Optimierung von PMD-basierten Tiefenkarten mittels Moving-Least-Squares-Verfahren

Masterarbeit

im Fach

Mathematik
Anwendungsfach Informatik

vorgelegt von

Stefan Hesse
Matrikelnummer: 632627
am 16.01.2012

Angefertigt am:

Lehrstuhl für Computergraphik und Multimediasysteme
Naturwissenschaftlich-Technische Fakultät
Universität Siegen

Betreuer:

Prof. Dr. Andreas Kolb
Prof. Dr. Hans-Jürgen Reinhardt

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Siegen, den 16.01.2012

Abkürzungsverzeichnis

LS	Least-Squares
WLS	Weighted-Least-Squares
MLS	Moving-Least-Squares
PMD	Photonic Mixer Device
RDO	RayDepthOptimizer - Optimierung durch den Schnittpunkt einer lokalen Approximation mit Sehstrahl.
PDO	PixelDepthOpitmizer - Optimierung durch die Minimierung des Fehlers der MLS-Projektion.
SEG	Segmentierung der Punktwolke
MLSS	Moving-Least-Squares-Schritt
P	Menge der Punkte $p \in \mathbb{R}^3$ die eine Oberfläche repräsentieren
S	Oberfläche, die durch eine Punktmenge $P \subset \mathbb{R}^3$ repräsentiert wird
$\tilde{S}$	Approximation von S
Θ	Gewichtsfunktion
$\|\cdot\|$	euklidische Distanz
$\langle \cdot , \cdot \rangle$	Skalarprodukt
Π_m^n	Menge der Polynome vom Grad m im $\mathbb{R}^n$
$\vec{\mathbf{n}}$	orientierter Richtungsvektor
$\mathbb{S}^2$	2-Sphäre
$\mathbb{P}^2$	projektive Ebene
$\|M\|$	Anzahl der Elemente einer Menge M

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die dreidimensionale Erfassung von Objekten im Raum hat in vielen Bereichen des täglichen Lebens Einzug erhalten, z. B. im Automotive-Bereich, der Medizintechnik, der Robotik, und der industrielle Bildverarbeitung.

Um eine solche dreidimensionale Objekterfassung zu ermöglichen, zeichnet eine spezielle Kamera die Entfernungen des Objektes von der Kamera auf. Diese Entfernungen werden in einer sogenannten Tiefenkarte gespeichert, ein Foto der Szene, welches in den jeweiligen Pixeln keine Farb- sondern Tiefeninformationen enthält (siehe Abbildung 4.1a). Daraus können dann Punkte im $\mathbb{R}^3$ generiert werden, die der aufgezeichneten Szene entsprechen.

Für die Aufzeichnung der Tiefenkarten werden vor allem Time-Of-Flight-Kameras, die auf dem Lichtlaufzeitverfahren basieren, eingesetzt. Die bekanntesten „Kameras“, die dieses Verfahren verwenden sind Laserscanner. In dieser Arbeit werden PMD-Kameras eingesetzt, die gegenüber den genaueren Laserscannern dadurch im Vorteil sind, dass sie schneller und günstiger ist.

Bei der Situation, die hier zu Grunde liegt, zeichnen mehrere dieser PMD-Kameras ein Objekt aus verschiedenen Perspektiven auf. Die Überlappungsbereiche der Tiefenkarten dieser Kameras weisen Inkonsistenzen auf. Hierauf und auf das Ziel dieser Arbeit, nämlich die Verringerung dieser Inkonsistenzen durch geeignete Optimierung der Tiefenkarten, wird in Kapitel 2 eingegangen.

Kapitel 3 gibt einen Überblick über andere Arbeiten, auf die hier zurückgegriffen bzw. aufgebaut wird.

Danach folgt ein Kapitel über einige Grundlagen, die im späteren Verlauf benötigt werden.

Das Kernstück dieser Arbeit, Kapitel 5, befasst sich mit dem Moving-Least-Squares-Approximationsverfahren, welches zur Optimierung der Tiefenkarten eingesetzt wird.

Im anschließenden Kapitel werden einige Details zur Implementierung dargelegt und die eingesetzten Algorithmen beschrieben. Auch die dabei verwendeten Minimierungsverfahren werden hier erläutert.

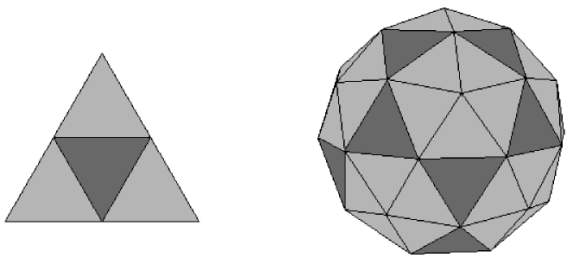
Es folgt Kapitel 7 mit den Ergebnissen dieser Arbeit und schließlich eine Zusammenfassung im letzten Kapitel.

Kapitel 2

Problemstellung

Die vorliegende Problemstellung bezieht sich auf eine Situation, die im Rahmen einer Forschungsarbeit am Lehrstuhl Computergrafik von Prof. Dr. Kolb von Bedeutung ist. In der gegebenen Situation sind mehrere Kameras so auf einer diskreten Sphäre positioniert, dass sie auf den Sphärenmittelpunkt ausgerichtet sind. Konkret wurden für diese Arbeit 42 Kameras verwendet, die auf den Ecken einer geodätischen Sphäre positioniert sind. Die geodätische Sphäre entsteht aus einem Ikosaeder, dessen Flächen in vier Dreiecke unterteilt werden (siehe 2.1). Durch die Projektion der Ecken dieser Dreiecke vom Ikosaeder-Mittelpunkt aus auf die Oberfläche der Umkugel entstehen neue Dreiecke, die in ihrer Gesamtheit eine geodätische Sphäre bilden.

Innerhalb dieser Kamerasphäre befindet sich ein Objekt, das von jeder Kamera auf der Sphäre aus einer anderen Perspektive aufgezeichnet wird. Die Menge der Kameras und deren Ausrichtung auf den Sphärenmittelpunkt sorgen dafür, dass sich Überschneidungen der Kamerabilder ergeben. Diese Redundanzen werden



(a) Unterteilung eines Dreiecks des Ikosaeders

(b) Geodätische Sphäre mit 42 Ecken

Abbildung 2.1

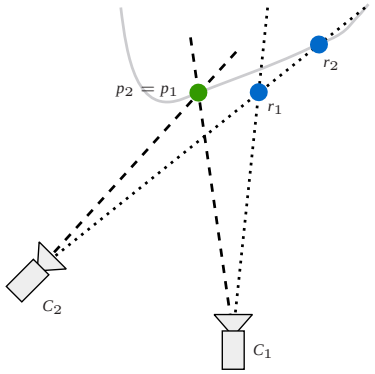


Abbildung 2.2: Die Punkte r_1 und p_1 wurden durch die Tiefenkarte von C_1 bestimmt. Von der Kamera C_2 aus gesehen entspricht der Tiefenwert in Richtung p_1 genau der Entfernung von C_2 zu p_1. Hier sind die Tiefendaten der beiden Kameras konsistent. Anders sieht es im Punkt r_1 aus: Die in diese Richtung durch C_2 gemessene Tiefe unterscheidet sich deutlich von der Distanz zwischen C_2 und r_1.

im Folgenden bei der Optimierung der Kamerabilder ausgenutzt.

Hierbei kommen allerdings keine gewöhnlichen Kameras, sondern PMD-Kameras zum Einsatz, die im nachfolgenden Kapitel beschrieben werden.

2.1 Ziel der Arbeit

Die PMD-Kameras auf der Sphäre liefern Tiefenkarten von einem Objekt innerhalb der Sphäre. Diese Tiefenkarten sind inkonsistent. Inkonsistent heißt in diesem Zusammenhang, dass die zu einem Punkt im $\mathbb{R}^3$ gehörigen Tiefeninformationen aus mehreren Tiefenkarten nicht übereinstimmen. Abbildung 2.2 zeigt ein Beispiel dieser Inkonsistenz. Der Punkt $r_1 \in \mathbb{R}^3$ wurde aus den Tiefeninformationen der Kamera C_1 bestimmt. Die Kamera C_2 hat in Richtung r_1 jedoch einen größeren Tiefenwert aufgezeichnet, so dass der daraus ermittelte Punkt $r_2 \in \mathbb{R}^3$ deutlich von r_1 abweicht. Hier liegen also fehlerhafte Tiefendaten vor, denn im Idealfall sollten r_1 und r_2 übereinstimmen. Solche Fehler entstehen durch die in Kapitel 4.1 beschriebenen Probleme wie Rauschen oder Flying Pixel.

Ziel dieser Arbeit ist es, ein Moving-Least-Squares-basiertes Verfahren zu entwickeln, welches die durch die PMD-Kameras aufgezeichneten Daten optimiert. Dadurch sollen die Inkonsistenzen zwischen den Aufnahmen der einzelnen Kameras reduziert werden. Der Vorteil der gegebenen Situation ist, dass sich die Bildbereiche der Kameras stark überschneiden und die so entstehenden Redundanzen der Tiefendaten eine Optimierung derselben erleichtern.

Kapitel 3

Verwandte Arbeiten

Liegen Daten in Form einer Punktwolke, also einer Punktmenge im $\mathbb{R}^n$ vor, gibt es verschiedene Verfahren, diese Daten global durch eine Funktion zu approximieren. Handelt es sich bei der Punktwolke allerdings um die Repräsentation einer Fläche im Sinne der Differentialgeometrie, so gibt es im Allgemeinen keine Möglichkeit diese global zu approximieren. Daher ist ein Verfahren nötig, welches die Punktwolke lokal approximiert. Häufig werden Gitter verwendet, die den Definitionsbereich in Zellen einteilen und für jede Zelle lokal ein Flächenstück bestimmt (siehe [Lev03]). Die Flächenstücke aller Zellen zusammen bilden dann die gesamte Fläche. Dies hat den Nachteil, dass die Fläche an den Stellen, wo sie von einer Zelle des Gitters in die andere übergeht nicht stetig ist.

Ein Ansatz, um dieses Problem zu lösen, ist das Moving-Least-Squares-Verfahren (kurz MLS-Verfahren). Es approximiert die Fläche in jedem Punkt lokal, bildet aber eine glatte Fläche, die MLS-Fläche. Die Idee zu dieser Methode wird von Levin in [Lev03] beschrieben. Er definiert einen Projektionsoperator, der zwei Funktionen erfüllt. Zum einen wird durch diesen die approximierte Fläche beschrieben und zum anderen kann er einen Punkt aus der näheren Umgebung der Fläche auf diese projizieren. Alexa et al. führen diese Idee in [ABCO$^+$01, ABCO$^+$03] in Form eines iterativen Algorithmus aus.

In [AK04b] zerlegen Amenta und Kil das für die Definition der MLS-Fläche verwendete Fehlerfunktional in eine Energiefunktion und ein Vektorfeld, was eine detailliertere Betrachtung des MLS-Fehlers ermöglicht. Außerdem geben Amenta und Kil in [AK04a] eine weitere Definition der MLS-Fläche an, die zwar einen größeren Definitionsbereich hat aber auch sehr rechenintensiev ist.

Adamson und Alexa verwenden in [AA03b, AA03a] eine implizite Beschreibung der Fläche für eine Ray-Tracing-Methode. Dabei wird ein Punkt aus dem Definitionsbereich der MLS-Fläche in Richtung eines Strahls, also einer Geraden im $\mathbb{R}^3$ auf die Fläche projiziert.

In [AA04b] wird ein Projektionsverfahren erläutert, welches in der Lage ist, Punkte orthogonal auf die MLS-Fläche zu projizieren.

Ist in der durch die Punktwolke repräsentierten Fläche eine Kante enthalten, so wird diese bei der MLS-Approximation geglättet. Um solche Kanten zu finden, segmentieren Fleishman et al. [FCOS05] die Punktmenge und erzeugen dann eine stückweise stetige Fläche.

Die Tiefenkarten der PMD-Kameras enthalten sogenannte Flying Pixel. Diese in Kapitel 4.1 beschriebenen fehlerhaften Pixel sollten erkannt werden. In [SK10] wird ein Verfahren beschrieben, welches durch die Segmentierung der Zeilen und Spalten der Tiefenkarte Tiefensprünge erkennt und somit die Flying Pixel klassifizieren kann.

In den oben beschriebenen Arbeiten werden Ansätze erläutert, die Flächen aus „sauberen“, also wenig bis gar nicht verrauschten Punktwolken rekonstruieren. Auf Basis dieser Ansätze wird in dieser Arbeit ein Verfahren entwickelt, welches Tiefenkarten, die eine hohe Fehlerquote aufweisen optimiert. Die Herausforderungen sind dabei das Rauschen in den Tiefenkarten sowie die Flying Pixel. Außerdem erfolgt die Optimierung der Tiefenwerte in eine ausgezeichnete Richtung.

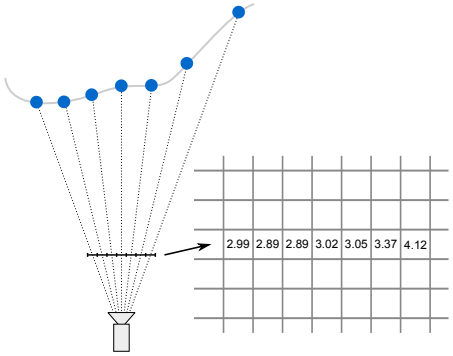
Kapitel 4

Grundlagen

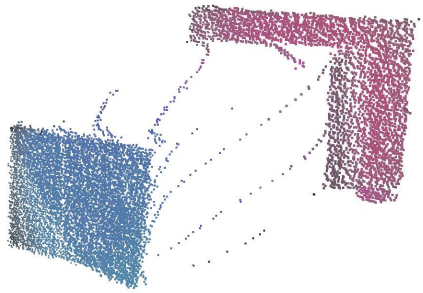
4.1 PMD-Kamera

Das Kürzel PMD steht für Photomischdetektor (engl.: Photonic Mixer Device). Es handelt sich dabei um einen Sensor, dessen Funktionalität auf dem Lichtlaufzeitverfahren (engl.: Time-Of-Flight (TOF)) basiert. Bei diesem Verfahren werden Lichtimpulse von der Kamera ausgesandt, durch das aufzuzeichnende Objekt reflektiert und von dem PMD-Sensor aufgenommen. Dieser misst die Zeit, welche der Lichtimpuls für diesen Vorgang benötigt und bestimmt aus dieser Information die Entfernung des Objektes von der Kamera. Damit ist die PMD-Kamera in der Lage, Tiefenkarten von der vorliegenden Szene zu erstellen, bei denen jeder Pixel der Tiefenkarte einen Wert für die Entfernung des Objektes in der entsprechenden Richtung enthält (siehe Abbildung 4.1a).

Diese Technik hat allerdings einige Nachteile. Die PMD-Kameras liefern aufgrund technischer Einschränkungen nur recht grobe Auflösungen. Die dieser Arbeit zugrunde liegenden Tiefenkarten haben beispielsweise eine Auflösung von 160×120 Pixeln. Des Weiteren liefert der PMD-Sensor für Pixel, die einen Bereich abbilden der Unstetigkeiten enthält, fehlerhafte Tiefeninformationen, sogenannte Flying Pixel. Diese enthalten einen Entfernungswert, der entweder näher am Hintergrund der Szene oder näher an der Kamera liegt als das tatsächliche Objekt. In Bild 4.1b ist ein Beispiel zu sehen. Auch Pixel, die keine Objektkanten abbilden, können kleine Abweichungen von der tatsächlichen Tiefe aufweisen. Dieses Rauschen entsteht unter anderem durch reflektierende Oberflächen und die Temperaturempfindlichkeit des PMD-Sensors (siehe [KK09]). Eine weitere Ein-



(a) Beispiel für eine Tiefenkarte



(b) Ein PMD-Bild einer ebene Fläche vor einer Wand weist an den Kanten Flying Pixel auf (Bsp. aus [KK09])

Abbildung 4.1

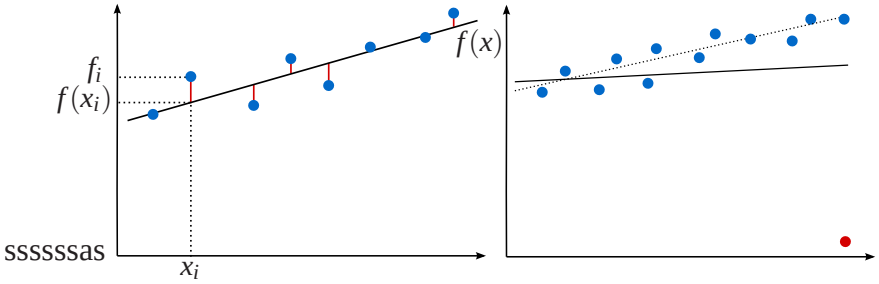
schränkung der PMD-Technik ist die maximale Entfernung, die erfasst werden kann. Bei aktuellen Modellen beträgt diese ca. sieben Meter.

Trotz dieser Nachteile haben PMD-Kameras auch positive Eigenschaften. Im Vergleich zu 3D-Laserscannern sind die Kosten für eine PMD-Kamera deutlich geringer. Außerdem kann die PMD-Kamera eine Szene komplett erfassen, also für jeden Pixel zeitgleich, wohingegen der Laserscanner die Entfernungsmessung für die einzelnen Pixel sukzessive vornimmt.

In der gegebenen Situation werden Kamerasphären verwendet, die beispielsweise 42 Kameras umfassen. Da zurzeit noch keine Datensätze aus einem derartigen Kameraaufbau verfügbar sind und es sehr aufwendig ist, diesen Versuchsaufbau in Betrieb zu nehmen, basieren die Ergebnisse dieser Arbeit auf simulierten Daten. Es wird der von Maik Keller und Prof. Kolb [KK09] entwickelte Simulator verwendet. Dieser simuliert die PMD-Kameras und kann so aus jeder Perspektive Tiefenkarten von einer dreidimensionalen Szene erzeugen. Auch die Flying Pixel und das Rauschen werden simuliert, so dass für diese Arbeit geeignete Testdaten generiert werden können.

4.2 Methode der kleinsten Quadrate

Bei der Auswertung und Interpretation einer Reihe von Messdaten wird häufig die Ausgleichsrechnung verwendet. Dabei soll ein Polynom bestimmt werden, welches die Menge dieser Messdaten möglichst gut approximiert. Dazu eignet sich die Methode der kleinsten Fehlerquadrate (engl.: Least-Squares), im Folgenden



(a) Die vertikalen Abstände zwischen den Punkten und dem Ausgleichspolynom sind hier rot gekennzeichnet.

(b) Die Abbildung zeigt zwei Ausgleichsgeraden für eine Punktmenge im $\mathbb{R}^2$: Die gestrichelte Linie ist das Ausgleichspolynom für die blauen Punkte, die schwarze Linie ist das Polynom für alle Punkte, inklusive dem roten Ausreißer.

Abbildung 4.2

mit LS-Verfahren bezeichnet. Bei dieser Methode wird ein Fehlerfunktional aus der Summe der quadratischen Abstände der Datenpunkte zum gesuchten Polynom gebildet. Die dabei verwendeten Abstände sind die vertikalen Distanzen der Punkte zum Polynom (siehe Abbildung 4.2a). Dieses Fehlerfunktional wird in Abhängigkeit von dem Ausgleichspolynom minimiert. Also ergibt sich ein Polynom, dessen quadratische Abstände zu den einzelnen Datenpunkten minimal sind. Das Minimierungsproblem wird im Folgenden formal beschrieben und gelöst.

Problemformulierung:

Eine Punktwolke, die aus l Tupeln der Form $(x_i, f_i) \in \mathbb{R}^{n-1} \times \mathbb{R}$ besteht soll durch ein lineares Polynom f vom Grad m approximiert werden. Also ist das Fehlerfunktional

$$e_{LS} = \sum_{i=1}^{l} \|f(x_i) - f_i\|^2$$

in Abhängigkeit von $f \in \Pi_m^n$ zu minimieren. Durch die Zerlegung von $f(x) = c_0 b_0(x) + c_1 b_1(x) + ... + c_k b_k(x)$ in seine Monome $b_j(x)$ und die zugehörigen unbekannten Koeffizienten c_j kann das Polynom in Vektorschreibweise dargestellt werden:

$$f(x) = b(x)^T c. \tag{4.1}$$

Dabei ist $b(x) = (b_1(x), ..., b_k(x))^T$ und $c = (c_1, ..., c_k)^T$. Beispielsweise hat $b(x)$ für ein Polynom vom Grad $m = 2$ und dem Definitionsbereich $\mathbb{R}^2$ die Form $b(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^T$. Diese Schreibweise ermöglicht die Bildung eines linearen Gleichungssystems zur Lösung des Minimierungsproblems

$$p(x) = \underset{f \in \Pi_m^n}{\operatorname{argmin}} e_{LS}.$$

Lösung des Minimierungsproblems:

Die notwendige Bedingung zur Lösung des Minimierungsproblems lautet bekanntlich

$$\nabla e_{LS} = 0. \tag{4.2}$$

Hierbei ist $\nabla = (\frac{\partial}{\partial c_1}, ..., \frac{\partial}{\partial c_k})^T$ und $\frac{\partial}{\partial c_j}$ ist die partielle Ableitung in Richtung des gesuchten Koeffizienten c_j. Wird die Gleichung (4.2) komponentenweise betrachtet, so ergibt sich:

$$\frac{\partial e_{LS}}{\partial c_j} = 0 \qquad \Leftrightarrow \qquad \sum_i 2b_j(x_i)(b(x_i)^T c - f_i) = 0$$

Die Bedingung (4.2) schreibt sich dann

$$\begin{aligned} &\sum_i 2b(x_i)(b(x_i)^T c - f_i) = 0 \\ \Leftrightarrow &\sum_i b(x_i)b(x_i)^T c - b(x_i)f_i = 0. \end{aligned}$$

Durch Matrix-Vektorschreibweise und Umsortierung ergibt sich unter Verwendung der Matrix $B_i = b(x_i)b(x_i)^T$ das lineare Gleichungssystem

$$\sum_i B_i c = \sum_i b(x_i)f_i \tag{4.3}$$

welches durch

$$c = \left(\sum_i B_i\right)^{-1} \sum_i b(x_i)f_i \tag{4.4}$$

gelöst wird. Ist $B = \sum_i B_i$ nicht singulär, so liefert (4.4) in (4.1) eingesetzt, eine Lösung für das Minimierungsproblem.

Ein großes Problem der Methode der kleinsten Fehlerquadrate sind Ausreißer.

Dies sind fehlerhafte Datenpunkte, wie z.B. Messfehler in einer Reihe von Messdaten, die stark von dem restlichen Datensatz abweichen. Ausreißer beeinflussen das Approximationspolynom recht stark, wie Abbildung 4.2b veranschaulicht. Eine Möglichkeit, um die Methode der kleinsten Fehlerquadrate gegen Ausreißer resistent zu machen, ist die Verwendung von Gewichten, wie es im folgenden Kapitel erläutert wird.

4.3 Weighted-Least-Squares

Das LS-Verfahren liefert eine globale Approximation einer Punktwolke. Es werden also alle Punkte gleichermaßen berücksichtigt. Wenn allerdings nur ein lokaler Bereich von Interesse ist, oder Ausreißer einen geringeren Einfluss haben sollen, so bietet sich eine lokale Approximation an.

Dazu wird eine Stützstelle $\bar{x} \in \mathbb{R}^{n-1}$ festgelegt, die das Zentrum des lokalen Definitionsbereichs darstellt. In einer Umgebung $K(\bar{x})$ um $\bar{x}$ soll die gesuchte Fläche bzw. die Punktmenge die diese repräsentiert, approximiert werden. Alle Punkte, die außerhalb von $K(\bar{x})$ liegen, haben dann keinen Einfluss auf die Approximation. Um die Methode der kleinsten Quadrate auf diese Situation anzupassen, wird eine Gewichtsfunktion eingeführt. Diese bewertet die Summanden des LS-Fehlerfunktionals so, dass nur die Funktionswerte, deren Stützstellen in der Umgebung $K(\bar{x})$ liegen, berücksichtigt werden. Das Verfahren heißt „gewichtete Methode der kleinsten Quadrate“ bzw. „Weighted-Least-Squares-Methode“ und wird im Folgenden mit WLS-Verfahren abgekürzt. Der zu minimierende WLS-Fehler schreibt sich für l Stützstellen wie folgt:

$$e_{WLS} = \sum_{i=1}^{l} \|f_{\bar{x}}(x_i) - f_i\|^2 \Theta(\|\bar{x} - x_i\|).$$

Hierbei ist Θ eine Gewichtsfunktion. Das gesuchte Approximationspolynom $f_{\bar{x}}$ hängt von der fixen Stützstelle $\bar{x}$ ab und hat in Vektorschreibweise die Form

$$f_{\bar{x}}(x) = b(x)^T c(\bar{x}).$$

Wie bei der LS-Methode erhält man die Lösung

$$p_{\bar{x}}(x) = \underset{f_{\bar{x}} \in \Pi_m^n}{\operatorname{argmin}} e_{WLS}$$

des Minimierungsproblems durch die Ableitungen nach $c(\bar{x})$. Das zu lösende Gleichungssystem

$$\sum_i \Theta(d_i) B_i c(\bar{x}) = \sum_i \Theta(d_i) b(x_i) f_i$$

unterscheidet sich von dem des LS-Verfahrens (4.3) nur durch die Gewichte $\Theta(d_i)$, wobei $d_i = \|\bar{x} - x_i\|$. Der Koeffizientenvektor hat dann die Form:

$$c(\bar{x}) = \left(\sum_i \Theta(d_i) B_i \right)^{-1} \sum_i \Theta(d_i) b(x_i) f_i \tag{4.5}$$

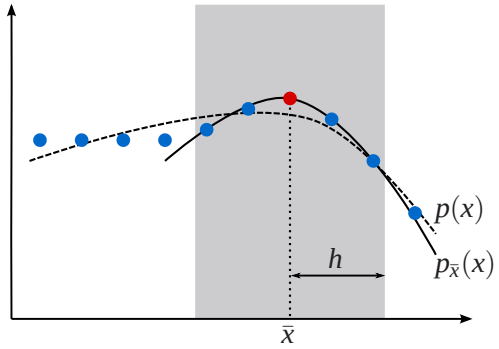


Abbildung 4.3: Vergleich zwischen LS und WLS: Das gestrichelt eingezeichnete Polynom wurde mit dem gewöhnlichen LS-Verfahren bestimmt, während das andere Polynom das Ergebnis der WLS-Methode ist. Der graue Bereich kennzeichnet die lokale Umgebung $K(\bar{x})$, die den Definitionsbereich zur Ermittlung des Approximationspolynoms bestimmt.

Gewichtsfunktionen

Im vorherigen Abschnitt bezieht sich die Gewichtung auf eine Stützstelle $\bar{x} \in \mathbb{R}^{n-1}$ aus dem Definitionsbereich. Eine andere Möglichkeit, die in dieser Arbeit verwendet wird, ist die Gewichtung in Abhängigkeit von den euklidischen Distanzen

$$d_i = \|\bar{p} - p_i\|.$$

Dabei sind $\bar{p} \in \mathbb{R}^n$ das Zentrum des lokalen Bereichs und $p_i = (x_i, f_k) \in \mathbb{R}^n$ die gegebenen Messdaten.

Es gibt einige Möglichkeiten, eine Gewichtsfunktion zu definieren. Die in der Literatur am häufigsten verwendete Funktion ist die Gaußfunktion

$$\Theta(d) = e^{-\frac{d^2}{h^2}}.$$

Der Parameter h wird verwendet, um den lokalen Bereich festzulegen. Näheres zu dieser und anderen Gewichtsfunktionen sowie deren Vor- und Nachteile werden in Kapitel 5.3 erläutert.

4.4 Bestimmung lokaler Koordinatensysteme mittels Kovarianzen

In dieser Arbeit soll ein Moving-Least-Squares-basiertes Approximationsverfahren einen fehlerbehafteten Datensatz, der eine Fläche repräsentiert, optimieren. Dabei wird für jeden Datenpunkt $p \in \mathbb{R}^3$ ein lokales Approximationspolynom bestimmt. Für diese lokalen Approximationen muss jeweils ein lokales Koordinatensystem konstruiert werden. Dazu wird, wie bei der Hauptkomponentenanalyse (siehe [Han02, Kap. 5.2]), die Varianz der Punktmenge in Bezug zum Ursprung eines lokalen Koordinatensystems betrachtet.

Ein solches Koordinatensystem wird durch eine Hyperebene und deren Normale bestimmt. Dazu sei $P \subset \mathbb{R}^3$ eine Menge von Datenpunkten, die eine Fläche S repräsentiert und $r \in P$ der Mittelpunkt der lokalen Umgebung. Die gesuchte Hyperebene soll durch r verlaufen und S möglichst gut approximieren. Die Normale der Ebene zeigt dann in Richtung der geringsten Varianz der Punktmenge. Damit diese Hyperebene eine lokale Approximation ist wird eine Gewichtung der Punkte in Abhängigkeit von deren Distanz zu r vorgenommen. Die Hyperebene mit dem kleinsten WLS-Fehler entspricht der gesuchten linearen Approximation.

Um diese Ebene zu finden reicht es, die Normale der Ebene im Punkt r zu bestimmen. Die dazu benötigte Kovarianzmatrix ergibt sich aus den gewichteten Distanzen der Punkte $p_i \in P$ zum Referenzpunkt r wie folgt:

$$B = \left(\sum_{i=1}^{|P|} (p_i - r)(p_i - r)^T \Theta(\|p_i - r\|) \right).$$

Die gesuchte Normale ist dann die Lösung des Minimierungsproblems

$$\underset{n}{\operatorname{argmin}}\, n^T B\, n$$

mit der Nebenbedingung

$$n^T n = 1.$$

Mit Hilfe eines Lagrange-Multiplikators λ wird daraus ein Problem ohne Nebenbedingung und es ergibt sich eine einfache Möglichkeit der Lösung des Extremwertproblems: Die Eigenvektoren der Kovarianzmatrix. Die Langrangefunktion hat folgende Form:

$$L(n, \lambda) = n^T B\, n - \lambda(n^T n - 1).$$

Die notwendigen Bedingungen für die Extremwerte

$$\frac{\partial}{\partial n} L(n, \lambda) = 2Bn - 2\lambda n = 0$$

und

$$\frac{\partial}{\partial \lambda} L(n, \lambda) = 1 - n^T n = 0$$

stellen ein Eigenwertproblem dar, wie folgende Umformung zeigt:

$$\begin{aligned} 2Bn - 2\lambda n = 0 \quad &\Leftrightarrow \quad Bn = \lambda n \\ 1 - n^T n = 0 \quad &\Leftrightarrow \quad \|n\| = 1 \end{aligned}$$

Die normierten Eigenvektoren der Kovarianzmatrix B liefern also jeweils einen Extremwert von $n^T B\, n$. Da eine symmetrische 3×3 Matrix drei Eigenvektoren und zugehörige Eigenwerte hat muss nun noch ein Eigenvektor bestimmt werden, der $n^T B\, n$ minimiert. Dies ist der Eigenvektor zum kleinsten reellen Eigenwert, denn es gilt:

$$n^T B\, n = n^T \lambda\, n = \lambda.$$

Der Eigenvektor, der im Punkt r die Richtung der geringsten Varianz angibt, wird im Folgenden als Normale der Punktmenge P im Punkt r bezeichnet.

Es kann vorkommen, dass die algebraische Vielfachheit des kleinsten Eigenwertes größer als eins ist. Im Beispiel in Abbildung 4.4 hat die Kovarianzmatrix

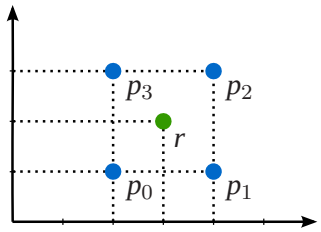


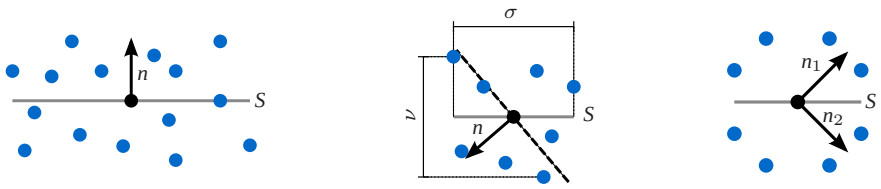
Abbildung 4.4: Beispiel für eine symmetrische Punktwolke.

mit fixer Gewichtsfunktion $\Theta = 1$ die Form

$$B = \sum (p_i - r)(p_i - r)^T = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$$

und offensichtlich den zweifachen Eigenwert $\lambda = 4$. Damit sind die Eigenvektoren zwei beliebige, linear unabhängige Vektoren, die den Eigenraum zu λ aufspannen. In einem solchen Fall kann keine eindeutige Normale für die gegebene Punktmenge ermittelt werden.

Dieser Fall tritt dann auf, wenn die Varianz der betrachteten Punktwolke in mindestens zwei orthogonal zueinander liegenden Richtungen minimal ist. Im Allgemeinen ist die Ausdehnung einer Punktwolke, die eine Fläche repräsentiert, orthogonal zur Fläche kleiner als in alle anderen Richtungen, so dass es eine eindeutige Normale gibt. Ist beispielsweise die Ausdehnung der Fläche in mindestens eine Richtung genau so klein wie das maximale Rauschen, so könnte ein zweifacher Eigenwert gefunden werden, wenn die Punktwolke entsprechend symmetrisch ist. Allerdings kann die in diesem Fall gefundene Normale ohnehin nichts über die Lage der Fläche im Raum aussagen (siehe Abbildung 4.5).



(a) (b) (c)

Abbildung 4.5: In (a) bildet der Eigenvektor zum kleinsten Eigenwert der Kovarianzmatrix eine gute Approximation der Normalen der Fläche *S*. Die Ausdehnung σ der Fläche *S*, bzw. der zugehörigen Punktwolke, in (b) entspricht dem maximalen Rauschen ν. In diesem Fall sagt *n* nichts über die Lage von *S* aus. Abbildung (c) zeigt eine symmetrische Punktwolke, bei der es zwei Eigenvektoren zum gleichen Eigenwert gibt.

4.5 Flying Pixel Detection

Wie bereits erwähnt, entstehen bei der Aufzeichnung der Tiefenkarten durch eine PMD-Kamera die sogenannten Flying Pixel. Das sind Pixel, die einen Bereich abdecken, der Tiefensprünge enthält. Also repräsentieren diese Pixel zum Teil ein Objekt und zum anderen Teil ein anderes Objekt, oder den Hintergrund. Mit Hintergrund ist hier eine Tiefe bzw. Entfernung von ca. sieben Metern gemeint, denn das ist die Reichweite, bis zu der die PMD-Technik Tiefeninformationen aufzeichnen kann.

In der Regel werden diese Flying Pixel eliminiert. Dies ist jedoch nicht die beste Art damit umzugehen, denn auch diese Pixel enthalten Informationen. Außerdem würde eine Tiefenkarte, die einen Tiefensprung zwischen zwei Objekten enthält dann Löcher aufweisen. Aus diesen Gründen sollen die Flying Pixel bei der Optimierung der Tiefendaten ebenfalls berücksichtigt werden. Dazu muss ein Flying Pixel zuerst als ein solcher identifiziert werden.

Ein Flying Pixel zeichnet sich in der Regel dadurch aus, dass die nähere Umgebung des daraus resultierenden Punkts im $\mathbb{R}^3$ sehr wenige oder gar keine Punkte aufweist. Diese Eigenschaft kann genutzt werden, um diese fehlerhaften Tiefeninformationen zu klassifizieren.

Sabov und Krüger greifen in [SK10] ein Verfahren von Hule et al. aus [HJS08] auf und verwenden es zur Klassifikation von Flying Pixel. Dabei werden die Differenzen zwischen dem Tiefenwert eines Pixels (x_r, y_r) und dem seiner Nachbarpixel untersucht. Der Mittelwert dieser Differenzen hat dann die Form:

$$s(x_r, y_r) = \frac{1}{(2WS+1)^2 - 1} \sum_{x=x_r-WS}^{x_r+WS} \sum_{y=y_r-WS}^{y_r+WS} |d(x,y) - d(x_r, y_r)|.$$

Hierbei legt *WS* (Abk. window size) die Nachbarschaftsgröße fest und $d(x,y)$ entspricht dem Tiefenwert an der Stelle (x,y) in der Tiefenkarte.

Sabov und Krüger vereinfachen dieses Verfahren, indem sie nur die Spalte und Zeile betrachten, welche den betrachteten Pixel enthalten:

$$s_C(x_r, y_r) = \frac{1}{2WS} \sum_{y=y_r-WS}^{y_r+WS} |d(x_r, y) - d(x_r, y_r)|$$

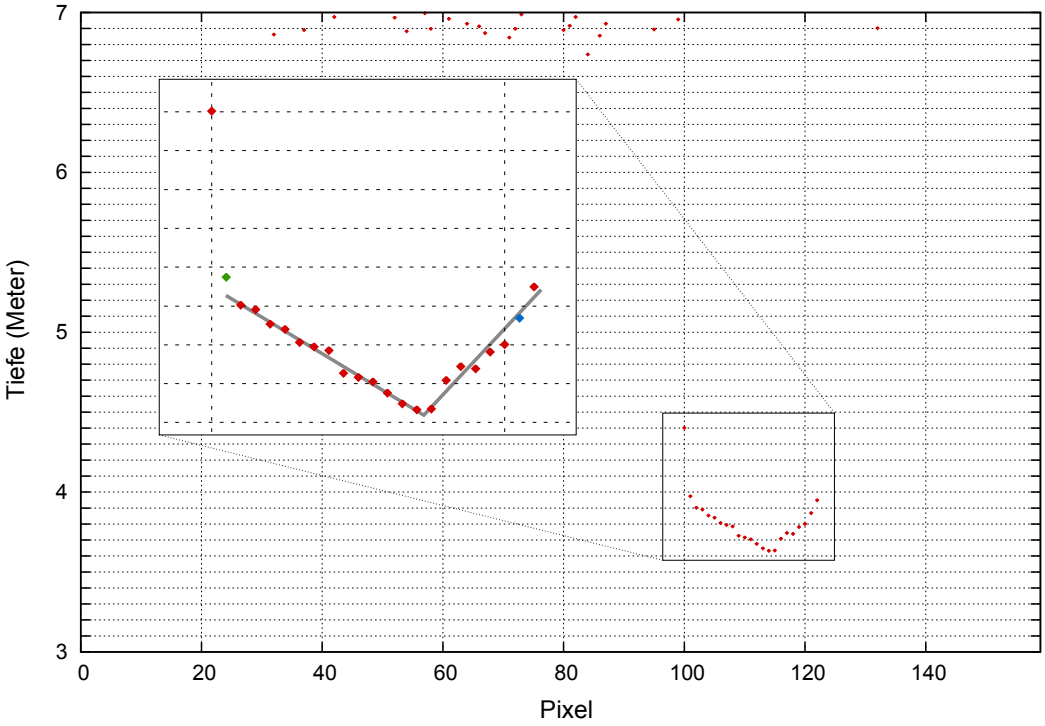


Abbildung 4.6: Hier sind die Tiefenwerte einer Zeile einer Tiefenkarte zu sehen. Der grün markierte Pixel ist ein Flying Pixel. Wird der Schwellenwert so festgelegt, dass er als solcher erkannt wird, so wird auch der blau gekennzeichnete Pixel als Flying Pixel markiert.

$$s_R(x_r, y_r) = \frac{1}{2WS} \sum_{x=x_r-WS}^{x_r+WS} |d(x, y_r) - d(x_r, y_r)|.$$

Sollte s_C oder s_R einen festgelegten Schwellenwert überschreiten, so handelt es sich bei (x_r, y_r) um einen Flying Pixel. Der hierzu notwendige Schwellenwert sollte ca. dem maximal vorkommenden Rauschen entsprechen, damit stark verrauschte Pixel nicht als Flying Pixel markiert werden.

Ein Problem stellen Flächen dar, die in einem sehr spitzen Winkel zur Kamera liegen (siehe Abbildung 4.6). Der festgelegte Schwellenwert sorgt hier ab einem bestimmten Winkel dafür, dass alle Pixel auf der entsprechenden Fläche als Flying Pixel erkannt werden. Um diesen Effekt zu verhindern, wird in dieser Arbeit für jeden als Flying Pixel erkannten Pixel geprüft, ob er auf einer ebenen Fläche liegt. Dazu wird sowohl die Zeile als auch die Spalte, in der sich der betrachtete Pixel befindet, untersucht. Sind in beiden Fällen (Zeile und Spalte) die Differenzen zwischen dem Tiefenwert des betrachteten Pixels und den Tiefenwerten der beiden Nachbarn betragsmäßig gleich, so handelt es sich lokal um eine Ebene.

Es kann weiterhin vorkommen, dass „gute“ Pixel, die sehr verrauscht sind,

als Flying Pixel markiert werden. Mit der obigen Verbesserung kommt dies nicht mehr so häufig vor und wird in Kauf genommen, da auch die Tiefenwerte der Flying Pixel im Rahmen der Optimierung korrigiert werden.

Für die spätere Optimierung ist es allerdings notwendig, die potentiellen Tiefenwerte für einen Flying Pixel grob zu bestimmen. Dazu wird zuerst entschieden, ob diese neuen Tiefenwerte aus der Zeile oder der Spalte die den betrachteten Pixel enthält, ermittelt werden. Enthält die Zeile in der Nachbarschaft des betrachteten Pixels weniger Flying Pixel als die Spalte, so wird diese verwendet, andernfalls wird die Spalte verwendet. Die benachbarten Pixel des Flying Pixel geben Aufschluss darüber, welche Distanz das Objekt links und rechts bzw. oberhalb und unterhalb des Flying Pixel hat. Dazu werden jeweils mehrere benachbarte Pixel in jeder Richtung betrachtet. Die Mittelwerte dieser benachbarten Pixel in beiden Richtungen können dann als Startwerte für die Optimierung verwendet werden.

Kapitel 5

Moving-Least-Squares zur Flächenrekonstruktion

Heute lassen sich durch diverse Techniken wie Laserscanner oder PMD-Kameras Tiefeninformationen einer Szene aufzeichnen. Die damit gewonnenen Daten können dann als Punktwolke im $\mathbb{R}^3$ betrachtet werden. Da diese Punkte eine oder mehrere Oberflächen repräsentieren ist es naheliegend, aus den Daten eine Fläche zu rekonstruieren bzw. eine Funktion zu finden, die diese Fläche beschreibt. Um diese Aufgabe zu lösen, wird hier das Moving-Least-Squares-Verfahren (kurz MLS), welches auf dem WLS-Verfahren beruht, verwendet.

Im ersten Abschnitt dieses Kapitels wird das grundlegende MLS-Verfahren beschrieben, welches auf dem WLS-Verfahren aufbaut, aber nicht auf einen lokalen Bereich beschränkt ist.

Das MLS-Verfahren ist die Grundlage eines Projektionsoperators, der eine Fläche definiert und Punkte aus der Umgebung der Fläche auf diese projizieren kann. Diese Projektion wird in Kapitel 5.2 beschrieben und es wird dessen Definitionsbereich untersucht.

Der dritte Abschnitt geht auf die Gewichtsfunktion ein, die das MLS-Verfahren mit sich bringt. Die Gewichtsfunktionen können, je nach Wahl des zugehörigen Parameters, Probleme verursachen. Unstetigkeiten oder kleine Verformungen der gesuchten Fläche können durch ungünstige Parameterwahl verloren gehen.

Um Unstetigkeiten, wie sie zum Beispiel an den Kanten eines Objektes vorkommen zu erhalten, wird im vierten Abschnitt eine Erweiterung des MLS-Verfahrens beschrieben. Dieses Verfahren basiert auf der Segmentierung lokaler Teil-

mengen der Punktwolke. Die dadurch rekonstruierten stetigen Teilflächen bilden dann die stückweise stetige Oberfläche des Objektes.

Der letzte Abschnitt beschreibt zwei Verfahren, die die Tiefenkarten der PMD-Kameras mit Hilfe der Erkenntnisse aus den vorherigen Abschnitten optimieren. Dabei werden die Sehstrahlen der PMD-Kameras betrachtet und die darauf liegenden Punkte entlang dieser Strahlen auf die gesuchte Fläche projiziert.

5.1 Moving-Least-Squares

Das WLS-Verfahren wurde bereits in Kapitel 4.3 beschrieben. Es ist zwar lokal genauer als eine globale Approximation durch das Verfahren des kleinsten Fehlerquadrates, allerdings werden die Punkte außerhalb des lokalen Bereichs gar nicht oder schlecht approximiert. Das MLS-Verfahren ändert dies. Um eine globale Approximation zu erhalten wird $\bar{x}$ nicht länger als fixer Referenzpunkt (wie in Kapitel 4.3) betrachtet. Stattdessen wird er über den gesamten Definitionsbereich bewegt, worauf sich auch die Bezeichnung **Moving**-Least-Squares begründet. Der Übergang $\bar{x} \mapsto x$ liefert die globale Approximation

$$p(x) = \lim_{\bar{x} \to x} p_{\bar{x}}(x),$$

wobei

$$p_{\bar{x}}(x) = \underset{f_{\bar{x}} \in \Pi_m^n}{\operatorname{argmin}} \sum_{i=1}^{l} \|f_{\bar{x}}(x_i) - f_i\|^2 \Theta(\|\bar{x} - x_i\|).$$

Das Polynom $p(x)$, welches sich aus den lokalen Funktionen $p_{\bar{x}}(x)$ zusammensetzt, ist stetig differenzierbar genau dann, wenn die Gewichtsfunktion stetig differenzierbar ist (siehe [Lev03, Lev98]).

5.2 MLS-Verfahren zur Flächenrekonstruktion

Die Interpolation oder Approximation einer Funktion aus Punkten im $\mathbb{R}^n$ ist ein weit verbreitetes Problem, für das es viele Lösungen gibt. Anders sieht es bei der Rekonstruktion von Flächen im $\mathbb{R}^n$ aus. Hierbei lässt sich im Allgemeinen kein globaler Definitionsbereich finden, so dass die Anwendung globaler Approximationsverfahren nicht möglich ist. Üblicherweise wird zur Lösung dieses Problems

ein Gitter verwendet, so dass lokal für jede Zelle dieses Gitters ein Flächenstück approximiert werden kann. Die Gesamtfläche wird dann aus der Summe dieser Teilflächen gebildet. Diese Fläche hängt allerdings von dem verwendeten Gitter ab und kann Unstetigkeiten an den Kanten der Gitterzellen aufweisen. Das hier beschriebene MLS-Approximationsverfahren kommt ohne ein Gitter aus. Es verwendet eine Projektion, die auf Techniken der Differentialgeometrie, nämlich ein lokales Koordinatensystem und eine lokale Abbildung zurückgreift.

Im Folgenden wird das MLS-Approximationsverfahren auf Basis des MLS-Verfahrens beschrieben und ein Projektionsoperator entwickelt, der jeden Punkt aus der Umgebung der durch die Punktmenge $P \subset \mathbb{R}^n$ beschriebenen Fläche S, auf die Approximation $\tilde{S}$ abbildet. Die durch diese Methode rekonstruierte Fläche $\tilde{S}$ wird als MLS-Fläche bezeichnet.

5.2.1 MLS-Projektionsoperator

Das MLS-Approximationsverfahren führt für jeden Punkt eine lokale Approximation durch. Dazu wird für jeden Punkt eine Projektion bestimmt. Eine Projektion ist eine idempotente lineare Abbildung eines Vektorraums in sich selbst. Als idempotent wird eine Funktion bezeichnet, die die Elemente ihrer Bildmenge auf sich selbst abbildet, also:

$$\mathcal{P}\left(\mathcal{P}(x)\right) = \mathcal{P}(x).$$

Bezogen auf die Rekonstruktion einer Oberfläche aus einer Punktwolke bedeutet dies, dass Punkte, die bereits auf der Fläche liegen durch die Projektion nicht weiter verändert werden.

Die Projektion besteht aus zwei Teilen. Zuerst wird für jeden Punkt $r \in P$, ein lokales Koordinatensystem definiert. Im zweiten Teil wird eine lokale Approximation der Punktmenge, auf Basis der zuvor bestimmten Koordinatensysteme, durchgeführt. Die Approximationspolynome liefern dann für jedes r den projizierten Punkt $\tilde{r}$ auf der gesuchten Fläche $\tilde{S}$.

Teil 1: Das lokale Koordinatensystem

Für jeden Punkt $r \in P$ wird eine Hyperebene gesucht, die mit ihrer Normalen das lokale Koordinatensystem bildet. Diese Hyperebene $H_r = \{x \in \mathbb{R}^n \mid \langle x, n\rangle = d\}$ im $\mathbb{R}^n$ soll die Fläche S, welche durch die Punktmenge $P \subset \mathbb{R}^n$ repräsentiert wird,

lokal approximieren. Gesucht werden dazu ein Punkt q auf H_r und eine Normale

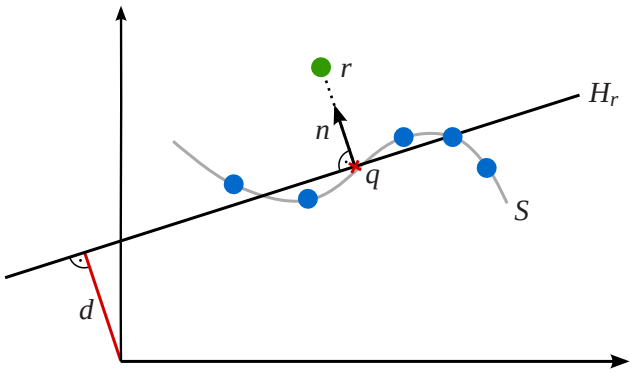


Abbildung 5.1: Für eine Hyperebene die den hier definierten Bedingungen für das lokale Koordinatensystem entspricht gilt $d = \langle n, q \rangle$.

$n \in \mathbb{R}^n$ der Ebene, die folgende Bedingungen erfüllen:

- n minimiert den WLS-Fehler

$$e_{H_r}(n, d) = \sum_{i=1}^{|P|} (\langle n, p_i \rangle - d)^2 \Theta(\|p_i - q\|), \tag{5.1}$$

  wobei Θ die Summanden $(\langle n, p_i \rangle - d)^2$ in Abhängigkeit von der Distanz der Punkte $p_i \in P$ zu q gewichtet,

- die Normale hat die gleiche Richtung wie die Gerade, die q und r verbindet:

$$\exists \alpha \in \mathbb{R},\ \alpha \neq 0 :\ n = \alpha(r - q),$$

- und die Richtungsableitung

$$\partial_n \sum_{i=1}^{|P|} (\langle n, p_i \rangle - d)^2 \Theta(\|p_i - q\|)$$

  ist, nach Auswertung an der Stelle q, gleich Null.

Die nach obigen Verfahren bestimmten Werte q und r werden durch die beiden Operatoren $Q(r) = q$ und $A(r) = n$ beschrieben. Dabei sei der Definitionsbereich U von Q und A eine Teilmenge von $\mathbb{R}^n$, so dass für jedes $r \in \mathbb{R}^n$ eine eindeutige Lösung q, n existiert, wobei die Strecke zwischen r und q komplett in U liegt.

Diese Einschränkung ist, wie weiter unten deutlich wird, aufgrund der Gewichtsfunktion notwendig.

Satz 5.2.1. *Für ein $r \in U$ sei $r^* \in U$ ein beliebiger Punkt auf der Geraden durch r und $Q(r) = q$. Es gilt:*

$$Q(r^*) = Q(r) \quad \textit{und} \quad A(r^*) = A(r).$$

Beweis. Die drei Bedingungen sorgen, zusammen mit dem eingeschränkten Definitionsbereich U, für die Existenz einer eindeutig bestimmten, ausgezeichneten Richtung n in Abhängigkeit vom Punkt r. Da zudem die entsprechende Richtungsableitung verschwindet, folgt für alle Punkte $r^* \in U$ auf der Geraden durch r und q der Satz. □

Der Satz gilt offensichtlich auch für den Punkt q. Daraus folgt, dass Q eine Projektion auf U ist:

$$Q(Q(r)) = Q(q) = q.$$

Die Ermittlung dieses lokalen Koordinatensystems ist ein nichtlineares Optimierungsproblem. Um es zügig zu lösen haben Alexa et al. in [ABCO$^+$01] ein iteratives Verfahren entwickelt. Dazu wird (5.1) mit $d = \langle n, q \rangle$ und $q = r + tn$ geschrieben:

$$e_{H_r}(n,t) = \sum_{i=1}^{|P|} \langle n, p_i - r - tn \rangle^2 \, \Theta(\|p_i - r - tn\|). \tag{5.2}$$

Diese Gleichung hängt jetzt von einer Richtung in Form der Normalen n und einer Distanz t ab. $e_{H_r}(n,t)$ wird nun abwechselnd in Abhängigkeit der beiden Parameter n und t minimiert. Ausgehend von dem initialen Wert $t = 0$ verläuft die Hyperebene durch den Punkt r und das Fehlerfunktional hat dann die Form:

$$\begin{aligned} e_{H_r}(n,0) &= \sum_{i=1}^{|P|} (n^T(p_i - r))^2 \, \Theta(\|p_i - r\|) \\ &= \sum_{i=1}^{|P|} n^T (p_i - r)(p_i - r)^T n \, \Theta(\|p_i - r\|) \\ &= n^T \left(\sum_{i=1}^{|P|} (p_i - r)(p_i - r)^T \Theta(\|p_i - r\|) \right) n \end{aligned}$$

$$= n^T B n. \tag{5.3}$$

Es wird jetzt die Normale n gesucht, die die Gleichung (5.3) minimiert. Wie in Kapitel 4.4 beschrieben, löst der Eigenvektor zum kleinsten Eigenwert der Kovarianzmatrix B dieses Problem.

Jetzt wird (5.2) unter Verwendung der soeben gewonnenen Normalen in Abhängigkeit von t minimiert. Hierbei handelt es sich um ein eindimensionales nichtlineares Problem. Anschaulich wird die Hyperebene H_r in Richtung der Normale verschoben bis das Fehlerfunktional minimalen Wert erreicht hat. Hierbei ist die Lösung nicht eindeutig, daher wird das kleinste t gewählt, welches (5.2) minimiert. Die Hyperebene mit dem kleinsten t hat offensichtlich den geringsten Abstand zum Referenzpunkt.

Mit dem neuen, fixen t lässt sich nun die Normale n verbessern. Durch n wird eine Richtung angegeben, und zwar die Richtung der Geraden durch q und r. Da $q = r + tn$ durch t und n bestimmt wird kann, in Abhängigkeit von q, minimiert werden. Alle möglichen q liegen auf einer Sphäre um r mit Radius t. Da nur geringe Abweichungen der Richtung n zu erwarten sind, und die Sphäre lokal durch die Hyperebene H_r approximiert wird, lässt sich ein Gradientenverfahren anwenden, um

$$e_{H_r}(n,q) = \sum_{i=1}^{|P|} \langle n, p_i - (r+tn)\rangle^2 \, \Theta(\|p_i - q\|) \tag{5.4}$$

in Abhängigkeit von q zu minimieren. Hierbei hängen die Gewichte nicht von dem gesuchten q ab, sondern von dem $q = r + tn$ aus dem vorherigen Schritt. Ohne diese Modifikation würde kein lokales Minimum bestimmt. Es würde ein $q \in H_r$ gefunden, welches so weit von der Punktmenge entfernt ist, dass die Gewichte näherungsweise Null sind.

Mit $n = \frac{q-r}{t}$ kann nun t verbessert werden. Durch die iterative Bestimmung von t und q ergibt sich schließlich die gesuchte Hyperebene H_r in Form der beiden Parameter q und n.

Teil 2: Die Projektion auf die MLS-Fläche

Der zweite Schritt der Projektion ist die Bestimmung eines lokalen Approximationspolynoms. Dazu werden alle Punkte p_i der Punktwolke in das zuvor bestimmte

lokale Koordinatensystem transformiert. Die orthogonale Projektion q vom Referenzpunkt r auf die Hyperebene H_r bildet den Ursprung des lokalen Koordinatensystems und die lokalen Koordinaten des Punktes p_i seien mit $(x_i, f_i) \in \mathbb{R}^{n-1} \times \mathbb{R}$ bezeichnet. Das Polynom $p \in \Pi_m^{n-1}$ vom Grad m, welches den WLS-Fehler

$$\min_{f \in \Pi_m^{n-1}} \sum_{i=1}^{|P|} \|f(x_i) - f_i\|^2 \Theta(\|p_i - q\|)$$

minimiert, liefert eine lokale Approximation der Fläche S. Die Abbildung des Referenzpunktes r auf das Approximationspolynom in Richtung der Normalen n liefert den Punkt $\tilde{r}$:

$$\mathcal{P}(r) = \tilde{r} = q + p(0)n.$$

Liegt das Ergebnis von $\mathcal{P}(r)$ genau wie r in U, so ist auch $\mathcal{P}$ eine Projektion auf U.

Beweis. Mit $q = Q(r)$ und $n = A(r)$ gilt:

$$\mathcal{P}(r) = Q(r) + p(0)A(r)$$

Es folgt:

$$\mathcal{P}(\mathcal{P}(r)) = Q(\mathcal{P}(r)) + p(0)A(\mathcal{P}(r))$$

Da $\mathcal{P}(r)$ per Definition auf der Geraden durch q und r liegt, gilt Satz 5.2.1 und wegen der Eindeutigkeit von Q und A in U folgt:

$$\mathcal{P}(\mathcal{P}(r)) = q + p(0)n = \mathcal{P}(r).$$

□

Die Menge der durch $\mathcal{P}$ projizierten Punkte repräsentieren die Fläche $\tilde{S}$, welche als MLS-Fläche bezeichnet wird. Levin zeigt in [Lev03] und [Lev98] das $\tilde{S}$ eine $(n-1)$-Fläche ist und das $\tilde{S} \in C^\infty$.

Da es in dieser Arbeit um dreidimensionale Daten geht, wird im Folgenden mit $n = 3$ gearbeitet. Die aus der Approximation resultierende Fläche $\tilde{S}$ ist dann eine 2-Fläche.

5.2.2 Analyse der MLS-Fläche

Für den ersten Teil des zuvor beschriebenen MLS-Verfahrens ist die Einschränkung des Definitionsbereichs wichtig. Es wird eine Umformulierung der Energiefunktion vorgenommen, um den Energiefehler und die Normalen getrennt voneinander zu betrachten.

Um diesen näher zu untersuchen wird eine Umformulierung vorgenommen, bei der die Energiefunktion und das Vektorfeld der Normalen getrennt voneinander betrachtet werden.

Der Definitionsbereich der Energiefunktion $e_{H_r}(n,d)$ in (5.1) ist $\mathbb{S}^2 \times \mathbb{R}$, wobei $\mathbb{S}^2$ die Menge der Richtungen im $\mathbb{R}^3$ ist. Die Funktion kann mit $d = \langle n, q \rangle$ in Abhängigkeit von n und q formuliert werden:

$$e_{MLS}(n,q) = \sum_{i=1}^{|P|} \langle n, p_i - q \rangle^2 \, \Theta(\|p_i - q\|).$$

Damit ist e_{MLS} unabhängig von r.

Sei nun der Vektor n mit einer Orientierung versehen und mit $\vec{\mathbf{n}}$ bezeichnet. Es ist offensichtlich, dass die Funktion $e_{MLS}(n,q)$ für die beiden Tupel $(\vec{\mathbf{n}}, q)$ und $(-\vec{\mathbf{n}}, q)$ den gleichen Wert annimmt. Also kann die Orientierung der Normalen hier unberücksichtigt bleiben. Der Definitionsbereich wird auf die nicht orientierten Richtungen $\mathbb{P}^2$ eingeschränkt und hat somit die Form $\mathbb{P}^2 \times \mathbb{R}^3$.

Auf den ersten Blick sieht es so aus, als ob der Definitionsbereich zwei Dimensionen mehr hat als zuvor, doch die Einschränkung $q = r + t\vec{\mathbf{n}}$ reduziert ihn auf die dreidimensionale Untermenge $J_r = \{(n,q) \in \mathbb{P}^2 \times \mathbb{R}^3 \mid \exists t \in \mathbb{R} : q = r + t\vec{\mathbf{n}}\}$.

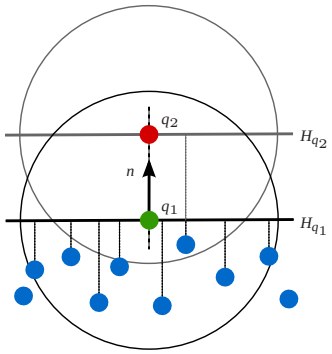
Als nächstes wird ein Vektorfeld definiert, welches die Richtung der Normalen angibt:

$$n_{MLS}(q) = \operatorname*{argmin}_n e_{MLS}(n,q). \tag{5.5}$$

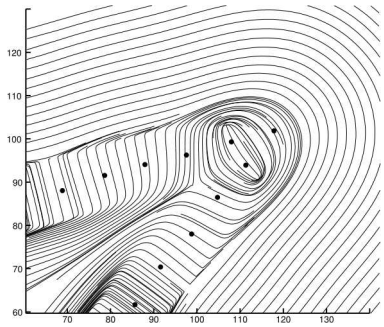
Dadurch ist in jedem Punkt $q \in \mathbb{R}^3$ eine Normale der Hyperebene durch q gegeben, wobei diese Hyperebenen die Fläche S lokal approximieren. Diese Normalen können ähnlich wie in (5.3) mit Hilfe der Kovarianzmatrix ermittelt werden.

Die Energiefunktion

Hier soll die Energiefunktion näher betrachtet werden. In einem Punkt $\tilde{r}$, der auf der MLS-Fläche liegt, hat die Energiefunktion ein lokales Minimum. Entfernt sich



(a) Bei fixem n ist der Energiefehler $e_{MLS}(n, q_1)$ größer als $e_{MLS}(n, q_2)$, obwohl H_{q_1} die bessere Approximation der Punktmenge ist.



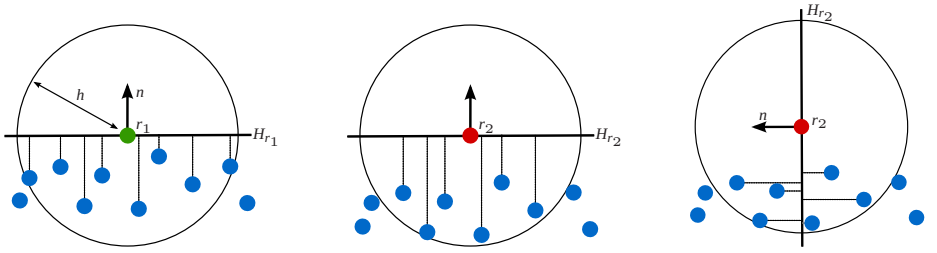
(b) Ein Beispiel für die Visualisierung des Vektorfeldes der Normalen durch ein Strömungsfeld aus [AK04a]. Die schwarzen Punkte repräsentieren die Fläche.

Abbildung 5.2

ein Punkt nun von $\tilde{S}$, nimmt der Fehler zuerst zu. Doch dies geschieht nur in einer engen Umgebung um die Fläche. Außerhalb dieser Umgebung nimmt e_{MLS} mit zunehmender Distanz von der Fläche wieder ab. Damit würden Punkte, die eine sehr große Distanz zur Fläche haben, ebenfalls als Teil der Fläche angesehen. Dieses Phänomen wird durch die Gewichtsfunktion verursacht. Hat ein Punkt eine Entfernung zu der Fläche, die den Parameter h der Gewichtsfunktion übersteigt, so werden die Punkte der Punktmenge P so gering gewichtet, dass das Energiefunktional einen sehr kleinen Wert annimmt (siehe Abbildung 5.2a). Dies lässt sich leicht durch eine Normierung der Gewichte beheben, wie es im Abschnitt 5.3.1 beschrieben wird. Mit dieser normierten Gewichtsfunktion kann die Einschränkung auf ein lokales Minimum entfallen.

Das Vektorfeld

Auch für das Vektorfeld der Normalen ist ein eingeschränkter Definitionsbereich notwendig, wie im Folgenden erläutert wird. Ist der Vektor $n_{MLS}(x)$ in einem Punkt $x \in \mathbb{R}^3$ orthogonal zur der MLS-Fläche, so wird die Hyperebene aus dem ersten Teil des MLS-Verfahrens korrekt ermittelt. Dies ist im Allgemeinen jedoch nicht der Fall. Liegt der Punkt x zu weit von der Punktwolke entfernt, so ist der Vektor $n_{MLS}(x)$ nicht mehr orthogonal zur Fläche, sondern parallel. Amenta und Kil haben in [AK04a] ein Vektorfeld durch Strömungslinien visualisiert, um dieses Verhalten deutlich zu machen (Beispiel in Bild 5.2b).



(a) Hyperebene und Normale für r_1

(b) Gewünschte Hyperebene für r_2

(c) Tatsächliche Hyperebene für r_2

Abbildung 5.3: Verhalten der Hyperebene in Abhängigkeit von der Entfernung zur Punktmenge

Auch dieses Scheitern des Verfahrens ist auf den Parameter h der Gewichtsfunktion zurückzuführen. Im Beispiel in Abbildung 5.3 ist folgendes zu erkennen: Die gegebene Punktmenge repräsentiert eine horizontal verlaufende Fläche. Für diese Menge wird eine Ebene gesucht, durch die sie in Abhängigkeit vom Referenzpunkt r_i lokal approximiert wird und die durch den Referenzpunkt verläuft. Im ersten Bild 5.3a entspricht das Ergebnis den Erwartungen an eine Approximation der Punktmenge. Die in 5.3b dargestellte Hyperebene durch r_2 würde einer globalen Approximation der Punktmenge entsprechen. Lokal gesehen ergibt die zu dieser Hyperebene gehörige Normale einen größeren Fehler $e_{MLS}(n, r_2)$ als die in Bild 5.3c dargestellte Normale. Es wird also die im dritten Bild dargestellte Ebene verwendet, die jedoch keine korrekte Approximation der gesuchten Fläche darstellt, sondern ungefähr orthogonal zu dieser verläuft. Der Grund für dieses Verhalten ist der größere Abstand des Punktes r_2 von der Punktmenge im Vergleich zu r_1.

Zur Lösung könnte der Parameter h der Gewichtsfunktion vergrößert werden, wodurch die Teilmenge der Punktmenge, die die lokale Approximation beeinflusst, vergrößert wird. Damit vergrößert sich zwar der Definitionsbereich des Verfahrens, aber die Approximation $\tilde{S}$ der Fläche S wird ungenauer. Mit zunehmender Größe von h geht die Lokalität der Approximation verloren und die Gewichtsfunktion hat keinen Nutzen mehr.

5.3 Gewichtsfunktion

Die Gewichtsfunktion, wie sie das WLS- sowie das MLS-Verfahren verwenden, ermöglicht die lokale Approximation. Sie sorgt dafür, das die Punkte in der näheren Umgebung vom Referenzpunkt r die Approximation stark beeinflussen und Punkte, die außerhalb dieser Umgebung liegen bei der Bestimmung des Approximationspolynoms keine Bedeutung haben. Die Distanz d der Punkte p_i vom Referenzpunkt wird durch den euklidischen Abstand $d = \|r - p_i\|$ bestimmt und damit gilt stets $d \geq 0$.

Es sei $\Theta : \mathbb{R}_{\geq 0} \to \mathbb{R}$ die Gewichtsfunktion und $K_h(r)$ die Umgebung vom Radius h um den Referenzpunkt r. Folgende Bedingungen sollten durch die Gewichtsfunktion erfüllt werden:

(1) $\Theta(d)$ ist monoton fallend für $d \geq 0$

(2) $\Theta(d) = 0$, außerhalb der Umgebung $K_h(r)$

(3) $\Theta(d) \geq 0$, innerhalb von $K_h(r)$

(4) $\Theta(d)$ ist stetig differenzierbar

Die ersten beiden Bedingungen sorgen dafür, dass die Approximation lokal ist. Durch (1) nimmt die Gewichtung der Punkte mit zunehmender Entfernung vom Referenzpunkt ab und (2) garantiert, dass Punkte außerhalb der Umgebung $K_h(r)$ keine Relevanz für die Approximation im Punkt r haben. Die dritte Eigenschaft ist notwendig, damit die Matrix in (4.5), die bei der Minimierung des WLS- bzw. MLS-Fehlers invertiert werden soll, nicht singulär ist. Der vierte Punkt ist Voraussetzung dafür, dass die Funktion, die den MLS-Fehler minimiert, stetig differenzierbar ist. Außerdem ist die Stetigkeit eine Voraussetzung dafür, dass die MLS-Fläche glatt ist.

Der Parameter h hat einen beträchtlichen Einfluss auf die Genauigkeit der MLS-Fehler. Je kleiner h, desto schneller konvergiert $\Theta(d)$ gegen Null und die Approximation ist „lokaler“. Im umgekehrten Fall, also bei großem h, ergibt sich eine „globalere“ Approximation, die scharfe Kanten und kleinere Merkmale glättet (siehe Abbildung 5.4).

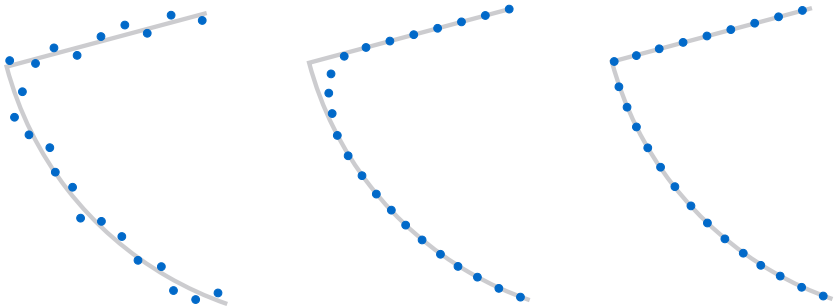


Abbildung 5.4: Das linke Bild zeigt eine fehlerbehaftete Punktmenge, die ein Objekt repräsentiert. Das mittlere Bild zeigt, wie eine scharfe Kante des Objektes durch den MLS-Projektionsoperator geglättet wird. Durch die Segmentierung der Punktmenge vor der Projektion (siehe Kapitel 5.4) kann eine Verbesserung wie im Bild rechts erreicht werden.

5.3.1 Gaußsche Gewichtsfunktion

Eine in der Literatur häufig verwendete Gewichtsfunktion ist die Gaußfunktion:

$$\Theta(d) = e^{-\frac{d^2}{h^2}}.$$

Dabei ist d der euklidische Abstand des gewichteten Punktes vom Referenzpunkt r und h der Radius der kugelförmigen Umgebung $K_h(r)$ um r. Diese Funktion hat allerdings den Nachteil, dass sie die zweite Bedingung nur näherungsweise erfüllt. Sie liefert auf dem gesamten Definitionsbereich positive Werte größer Null. Außerhalb der Umgebung $K_h(r)$ ist $\Theta(d)$ zwar schon sehr klein und nimmt mit zunehmender Distanz d weiter stark ab, aber bei der Bestimmung der MLS-Fläche werden so alle Punkte in die Berechnung mit einbezogen, wenn auch mit einem sehr geringen Gewicht. Das sorgt für einen hohen Rechenaufwand. Wäre die zweite Bedingung erfüllt, so würden bei der lokalen Approximation nur die wenigen lokal einflußreichen Punkte mit einem Gewicht größer Null bewertet.

Normierte Gewichtsfunktion

Ein anderes Problem tritt bei der Suche einer Normale bzw. eines lokalen Koordinatensystems im ersten Schritt des MLS-Approximationsverfahrens (Kapitel 5.2.1) auf. Sei die Normale des lokalen Koordinatensystems fix und orthogonal zur gesuchten Fläche. So wird der Koordinatenursprung in Richtung der Normalen durch minimieren der Energiefunktion gesucht. Das globale Minimum wird erreicht, wenn die Distanz d des gesuchten Koordinatenursprungs q vom Referenzpunkt und auch von der gesamten Punktmenge P, unendlich groß wird, also

$d = \infty$. Um diesen Effekt zu vermeiden kann eine normierte Version der Gewichtsfunktion verwendet werden:

$$\Theta_N(d) = \frac{e^{-\frac{d^2}{h^2}}}{\sum_{i=1}^{|P|} e^{-\frac{d_i^2}{h^2}}} \tag{5.6}$$

wobei $d_i = \|r - p_i\|$ und $p_i \in P$.

5.3.2 Wendlands Gewichtsfunktion

Die folgende von Wendland (siehe [Nea04]) entwickelte Gewichtsfunktion hat einen kompakten Träger, so dass die meisten Punkte der Punktwolke mit Null gewichtet werden. Das hat den Vorteil, dass der numerische Aufwand deutlich reduziert wird. Die Funktion

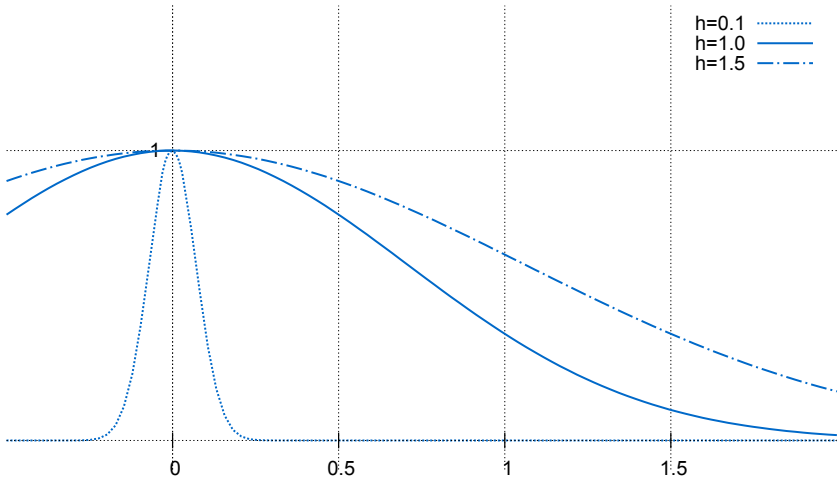
$$\Theta_w(d) = \left(1 - \frac{d}{h}\right)^4 \left(\frac{4d}{h} + 1\right)$$

ist wohldefiniert für $d \in [0, h]$. Es gilt $\Theta_w(0) = 1$, $\Theta_w(h) = 0$, $\Theta_w'(h) = 0$, $\Theta_w''(h) = 0$ und damit ist Θ_w C^2-stetig.

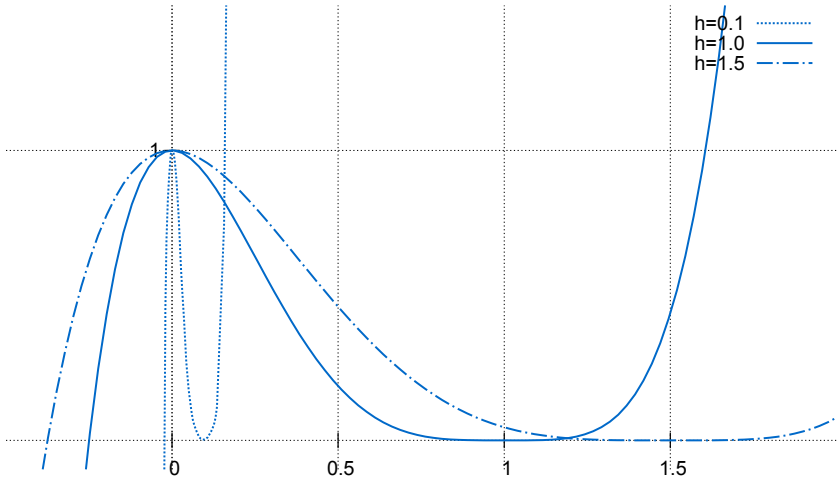
5.3.3 Adaptive Gewichtsfunktion

Im vorherigen Abschnitt wurde bereits erläutert, dass der Parameter der Gewichtsfunktion das Approximationsverhalten beeinflusst. Wird dieser Parameter für die gesamte MLS-Fläche als konstanter Wert festgesetzt, so führt dies zu folgendem Problem: Ist der Parameter zu groß, werden sehr viele Punkte für die lokale Approximation verwendet. Dadurch werden in der Fläche S vorkommende scharfe Kanten abgerundet und kleinere Merkmale, die nur durch wenige Punkte repräsentiert werden, verschwinden ganz. Ist der Parameter allerdings zu klein, so kommt es in Regionen mit einer geringen Punktdichte vor, dass sich das MLS-Approximationsverfahren nicht mehr anwenden lässt, da zu wenige Punkte für die Berechnung der lokalen Approximation zur Verfügung stehen.

Aus diesem Grund ist es naheliegend, eine Gewichtsfunktion zu verwenden, die sich an die Punktdichte in der Umgebung des jeweiligen Referenzpunktes r anpasst. Bei einer solchen adaptiven Gewichtsfunktion, wird diese Anpassung über den Parameter h vorgenommen. Dazu soll h der Distanz entsprechen, die unter



(a) Gaußsche Gewichtsfunktion



(b) Gewichtsfunktion nach Wendland

Abbildung 5.5

den Distanzen zwischen r und allen Punkten p_i den k-ten Rang hat. Es gilt also:

$$k = |\{d_i = \|r - p_i\| \,|\, d_i \leq h\}|.$$

Damit ist gewährleistet, dass jeweils genau k Punkte die Berechnung einer lokalen Approximation bestimmen.

5.4 Robustes MLS

Das MLS-Approximationsverfahren nach Levin [Lev03] hat den Nachteil, dass es auf stetige Flächen ausgelegt ist. Wird es auf Flächen angewandt, die Unstetigkeiten enthalten, so werden diese, je nach Parameter der Gewichtsfunktion, mehr oder weniger stark geglättet (siehe Beispiel in Bild 5.4). Fleishman et al. lösen dieses Problem in [FCOS05], indem sie eine Segmentierung der Daten durchführen. In [FCOS05] beschreiben Fleishman et al. ein Verfahren, welches Unstetigkeiten erhalten kann.

Für dieses Verfahren werden die Punkte in der Umgebung des Referenzpunktes r bestimmt. Der Radius dieser Umgebung wird durch den Parameter der Gewichtsfunktion bestimmt, so dass alle Punkte die für die Optimierung von r relevant sind in dieser Umgebung enthalten sind. Im Folgenden wird diese Menge von Punkten mit U bezeichnet.

Ist das durch U repräsentierte Flächenstück glatt, so wird die MLS-Projektion auf den Referenzpunkt angewendet. Kommt in U eine Unstetigkeit vor, dann erstreckt sich die Umgebung des Punktes r über zwei oder mehr glatte Flächenstücke. Zunächst wird die Punktmenge U mit dem sogenannten Forward-Search-Algorithmus segmentiert und das Segment ausgewählt, auf das der Referenzpunkt projiziert werden soll. Für das MLS-Verfahren wird dann nicht mehr die gesamte Punktmenge verwendet, sondern nur noch dieses Segment.

Im Folgenden wird das Segmentierungsverfahren beschrieben und durch ein paar Modifikationen auf die Situation in dieser Arbeit angepasst.

5.4.1 Forward-Search

Der Forward-Search-Algorithmus soll eine Teilmenge $\bar{Q}$ der Punktmenge U bestimmen, welche ein glattes Flächenstück repräsentiert. Dafür wird eine initiale Menge Q von Punkten benötigt, die in der zu suchenden Teilmenge liegt. Diese Menge Q wird, wie im Abschnitt 5.4.2 beschrieben, durch das Least-Median-of-Squares-Verfahren bestimmt.

Der Forward-Search-Algorithmus iteriert über alle Punkte in der Umgebung U des Referenzpunktes r. Im ersten Schritt liefert das LS-Verfahren zu der initialen Punktmenge Q ein bivariates Approximationspolynom und einen Approximationsfehler für jeden Punkt aus U. Der Punkt mit dem kleinsten Fehler bildet

zusammen mit der initialen Menge Q die Punktmenge $\bar{Q}$. Pro Iterationsschritt i wird wieder ein Approximationspolynom zu den Punkten aus $\bar{Q}$ berechnet und die i Punkte mit den kleinsten LS-Fehlern bilden mit der initialen Punktmenge Q das neue $\bar{Q}$. Die Iteration läuft so lange, bis $\bar{Q}$ alle Punkte enthält, die eine glatte Fläche des Objektes repräsentieren. Als Abbruchbedingung wird eine Schranke s festgelegt, die für die Punktmenge $\bar{Q}$ nur Punkte erlaubt, deren LS-Fehler unterhalb dieser Schranke liegen.

Zur Bestimmung einer solchen Schranke markieren Fleishman et al. [FCOS05] eine Teilmenge der Punktmenge U, die ein glattes Flächenstück repräsentiert. Durch das LS-Verfahren wird hierzu ein Approximationspolynom bestimmt. Der größte Wert unter den LS-Fehlern der markierten Punkte zu dem Polynom wird als Schranke verwendet. Diese Schranke lässt Punkte zu, die nicht unmittelbar auf der Fläche liegen und erlaubt somit auch die Segmentierung von verrauschten Punktmengen.

Bei den in dieser Arbeit betrachteten Daten entspricht die Schranke s der Hälfte des maximalen Rauschens, welches in der Punktmenge vorkommt. Ist das Rauschen der Tiefenkarten bekannt, so entfällt das von Fleishman et al. händisch durchgeführte Auswahlverfahren eines glatten Flächenstücks.

Ist eine Teilmenge $\bar{Q} \subset U$ bestimmt, so wird die übrige Punktmenge $U \setminus \bar{Q}$ auf weitere Teilmengen untersucht, die ein anderes glattes Flächenstück repräsentieren. Einzelne Punkte, die nach dieser Segmentierung übrig bleiben, sind Ausreißer und werden bei der nachfolgenden Projektion des Referenzpunktes ignoriert.

Sind alle Segmente gefunden, so wird im Rahmen dieser Arbeit ein letzter Schritt zur Vervollständigung der Segmente durchgeführt. An den Stellen, wo mehrere Flächenstücke eine Kante oder Ecke bilden, kann es Punkte geben, die nicht eindeutig einem Flächenstück zugeordnet werden können. Diese Punkte sollten auch in jedem der entsprechenden Segmente enthalten sein. Daher werden die Approximationspolynome der Segmente verwendet, um Punkte aus anderen Segmenten zu finden, die eine Distanz zum Polynom haben, welche unterhalb der Schranke s liegt.

5.4.2 Initiale Punktmenge

Für den Forward-Search-Algorithmus wird eine initiale Punktmenge benötigt, welche durch ein modifiziertes Least-Median-of-Squares-Verfahren (LMS) be-

stimmt werden kann. Dabei wird zu jedem Punkt x_j aus der betrachteten Teilmenge U ein Approximationspolynom bestimmt. Danach wird aus dieser Menge von Polynomen das Polynom ausgewählt, welches eine Teilmenge von U besonders gut approximiert. Die Menge, die dieses Polynom definiert, ist die initiale Punktmenge für den Forward-Search-Algorithmus. Im Folgenden wird dieser Vorgang im Detail beschrieben.

Bestimmung der Approximationspolynome

Bei dem LMS-Verfahren wird eine feste Anzahl von Punkten aus U zufällig ausgewählt. Um das durch diese Punkte repräsentierte Flächenstück zu approximieren, kommt das gewöhnliche LS-Verfahren zum Einsatz. Dies wird für jeden Punkt in U durchgeführt. Eine Modifikation dieses Verfahrens nutzt eine besondere Eigenschaft der vorgegebenen Punktmenge: Es handelt sich bei der Menge U, wie auch bei der Gesamtpunktmenge P, um eine Repräsentation einer Fläche. Also kann davon ausgegangen werden, dass ein Punkt $x_j \in U$ mitsamt seiner unmittelbaren Umgebung $K_\epsilon(x_j)$ im Allgemeinen ein glattes Flächenstück repräsentiert. Zur Bestimmung eines Polynoms zum Punkt x_j werden jetzt statt den zufällig gewählten Punkten die Punkte aus $K_\epsilon(x_j)$ verwendet. Ein quadratisches Polynom ist bei dieser Approximation vollkommen ausreichend, da es sich um ein glattes Flächenstück handelt.

Auswahl der initialen Punktmenge

Aus der Menge der zuvor bestimmten Polynome soll nun eines ausgewählt werden. Es wird zunächst für jedes Polynom p_j der Median m_j aller Residuen r_i bestimmt:

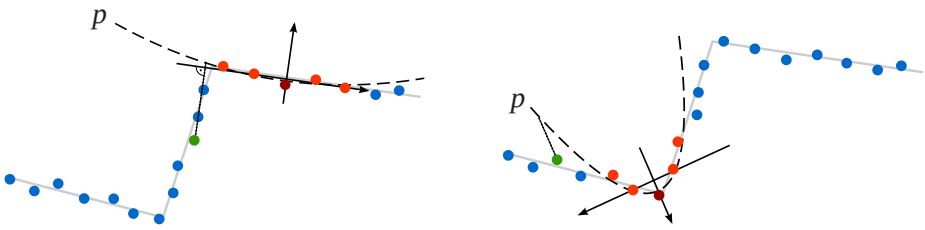
$$m_j = \operatorname*{median}_i r_i.$$

Dabei gilt:

$$r_i = |p_j(x_{i1}, x_{i2}) - x_{i3}| \quad \text{und} \quad x_i = (x_{i1}, x_{i2}, x_{i3}) \in U.$$

Die Mediane geben Aufschluss darüber, wie gut die jeweiligen Polynome einen glatten Bereich der gesuchten Oberfläche approximieren. Das „beste“ Polynom, also das, welches den kleinsten Median liefert, definiert die initiale Menge Q für den Forward-Search-Algorithmus. Dabei enthält Q genau die Punkte, die durch

dieses Polynom approximiert wurden, also die Punkte aus der Umgebung $K_\epsilon(x_j)$.



(a) Das Polynom p ist eine gute Approximation eines Flächestücks, aber der Median (Distanz des grünen Punktes zum Polynom) ist sehr groß.

(b) Hier werden zwei Flächenstücke von einem Polynom p approximiert. Das ist nicht gewünscht, wird aber als besseres Polynom angesehen, weil der Median deutlich kleiner ist als im linken Bild.

Abbildung 5.6: Beispiel der Verwendung des gewöhnlichen Medians bei der Suche nach einem initialen Polynom.

Das LMS-Verfahren ist ein robustes Schätzverfahren, da es einen Bruchpunkt von 50 % hat. Das heißt, dass das Verfahren erst ab einem Ausreißeranteil von 50 % versagt (siehe [Rou84]). Bei Punktmengen, die bis zu zwei Flächenstücke repräsentieren, was meistens der Fall ist, ist diese Eigenschaft sehr nützlich. Handelt es sich jedoch um drei oder mehr Flächenstücke, so kann die Verwendung des Medians zu einer ungünstigen Polynomwahl führen, nämlich zu Polynomen, die durch Punkte aus verschiedenen Flächenstücken bestimmt werden und nicht genau eine glatte Fläche approximieren (siehe Abbildung 5.6). Daher wird das LMS-Verfahren so modifiziert, dass der Bruchpunkt noch höher liegt. Anstelle des Medians kommt jetzt das Residuum mit dem k-ten Rang zum Einsatz. Der Median ist bei $n \in \mathbb{N}$ berechneten Distanzen genau der Wert mit dem $\frac{n}{2}$-ten Rang in der aufsteigend sortierten Menge der Distanzen. Wird davon ausgegangen, dass die Punktdichte überall ungefähr gleich groß ist und t die Anzahl der Flächenstücke ist, die in $K_\epsilon(x_j)$ liegen, so kann k wie folgt gesetzt werden:

$$k = \frac{1}{t}|K_\epsilon(x_j)|.$$

Dabei entspricht der zweite Faktor der Anzahl der Punkte in $K_\epsilon(x_j)$. In dem Beispiel in Abbildung 5.6 ist $k = 7$ eine gute Wahl, denn damit ist das Polynom im linken Bild das initiale Polynom.

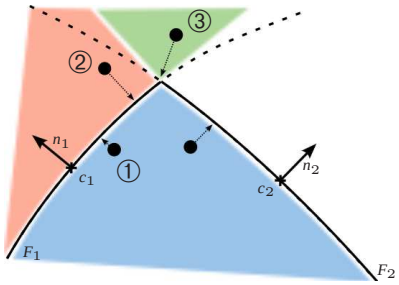


Abbildung 5.7: Drei Fälle der Projektion von Punkten auf eine Fläche aus zwei Flächenstücken.

5.4.3 Spezifizierung der Projektion

Nach der Segmentierung von U muss nun entschieden werden, auf welches Flächenstück der Referenzpunkt r projiziert werden soll. Dazu beschreiben Fleishman et al. in [FCOS05] einen Algorithmus, der die Segmente paarweise betrachtet. Ausgehend von zwei Segmenten, bilden die dadurch repräsentierten Flächenstücke einen konkaven Bereich. Jedes Flächenstück ist Teil einer Fläche, die den $\mathbb{R}^3$ in zwei Teilräume unterteilt. Der Teilraum, der den konkaven Bereich enthält, soll hier als Innenseite der Fläche bezeichnet werden und der andere dementsprechend als Außenseite. Der Referenzpunkt wird auf beide Flächen projiziert und der euklidische Abstand zwischen Projektion und Punkt wird ermittelt. Nun gibt es drei Fälle (siehe ① bis ③ in Abbildung 5.7) für die Lage des Punktes, die unterschieden werden müssen.

Der erste Fall ist der, dass der Referenzpunkt auf der Innenseite beider Flächen, also im konkaven Bereich liegt. Es wird die Projektion gewählt, die den kleineren Abstand zum Referenzpunkt hat.

Im zweiten Fall liegt der Punkt innerhalb der Fläche F_2 und außerhalb von F_1. Hier wird die Projektion auf die Fläche F_1 verwendet.

Etwas umständlicher ist der letzte Fall. Der Punkt liegt hierbei auf der Außenseite beider Flächen und muss daher auf die Schnittkurve der beiden Flächen projiziert werden. Dazu wird der Punkt zuerst auf die näher liegende Fläche projiziert. Dann wird der abgebildete Punkt auf die zweite Fläche projiziert. Durch iteratives Projizieren zwischen den beiden Flächen konvergiert der Punkt gegen einen Punkt auf der Schnittkurve der beiden Flächen.

Sind drei oder mehr Flächen involviert, so sind deutlich mehr Fallunterscheidungen zu treffen um das passende Segment zu finden.

Bestimmung der Lage des Referenzpunktes

Um die Lage eines Punktes zu den Flächen zu definieren, werden orientierte Normalen benötigt. Dazu werden zuerst die Schwerpunkte c_1, c_2 der Punktmengen gebildet, die jeweils ein Flächenstück repräsentieren. Mittels Kovarianzen kann nun eine Normale n_i in jedem dieser Schwerpunkte bestimmt werden (siehe 4.4). Da diese Normalen noch nicht orientiert sind, werden die Geraden betrachtet, die durch die Normalen und die Schwerpunkte bestimmt werden. Die Richtung, in der sich der Abstand der beiden Geraden verringert, wird als konkaver Bereich der beiden Flächen definiert.

Der Vektor v_i, der den Referenzpunkt r und seine Projektion auf die Fläche F_i verbindet, wird nun im Bezug zu den Normalen n_i dieser Fläche betrachtet. Liegt der Punkt auf der Außenseite, so wird der Winkel zwischen diesen beiden Vektoren sehr klein sein, liegt er auf der Innenseite wird er sehr groß sein.

$$\begin{aligned} \langle v_i, n_i \rangle > 0 \quad &\Leftrightarrow \quad r \text{ liegt auf der Außenseite von } F_i \\ \langle v_i, n_i \rangle < 0 \quad &\Leftrightarrow \quad r \text{ liegt auf der Innenseite von } F_i \end{aligned}$$

5.4.4 Spezifizierung des Segmentes durch die Kameraausrichtung

Im Rahmen der Aufgabenstellung dieser Arbeit werden Tiefenwerte in Richtung eines Sehstrahls optimiert. Der optimale Tiefenwert liegt theoretisch genau im Schnittpunkt des Sehstrahls mit dem aufgezeichneten Objekt. Diese Eigenschaft wird hier ausgenutzt, um eine einfachere und genauere Spezifizierung des gesuchten Segmentes zu erreichen.

Zuerst werden die Schnittpunkte s_i des Sehstrahls mit den zu den Segmenten gehörenden Approximationspolynomen bestimmt. Gibt es zwei Schnittpunkte, so wird der verwendet, der dem Referenzpunkt näher ist, weil der durch die Tiefenkarten gewonnene Referenzpunkt bereits in der Nähe des Objektes liegt, sofern er nicht durch einen Flying Pixel oder einen Hintergrundpixel entstanden ist.

Dann wird die Anzahl k_i der Punkte in einer vordefinierten Umgebung der Schnittpunkte s_i untersucht. Das Segment mit dem größten k_i wird für die MLS-Projektion ausgewählt. Gibt es mehrere k_i die bis auf eine Toleranz *tol* gleich groß sind, so wird das Segment verwendet, welches durch den Sehstrahl aus Sicht der

Kamera als erstes getroffen wird. Sind alle k_i sehr klein, so liegen in der Umgebung von x_i nur sehr wenige Punkte. Es kann davon ausgegangen werden, dass der Sehstrahl in diesem Fall innerhalb der Umgebung U nicht auf das aufgezeichnete Objekt trifft.

5.5 MLS zur Optimierung von PMD-Daten

Die PMD-basierten Tiefendaten liegen in Form von Tiefenkarten vor, die pro Pixel einen Tiefenwert enthalten, der in einer durch die Kamera vorgegebenen Richtung ermittelt wurde. Die aus den Tiefeninformationen gewonnenen Punkte $p_i \in \mathbb{R}^3$ werden durch den Projektionsoperator des MLS-Approximationsverfahrens orthogonal zur gesuchten Fläche $\hat{S}$ auf diese projiziert, wie Abbildung 5.8 zeigt.

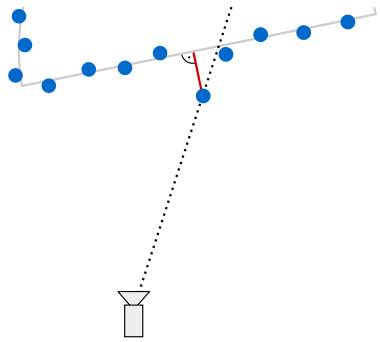


Abbildung 5.8: Die Projektion eines Punktes erfolgt orthogonal zur Oberfläche S wie hier durch die rote Linie gekennzeichnet ist.

Folglich müsste die durch das MLS-Verfahren erzeugte Fläche auf die Bildebene der PMD-Kamera projiziert werden, um ein optimiertes Tiefenbild zu erhalten. Durch diese Rückprojektion entstehen nun Ungenauigkeiten in den Tiefenkarten. Daher wird in dieser Arbeit ein strahlbasierter Ansatz gewählt. Dabei wird davon ausgegangen, dass die Punktwolke, die aus den PMD-Daten gewonnen wurde, die Oberfläche des Objektes repräsentiert. Statt, wie im vorherigen Kapitel, die gesamte Punktwolke zu optimieren, wird nun der Tiefenwert des zu optimierenden Pixels in Richtung des Sehstrahls optimiert. Im Folgenden werden zwei Ansätze vorgestellt, um diese Optimierung in Richtung des Sehstrahls zu erreichen.

5.5.1 Minimierung des Fehlers der MLS-Projektion

Das in Kapitel 5 beschriebene MLS-Approximationsverfahren bildet einen Punkt auf die gesuchte Fläche $\hat{S}$ durch eine orthogonale Projektion statt in Richtung des Sehstrahls ab. Um das MLS-Verfahren trotzdem zu nutzen, wird hier die Projektion des Sehstrahls auf die Oberfläche betrachtet. Der optimale Tiefenwert wird dann durch den Punkt repräsentiert, der auf dem Sehstrahl liegt und auf sich selbst

projiziert wird. Geometrisch gesehen, ist das der Schnittpunkt von Sehstrahl und der Objektoberfläche.

Durch die Anwendung eines eindimensionalen Minimierungsverfahrens kann der Schnittpunkt ermittelt werden. Dazu wird der Abstand der Punkte r_i auf der Geraden R von deren Projektionen $\tilde{r}_i \in \tilde{S}$ bestimmt. Das Minimum dieser Abstände ist Null, falls die Gerade R die Fläche $\tilde{S}$ in einem oder mehreren Punkten schneidet. Ist dies nicht der Fall, so gibt es keinen Schnittpunkt und das Objekt wird nicht von diesem Sehstrahl R erfasst. Beginnend mit einem Startpunkt $r_0 \in R$ wird die Distanz d_i in Abhängigkeit der Punkte r_i entlang des Geraden minimiert. Falls mehrere Minima auftreten, so schneidet R das Objekt an mehreren Stellen, was bei einem dreidimensionalen Objekt häufig der Fall ist. Dabei sollte das Minimum gewählt werden, welches der Kamera am nächsten liegt, da dieser Punkt von der Kamera auch zuerst erfasst wird.

Ein Nachteil dieses Ansatzes ist die Anzahl der Punkte $\tilde{r} \in \tilde{S}$, die bestimmt werden müssen. Für jeden dieser Punkte wird das MLS-Verfahren durchgeführt, welches jeweils die Lösung eines nichtlinearen Optimierungsproblems erfordert.

5.5.2 Schnittpunkt einer lokalen Approximation mit Sehstrahl

Die elegantere Lösung, wie sie auch in [AA03b, AA03a, AA04a] Verwendung findet, konvergiert deutlich schneller gegen den gesuchten Schnittpunkt. Statt wie im vorherigen Ansatz die Projektionen verschiedenster Punkte r_i zu bestimmen bis eine Distanz von annähernd Null erreicht wird, sollen nun bestimmte Punkte r_i ausgewählt werden. Ausgehend von einem Startpunkt r_0 wird auch hier ein lokales Koordinatensystem und schließlich eine lokale Approximation p der Fläche S bestimmt. Hierbei genügt es vorerst, eine lineare Approximation zu verwenden. Der Schnittpunkt dieses Polynoms p, im linearen Fall eine Ebene, mit der Geraden R liefert einen neuen Punkt r_{i+1}. Von dem neu gewonnen Schnittpunkt aus, wiederholt man diesen Vorgang bis dieses iterative Verfahren gegen den gesuchten Schnittpunkt konvergiert.

Das lokale Koordinatensystem lässt sich wie folgt schnell und einfach bestimmen. Es wird der in Abhängigkeit von r_i gewichtete Mittelwert aller Punkte bestimmt:

$$a(r_i) = \frac{\sum_{p_j \in P} \Theta(\|p_j - r_i\|) p_j}{\sum_{p_j \in P} \Theta(\|p_j - r_i\|)}.$$

In diesem Punkt $a(r_i)$ kann nun wieder mittels Kovarianzmatrix eine Normale und somit auch ein lokales Koordinatensystem bestimmt werden. Hierbei ist zu beachten, dass diese Normale nicht orthogonal zur MLS-Fläche ist. Da sie nur verwendet wird, um ein neues r_i zu bestimmen, ist diese Ungenauigkeit jedoch irrelevant. Je näher der Punkt r_i dem gesuchten Schnittpunkt kommt, desto besser wird die Approximation der Normalen. Im Falle der Konvergenz im Punkt $\tilde{r} \in \tilde{S}$ wird dann auch eine Normale erreicht, die orthogonal zur Fläche ist.

Um einen Startpunkt r_0 zu finden, definieren [AA03a] dazu die Umgebung $K(S)$ von S durch die Vereinigung der Umgebungen aller Punkte aus P:

$$K(S) = \bigcup_{p \in P} K(p).$$

Der Schnittpunkt des Sehstrahls mit dem Rand der Umgebung $K(S)$ wird als Startpunkt r_0 verwendet. Im Kontext dieser Arbeit ist dies nicht nötig, denn die Tiefenkarten der PMD-Kameras liefern, in Form der gemessenen Tiefenwerte, bereits einen initialen Punkt für jeden Sehstrahl. Im Allgemeinen kann davon ausgegangen werden, dass dieser Punkt in der näheren Umgebung von S liegt. Im Falle von Flying Pixel ist dies zwar nicht der Fall, aber für diese können, wie in Kapitel 4.5 beschrieben, initiale Punkte bestimmt werden.

Kapitel 6

Umsetzung und Implementierung

Die für diese Arbeit verwendeten Tiefenkarten wurden mit dem am Lehrstuhl für Computergrafik in Siegen entwickelten PMD-Simulator erzeugt. Die dazu simulierten PMD-Kameras sind bereits kalibriert. Wie in Kapitel 2 beschrieben sind 42 dieser virtuellen Kameras auf den Ecken einer geodätischen Sphäre positioniert, die einen Radius von vier Metern hat. Als Ausgabeformat für diese Tiefenbilder steht unter anderem das Portable Floating-point Streams (pfs) Format zur Verfügung. In diesem Format lassen sich neben den Tiefeninformationen auch Kamerainformationen speichern. Dazu gehören die Position und Ausrichtung der Kamera in Form einer Transformationsmatrix, die Größe der Pixel und des Bildes, sowie die Entfernung der Bildebene vom Projektionszentrum.

Die Implementierung ist in C++ erfolgt, wobei die OpenSceneGraph und Numerical Recipes Bibliotheken zum Einsatz kamen.

Abbildung 6.1 zeigt in einem Ablaufdiagramm welche Schritte zur Optimierung einer Tiefenkarte durchgeführt werden.

6.1 Vorverarbeitung der Tiefenkarten

Die Informationen aus den pfs-Dateien werden in der Klasse `DepthImage` gekapselt.

Zuerst muss auf die korrekten Koordinaten der Tiefenkarte geachtet werden. Der Ursprung des zweidimensionalen Koordinatensystems der Tiefenkarte liegt links unten. Zur einfacheren Handhabung wird der Koordinatenursprung in die Bildmitte verschoben. Um nun Punkte im $\mathbb{R}^3$ zu erhalten, wird der Differenz-

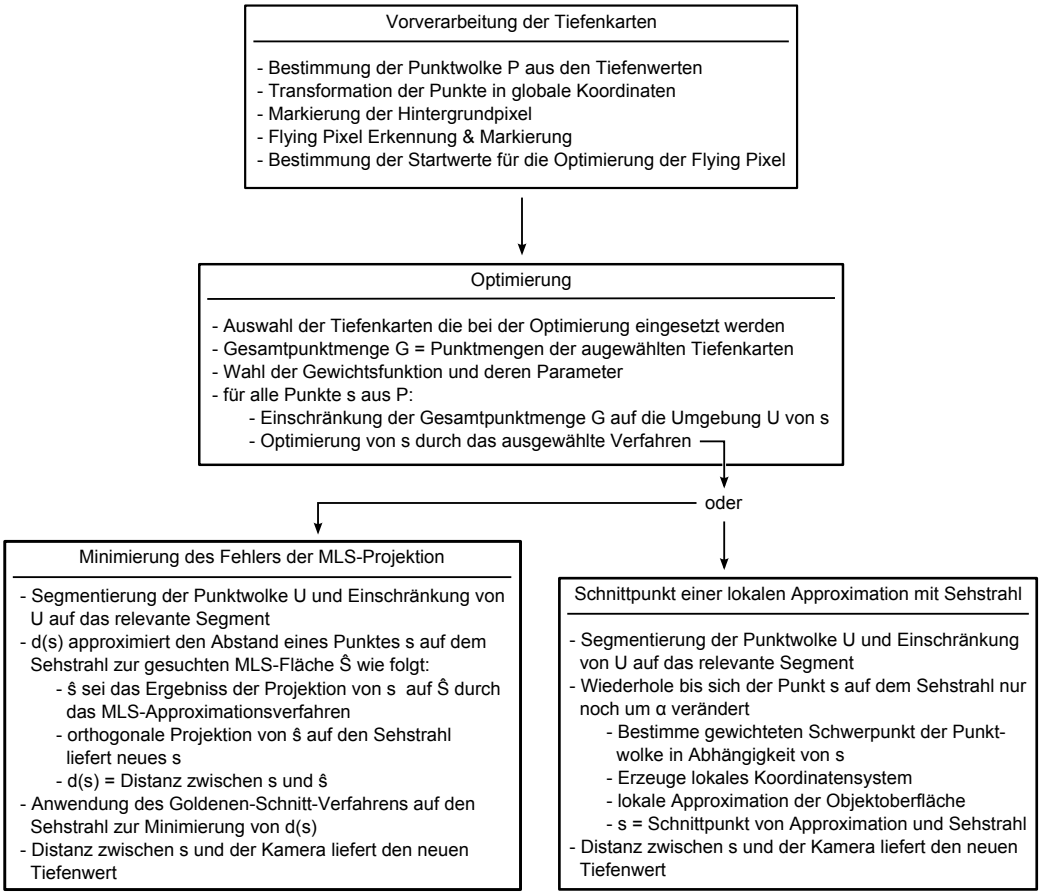


Abbildung 6.1: Das Ablaufdiagramm veranschaulicht die Schritte, die zur Optimierung einer Tiefenkarte durchlaufen werden.

vektor eines Pixels des Tiefenbildes und des Projektionszentrums ermittelt, der dann dem Richtungsvektor eines Sehstrahls entspricht, wobei davon ausgegangen wird, dass der Sehstrahl durch den Mittelpunkt des Pixels verläuft. Wird der in dem Pixel gespeicherte Entfernungswert zur Skalierung des normierten Richtungsvektors verwendet, so ergibt sich ein Punkt im lokalen Koordiatensystem. Alle so aus einer Tiefenkarte gewonnenen Punkte repräsentieren somit die Oberfläche des aufgezeichneten Objektes. Zur besseren Handhabung werden die zu den Tiefenbildern gehörigen Punktmengen aus ihren lokalen in ein globales Koordinatensystem transformiert, dessen Ursprung in der Mitte der geodätischen Sphäre liegt.

Alle Pixel die kein Objekt erfassen, also einen Tiefenwert von ca. sieben Metern aufweisen, werden als „Hintergrundpixel“ markiert, da sie für die Optimierung keine Relevanz haben. Des Weiteren werden alle Flying Pixel bestimmt, als

solche markiert und grobe potentielle Verbesserungen der Flying Pixel ermittelt.

Listing 6.1: Flying Pixel Erkennung

```
for (y = 0; y < imageHeight; ++y) {
  for (x = 0; x < imageWidth; ++x) {
    /* Hintergrundpixel werden nicht betrachtet. Dies gilt
       auch für die Pixel am Rand der Tiefenkarten die, bei
       einem Objekt in der Mitte der Kamerasphäre, ebenfalls
       keine verwertbaren Tiefeninformationen enthalten. */
    if (!isBackground(x,y) && !isBorder(x,y)) {
      /* Der Unterschied zwischen den Tiefenwerten von (x,y)
         und dessen Nachbarpixeln wird ermittelt. */
      /* (a) Zeilennachbarn werden untersucht */
      preX = abs(depth(x,y) - depth(x-1,y));
      pre2X = abs(depth(x-1,y) - depth(x-2,y));
      postX = abs(depth(x,y) - depth(x+1,y));
      post2X = abs(depth(x+1,y) - depth(x+2,y));
      if (preX>fpThreshold || postX>fpThreshold) {
        if (!((abs(preX-pre2X)>nonFpThresh || abs(postX-post2X
            )>nonFpThresh)
              && abs(preX - postX)>nonFpThresh )) {
          isFlyingPixel(x,y) = true;
          continue;
        }
      }
      /* (b) Spaltennachbarn werden untersucht */
      ...
    }
  }
}
```

Der Algorithmus in Listing 6.1 klassifiziert die Flying Pixel einer Tiefenkarte. Dazu wird geprüft, ob die Differenz zwischen dem Tiefenwert eines Pixels (x, y) und dem Tiefenwert eines seiner Zeilennachbarn den Schwellenwert `fpThreshold` überschreitet. Ist dies der Fall, so könnte es sich bei (x, y) um einen Flying Pixel handeln. Auf diese Weise würden aber auch einige Pixel markiert, die gar keine Flying Pixel sind. Dies tritt insbesondere dort auf, wo der Strahl, der die Blickrichtung der Kamera repräsentiert, in einem spitzen Winkel auf die Oberfläche des aufgezeichneten Objektes trifft. In diesem Fall ist die Tiefendifferenz zu den benachbarten Pixeln deutlich größer als bei einer Fläche, die orthogonal zur Blick-

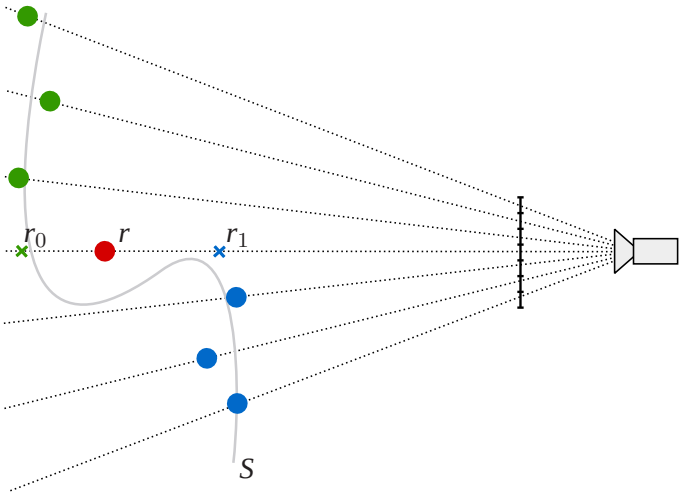


Abbildung 6.2: Die runden Punkte repräsentieren die graue Fläche S. Für den Flying Pixel r werden zwei potentielle Startpunkte für die Optimierung bestimmt (r_0 und r_1). Der Tiefenwert von r_i wird durch den Mittelwert der Tiefenwerte der blauen bzw. grünen Nachbarn gebildet.

richtung liegt (siehe Abbildung 4.1a). Um die Anzahl der fehlerhaft markierten Pixel gering zu halten, werden die Differenzen zwischen dem Referenzpixel und dessen Zeilennachbarn untersucht. Sind diese Differenzen bis auf eine Toleranz von `nonFPThresh` gleich, so liegt der untersuchte Pixel, zumindest im Bezug auf diese Zeile, auf einer Fläche und ist daher kein Flying Pixel. Anschließend werden die Steigungen zwischen dem Referenzpixel und den Spaltennachbarn auf die gleiche Weise untersucht.

Sind die Flying Pixel ausgemacht, so sollen potentielle Tiefenwerte für diese Pixel bestimmt werden, auf die dann das Optimierungsverfahren angewandt werden kann. Dazu werden die benachbarten Pixel in horizontaler und vertikaler Richtung betrachtet. Die Richtung, welche unter den benachbarten Pixeln weniger Flying Pixel aufweist, kann den Tiefensprung besser repräsentieren und wird daher verwendet, um die gesuchten Tiefenwerte zu bestimmen. Für die Nachbarschaft vor und nach dem Flying Pixel wird jeweils eine potentielle Entfernung für den Flying Pixel bestimmt. Der mittlere Tiefenwert der Nachbarpixel vor dem Flying Pixel und der nach diesem, liefern zwei potentielle Entfernungen für den Flying Pixel (siehe Abbildung 6.2). Ist einer der unmittelbaren Nachbarn ein „Hintergrundpixel“, also ein Pixel der entsprechend seines Tiefenwertes als Hintergrund markiert wurde, so wird die potentielle Entfernung des Referenzpixels für diese Nachbarschaft auf die Entfernung des Hintergrunds gesetzt.

6.2 Optimierung der Tiefenkarten

In der Klasse `DepthImageOptimizer` werden Parameter gesetzt, die Tiefenkarten in Form von Objekten der Klasse `DepthImage` geladen und die Optimierung gestartet.

Zuerst einmal wird festgelegt, von welcher Kamera C_r die Tiefenkarte optimiert werden soll. Zur Optimierung werden nicht alle Tiefenkarten verwendet, sondern nur die direkten, also die über eine Kante der geodätischen Sphäre erreichbaren, Nachbarn von C_r. Das sind aufgrund der Topologie von C_r fünf bis sechs Tiefenkarten. Dadurch kann die Datenmenge, die zur Optimierung der Tiefenkarte dieser Referenzkamera C_r verwendet wird, im Vorhinein stark reduziert werden. Eventuell könnten die Daten weiterer Kameras die Optimierung noch verbessern. Allerdings steigt mit wachsender Datenmenge auch die Rechenzeit.

Auch die Wahl des fixen oder adaptiven Parameters der Gewichtsfunktion wird festgelegt. Bei der Wahl einer adaptiven Gewichtsfunktion wird der Parameter so bestimmt, dass eine festgelegte Anzahl x von Punkten in der relevanten Umgebung des Referenzpunktes liegen. D.h. diese x Punkte werden durch die Gewichtsfunktion stark gewichtet, während der Rest der Punkte die Optimierung kaum bis gar nicht beeinflusst.

Zur Durchführung der Optimierung wird die Tiefenkarte pixelweise durchlaufen. Für jeden Pixel, der kein „Hintergrundpixel“ ist, wird eine Optimierung durchgeführt. Dabei dient der Messwert aus der Tiefenkarte als Startpunkt für die Optimierung. Für Flying Pixel werden die beiden potentiellen Tiefenwerte, sofern sie nicht dem Hintergrundwert entsprechen, als Ausgangspunkte für die Optimierung verwendet. Es wird mit dem kleineren Tiefenwert begonnen und falls das Verfahren mit diesem Startwert nicht konvergiert, der zweite potentielle Tiefenwert zum Einsatz gebracht. Konvergiert das Verfahren mit keinem der beiden Werte, so muss es sich um eine Lücke im aufgezeichneten Objekt handeln und es kann der Hintergrundwert angenommen werden.

Die Optimierung kann wahlweise durch einen der beiden Ansätze aus Kapitel 5.5 durchgeführt werden. Für den ersten Ansatz, die Minimierung der Distanz des Punktes auf dem Sehstrahl zur MLS-Fläche, ist die Klasse `PixelDepthOptimizer` , die im folgenden Kapitel 6.2.1 erläutert wird, zuständig. Im Kapitel 6.2.2 wird die Klasse `RayDepthOptimizer` beschrieben, die den zweiten Ansatz, die strahlbasierte Optimierung, umsetzt.

Die optimierte Tiefenkarte wird schließlich in einer neuen Datei gespeichert.

6.2.1 Minimierung des Fehlers der MLS-Projektion

Dieser Ansatz zur Optimierung des Tiefenwertes eines Pixels, wird in `PixelDepthOptimizer` umgesetzt. Ein eindimensionales nichtlineares Verfahren, das Goldener-Schnitt-Verfahren (siehe 6.4.1), minimiert hier die Distanz des Punktes auf dem Sehstrahl zu der MLS-Projektion dieses Punktes. Die Projektion der Punkte vom Sehstrahl auf die MLS-Fläche wird in `MovingLeastSquares` durchgeführt.

In `MovingLeastSquares` wird das in Kapitel 5.2 beschriebene Verfahren implementiert. Der erste Teil dieser Prozedur bestimmt eine Hyperebene H, die die Punktmenge lokal approximiert. Die Normale dieser Ebene und der Punkt $q \in H$ minimieren den Fehler $e_{H_r}(n,q)$ aus (5.4). Dazu werden folgende Schritte durchlaufen: die Bestimmung der initialen Normale und die iterative Verbesserung von t und q. Die Iteration endet theoretisch mit der Konvergenz der Minimierung des Fehlers $e_{H_r}(n,q)$. In der Praxis wird die Iteration abgebrochen sobald sich die Normale nur noch geringfügig (z.B. um einem Winkel von maximal einem Grad) ändert. Die so definierte Hyperebene und die Normale bilden ein lokales Koordinatensystem, welches die Grundlage für den zweiten Teil des Verfahrens bildet. Der zweite Teil des Verfahrens sieht eine Approximation der Fläche und die Projektion des Referenzpunktes auf diese Approximation vor.

Teil I: Lokales Koordinatensystem

Listing 6.2: Lokales Koordinatensystem

```
calculateInitialNormalOfHyperplane();
/* Wiederhole solange, bis sich die Normale nur noch
   geringfügig ändert. */
while (fDifferenceOfNormals > fMaxAngle)
{
    moveHyperplaneInNormalDirection();
    recalculateNormal(fDifferenceOfNormals);
}
```

Schritt 1: Initiale Normale

Die initiale Normale wird wie in Gleichung (5.3) als kleinster Eigenvektor der Kovarianzmatrix bestimmt. Dabei kommen die Funktionen `Jacobi` und `eigsrt` aus [PTVF07] zum Einsatz, um die Eigenvektoren, nach der Größe der Eigenwerte sortiert, zu bestimmen.

Schritt 2: Bestimmung der Distanz t

In der Funktion `moveHyperplaneInNormalDirection` wird q entlang der Normalen n um den Betrag t verschoben, so dass der Fehler $e_{H_r}(n,t)$ aus (5.2) minimal wird. Da $e_{H_r}(n,t)$ lokal um das Minimum näherungsweise quadratisch ist kann hier ein Verfahren auf Basis der quadratischen Interpolation eingesetzt werden, welches deutlich schneller konvergiert als die Intervallschachtelung nach dem goldenen Schnitt. Der Ansatz des hier eingesetzten Verfahrens aus [PTVF07] wird in Kapitel 6.4.1 erläutert.

Ist das Minimum erreicht, so ergibt sich ein neuer Punkt $q = r + tn$.

Schritt 3: Minimierung in Abhängigkeit von q

Die Verbesserung von q durch ein Gradientenverfahren erfolgt in `recalculateNormal`. Hier wird $e_{H_r}(n,q)$ in Abhängigkeit von q lokal minimiert. Mit Hilfe des Gradienten

$$\nabla_q e_{H_r}(n,q) = \sum_i -2\langle n, p_i - q\rangle \,\Theta(\|p_i - q\|)\, n + \langle n, p_i - q\rangle^2 \,\nabla_q \Theta\left(\|p_i - q\|\right)$$

liefert das Konjugierte-Gradienten-Verfahren `frprmn` aus [PTVF07] ein entsprechendes q. Θ ist dabei die Gewichtsfunktion.

Aus q wird die neue Normale $n = r - q$ gebildet. Damit wird dann in Schritt zwei eine neue Distanz t bestimmt.

Teil II: Approximation und Projektion

Die zuvor ermittelten Parameter q und n bilden den Ursprung und die z-Achse eines lokalen Koordinatensystems, welches in `localCoordSystem` bestimmt wird. Auf Basis dieses Koordinatensystems wird in `weightedLeastSquares`

eine lokale Approximation $p(x, y)$ bestimmt. Damit ist die Projektion des Referenzpunktes in $\tilde{r} = q + p(0, 0)n$ gegeben.

6.2.2 Schnittpunkt einer lokalen Approximation mit Sehstrahl

In `RayDepthOptimizer` wird der zweite Ansatz zur Optimierung eines Tiefenwertes verfolgt. Hierbei wird von dem gegebenen Startpunkt ausgegangen. Durch Anwendung des folgenden Algorithmus konvergiert dieser Punkt gegen den gesuchten Punkt auf der MLS-Fläche.

Listing 6.3: Strahlbasierte Optimierung

```
/* gegeben ist der Startpunkt s, die Punktwolke P und der
   Sehstrahl R */
while (alpha > maxOptimizationError)
{
  /* bestimme Lokales Koordinatensystem */
  c = calcWeightedCloudCenter(s, P);
  generateWeightedCoordinateSystem(c);
  /* lokale Approximation */
  f = calcLocalApproximation(P);
  newPoint = calcRayIntersection(f, R);
  alpha = abs(newPoint - s);
  s = newPoint;
}
```

In jedem Schritt des Algorithmus muss der Schnittpunkt $R(\alpha)$ mit dem Approximationspolynom $f(x, y)$ bestimmt werden. Es gilt:

$$f(x, y) = \langle c,\, b(x, y)^T \rangle$$

mit $c = (c_0, c_1, ..., c_5)^T$ und $b(x, y) = (1, x, y, x^2, xy, y^2)$, sowie

$$R(\alpha) = v + \alpha r$$

mit $v = (v_0, v_1, v_2)^T$ und $r = (r_0, r_1, r_2)^T$. Damit ist folgende Gleichung zu lösen

$$\begin{aligned} v_2 + \alpha r_2 &= f(v_0 + \alpha r_0, v_1 + \alpha r_1) \\ &= \langle c,\ (1, v_0, v_1, v_0^2, v_0 v_1, v_1^2)^T \rangle \\ &\quad + \alpha \langle c,\ (0, r_0, r_1, r_0 v_0, r_1 v_0 + r_0 v_1, r_1 v_1)^T \rangle \\ &\quad + \alpha^2 \langle c,\ (0, 0, 0, r_0^2, r_0 r_1, r_1^2)^T \rangle. \end{aligned}$$

Diese lässt sich mit

$$\begin{aligned} X &= \langle c,\ (0, 0, 0, r_0^2, r_0 r_1, r_1^2)^T \rangle, \\ Y &= \langle c,\ (0, r_0, r_1, r_0 v_0, r_1 v_0 + r_0 v_1, r_1 v_1)^T \rangle - r_2 \quad \text{und} \\ Z &= \langle c,\ (1, v_0, v_1, v_0^2, v_0 v_1, v_1^2)^T \rangle - v_2 \end{aligned}$$

zusammenfassen:

$$0 = \alpha^2 X + \alpha Y + Z.$$

Daraus ergibt sich der folgende Skalar zur Bestimmung des Schnittpunktes s:

$$\alpha_{1,2} = -\frac{1}{2X} \left(Y \pm \sqrt{Y^2 - 4XZ} \right).$$

Ist die Diskriminante negativ, so verläuft der Sehstrahl parallel und es gibt keinen Schnittpunkt. Im Falle einer positiven Diskriminanten wird der betragsmäßig kleinere Wert verwendet, da der zugehörige Schnittpunkt näher an der PMD-Kamera liegt.

6.3 Segmentierung einer Teilpunktwolke

Die Segmentierung aus Kapitel 5.4 wird durch die Klasse `Segmentation` implementiert. Eine übergebene Punktwolke U wird daraufhin untersucht, ob sie bereits ein einzelnes Flächenstück repräsentiert, denn dann ist keine Segmentierung notwendig. Dazu wird durch die Methode des kleinsten Fehlerquadrates ein Polynom bestimmt. Sind die Abstände aller Punkte aus U zu diesem Polynom geringer als ein vorgegebener Schwellenwert, so handelt es sich bei U um ein Segment. Andernfalls wird die Segmentierung durchgeführt. Der Pseudocode in Listing 6.4 beschreibt die Umsetzung der Segmentierung. Abschließend wird durch

einen vorgegebenen Referenzpunkt r sowie der Richtung des zugehörigen Sehstrahls das für die Optimierung von r relevante Segment bestimmt.

Listing 6.4: Segmentierung

```
/* Gegeben ist die Umgebung setU */
while (usedCounter > wholePointCloud.size())
{
  /* Initiale Punktmenge bestimmen */
  aSegment = calcInitalSubPointCloud(setU);
  /* Ausreißer ausschließen */
  if (aSegment.size() < minNumberOfElements)
  {
    /* Bestimmung eines Segmentes durch den FS-Algorithmus */
    aSegment = forwardSearch(aSegment, setU);
    /* Sind zu wenig Punkte im Segment, so wird dieses
       ignoriert. Die entsprechenden Punkte werden als
       Ausreißer betrachtet. */
    if (aSegment.size() > minSizeOfSegment)
      allSegments.push_back(aSegment);
  }
  /* Die zum Segment gehörigen Punkte werden aus der Menge U
     entfernt. */
  setU -= aSegment;
  usedCounter += aSegment.size();
}
```

6.4 Numerische Minimierungsverfahren

In der zu dieser Arbeit entstandenen Implementierung der MLS-Approximation wurden drei Minimierungsverfahren eingesetzt.

In der Klasse `PixelDepthOptimizer` wird ein Punkt auf einer Geraden gesucht, so dass die Distanz dieses Punkts zu seiner MLS-Projektion minimal wird. Für diese eindimensionale nichtlineare Minimierung kommt das Verfahren des goldenen Schnittes zum Einsatz.

Das in `MovingLeastSquares` benötigte lokale Koordinatensystem wird iterativ durch die Minimierung zweier Energiefunktionen bestimmt. Zuerst wird die Energiefunktion $e_{H_r}(n,t)$ in Abhängigkeit vom Skalar $t \in \mathbb{R}$ minimiert. Die initiale Normale n ist dabei fix. Da es sich auch hier um ein eindimensionales

Minimierungsproblem handelt und $e_{H_r}(n,t)$ quadratisch ist kann eine schnellere Minimierung auf Basis der quadratischen Interpolation durchgeführt werden. Es kommt dabei das Verfahren von Brent in der Implementierung aus [PTVF07] zum Einsatz. Im weiteren Verlauf muss die zweite Energiefunktion $e_{H_r}(n,q)$ in Abhängigkeit von q minimiert werden. Da q auf der Hyperebene H_r liegen soll und n fix ist, wird hier eine zweidimensionale Minimierung durchgeführt. Diese erfolgt durch ein cg-Verfahren.

6.4.1 Eindimensionale nichtlineare Minimierung

Für die hier beschriebenen Verfahren werden drei Stützstellen $a,b,c \in \mathbb{R}$ mit $a < c < b$, sowie eine stetig Funktion $f : [a,b] \to \mathbb{R}$ vorausgesetzt, sodass $f(c) < f(a)$ und $f(c) < f(b)$.

Intervallschachtelung nach dem goldenen Schnitt

Die Intervallschachtelung nach dem golden Schnitt verkleinert das Intervall $[a,b]$ sukzessive, sodass die Intervallgrenzen gegen den Wert konvergieren, in dem f ein lokales Minimum hat (siehe dazu auch [JS04, Kap. 6.1.2]). Das Verhältnis $1 : \Phi$ in dem die Intervalle unterteilt werden, ist durch die golden Zahl $\Phi = \frac{1+\sqrt{5}}{2}$ bestimmt. Der Algorithmus wird in folgenden Schritten durchlaufen:

1. Das größere der beiden Intervalle $[a,c]$ und $[c,b]$ wird durch d unterteilt:

$$d = \begin{cases} c + \frac{b-c}{1+\Phi} & \text{falls } \; c < \frac{b+a}{2} \\ c - \frac{c-a}{1+\Phi} & \text{sonst} \end{cases}$$

2. Falls $f(d) \leq f(c)$ vertausche die Werte von c und d

3. Eine der beiden Stützstellen a und b wird durch d ersetzt. Ist $c < d$ erhält b den Wert aus d, sonst a.

4. Abbruch falls $b - a < \epsilon$

5. Gehe zu Schritt eins.

Nach Abbruch des Verfahrens ist an der Stelle c durch $f(c)$ eine Lösung für das Minimum von f gegeben.

Das Verfahren des goldenen Schnittes ist ein sicheres Verfahren, da die Bedingungen $f(c) < f(a)$ und $f(c) < f(b)$ bei der sukzessiven Verkleinerung des Intervalls erfüllt bleiben. Der hohe Aufwand einer solchen Intervallschachtelung ist der Nachteil dieses Algorithmus.

In `PixelDepthOptimizer` wird die Implementierung dieses Verfahrens aus [PTVF07] verwendet.

Eindimensionale nichtlineare Minimierung nach Brent

Eine deutlich schnellere Minimierung wird durch die quadratische Interpolation erreicht. Hier wird ausgenutzt dass eine Funktion in der Nähe eines Minimums näherungsweise parabolisch ist. Die quadratische Interpolation bestimmt das Minimum einer Funktion wie folgt:

1. Bestimme den Punkt d in dem das quadratische Interpolationspolynom $p(x)$ durch die Punkte a, b, c eine Extremstelle hat:

$$d = c - \frac{(c-a)^2 (f(c) - f(b)) - (c-b)^2 (f(c) - f(a))}{2(c-a)(f(c) - f(b)) - 2(c-b)(f(c) - f(a))}$$

2. Falls $c > d$ tausche c und d, sodass $a < c < d < b$ gilt.

3. Abbruch, falls $d - c$ ausreichend klein ist. Der kleinere der beiden Funktionswerte $f(c)$ und $f(d)$ approximiert das Minimum von f.

4. Ersetze a durch c falls $f(c) > f(d)$ sonst ersetzte b durch d.

5. Gehe zu Schritt eins.

Dieses Verfahren scheitert, wenn die Extremstelle in Schritt eins ein Maximum von p ist oder die Punkte a, b und c kollinear sind.

Durch die Kombination des Intervallschachtelungsverfahrens nach dem goldenen Schnitt und der inversen quadratischen Interpolation entsteht ein Verfahren, welches die Vorteile beider Methoden, die Sicherheit der Intervallschachtelung und die Geschwindigkeit der quadratischen Interpolation, vereint. Dabei wird immer, wenn sich die Stützstellen dazu eignen, ein Interpolationschritt durchgeführt und sonst ein Intervallschachtelungsschritt. Auf Basis der von Brent in [Bre71] beschriebenen Modifikationen wurde dieses Verfahren in [PTVF07] implementiert.

6.4.2 Gradientenverfahren

Die Zielfunktion $e_{H_r}(n, q)$ aus (5.4) soll in Abhängigkeit von q minimiert werden, wobei q auf der Hyperebene H_r liegt. Hierbei handelt es sich um eine quadratische Zielfunktion, so dass ein Gradientenverfahren zur Optimierung eingesetzt werden kann. Das Konjugierte-Gradienten-Verfahren, im Folgenden mit cg-Verfahren bezeichnet, ist besonders interessant, da es gegenüber herkömmlichen Gradientenverfahren mit geringfügig höherem Aufwand deutlich bessere Konvergenzeigenschaften aufweist.

Sei f eine konvexe quadratische Funktion

$$f(x) = \frac{1}{2}x^T Ax + b^T x + c,$$

wobei $A \in \mathbb{R}^{n\times n}$ eine symmetrische positiv definite Matrix, $b \in \mathbb{R}^n$ und $c \in \mathbb{R}$ sind. Bei einem Gradientenverfahren wird gewöhnlich in Richtung des steilsten Abstiegs

$$-\nabla f\,(x) = b - Ax$$

minimiert. Im ersten Schritt wird diese Richtung in einem Startwert x_0 bestimmt und in den darauf folgenden Schritten wird der Punkt x_k, in dem zuvor ein Minimum gefunden wurde, verwendet um die nächste Minimierungsrichtung festzulegen. Beim cg-Verfahren werden die Abstiegsrichtungen so gewählt, das sie zueinander A-konjugiert sind. Aufgrund der Wahl dieser Richtungen konvergiert das cg-Verfahren schon nach $n = \dim A$ Schritten (siehe [JS04, Kapitel 6.3]).

Der folgende Algorithums beschreibt das cg-Verfahren:

1. Mit dem Startwert x_0 wird das Residuum $g_0 = Ax_0 - b$ und die initiale Minimierungsrichtung $d_0 = -g_0$ bestimmt.

2. Abbruch falls $\|g_0\| < tol$

3. Berechnung der optimalen Schrittweite für die nächste Abstiegsrichtung:

$$\alpha_k = \frac{\|g_k\|^2}{d_k^T A d_k}$$

4. Näherung des Punktes, in dem die Funktion f ihr Minimum hat:

$$x_{k+1} = x_k + \alpha_k d_k$$

5. Aktualisiere Residuum:

$$g_{k+1} = g_k + \alpha_k A d_k$$

6. Bestimme die neue Abstiegsrichtung:

$$d_{k+1} = -g_{k+1} + \beta_k d_k$$

   wobei $\beta_k = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}$

7. Inkrementiere k und gehe zu Schritt zwei.

In der Praxis liegt die zu minimierende Funktion im Allgemeinen nicht als quadratische Form vor. Da die Matrix A dann nicht bekannt ist, können die entsprechenden Stellen im Algorithmus wie folgt angepasst werden (siehe [JS04, PTVF07]):

- Das Residuum entspricht dem Gradienten der Funktion:

$$g_k = \nabla f(x_k)$$

- Die optimale Schrittweite α_k lässt sich durch eine eindimensionale Minimierung in Richtung d_k ermitteln:

$$\alpha_k = \underset{\alpha>0}{\operatorname{argmin}} f(x_k + \alpha d_k)$$

In `MovingLeastSquares` wir die Implementierung des cg-Verfahrens von Press et al. aus [PTVF07] verwendet.

Kapitel 7

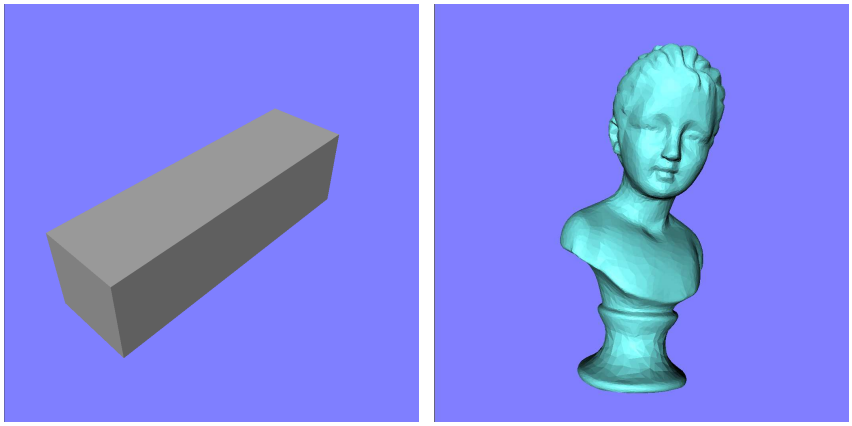
Ergebnisse und Auswertung

Zur Auswertung der beschriebenen Verfahren wurden zwei Datensätze zu je 42 Tiefenkarten erzeugt. Die Tiefenkarten wurden von den simulierten PMD-Kameras aufgezeichnet und haben eine Auflösung von 160×120 Pixeln. Die geodätische Sphäre, auf der die Kameras positioniert sind, hat einen Durchmesser von 6 Metern und die aufgezeichneten Objekte befinden sich im Zentrum dieser Sphäre.

Als Testobjekte kamen ein einfaches Objekt mit ebenen Flächen und scharfen Kanten in Form eines Quaders sowie ein komplexeres Objekt, eine Büste mit vielen kleinen Unebenheiten, zum Einsatz (siehe Abbildung 7.1). Die Büste bietet außerdem die Möglichkeit, die Auswirkungen des MLS-Approximationsverfahrens auf mehrere separate Flächenstücke zu untersuchen, denn je nach Kameraausrichtung werden von der Büste separate Flächenstücke erfasst (Beispiel: von oben betrachtet sind drei Flächenstücke zu sehen, und zwar der Kopf und die beiden Schultern). Des Weiteren wurde zur Analyse der Verfahren zu jeder Tiefenkarte eine zweite Tiefenkarte mit theoretischen Idealwerten erzeugt (Bsp. in Abbildung 7.3).

7.1 Voraussetzung der lokalen Approximation einer Punktmenge

Die Grundlage der lokalen Approximation bildet ein lokales Koordinatensystem. Dazu werden, wie in Kapitel 4.4 beschrieben, die Eigenwerte einer Kovarianzmatrix bestimmt und der Eigenvektor zum kleinsten Eigenwert verwendet. Hierbei kann der kleinste Eigenwert theoretisch eine algebraische Vielfachheit haben, die



(a) Quader (b) Büste

Abbildung 7.1

größer als eins ist. In diesem Fall ist die Punktmenge symmetrisch, was aufgrund des Rauschens in den Tiefenkarten sehr unwahrscheinlich ist. Außerdem muss die Ausdehnung σ der Punktmenge parallel zur Fläche dem maximalen Rauschen ν entsprechen. Gilt jedoch $\sigma \leq \nu$, so sagt der Eigenvektor zum kleinsten Eigenwert nichts über die Fläche aus (siehe Bild 4.5). Damit kann auch keine geeignete Approximation der Fläche gefunden werden.

Für die hier beschriebene lokale MLS-Approximation einer Punktmenge muss also gelten:

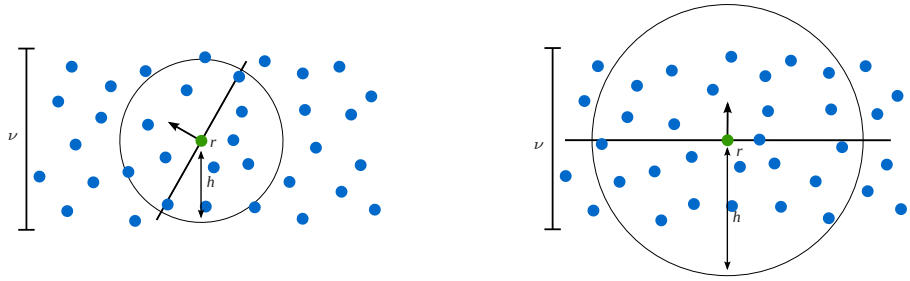
$$\sigma > \nu.$$

7.2 Rauschen

Eine Herausforderung bei der Optimierung der PMD-basierten Tiefendaten ist das Rauschen. Je größer das Rauschen, desto größer muss auch der Parameter h der Gewichtsfunktion gewählt werden, da dieser den Einflussbereich der Punktmenge auf die beschriebenen Verfahren reguliert. h wird durch das maximal vorkommende Rauschen ν nach unten beschränkt:

$$h > \frac{\nu}{2}.$$

Dabei entspricht ν der maximalen Ausdehnung der Punktwolke orthogonal zur Fläche. Ist der Parameter h zu klein, so kann ein WLS-Verfahren ungeeignete



(a) (b)

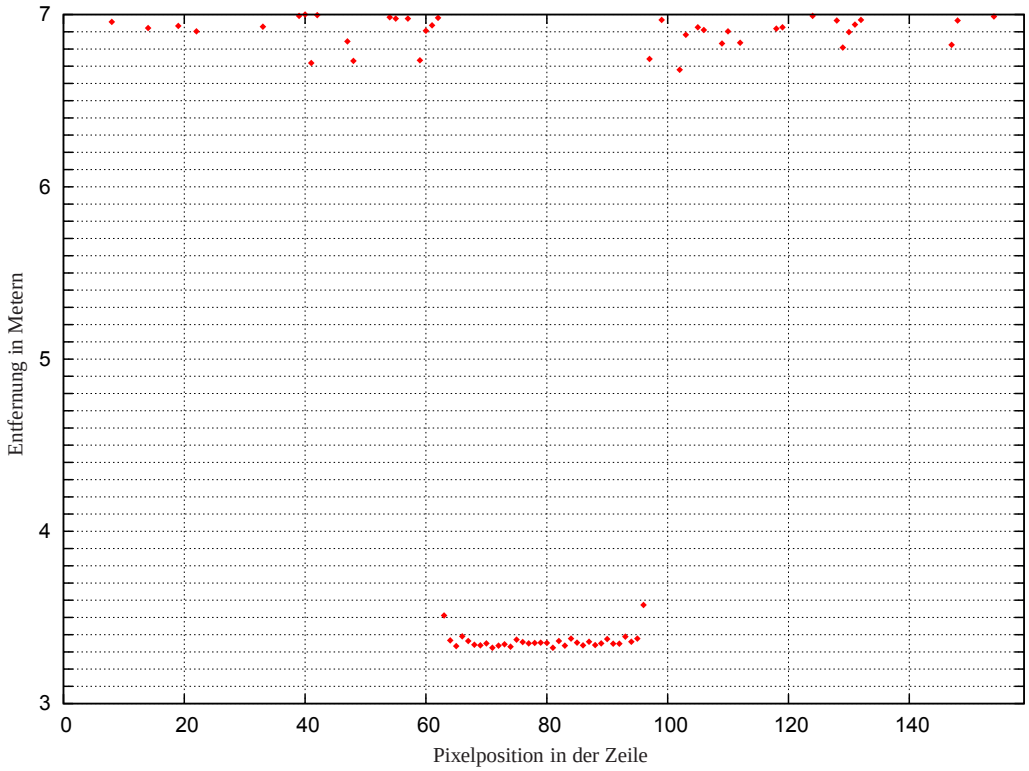
Abbildung 7.2: In (a) ist $h \leq \frac{\nu}{2}$ und damit kann die Ausrichtung der durch die Punktmenge repräsentierten Fläche nicht ermittelt werden. Mit $h > \frac{\nu}{2}$ wie in (b) wird die Lage der Fläche korrekt bestimmt.

Approximationen der Fläche liefern (siehe Abbildung 7.2).

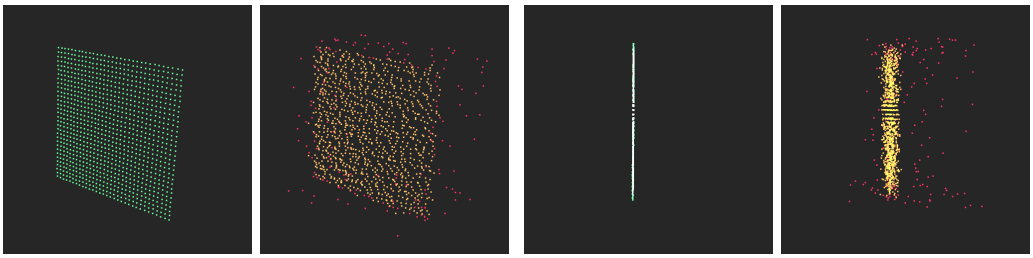
Bei einigen Testaufnahmen mit der PMD-Kamera hat sich ein maximales Rauschen von ca. 10 cm bei einer Entfernung des Objektes von ca. 3 bis 4 Metern ergeben. Das Rauschen eines PMD-Bildes nimmt zwar vom Mittelpunkt des Bildes radial nach außen zu, aber diese Zunahme ist relativ gering und wird hier nicht weiter untersucht. Die Parameter des PMD-Simulators wurden so eingestellt, dass die damit erzeugten Tiefenkarten ebenfalls ein maximales Rauschen von ca. 10 cm aufweisen. Ein Beispiel für die verrauschten Daten einer Tiefenkarte ist in Abbildung 7.3 zu sehen.

7.3 Flying Pixel Erkennung

Der Schwellenwert, der für die Erkennung der Flying Pixel festgelegt wird, hängt von zwei Faktoren ab: dem Rauschen und dem Winkel in dem die Blickrichtung der Kamera auf die Punktmenge trifft. Das maximale Rauschen ist in dieser Untersuchung fix. Bei der Neigung der Fläche zur Kamera verhält es sich so, dass die aus der Tiefenkarte gewonnenen Punkte bei einem sehr spitzen Winkel einen größeren euklidischen Abstand zu ihren Nachbarn haben als bei einem näherungsweise rechten Winkel. Wird der Schwellenwert `fpThreshold` so festgelegt, dass alle Flying Pixel erkannt werden, so werden auch „gute“ Pixel auf einer solchen schrägen Fläche als Flying Pixel markiert (siehe Abbildung 4.6). Eine Vergrößerung von `fpThreshold` behebt dieses Problem, bewirkt aber, dass nicht mehr alle Flying Pixel erkannt werden (Abb. 7.4 (b)). Theoretisch ist das maximale



(a)



(b) (c)

Abbildung 7.3: Drei Ansichten der Daten aus einer Tiefenkarte. Die aufgezeichnete Fläche hat eine Entfernung von der Kamera von ca. 3,5 m. In (a) ist eine Zeile der Tiefenkarte abgebildet, auf der ein maximalen Rauschen von ca. 10 cm zu erkennen ist. Die aus der Tiefenkarte gewonnene Punktwolke ist in (b) von schräg vorne und in (c) von der Seite dargestellt. Die linken Abbildungen in (b) und (c) zeigen ideale Daten, während die rechten Abbildungen simulierte Realdaten mit rot markierten Flying Pixel darstellen.

Rauschen der ideale Wert für `fpThreshold`, denn zwei benachbarte Pixel, die keine Flying Pixel sind, haben eine maximale Tiefendifferenz, die dem maximalen Rauschen entspricht. In der Praxis kann dieser Wert etwas kleiner gewählt werden, da das maximale Rauschen nur durch wenige Ausreißer erreicht wird.

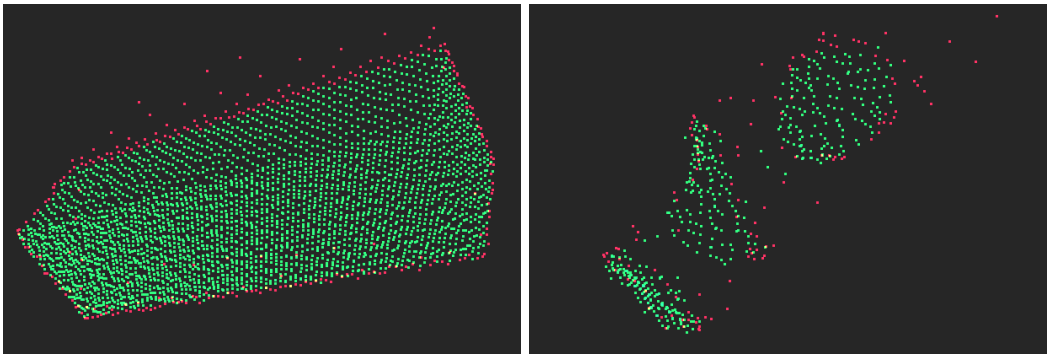
Ist `fpThreshold` so festgelegt, dass alle Flying Pixel erkannt werden, so kann der zweite Schwellenwert `nonFPThresh` definiert werden, um die fehlerhafte Markierung von Pixeln, die eine glatte Fläche repräsentieren zu vermeiden. Dieser Schwellenwert hängt offensichtlich von der Steigung der Fläche und dem maximalen Rauschen ab. Die in dieser Arbeit durchgeführten Optimierungen haben gezeigt, dass gute Ergebnisse erzielt werden wenn sich `nonFPThresh` nur geringfügig von `fpThreshold` unterscheidet.

7.4 Der Parameter der Gewichtsfunktion

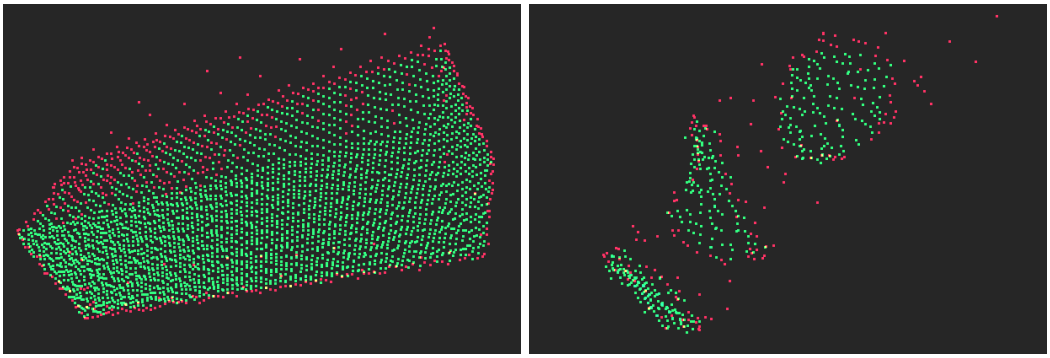
Der Parameter *h* der Gewichtsfunktion spielt bei der Optimierung eine zentrale Rolle. Je kleiner er gewählt wird, desto lokaler wird die Flächenrekonstruktion. Er regelt also die Genauigkeit der rekonstruierten Fläche und bestimmt den Definitionsbereich des Projektionsoperators.

In Kapitel 5.3 wird die Verwendung eines adaptiven Parameters beschrieben. Dieser hängt von der Dichte der Punktmenge in der lokalen Umgebung des zu optimierenden Punktes ab. Damit ist es möglich, die Genauigkeit der Flächenrekonstruktion an Stellen mit einer hohen Punktdichte zu verbessern. Die hier betrachteten Daten sind stark verrauscht, wodurch der Gewichtsparameter *h* nach unten beschränkt ist (siehe Kapitel 7.2). Damit ist die Einsatzmöglichkeit und auch die Wirkung der adaptiven Gewichtsfunktion bei dieser Problemstellung sehr begrenzt. In der Praxis hat sie keine Verbesserung der Optimierung bewirkt.

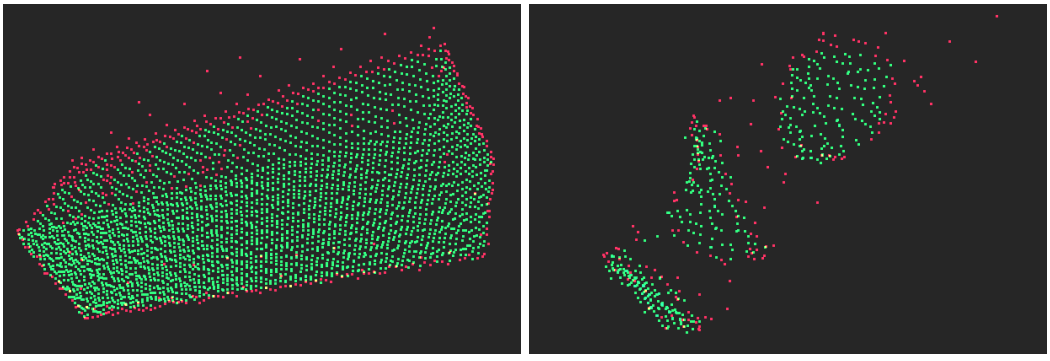
Für die Gaußsche sowie Wendlands Gewichtsfunktion kann im Allgemeinen ein Parameter gewählt werden, der leicht über der Hälfte des maximalen Rauschens liegt. So wird größtmögliche Genauigkeit erreicht. Ein größerer Parameter macht nur dann Sinn, wenn das durch die PMD-Kameras aufgezeichnete Objekt aus großen Flächenstücken besteht, wie zum Beispiel der Quader, und eine Segmentierung eingesetzt wird.



(a) `fpThreshold` = 0.12



(b) `fpThreshold` = 0.07



(c) `fpThreshold` = 0.07, `nonFPThresh` = 0.065

Abbildung 7.4: Die Abbildungen zeigen die aus jeweils einer Tiefenkarte hervorgegangenen Punktwolken, in denen die Flying Pixel (FP) rot markiert sind. In (a) ist der Schwellenwert `fpThreshold` zur FP-Erkennung für den Quader sehr gut, aber bei der Büste wurden einige FP nicht erkannt. Dann wurde `fpThreshold` so weit verkleinert, dass alle FP in der Punktwolke der Büste erkannt werden (b), allerdings sind nun auch einige „gute“ Pixel in der linken Abbildung markiert worden. Um letzteres wieder auszugleichen wurde in (c) der zweite Schwellenwert `nonFPThresh` so festgelegt, dass die Menge der fehlerhaft markierten Pixel reduziert wird.

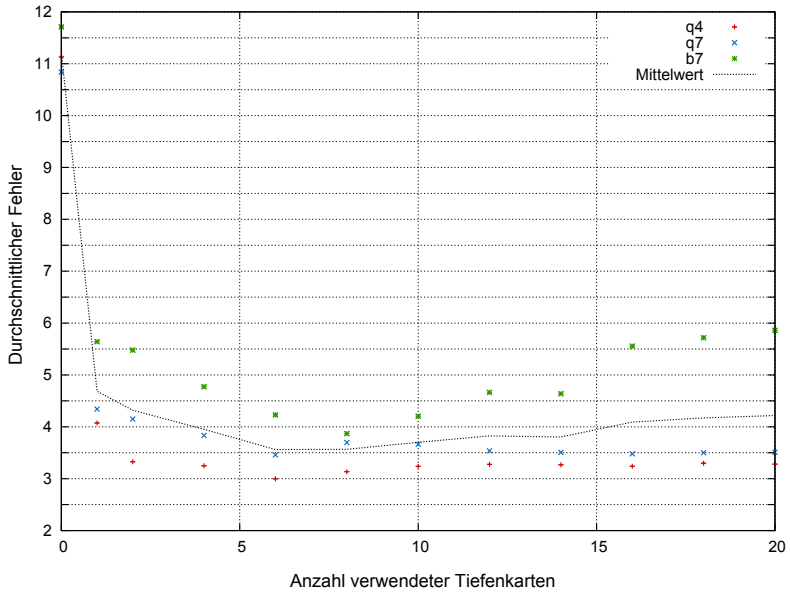


Abbildung 7.5: In diesem Diagramm gibt die Ordinate den durchschnittlichen Fehler einer optimierten Tiefenkarte im Vergleich zu den Idealdaten an. Auf der Abszisse wird die Anzahl der zur Optimierung verwendeten Tiefenkarten angegeben. Exemplarisch sind hier die Ergebnisse von drei Tiefenkarten und deren Mittelwert angegeben. Bis zu einer Kameraanzahl von sechs nimmt die Qualität der Optimierung zu. Werden die Daten von mehr Kameras verwendet, verschlechtert sich das Optimierungsergebnis wieder.

7.5 Anzahl der zur Optimierung verwendeten Kameras

Für die Optimierung einer Tiefenkarte stehen die Daten aus 42 Kameras zur Verfügung. Die Menge der verwendeten Daten kann also von einer Tiefenkarte, der zu optimierenden Karte, oder von bis zu 42 Tiefenkarten stammen. Wird nur eine Tiefenkarte eingesetzt, so sind die Informationen über die Objektoberfläche sehr lückenhaft. Die Optimierung reduziert zwar in diesem Fall das Rauschen deutlich, aber die Objektoberfläche wird dabei auch stark geglättet. Auch die Flying Pixel lassen sich aufgrund mangelnder Daten schlechter korrigieren. Gegen die Verwendung aller verfügbaren Daten spricht das große Datenvolumen und der daraus resultierende Rechenaufwand.

Die zu optimierende Tiefenkarte sei von einer Referenzkamera aufgezeichnet. Von dieser Kamera wird ein Objekt von einer Seite aufgezeichnet. Die Kameras, die auf der geodätischen Sphäre der Referenzkamera gegenüberliegen, erfassen das Objekt von der Rückseite. Diese Informationen sind für die Optimierung der Referenztiefenkarte nicht relevant und können die Optimierung in einigen Fällen

sogar negativ beeinflussen.

Es hat sich herausgestellt, dass sich die Tiefenkarte der Referenzkamera im Allgemeinen am Besten optimieren lässt, wenn die Tiefenkarten der benachbarten Kameras verwendet werden. Als Nachbarn werden dabei die Kameras bezeichnet, die auf der geodätischen Sphäre durch eine Kante mit der Referenzkamera verbunden sind. Das sind bei der hier verwendeten Kamerasphäre, je nach Position der Referenzkamera, fünf oder sechs Nachbarn. Werden mehr Tiefenkarten verwendet, verbessert sich das Optimierungsergebnis kaum bis gar nicht, teilweise wird es sogar wieder schlechter (siehe Abbildung 7.5). Je mehr Tiefenkarten zur Optimierung der Referenzkarte hinzugezogen werden, desto größer wird auch der Rechenaufwand.

7.6 Qualität der optimierten Tiefenkarten

Die Qualität der optimierten Tiefenkarten wird hier auf zweierlei Weise untersucht. Zum einen wird eine aufgezeichnete Tiefenkarte und ihr optimiertes Pendant mit der entsprechenden idealen Tiefenkarte aus dem PMD-Simulator verglichen. Die zweite Betrachtung bezieht sich auf die Konsistenz der optimierten Daten.

7.6.1 Analyse der optimierten Daten anhand der Idealdaten

Einen ersten Eindruck über die Qualität der optimierten Daten liefert der Vergleich mit den theoretischen Idealdaten. Hierzu wurde aus den beiden zur Verfügung stehenden Datensätzen je eine Tiefenkarte gewählt:

Quader 1: Diese Tiefenkarte repräsentiert zwei große ebene Flächen, die durch eine Kante miteinander verbunden sind (siehe Abbildung 7.7).

Büste 7: Hier wurde die Büste in etwa von unten vorne aufgezeichnet (siehe Abbildung 7.8). Dadurch kommt es zu mehreren kleinen Flächenstücken, die andere Teile der Oberfläche überdecken.

Für jeden Pixel einer Tiefenkarte wurde die Differenz seines Tiefenwertes zu dem korrespondierenden Tiefenwert in der idealen Tiefenkarte ermittelt. Der Mittelwert dieser Differenzen über eine Tiefenkarte dient schließlich zum Vergleich

Optimierungsverfahren	Quader 1			Büste 7		
	Fehler (mm)	Fehler (FP) (mm)	Zeit (sec)	Fehler (mm)	Fehler (FP) (mm)	Zeit (sec)
keins (Realdaten)	11.15	617	–	12.01	495	–
RDO	2.65	703	24	4.34	610	3
RDO+MLSS	2.52	707	61	4.38	610	6
RDO+SEG	1.57	905	3144	4.54	606	184
RDO+SEG+MLSS	1.59	905	3166	4.66	606	187
PDO	2.74	959	1158	5.22	647	112
PDO+SEG	1.63	794	4636	5.51	578	286

Tabelle 7.1: Durchschnittlicher Fehler von einigen optimierten Tiefenkarten in Bezug zur idealen Tiefenkarte. Die Tiefenkarten Quader 1 und Büste 7 wurden unter Verwendung der Tiefenkarten der fünf angrenzenden Kameras optimiert. In Tiefenkarte Quader 1 wurden 258 Flying Pixel unter 2867 für das Objekt Quader relevanten Pixel ausgemacht und bei Büste 7 wurden 165 von 570 Pixeln als Flying Pixel klassifiziert. Der für die Optimierung nötige Parameter der Gewichtsfunktion wurde für Quader 1 mit h=0.05 und für Büste 7 mit h=0.04 festgelegt. Die Optimierungen wurden unter WinXP an einem Athlon X2 4200+ (2.2Ghz) mit 3.5GB Ram durchgeführt.

zwischen den optimierten Daten und den Realdaten. Dabei werden die Pixel, welche in den Realdaten als Flying Pixel gekennzeichnet wurden, separat betrachtet. In Tabelle 7.1 sind diese mittleren Fehler der Realdaten und optimierten Daten aufgeführt.

Die Betrachtung der Pixel die nicht als Flying Pixel markiert wurden zeigt bei allen Varianten der Optimierung eine deutliche Verbesserung der Tiefendaten. Bei Quader 1 wurde eine Fehlerreduktion von mindestens 75 Prozent, bei der Büste 7 um mindestens 50 Prozent erreicht. Der große Unterschied zwischen den Fehlern der beiden Tiefenkarten ist in der Beschaffenheit der Objekte begründet. Die Optimierung zweier großer ebener Flächen ist deutlich einfacher als die Optimierung mehrerer kleiner Flächenstücke.

Das Verfahren „Schnittpunkt einer lokalen Approximation mit Sehstrahl“ (Abk.: RDO) unterscheidet sich von der „Minimierung des Fehlers der MLS-Projektion“ (Abk.: PDO) qualitativ kaum, allerdings benötigt ersteres Verfahren nur eine Bruchteil der Zeit um die Optimierung durchzuführen.

Bei Quader 1 ist eine Segmentierung (Abk.: SEG) der Fläche in die beiden ebenen Flächenstücke sinnvoll, was sich auch in dem mittleren Fehler bemerkbar macht. Der Einsatz der Segmentierung sorgt bei beiden Verfahren (RDO und PDO) für eine Fehlerreduktion von 85 Prozent gegenüber den Realdaten. Diese Verbesserung ist auch deutlich in den Repräsentationen der Tiefenkarten zu erkennen: in den Bildern 7.7c und 7.7g ist eine deutliche Abrundung der Kante zu

erkennen, wohingegen die Bilder 7.7e und 7.7h eine scharfe Kante aufweisen. Der Einsatz der Segmentierung bei der Tiefenkarte Büste 7 bringt keine Verbesserung mit sich, da hier auch keine Segmente vorliegen, die eine Kante bilden. Einen offensichtlichen Nachteil der Segmentierung ist die deutlich höhere Laufzeit der Optimierung.

Ein zusätzlicher Moving-Least-Squares-Schritt (Abk.: MLSS) im Anschluss an die RDO-Optimierung zur Verbesserung der Ergebnisse zeigt kaum praktische Auswirkungen.

Der Vergleich der Pixel die in den Realdaten als Flying Pixel (Abk.: FP) klassifiziert wurden mit den Idealdaten deutet auf eine Verschlechterung der Daten durch die Optimierung hin. Doch die Konsistenzanalyse der Flying Pixel in Abschnitt 7.6.2 zeigt, dass die Ursache dieses großen Fehlers in den Idealdaten liegt.

7.6.2 Konsistenz der optimierten Daten

Besonders interessant ist, ob und wie stark sich die Konsistenz eines Datensatzes von Tiefenkarten durch die Optimierung verbessert. Um die Konsistenz zu messen, wird im Folgenden ein Verfahren beschrieben, welches auf dem Hausdorff-Abstand beruht. Der Hausdorff-Abstand misst die Distanz zweier Menge. Je besser diese beiden Mengen wechselseitig übereinstimmen, desto kleiner ist der Hausdorff-Abstand. Dieses Verfahren wird hier auf die Tiefenkarten angewandt und es werden nicht nur zwei, sonder alle 42 Tiefenkarten paarweise verglichen. Je konsistenter der Datensatz ist, desto kleiner ist der so definierte Abstand.

Verfahren zur Konsistenzmessung

Seien $p_i^k \in \mathbb{R}^3$ die Punkte, die aus den Pixeln der Tiefenkarte T_k einer Kamera C_k gewonnen werden. In der Tiefenkarte einer zweiten Kamera C_l wird der zum Punkt p_i^k gehörige Pixel in T_l ermittelt. Dieser Pixel liefert durch den enthaltenen Tiefenwert einen neuen Punkt $p_i^l \in \mathbb{R}^3$ und es kann die euklidische Distanz $\|p_i^l - p_i^k\|$ bestimmt werden. Zu p_i^k wird für alle Kameras C_l diese Distanz ermittelt. Die kleinste dieser Distanzen

$$d_i^k = \min \left\{ \|p_i^k - p_i^l\| \,\middle|\, l = 0, \ldots, 41,\ l \neq k \right\}$$

wird als Fehler für den Pixel, aus dem p_i^k hervorging, verwendet. Für alle Punkte p_i^k gibt das größte d_i^k den Fehler der Tiefenkarte T_k vor:

$$d_k = \max_i d_i^k. \tag{7.1}$$

Um nun eine Distanz für einen ganzen Datensatz von Tiefenkarten zu bestimmen, wird das Maximum aus allen d_k gewählt:

$$d = \max_{k \in \{0,\ldots,41\}} d_k.$$

Mit d wird im Folgenden der Konsistenzfehler eines Datensatzes angegeben. Um diese Messung gegen Ausreißer stabil zu machen, reicht eine Abwandlung von Gleichung (7.1). Soll ein Ausreißeranteil von j Prozent zugelassen bzw. bei der Konsistenzanalyse ignoriert werden, so kann aus den Distanzen d_i^k statt dem Maximum das Element mit dem r_k-ten Rang gewählt werden:

$$r_k = |\{d_i^k \,|\, p_i^k \in T_k\}| \left(1 - \frac{j}{100}\right).$$

Ergebnisse der Konsistenzmessung

Die Konsistenzanalyse wurde für die simulierten Realdaten, die Idealdaten und die optimierten Daten der Objekte Quader und Büste durchgeführt. Die Tabellen 7.2a und 7.2b enthalten die Ergebnisse für alle Pixel, die in den Realdaten nicht als Flying Pixel markiert wurden. Die Flying Pixel wurden separat untersucht und die Ergebnisse in Tabelle 7.2c festgehalten.

Die Ergebnisse zeigen, dass alle optimierten Datensätze einen kleinen Anteil Ausreißer, also Pixel die sehr inkonsistent sind, aufweisen (siehe 0%-Spalte in 7.2a und 7.2b). Werden die Ausreißer bei der Konsistenzanalyse ignoriert, so sind alle optimierten Datensätze deutlich konsistenter als die Realdaten. Das Verfahren, welches durch die Schnittpunkte lokaler Approximationen mit dem Sehstrahl (RDO) eine Optimierung der Tiefenwerte erzielt, ergibt bei beiden Datensätzen eine deutliche Konsistenzverbesserung bei maximal einem Prozent Ausreißern. Durch den zusätzlichen MLS-Schritt (RDO+MLSS) wird eine leichte Verbesserung erreicht. Durch den Einsatz des Segmentierungsverfahrens (RDO+SEG) verändert sich die Konsistenz der Datensätze, bei entsprechendem Ausreißeranteil, kaum. Bei der Minimierung der Distanz zur MLS-Fläche (PDO) sind die Resul-

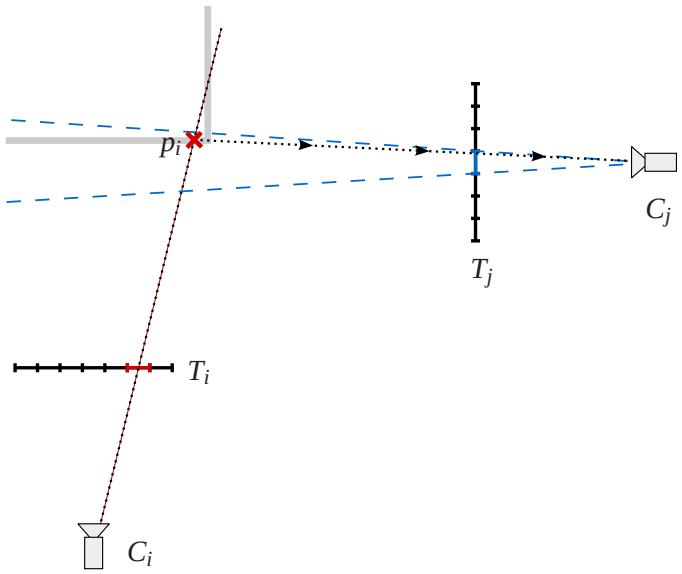


Abbildung 7.6: Die diskreten Tiefenkarten können auch bei idealen Daten Inkonsistenzen enthalten. Hier wird durch einen Sehstrahl der Kamera C_i und den Tiefenwert in dem roten Pixel der Punkt $p_i \in \mathbb{R}^3$ bestimmt. Der korrespondierende blaue Pixel in der Tiefenkarte T_j enthält einen Tiefenwert der aus den Objekten in dem gekennzeichneten Kegel resultiert. In diesem Beispiel könnte es sich um einen Flying Pixel oder einem Hintergrundpixel handeln, sodass die Tiefenwerte hier sehr inkonsistent sind.

tate ohne den Ausschluss von Ausreißern deutlich besser als bei den durch RDO optimierten Tiefenkarten. Wird jedoch ein Ausreißeranteil bei der Analyse ausgeschlossen, so unterscheiden sich die Konsistenzen der beiden Verfahren kaum noch; die durch RDO erreichten Konsistenzen sind dann sogar etwas besser.

Die Konsistenzanalyse der Flying Pixel offenbart, dass diese auch nach der Optimierung deutlich weniger konsistent sind als die „guten Pixel“. Trotzdem wurde eine deutliche Reduzierung des Konsistenzfehlers gegenüber den Realdaten um über 80 Prozent erreicht (bei einem Ausreißeranteil von fünf Prozent).

Selbst die idealen Daten sind nicht vollständig konsistent. Diese Inkonsistenz resultiert aus der Diskretheit der Daten. Ein Pixel der hier verwendeten Tiefenkarten repräsentiert von einem drei Meter entfernten Objekt ca. einen Quadratzentimeter Oberfläche. Betrachtet man zu dem resultierenden Punkt im $\mathbb{R}^3$ den korrespondierenden Tiefenwert einer Tiefenkarte, die aus einer anderen Perspektive aufgezeichnet wurde, so kann dies zu einem gänzlich anderem Tiefenwert führen (siehe Abbildung 7.6). Dies macht sich besonders in den Randbereichen bemerkbar, die in den Realdaten in der Regel als Flying Pixel auftreten. In diesen Randbereichen ist der Idealdatensatz stark inkonsistent wie Tabelle 7.2c belegt.

Die Auswirkung der Segmentierung auf die Tiefenkarten kann durch die Konsistenzanalyse nicht vollständig erfasst werden. Sind nämlich die Kanten des aufgezeichneten Objektes in allen Tiefenkarten eine Datensatzes geglättet, so kann dies durchaus zu einer guten Konsistenz führen.

Optimierungsverfahren	Ausreißeranteil			
	0%	1%	2%	5%
keins (Realdaten)	0.0453	0.0256	0.0219	0.0172
keins (Idealdaten)	0.0067	0.005	0.0047	0.0042
RDO	0.2108	0.0057	0.0052	0.0046
RDO+MLSS	0.097	0.0055	0.005	0.0045
RDO+SEG	0.2682	0.0077	0.0056	0.0045
RDO+SEG+MLSS	0.297	0.007	0.0056	0.0046
PDO	0.0662	0.021	0.0082	0.0049
PDO+SEG	0.04	0.007	0.0054	0.0045

(a) Quader (ohne Flying Pixel)

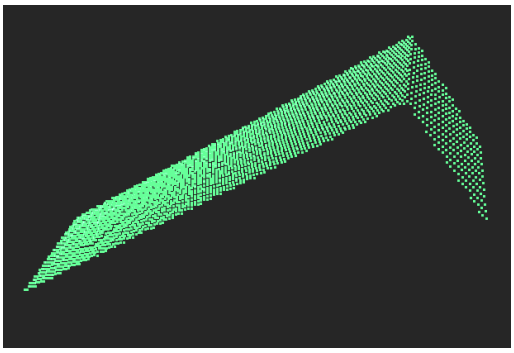
Optimierungsverfahren	Ausreißeranteil			
	0%	1%	2%	5%
keins (Realdaten)	0.0523	0.028	0.022	0.0154
keins (Idealdaten)	0.0077	0.0058	0.0053	0.0047
RDO	0.0508	0.0087	0.0066	0.0053
RDO+MLSS	0.0442	0.0089	0.0065	0.0055
RDO+SEG	0.135	0.029	0.0078	0.0059
RDO+SEG+MLSS	0.2206	0.0361	0.0128	0.0058
PDO	0.0436	0.0123	0.0095	0.006
PDO+SEG	0.0538	0.0274	0.019	0.0068

(b) Büste (ohne Flying Pixel)

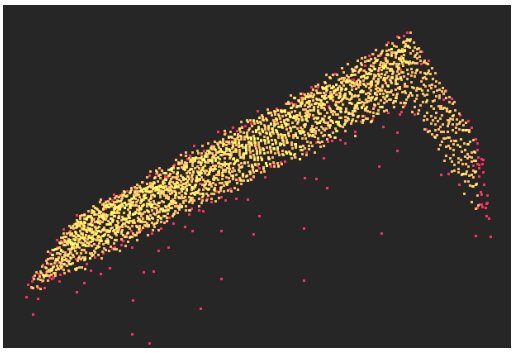
Optimierungsverfahren	Flying Pixel Quader		Flying Pixel Büste	
	Ausreißeranteil		Ausreißeranteil	
	0%	5%	0%	5%
keins (Realdaten)	1.8458	1.0105	2.4949	1.7764
keins (Idealdaten)	3.7879	3.6789	6.1012	0.4724
RDO	3.6961	0.206	5.9858	0.2796
RDO+MLSS	3.696	0.206	5.9858	0.2796
RDO+SEG	3.6782	0.1977	0.3161	0.2514
RDO+SEG+MLSS	3.6782	0.1977	0.3161	0.2514
PDO	3.6832	0.2146	5.9858	0.2403
PDO+SEG	3.6782	0.1895	0.303	0.2476

(c) Konsistenzanalyse der Flying Pixel

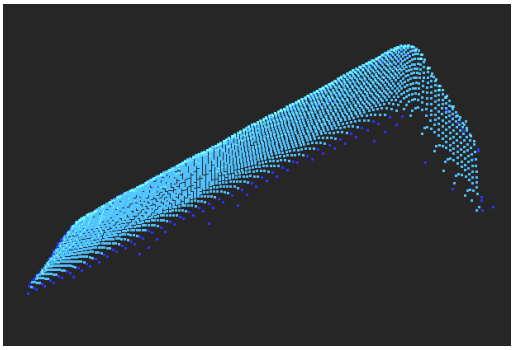
Tabelle 7.2: Die Tabellen zeigen die Ergebnisse der Konsistenzanalyse der Datensätze Quader und Büste (je 42 Tiefenkarten). Es wurden die Realdaten und idealen Daten sowie die optimierten Datensätze untersucht. Die Konsistenzfehler der Datensätze wurden unter Annahme eines Ausreißeranteils von $j \in \{0, 1, 2, 5\}$ Prozent bestimmt. Dabei wurden in (a) und (b) nur die Pixel berücksichtigt, die in den Tiefenkarten welche die Realdaten enthalten, nicht als Flying Pixel markiert wurden. In Tabelle (c) sind die Konsistenzfehler der Flying Pixel aufgeführt.



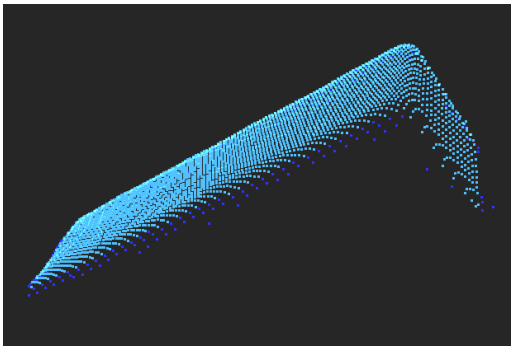
(a) Simulierte Idealdaten



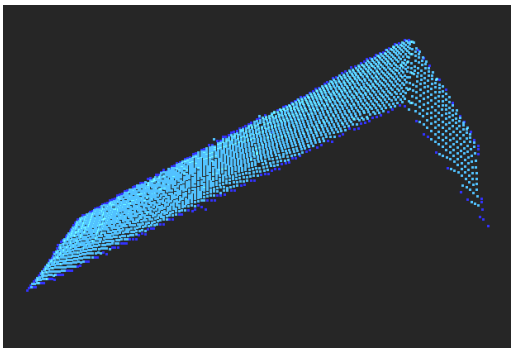
(b) Simulierte Realdaten



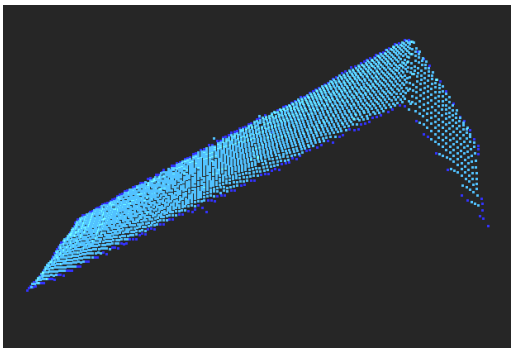
(c) RDO



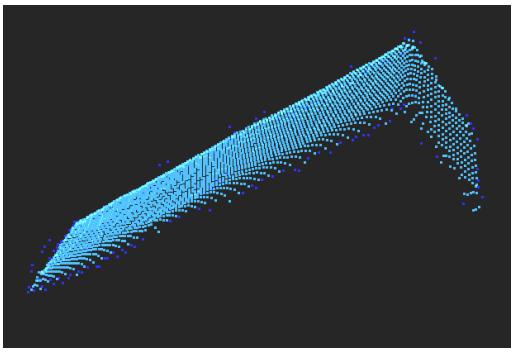
(d) RDO + MLS-Schritt



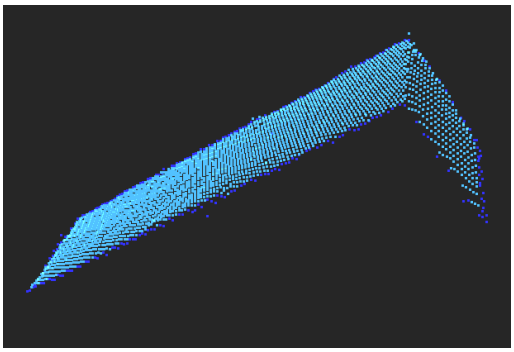
(e) RDO + Segmentierung



(f) RDO + Segmentierung + MLS-Schritt

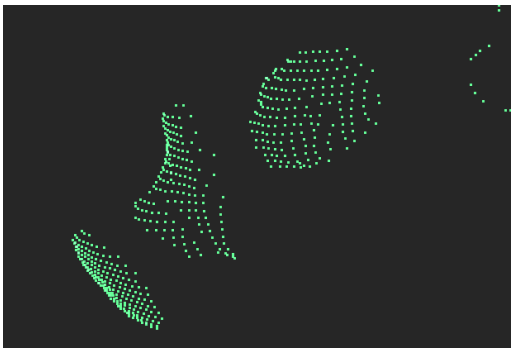


(g) PDO

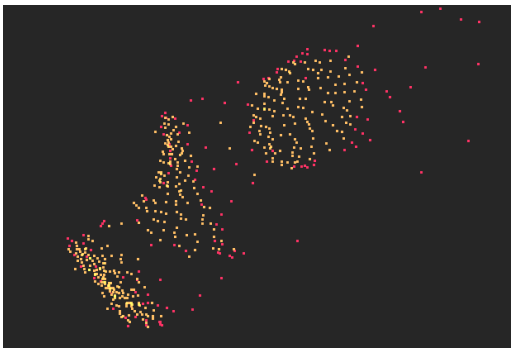


(h) PDO + Segmentierung

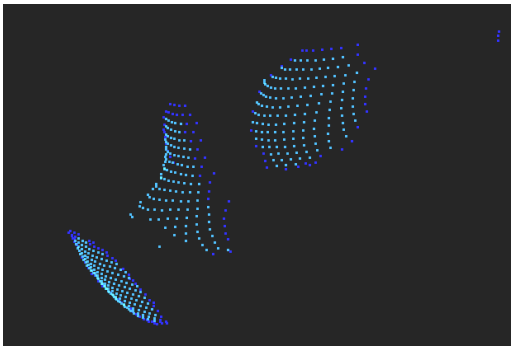
Abbildung 7.7: Darstellung der Punktwolken aus den idealen, realen und optimierten Tiefenkarten einer Kamera, die das Objekt Quader aufgezeichnet hat.



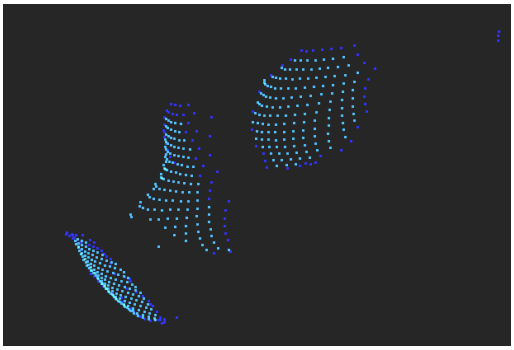
(a) Simulierte Idealdaten



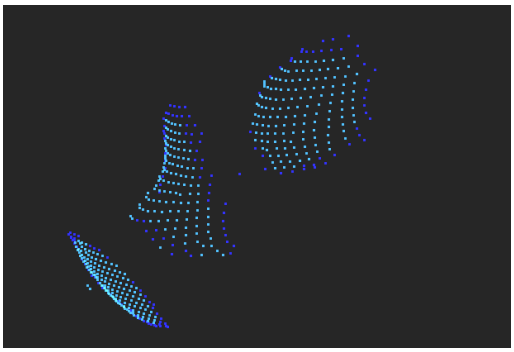
(b) Simulierte Realdaten



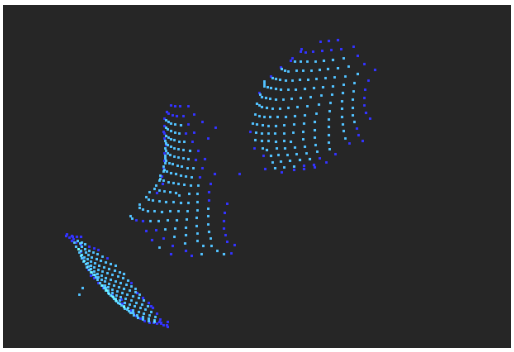
(c) RDO



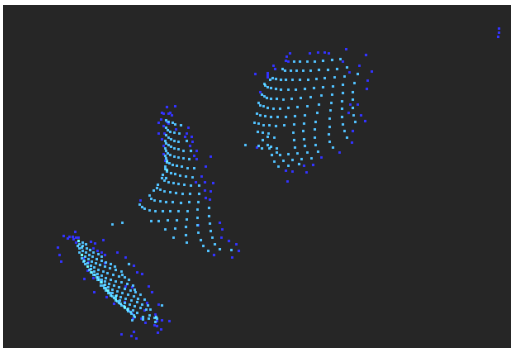
(d) RDO + MLS-Schritt



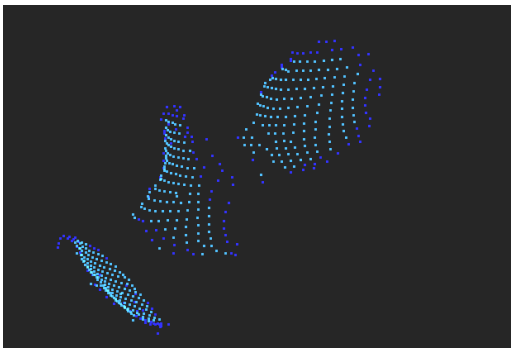
(e) RDO + Segmentierung



(f) RDO + Segmentierung + MLS-Schritt



(g) PDO



(h) PDO + Segmentierung

Abbildung 7.8: Darstellung der Punktwolken aus den idealen, realen und optimierten Tiefenkarten einer Kamera, die das Objekt Büste aufgezeichnet hat.

Kapitel 8

Zusammenfassung und Ausblick

Die Ausgangssituation dieser Arbeit sind Datensätze von Tiefenkarten, die durch PMD-Kameras, welche auf einer geodätischen Sphäre angeordnet sind, aufgezeichnet wurden. Die Tiefenkarten eines solchen Datensatzes überschneiden sich und sind in diesen Überlappungsbereichen inkonsistent.

In dieser Arbeit wurden zwei Optimierungsverfahren beschrieben, die solche Tiefenkarten durch den Einsatz des MLS-Approximationsverfahrens verbessern und so die Inkonsistenzen innerhalb des Datensatzes reduzieren können. Das Verfahren, welches durch die Schnittpunkte lokaler Approximationen mit dem Sehstrahl eine Optimierung der Tiefenwerte erzielt, liefert in kurzer Zeit einen deutlich konsistenteren Satz von Tiefenkarten. Die Optimierung durch die „Minimierung des MLS-Fehlers“ ist zwar theoretisch genauer, aber in der Praxis konnten keine signifikanten Unterschiede in der Qualität der optimierten Daten ausgemacht werden. Außerdem ist das zweite Verfahren um ein Vielfaches langsamer.

Um die Glättung von Unstetigkeiten der aufgezeichneten Objekte durch die MLS-basierten Optimierungen zu vermeiden, wurde ein Segmentierungsverfahren eingesetzt. Dadurch konnten die Kanten der Objekte erhalten werden. Allerdings hat diese Segmentierung den Aufwand für die Optimierung stark erhöht.

Die Konsistenz der Flying Pixel konnte durch die Optimierungen erheblich verbessert werden. Allerdings kann es sein, dass das Objekt auf der optimierten Tiefenkarte leicht vergrößert ist, was sich durch die Herkunft der Flying Pixel erklärt.

Eine weitere Herausforderung ist die Beschleunigung der MLS-basierten Optimierung. In der Implementierung zu dieser Arbeit wurden dynamische Arrays

zur Speicherung der Punktwolken verwendet. Die Nutzung eines Baumes, wie dem Octree, als Datenstruktur würde das Einfügen, Löschen und Suchen in der Punktwolke beschleunigen. Eine weitere Möglichkeit, die Geschwindigkeit der Optimierung erheblich zu verbessern, könnte eine GPU-basierte Implementierung sein. Dann könnten alle Pixel aller Tiefenkarten eines Datensatzes parallel optimiert werden.

Literaturverzeichnis

[AA03a] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In Leif Kobbelt, Peter Schröder, and Hugues Hoppe, editors, *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 245–254, June 2003.

[AA03b] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *Proceedings of the Shape Modeling International 2003*, pages 272–279, Washington, DC, USA, 2003. IEEE Computer Society.

[AA04a] Anders Adamson and Marc Alexa. Approximating bounded, non-orientable surfaces from points. In *SMI*, pages 243–252, 2004.

[AA04b] Marc Alexa and Anders Adamson. On normals and projection operators for surfaces defined by point sets. In Marc Alexa, Markus Gross, Hanspeter Pfister, and Szymon Rusinkiewicz, editors, *Proceedings of Eurographics Symposium on Point-based Graphics*, pages 149–156. Eurographics, 2004.

[ABCO+01] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28. IEEE Computer Society, October 2001.

[ABCO+03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9:3–15, January 2003.

[AK04a] Nina Amenta and Yong J Kil. The domain of a point set surfaces. *Eurographics Symposium on Point-based Graphics*, 1(1):139–147, June 2004.

[AK04b] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 264–270, New York, NY, USA, 2004. ACM.

[Bre71] Richard P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *Comput. J.*, 14(4):422–425, 1971.

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 544–552, New York, NY, USA, 2005. ACM.

[Han02] Andreas Handl. *Multivariate Analysemethoden*. Statistik und ihre Anwendungen. Springer, Berlin [u.a.], 2002.

[HJS08] Benjamin Huhle, Philipp Jenke, and Wolfgang Straßer. On-the-fly scene acquisition with a handy multi-sensor system. *IJISTA*, 5(3/4):255–263, 2008.

[JS04] Florian Jarre and Josef Stoer. *Optimierung*. Springer, 2004.

[KK09] Maik Keller and Andreas Kolb. Real-time simulation of time-of-flight sensors. *Simulation Modelling Practice and Theory*, 17(5):967–978, 2009.

[Lev98] David Levin. The approximation power of moving least-squares. *Math. Comput.*, 67(224):1517–1531, 1998.

[Lev03] D. Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[Nea04] Andrew Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. Technical report, TU Darmstadt, 2004.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, August 2007.

[Rou84] Peter J. Rousseeuw. Least Median of Squares Regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[SK10] Alexander Sabov and Jörg Krüger. Identification and correction of flying pixels in range camera data. In *Proceedings of the 24th Spring Conference on Computer Graphics*, SCCG '08, pages 135–142, New York, NY, USA, 2010. ACM.