven Verlauf der Funktion ermittelt haben, weist hier die O3D303 ebenfalls negative Werte auf (siehe Abbildung 4.17a). Jamtsho und Lichti näherten die Lens Scattering Funktion mittels einer Summe von Gaußfunktionen an, was für die Ergebnisse der O3D303 und Kinect v2 nicht möglich ist, da die Funktion im positiven Bereich nicht monoton fallend und im negativen Bereich nicht monoton steigend ist. Der Funktionsverlauf erinnert dabei an den Verlauf von *Beugungsscheibchen* oder auch *Airy-Scheibchen* genannt, die durch die Beugung eines Lichtstrahls an einer Blende entstehen [Pad09], weshalb ein Zusammenhang vermutet wird.

4.0.7 Mixed Pixels Fehler



(a) Versuchsaufbau zur Mixed Pixels Fehler Messung

(b) Punktewolke des Versuchsaufbaus mit resultierendem Mixed Pixels Fehler

Abbildung 4.18: Versuchsaufbau zur Evaluierung des Mixed Pixels Fehler. Zu erkennen ist der Fehler an den fliegenden Punkten zwischen Vordergrundobjekt und Hintergrund.

Der *Mixed Pixels* Fehler, häufig auch *Flying Pixels* Fehler genannt, beschreibt einen aus der Literatur bekannten Fehler im Tiefenbild, der an direkt benachbarten Tiefenwerten zu beobachten ist und der dadurch verursacht wird, dass sich ein Tiefenwert für einen Pixel aus mehreren gemessenen Tiefen zusammensetzt [GVS18][TMT13][But14][HF14]. Dieser Fehler ist besonders in Bereichen mit stark variierenden Tiefenwerten zu beobachten. Da der Einfluss des Lens Scattering für die Evaluation des Mixed Pixels Fehlers

ausgeschlossen werden muss, wird der Fehler zunächst isoliert, indem der Einfluss des Lens Scattering zunächst entfernt wird. Zur Evaluation des Mixed Pixels Fehlers wurde daher ein schwach reflektierendes Objekt vor einem stark reflektierenden Hintergrund positioniert und der Einfluss der schwach reflektierenden Oberfläche auf den Hintergrund wurde dabei beobachtet. Der Einfluss des Lens Scattering Effekts ist für diese Beobachtung nicht vollständig auszuschließen, allerdings sollte der Vordergrund einen verhältnismäßig geringen Einfluss auf den Hintergrund ausüben. Auf diese Weise kann der Mixed Pixels Fehler für den Hintergrund isoliert vom Lens Scattering Fehler betrachtet werden.

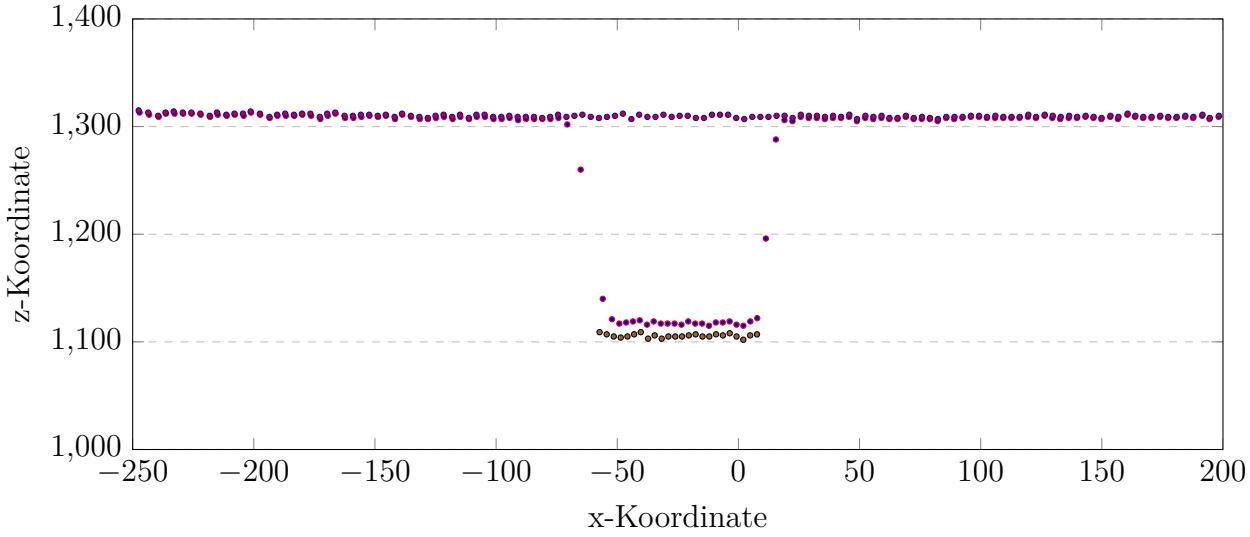


Abbildung 4.19: Top-Down Darstellung der Tiefenwerte aus dem mittleren horizontalen Schnitt des Bildes.

Zur Evaluation wurden mit der O3D303 drei separate Aufnahmen angefertigt. Zunächst wurde nur der Hintergrund aufgenommen, um den Einfluss des Vordergrundobjektes auf den Hintergrund zu evaluieren. Dabei wurden 10.000 Tiefenbilder erstellt und für jeden Pixel der Durchschnitt des Tiefenwertes gebildet, um den zufälligen Fehler zu eliminieren. Anschließend wurde ein Objekt vor dem Hintergrund platziert und auf dieselbe Weise eine weitere Aufnahme erstellt. Abschließend wurde der Hintergrund wieder entfernt und eine Aufnahme vom Vordergrundobjekt gemacht, um den Einfluss des Hintergrundes auf den Vordergrund zu evaluieren. In Abbildung 4.19 werden die aus den Aufnahmen resultierenden Tiefenwerte aus der horizontalen Scanlinie des Tiefenbildes dargestellt. Die blauen Tiefenwerte stehen für den separat aufgenommenen Hintergrund, die schwarzen Tiefenwerte sind durch eine separate Aufnahme des Vordergrundobjektes entstanden und die roten Tiefenwerte bilden die gemeinsame Aufnahme von Vordergrund und Hintergrund. Dabei ist zu erkennen, dass das Vordergrundobjekt die Tiefenwerte des Hintergrundes kaum beeinflusst, auf der anderen Seite werden aber die Tiefenwerte des Vordergrundobjektes durch den Hintergrund verfälscht. Um den Einfluss des in Unterabschnitt 4.0.6 betrachteten Lens Scattering auszuschließen, wurden daher alle Tiefenwerte, die zum Vordergrund gehören maskiert.

Die Abbildung 4.20 zeigt die gemeinsame Aufnahme des Vordergrundobjektes und des Hintergrundes, wobei die Tiefenwerte der gemeinsamen Aufnahme so modifiziert wurden, dass die Punkte entfernt wurden, die durch die separate Aufnahme des Vordergrundobjektes als Teil des Vordergrundes identifiziert wurden. Dabei ist zu erkennen, dass der

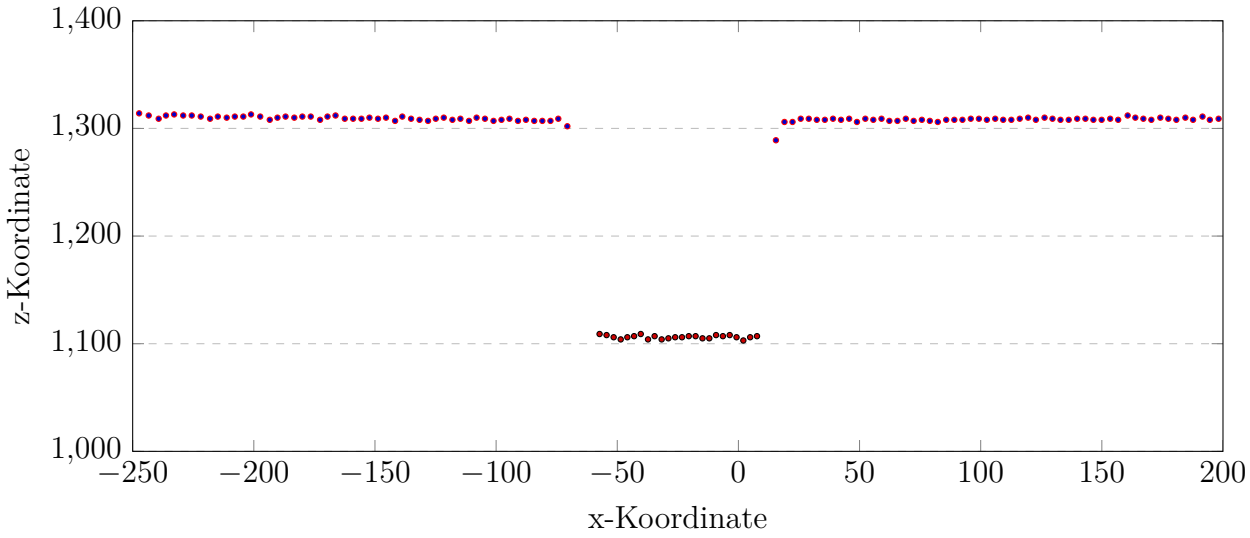
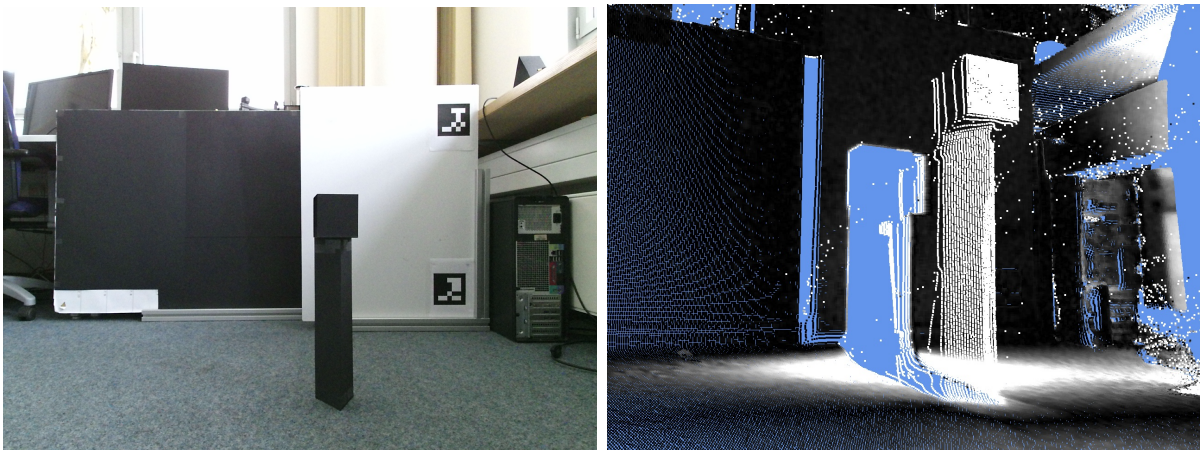


Abbildung 4.20: Top-Down Darstellung der Tiefenwerte aus dem mittleren horizontalen Schnitt des Bildes. Dabei wurden alle Tiefenwerte aus der roten Punktewolke entfernt, von denen sicher ist, dass diese zum Vordergrundobjekt gehören.

Mixed Pixels Fehler fast vollständig verschwunden ist. Daher liegt die Vermutung nahe, dass der aus der Literatur bekannte Mixed Pixels Fehler allein auf den Lens Scattering Effekt zurückzuführen ist, da der Fehler bei dem Versuch der isolierten Betrachtung verschwindet.

4.0.8 Positionen der Infrarot LEDs



(a) Versuchsaufbau zur Messung des Fehlers durch Schattenwurf im Infrarotbild

(b) Punktewolke des Versuchsaufbaus mit resultierendem Fehler

Abbildung 4.21: Versuchsaufbau zur Evaluierung des Fehlers durch Schatten. Zu erkennen ist der Fehler an den weißen Punkten im Hintergrund, dessen Tiefe zu hoch geschätzt wurde.

Bei der Analyse des Mixed Pixels Fehlers ist aufgefallen, dass im Falle der Kinect v2 und der Xtion 2 ein Teil des Bildes im Schatten liegt und nicht direkt von den Infrarot

LEDs bestrahlt wurde, wodurch Artefakte im Bild entstanden sind. Diese treten bei der O3D303 nicht auf, da der Sensor von vier LEDs umgeben ist. Die LEDs der Kinect v2 und der Xtion 2 sind einseitig neben dem Sensor positioniert, wodurch sich Schatten bilden können und Bildbereiche ausschließlich indirekt beleuchtet werden, was zu einer höheren Schätzung der Distanz führt.

Zur Untersuchung des Fehlers wurde zunächst eine Referenzaufnahme eines Hintergrundes angefertigt. Anschließend wurde ein Objekt im Vordergrund positioniert und eine zweite Aufnahme angefertigt. Anschließend wurden die Aufnahmen miteinander verglichen und die Abweichung vom Referenzbild ermittelt. Abbildung 4.21 zeigt den Versuchsaufbau und die resultierende Punktewolke. Dabei werden die Abweichungen vom Referenzwert über den Grauwert dargestellt. Eine helle Farbe bedeutet in diesem Fall, dass eine Abweichung vorhanden ist, während schwarz dafür steht, dass der Tiefenwert mit der Referenzaufnahme übereinstimmt.

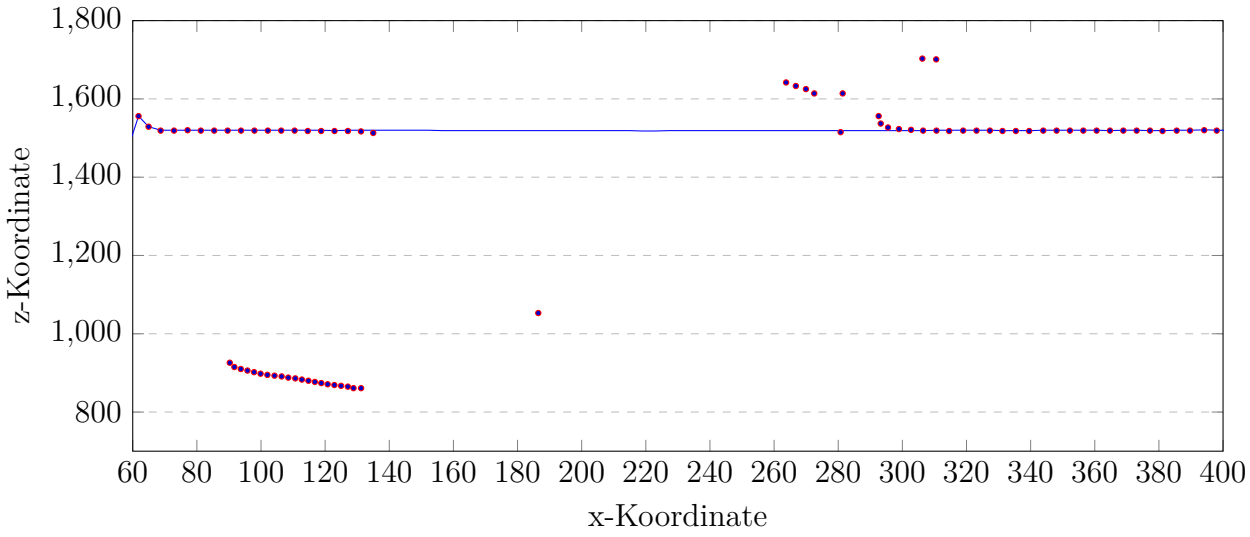


Abbildung 4.22: Top-Down Darstellung der Tiefenwerte aus dem mittleren horizontalen Schnitt des Bildes.

Abbildung 4.22 veranschaulicht mittels der Koordinaten entlang der horizontalen Schnittlinie den Fehler im Tiefenbild, der durch den Schatten eines Objektes verursacht wird. Dabei zeigt die blaue Linie den Verlauf des Hintergrundes und die roten Punkte die Koordinaten der Aufnahme mit einem Objekt, das vor diesen Hintergrund platziert wurde. Die Punkte mit den X-Koordinaten im Bereich zwischen 86 und 131 sind die Koordinaten des Objektes, das im Bild platziert wurde und den Schatten wirft. An der X-Koordinate 186 ist ein Flying Pixel zu erkennen, der durch den Lens Scattering Fehler verursacht wurde. Bei den Punkten ab einer X-Koordinate von 263 sind deutliche Abweichungen von den Ground Truth Werten zu erkennen, die dadurch verursacht werden, dass der Bereich im Schatten liegt und ausschließlich indirekt beleuchtet wird. Hertzberg und Frese [HF14] wiesen zwar darauf hin, dass durch die Position der LEDs das Tiefenbild beeinflusst werden kann, allerdings wurde dieser Einfluss in der Literatur bisher weder untersucht, noch simuliert.

5 Simulation

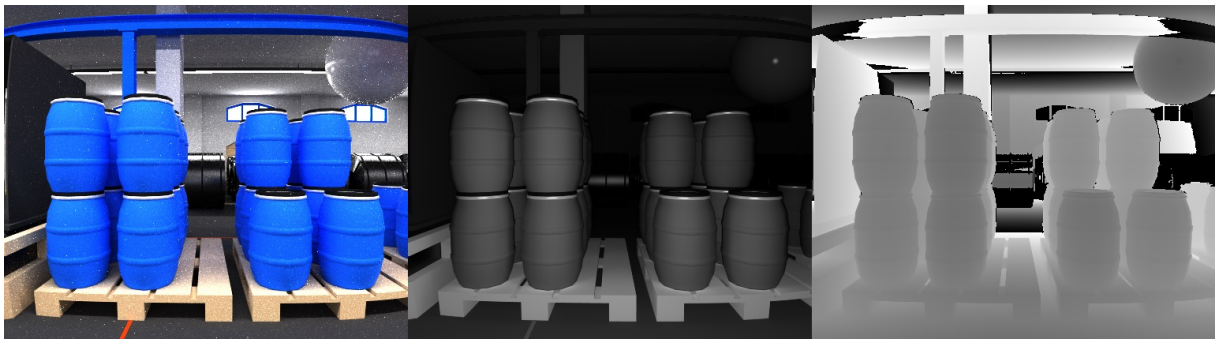


Abbildung 5.1: Screenshots der Simulation mit einem Farbbild, einem Infrarotbild und einem Tiefenbild.

Dieses Kapitel beschäftigt sich mit der Erläuterung der Time-of-Flight Sensor Simulation mittels Path Tracing und der Generierung der Tiefenbilder. Die Simulation lässt sich grob in zwei Phasen unterteilen. Zunächst wird die Simulation des Time-of-Flight Sensors und der Lichtausbreitung auf der GPU mittels Path Tracing durchgeführt. Das Ergebnis der Simulation sind die einzelnen Phasenbilder, die in Abschnitt 2.2 erläutert wurden. Mit Hilfe dieser Phasenbilder werden anschließend auf der CPU die Tiefenwerte unter Berücksichtigung des Lens Scattering und des zufälligen Fehlers berechnet.

In den folgenden Unterkapiteln werden daher zunächst der Path Tracing Algorithmus im Detail erläutert und im Anschluss die Berechnung der Tiefenwerte behandelt.

5.1 Path Tracing

Bei dem Path Tacing Algorithmus handelt es sich um eine Implementierung nach Kajiya [Kaj86]. Dabei werden für jeden Pixel des Bildes Strahlen in die Szene geschickt. Anschließend wird bestimmt, ob der Strahl mit der Szene kollidiert. Falls dieser Strahl mit einer Oberfläche kollidiert, wird ermittelt, ob dieser Punkt von Lichtquellen beleuchtet wird und die anteilige Strahlung bestimmt, die in die Richtung des Betrachters reflektiert wird. Für die indirekte Beleuchtung durch andere Oberflächen wird ein neuer Strahl in eine zufällige Richtung bestimmt, der seinen Ursprung im Schnittpunkt hat. Es wird nun wiederholt ermittelt, ob dieser neue Strahl mit der Szene kollidiert und die Reflexion in Richtung des Ursprungs berechnet. Dies wird solange wiederholt, bis eine maximale Länge für den Pfad erreicht wurde, oder der Strahl die Szene verlässt. Abbildung 5.2 veranschaulicht die Funktionsweise des Algorithmus für mehrere Sichtstrahlen, die für denselben Pixel berechnet werden. Dabei verlässt jeder Strahl den Betrachter in dieselbe Richtung und

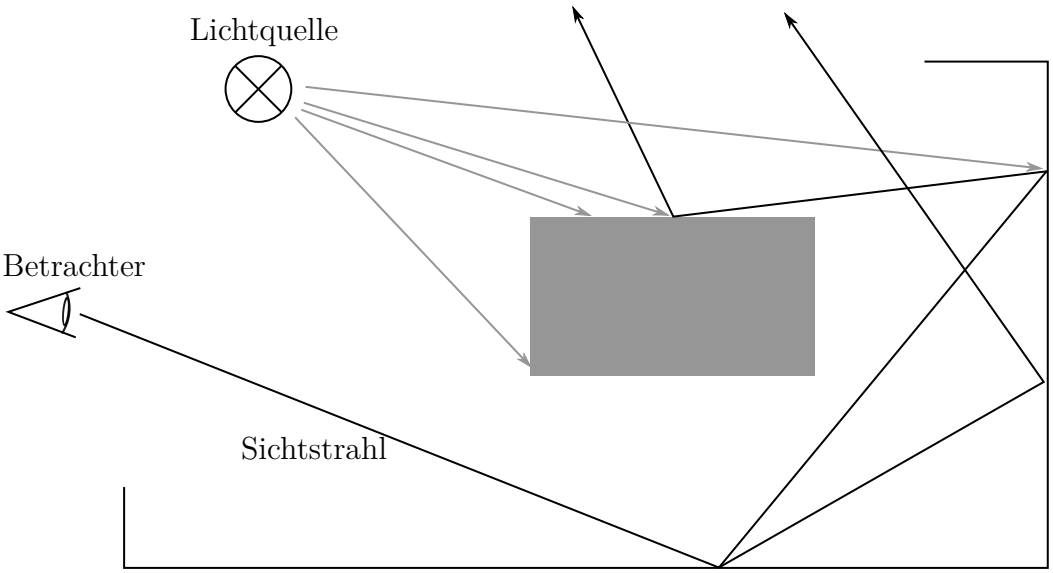


Abbildung 5.2: Schematische Darstellung der des Path Tracing Algorithmus mit zwei Pfaden für den selben Pixel.

erhält für jeden Schnittpunkt jeweils eine zufällige neue Richtung. Für jeden Schnittpunkt wird dabei berechnet, ob dieser direkt durch eine oder mehrere Lichtquellen beleuchtet wird.

Der Path Tracing Algorithmus wurde unter Verwendung der NVIDIA OptiX Ray Tracing Engine entwickelt. Dabei handelt es sich um ein Framework, das eine API zur Erstellung von Ray Tracing Applikationen anbietet [PRS+10]. Daher wird neben der Installation von OptiX auch die des CUDA Toolkits benötigt und zur Verwendung dessen ist eine Grafikkarte notwendig, die CUDA unterstützt. Bei CUDA handelt es sich um eine API, die von NVIDIA entwickelt wurde und es dem Nutzer ermöglicht die Ausführung von Programmteilen mittels *General-Purpose Computing on Graphics Processing Units* (GPGPU) auf der Grafikkarte durchführen zu lassen. Dabei zeichnet sich die Berechnung auf der Grafikkarte durch die hochgradig parallele Abarbeitung aus. Im Falle des Path Tracings kann für die Berechnung von jedem Strahl ein eigener Thread genutzt werden und so kann die Programmausführung für tausende von Strahlen gleichzeitig durchgeführt werden.

Das OptiX Framework bietet Schnittstellen an, in denen Teile der Funktionalität mittels CUDA C Kernels definiert werden können, die im Folgenden erläutert werden. Die Implementierung eines Path Tracing Algorithmus mittels OptiX involviert dabei folgende Schritte:

1. die Implementierung eines *Ray Generation* Programms, das definiert, wie die Strahlen in die Szene geschossen werden
2. die Definition eines *Bounding Box* Programms, welches zur Beschleunigung des Algorithmus benötigt wird und einen groben Test beschreibt und Feedback darüber gibt, ob der Strahl sich in der Nähe des Objektes befindet, bevor eine genaue und rechenintensive Schnittberechnung mit der Oberfläche des Objektes durchgeführt wird

3. die Implementierung eines *Intersection* Programms, das die genaue Schnittberechnung des Strahl mit der Oberfläche durchführt
4. die Definition eines *Any Hit* und eines *Closest Hit* Programms für jeden Oberflächentyp. Diese beschreiben das Verhalten des Strahls, bei Kollision mit der Oberfläche. Für jedes Objekt kann ein eigenes Closest Hit Programm definiert werden, das die spezifische BRDF des Objektes beschreibt. Das Any Hit Programm wird hier zur Berechnung der Schatten benötigt und beschreibt die transparenten Eigenschaften des Objektes und definiert, welcher Teil der Strahlung hindurchscheint, falls sich zwischen dem Schnittpunkt und der Strahlungsquelle ein Objekt befindet
5. die Implementierung des *Ray Missing* Programms, das das Verhalten definiert, falls der Strahl die Szene verlässt und es keinen Schnittpunkt mehr geben kann

Das Bounding Box Programm und die Intersection wurden dem Beispiel der Implementierung eines Ray Tracers des OptiX SDKs entnommen, weshalb hier darauf nicht im Detail eingegangen wird. Die Erläuterung des Any Hit Programms wird ebenfalls ausgelassen, da in dieser Arbeit angenommen wird, dass alle Oberflächen vollständig opak sind und die Implementierung daher trivial ist.

Im Kontrast zur Implementierung durch Meister et al. [MNK13] wird in dieser Arbeit ein unidirektionaler Path Tracer verwendet, da sich die Lichtquelle in unmittelbarer Nähe zum Sensor befindet und das gerenderte Bild fast vollständig direkt beleuchtet wird. Ein bidirektionaler Path Tracer spielt seine Stärken besonders in komplexen Lichtsituationen aus, in denen die Szene größtenteils indirekt beleuchtet wird, weshalb im Rahmen dieser Arbeit auf die zusätzliche Komplexität verzichtet wird, die mit der Berechnung der zusätzlichen Strahlen, die ihren Ursprung bei der Lichtquelle haben, einhergeht.

Bei der Implementierung des Path Tracings wird vereinfachend angenommen, dass die Photonen durchs Vakuum reisen und keiner atmosphärischen Streuung ausgesetzt sind. Außerdem wird für die Berechnung der Fresnel Reflexion die optische Dichte der Luft auf Meeresebene bei Raumtemperatur angenommen.

5.1.1 Ray Generation Programm

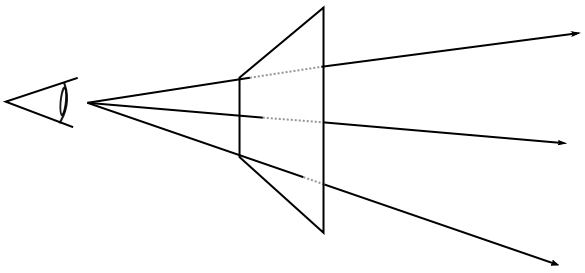
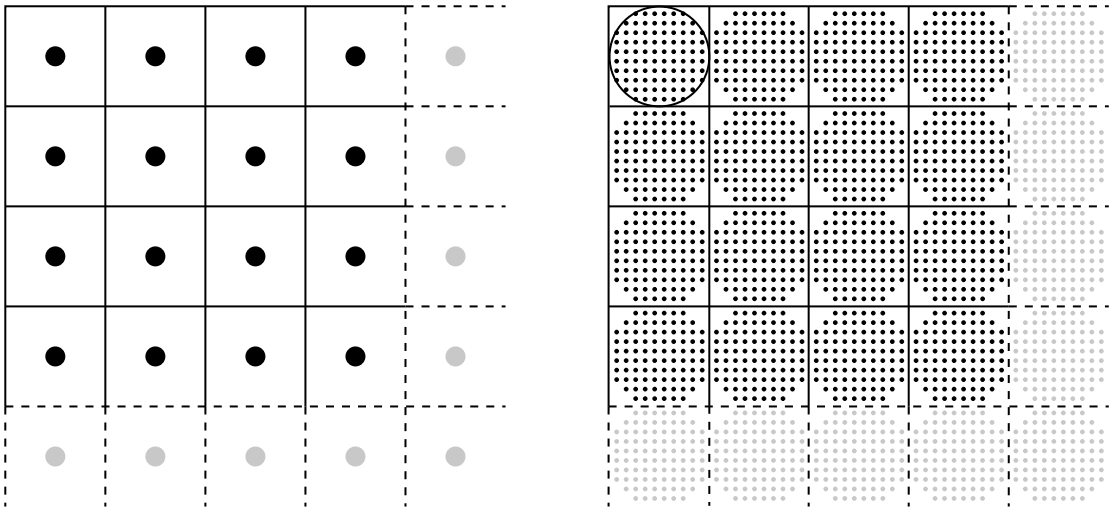


Abbildung 5.3: Schematische Darstellung der Strahlen, die für jeden Pixel im Bild berechnet werden.

Das Ray Generation Programm bildet den Startpunkt des Path Tracings Algorithmus. Dieses Programm generiert die Strahlen, die in die Szene geschossen werden und summiert

die Teilergebnisse der einzelnen Strahlen zu einem Endergebnis auf. Bei der Implementierung in dieser Arbeit handelt es sich um einen Progressive Path Tracing Algorithmus, der für jedes gerenderte Bild eine Anzahl an Strahlen generiert und dem Nutzer das Zwischenergebnis präsentiert wird, während auf der Grafikkarte bereits die nächsten Strahlen generiert werden und das Bild mit fortschreitender Berechnungszeit immer weiter gegen das Endergebnis konvergiert. Sobald sich die Szene ändert und z. B. die Kamera bewegt wird, wird das Bild komplett neu berechnet.

Der Algorithmus verschickt, wie in Abbildung 5.3 dargestellt, für jeden Pixel ununterbrochen neue Strahlen in die Szene. Nach einer bestimmten Anzahl an Strahlen pro Pixel, die im Folgenden als *Samples* bezeichnet werden, unterbricht das Programm den Prozess, um das Zwischenergebnis in einen Speicherbereich (hier als *Buffer* bezeichnet) zu schreiben, der dem Nutzer im laufenden Betrieb präsentiert wird. Eines dieser Zwischenergebnisse wird im Folgenden als *Frame* bezeichnet. Anschließend wird die Bearbeitung des nächsten Frames angestoßen und das Ergebnis um weitere Samples erweitert.



(a) Vorberechneter Buffer mit einem Strahl pro Pixel

(b) Vorberechneter Buffer in zehnfacher Auflösung mit 100 Strahlen pro Pixel

Abbildung 5.4: Gegenüberstellung des Buffers mit einem Strahl pro Pixel und einer Sammlung von 100 Strahlen für jedes Pixel, von denen für jedes Sample ein zufälliger Strahl innerhalb eines Radius gewählt wird. Jeder Punkt im Bild repräsentiert eine Koordinate (u, v), aus dem ein Strahl $\vec{d}$ berechnet wird.

Bei der Berechnung der Strahlen für den Pixel wird bereits die Verzeichnung durch die Linse berücksichtigt, die in Unterabschnitt 4.0.1 erläutert wurde. Dazu wurde ein Buffer mit entzerrten Strahlen vorberechnet und in den Speicher der Grafikkarte kopiert. Dafür wurde für jede Pixelkoordinate (u, v) eine entzerrte Pixelkoordinate $(u^{'}, v^{'})$ bestimmt und daraus der entsprechende Strahl berechnet. Dazu wurde zunächst durch Verwendung der ArUco Bibliothek aus der Kameramatrix, die durch die Kalibrierung mittels Schachbrett ermittelt wurde, eine Projektionsmatrix P berechnet, mit deren Hilfe für einen 3D Punkt im Raum die 2D Pixelkoordinate auf dem Bildschirm bestimmt werden kann. Zur Berechnung der Strahlen wird die Projektionsmatrix invertiert, um für die 2D Pixelkoordinaten

auf dem Bildschirm einen 3D Punkt im Raum zu berechnen, der sich an einer beliebigen Stelle auf dem Strahl befindet. Dieser Punkt wird anschließend normiert, um eine Richtung zu erhalten:

$$\begin{aligned} \langle p_x, p_y, p_z, p_w \rangle &= P^{-1} \cdot \left\langle u^{'}, v^{'}, 1, 1 \right\rangle \\ \vec{d} = \langle d_x, d_y, d_z \rangle &= \frac{\langle p_x, p_y, p_z \rangle}{|\langle p_x, p_y, p_z \rangle|}. \end{aligned} \tag{5.1}$$

Da das Verwenden eines einzelnen Strahls pro Pixel zur Bildung von Aliasing Effekten an Kanten führen würde, wird in Ray Tracing Anwendungen häufig für jedes Sample ein zufälliger Strahl generiert, der sich innerhalb des Pixels befindet, um ein ruhigeres Bild zu erzeugen. Da die Blende nicht unendlich klein ist, wird auch in dieser Arbeit nicht nur ein Strahl pro Pixel verwendet, sondern die Öffnung der Blende angenähert, indem für jedes Sample ein zufälliger Punkt $(u^{'}, v^{'})$ innerhalb eines Radius gewählt wird. Da die Berechnung einer entzerrten Pixelkoordinate für zufällig generierte Pixelkoordinaten (u, v) auf der Grafikkarte zu aufwendig wäre, wird ein Buffer in zehnfacher Auflösung vorbereitet und für jeden Pixel werden 100 Strahlen vorberechnet. Für jedes Sample wird einer dieser vorberechneten Strahlen nach dem Zufallsprinzip gewählt. In Abbildung 5.1.1 ist dieses Prinzip veranschaulicht, indem die Pixelkoordinaten, die zur Berechnung des Strahls verwendet werden, dargestellt werden.

Der folgende vereinfachte Codeauszug beschreibt den Prozess, der im Ray Generation Programm abläuft:

```
RT_PROGRAM void pathtrace_camera() {
    // Pro Durchgang werden fuer jeden Pixel eine bestimmte Anzahl an Strahlen generiert
    float3 result = make_float3(0.0f);
    unsigned int samples_per_pixel = sqrt_num_samples * sqrt_num_samples;
    do {
        // Berechne Pixelkoordinate, fuer den der Strahl berechnet werden soll
        float2 relativeCoordinate = calculatePixel(launch_index);

        // Berechne Strahl fuer den Pixel
        float3 calculated_ray = calculate_ray_for_pixel(rayDirectionsBuffer, relativeCoordinate);

        // Transformiere den Strahl aus dem Kamera-Koordinatensystem ins Welt-Koordinatensystem
        float3 ray_origin = eye_position;
        float3 ray_direction = normalize(calculated_ray.x*U + calculated_ray.y*V + -calculated_ray.z*W);

        // Jede Iteration berechnet ein Segment des Strahls
        PerRayData_radiance prd;
        for(;;) {
            Ray ray = make_Ray(ray_origin, ray_direction, radiance_ray_type, scene_epsilon, RT_DEFAULT_MAX);

            // An dieser Stelle wird ein Closest Hit oder ein Miss Programm aufgerufen und das Ergenis in die Variable prd geschrieben.
            rtTrace(top_object, ray, prd);

            prd.depth++;
            prd.result += prd.radiance;

            // Wenn eine Terminierungsbedingung erfuellt wurde, wird die Verfolgung des Strahls beendet
            if(prd.done || prd.depth >= max_depth)
                break;

            // Der Strahl wird fuer den naechsten Durchlauf aktualisiert
            // Die neue Richtung wird im Closest Hit Programm bestimmt
```

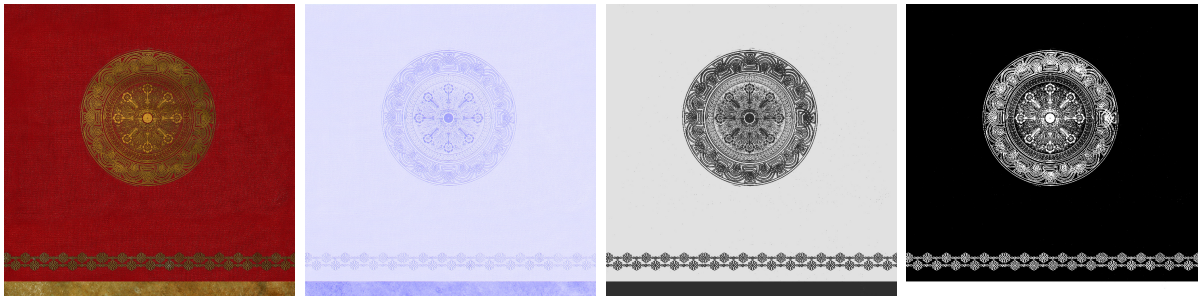
```
            ray_origin = prd.origin;
            ray_direction = prd.direction;
        }

        result += prd.result;
    } while (--samples_per_pixel);

    float3 pixel_color = result/(sqrt_num_samples*sqrt_num_samples);

    // Das Ergebnis wird in den Puffer uebertragen, der anschliessend angezeigt wird
    if (frame_number > 1) {
        // Lineare interpolation
        float a = 1.0f / (float)frame_number;
        float3 old_color = make_float3(output_buffer[launch_index]);
        output_buffer[launch_index] = make_float4( lerp( old_color, pixel_color, a ), 1.0f);
    } else {
        output_buffer[launch_index] = make_float4(pixel_color, 1.0f);
    }
}
```

Jeder Thread auf der Grafikkarte führt dabei das Programm parallel für einen anderen Pixel im Bild aus. Dieser Vorgang wird so lange wiederholt, bis die Anzahl an Samples für den Pixel erreicht wurde. Im Anschluss wird für das nächste Frame die `frame_number` erhöht und die Programmausführung wiederholt. Falls der Nutzer die Kamera bewegt und das Bild neu berechnet werden muss, wird die `frame_number` wieder mit 0 beschrieben, was dazu führt, dass in Zeile 47 die vorherigen Berechnungen überschrieben werden. Das Kamera Koordinatensystem wird in Form einer Rotationsmatrix

$$R = \left(\hat{u}_{right}, \hat{v}_{up}, \hat{w}_{forward}\right) \tag{5.2}$$

und der Position der Kamera $\mathbf{p}_{eye}$ übergeben. Die Rotationsmatrix stellt sich dabei aus drei normierten Vektoren zusammen, die in Zeile 12 verwendet werden, um den Strahl aus dem Kamerakoordinatensystem ins Weltkoordinatensystem zu transformieren.

5.1.2 Closest Hit Programm



(a) Diffuser Reflexionsgrad **(b)** Oberflächennormale **(c)** Rauheit der Oberfläche **(d)** Metallische Eigenschaften

Abbildung 5.5: Beispieltexturen einer verwendeten Testszene.

Das Closest Hit Programm wird ausgeführt, wenn der Strahl mit einer Oberfläche kollidiert. Dabei kann für jede Oberfläche ein eigenes Closest Hit Programm definiert werden. In dieser Arbeit wurde nur ein Programm für alle Oberflächen implementiert und die Materialeigenschaften werden als Texturen an die Grafikkarte übergeben. Diese Texturen

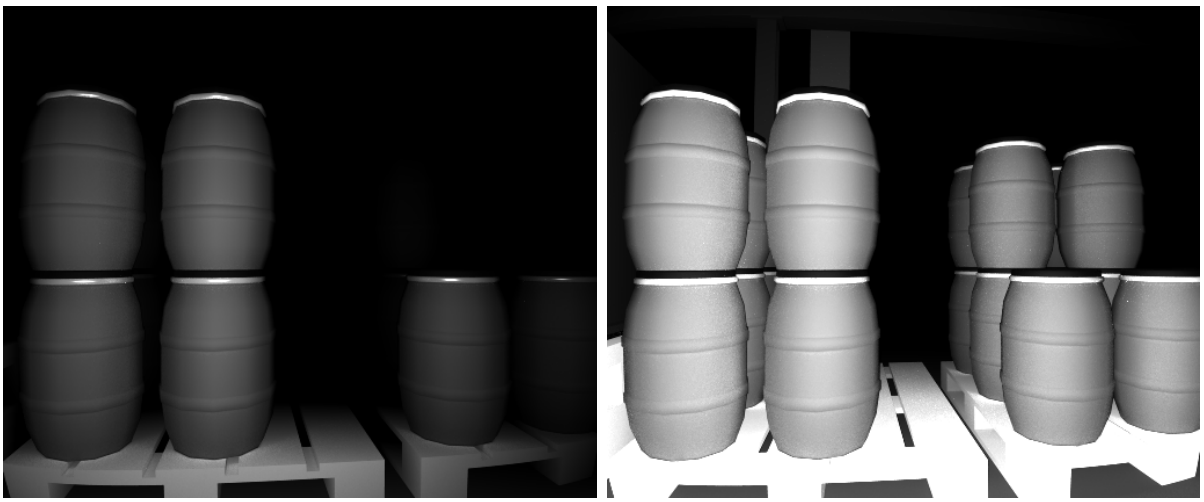
enthalten dabei den diffusen Reflexionsgrad in Form einer 24 Bit RGB Farbe, die Rauheit der Oberfläche als 8 Bit Grauwert, die metallischen Eigenschaften als 8 Bit Maske und eine weitere Textur, die Details der Oberflächenstruktur in Form von Normalen der Oberfläche enthalten, die als 24 Bit RGB Farbe kodiert wurde. In Abbildung 5.5 ist ein gewähltes Beispiel aus der Sponza Testszene zu sehen, die von Crytek erstellt wurde und häufig zum Testen von Lichtsimulationen verwendet wird.

Die Informationen, die in den Texturen übergeben werden, finden bei der Berechnung der BRDF Verwendung. Das Closest Hit Programm berechnet nach einer Kollision des Strahls mit der Oberfläche die Reflexionseigenschaften. Dazu wird wie in Unterabschnitt 2.1.6 erläutert bei jedem Schnittpunkt ein neuer Strahl in zufälliger Richtung generiert und so für jedes Sample ein Pfad verfolgt. In dieser Arbeit wird zur Optimierung eine cosinusgewichtete Dichte der Strahlen verwendet, weshalb die Multiplikation mit $cos\theta_i$ aus der Rendergleichung entfällt. Der folgende Codeauszug veranschaulicht die Berechnung der indirekten Beleuchtung des Closest Hit Programms:

```
RT_PROGRAM void microfacet_closest_hit() {
    // Die gereiste Distanz des Strahls wird hier aufsummiert
    float3 hit_point = old_ray_origin + (ray_length * old_ray_direction);
    prd_radiance.ir_traveled_distance += length(old_ray_origin - hit_point);

    // Hier wird der Einfluss des direkten Lichts berechnet
    // Dieser unterscheidet sich je nach gewaehlter Modulation
    // ...

    // Der neue Strahl hat seinen Ursprung am Schnittpunkt
    prd_radiance.origin = hit_point;

    // Es wird eine neue Richtung fuer den Strahl berechnet
    prd_radiance.direction = cosine_sample_hemisphere(surface_normal);

    // Der Abschwaechungfaktor der Reflextion wird bestimmt:
    // Es ist dabei bekannt, aus welcher Richtung der Strahl kommt und in welche
    // Richtung er reflektiert wird.

    // Es wird zwischen metallen und nicht-metallen unterschieden
    float3 fresnel;
    if(metallic) {
        // Hier wird der Fresnel Reflexionsgrad berechnet, falls die Oberflaeche ↩
            metallisch ist
        fresnel = diffuse_reflectance * FrConductor(dot(prd_radiance.direction, ↩
            half_vector), current_index_of_refraction, index_of_refraction, ↩
            absorption_coefficien);
    } else {
        // Hier wird der Fresnel Reflexionsgrad berechnet, falls die Oberflaeche nicht ↩
            metallisch ist
        fresnel = FrDielectric(dot(prd_radiance.direction, half_vector), ↩
            current_index_of_refraction, index_of_refraction);
    }

    // Berechnung des diffusen Reflexionsanteils mittels Oren Nayar BRDF
    float3 diffuse = OrenNayar_f(diffuse_reflectance, surface_normal, -↩
        old_ray_direction, prd_radiance.direction, roughness) * (1.0f - fresnel) * (1.0↩
        f - metallic);

    // Berechnung des spiegelnden Reflexionsanteils mittels Torrance Sparrow BRDF
    float3 specular = TorranceSparrow_f(surface_normal, -old_ray_direction, ↩
        prd_radiance.direction, fresnel, roughness);

    // Neue Berechnung des Abschwaechungsfaktors entsprechend des berechneten ↩
        Reflexiongrades
    prd_radiance.attenuation = (diffuse + specular) * prd_radiance.attenuation * PI;
}
```

Auch hier handelt es sich um einen stark vereinfachten Auszug der Implementierung, der nur der Veranschaulichung dient. Dabei wurde die Berechnung der direkten Beleuchtung ausgeklammert, da diese sich für jede Art von Lichtquelle unterscheidet. Die Implementierung der direkten Beleuchtung wird daher in den folgenden Unterkapiteln jeweils im Detail erläutert.

Pulsdauermodulation



(a) Intensität von C_1 **(b)** Intensität von C_2

Abbildung 5.6: Die Intensitäten der beiden Buckets C_1 und C_2, die bei der Simulation entstehen.

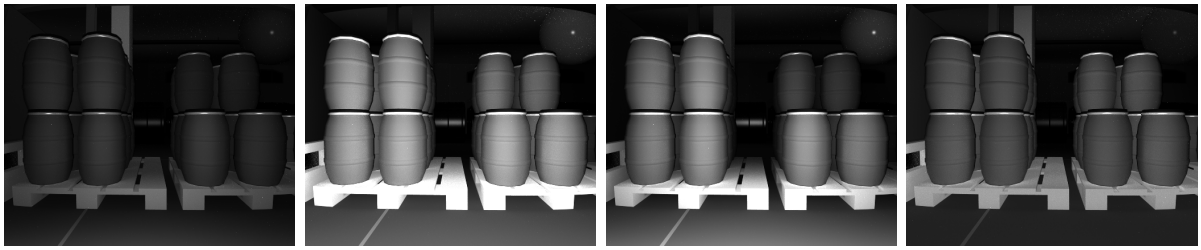
Im Folgenden wird die Berechnung der Intensitätsbilder für die Pulsdauermodulation erläutert. Bei der Pulsdauermodulation werden, wie in Unterabschnitt 2.2.1 erläutert, zwei Intensitätsbilder erstellt, aus denen anschließend das Tiefenbild berechnet wird. Abbildung 5.6 zeigt die beiden Intensitäten der Buckets C_1 und C_2, die in der Simulation berechnet werden. Dazu wird die Berechnung des Lichts des Path Tracing Algorithmus um den Faktor Zeit erweitert, die das Licht benötigt, um von einer Oberfläche reflektiert zu werden und beim Sensor aufzutreffen. Wie in Unterabschnitt 5.1.2 ausgeführt, wird bei der Berechnung der indirekten Beleuchtung die Distanz, die das Licht gereist ist, aufsummiert. Bei jedem Schnittpunkt des Strahls mit der Oberfläche wird berechnet, ob dieser Punkt direkt beleuchtet wird. Dazu wird vom Schnittpunkt ein Strahl in die Richtung der Infrarot LEDs geschickt und getestet, ob sich auf dem direkten Weg zwischen der Oberfläche und der Lichtquelle ein Hindernis befindet. Falls die Lichtquelle vom Schnittpunkt aus sichtbar ist, wird die Oberfläche direkt durch das modulierte Licht der Infrarot LEDs beleuchtet. Zur Berechnung des reflektierten Anteils des Lichts in dem Zeitraum, in dem die Buckets geöffnet sind, wird zu der bis dahin zurückgelegten Strecke des Pfades die Distanz von der Lichtquelle zum Schnittpunkt hinzuaddiert, um die Gesamtstrecke zu erhalten, die das Licht zurückgelegt hat. Aus der Strecke und der Lichtgeschwindigkeit wird der Zeitversatz ermittelt, mit dem der Lichtpuls beim Sensor ankommt. Daraus lässt sich berechnen, welcher Teil des reflektierten Lichts im Bucket C_1 und welcher im Bucket C_2 gespeichert wird.

Der folgende Ausschnitt der Implementierung füllt die Lücke des vorangegangenen Codeauszugs:

```
// ...

// Berechnung der Pulslaenge anhand der genutzten Frequenz
const double pulselength = (1.0f / frequency) * 0.5f;

// Da das reflektierte Licht in zwei Buckets aufgeteilt wird, wird hier ein 2D Vektor zum Speichern verwendet
float2 ir_result_pulse = make_float2(0.0f);

unsigned int num_ir_lights = ir_lights.size();
for(int i = 0; i < num_ir_lights; ++i) {
    // Berechnung einiger nuetlicher Variablen
    BasicLight light = ir_lights[i];
    float3 light_direction = light.pos - hit_point;
    float light_distance = sqrt(dot(light_direction, light_direction));
    light_direction = light_direction / light_distance;

    float NdotL = saturate(dot(surface_normal, light_direction));
    if (NdotL > 0.0f) {

        // Berechne einen Strahl vom Schnittpunkt zur Lichtquelle um zu bestimmen, ob sich die Oberflaeche im Schatten befindet oder direkt beleuchtet wird
        PerRayData_shadow shadow_prd;
        Ray shadow_ray = make_Ray( hit_point, light_direction, shadow_ray_type, scene_epsilon, light_distance - scene_epsilon );
        rtTrace(top_object, shadow_ray, shadow_prd);

        if(!shadow_prd.inShadow) {

            // Hier wird genau wie bereits beschrieben der diffuse und spekulare Anteil der Reflexion berechnet
            // ...

            float Intensity = prd_radiance.ir_attenuation * luminanceCIE((diffuse + specular) * (light.color * light.intensity * (NdotL / dot(light_direction, light_direction))));

            // Berechne anhand der gereisten Distanz den Zeitversatz, an dem die Strahlung wieder beim Sensor auftrifft
            float sourceToSensorDistance = light_distance + prd_radiance.ir_traveledDistance;
            float deltaTime = sourceToSensorDistance / speedOfLight;

            // Bestimme den Startzeitpunkt, an denen die Buckets beginnen Licht aufzunehmen
            float C1_start = 0.0f;
            float C2_start = pulselength;

            // Bestimme den Anfangszeitpunkt und den Endzeitpunkt des reflektierten Lichtpulses
            float begin = deltaTime;
            float end = deltaTime + pulselength;

            // Bestimme den Anteil der Ladung, der von C1 aufgenommen wird
            if((end >= C1_start && end <= C1_start + pulselength) || (begin >= C1_start && begin <= C1_start + pulselength)) {
                if(begin > C1_start)
                    ir_result_pulse.x += (((C1_start + pulselength) - begin) / pulselength) * Intensity;
                else
                    ir_result_pulse.x += ((end - C1_start) / pulselength) * Intensity;
            }

            // Bestimme den Anteil der Ladung, der von C2 aufgenommen wird
            if((end >= C2_start && end <= C2_start + pulselength) || (begin >= C2_start && begin <= C2_start + pulselength)) {
                if(begin > C2_start)
                    ir_result_pulse.y += (((C2_start + pulselength) - begin) / 
```

```
                        pulselength) * Intensity;
                else
                    ir_result_pulse.y += ((end - C2_start) / pulselength) * Intensity;
            }
        }
    }
}

// Gebe das Ergebnis an das Ray Generation Programm zurueck
prd_radiance.radiance = ir_result_pulse;

// ...
```

Rechteckförmige Wellenfunktion



(a) Intensität C_1 **(b)** Intensität C_2 **(c)** Intensität C_3 **(d)** Intensität C_4

Abbildung 5.7: Die Intensitäten der Buckets C_1, C_2, C_3 und C_4, die bei der Simulation der Continuous-Wave Modulation entstehen.

Die Simulation der Phasenbilder für die Continuous-Wave Modulation verläuft ähnlich zur Implementierung der Pulsdauermodulation. Die Continuous-Wave Modulation unterscheidet sich von der Pulsdauermodulation in zwei Aspekten:

1. Es werden, wie in Unterabschnitt 2.2.2 erläutert, vier statt zwei Buckets verwendet, um das reflektierte Licht aufzunehmen.
2. Statt eines Pulses wird eine Welle emittiert, wobei die rechteckförmige Wellenfunktion als eine sich periodisch wiederholende, Pulsdauermodulation betrachtet werden kann.

Die Ähnlichkeit der rechteckförmigen Wellenfunktion zur Pulsdauermodulation wird genutzt und die Ladung der Buckets wird auf dieselbe Weise simuliert. Dazu wird die Berechnung zunächst um zwei zusätzliche Buckets erweitert und das Zeitfenster, in dem die Buckets offen sind, wie zuvor berechnet. Da im Gegensatz zur Pulsdauermodulation eine periodische Welle emittiert wird, müssen alle vorherigen Lichtpulse der Berechnung berücksicht werden. Dazu wird der Lichtpuls so lange um seine doppelte Pulslänge verschoben und die Verteilung auf die vier Buckets für jeden Lichtpuls berechnet, bis die komplette Welle bei der Berechnung der Ladung der Buckets berücksichtigt worden ist.

Der folgende Auszug aus der Implementierung veranschaulicht die Berechnung der Phasenbilder für die Continuous-Wave Modulation von rechteckförmigen Wellenfunktionen:

```
// ...

// Da das reflektierte Licht in vier Buckets aufgeteilt wird, wird hier ein 4D Vektor zum Speichern verwendet
float4 ir_result_rect = make_float4(0.0f);

// ...

// Berechne den Zeitraum, in denen die Buckets offen sind und Licht aufnehmen
float C1_start = 0.0f;
float C2_start = pulselength;
float C3_start = pulselength * 0.5f;
float C4_start = (pulselength * 0.5f) + pulselength;

// Verschiebe den Lichtpuls so lange in die Vergangenheit, bis er ausserhalb des Messrahmens der vier Buckets liegt
while(deltaTime < C4_start + pulselength) {
    deltaTime = deltaTime + (pulselength * 2.0);
}

// Berechne die Intensitaetsverteilung analog zur Pulsdauermodulation fuer jeden einzelnen Lichtpuls
while(deltaTime + pulselength > 0.0f) {
    float begin = deltaTime;
    float end = deltaTime + pulselength;

    if((end >= C1_start && end <= C1_start + pulselength) || (begin >= C1_start && begin <= C1_start + pulselength)) {
        if(begin > C1_start)
            ir_result_rect.x += (((C1_start + pulselength) - begin) / pulselength) * Intensity;
        else
            ir_result_rect.x += ((end - C1_start) / pulselength) * Intensity;
    }

    if((end >= C2_start && end <= C2_start + pulselength) || (begin >= C2_start && begin <= C2_start + pulselength)) {
        if(begin > C2_start)
            ir_result_rect.y += (((C2_start + pulselength) - begin) / pulselength) * Intensity;
        else
            ir_result_rect.y += ((end - C2_start) / pulselength) * Intensity;
    }

    if((end >= C3_start && end <= C3_start + pulselength) || (begin >= C3_start && begin <= C3_start + pulselength)) {
        if(begin > C3_start)
            ir_result_rect.z += (((C3_start + pulselength) - begin) / pulselength) * Intensity;
        else
            ir_result_rect.z += ((end - C3_start) / pulselength) * Intensity;
    }

    if((end >= C4_start && end <= C4_start + pulselength) || (begin >= C4_start && begin <= C4_start + pulselength)) {
        if(begin > C4_start)
            ir_result_rect.w += (((C4_start + pulselength) - begin) / pulselength) * Intensity;
        else
            ir_result_rect.w += ((end - C4_start) / pulselength) * Intensity;
    }

    // Verschiebe den Lichtpuls um seine doppelte Phasenlaenge um den zeitlich folgenden Lichtpuls zu berechnen
    deltaTime = deltaTime - (pulselength * 2.0);
}

// ...
```

Die aus der Simulation resultierenden Ladungen der einzelnen Buckets werden in Form von Phasenbildern in Abbildung 5.7 veranschaulicht.

Sinusförmige Wellenfunktion

Bei der sinusförmigen Wellenfunktion handelt es sich um die simpelste Berechnung zur Phasenverschiebung. Dazu wird für jeden Bucket der Flächeninhalt unter der Wellenfunktion für den Abschnitt berechnet, in denm die Buckets offen sind:

$$Q_k = \frac{\alpha \cdot \Big(\cos(2\pi f \cdot t_{start}) - \cos(2\pi f \cdot t_{end}) \Big)}{2\pi f} - (\beta \cdot t_{start}) + (\beta \cdot t_{end}). \tag{5.3}$$

Dabei ist α die Amplitude der Funktion, f die Frequenz, β der Versatz, k der Index des Buckets und t_{start} und t_{start} die End- und Startzeit, in der ein Bucket geöffnet ist.

Folgender Codeauszug zeigt abschließend die Berechnung der Phasenbilder für die sinusförmige Wellenfunktion:

```
// ...

// Da das reflektierte Licht in vier Buckets aufgeteilt wird, wird hier ein 4D Vektor zum Speichern verwendet
float4 ir_result_sin = make_float4(0.0f);

// ...

ir_result_sin.x += Intensity * getArea(frequency, C1_start - deltaTime, (C1_start - deltaTime) + pulselength);
ir_result_sin.y += Intensity * getArea(frequency, C2_start - deltaTime, (C2_start - deltaTime) + pulselength);
ir_result_sin.z += Intensity * getArea(frequency, C3_start - deltaTime, (C3_start - deltaTime) + pulselength);
ir_result_sin.w += Intensity * getArea(frequency, C4_start - deltaTime, (C4_start - deltaTime) + pulselength);

// ...
```

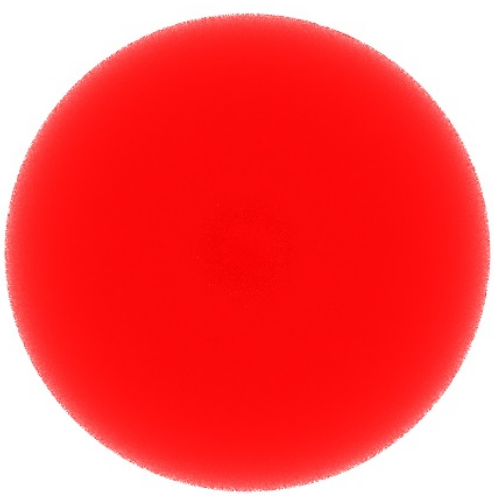
Die Phasenbilder unterscheiden sich nur minimal von denen der rechteckförmigen Wellenfunktion, die in Abbildung 5.7 zu sehen sind.

5.1.3 Miss Programm

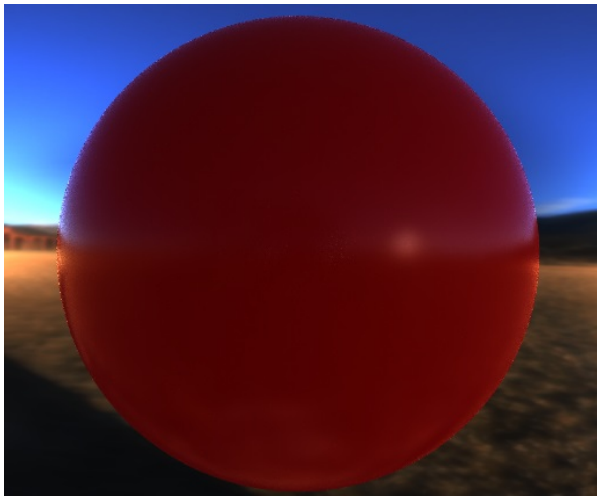
Das Miss Programm wird aufgerufen, sobald der Strahl die Szene verlässt und keine Kollision mit der Geometrie mehr stattfinden kann. Dabei kann der Pfad einfach beendet werden oder für den Hintergrund eine Strahlung festgelegt werden, was zu einer Umgebungsbeleuchtung führt. Die Simulation des Umgebungslichts wirkt sich auf den zufälligen Fehler der Tiefenwerte aus, weshalb diese in dieser Arbeit relevant ist.

Folgender Codeauszug veranschaulicht die Verwendung eines konstanten Hintergrundes als Strahlungsquelle:

```
RT_PROGRAM void miss()
{
    prd_radiance.radiance = prd_radiance.attenuation * background_color;
    prd_radiance.done = true;
}
```



(a) Rote Plastiksphäre bei konstanter Hintergrundbeleuchutng



(b) Rote Plastiksphäre bei einer Beleuchtung durch eine Environment Map

Abbildung 5.8: Mit einem Miss Programm können sowohl eine konstante Hintergrundbeleuchtung als auch eine eingemessene Beleuchtung durch eine Environment Map genutzt werden.

Falls für den Hintergrund ein Wert größer als Null gewählt wird, wird die Umgebung als Strahlungsquelle betrachtet, falls der Strahl die Szene verlässt (siehe Abbildung 5.8a). Alternativ lässt sich hier eine eingemessene Umgebungsbeleuchtung nutzen, die als Strahlungsquelle dient, falls der Strahl die Szene verlässt (siehe Abbildung 5.8b). Dazu wird eine sogenannte *Environment Map* aufgenommen und auf eine Sphäre projiziert, die die Kamera umgibt. Falls der Strahl die Szene verlässt, wird die Schnittstelle mit der Sphäre berechnet und die Strahlung aus der entsprechenden Richtung bestimmt.

Folgender Codeauszug veranschaulicht die Verwendung einer Environment Map als Strahlungsquelle:

```
rtTextureSampler<float4, 2> envmap;
RT_PROGRAM void envmap_miss()
{
    float theta = atan2f( ray.direction.x, ray.direction.z );
    float phi   = M_PIf * 0.5f -  acosf( ray.direction.y );
    float u     = (theta + M_PIf) * (0.5f * M_1_PIf);
    float v     = 0.5f * ( 1.0f + sinf(phi) );

    prd_radiance.radiance = prd_radiance.attenuation * make_float3( tex2D(envmap, u, v)↩
        );
    prd_radiance.done = true;
}
```

5.2 Tiefenbildgenerierung aus den Phasenbildern

m folgenden Abschnitt wird darauf eingegangen, wie aus den einzelnen Phasenbildern Tiefenbilder errechnet werden. Die Simulation der Lichtausbreitung mittels Path Tracing liefert je nach gewählter Modulation und Anzahl der verwendeten Buckets zwei oder vier Phasenbilder, aus denen die Tiefeninformationen berechnet werden. Dazu wird, wie

in Abschnitt 2.2 erläutert, vorgegangen und die Distanz wird entweder direkt aus dem Verhältnis der zwei Buckets berechnet oder über die Phasenverschiebung ermittelt. Bei den Tiefeninformationen handelt es sich um eine radiale Distanz von der Blende der Kamera zum Objekt. Bei der radialen Distanz handelt es sich um die Länge eines Vektors, dessen Ursprung mit dem des Kamerakoordinatensystems übereinstimmt. Da für jeden Pixel für den Path Tracing Algorithmus Strahlen berechnet wurden, ist bereits bekannt, in welche Richtung der Vektor zeigt, aus dem die Tiefeninformation ermittelt wurde. Um aus den Tiefeninformationen eine 3D Koordinate zu berechnen, wird der Vektor um die ermittelte radiale Distanz verlängert, die bei der Berechnung der Phase ermittelt wurde.

Time-of-Flight Kamerasysteme liefern häufig anstelle einer radialen Distanz eine Tiefe als Ergebnis. Dazu wird zunächst die 3D Koordinate im Raum berechnet und ausschließlich der Z-Wert der Koordinate in einem Tiefenbild an den Nutzer übergeben, worauf hier allerdings verzichtet wird, weshalb im Folgenden von Distanzbildern gesprochen wird.

Wie in Kapitel 4 erläutert, sind die Distanzen äußeren Einflüssen unterworfen, was zu Fehlern im Distanzbild führt. Die Simulation dieser Fehler wird im Folgenden erläutert.

5.2.1 Simulation des Lens Scattering und des Mixed Pixels Fehlers

Bei der genaueren Untersuchung des Mixed Pixels Fehler in Unterabschnitt 4.0.7 wurde die Vermutung angestellt, dass sich dieser auf den Lens Scattering Effekt zurückführen lässt, weshalb nur der Lens Scattering Fehler simuliert wird und der Mixed Pixels Fehler dadurch ebenfalls berücksichtigt wird. Bei der genauen Untersuchung des Einflusses heller Pixel auf die benachbarten Pixel in Unterabschnitt 4.0.6 wurde eine Punktspreizfunktion ermittelt, mit der sich dieser Effekt nachbilden lässt. Dazu wird aus der Funktion eine *Faltungsmatrix* definiert, die auf jedes Phasenbild angewandt wird. Das führt dazu, dass sich die Intensitäten auf die Nachbarpixel verteilen, wordurch die Fehler im Tiefenbild verursacht werden. Die Ergebnisse der Simulation werden in Kapitel 6 ausführlich Detail besprochen.

5.2.2 Simulation des zufälligen Fehlers

Der zufällige Fehler wird nach der Gleichung 2.43 implementiert. Dazu werden die Intensität A des Infrarotsignals und der Versatz B aus den Phasenbildern berechnet. In den Versatz B fließt zusätzlich das Umgebungslicht ein, das durch das Miss Programm simuliert wird, während die Intensität A ausschließlich die modulierte Strahlung der Infrarot LEDs enthält, selbst wenn das Umgebungslicht einen Einfluss auf die einzelnen Intensitätsbilder ausübt. Die Pulsdauermodulation ist davon allerdings ausgeschlossen, da sich hier das Umgebungslicht nicht aus den Phasenbildern herausrechnen lässt.

Der zufällige Fehler wird dabei zum resultierenden Distanzbild addiert. Dazu wird die Implementierung der Normalverteilung der Standard Template Library verwendet. Diese erwartet die Standardabweichung σ als Parameter, der in der Gleichung 2.43 berechnet wurde.

Der folgende Codeauszug demonstriert die Berechnung der modifizierten Distanz mittels einer Normalverteilung:

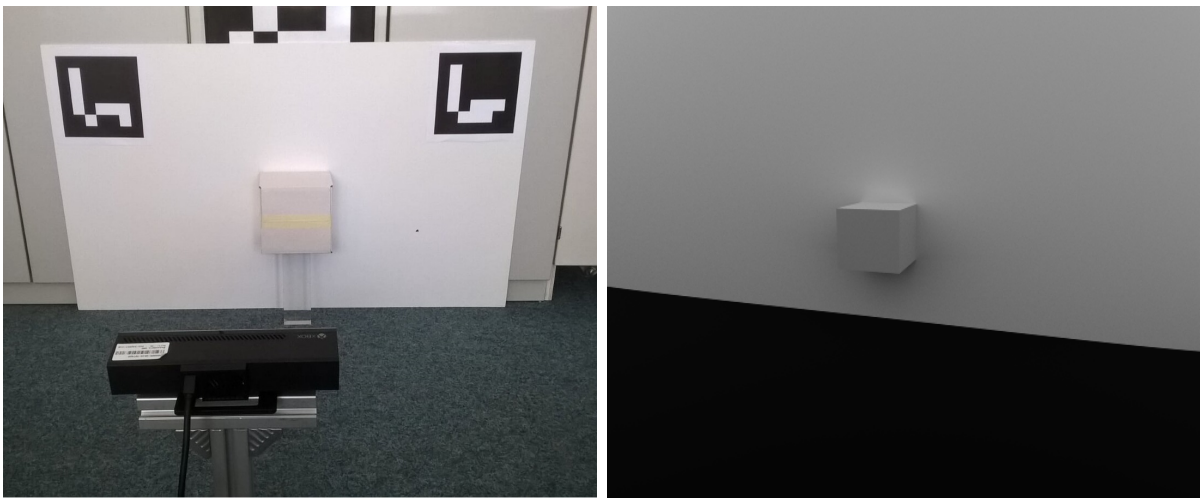
```
// ...

// Definition eines Zufallsgenerators
std::default_random_engine g_generator;

// ...

float distance = (speedOfLight / (4.0f * M_PIf * frequency)) * phi;
if (distance != 0.0f && m_noise_enabled)
{
    // Berechnung des Rauschens mittels der Normalverteilung
    std::normal_distribution<double> distribution(distance, standard_deviation);
    double imperfectDistance = distribution(g_generator);

    // Umrechnung von Meter auf Millimeter
    output_depth[launch_index] = imperfectDistance * 1000.0f;
}

// ...
```

6 Evaluation

Dieses Kapitel beschäftigt sich mit der Evaluation des implementierten Algorithmus. Dabei wird ausschließlich die Qualität der Bilder berücksichtigt und die Berechnungszeit ausgeklammert, da diese im Falle des Path Tracings stark von der gewählten Konfiguration und der zu rendernden Szene abhängt. Während einfache Szenen, die durch Environment Maps beleuchtet werden, in Echtzeit gerendert werden können, benötigen Szenen mit komplexen Lichtsituationen, einer hohen Anzahl an Samples und großen Pfadlängen einige Minuten zum Berechnen des Bildes. Da die Evaluation jedes denkbaren Szenarios über den Rahmen dieser Arbeit hinausgehen würde, wurde hier auf die Evaluation der Berechnungszeiten verzichtet.

Der letzte Abschnitt beschäftigt sich mit dem Vergleich zu anderen Forschungsarbeiten, die dasselbe Ziel verfolgen. Da ein direkter und ausführlicher Vergleich mit anderen Untersuchungen zu viel Raum einnehmen würde, erfolgt ein skizzenhafter Vergleich der Ergebnisse.

6.1 Vergleich der Simulation mit der Kinect v2

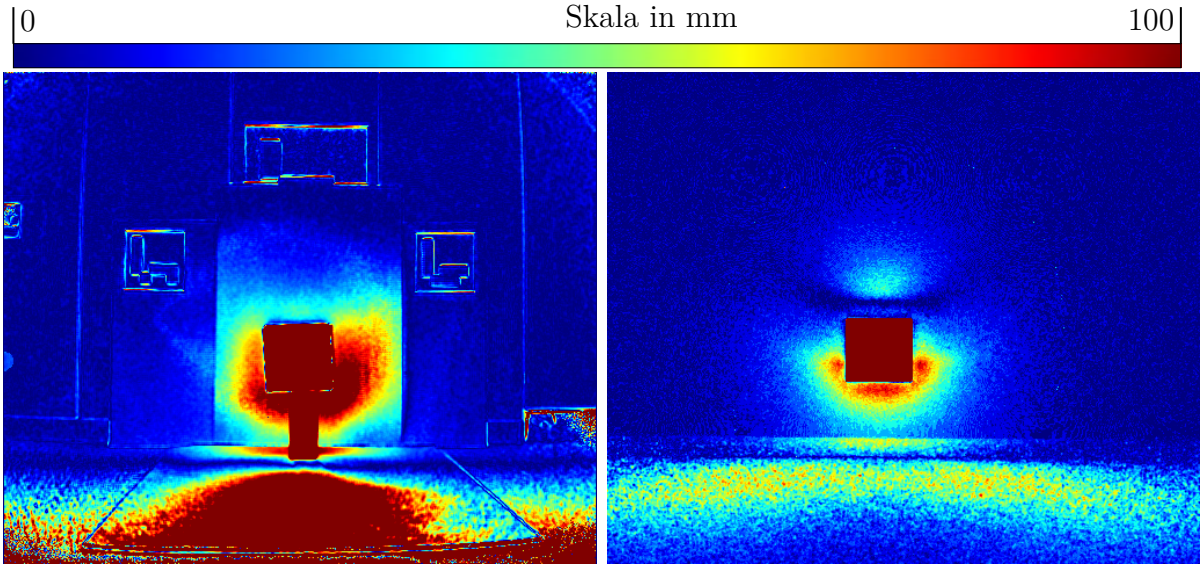


(a) Testaufbau mit der Kinect v2 **(b)** Testaufbau der Simulation

Abbildung 6.1: Versuchsaufbau mit der Kinect v2 und einer vereinfachten Szene innerhalb der Simulation

In diesem Unterkapitel wird das Ergebnis der Simulation mit echten Tiefenwerten der Kinect v2 verglichen. Dazu wurde ein Testszenario aufgebaut, in dem ein quadratischer Karton vor einen stark reflektierenden Hintergrund positioniert wurde. Diese Szene wurde anschließend in einer vereinfachten Form in der Simulation nachgebildet und die Kamera

wurde ähnlich positioniert (siehe Abbildung 6.1). Zum Vergleich wurde zunächst ein Tiefenbild des Hintergrundes ohne Karton aufgenommen und anschließend eine Aufnahme mit Karton angefertigt. Anschließend wurden sowohl für die Simulation und für die Aufnahmen der Kinect v2 die Tiefenwerte miteinander verglichen und der Fehler ermittelt, der durch den Karton verursacht wurde.



(a) Kinect v2 **(b)** Simulation

Abbildung 6.2: Vergleich des Fehlers der Tiefenwerte der Simulation mit echten Tiefenwerten der Kinect.

In der Abbildung 6.2 ist die Abweichung der Tiefenwerte nach der Positionierung des Kartons deutlich zu erkennen. Dabei ist anzumerken, dass die Rauheit und die optische Dichte der untersuchten Materialien nicht bekannt waren. Da die Analyse der Materialien im Rahmen der Arbeit nicht vorgenommen wurde, um dem vorgegebenen Rahmen der Untersuchung zu entsprechen, wurden daher plausible Werte für die Rauheit sowie die optische Dichte und den Reflexionsgrad gewählt, die allerdings nicht auf Messungen basieren, weshalb keine exakten Übereinstimmungen erwartet wurden. Das Ergebnis der Simulation zeigt, dass Abweichungen vorhanden sind, die allerdings wie erwartet in ihrer Intensität deutlich von den Ergebnissen der echten Aufnahme abweichen, da die Materialeigenschaften nicht übereinstimmen. Dennoch ist zu sehen, dass die Positionen, an denen die Abweichungen vorhanden sind, teilweise übereinstimmen, was besonders am Boden und unterhalb des Kartons zu erkennen ist. Diese Abweichungen am Boden werden überwiegend durch indirekte Beleuchtungen verursacht, während die Abweichungen in direkter Umgebung des Kartons überwiegend dem Lens Scattering zuzuschreiben sind.

Besonders starke Abweichungen finden sich im unteren Bildbereich, die durch starke indirekte Beleuchtung verursacht wird, wodurch die Punktewolke des Bodens deutlich verfälscht wird. Abbildung 6.3 zeigt die verfälschten Tiefenwerte des Bodens. Da die Vignettierung der Lichtquelle in der Simulation nicht nachgebildet wurde und der Boden in der Simulation daher stärtker direkt beleuchtet wird, ist diese Verzerrung schwächer ausgeprägt, was vergleichend in Abbildung 6.2 zu sehen ist.

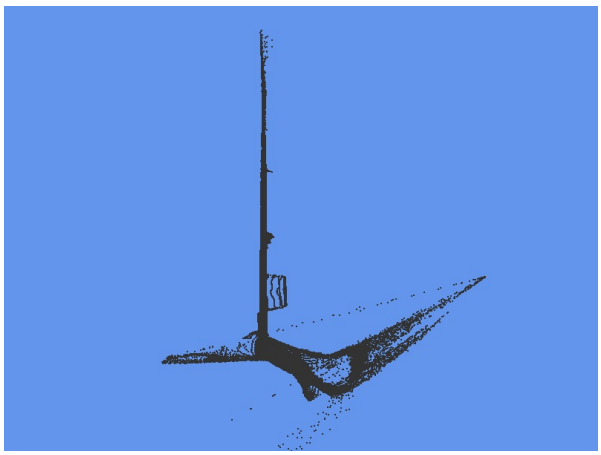
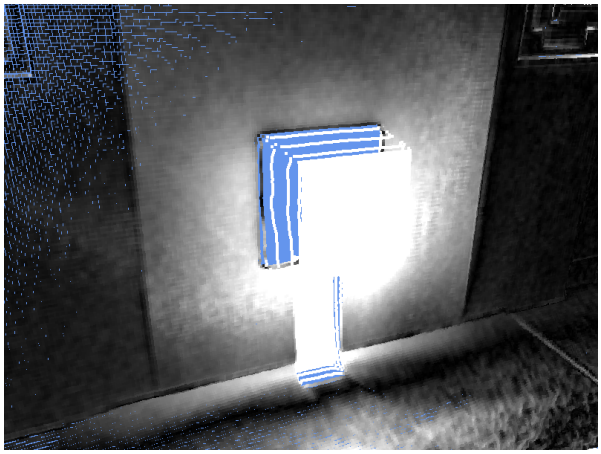


Abbildung 6.3: Aufnahme des Testaufbaus, die mit der Kinect v2 angefertigt wurde.

6.2 Lens Scattering und Mixed Pixels Fehler



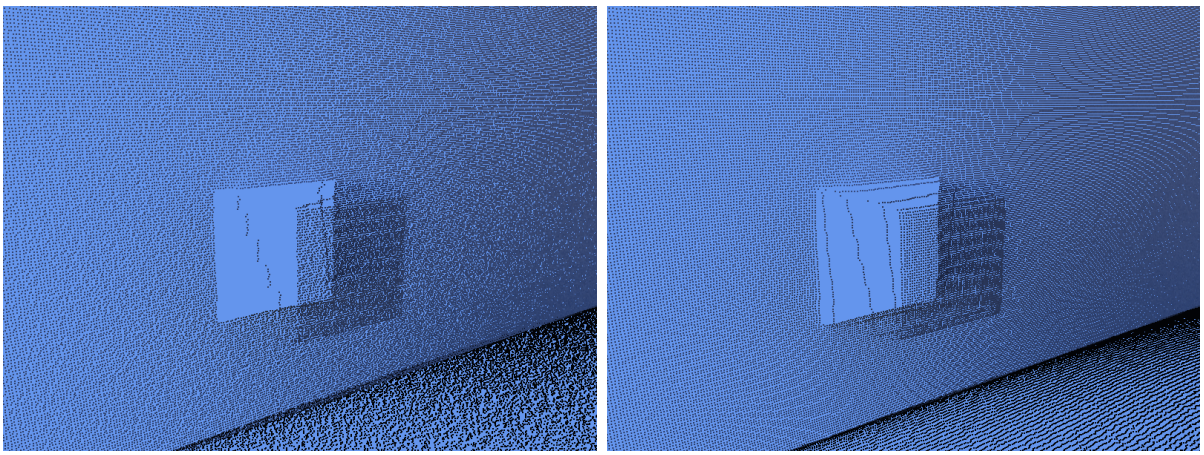
(a) Ohne Simulation von Lens Scattering

Abbildung 6.4: Fliegende Pixel, die durch Lens Scattering und Mixed Pixels verursacht wurden.

In Unterabschnitt 5.1.1 wurde erläutert, dass zum Rendern eines Pixels mehrere Samples mit Strahlen in unterschiedliche Richtungen berechnet wurden. Dies führt zwar dazu, dass Mixed Pixels Fehler verursacht werden. Das reicht allerdings nicht, um Ergebnisse zu erzeugen, die mit der Aufnahme der Kinect v2 vergleichbar sind. In Abbildung 6.4 wird anhand einer Aufnahme des Versuchsaufbaus gezeigt, wie die fliegenden Pixel durch Lens Scattering und Mixed Pixels verursacht werden.

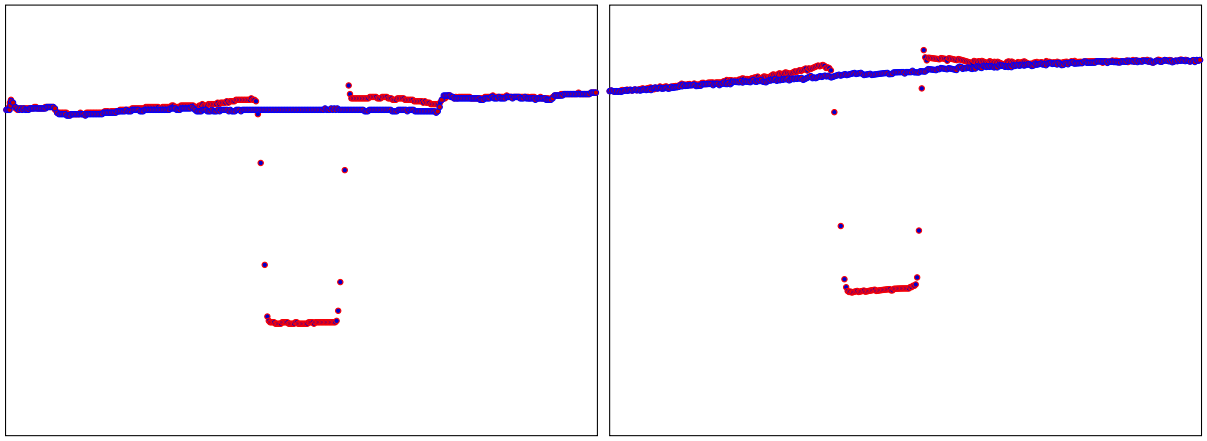
In Abbildung 6.5a sind die fliegenden Pixel zu sehen, die ausschließlich durch Mixed Pixels verursacht werden. Das Ergebnis weicht wie erwartet deutlich von dem der Aufnahme der Kinect v2 ab. Durch Berücksichtigung des Lens Scattering werden Ergebnisse erzielt, die mit den Aufnahmen der Kinect v2 übereinstimmen.

Neben den fliegenden Pixel verursacht das Lens Scattering außerdem, dass Tiefenwerte im Hintergrund durch stark reflektierende Objekte im Vordergrund verfälscht werden und umgekehrt. In Abbildung 6.6 werden die Tiefenwerte der Simulation mit den Aufnahmen



(a) Ohne Simulation von Lens Scattering **(b)** Mit Simulation von Lens Scattering

Abbildung 6.5: Vergleich des Fehlers der Punktewolke der Simulation mit der gemessenen Punktewolke der Kinect.



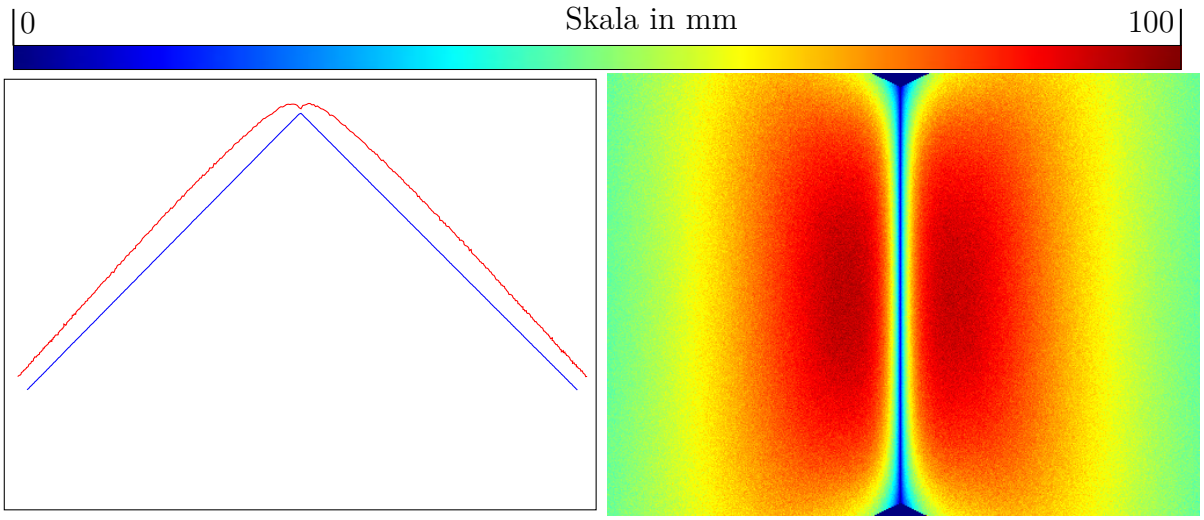
(a) Echte Tiefenwerte der Kinect v2 **(b)** Simulierte Tiefenwerte

Abbildung 6.6: Vergleich der gemessenen Tiefenwerte des Testaufbaus mit der Simulation.

der Kinect v2 verglichen. Die Betrachtung der Tiefenwerte der Simulation zeigt eine hohe Übereinstimmung mit den Tiefenwerten der Kinect v2. Für die Simulation wurde dafür die LED horizontal um 2,5 cm versetzt positioniert, wodurch die verfälschten Tiefenwerte im Hintergrund rechts neben der Box verursacht wurden. Zusätzlich dazu erzeugt die Simulation des Lens Scattering fliegende Pixel, die eine hohe Übereinstimmung mit den gemessenen Daten zeigen. Außerdem werden sowohl in der Simulation, als auch in den gemessenen Daten die Kanten des Kartons abgerundet und die Tiefenwerte im Hintergrund verfälscht.

6.3 Multiple Path Reflexionen

In Unterabschnitt 4.0.5 wurde der Fehler in Bezug auf Interreflexionen und indirekter Beleuchtung analysiert und mit den Tiefenwerten einer Punktewolke verglichen. Da es in dieser Arbeit nicht möglich war, Punktewolken in der Simulation zu rendern, wurden



(a) Horizontaler Schnitt der Tiefenwerte

(b) Verteilung der Abweichung

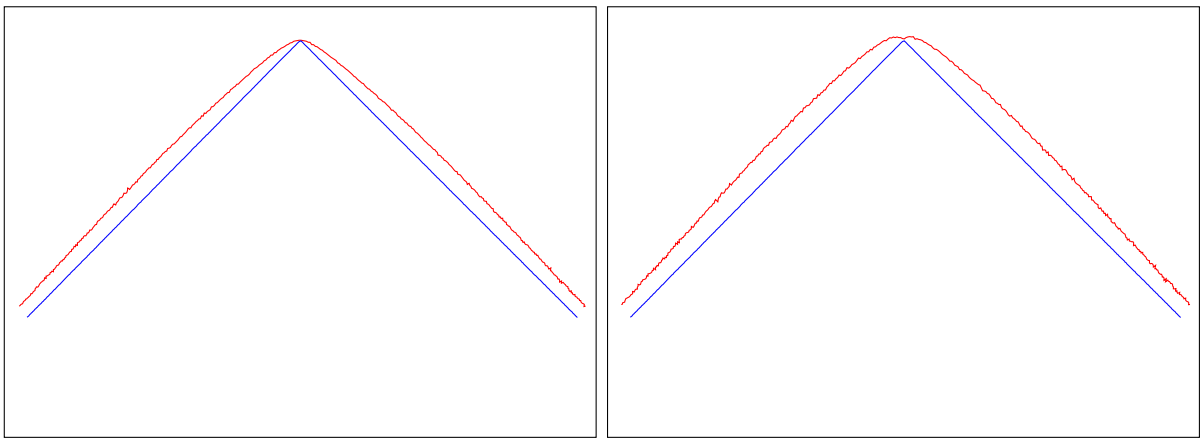
Abbildung 6.7: Darstellung der simulierten Tiefenwerte mit Multiple Path Fehler bei einer Pfadlänge von 16.

zum Vergleich zwei orthogonal zueinander ausgerichtete Wände vor der simulierten Kamera positioniert. Abbildung 6.7b zeigt das Ergebnis der Simulation mit einer Pfadlänge von 16 während für die Oberfläche eine Rauheit von $\sigma = 0.5$ und eine optische Dichte von 1.52 angesetzt wurde. In Abbildung 6.7 ist zu erkennen, dass die Tiefenwerte der Simulation größer sind, als die der Referenzwerte. Dieses Verhalten entspricht der Analyse, so wie es in Unterabschnitt 4.0.5 beobachtet wurde. Allerdings weicht das Ergebnis am Berührungspunkt der beiden Flächen, welcher sich im Zentrum des Bildes befindet, von den Erwartungen ab. Dies ist darauf zurückzuführen, dass Optix Schnittpunkte ignoriert, die sich zu nah am Ursprung des Strahls befinden. Dadurch werden Schnittpunkte direkt an der Kante ignoriert, was dazu führt, dass die Tiefenwerte der Simulation von denen der analysierten Kameras abweichen. Dabei handelt es sich allerdings um einen einstellbaren Parameter, welcher der Größe der Szene angepasst werden kann. Durch das Wählen eines kleineren Epsilons, das die Mindestentfernung zwischen zwei Schnittpunkten angibt, wird dieser Fehler behoben. Dies führt allerdings zu Artefakten, falls die Distanzen größer werden.

In den folgenden Abschnitten werden der Einfluss der maximalen Pfadlänge und der Rauheit der Oberfläche auf das Ergebnis im Detail untersucht.

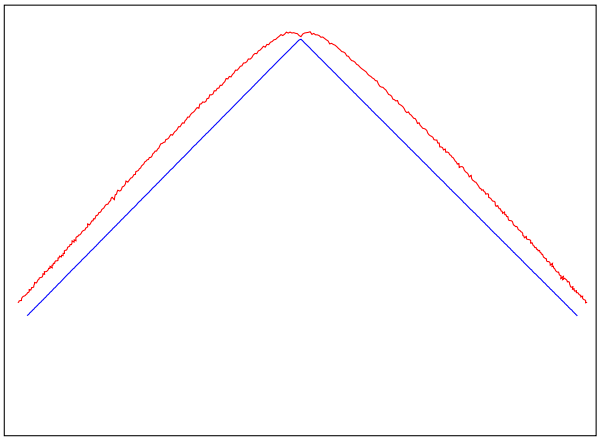
6.3.1 Fehler in Abhängigkeit zur Pfadlänge

Bei der Simulation der indirekten Beleuchtung spielt die Wahl der maximalen Pfadlänge eine bedeutende Rolle. So erhält man bei einer Pfadlänge von 1 die Referenztiefe ohne den Einfluss indirekter Beleuchtung und ab einer Pfadlänge von 2 übt die indirekte Beleuchtung einen Einfluss auf die Tiefenwerte aus. Zur Evaluation wurden zwei orthogonal zueinander ausgerichtete Oberflächen verwendet, die ebenfalls mit einer Rauheit von $\sigma = 0.5$ und einer optischen Dichte von 1.52 gerendert wurden. Für jede Aufnahme wurde die maximale Pfadlänge erhöht und es wurden genug Samples pro Pixel berechnet, damit

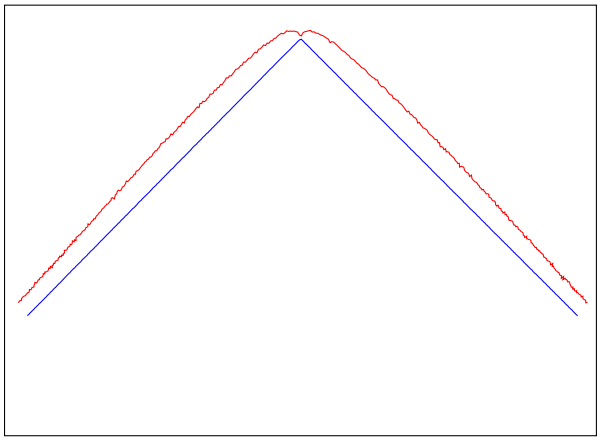


(a) Maximale Pfadlänge von 2

(b) Maximale Pfadlänge von 3



(c) Maximale Pfadlänge von 4



(d) Maximale Pfadlänge von 5

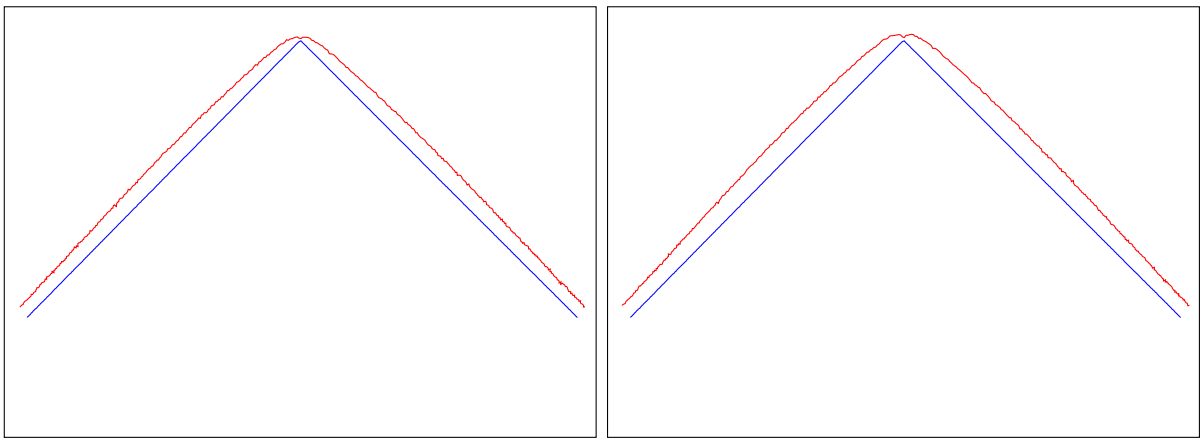
Abbildung 6.8: Vergleich der Tiefenwerte des horizontalen Schnitts von einer maximalen Pfadlänge von 2 bis 5.

das Bild vollständig berechnet wurde und sich dem korrekten Ergebnis annähert. Abbildung 6.8 zeigt den Vergleich der Tiefenwerte in Abhängigkeit von der maximalen Pfadlänge. Es ist zu erkennen, dass der Fehler, der durch eine indirekte Beleuchtung verursacht wurde mit der Pfadlänge zunimmt. Dabei wurde festgestellt, dass es ab einer Pfadlänge von 5 kein Unterschied zu einer Pfadlänge von 16 gibt.

6.3.2 Fehler in Abhängigkeit zur Rauheit der Oberfläche

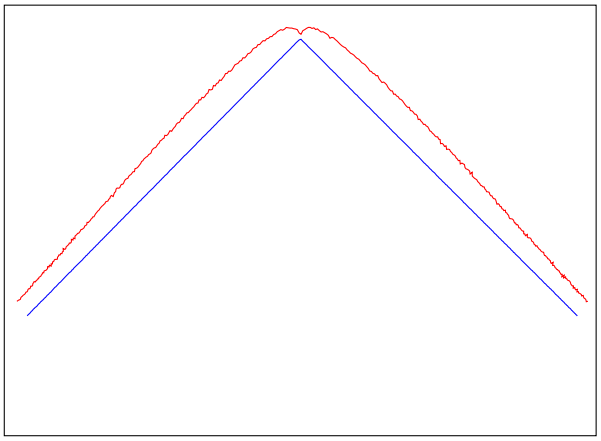
Zusätzlich zur Pfadlänge beeinflussen auch die Reflexionseigenschaften der Oberfläche den Fehler, der durch indirekte Reflexionen verursacht wird. Daher wurde die Simulation dahingehend näher untersucht. Dazu wurden ebenfalls wie zuvor in der Szene zwei orthogonal zueinander ausgerichtete Flächen gewählt und die Rauheit wurde variiert. Für diese Untersuchung wurde für jede Aufnahme wieder eine optische Dichte von 1.52 gewählt.

Die Ergebnisse der Untersuchung sind in Abbildung 6.9 zu sehen. Dabei ist zu erkennen, dass für eine Rauheit von $\sigma = 1.0$ der Fehler zunächst gering ausfällt und dieser sich für

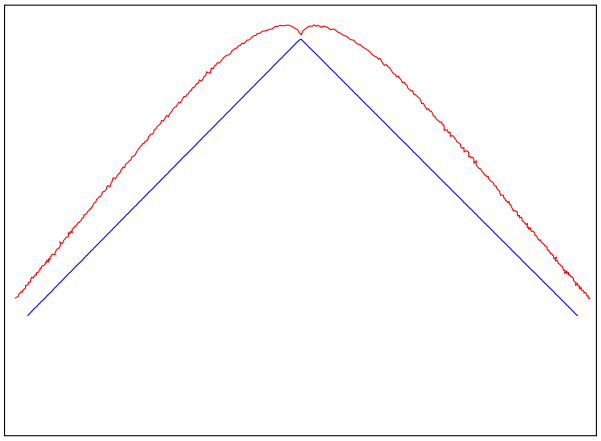


(a) $\sigma = 1.0$

(b) $\sigma = 0.7$



(c) $\sigma = 0.4$



(d) $\sigma = 0.1$

Abbildung 6.9: Vergleich der Tiefenwerte des horizontalen Schnitts in Abhängigkeit zur Rauheit der Oberfläche.

glattere Oberflächen erhöht. Dies ist damit zu erklären, dass rauere Oberflächen einen größeren Anteil der Strahlung in die Richtung der Strahlungsquelle reflektieren, was zu einer geringeren indirekten Beleuchtung für raue Oberflächen führt.

6.4 Systematische Fehler

Die Ermittlung des systematischen Fehlers der Simulation erfolgte analog zum Vorgehen, das in Unterabschnitt 4.0.4 beschrieben wurde. Allerdings wurden die Tiefenwerte der sinusförmiger Wellenfunktionen als Referenzwert herangezogen, da festgestellt wurde, dass die Continuous-Wave Modulation mittels sinusförmigen Wellenfunktionen keinen systematischen Fehler aufweist. Da anhand der Mehrdeutigkeitsdistanz festgestellt wurde, dass die IFM O3D03 mit einer Frequenz von 30 Mhz arbeitet und die ASUS Xtion 2 eine Frequenz von 20 Mhz verwendet, wird die Simulation mit diesen Frequenzen untersucht.

In Abbildung 6.10 ist der systematische Fehler der Simulation unter Verwendung der rechteckförmigen Amplutudenmodulation dargestellt. Dabei ist zu sehen, dass die Verwendung dieser Modulation einen sinusförmigen Fehler einführt. Die Analyse der O3D303

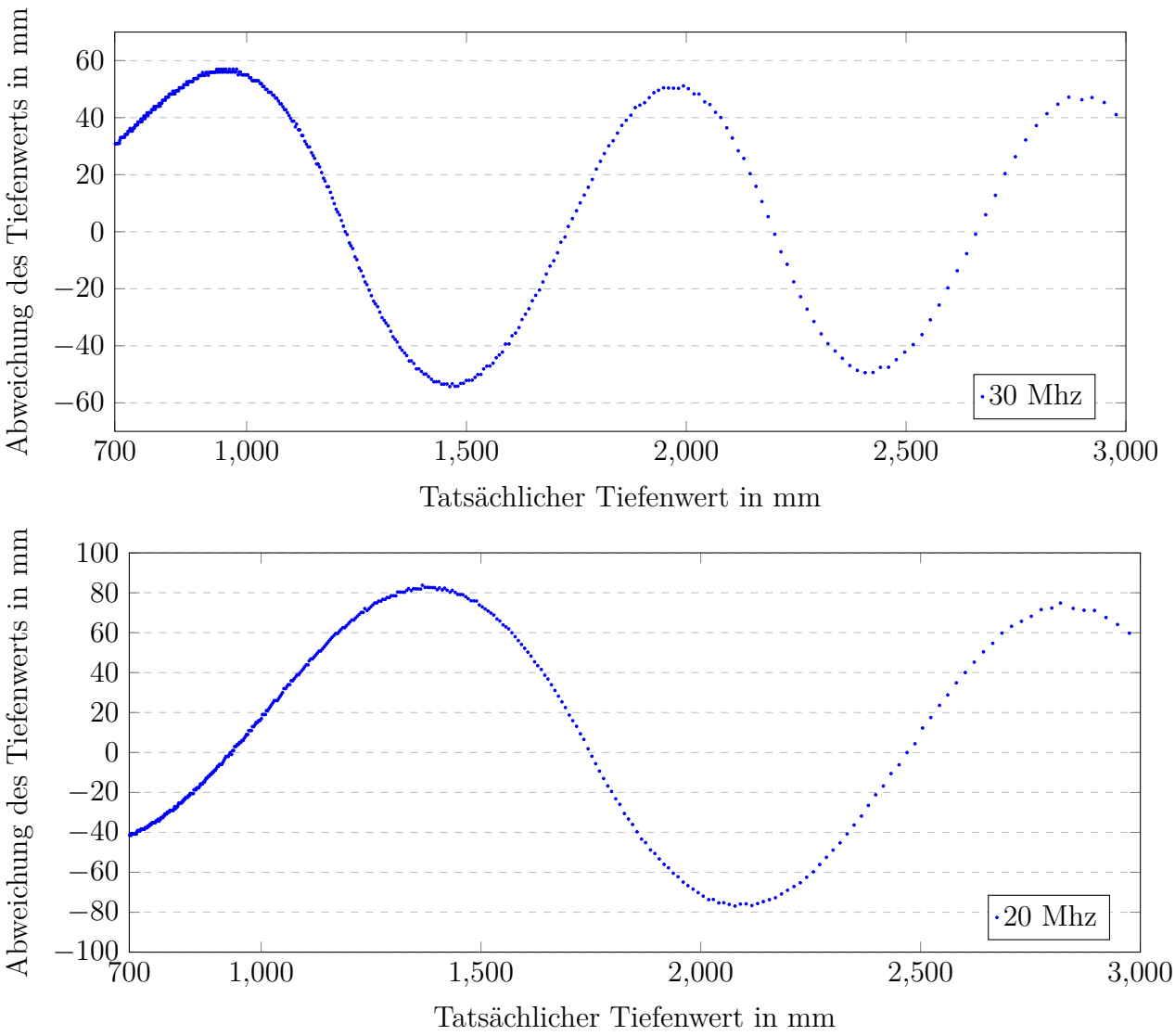


Abbildung 6.10: Systematischer Fehler bei der Simulation der Tiefenwerte mittels rechteckförmige Wellenfunktionen in Abhängigkeit zur tatsächlichen Distanz.

in Unterabschnitt 4.0.4 lässt erahnen, dass es sich bei dem Verlauf des systematischen Fehlers ebenfalls um einen Sinusverlauf handelt. Es wurde außerdem festgestellt, dass sowohl die Amplitude als auch die Wellenlänge des systematischen Fehlers von der genutzten Frequenz abhängig ist.

Weiter wurde auch der systematische Fehler der Pulsdauermodulation untersucht. Ohne den Einfluss jeglichen Umgebungslichtes konnte ebenfalls kein Fehler festgestellt werden. Die Tiefenwerte werden bei Verwendung der Pulsdauermodulation allerdings durch Umgebungslicht beeinflusst, das durch externe Quellen wie der Sonne oder künstlicher Beleuchtung im Raum verursacht wird. Abbildung 6.11 zeigt den Fehler der Simulation unter Einfluss von Umgebungslicht, das durch eine Environment Map verursacht wurde. Dabei sind Ähnlichkeiten zur Analyse des systematischen Fehlers der Xtion 2 zu erkennen. Daher liegt die Vermutung nahe, dass es sich bei der Xtion 2 um ein Time-of-Flight Kamerasystem handelt, das mit zwei Buckets arbeitet. Da allerdings keine Details zur internen Funktionsweise der Xtion 2 bekannt sind und eine genaue Untersuchung

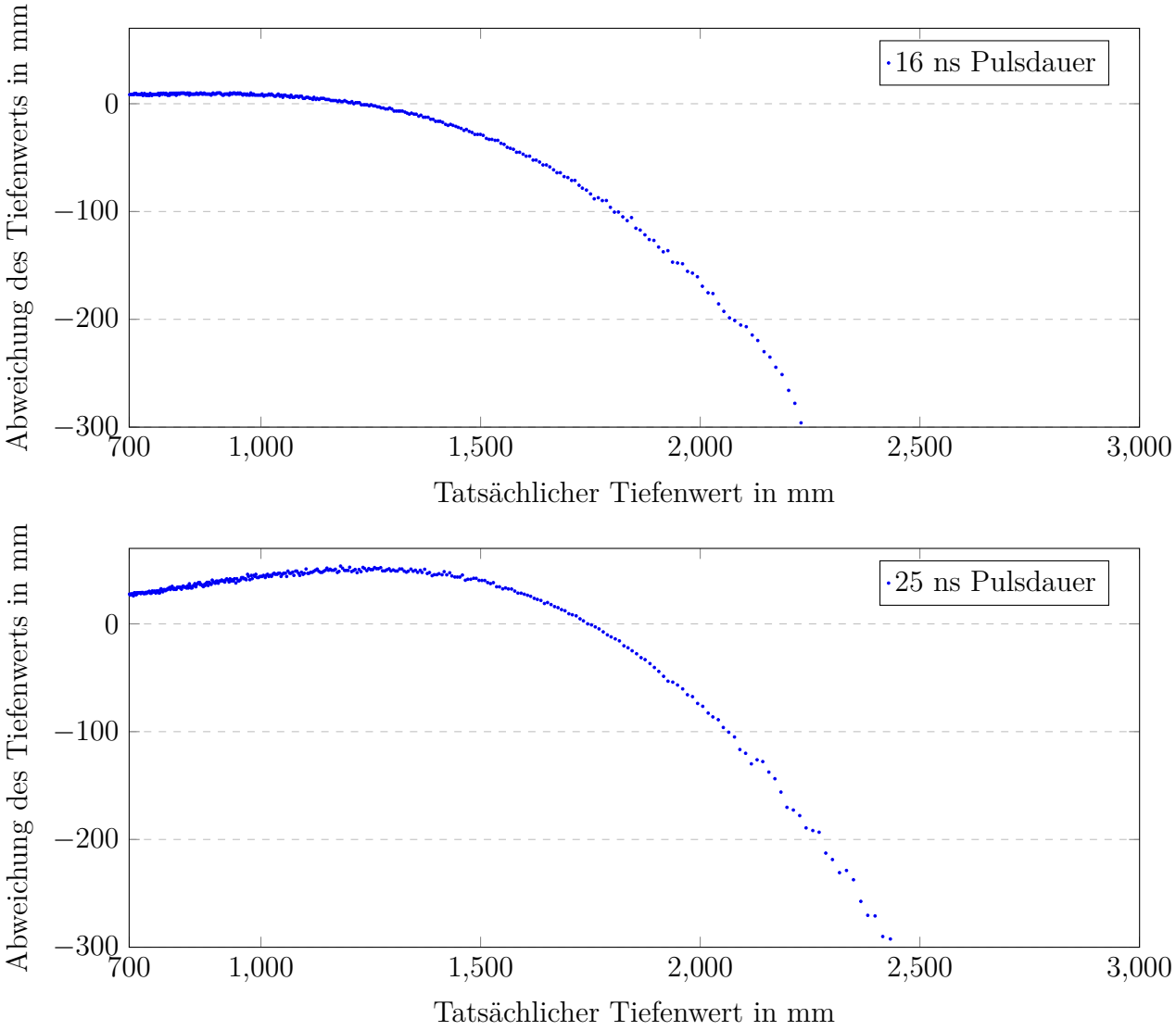


Abbildung 6.11: Systematischer Fehler bei der Simulation der Tiefenwerte mittels pulsdauermodulation in Abhängigkeit zur tatsächlichen Distanz.

den Rahmen der Arbeit übersteigt, kann keine genaue Aussage darüber getroffen werden.

6.5 Vergleich mit anderen Arbeiten

In Kapitel 3 wurden verwandte Arbeiten erläutert, die ein ähnliches Ziel verfolgen wie die vorliegende. Dabei wurde auf die Arbeiten von Meister et al. [Mei12][MNK13] Bezug genommen, die sich auf die Simulation der Lichtausbreitung konzentrieren. Zum einen wurde die Lichtausbreitung mittels Photon Mapping simuliert und zum anderen wurde bidirektionales Path Tracing verwendet. Dabei wurde die Linsenkrümmung nicht berücksichtigt und ausschließlich eine Continuous-Wave Modulation mit sinusförmigen Wellenfunktionen simuliert. Auch der systematische Fehler wurde nicht untersucht. Aus den Arbeiten geht ebenfalls nicht hervor, ob die Mehrdeutigkeit für größere Distanzen simuliert wurde.

Im Gegensatz zu den Ergebnissen, die in Unterabschnitt 6.3.1 erläutert wurden, weist die Simulation von Meister et al. [MNK13] keinen Fehler am Berührungspunkt der zwei Oberflächen auf. Allerdings kommen Meister et al. zum Schluss, dass eine Pfadlänge von mindestens 8 notwendig ist, um ein Tiefenbild zu erzeugen, das sich nicht von Tiefenbildern mit Pfadlängen mit einem höheren Wert unterscheidet. Die radiale Verzeichnung und Effekte, die durch Lens Scattering verursacht werden, welche in dieser Arbeit erfolgreich simuliert werden konnten, wurden in der Simulation von Meister et al. ebenfalls nicht berücksichtigt. Es wurde außerdem angenommen, dass die LED Position mit der Position der Kamera übereinstimmt, weshalb Fehler, wie in Unterabschnitt 4.0.8 erläutert, nicht adressiert wurden.

Keller [Kel15] konzentriert sich in seiner Dissertation auf die Echtzeitsimulation eines Time-of-Flight Sensors. Dabei wurde ebenfalls auf die Simulation der Linsenkrümmung und auf Lens Scattering verzichtet. Allerdings wurden die systematischen Fehler erfolgreich simuliert. Im Gegensatz zu dieser Arbeit hat Keller den systematischen Fehler eines Kamerasystems eingemessen um es anschließend in einer Simulation angewandt. Der Fehler durch indirekte Beleuchtung und sonstige Fehler, die durch die Lichtausbreitung verursacht wurden, wurden von Keller nicht adressiert. Allerdings ist die Simulation, die Keller in seiner Dissertation beschreibt, im Gegensatz zur Simulation in dieser Arbeit echtzeitfähig. Auch wenn das progressive Path Tracing Interaktionen des Nutzers während des Renderings erlaubt, wird bedeutend mehr Rechenzeit benötigt, bis das Bild gegen das richtige Ergebnis konvergiert. Das Endresultat des Bildes wird daher nicht in Echtzeit geliefert.

7 Fazit

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurden drei Time-of-Flight Kamerasysteme im Detail untersucht und anschließend simuliert. Während der Analyse der Kameras wurden mehrere Fehler untersucht, die durch interne und externe Einflüsse verursacht wurden. Dabei ist die Bedeutung des systematischen Fehlers deutlich geworden, dem alle untersuchten Kamerasysteme unterworfen waren. Ebenfalls wurde der Einfluss des Lens Scatterings untersucht, der sich auf das gesamte Tiefenbild erstreckt.

Die Simulation der Time-of-Flight Sensoren wurde auf der Grafikkarte mittels OptiX implementiert, während zur Simulation der Oberflächen das Mikrofacetten Modell verwendet wurde. Die Simulation verwendete das von Torrance und Sparrow [TS67] entwickelte Modell für raue, glänzende Oberflächen und das von Oren und Nayar [ON94] entwickelte Modell für raue, diffuse Oberflächen. Zur Simulation der Lichtausbreitung wurde ein Path Tracing Algorithmus implementiert. In der Simulation wurde die Linsenverzeichnung berücksichtigt, indem zunächst die Infrarotkameras der Time-of-Flight Systeme mittels eines Schachbrettmusters kalibriert und zur Erzeugung der Strahlen verwendet wurden. Der systematische Fehler wurde berücksichtigt, indem die Lichtausbreitung unter Berücksichtigung der Amplitudenmodulation der Infrarot LEDs simuliert wurde. Der Lens Scattering Fehler wurde dabei eingemessen und anschließend auf das Bild angewandt.

Abschließend wurde in einer Evaluation gezeigt, dass die Simulation der Time-of-Flight Sensoren den analysierten Fehlern unterworfen war. Dabei wurde die Simulation im Wesentlichen den Tiefenbildern der Kinect v2 gegenübergestellt und der Einfluss der indirekten Beleuchtung, der LED Positionierung und des Lens Scattering untersucht.

7.2 Kritische Reflextion

In Kombination mit der Simulation des Lens Scattering und der Berücksichtung der LED Position konnten Effekte in Tiefenbildern erzeugt werden, die im vorangegangenen Kapitel beobachtet wurden. Der systematische Fehler wurde erstmals physikalisch plausibel simuliert, während andere Arbeiten diesen Fehler erzeugten, indem sie diese Abweichungen einmessen und anschließend auf das Endresultat der Simulation aufrechnen. Für die Simulation des Lens Scattering und der Linsenverzeichnung wurde allerdings ebenfalls auf Messungen existierender Kameras zurückgegriffen.

Es wurde zwar gezeigt, dass die Simulation einen systematischen Fehler verursacht, der allerdings von den analysierten Kameras abweicht, was darauf zurückzuführen ist, dass

die interne Funktionsweise und Implementierung der Schätzung der Phasenverschiebung nicht bekannt ist und deshalb nur Vermutungen über genutzte Technologien aufgestellt werden konnten. Ausschließlich über die Funktionsweise der Kinect v2 war mehr bekannt, da sich Arbeiten mit der detaillierten Analyse des Systems beschäftigten [GVS18]. Die Simulation des Fehlers der Kinect v2 war im Rahmen dieser Arbeit allerdings nicht möglich.

Die Simulation des Multiple Path Fehlers ähnelt den Ergebnissen der vorangegangenen Analyse der Kameras, mit Ausnahme der Abweichung an dem Berührungspunkt der beiden orthogonal zueinander positionierten Oberflächen. Da im Rahmen der Arbeit kein direkter Vergleich eines exakt nachgebildeten Versuchsaufbaus innerhalb der Simulation möglich war, bleibt die Frage offen wie genau die Abweichung der simulierten Tiefenwerte den echten Messungen entspricht. Darüber hinaus waren die Rauheit und die optische Dichte der untersuchten Materialien nicht bekannt und mussten in der Evaluation geschätzt werden, weshalb kein genauer Vergleich möglich war. Neben den Eigenschaften der Oberflächen war außerdem die genutzte Wellenlänge der Infrarot LED nicht bekannt, weshalb keine genauen Reflexionseigenschaften der Strahlung an der Oberfläche bestimmt werden konnten, da diese von der Wellenlänge abhängig sind.

Wie auch in den vorangegangenen Arbeiten wurde auch in dieser Arbeit der Mixed Pixels Fehler simuliert. Im Kontrast zu vergleichbaren Arbeiten wurde hierfür allerdings die Simulation des Lens Scattering Fehlers priorisiert, der den Mixed Pixels Fehler nach sich zieht, was in der Analyse und der Evaluation gezeigt wurde. Vorangegangene Arbeiten verzichteten auf die Simulation des Lens Scattering und näherten ausschließlich den Mixed Pixels Fehler an, indem mehrere Samples pro Pixel berechnet wurden, was laut den Analysen dieser Arbeit den gemessenen Ergebnissen widerspricht.

Durch die Nutzung von OptiX zur Berechnung des Path Tracing Algorithmus war es möglich den Algorithmus zu beschleunigen und Eingaben des Nutzers während der Berechnung zu erlauben, was für den Einsatz einen Vorteil bietet, da mögliche Positionen für das Kamerasystem getestet und das Ergebnis innerhalb kurzer Zeit evaluiert werden kann. Außerdem können Parameter der Kamera zur Laufzeit modifiziert werden, was einer der Gründe in Kellers Dissertation ist, weshalb die Simulation der Lichtausbreitung nicht berücksichtigt wurde. Dieser Vorteil kommt allerdings mit dem Preis, dass die Simulation an Kanten ungenauer wird und der Einfluss der indirekten Beleuchtung daher fehlerhaft berechnet wird.

7.3 Ausblick

Die Simulation eines Time-of-Flight Sensors mittels Path Tracing lieferte Ergebnisse, die mit den Untersuchungen eine gute Übereinstimmung zeigen. Da es sich bei der Simulation der Lichtausbreitung um ein rechenintensives Problem handelt, profitiert der Algorithmus von der Implementierung und der parallelen Ausführung auf der GPU. Allerdings ist die Implementierung, die im Rahmen dieser Arbeit angefertigt wurde nicht echtzeitfähig. Durch das Verzichten auf hohe Pfadlängen kann die Berechnungszeit deutlich reduziert werden, was allerdings eine inkorrekte Berechnung der indirekten Beleuchtung zur Folge hätte. Besonders die hohe Komplexität der verwendeten Beleuchtungsmodelle führt zu

langen Ausführungszeiten, die durch die Nutzung von vereinfachten Annäherungen reduziert werden können. In zukünftigen Arbeiten können daher unterschiedliche Modelle zur Beleuchtung der Oberflächen untersucht werden, um die Berechnung zu beschleunigen und eine interaktive Anwendung bei gleichzeitig korrekter Berechnung der Lichtausbreitung zu ermöglichen. Darüber hinaus besteht die Möglichkeit eingemessene BRDFs zu verwenden und die Ergebnisse der genutzten Modelle zu evaluieren.

Im Rahmen dieser Arbeit wurde der Einfluss der Temperatur auf die Tiefenwerte zwar untersucht, allerdings nicht simuliert. Zukünftige Arbeiten könnten sich daher im Detail mit dem Einfluss der Temperatur beschäftigen und diese in der Simulation berücksichtigen. Zusätzlich wurde Bewegungsunschärfe im Rahmen der Arbeit nicht simuliert. OptiX bietet die Möglichkeit Bewegungen während der Berechnung des Bildes zu berücksichtigen, wodurch die Auswirkungen von Bewegungen der Kamera oder Objekte in der Szene in die Simulation einbezogen werden können. Dazu sind allerdings Details über die Architektur der verwendeten Sensorchips notwendig, da diese sich auf die Artefakte auswirken [LHK15].

Während der Untersuchung der Xtion 2 wurde festgestellt, dass die Tiefenwerte durch die künstliche Beleuchtung im Raum beeinflusst werden. Dabei ist zu beobachten, dass die Tiefenwerte zwischen den einzelnen Tiefenbildern stark variieren, was durch das Flimmern der Lampen mit einer Frequenz von 50 Hz verursacht wird. Dieses Flimmern war im Falle der Xtion 2 im Tiefenbild in Form von verfälschten Tiefenwerten erkennbar, während die Kinect v2 und die O3D303 davon unbeeinflusst blieben. Für den Einsatz wäre daher eine Simulation von künstlicher Beleuchtung von Vorteil, damit evaluiert werden kann, wie stark sich die Einflüsse der künstlichen Beleuchtung auf die Tiefenwerte unter verschiedenen Positionierungen auswirken.

Literaturverzeichnis

[But14] Thomas Butkiewicz. Low-cost coastal mapping using kinect v2 time-of-flight cameras. In *2014 Oceans - St. John's*. IEEE, sep 2014. doi:10.1109/oceans.2014.7003084.

[CCD06] Mark Claypool, Kajal Claypool, and Feissal Damaa. The effects of frame rate and resolution on users playing first person shooter games. *Proceedings of SPIE - The International Society for Optical Engineering*, 6071, 01 2006. doi:10.1117/12.648609.

[CCMDH07] James Christian Charles Mure-Dubois and Heinz Hügli. Optimized scattering compensation for time-of-flight camera - art. no. 67620h. September 2007. doi:10.1117/12.733961.

[CCMDH09] James Christian Charles Mure-Dubois and Heinz Hügli. Time-of-flight imaging of indoor scenes with scattering compensation. January 2009.

[Cre88] Katherine Creath. V phase-measurement interferometry techniques. *Progress in optics, 26*, 1988.

[CT81] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphicsa reflectance model for computer graphics. In *ACM SIGGRAPH 1981*, 1981.

[FSK+14] Daniel Freedman, Yoni Smolin, Eyal Krupka, Ido Leichter, and Mirko Schmidt. SRA: Fast removal of general multipath for ToF sensors. In *Computer Vision – ECCV 2014*, pages 234–249. Springer International Publishing, 2014. doi:10.1007/978-3-319-10590-1_16.

[GVS18] Silvio Giancola, Matteo Valenti, and Remo Sala. *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies*. Springer-Verlag GmbH, 2018.

[HF14] Christoph Hertzberg and Udo Frese. Detailed modeling and calibration of a time-of-flight camera. In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS - Science and and Technology Publications, 2014. doi:10.5220/0005067205680579.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Seventh Eurographics Workshop on Rendering*, 1996.

[JL10] Sonam Jamtsho and Derek Lichti. Modelling scattering distortion in 3d range camera. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38, January 2010.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH) 1986, ACM Press*, 1986.

[Kel15] Maik Keller. *Real-time Simulation of Time-of-Flight Sensors and Accumulation of Range Camera Data*. PhD thesis, University of Siegen, 2015.

[Lam60] J. H. Lambert. *Photometria Sive De Mensura Et Gradibus Luminis, Colorum Et Umbrae*. Eberhard Klett, 1760.

[LHK15] Martin Lambers, Stefan Hoberg, and Andreas Kolb. Simulation of time-of-flight sensors for evaluation of chip layout variants. *IEEE Sensors Journal*, 15(7):4019–4026, jul 2015. doi:10.1109/jsen.2015.2409816.

[Li14] Larry Li. Time-of-flight camera - an introduction. *Texas Instruments-Technical White Paper*, 2014.

[Mei12] Stephan Meister. Photon mapping based simulation of multipath reflection artifacts in time-of-flight sensors. 2012.

[MNK13] Stephan Meister, Rahul Nair, and Daniel Kondermann. Simulation of time-of-flight sensors using global illumination, 2013. doi:10.2312/pe.vmv.vmv13.033-040.

[MPBM03] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003.

[MSMJYBMC17] Rafael Muñoz-Salinas, Manuel J. Marin-Jimenez, Enrique Yeguas-Bolivar, and R. Medina-Carnicer. Mapping and localization from planar markers. *Pattern Recognition*, 73:158–171, January 2017. doi:10.13140/RG.2.2.31751.65440.

[NML+13] Rahul Nair, Stephan Meister, Martin Lambers, Michael Balda, Hannes Hofmann, Andreas Kolb, Daniel Kondermann, and Bernd Jähne. Ground truth for evaluating time of flight imaging. In *Lecture Notes in Computer Science*, pages 52–74. Springer Berlin Heidelberg, 2013.

[ON94] Michael Oren and Shree K. Nayar. Generalization of lamberts reflectance model. *SIGGRAPH*, 1994.

[Pad09] Paul Padley. *Waves and Optics*. 2009.

[PHJ16] Matt Pharr, Greg Humphreys, and Wenzel Jakob. *Physically Based Rendering*. Elsevier LTD, Oxford, 2016.

[PRS+10] Steven G. Parker, Austin Robison, Martin Stich, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, and Keith Morley. Optix: A general purpose ray tracing engine. In *ACM SIGGRAPH 2010 papers on - SIGGRAPH*. ACM Press, 2010. doi:10.1145/1833349.1778803.

[RRMSMC18] Francisco Romero Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018. doi:10.1016/j.imavis.2018.05.004.

[SGJMSMCMC15] Sergio S. Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015. doi:10.1016/j.patcog.2015.09.023.

[SHWH18] Shuochen Su, Felix Heide, Gordon Wetzstein, and Wolfgang Heidrich. Deep end-to-end time-of-flight imaging. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018. doi:10.1109/cvpr.2018.00668.

[TMT13] Alex Teichman, Stephen Miller, and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Robotics: Science and Systems 2013*, 06 2013. doi:10.15607/RSS.2013.IX.027.

[TS67] Kenneth E. Torrance and Ephraim M. Sparrow. Theory of off-specular reflection from roughened surfaces. *Journal of the Optical Society of America*, 1967.

[VMFGAL+17] Víctor Villena-Martínez, Andrés Fuster-Guilló, Jorge Azorín-López, Marcelo Saval-Calvo, Jeronimo Mora-Pascual, Jose Garcia-Rodriguez, and Alberto Garcia-Garcia. A quantitative comparison of calibration methods for RGB-d sensors using different technologies. *Sensors*, 17(2):243, jan 2017. doi:10.3390/s17020243.

[WS17] Oliver Wasenmüller and Didier Stricker. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Computer Vision – ACCV 2016 Workshops*, pages 34–45. Springer International Publishing, 2017. doi:10.1007/978-3-319-54427-4_3.

[Wya82] James C. Wyant. Interferometric optical metrology-basic principles and new systems. *Laser Focus with Fiberoptic Technology*, 1982.

[Wyn00] Chris Wynn. An introduction to brdf-based lighting. Technical report, NVIDIA Corporation, 2000.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und ausschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Unterschrift	Datum