UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Texturierte 3-D-Mesh Generierung aus Stereobildsequenzen

Diplomarbeit

zur Erlangung des Grades eines/r

Diplom-Informatikers / Diplom-Informatikerin
im Studiengang Computervisualistik

vorgelegt von

Marco Mengelkoch

Betreuer:	Dipl.-Inf. Peter Decker, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Dipl.-Inf. Peter Decker, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Oktober 2008

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass 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.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Ein besonderer Dank geht an meinen Betreuer Dipl.-Inf. Peter Decker, der sich viel Zeit für regelmäßge Treffen genommen und mir bei technischen Problemen wertvolle Hinweise und Anregungen gegeben hat. Auch möchte ich mich bei Prof. Dr.-Ing. Dietrich Paulus bedanken, der bei mir das Interesse im Bereich Rechnersehen geweckt und mit mir zusammen dieses Thema ausgewählt hat.

Ebenso danke ich meinen Eltern, die mir dieses Studiums ermöglicht haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

3-D Modelle sind heute aus der Computerwelt nicht mehr wegzudenken. Sie dienen z. B. der Unterhaltung (im Kino oder in Videospielen), helfen bei Simulationen oder bei der Planung von Gebäuden. Der Vorteil von 3-D Modellen gegenüber einfachen Bildern ist die Möglichkeit, den Blickwinkel zum Modell zu verändern und somit eine bessere Vorstellung vom zu betrachtenden Objekt zu bekommen. Zur Erstellung dieser Modelle werden häufig komplizierte Softwarewerkzeuge wie z. B. Autodesk Maya oder Blender verwendet, die es ermöglichen real aussehende Objekte zu erstellen.

Für einige Anwendungsszenarien werden 3-D Oberflächenmodelle von real existierenden Objekten benötigt. Hierzu wird häufig für die Vermessung und Erstellung der Modelle sehr teure Hardware, wie z. B. ein 3-D Laser-Range-Scanner verwendet, um ein genaues Abbild der Szene oder des Objekts schnell und zuverlässig zu erstellen. Laser-Range-Scanner können aber keine Grauwert- bzw. Farbinformationen vom Objekt erfassen. Sollen zusätzlich zur topografischen 3-D Struktur farbige Texturen im Modell hinzugefügt werden, so muss das Objekt zusätzlich abfotografiert und das Modell mit den Bildern registriert werden.

Digitale Kameras sind klein, weit verbreitet und kosten nur einen Bruchteil von 3-D

Laser-Range-Scannern. Durch die Verwendung eines kalibrierten Stereokamerasystems ist es ebenfalls möglich korrekte Tiefeninformationen aus Kamerabildern zu erhalten. Diese Tiefeninformationen sind natürlich wesentlich ungenauer, als die eines Laser-Range-Scanners. Allerdings würde die Verwendung eines Stereokamerasystems eine kostengünstige, platzsparende und schnelle Alternative zur Gewinnung von 3-D Informationen mittels Laser-Range-Scannern darstellen.

1.2 Zielsetzung

Die hier vorgestellte Diplomarbeit beschäftigt sich mit der Rekonstruktion eines texturierten 3-D Meshs aus 2-D Stereobildsequenzen. Ziel der Arbeit ist, ein System zu entwickeln, welches aus einer Sequenz von Stereobildern aus einem kalibrierten Stereokamerasystem ein texturiertes Oberflächenmodell der Umgebung erstellt.

Ein mögliches Szenario ist die Rekonstruktion von einem oder mehreren Gebäuden mit einer Sequenz von Stereobildern. Die Bilder werden hierzu z. B. von einem Fahrzeug aufgenommen, welches sich parallel zum Gebäude bewegt, ähnlich wie es in Google Street View zu sehen ist. Es soll aber auch die Rekonstruktion aus Aufnahmen von unregelmäßigen Positionen mit unterschiedlichen Kamerawinkeln möglich sein.

Hierbei soll weder die genaue Aufnahmeposition der Stereobilder, noch die aufgenommene Umgebung bekannt sein. Die Position wird alleine anhand der Kamerabilder geschätzt. Von der aufgenommenen Szene soll dann die 3-D Oberflächenstruktur geschätzt werden. Aus den Stereobildern werden dichte Tiefenkarten erstellt. Überlappende Bereiche innerhalb der verschiedenen Tiefenkarten sollen mit geeigneten Methoden gefiltert werden, um aus den redundanten Informationen fehlerhafte Werte auszuschließen.

Die daraus entstandene 3-D Punktewolke soll zu einem Dreiecksmesh verbunden werden, um dieses mit aktueller 3-D Hardware effizient darstellen zu können. Die Anzahl der Dreiecke soll möglichst klein sein. Kleinere Dreiecke, welche die selben Flächen darstellen, sollen zu größeren zusammengefasst werden. Aus den aufgenommenen Stereobildern soll eine geeignete Textur für die Dreiecke ausgesucht werden, um die das Dreiecksmesh zu texturieren.

1.3 Inhalt der Kapitel

Kapitel 2 stellt einige aktuelle Methoden zur Rekonstruktion vor, auf deren Ideen und Prinzipien diese Diplomarbeit aufbaut. In Kapitel 3 wird detailliert auf die verwendeten Algorithmen eingegangen, die für den Rekonstruktionsansatz in dieser Diplomarbeit verwendet werden, während in Kapitel 4 gezeigt wird, dass der gewählte Lösungsansatz inhaltlich sowie in der Ausführung korrekt und mit anderen Ansätzen vergleichbar ist. In Kapitel 5 werden die Ergebnisse, die in dieser Diplomarbeit erzielt wurden zusammengefasst.

Kapitel 2

Rekonstruktion eines Oberflächenmodells aus Bildsequenzen

Die Idee der Rekonstruktion von 3-D Modellen aus Bildsequenzen ist nicht neu. Seit mehreren Jahren gibt es bereits eine Vielzahl von Verfahren, deren Eingangsdaten unterschiedliche Voraussetzungen erfüllen müssen und welche dann, bei geeigneten Eingangsdaten gute Ergebnisse liefern. Zur Rekonstruktion von Modellen aus Bildsequenzen mit bekannter Aufnahmeposition gibt es beispielsweise eine Evaluation von über 20 (online über 30) verschiedenen Ansätzen [SCD+06]. Jeder hat die Möglichkeit das Evaluationspaket herunterzuladen, mit dem eigenen Verfahren auszuwerten und die Ergebnisse online zu veröffentlichen. Das Evaluationspaket enthält eine Sequenz von Bildern, deren Aufnahmepositionen und die Ground-Truth Daten des Modells (siehe Bild 2.1).

2.1 Multiview Rekonstruktion

Multiview Rekonstruktion bedeutet, dass ein Objekt aus mehreren vorher bekannten Kamerapositionen aufgenommen wurde. Das Objekt ist in allen aufgenommenen Bildern sichtbar und hebt sich vom Hintergrund ab. Der Hintergrund ist einfarbig, und somit auch leicht von dem Objekt selber zu segmentieren. Somit hat man hier die Möglichkeit, die visuelle Hülle zu errechnen, wodurch die Ergebnisse stark verbessert werden.

Bild 2.1: Ground Truth des TempleRing Modells (Quelle: [SCD+06])

Die Zielsetzung dieser Diplomarbeit ist allerdings nicht die Rekonstruktion eines Modells, welches aus mehreren Positionen aufgenommen wurde, sondern die Rekonstruktion einer kompletten Szene, z. B. das Oberflächenmodell eines Raums oder eines Gebäudes.

Es werden jedoch einige Ideen von verschiedenen Verfahren zur Multiview Rekonstruktion aufgegriffen. Ebenso werden Methoden aus Verfahren zur Rekonstruktion mit Bildern aus einer unkalibrierten Handkamera benutzt.

2.1.1 Goesele und Curless

Michael Goesele und Brian Curless [GCS06] haben beispielsweise gezeigt, dass bereits eine leichte Modifikation eines Window-Match Algorithmus zu einem Multi-Baseline Matcher ziemlich genaue Ergebnisse in der Tiefenrekonstruktion erzielen kann. In diesem neuen Algorithmus wurde ein Strahl in einem Bild, welcher durch einen Pixel geht, in andere Bilder rückprojiziert. Entlang dieses Strahls wurde dann ein Window-Match Korrelations Algorithmus durchgeführt. Nur die Pixel, die in anderen Bildern gut korrelierten, wurden in eine Tiefenkarte (siehe Kapitel 3.2) eingefügt. Die Fusion der Tiefenkarten und die Rekonstruktion des Mesh wurden mit einer frei erhältlichen Implementierung von Curless und Levoy [CL96] durchgeführt, welche aber eigentlich zur Fusion von Laser-Range

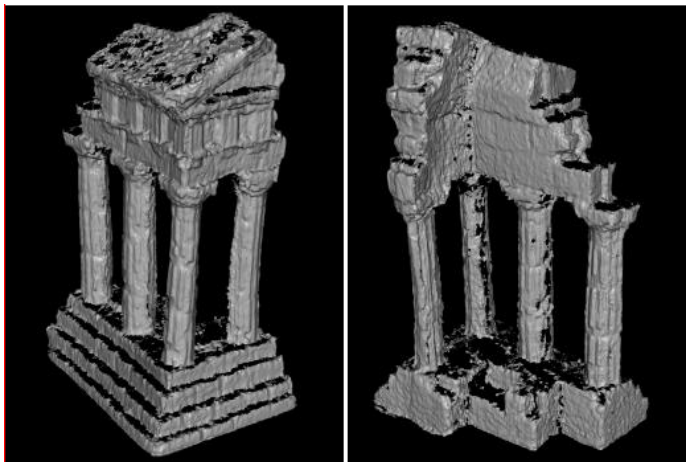


Bild 2.2: Rekonstruktion des TempleRing Modells nach Goesele (Quelle: [SCD+06])

Tiefenbildern entwickelt wurde. Durch die Auswahl von nur wenigen 3-D Punkten und die Wahl des Meshgenerierungsalgorithmus war die Rekonstruktion allerdings bei einer geringen Anzahl von Eingangsbildern löchrig (siehe Bild 2.2) und optisch nicht so anspruchsvoll wie die von anderen Verfahren.

2.1.2 Bradley, Boubekeur und Heidrich

Der Ansatz von Bradley, Boubekeur und Heidrich[BBH08] ist bereits komplizierter und besteht aus folgenden Schritten:

- **3-D Gewinnung:** Schätzung von Tiefenkarten und Eintragung in eine 3-D Punktewolke.
- **Downsampling:** Reduktion der Punktewolke durch Clustering.
- **Cleaning:** Zusätzliches Entfernen von Rauschen in der 3-D Punktewolke.
- **Meshing:** Erzeugung eines Dreiecksgitternetzes aus der Punktewolke.

Auch wenn der Ansatz von Bradley, Boubekeur und Heidrich laut eigenen Angaben auf dem von Goesele und Curless aufbaut, unterscheidet sich der Ablauf der Rekonstruktion

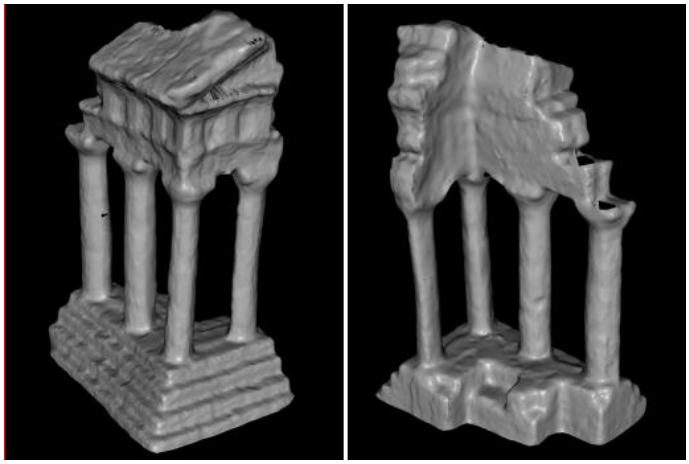


Bild 2.3: Rekonstruktion des TempleRing Modells nach Bradley, Boubekeur und Heidrich (Quelle: [SCD+06])

deutlich. Die Tiefenkarten werden nicht durch den Vergleich von mehreren Bildern gemacht sondern werden durch Paare von immer zwei Bildern erstellt. Wie bei Goesele und Curless werden die Tiefenkarten durch einen Window-Match Algorithmus gewonnen. Allerdings werden dadurch, dass immer nur zwei Bilder verglichen werden, mehr Punkte zugelassen, wodurch die Rekonstruktion auch bei wenigen Eingangsbildern kaum Löcher aufweist (siehe Bild 2.3). Die Rekonstruktion des Mesh wurde für 3-D Punktewolken aus dichten Tiefenkarten optimiert. Der Algorithmus wird in Kapitel 3.6.1 erläutert.

2.2 Rekonstruktion aus Bildern einer Handkamera

Die Rekonstruktion von Bildern aus einer Handkamera benötigt eine Sequenz von Bildern, deren Aufnahmeposition und Kalibrierdaten nicht bekannt sein müssen. Die Rekonstruktion ist somit nur bis auf einen Skalierungsfaktor genau.

2.2.1 David Nistér

Nistér [NS01] hat ein System entwickelt, welches aus einer Sequenz von unkalibrierten Bildern, beispielsweise aus einer handelsüblichen Digitalkamera, texturierte 3-D Modelle einer größeren Szene erstellen kann.

Da hier weder Kameraparameter, noch deren Positionen bekannt sind, müssen die aufgenommenen Bilder in einem ersten Schritt kalibriert werden. Hierzu wird ein Autokalibrierungsverfahren verwendet, welches kein Kalibriermuster benötigt, sondern die Kalibrierung anhand selektierter Bildmerkmale durchführt. Sind die Positionen und Parameter errechnet, so können Bildpaare ausgesucht und rektifiziert werden (siehe Kapitel 3.2.2). Aus den rektifizierten Bildpaaren wird dann mittels Graph-Cut Algorithmus jeweils ein Tiefenbild erstellt. Die einzelnen Tiefenkarten werden mit einem Stability-Based-Fusion Algorithmus (siehe Kapitel 3.5.2) kombiniert und die anfangs selektierten Bildmerkmale zu einem Mesh verbunden und texturiert.

Die entstandenen Ergebnisse sind beeindruckend (siehe Bild 2.4 und 2.5), jedoch sind die Methoden, um zu diesem Ergebnis zu gelangen kompliziert und würden den Rahmen dieser Diplomarbeit sprengen.

Bild 2.4: Rekonstruktion von Nistér. Modell wurde aus mehreren Ansichten erstellt. (Abbildung aus [NS01])

Bild 2.5: Rekonstruktion von Nistér. Andere Ansicht des Modells. Mesh und Textur sind identisch mit Abbildung 2.4. (Abbildung aus [NS01])

Bild 2.6: Eines der Eingangsbilder aus Pollefeys und Kochs Rekonstruktion. (Abbildung aus [PGV+04])

2.2.2 Pollefeys und Koch

Ebenso wie David Nistér beschäftigt sich das Verfahren von Pollefeys und Koch (u.a.) [PGV+04] mit der Rekonstruktion von 3-D Szenen aus Bildern einer unkalibrierten Handkamera. Die Schritte zur Autokalibrierung und der Positionsbestimmung ähneln denen von Nistér. Der Unterschied zur Arbeit von Nistér ist der, dass stark überschneidende Bilder vorausgesetzt werden, wodurch sehr detaillierte Tiefeninformationen extrahiert werden, während bei Nistér eine größere Szene aus sich weniger überschneidenden Bildern rekonstruiert wird und die Tiefeninformationen eher ungenau wirken.

Ermöglicht wird die genaue Erfassung von Tiefeninformationen dadurch, dass ähnlich wie bei Goesele und Curless ein Multi-Baseline Algorithmus eingesetzt wird.

Die 3-D Ansicht wird in den meisten Beispielen anhand eines einzigen Bildes erstellt, welches kaum verdeckte Flächen enthält, aus einer weit entfernten Position aufgenommen wurde, und somit den Großteil der aufgenommenen Bilder überdeckt. Eines dieser Ursprungsbilder ist auf Bild 2.6 zu sehen. Die erstellten Modelle haben dadurch eine sehr gute optische Qualität und ebenfalls eine hohe Genauigkeit, wie man in Bild 2.7 und 2.8 sehen kann. Auch wenn die optische Qualität dieses Verfahrens auf den ersten Blick das Verfahren von Nistér übertrifft, so scheint es eher weniger dafür geeignet zu sein, größere Szenen aus sich weniger überschneidenden Bildern zu fusionieren.

Bild 2.7: Pollefeys und Kochs Rekonstruktion. (Abbildung aus [PGV+04])

Bild 2.8: Andere Ansicht von Pollefeys und Kochs Rekonstruktion. Nur ein Bild wurde für die Texturierung benutzt. (Quelle: [PGV+04])

2.3 Urban Reconstruction

Das Urban Reconstruction Projekt wurde von der DARPA (Defense Advanced Research Projects Agency) gefördert und in Zusammenarbeit der University of Kentucky mit der University of North Carolina, Chapel Hill entwickelt. In den wissenschaftlichen Veröffentlichungen [AFM+06] [MFA+07] [PNF+08] werden unter den 18 Autoren auch Marc Pollefeys und David Nistér genannt, wodurch dieses System den beiden oben genannten Verfahren zur Rekonstruktion aus Bildern einer Handkamera in vielen Punkten stark ähnelt. Ziel des Urban Reconstruction Projekts war, ein Verfahren zur 3-D Rekonstruktion zu entwickeln, welches in (nahezu) Echtzeit aus mehreren, kalibrierten Kameras, die auf das Dach eines Fahrzeugs montiert wurden und einer Vielzahl von aufgenommenen Bildern, ein 3-D Modell der Umgebung erstellt (siehe Bild 2.9).

Die Position des Fahrzeugs und somit auch der Kameras wird mittels GPS-Satellitendaten, KLT-Tracker [KL81] [KL81] [ST94] und Kalman Filter [WB01] geschätzt. Die 3-D Informationen werden durch einen Plane-Sweep Algorithmus auf der Grafikkarte berechnet und mit einem Confidential-Based-Fusion (siehe Kapitel 3.5.2) Algorithmus bzw. alternativ mit einem Stability-Based Algorithmus fusioniert. Das 3-D Mesh wird aus den Referenzansichten der fusionierten Tiefenkarten erstellt und bereits in den Tiefenkarten selber reduziert (siehe Kapitel 3.6.7). Aus den Texturen der einzelnen Dreiecke werden jeweils Median-Texturen errechnet (siehe Kapitel 3.6.8). Die ähnlichste Textur zu der Median Textur wird zur Texturierung des Dreiecks verwendet.

Das Urban Reconstruction Projekt kommt der Zielsetzung dieser Diplomarbeit am nächsten. Allerdings müssen die verwendeten Methoden so stark vereinfacht werden, dass ein Rekonstruktionsverfahren in der vorhandenen Zeit auch implementiert werden kann. Ein Anspruch auf Echtzeitfähigkeit wird in dieser Diplomarbeit nicht verfolgt.

Bild 2.9: Rekonstruktionen aus dem Urban Reconstruction Projekt. (Abbildung aus [AFM+06]) und [MFA+07])

Kapitel 3

Texturierte 3-D-Mesh Generierung aus Stereobildsequenzen

3.1 Der Lösungsansatz

Das hier vorgestellte System besteht aus 5 voneinander trennbaren Schritten:

1. Erstellen von dichten Tiefenkarten aus den Stereobildern.

   Die Bilder werden anhand der Kalibrierungsinformationen entzerrt und rektifiziert. Aus jedem Stereobild wird ein Tiefenbild mit dem Dynamic Programming Ansatz von Birchfield [BT98] errechnet (siehe Kapitel 3.2).

2. Schätzen der Aufnahmepositionen mittels ICP Algorithmus.

   Die Tiefenbilder werden mittels ICP Algorithmus registriert. Die Korrespondenzen werden allerdings mittels SIFT Bildmerkmalen berechnet, um negative Eigenschaften der Tiefenbilder (bei der Verwendung des ICP Algorithmus, siehe Kapitel 3.4.3) zu kompensieren.

3. Gefilterte 3-D Rückrechnung der Tiefenkarten (Fusion der Tiefenkarten).

   Die Tiefenkarten werden mit einem Bayes Filter fusioniert um fehlerhafte Werte herauszufiltern (siehe Kapitel 3.5.3).

4. Verbinden der 3-D Punkte zu einem Dreiecksmesh.

   Aus jeder Ansicht wird wird ein Mesh erstellt (siehe Kapitel 3.6). Doppelt vorkommende Dreiecke werden gelöscht. Segmentierung verhindert das Verbinden von Dreiecke mit zu großen Tiefenunterschieden.

5. Ausdünnen des Meshs und Texturierung.

   Die Dreiecksmenge wird mit einem Memoryless Simplification Algorithmus reduziert und anschließend texturiert (siehe Kapitel 3.6.6).

Falls vorhanden, wurden fertige Implementierungen der einzelnen Schritte übernommen und modifiziert. So konnte beispielsweise Schritt 1 zum Teil aus vorhandenen Projekten der AGAS übernommen werden. Bei Schritt 2 konnte der Algorithmus zum Extrahieren und Zuordnen ähnlicher SIFT Merkmale aus [Dec07] verwendet werden, wodurch lediglich der ICP Algorithmus selber implementiert werden musste. Schritt 3 dagegen wurde komplett in Eigenarbeit erstellt, aufbauend auf der Beschreibung des Confidential-Based Algorithmus in Kapitel 3.5.2. Das Verbinden der Punkte zu einem Dreiecksmesh (Schritt 4) wurde ebenfalls in Eigenarbeit implementiert, wobei die Delaunay Triangulierung mit Hilfe der *Triangle* Bibliothek durchgeführt wurde, um aus einer 2-D Punktemenge ein 2-D Mesh für jede Ansicht zu erstellen. Das Ausdünnen des Meshs in Schritt 5 wird mit Hilfe des Memoryless Simplification Algorithmus aus der *GNU Triangulated Surface Library* durchgeführt, während die Texturierung basierend auf der Beschreibung von [PGV+04] und [MFA+07] implementiert wurde.

3.2 Tiefenkarten

Tiefenkarten beinhalten die Tiefenwerte von Pixeln aus einem Bild. Sie können entweder aus 3-D Punkten errechnet werden, deren Tiefenwerte in einer Ansicht projiziert werden oder aus Disparitätskarten von Stereobildern geschätzt werden, bei denen die Kalibrierungsinformationen der Kameras bekannt sind.

3.2.1 Disparitätsschätzung

Die Disparität (oder horizontale Paralaxe) beschreibt die absolute Differenz der x-Koordinaten zweier korrespondierender Pixel aus dem Bildpaar eines perfekt parallelen Stereosystems.

$$d = |p_x^p - q_x^p| \tag{3.1}$$

Aus der Disparität d zweier Pixel, dem Kamerafokus F und dem Abstand der Kameras b, lässt sich deren Tiefe (z-Koordinate) errechnen.

$$p_z^w = \frac{F}{d} b \tag{3.2}$$

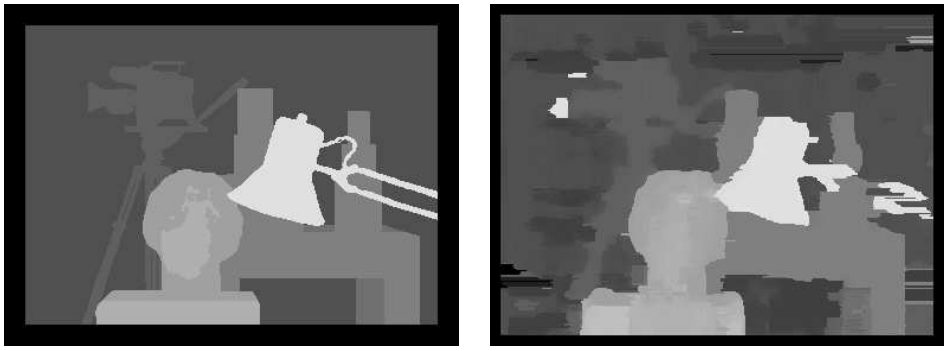
Für die Schätzung der Disparität wurde in dieser Diplomarbeit die bereits in *OpenCV* enthaltene Implementierung des Dynamic-Programming Birchfield Algorithmus [BT98] verwendet, welcher einen guten Kompromiss zwischen Genauigkeit und Performanz bietet. Einen Überblick über andere Algorithmen, die schneller oder genauer sind, findet man in [SS02]. Einige Beispiele sind in Bild 3.1 zu sehen.

Während z. B. der Realtime Correlation Algorithmus [HIG02] für eine 384x288 große Tiefenkarte in etwa eine zehntel Sekunde benötigt (siehe [SS02]), braucht ein optimierter Graph Cut Algorithmus [BVZ01] fast eine Minute. Der Birchfield Algorithmus [BT98] benötigt etwa eine Sekunde.

3.2.2 Rektifikation

Die Annahme eines perfekt parallelen Stereosystems vereinfacht die Erstellung von Disparitätskarten. Ein perfekt paralleles Stereosystem zeichnet sich dadurch aus, dass die epipolaren Linien auf gleicher Höhe parallel zur x-Achse im Bild verlaufen. Dies bedeutet, dass korrespondierende Punkte stets in der selben Zeile des anderen Bildes zu finden sind. Diese Einschränkung des Suchraums erhöht die Effizienz von Algorithmen zur Tiefenkartenerstellung erheblich.

Da Stereokamerasysteme in den seltensten Fällen perfekt parallel sind, müssen die aufgenommenen Bilder verzerrt werden, um ein perfektes Stereosystem zu simulieren. Diese



(a) Ground Truth (b) Realtime Correlation

(c) Birchfield (d) Graph Cut

Bild 3.1: Disparitätskarten, welche mit unterschiedlichen Algorithmen erstellt wurden. (Abbildungen aus [SS02])

Simulation (wie in Bild 3.2) nennt man Rektifikation. Für die Rektifikation von Bildern gibt es verschiedene Ansätze, die z. B. die Bilder mit einer Homographie-Matrix verzerren [FTV97] oder erst durch Polarkoordinaten parametrisieren [PKG99], um eine genauere Simulation eines perfekten Stereosystems zu erhalten.

In der Implementierung dieser Diplomarbeit wurde die einfachere Variante der Rektifikation durch eine Homographie-Matrix verwendet, da diese bereits in *OpenCV* enthalten ist und ausreichend gute Ergebnisse liefert. Ein Beispiel der Rektifikation sieht man auf Bild 3.3.

Bevor die Bilder rektifiziert werden müssen sie erst nach [Zha00] entzerrt werden, um die Linsenkrümmung auszugleichen.

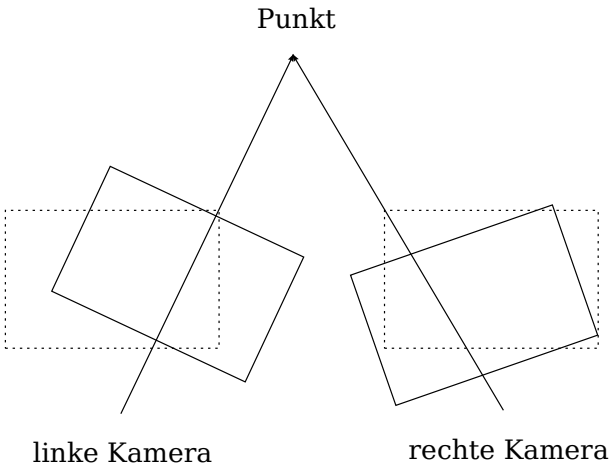


Bild 3.2: Rektifikation eines Stereobildes. Das gestrichelte Rechteck stellt die gewünschte Bildebene nach der Rektifikation dar. (Abbildungen aus [Dec06])

3.3 Merkmalsextraktion aus Bildern

Die meisten Algorithmen zum Schätzen der Kameraposition aus Bilddaten erfordern Korrespondenzen zwischen den Bildern. Die Auswahl von Punkten, auch Bildmerkmale oder Features genannt, die in beiden Bildern vorkommen und den selben Pixel darstellen, ist essentiell für die Positionsschätzung. Sowohl die Position in den Bildern, als auch deren richtige automatische Zuordnung ist ausschlaggebend für die Genauigkeit der geschätzten Postion. Im Idealfall sind die Bildmerkmale gleichmäßig im Bild verteilt und im nächsten Bild mit minimaler Abweichung korrekt wiederzufinden. Für die richtige Zuordnung zweier Merkmale, sollte jedes Merkmal einen eindeutigen Deskriptor haben, welcher dieses identifiziert und der in jedem Bild unter verschiedenen Aufnahmeparametern wie Blickwinkelposition oder Beleuchtung gleich ist.

Vor der Extraktion von Bildmerkmalen muss die Linsenkrümmung nach [Zha00] ausgeglichen werden. Dieser Schritt sollte direkt nach der Aufnahme der Bilder durchgeführt werden, da auch die Erstellung der Disparitätskarten und die Extraktion von Texturen ein entzerrtes Bild voraussetzen.

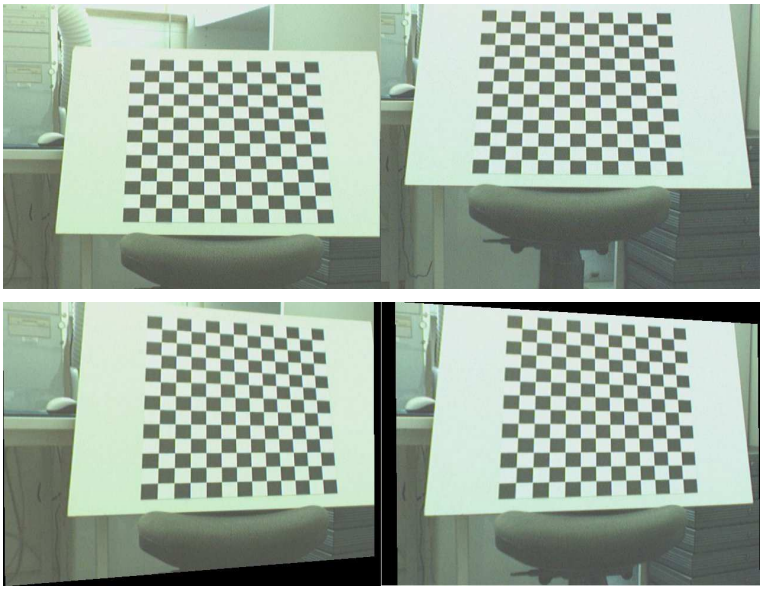


Bild 3.3: Oben: Unrektifiziertes Bild, Korrespondenzen sind schwer zu finden. Unten: Rektifiziertes Bild, Korrespondenzen sind in der selben Zeile des anderen Bildes (Abbildungen aus [Dec06]).

3.3.1 KLT Tracker

Der Kanade-Lucas-Tomasi Tracker [KL81] ist eine Methode um den optischen Flow zwischen zwei Bildern zu schätzen. Das heißt, es werden bestimmte Bildmerkmale selektiert und im nächsten Bild deren Position geschätzt. Als Merkmal werden Ecken detektiert, da diese in alle Richtungen starke Veränderungen aufweisen und deren Position in nachfolgenden Bildern gut wiedergefunden werden kann (good features to track [ST94]). Um den Verschiebungsvektor der Bildmerkmale schnell und einfach zu berechnen, wird der Algorithmus auf verschiedenen Ebenen einer Bildpyramide berechnet [Bou02]. Das Problem, beim KLT Tracker ist zum einen, dass die Merkmale weder bei Skalierung, noch bei Rotation robust wiederzufinden sind. Zum anderen werden Bildmerkmale immer an Ecken detektiert, was auch dazu führt, dass in einem 3-D Modell die Merkmale immer an Stellen detektiert werden, wo ein großer Tiefenunterschied zu erwarten ist. Das führt dazu, dass die gefundenen Bildmerkmale in einer Bildsequenz schon bei kleinen Verschiebungen in

Richtung der x oder y Achse sehr unterschiedliche Tiefenwerte haben.

Nachteilig ist ebenfalls, dass der KLT Algorithmus nur in einem bestimmten Fenster des anderen Bildes sucht. Die Größe des Fensters ist parametrisierbar und wird durch die Anzahl der Ebenen in der Bildpyramide vergrößert. Ein zu großes Fenster führt jedoch zu einer weniger genauen Position des Merkmals, während bei einem zu kleinen Fenster oft falsche Korrespondenzen gefunden werden.

3.3.2 SIFT Merkmale (Scale-invariant feature transform)

Um die Nachteile der KLT Bildmerkmalen zu beheben wurden Ende der 90er SIFT Bildmerkmale [Low04] entwickelt. Ebenso wie KLT Merkmale werden SIFT Merkmale mit Subpixelgenauigkeit detektiert, allerdings sind sie zusätzlich invariant gegen Rotation und Skalierung. Zudem sind sie auch recht robust gegen Bildrauschen, Beleuchtungsänderungen und bei Veränderung der Kameraposition.

Erreicht wird dies indem das Graubild in mehreren Skalen und mit mehreren Gaußfiltern (mit verschiedenen Kernelgrößen) geglättet wird. Die Merkmale befinden sich in lokalen Minima bzw. Maxima der Unterschiede zwischen den geglätteten Bildern (Difference of Gaussians - DoG). Im späteren Verlauf des Algorithmus werden einige Merkmale, die nicht die oben genannten Eigenschaften haben verworfen (siehe Bild 3.4). Durch den Aufbau des Algorithmus werden SIFT Merkmale nicht an Ecken detektiert und entstehen auch in Bereichen mit weniger Texturierung.

Jedes Merkmal wird (zusätzlich zur Bildposition und Skalierung) durch einen Vektor mit 128 Zeilen beschrieben, welcher aus den Histogrammen der Richtungen von den errechneten Veränderungen im DoG besteht. Die Zuordnung (bzw. das Matching) von Merkmalen wird mit der euklidischen (oder alternativ der absoluten) Distanz der Deskriptorvektoren errechnet, wobei der mittels Nearest-Neighbour gefundene Vektor derjenige, mit dem korrespondierendem Merkmal ist. Es wird also nicht mehr, wie beim KLT-Tracker von einer bisherigen Bildposition ausgegangen, wo ein Bildmerkmal gesucht wird, sondern die Merkmale werden aus beiden Bildern unabhängig voneinander extrahiert, anhand der Deskriptoren verglichen und dann einander zugeordnet.

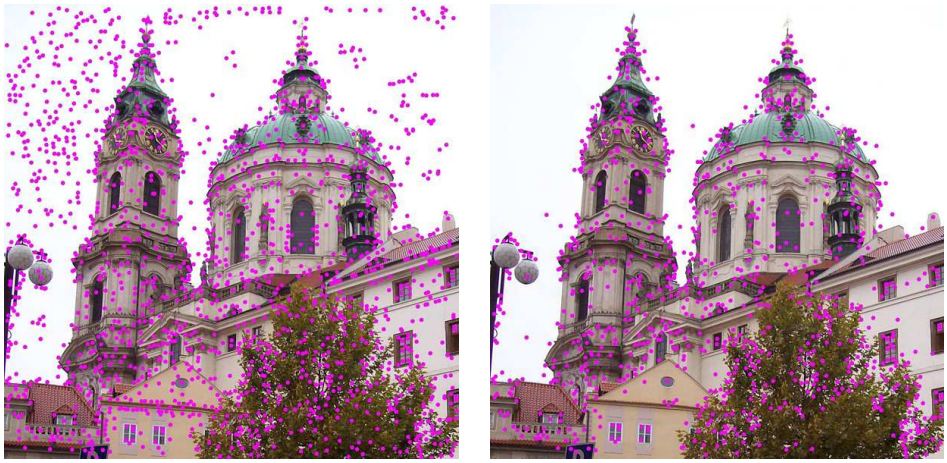


Bild 3.4: Links: Lokale Extrema (DoG), welche aus dem Bild selektiert wurden. Rechts: Lokale Extrema (DoG), welche nach dem SIFT Algorithmus als zuverlässig (hoher Kontrast, skalierungs- und rotationsinvariant) eingestuft wurden. (Abbildungen aus [SIF08])

Um die Stabilität zu erhöhen werden die Zuordnungen verworfen, deren Quotient zum zweitbesten Kandidaten (Distance-Ratio) größer als $0,8$ ist. Dies würde darauf hindeuten, dass beide Merkmale sich zu sehr ähneln um stabil zugeordnet zu werden. Das Berechnen der euklidischen Distanz zu allen möglichen Merkmalen benötigt enorm viel Rechenzeit und steigt mit der Anzahl der zu vergleichenden Merkmale. Somit dauert der Vergleich möglicher Merkmale öfters länger als das Selektieren aus den Bildern.

Zur Beschleunigung kann auch ein auf k-d-Bäume aufbauender Best-Bin-First (BBF) Algorithmus [BL97] verwendet werden, wodurch die Berechnung der Korrespondenzen auf einen Bruchteil reduziert wird. In dieser Diplomarbeit wurde die Methode ohne BBF aus [Dec07] verwendet.

Zusammengefasst lässt sich sagen, dass SIFT Merkmale eine robuste Alternative zu KLT Merkmalen sind.

3.4 Positionsbestimmung

Für eine funktionierende Rekonstruktion aus Bildsequenzen muss die Position der einzelnen Kameras von denen die Aufnahmen gemacht wurden, bekannt sein. Jedoch ist dies nur selten der Fall, und wenn doch, so fehlen häufig genaue Angaben über den Blickwinkel, der die Rotation in *x*,*y* und *z* Achse enthält. In den vergangenen Jahren wurde dieses Problem allerdings sehr genau analysiert, wodurch es mehrere unterschiedliche Vorgehensweisen zur Positionsbestimmung gibt. Die Wahl des Algorithmus richtet sich danach, welche Eingangsdaten man zur Verfügung hat, und nach welchem mathematischen Modell man das Problem der Positionsbestimmung zu lösen versucht.

3.4.1 Der 8-Punkte Algorithmus

Der 8-Punkte Algorithmus [Har97] erschließt sich aus der Sicht der Epipolargeometrie. Es werden mindestens 8 Punkte aus einem Bild korrespondierenden Punkten eines anderen Bildes zugeordnet. Punkte aus einem Bild liegen auf der zugehörigen epipolaren Linie im anderen Bild. Diese epipolaren Linien können durch die Fundamental Matrix F und den zugehörigen Punkten ausgedrückt werden. Sei p^p ein Punkt in einem und q^p ein Punkt im anderen Bild, so soll eine Matrix F gefunden werden für die gilt

$$p^{p\mathrm{T}} F q^p = 0 \tag{3.3}$$

Aus der F Matrix kann die Rotationsmatrix R und ein Translationsvektor t extrahiert werden, welcher allerdings nur bis auf einen Skalierungsfaktor genau ist. Somit ist die Richtung des Translationsvektors zwar korrekt, der Skalierungsfaktor, um eine metrisch korrekte Positionsschätzung zu erhalten, muss jedoch aus geschätzten 3-D Punkten errechnet werden.

RANSAC

Beim 8-Punkte Algorithmus werden alle Korrespondenzen im Gleichungssystem berücksichtigt. Das heißt, dass auch falsch zugewiesene Korrespondenzen das Ergebnis des Gleichungssystems beeinflussen. Werden die Korrespondenzen automatisch zugewiesen, ist

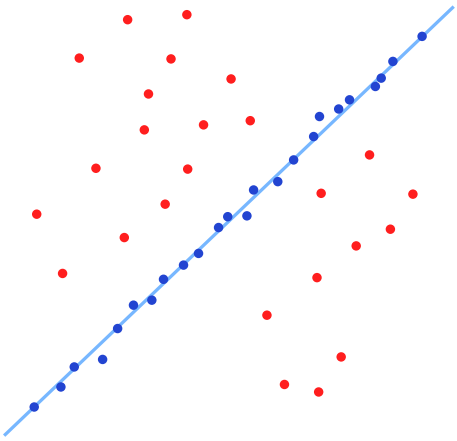


Bild 3.5: Beispiel des RANSAC Algorithmus zum Finden von Punkten, die eine Linie bilden. (Aus [Ran08])

ein RANSAC Algorithmus [FB81] in Verbindung mit dem 8-Punkte Algorithmus eine gute Möglichkeit, diese falschen Korrespondenzen zu finden und aus dem Gleichungssystem zu entfernen. Der RANSAC Algorithmus (**Ran**dom **Sa**mple **C**onsensus) sucht mit zufälligen Schätzungen iterativ nach Inliern, einer Teilmenge der automatisch selektierten Merkmale, welche das Gleichungssystem mit minimalem Fehler löst. Bild 3.5 zeigt die Verwendung eines RANSAC Algorithmus zum Finden einer Linie.

3.4.2 Der ICP Algorithmus

Der ICP-Algorithmus arbeitet mit 3-D Punktewolken. Im Gegensatz zum 8-Punkte Algorithmus löst der Iterative Closest Point Algorithmus [Nüc06] das Gleichungssystem nicht nur nach Rotation R und Translation t auf, sondern sucht auch gleichzeitig nach Punktkorrespondenzen. Der Translationsvektor ist zusätzlich auch metrisch korrekt und muss nicht mehr skaliert werden.

Gesucht werden die Korrespondenzen mit dem geringsten quadratischen Abstand des Modells M zur Messung (Datenmenge) D, was durch die Minimierung folgender Fehlerfunk-

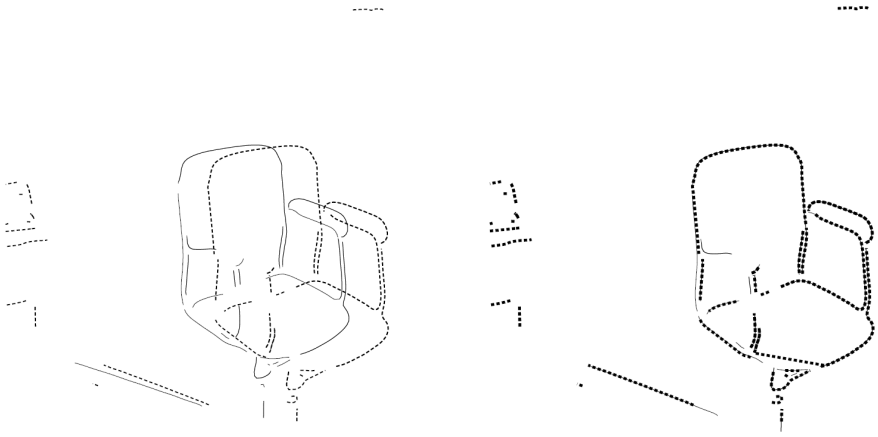


Bild 3.6: Beispiel ICP Algorithmus. Links sind zwei unabhängige Punktewolken vor dem ICP Algorithmus, welche das selbe Objekt abbilden. Rechts sind die Punktewolken nach dem ICP Algorithmus. Die Punkte innerhalb der Puktewolken haben einen minimalen Abstand zueinander. (Abbildung aus [Zha92]).

tion errechnet wird:

$$\mathrm{E}(R,t) = \sum_{i=1}^{N} \left\| \mathsf{m}_\mathsf{i} - (R\mathsf{d}_\mathsf{i} + t) \right\|^2 \tag{3.4}$$

Für alle Punkte $\mathsf{m} \in M$ existiert ein Punkt $\mathsf{d} \in D$ wodurch die Anzahl der Punkte N in beiden Mengen gleich groß ist.

Sind die Verhältnisse zwischen Modell- und Datenmenge ungleich, wie in Bild 3.6, so muss ein trimmed ICP (trICP) Algorithmus verwendet werden, der nur einen Teil der gemessenen Punkte D dem Modell M zuweist.

Die Punktkorrespondenzen müssen für die Berechnung durch den ICP Algorithmus vorher nicht bekannt sein. Die Suche nach dem nächsten Nachbarn mit dem geringsten quadratischen Abstand kann z. B. über einen k-d-Baum realisiert werden. [NLH07]

Um die Rotation R zu errechnen werden die Punktemengen M und D in deren Zentren c_m und c_d verschoben:

$$\mathsf{c_m} = \frac{1}{N}\sum_{i=1}^{N} \mathsf{m_i}, \qquad \mathsf{c_d} = \frac{1}{N}\sum_{i=1}^{N} \mathsf{d_i} \tag{3.5}$$
$$M' = \{\mathsf{m'_i} = \mathsf{m_i} - \mathsf{c_m}\}_{1,\ldots,N}, \qquad D' = \{\mathsf{d'_i} = \mathsf{d_i} - \mathsf{c_d}\}_{1,\ldots,N}$$

Somit kann die Rotation durch die Minimierung von

$$\mathrm{E}(R,t) = \sum_{i=1}^{N} \|\mathsf{m'_i} - R\mathsf{d'_i}\|^2 \tag{3.6}$$

errechnet werden. Die Translation t ergibt sich aus $t = \mathsf{c_m} - R\mathsf{c_d}$. Die Herleitung der Lösung kann in [Nüc06] nachgelesen werden. Die Minimierung der Gleichung 3.6 löst R mit der Faktorisierung $R = VU^{\mathrm{T}}$, wobei V und U aus der Singulärwertzerlegung von $H = U\Sigma V^{\mathrm{T}}$ stammen. H ist die Korrelation

$$H = \sum_{i=1}^{N} \mathsf{d'_i}\mathsf{m'_i}^{\mathrm{T}} = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}, \tag{3.7}$$

mit $S_{xx} = \sum_{i=1}^{N} \mathsf{m'_{ix}}\mathsf{d'_{ix}}$, $S_{xy} = \sum_{i=1}^{N} \mathsf{m'_{ix}}\mathsf{d'_{iy}}$, ...

Die Minimierung der quadratischen Fehlerfunktion wird nun iterativ in jedem Schritt des ICP Algorithmus durchgeführt. Nach jedem neu approximierten R und t werden die Korrespondenzen mittels k-d-Baum neu errechnet, bis entweder eine bestimmte Anzahl an Iterationen erreicht ist oder der quadratische Fehler einen vorher bestimmten Schwellwert unterschritten hat.

3.4.3 ICP in Tiefenbildern

Der ICP Algorithmus lässt sich theoretisch auch auf Tiefenbilder von Stereokamerasystemen anwenden. Hierfür wird ein vorheriges Tiefenbild in 3-D Koordinaten umgewandelt. Die errechneten Punkte ergeben das Modell M. Der neue Datensatz D wird aus einem neuen Tiefenbild errechnet, deren Punkte mit Hilfe eines k-d-Baums mit dem alten Datensatz verknüpft werden.

Hierbei erzeugen jedoch folgende Faktoren Probleme:

- Ist keine initiale Schätzung der neuen Position bekannt, so müssen sehr viele Iterationen durchgeführt werden.

- Die Iterationen kosten aufgrund der Bildgröße von Tiefenbildern sehr viel Rechenzeit. Bei einer Bildgröße von 640×480 müssen bereits bis zu 307200 Korrespondenzen pro Iteration berechnet werden.

- Dadurch, dass die meisten Algorithmen von Tiefenbildern nur eine bestimmte Anzahl von Tiefenwerten erzeugen, entstehen wie in Bild 3.7 Ebenen von Punkten parallel zur Bildebene.

- Durch sich überlagernde Ebenen entstehen lokale Minima, welche die Minimierung des quadratischen Fehlers der Rotation verhindern.

Versuche haben gezeigt, dass durch die Struktur der Tiefenbilder aus Stereokameras der ICP Algorithmus auch bei korrekter initialer Positionsschätzung fast immer zu einem falschen lokalen Minima führt und somit eine genaue Positionsbestimmung mittels ICP nicht möglich ist. Selbst eine Reduktion der Datenmenge *D* auf deren zweite Ableitung (siehe [Nüc06, S.86ff.]), also durch die Selektion von Punkten die an großen Unterschieden im Tiefenbild grenzen, löst dieses Problem nicht. Es reduziert lediglich die Rechenzeit bis zum Erreichen des falschen Minimas.

Um die Korrespondenzen zuverlässig schätzen zu können, müssten ähnliche Punkte idealerweise nicht allein anhand der Tiefenwerten, sondern auch durch deren Farb- oder Grauwerte, bzw. anhand der Nachbarschaftswerte im Kamerabild gesucht werden.

3.4.4 ICP mit Bildmerkmalen

Um bessere Korrespondenzen aus Tiefenbildern für den ICP Algorithmus zu erhalten, sollten auch Grauwerte im Kamerabild, bzw. deren Nachbarschaftswerte berücksichtigt werden. Im Idealfall werden Punkte gewählt, deren Korrespondenz im nächsten Bild möglichst eindeutig ist. Dies lässt sich sehr leicht mit Bildmerkmalen realisieren (siehe Kapitel 3.3), welche vor der Positionsbestimmung in den Kamerabildern gesucht werden müssen.

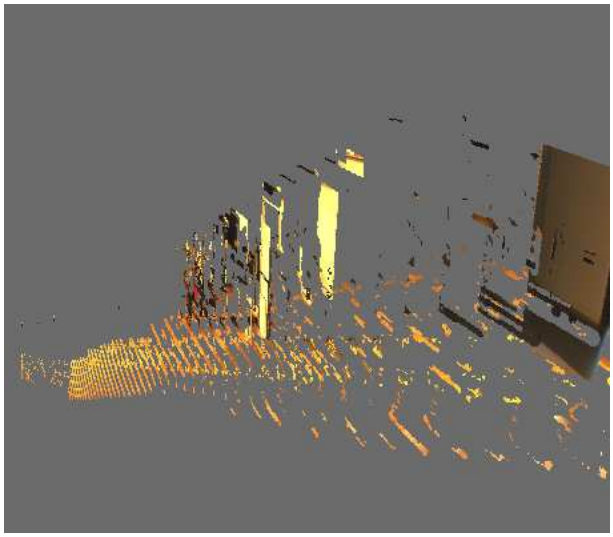


Bild 3.7: Texturiertes Tiefenbild von der Seite betrachtet. Sichtbar sind die Ebenen von Punkten, welche parallel zur Bildebene sind, sowie auch die Blickwinkelpyramide.

Deren 3-D-Wert wird (als $m \in M$ für das Referenzbild und $d \in D$ für das neu aufgenommene Bild) im ICP Algorithmus verwendet. Es werden also für den ICP Algorithmus in diesem Fall nur 3-D Punkte aus dem umgerechneten Tiefenbild genommen, die als Merkmal gefunden und deren Korrespondenz durch Deskriptoren bereits bestimmt wurde. Dies erspart die Korrespondenzsuche durch einen k-d-Baum, aber verhindert nicht die Zuordnung falscher Korrespondenzen und auch nicht, dass die Tiefenbilder stark verrauscht sind und einen falschen Tiefenwert am Bildmerkmal erzeugen können.

Um diese Punkte als Outlier zu identifizieren und aus dem Gleichungssystem zu entfernen, lässt sich eine Variante des ICP Algorithmus verwenden [RL01], welche iterativ Korrespondenzen mit großem räumlichen (quadratischen) Abstand auflöst. Als Schwellwert, ob ein Punkt Outlier ist, lässt sich z. B. die Standardabweichung nehmen. Ist der Fehler zwischen m_i und d_i um einen bestimmten Faktor größer als die Standardabweichung, so wird diese Korrespondenz bei der nächsten Berechnung von Rotation und Translation nicht berücksichtigt. Es sollte ebenfalls eine maximale Anzahl an Iterationen und eine minimale Anzahl an Korrespondenzen definiert werden. Die Anzahl der Inlier Korrespondenzen

sollte mindestens ein Drittel der Ursprungskorrespondenzen betragen.

Somit bleibt nur noch die Teilmenge der Merkmale übrig deren Standardabweichung minimal ist und mit denen sich das Gleichungssystem mit minimalen Fehler lösen lässt.

Wichtig ist hierbei, dass Outlier nur für den jeweils nächsten Iterationsschritt und der Berechnung der Standardabweichung aus dem Gleichungssystem entfernt werden. Durch die Veränderung von Rotation und Translation in jedem Iterationsschritt verändert sich auch wieder der Fehler von jeder Korrespondenz. Somit kann eine Korrespondenz, die anfangs z. B. durch eine falsche Berechnung der Rotation als Outlier gekennzeichnet wurde, in einem späteren Schritt wieder berücksichtigt werden.

3.4.5 SLAM mit ICP und Bildmerkmalen

Das Problem der Positionsbestimmung kann zu einem SLAM (Self Localisation and Mapping) Problem umformuliert werden. Vergleicht man nur aufeinanderfolgende Bildfolgen, so kann man die Transformation zwischen den Bildfolgen errechnen. Allerdings ist diese Transformation durch Bildrauschen, Rundungsfehler und ungenau lokalisierte Bildmerkmale nie perfekt. Es entsteht immer ein Fehler, der sich inkrementell addiert, je mehr Bildfolgen betrachtet werden, bzw. je weiter die Bildfolge sich von der Ursprungsposition entfernt ist. Die SLAM Problemstellung versucht diesen Fehler zu minimieren, indem nicht nur zwei Aufnahmepositionen verglichen werden, sondern indem eine Karte erstellt, das aufgenommene Bild mit dieser Karte verglichen und danach eine Position errechnet wird. Die Karte sollte möglichst einfach mit dem Bild vergleichbar sein, weshalb sie ebenfalls aus Bildmerkmalen bestehen sollte. Um nun die additiven Fehler in der Karte zu minimieren, werden Filter verwendet.

Gefundene Bildmerkmale sollen nur direkt in die Karte eingefügt werden, wenn sie neu sind. Werden Bildmerkmale wiedergefunden, so wird deren Position in der Karte aktualisiert. Diese Aktualisierung kann mit einem Kalman Filter realisiert werden. Um diesen Ansatz einfach zu halten und nicht von weiteren Parametern wie Mess-, Positions- und Systemrauschen abhängig zu machen wurde ein eindimensionales Kalman Filter mit festem Mess-, Positions und Systemrauschen verwendet.

In diesem Fall entspricht die Umsetzung des Kalman Filters einem einfachen Mittelwert Filter. Der Preis für eine derartige Vereinfachung ist natürlich eine geringere Genauigkeit von der Position der Bildmerkmale in der Karte, was aber durch die hohe Anzahl an SIFT Bildmerkmalen aufgefangen wird. (Es werden drei Merkmalskorrespondenzen benötigt, um eine Positionsbestimmung durchzuführen, SIFT liefert bei einer Auflösung von bereits $640x480$ durchschnittlich etwa 200 Merkmale.)

Die 3-D Koordinaten der gefundenen Bildmerkmale werden also als Menge D und die Mittelwerte der bisher gefundenen Merkmale als Menge M im ICP Algorithmus verwendet. Nach erfolgreicher Positionsschätzung werden gefundene Bildmerkmale aus der Kamerasicht ins Weltkoordinatensystem transformiert. Neue Merkmale werden der Karte M hinzugefügt, wiedergefundene Merkmale aktualisieren die 3-D Position des korrespondierenden Merkmals $\mathsf{m} \in M$ in der Karte.

3.5 Fusion von Tiefenkarten

Tiefenkarten aus Stereobildern unterliegen einem Rauschen und sind im Allgemeinen recht ungenau. Fusioniert man mehrere Tiefenkarten zu einer einzigen 3-D Punktewolke, überlagern falsch geschätzte Tiefenwerte, welche näher an der Kamera sind, korrekt geschätzte Tiefenwerte. Die 3-D Punktewolke wird also mit jeder Tiefenkarte eher schlechter als besser. Unterschiedliche Tiefenwerte bilden somit Konflikte in einer Sequenz von Tiefenkarten.

Die Idee in der Fusion von Tiefenkarten [NS01] [MAW+07] besteht darin, diese Konflikte aufzulösen, indem benachbarte Tiefenkarten welche ähnliche Bereiche abdecken verglichen werden. Es werden also fusionierte Tiefenkarten $\hat{F}_i(x)$ errechnet, welche dann ohne Konflikte zu einer 3-D Punktewolke zusammengesetzt werden können.

3.5.1 Stability-Based Fusion

Um die Tiefenkarten D_i miteinander zu vergleichen, werden sie nacheinander in eine Referenzansicht D_{ref} projiziert. Es ergeben sich daraus drei verschiedene visuelle Verhältnisse

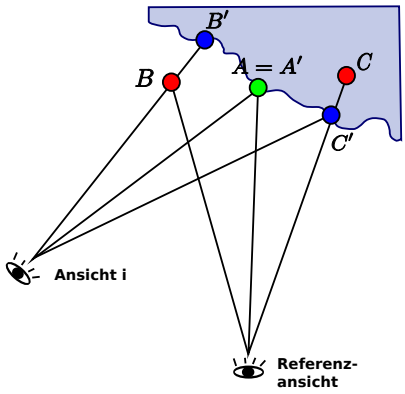


Bild 3.8: Unterschiedliche Formen von Konflikten. Punkt B verursacht einen Free-Space Konflikt mit B', während C' den Punkt C verdeckt. Aus [MAW+07]

zwischen den abgebildeten Tiefenwerten. Siehe hierzu Bild 3.8 .

a) Die Tiefenwerte $\hat{F}_i(x)$ und $D_i(x)$ stimmen überein (bzw. sind innerhalb eines Toleranzbereiches) wie Punkt A und A'.

b) Der neue Tiefenwert $D_i(x)$ würde vom geschätzten Wert von $\hat{F}_i(x)$ verdeckt werden (Free-Space Violation, Punkt B' wird von B verdeckt).

c) Der neue Tiefenwert $D_i(x)$ würde den geschätzten Wert $\hat{F}_i(x)$ verdecken (Okklusion, Punkt C' verdeckt C).

Um bei mehreren unterschiedlichen Tiefenwerten den vorraussichtlich besten Wert zu nehmen, wird eine Stabilität $S(x)$ gespeichert. Diese Stabilität ergibt sich aus der Anzahl von Free-Space Violations wie in b) minus der Anzahl von Okklusionen wie in c). Die Stabilität stellt also das Verhältnis der zwei verschiedenen Formen von Konflikten dar. Ein Beispiel hierzu ist in Bild 3.9 zu sehen. Ein Tiefenwert ist stabil, wenn $S(x)$ größer gleich Null ist. Ist der Wert negativ, so bedeutet dies, dass der Tiefenwert $\hat{F}_i(x)$ zu nahe an der Kamera ist. Ist er zu groß, so ist der Punkt wahrscheinlich zu weit weg. Es wird der Punkt genommen, welcher der Kamera am nächsten und dessen Stabilität größer gleich Null ist.

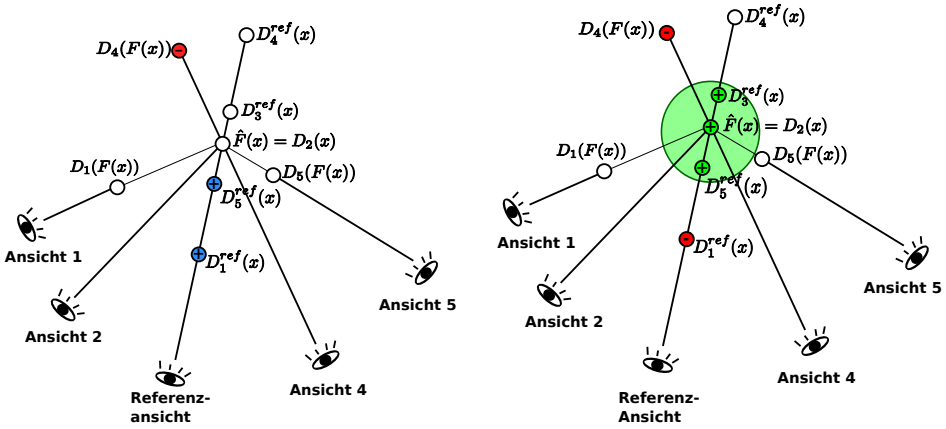
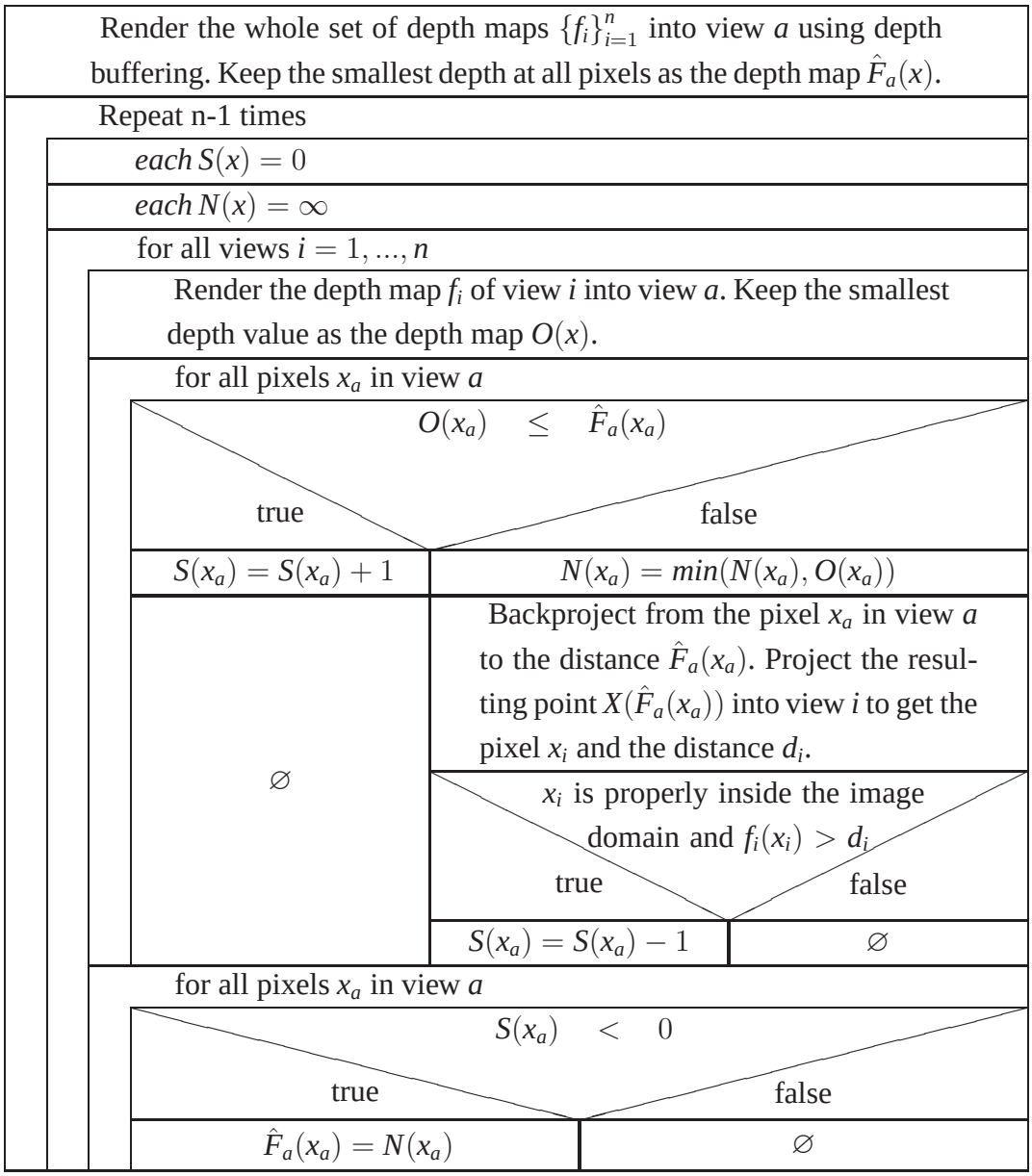


Bild 3.9: Links: Errechnung der Stabilität eines Punktes. Die Tiefenwerte werden in die Referenz Ansicht projiziert. Der geschätzte Wert $\hat{F}(x)$ wird von zwei Werten verdeckt und würde einen anderen Wert verdecken. Er hat somit eine Stabilität von $+1$ und ist von der Kamera aus gesehen der erste Punkt, dessen Stabilität größer 0 ist.
Rechts: Errechnen der Konfidenz. Drei Punkte sind innerhalb eines Toleranzbereiches, während zwei andere Punkte Konflikte erzeugen. Aus [MAW+07]

Dies hat zur Folge, dass ein Punkt genommen wird, der in etwa gleiche Anzahl an Tiefenwerten verdeckt, wie solche, die von ihm verdeckt werden.

Während Free-Space Violations in der Sicht der Tiefenkarte, werden Okklusionen in der Referenzansicht ausgewertet.

Der Algorithmus wird $n - 1$ mal iteriert, wobei n die Anzahl der Tiefenkarten ist. Somit kann sichergestellt werden, dass alle Tiefenwerte als mögliche Schätzungen $\hat{F}(x)$ in Betracht gezogen werden können. Der Großteil der Rechenoperationen besteht also aus dem Umrechnen der verschiedenen Ansichten, was sich allerdings effizient auf der GPU berechnen lässt.



Struktogramm 3.1: Stability-Based Fusion Algorithmus, aus [NS01]

3.5.2 Confidence-Based Fusion

Der Aufwand zum Errechnen der Stability-Based Tiefenkarte ist $\mathrm{O}(N^2)$. Die Unterschiede zwischen den Iterationen der Tiefenkarten sind allerdings laut [MAW+07] äußerst gering. In der Confidence-Based Fusion werden ähnliche Tiefenwerte in einen gemittelten Tiefenwert zusammengefasst und somit die Anzahl an notwendigen Tests reduziert. Der Aufwand reduziert sich auf $\mathrm{O}(N)$. Diese Vorgehensweise ist schneller als bei der Stability-Based Fusion, was aber aufgrund der früheren Festlegung auf einen Wert, zu mehr Fehlern in der fusionierten Tiefenkarte führen kann.

Der Confidence-Based Fusion Algorithmus benötigt Tiefenkarten, welche zusätzlich zu jedem Tiefenwert $D_i(x)$ eine Konfidenz (confidence) $C_i(x)$ enthalten, die aussagt, in wie weit man diesem Tiefenwert vertrauen kann. Diese Konfidenz muss direkt bei der Erzeugung der Tiefenwerte mitberechnet werden [MAW+07].

Es wird dann ein initialer Tiefenwert $\hat{f}_0(x)$ und eine initiale Konfidenz $\hat{C}_0(x)$ für jeden Pixel ermittelt, welcher dem Tiefenwert mit der höchsten Konfidenz $C_i(x)$ aus allen Ansichten entspricht. Dieser Geschätzte Tiefenwert $\hat{f}_k(x)$ und dessen Konfidenz $\hat{C}_k(x)$ wird iterativ mit jeder Tiefenkarte aktualisiert.

Wie auch bei der Stability Based Fusion werden die einzelnen Tiefenkarten in eine Referenzansicht projiziert. Ebenso werden Free-Space Violations in der Sicht der Tiefenkarte und Verdeckungen in der Referenzansicht ausgewertet.

Entspricht der Wert in einer Tiefenkarte $D_i^{ref}(x)$ dem geschätzten Tiefenwert $\hat{f}_0(x)$, werden die beiden Tiefenwerte kombiniert und die Konfidenz aktualisiert:

$$\hat{f}_k(x) = \frac{\hat{f}_{k-1}(x)\hat{C}_{k-1}(x) + D_i^{ref}(x)C_i(x)}{\hat{C}_{k-1}(x) + C_i(x)} \tag{3.8}$$

$$\hat{C}_k(x) = \hat{C}_{k-1}(x) + C_i(x) \tag{3.9}$$

Verdeckt der Wert $D_i^{ref}(x)$ den Tiefenwert $\hat{f}_0(x)$, so wird nur die Konfidenz aktualisiert:

$$\hat{C}_k(x) = \hat{C}_{k-1}(x) - C^{ref}{}_i(x) \tag{3.10}$$

Ebenso wird bei einem Free-Space Konflikt die Konfidenz reduziert:

$$\hat{C}_k(x) = \hat{C}_{k-1}(x) - C_i(P_i(\hat{F}(x))) \tag{3.11}$$

Somit erhöhen ähnliche Tiefenwerte die Konfidenz, während Konflikte die Konfidenz reduzieren. Vertrauenswürdig sind solche Punkte, dessen Konfidenz größer gleich 0 ist.

3.5.3 Eigene Implementierung eines Bayesfilters

Die Stability-Based Fusion erfordert einen großen Aufwand an Umrechnungen zwischen den verschiedenen Ansichten. Es müssen für jede Tiefenkarte $n - 1$ Rendering Schritte durchgeführt werden, was dazu führt, dass der Algorithmus nur auf der Grafikkarte effizient zu implementieren ist. Die Confidence-Based Fusion dagegen ließe sich auch auf der CPU effizient implementieren, da hier weniger Rendering Schritte durchgeführt werden müssen. Jedoch benötigt dieser Algorithmus eine Konfidenz für die Tiefenwerte. Hierfür wäre eine Anpassung der Implementierung (vom Algorithmus zur Disparitätskartenerstellung) erforderlich, um während der Erstellung der Disparitätskarte diese Konfidenz zu errechnen. Die verwendete Implementierung des Birchfield Algorithmus [BT98] von *OpenCV* wird allerdings in dieser Diplomarbeit nicht verändert, wodurch dieser Schritt (das Erstellen von Disparitätskarten) schnell ausgetauscht und durch einen neueren, besseren Algorithmus ersetzt werden kann.

Der hier verwendete Algorithmus stellt eine starke Vereinfachung des Confidence-Based Fusion Algorithmus dar.

Die fusionierten Tiefenwerte werden in zwei Schritten komplett in die Ansicht der jeweils nächsten Disparitätskarte projiziert. Es wird kein Test durchgeführt (mit Ausnahme des Viewports), welcher der vordere Wert in der Ansicht ist und somit werden auch Werte in die Ansicht projiziert, die durch andere verdeckt würden. Selbst mögliche Outlier werden in die Ansicht projiziert und dort getestet. Somit können alle bisherigen Tiefenwerte verglichen werden ohne eine Schätzung durchzuführen, welcher der Vertrauenswürdigste ist, bzw. ohne mehrere Iterationen durchzuführen wie beim Stability-Based Fusion.

Im ersten Schritt werden neue Punkte hinzugefügt und ähnliche Punkte zusammengefasst.

Die bisherige Punktewolke wird in die Ansicht der Disparitätskarte projiziert.

- Ist kein übereinstimmender Tiefenwert für einen Disparitätswert (innerhalb eines Toleranzbereichs) in der bisherigen fusionierten Tiefenkarte vorhanden, wird die 3-D Weltkoordinate des Disparitätswertes berechnet und der fusionierten Tiefenkarte mit der Konfidenz 0 hinzugefügt.
- Entspricht der bisherige Tiefenwert dem Wert in der Disparitätskarte, wird die Konfidenz des 3-D Punktes um 1 erhöht und der Tiefenwert wird mit einem eindimensionalen Kalman-Filter (bzw. Mittelwertfilter) aktualisiert.

Dies wird für alle Tiefenkarten einmal durchgeführt. Im zweiten Schritt werden die Konflikte erkannt und die Konfidenz von Punkten mit Konflikten reduziert. Es werden jedoch keine Punkte hinzugefügt, gelöscht oder dessen Position geändert. Es wird die in Schritt eins errechnete Punktewolke in die Ansicht der Disparitätskarte projiziert.

- Die Konfidenz der Punkte, die (aus Kamerasicht) vor dem Wert aus der Disparitätskarte liegen, wird um 1 reduziert. Dahinter liegende Punkte bleiben unberührt.

Es ist also nur noch nötig eine Art von Konflikten zu berücksichtigen. Okklusions Konflikte werden auf diese Art und Weise implizit erkannt. Der Unterschied zwischen Okklusions Konflikten und Free-Space Konflikten ist die Reihenfolge, in der die Verdeckung stattfindet. Während beim Free-Space Konflikt ein 3-D Punkt den Tiefenwert der Disparitätskarte verdeckt, wird bei einem Okklusions Konflikt ein vorhandener Punkt durch den Disparitätswert überlagert. Wäre der Disparitätswert des Okklusions Konflikts als erstes als 3-D Punkt eingetragen worden, würde er einen Free-Space Konflikt hervorrufen.

Da aber durch den ersten Schritt bekannt ist, welche bisherigen Tiefenwerte vorhanden sind und welche Punkte durch einen Disparitätswert verdeckt würden, müssen diese beiden Konflikte nicht mehr getrennt behandelt werden.

Punkte, die hinter den Disparitätskarten liegen können so jedoch nicht berücksichtigt werden. Da keine Verdeckung berechnet wird, können korrekt geschätzte Disparitätswerte andere korrekt geschätzten Punkte verdecken. In diesem Fall darf deren Konfidenz nicht

reduziert werden. Dies ist aber auch nicht nötig, da falsch geschätzte Punkte, sollten sie hauptsächlich hinter den meisten Disparitätskarten liegen, durch andere Punkte verdeckt und somit in der späteren Rekonstruktion nicht berücksichtigt werden. Gerendert werden später nur die vordersten Punkte, deren Konfidenz größer oder gleich Null ist. Für Punkte, die also meistens hinter den Disparitätskarten liegen ist es irrelevant, ob deren Konfidenz korrekt geschätzt wurde.

Weiterhin bleibt auch die Anzahl der 3-D Punkte gering, da redundante Punkte zusammengefasst und gefiltert werden. Outlier bleiben erhalten und werden erst später in der Ansicht entfernt.

3.6 Mesh Generierung

3-D Modelle sind mehr als nur Punktwolken. Sie bestehen in den meisten Fällen aus Dreiecksnetzen (Mesh), bzw. werden später für die Darstellung in Dreiecksnetze umgewandelt. Die Dreiecke stellen Flächen dar, z. B. den Teil einer Wand oder einen Teil vom Boden. Für die Darstellung werden diese Flächen mit den aufgenommenen Bilddaten texturiert. Somit lässt sich ein guter realer räumlicher Eindruck erzielen. Die aufgenommenen Bilder werden während des Rendering Prozesses der aktuellen Betrachterposition entsprechend räumlich verzerrt.

Das Problem bei der Generierung eines solchen Dreiecksmeshs ist, dass nicht bekannt ist, welche Punkte eine Fläche bilden und durch Dreiecke dargestellt werden können. Man kann das Mesh lediglich approximieren indem man bestimmte Einschränkungen definiert, wann Punkte zu Dreiecken verbunden werden sollen und wann nicht.

Für den 2-D Fall gibt es eine sehr effiziente Möglichkeit der Dreiecksbildung - die Delaunay Triangulation. Bei der Delaunay Triangulation werden Punkte so miteinander zu Dreiecken vernetzt, dass innerhalb eines Kreises auf dessen Radius drei Punkte liegen, keine weiteren Punkte enthalten sind. Alle Dreiecke in dem Dreiecksnetz erfüllen diese Bedingung, wodurch der kleinste Innenwinkel über alle Dreiecke maximiert wird. In Bild 3.10 wurden 25 Punkte mit der Delaunay Triangulation zu Dreiecken verbunden.

Die Delaunay Triangulierung ist jedoch nicht eindeutig, falls auf einem Umkreis mehr

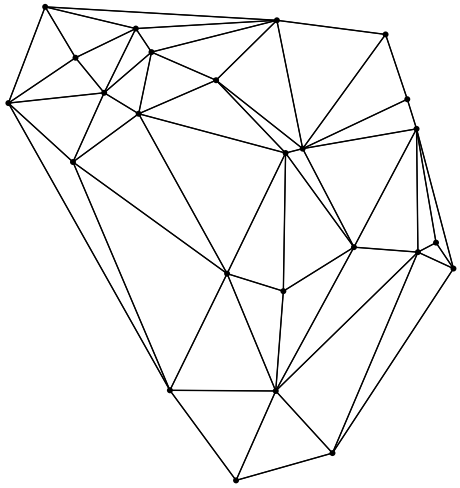


Bild 3.10: Delaunay Triangulierung. Aus [Del08]

als drei Punkte liegen (z. B. wenn vier Punkte die Ecken eines Quadrats bilden). Sollen aus einem mit dem Delaunay Verfahren triangulierten Mesh, doppelte Dreiecke entfernt werden, muss dies berücksichtigt werden.

In dieser Diplomarbeit wurde zur 2-D Triangulierung die *Triangle* Bibliothek verwendet, die effizient und einfach eine Punktemenge zu einem Mesh verbindet. Mit der zur Ausdünnung verwendeten *GNU Triangulate Surface Library* wäre dies zwar ebenfalls möglich, jedoch ist die API nicht primär zur Triangulation ausgelegt und weniger komfortabel als bei der *Triangle* Bibliothek. Der in *OpenCV* integrierte Delaunay Algorithmus ist nicht geeignet um daraus schnell ein Delaunay Mesh zu erstellen. Die *OpenCV* Implementierung liefert nur unsortierte Kantenpaare von jeweils zwei Punkten, aus welchen zwar ein Dreiecksmesh gezeichnet werden kann, aber aus denen nur mit weiteren aufwendigen Bearbeitungsschritten Dreiecke erstellt werden können.

Für den 3-D Fall gibt es ebenfalls die 3-D Delaunay Triangulation, bei der vier Punkte einen Tetraeder formen. Jedoch hat diese das Problem, dass auch die komplette konvexe Hülle verbunden wird. In der Darstellung würde man bei der Triangulation einer Punktewolke nur noch die konvexe Hülle der Blickwinkelpyramiden sehen können. Alle darin enthaltenen Objekte und Strukturen würden verdeckt. Viele Algorithmen versuchen nun

die Dreiecke wieder zu löschen, die zur konvexen Hülle gehören. Allerdings ist diese Methode eher ineffizient. Zum einen ist die 3-D Delaunay Triangulation ein äußerst rechenintensiver Prozess, zum anderen ist die Auflösung der konvexen Hülle ebenfalls nicht trivial und dauert fast genauso lange. Somit kann dieser Prozess bei großen Punktewolken von einer Millionen Punkten selbst auf moderner Computerhardware bis zu einer Stunde dauern. Die anderen hier vorgestellten Ansätze lösen dieses Problem in einem Bruchteil der Zeit.

3.6.1 Schnelle Mesh Generierung aus einer Punktewolke

Dieser Ansatz zum Erzeugen eines Mesh aus einer Punktewolke wird in dem Rekonstruktionsansatz von Bradley, Boubekeur und Heidrich [BBH08] beschrieben.

Für die Mesh Generierung, wird eine Art Octree benutzt, in den die Punktewolke eingefügt wird. Die Besonderheit dieses Octrees ist die Aufteilung eines Blattes in weitere Blätter, falls die enthaltenen Punkte keine planare Ebene bilden. Die in den Blättern übrig gebliebenden Punkte liegen also voraussichtlich auf einer Ebene. Dadurch wird die Triangulierung zu einem Mesh vereinfacht. Die 3-D Triangulierung kann somit zu einer 2-D Delaunay Triangulierung für jedes Blatt im Octree reduziert werden. Die gewonnenen Flächen müssen dann noch verbunden werden. Dies wird ermöglicht, indem für die Triangulation Punkte aus benachbarten Blättern miteinbezogen werden. Hierbei müssen folgende Dreiecke aus dem resultierenden Mesh wieder gelöscht werden:

- Doppelt vorkommende Dreiecke.
- Dreiecke, die aufgrund der nicht eindeutigen Delaunay Triangulierung die selbe Fläche repräsentieren.
- Dreiecke, die weit außerhalb der Flächen liegen und durch falsche Verbindungen zwischen den Flächen entstanden sind.

Allerdings entstehen beim Verwenden dieser Methode mehrere Probleme. So ist die Auswahl der benachbarten Punkte eher schwierig. Werden zu wenige Punkte des Nachbarblat-

tes genommen, so entstehen Löcher. Werden zu viele Punkte genommen, entstehen Verbindungen zu sehr weit entfernten Objekten, wodurch die eigentliche Ebene wieder verdeckt wird. Würde man beispielsweise alle Punkte von allen Blättern nehmen, so hätte man lediglich die Konvexe Hülle der Blickwinkepyramide. Hierbei ist eine weitere Schwierigkeit, dass die Blätter durch die Unterteilung unterschiedlich viele Punkte enthalten und ein Blatt auch unterschiedlich viele Nachbarn haben kann. Auch besteht die Gefahr, dass einige (räumlich) sehr große Blätter entstehen, die aus so wenigen Punkten bestehen, dass diese nicht weiter unterteilbar sind. Diese Blätter müssen erkannt und gelöscht werden. Die Unterteilung der Blätter wird nicht an Stellen mit großen Unterschieden gemacht, sondern in der Mitte des Volumens eines Blattes. Die Texturkoordinaten der Dreiecke müssen ebenfalls mit anderen Methoden errechnet werden.

Nachdem einige Prototypen keine zufriedenstellenden Ergebnisse lieferten, wurde wegen der oben genannten Schwierigkeiten entschieden, eine einfachere Methode zu nehmen.

3.6.2 Mesh Generierung aus einer Ansicht

Eine einzelne Tiefenkarte zu einem Mesh zu triangulieren ist unkompliziert. Hierbei kann die 2-D Tiefenkarte als Ebene betrachtet werden. Die einzelnen Pixel in der Ebene werden mittels Delaunay Triangulierung verbunden. Danach können die 3-D Koordinaten der einzelnen Pixel in der Tiefenkarte errechnet werden, wobei die Dreiecke des 2-D Meshs nun Dreiecke im 3-D Mesh bilden.

Hat man nun aus mehreren Tiefenkarten eine gefilterte Punktewolke (bzw. fusionierte Tiefenkarte) errechnet, lässt sich daraus ebenfalls sehr leicht für eine Ansicht ein Mesh erstellen. Die Punktewolke wird hierzu in eine Ansicht projiziert, wodurch man eine 2-D Tiefenkarte errechnen kann. Diese Tiefenkarte wird mit der oben genannten Methode wieder zu einem Mesh verbunden.

Fusionierte Tiefenkarten enthalten allerdings noch Fehler, die daraus resultieren, dass die Projektion von Tiefenkarten in eine andere Ansicht nie perfekt ist. Durch unterschiedliche Skalierungen entstehen kleine Löcher bei der Projektion, wodurch einige Punkte einen falschen Tiefenwert oder eine falsche Konfidenz erhalten. Diese Punkte werden nach der Projektion in eine Ansicht durch einen Median Filter rausgefiltert.

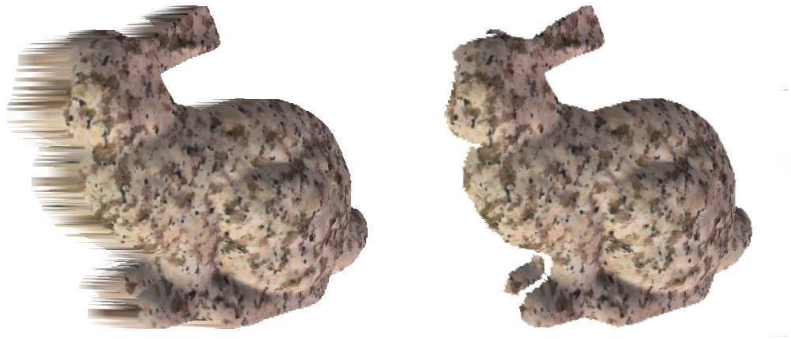


Bild 3.11: Links: Rubber-Sheet Artefakte. Durch Veränderung der Ansicht verzerren Dreiecke mit stark unterschiedlichen Tiefenwerten. Rechts: Segmentierung von Objekten mit unterschiedlichen Tiefenwerten. Dreiecke mit starken Unterschieden in der Tiefe werden entfernt. Aus [PSM04]

Durch die Projektion der 3-D Punkte in die Ansicht eines Bildes können auch direkt die Texturkoordinaten für das spätere Rendering gespeichert werden. Das Texture-Mapping kann dann effizient mit einer hardwarebeschleunigten Grafikkarte dargestellt werden.

Das Problem hierbei ist natürlich, dass das resultierende 3-D Modell nur für eine Ansicht korrekt ist, nämlich für die, in der es erstellt wurde. Sobald es aber Flächen gibt, die in der Original Ansicht verdeckt sind und die durch Rotation des 3-D Modells sichtbar werden würden, kommt es zu Verzerrungen am Rand von diesen verdeckten Flächen (Rubber-Sheet Artefakte). Siehe hierzu Bild 3.11.

3.6.3 Segmentierung von Tiefenwerten

Um sogenannte Rubber-Sheet Artefakte zu vermeiden müssen Objekte mit unterschiedlichen Tiefenwerten segmentiert werden. Hierzu werden die Verbindungen zwischen diesen Objekten entfernt. Es reicht, diese Verbindungen zu erkennen und zu entfernen, welche in Dreiecken an großen Unterschieden im Tiefenbild entstehen. Hierzu werden jeweils drei benachbarte Punkte im Tiefenbild verglichen [MFA+07].

$$\left|\frac{z_0 - z_1}{z_0} - \frac{z_1 - z_2}{z_1}\right| < t \tag{3.12}$$

Dieser Vergleich wird horizontal in der Bildzeile, sowie vertikal für jede Spalte durchgeführt. Der Wert *t* ist hierbei ein Schwellwert. Ist der Unterschied der Tiefenwerte in einem der Vertices eines Dreiecks horizontal oder vertikal über diesem Schwellwert, wird das Dreieck aus dem Mesh entfernt.

3.6.4 Zusammensetzen mehrerer Meshs

Lässt sich aus einer Ansicht ein Mesh generieren, so müssten sich auch aus mehreren Ansichten einfach mehrere Meshs generieren und zusammensetzen lassen. Hierbei muss wieder darauf geachtet werden, dass wie in Kapitel 3.6.1 weder Dreiecke doppelt vorkommen sollen, noch solche welche die gleiche Fläche abbilden in das zusammengesetzte Mesh übernommen werden.

Um effizient neue Dreiecke hinzuzufügen, werden bereits durch Dreiecke dargestellte Punkte gespeichert.

Ist ein Punkt in einem Mesh aus einer Ansicht neu, wurde das gesamte Dreieck noch nicht dargestellt und kann hinzugefügt werden. Nicht nur die Vertices der Dreiecke, sondern alle Punkte innerhalb eines Dreiecks werden gespeichert. Bei der Umrechnung der 3-D Punkte in eine Tiefenkarte werden also alle Punkte, die im Pixel der Tiefenkarte liegen und einen ähnlichen Tiefenwert haben gespeichert. Dies verhindert das Hinzufügen von neuen Dreiecken innerhalb eines anderen Dreiecks einer größer skalierten Ansicht. Die Neuen könnten durch die Vorhandenen teilweise verdeckt werden, was zu einem inkonsistenten Mesh führen würde.

Die Mesh Generierung wäre auch möglich, wenn man bei der Umrechnung der Tiefenkarte nicht mit Pixeln arbeitet, sondern das Mesh einer Ansicht aus allen nach 2-D projizierten 3-D Punkten erstellt (die ebenfalls eine ähnlichen Tiefenwert haben). Allerdings hat sich gezeigt, dass diese Methode zu einer eher unruhigen Oberfläche führt, wodurch mehr Fehler beim Texture-Mapping entstehen und sehr kleine Rubber-Sheets die gerenderte Ansicht extrem pixelig erscheinen lassen.

In den vorher aussegmentierten Bereichen mit großen Tiefenunterschieden können neue Dreiecke erstellt werden, falls in der neuen Ansicht an diesen Stellen weniger große Tie-

fenunterschiede sind.

3.6.5 Reihenfolge der einzelnen Meshs beim Zusammensetzen

Die Reihenfolge der Ansichten ist beim Zusammensetzen mehrerer Meshs leider ausschlaggebend für die Form und das Aussehen des zusammengesetzten Meshs. Wird bei der Wahl der Textur die erste Ansicht eines Dreiecks genommen in der das Dreieck sichtbar ist, so lässt sich dadurch, dass man erst Nahansichten beim Zusammensetzen nimmt, ein sehr gutes detailliertes Ergebnis erzielen.

Weiter ist das Erkennen von Rubber-Sheet Artefakten nicht immer perfekt. So werden einige Dreiecke, die noch gerade den Schwellwert von Formel 3.12 unterschreiten, aus der ersten Ansicht in der sie vorkommen erstellt und texturiert, auch wenn sie in einer späteren Ansicht womöglich besser sichtbar sind. Die Genauigkeit ist bei diesen Dreiecken sowohl in der Textur, als auch in der Topologie eher schlecht, da durch die Pixelungenauigkeit der Tiefenkarten viele Punkte nicht berücksichtigt werden können. Würden diese Dreiecke in einer anderen Ansicht besser dargestellt, so wird dies ignoriert, da die Dreiecke als doppelt erkannt werden.

Die Reihenfolge der Ansichten kann hier nicht einfach verändert werden, da eine Ansicht die Genauigkeit von einigen Dreiecken verbessern, aber von anderen verschlechtern kann.

Was leider nicht mehr umgesetzt wurde ist die Erstellung einer Maske, in der Flächen aus der 3-D Punktewolke ihren idealen Ansichten zugewiesen wird. Indem man aus umliegenden Punkten eine Normale für jeden Punkt errechnet, könnte man jedem Pixel eine bessere Ansicht zuordnen.

Somit würde nur aus der jeweils idealen Ansicht ein texturiertes Mesh für eine Fläche erstellt. Das Problem hierbei ist natürlich, dass es starke Abweichungen in den Normalen gibt, da die Flächen öfters kleinere Treppenstrukturen aufweisen, welche durch die Disparitätskarten hervorgerufen werden. Dadurch kann es vorkommen, dass mehrere kleine Flächen sehr vielen unterschiedlichen Ansichten zugewiesen werden, wobei einige wenige Ansichten zu wesentlich besseren Ergebnissen führen würden (siehe [PNF+08]).

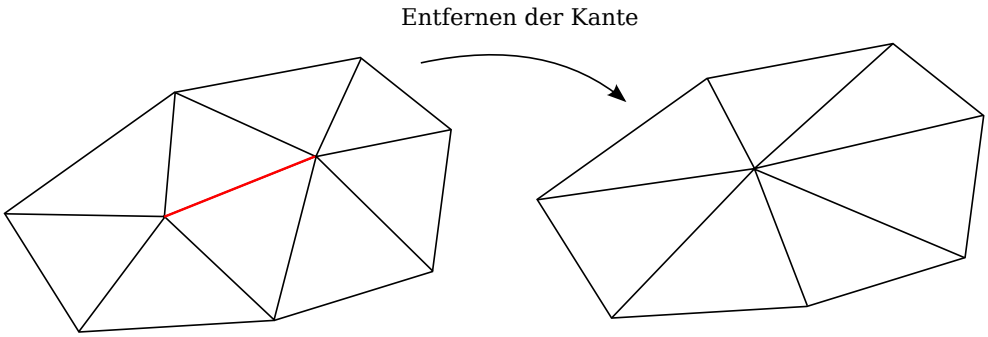


Bild 3.12: Entfernen einer Dreieckskante, aus [LT98].

3.6.6 Ausdünnung eines Meshs

Die rekonstruierten Modelle enthalten sehr viele Dreiecke. Wenn man bedenkt, dass die Dreiecke zwischen den einzelnen Pixeln erzeugt werden, entstehen hierdurch sehr viele Flächen, die auch durch deutlich weniger Dreiecke womöglich mit den selben visuellen Ergebnis dargestellt werden könnten.

Der Zeitaufwand zum Berechnen eines ausgedünnten Meshs ist allerdings groß und es besteht die Gefahr, dass sich durch das Entfernen von Dreiecken Fehler in das Modell einschleichen.

Zum Ausdünnen des Mesh wurde hier eine Implementierung des Memoryless-Simplification Algorithmus von Lindstrom und Turk [LT98] aus der *GNU Triangulated Surface Library* verwendet.

Der Memoryless-Simplification Algorithmus löscht Dreieckskanten, fügt an ihrer Stelle einen neuen Vertex ein und verbindet die übrig gebliebenen Vertices wieder zu Dreiecken. In Bild 3.12 wurde auf diese Weise die mittlere Kante entfernt. Das Problem hierbei ist die Auswahl, der zu entfernenden Dreieckskanten und die Stelle wo neuer Vertex eingefügt werden soll, um den entstehenden Fehler im Mesh zu minimieren.

Um zu entscheiden, welche Dreieckskanten gelöscht werden sollen, werden für jede Kante Kosten ermittelt, die den Fehler im Mesh approximieren der entsteht, wenn die Kante

entfernt wird. Diese Kosten basieren auf dem Fehler

- im neuen Volumen,
- in der Umrandung (Boundary) und
- in der Dreiecksform,

welcher durch ein Entfernen der Kante erzeugt werden.

Der Fehler im Volumen wird errechnet, indem der potentiell neu eingefügte Vertex mit den ursprünglichen Vertices (eines Dreiecks) einen Tetraeder bilden. Dieser Test wird für alle Dreiecke errechnet, die vom Entfernen einer Kante betroffen sind und für die Kante aufsummiert. Der Fehler im Volumen des Modells steigt mit dem Volumen dieser Tetraeder.

Analog wird auch die Umrandung berechnet. Hier wird anstelle des Volumens der Flächeninhalt der von den Vertices eingeschlossenen Fläche berechnet.

Für planare Punkte könnten dadurch jedoch unendlich viele Kandidaten gefunden, für die das Volumen des Tetraeders immer Null ist. Somit wird zusätzlich noch die Dreiecksform, also die Winkel der Dreiecke berücksichtigt, um aus planaren Flächen neue Dreiecke zu bilden, welche möglichst wenig spitze Winkel enthalten.

Die Einzelheiten zur Approximation des Fehlers, sowie deren Herleitung können [LT98] entnommen werden.

3.6.7 Ausdünnen des Meshs bereits vor dem Zusammensetzen

Das Ausdünnen des Meshs mit dem Memoryless Simplification Algorithmus dauert, oft mehrere Minuten. Andere Ansätze haben gezeigt, dass dieses Ausdünnen des Meshs in (nahezu) Echtzeit möglich ist, wenn das Ausdünnen direkt bei der Umrechnung von der Tiefenkarte in ein Mesh ausgeführt wird.

Hierzu wird bereits in der Tiefenkarte ein Quadtree erstellt [MFA+07]. Es wird von einer sehr groben Unterteilung ausgegangen. Die einzelnen Knoten werden nach dem Top-Down Prinzip weiter unterteilt, wenn diese einen sehr großen Tiefenunterschied aufweisen

(siehe hierzu Gleichung 3.12). Die Knoten werden dann zur Triangulierung in Dreiecke unterteilt.

Doppelte Dreiecke werden durch den Vergleich zu einem vorher gerenderten Mesh vermieden.

Dieser Ansatz zum Ausdünnen des Meshs wurde in dieser Diplomarbeit nicht benutzt, da sowohl die korrekte Unterteilung der Knoten des Quadtrees in Dreiecke (siehe [Paj02]), als auch das Rendering der vorherigen Tiefenkarte aus dem ausgedünnten Mesh den Zeitplan überschritten hätte.

3.6.8 Texturierung

Nachdem das Dreiecksmesh erstellt und ausgedünnt ist, muss für jedes Dreieck eine geeignete Textur gefunden werden. Als Texturkoordinaten dienen die nach 2-D projizierten 3-D Koordinaten der Dreiecke, für die verschiedenen Ansichten der Stereobilder. Nun muss noch für jedes Dreieck entschieden werden, welche Ansicht (bzw. welches Bild) die beste Textur liefert. Hierbei kann z. B. die erste Ansicht genommen werden, in der das Dreieck zu sehen ist, was bereits zu sehr guten Ergebnissen mit sehr gleichmäßiger Beleuchtung führt. In Kapitel 4.2 sind alle gezeigten Modelle, die nicht ausgedünnt wurden auf diese Weise texturiert worden.

Soll jedoch eine Textur genommen werden, die möglichst frei von Ausreißern, unterschiedlichen Beleuchtungen und beweglichen Objekten ist, wird in [NS01] vorgeschlagen, erst eine Median Textur von allen Texturen in denen das Dreieck sichtbar ist zu erstellen. Die beste Textur ist diejenige, die den geringsten Abstand zur Median Textur hat. [NS01] nimmt hierfür die Summe der absoluten Differenzen (SAD).

Das Erstellen der Median Textur kann leicht auf der Grafikkarte implementiert werden. Hierfür wird ein Sichtfenster in z. B. 50 mal 50 Quadrate unterteilt, wobei jedes Quadrat zwei Dreiecke enthält. Die Textur der vorher errechneten Dreiecke wird nun auf diese neuen Dreiecke gemappt. Die Dreiecke sind somit unabhängig von deren Größe im Mesh und unabhängig von deren Größe in der Ansicht. Diese erhebliche Vereinfachung erleichtert sowohl die Erstellung der Median Textur, als auch den Vergleich zur Median Textur. Es

dürfen natürlich nur Texturen bei der Berechnung des Medians berücksichtigt werden, auf denen das Dreieck sichtbar ist. Verdeckte Dreiecke dürfen nicht gerendert werden.

Die Verdeckung kann mit den zuvor gespeicherten fusionierten Tiefenkarten getestet werden. Da sich die Vertices geändert haben und womöglich nicht mehr genau den selben Tiefenwert haben muss die Verdeckung jedoch mit einem leichten Ungenauigkeit berechnet werden. Dreiecke werden also nur verdeckt, wenn sich deren Vertices weit (um einen vorher festgelegten Wert) hinter der Tiefenkarte befinden.

Kapitel 4

Experimente und Ergebnisse

4.1 Versuchsaufbau und Durchführung

Zum Testen des Rekonstruktionsverfahrens wurde ein Stereokamerasystem aus zwei *Guppy F-146C* Firewire Kameras fertiggestellt. Die Kameras wurden auf eine Schiene geschraubt, so dass die Blickwinkel der Kameras möglichst parallel verlaufen. Um Fehler bei der Montage sowie die Linsenverzerrung der Objektive auszugleichen wurden sie zusätzlich mit einem vorhandenen Schachbrettmuster und dem Programm aus Anhang B.3 kalibriert.

Die aufgenommenen Bilder wurden sowohl für die Kalibrierung als auch für die Verarbeitung mit dem Rekonstruktionsalgorithmus vorher von 1280×960 auf 640×480 Pixel runterskaliert. In der Originalgröße war die Ausführungszeit erheblich größer, während die Qualität aufgrund der beschränkten Bildschirmgröße kaum zugenommen hat.

Es wurden folgende Bildsequenzen erzeugt:

Robbie1: Diese Szene beinhaltet hauptsächlich Rotationen der Kamera. Die Betrachterposition wurde kaum verändert. Die Hauptaufgabe ist hier aus den einzelnen Bildern, die verschiedene Bereiche des Raums abdecken, eine zusammenhängende Gesamtszene zu machen und in Bereichen mit großem Tiefenunterschied die erstellten Dreiecke wieder zu löschen. Als zusätzliche Schwierigkeit waren hier die Kame-

ras schlecht synchronisiert. Die Stereobilder wurden teilweise zu unterschiedlichen Zeitpunkt aufgenommen, wodurch in der Bewegung falsche Tiefenwerte ermittelt wurden. Auch verursachen die Lichtverhältnisse eine leichte Überbelichtung, wodurch die Textur schlechter für die Tiefenkartenschätzung geeignet ist. Die Szene besteht aus 15 Stereobildern. Siehe hierzu Bild 4.1.

Robbie2: In dieser Szene wird hauptsächlich der selbe Bereich wiederholt aufgenommen, allerdings aus stark unterschiedlichen Blickwinkeln. Die Szene unterscheidet sich von der Robbie1 Szene dadurch, dass die Kamera sich stärker bewegt (sowohl Rotation und Translation), durch eine bessere Synchronisation der Kameras, durch Spiegelungen am Boden an der rechten Seite und durch eine bessere Belichtung. Siehe hierzu Bild 4.2. Ein starker Schwierigkeitsgrad in dieser Szene ist die Aufnahmeposition der Bilder, bei der die Flächen der Szene so gut wie nie parallel zur Bildeben sind. Dadurch wird im Tiefenbild immer eine Treppenstruktur geschätzt. Die Szene besteht aus 13 Stereobildpaaren.

Gesicht: In dieser Szene wird getestet, wie gut ein Gesicht verfolgt und rekonstruiert werden kann. Das Gesicht wird aus nur vier unterschiedlichen Positionen aufgenommen. Mehrere Positionen wären schwierig gewesen, da ein Gesicht nicht statisch ist und sich immer leicht bewegt. Diese Bewegungen kann man auch in Bild 4.3 erkennen.

Neon&Chrome: Hier wurde eine künstliche Szene mit *Autodesk Maya* simuliert. Die Szene zeigt eine Kamerafahrt parallel zu einem Gebäude. Es gab hier keinerlei Rotation sondern nur eine Translation senkrecht zur Fahrtrichtung. Ein Anwendungszenario könnte hier z. B. eine Kamerafahrt von einem *Google Street View* Fahrzeug sein, welches auf einer Straße fährt und mit seitlich ausgerichteten Kameras die Gebäude filmt.

Die *Neon&Chrome Maya* Szene wurde aus [Dec06] entnommen. Da keine Beleuchtung und ebenfalls keine Texturen in der ursprünglichen Szene vorhanden waren, wurden in Eigenarbeit mehrere Lichtquellen, Bump Maps und Texturen hinzugefügt. Drei der 17 künstlich erzeugten Stereobilder sind in Bild 4.4 zu sehen.

Bild 4.1: Drei Stereobilder der Szene *Robbie1*. Die Sequenz zeigt abwechselnd das Bild der linken und der rechten Kamera.

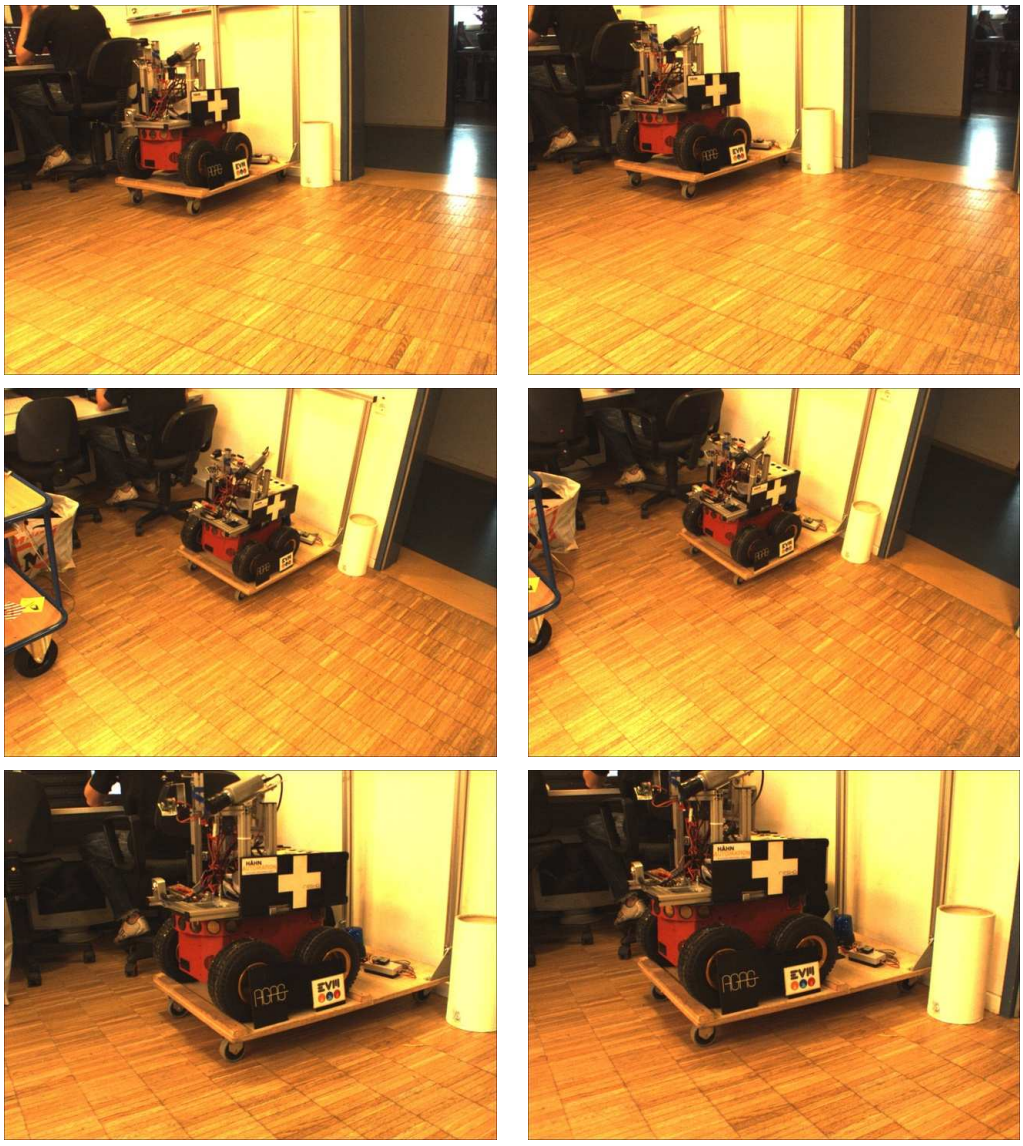


Bild 4.2: Drei Stereobilder Szene *Robbie2*. Die Sequenz zeigt abwechselnd das Bild der linken und der rechten Kamera.

Bild 4.3: Drei Stereobilder der Szene *Gesicht*. Die Sequenz zeigt abwechselnd das Bild der linken und der rechten Kamera.

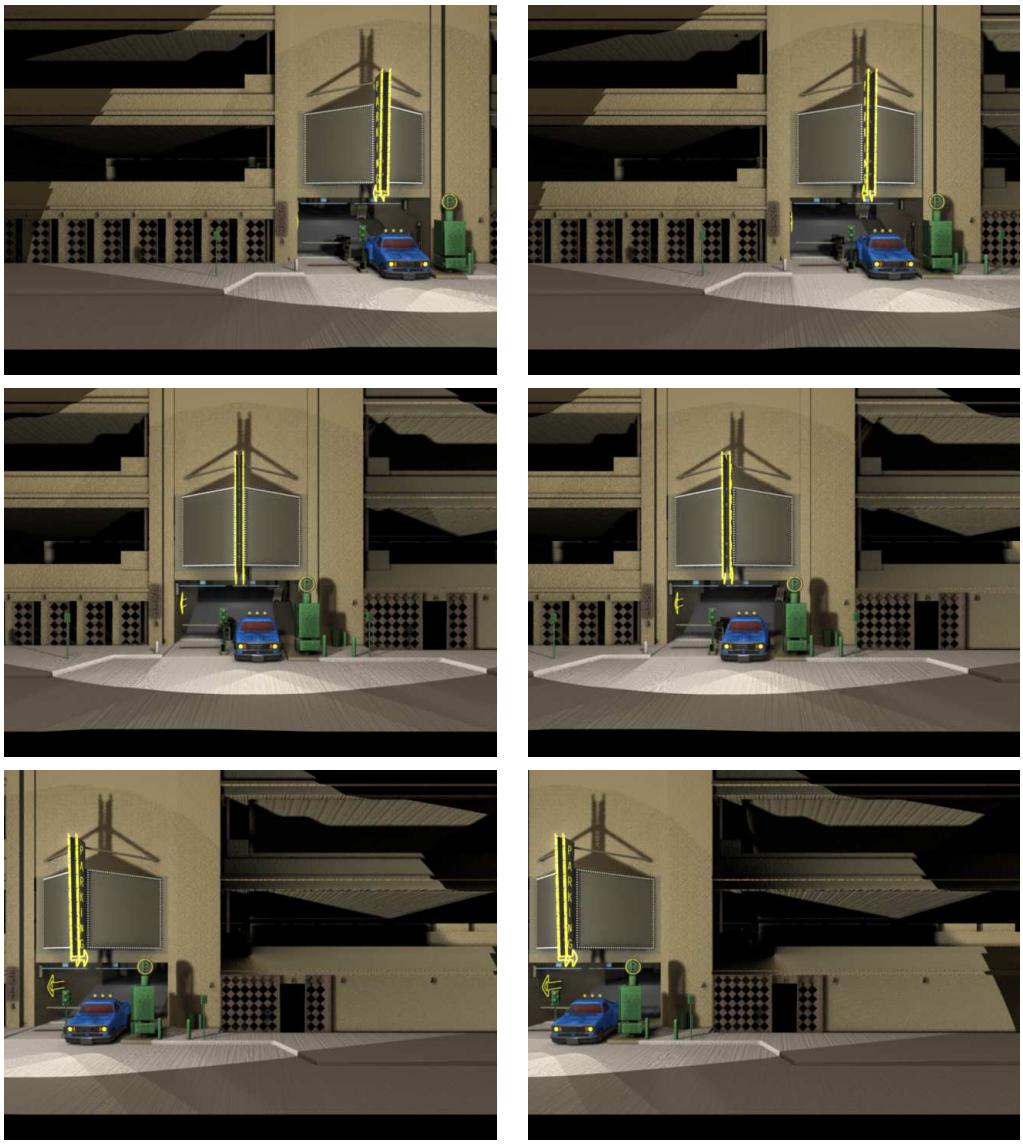


Bild 4.4: Drei Stereobilder der Szene *Neon&Chrome*. Die Sequenz zeigt abwechselnd das Bild der linken und der rechten virtuellen Kamera.

4.2 Ergebnisse

Robbie1: Durch die schlechte Synchronisation der Kameras entstehen fehlerhafte Tiefenkarten, die wieder zu einer falschen Positionsbestimmung führen. Falsche Positionsbestimmungen führen wiederum zum schlechten Matchen der Tiefenkarten, was wieder den Algorithmus zur Fusion der Tiefenkarten negativ beeinflusst und zu einer fehlerhaften Topologie der Oberfläche führt. Bis auf kleinere Fehler wurde allerdings die Oberfläche in Bild 4.5 gut rekonstruiert. Die Ansichten wurden jeweils so gewählt, dass entweder die rechte oder die linke Seite in der Rekonstruktion weniger verzerrt.

Robbie2: Die enormen Bewegungen und unterschiedlichen Aufnahmepositionen führen zu schlecht vergleichbaren Bildmerkmalen. Die Verdeckungen und Tiefenunterschiede führen dazu, dass die Szene nicht aus allen aufgenommenen Blickwinkeln betrachtet gut aussieht. Die Rekonstruktion (Bild 4.6) hat jedoch ein recht gutes optisches Ergebnis. Es wurde nachträglich manuell noch die Reihenfolge der Bilder geändert, um Nahaufnahmen des Roboters zuerst auszuwerten. Dadurch wird die Textur in diesem Bereich mit sehr hohem Detailgrad erstellt. Die Reihenfolge hätte auch bereits bei der Aufnahme berücksichtigt werden können, was zum selben Ergebnis geführt hätte.

Gesicht: Die Szene „Gesicht“ war eher schwierig zu rekonstruieren. Außer dem Gesicht und dem Oberkörper sind kaum Texturen vorhanden. Fast alle Bildmerkmale zur Bestimmung der Position sind somit im Gesicht und auf dem aufgenommenen Oberkörper. Zwischen den aufgenommen Bildern liegen mehrere Sekunden, in der sich die Person hätte bewegen können. Es konnten auch nur zwischen 42 und 89 Bildmerkmale zwischen zwei Stereobildern einander zugeordnet werden, von denen mehrere Korrespondenzen aufgrund falscher Zuordnung wieder aufgelöst werden mussten. Dennoch ist das Ergebnis in Bild 4.7 durchaus zufriedenstellend. Einige Blickwinkel, wie im unteren Bild führen jedoch zu einer starken Verzerrung des Gesichts.

Neon&Chrome: Durch die Texturen, sowie durch die unterschiedlichen Licht und Schattenspiele wirkt die Szene nicht nur realistischer, sondern es ermöglicht auch ein besseres Schätzen von Disparitätskarten. Nachteilig hat sich dies allerdings auf die Suche nach ähnlichen SIFT Merkmalen ausgewirkt. Die Textur wurde gekachelt auf die Objekte gelegt, wodurch einige Bereiche sich nur noch durch Rotation, Skalierung oder Beleuchtung der Textur unterscheiden. Obwohl die Methoden aus [Dec06] zur Vermeidung ähnlicher Zuordnungen übernommen wurden, sind trotzdem im Vergleich zu den realen Bildern enorm viele Bildmerkmale falsch zugeordnet. Dies wurde allerdings vom ICP Algorithmus in allen Fällen komplett ausgeglichen. Die Szene konnte bis auf die Verkehrsschilder und die Verdeckung durch das Auto gut rekonstruiert werden (Bild 4.8).

Bild 4.5: Zwei gerenderte Bilder der Szene Robbie1.

Bild 4.6: Drei gerenderte Bilder der Szene Robbie2.

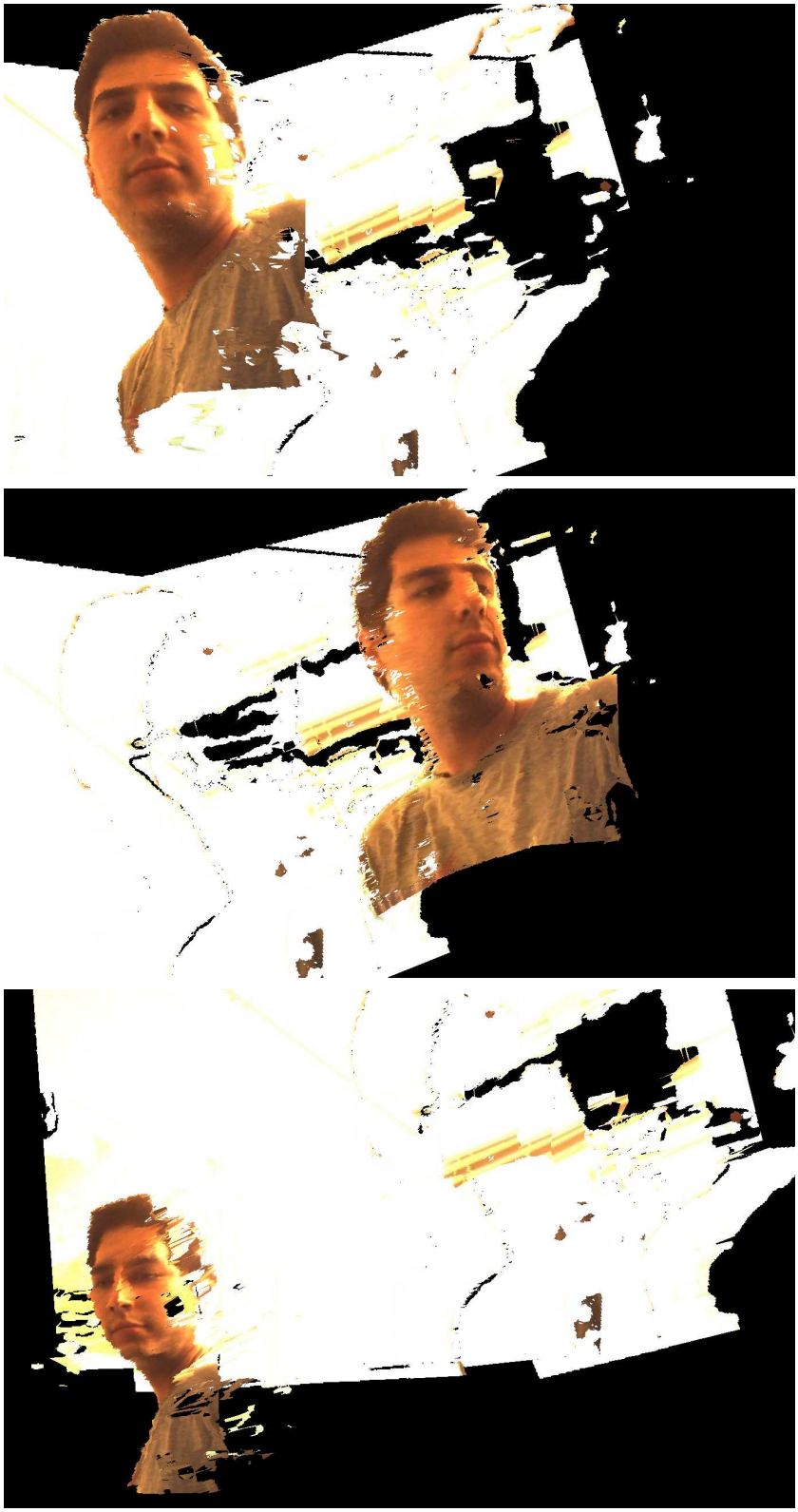


Bild 4.7: Drei gerenderte Bilder der Szene Gesicht.

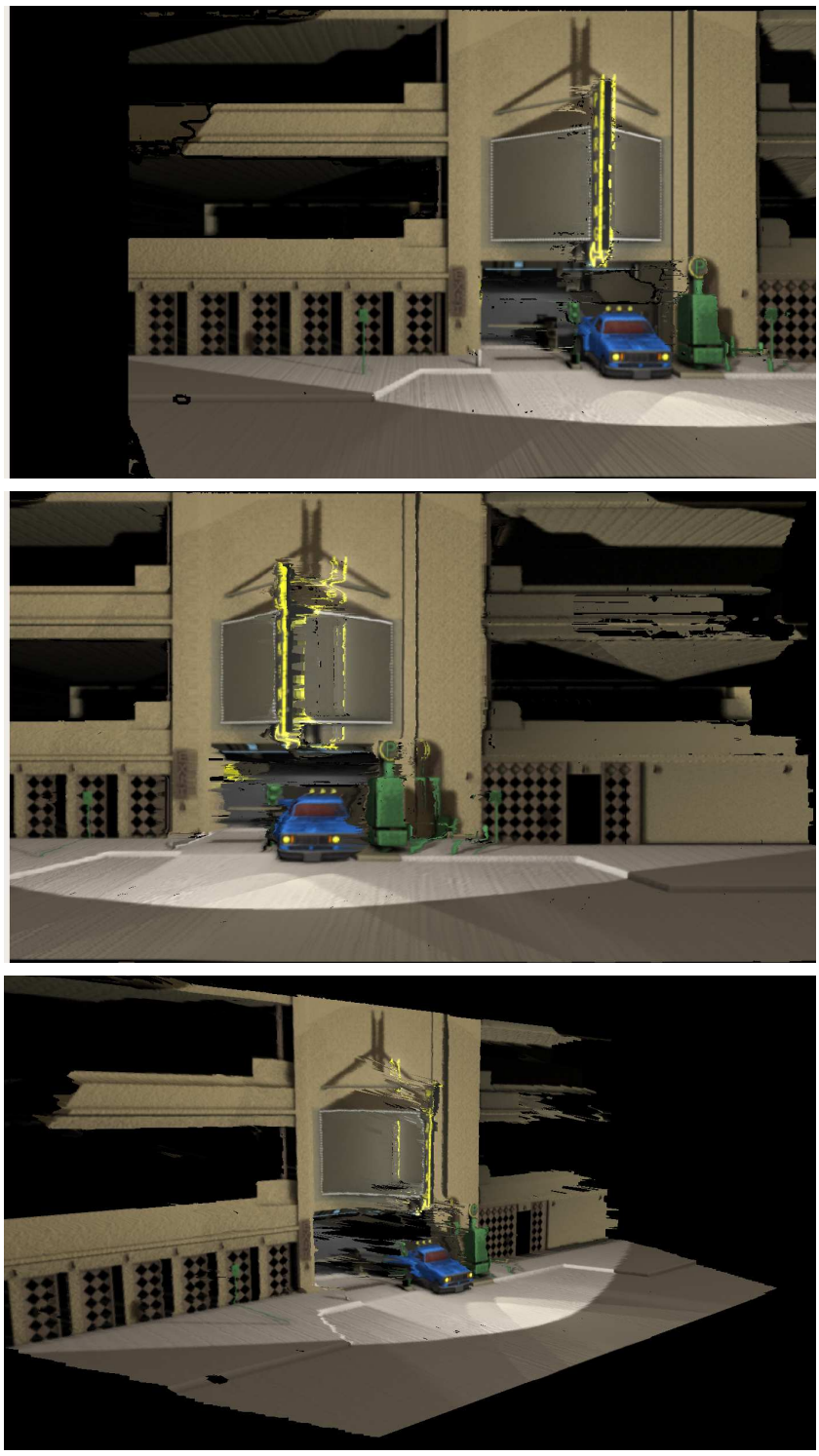


Bild 4.8: Drei gerenderte Bilder der Szene Neon&Chrome. Oben: Das erste Bild ist aus der selben Ansicht, wie das erste Stereobild, wodurch die Texturen nahezu fehlerfrei auf das Modell gelegt werden können. Unten: Die Szene wurde in eine Ansicht rotiert, die aus keinem der Stereobilder sichtbar ist, da diese parallel zum Gebäude erzeugt wurden.

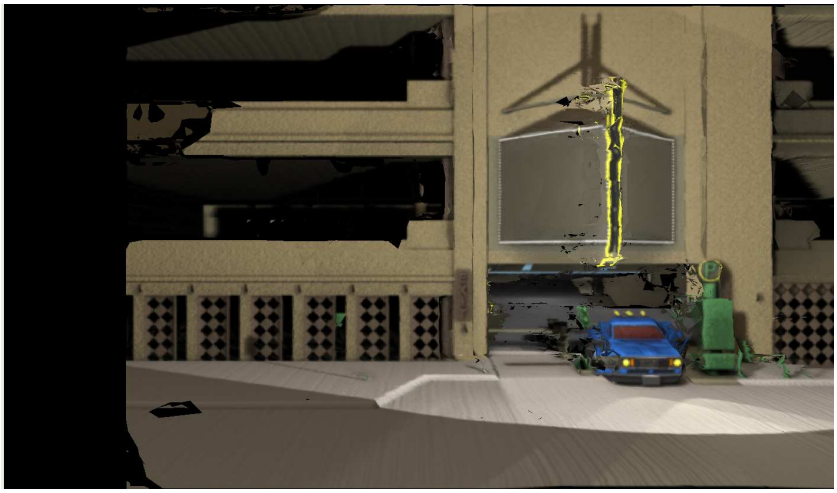


Bild 4.9: Ansicht des ausgedünnten Meshs, ähnlichste Textur zur Median Textur.

4.3 Ergebnisse nach Ausdünnung des Meshs

Durch die Ausdünnung des Meshs konnte die Anzahl der Dreiecke auf etwa ein Fünfzehntel reduziert werden, ohne dass die Oberflächenstruktur gravierend verändert wurde. Ein Problem war lediglich bei starken Tiefenunterschieden festzustellen. So konnte die Szene „Gesicht“ nicht zufriedenstellend rekonstruiert werden, da das aufgenommene Gesicht im Vergleich zum Hintergrund sehr klein ist und somit stärker ausgedünnt wurde als der Hintergrund. Die anderen Szenen ließen sich jedoch ohne größere Probleme ausdünnen.

In Bild 4.10 wurde die Anzahl der Vertices der Neon&Chrome Szene von 379432 auf 22752 und die Anzahl der Dreiecke von 745683 auf 42129 reduziert. Das untere Bild wurde mit der ähnlichsten Textur zur Median Textur texturiert. Auffällig ist die bessere Texturierung von Objekten, die erst in späteren Bildsequenzen sichtbar waren, wie z. B. der Schatten der Wand, links neben dem Auto. Ebenfalls wurden einige Verkehrsschilder (rechts neben dem Auto) rausgefiltert deren Tiefe falsch geschätzt wurde. Aufgrund des ungenauen Verdeckungstests war die erste Ansicht in Bild 4.9 an einigen Stellen mit großem Tiefenunterschied fehlerhaft.

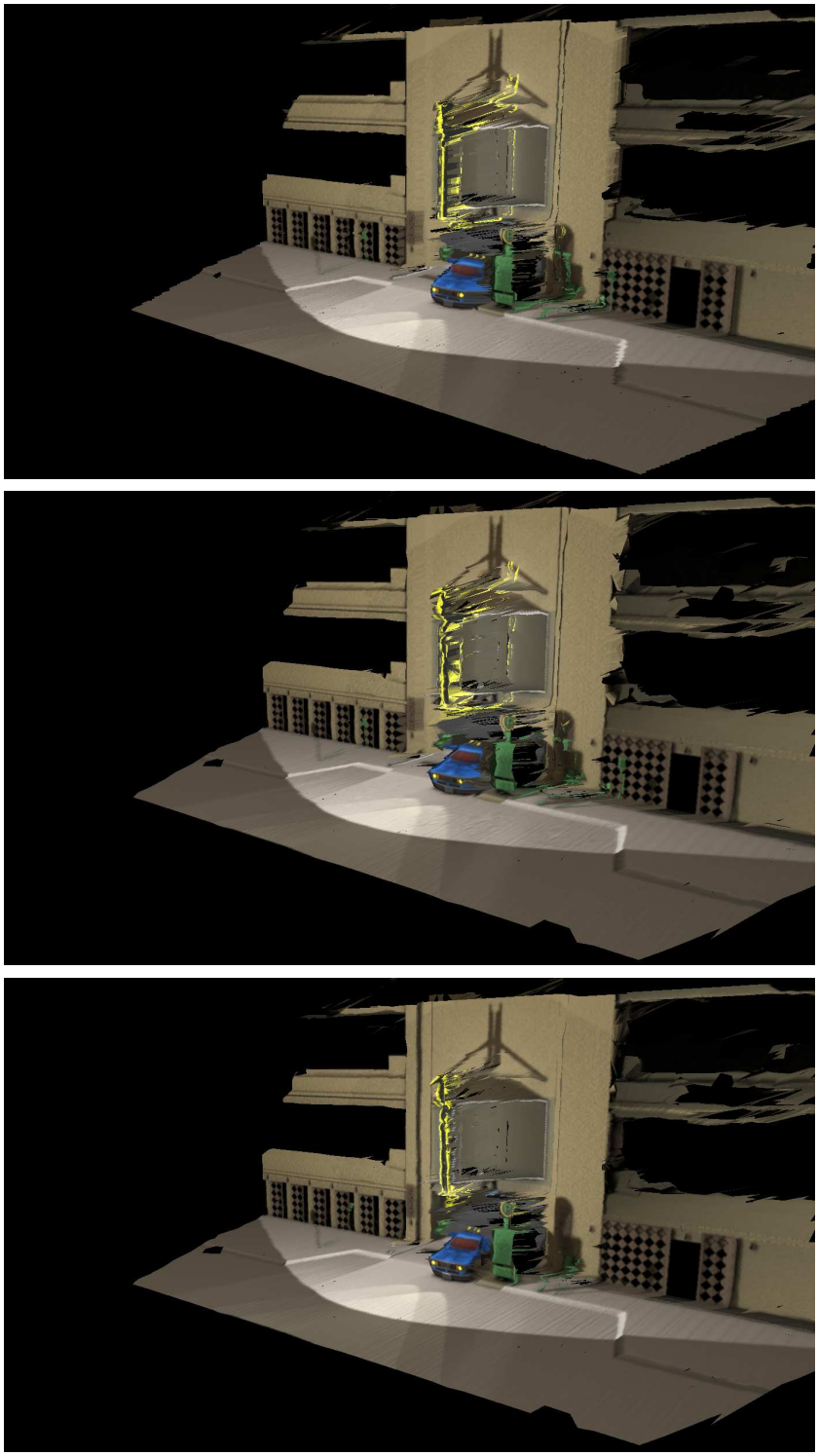


Bild 4.10: Oben: Das nicht ausgedünnte Mesh der Szene Neon&Chrome. Mitte: Das ausgedünnte Mesh ohne Median Textur. Unten: Das ausgedünnte Mesh der Szene Neon&Chrome, welches mit der ähnlichsten Textur zur Median Textur texturiert wurde.

	Zeitaufwand in ms	ms pro Stereobild	in %
Tiefenkarten	9678	744,46	11,5
Positionsbestimmung	14687	1129,77	17,4
Fusion der Tiefenkarten	25856	1988,92	30,7
Mesh Generierung	25040	1926,15	29,7
Ausdünnung des Meshs	237960	/	(*)
Median Textur	3090	/	(*)

Tabelle 4.1: Zeitaufwand zur Berechnung einer Rekonstruktion aus 13 Stereobildern der Robbie2 Szene. (*) Um die Zeiten der einzelnen Prozesse übersichtlicher zu halten, wurde die Zeit für die Ausdünnung des Meshs und die Texturierung mittels Median Textur in der Berechnung der Prozentangaben nicht berücksichtigt. Die Zeit zum Ausdünnen des Meshs beträgt mit nachträglicher Texturierung bei 13 Bildern 74% des kompletten Rekonstruktionsprozesses.

4.4 Zeitaufwand

Zum Berechnen des Zeitaufwands wurden die 13 Bilder der Robbie2 Szene verwendet. Als Testplattform dient ein $2,5$ *Ghz Intel Pentium Quad-Core Q6600* mit 2 GB DDR2 Arbeitsspeicher, bei dem allerdings aufgrund der Programmarchitektur nur ein Kern aktiv war. Das Betriebssystem ist *Ubuntu 8.04*.

Die Ausdünnung des Meshs nimmt, wie in Tabelle 4.1 zu sehen ist, den Großteil der Zeit für die Rekonstruktion in Anspruch. Die Zeit zum Ausdünnen hängt von der Anzahl der vorher triangulierten Dreiecke ab. Dadurch, dass bei der Generierung der Dreiecke Doppelte entfernt werden, bleibt deren Anzahl nahezu konstant. Die Anzahl der Dreiecke nimmt nur zu, falls neue noch nicht in einem anderen Bild erfassten Bereiche hinzugefügt werden. Hierdurch ist die Zeit zum Ausdünnen nicht von der Anzahl der Bilder abhängig, sondern davon, wieviele Bereiche in den Bildern neu sind.

Beim Erstellen der Tiefenkarten in Tabelle 4.2 nimmt die Suche nach Stereokorrespondenzen die meiste Zeit in Anspruch, während beim Schätzen der Position in Tabelle 4.3 die Suche nach korrespondierenden SIFT Deskriptoren die meiste Zeit in Anspruch nimmt.

	Zeitaufwand in ms	ms pro Stereobild	in %
Radiale Entzerrung	369	28,38	3,81
Rektifizierung	345	26,54	3,56
Stereo Korrespondenzen	8964	689,54	92,62

Tabelle 4.2: Zeitaufwand zur Berechnung der Disparitätskarten aus 13 Stereobildern der Robbie2 Szene. Die maximale Disparität war auf 150 Pixel eingestellt. Eine Reduktion dieses Wertes würde die Suche nach Stereo Korrespondenzen beschleunigen. Allerdings würden dadurch Objekte in der Nähe der Kamera falsch geschätzt. Die Wahl der maximalen Disparität ist sowohl von der Auflösung der Bilder, von der Entfernung der Objekte, wie auch von der Entfernung der Kameras zueinander abhängig.

	Zeitaufwand in ms	ms pro Stereobild	in %
SIFT Merkmale extrahieren	6093	468,69	41,49
Matching der Bildmerkmale	8554	658,00	58,28
ICP	40	3,07	0,03

Tabelle 4.3: Zeitaufwand zur Schätzung der Position der Robbie2 Szene. Der Anteil des ICP Algorithmus ist niedrig. Durch die bekannten Korrespondenzen und die geringe Anzahl an Bildmerkmalen waren durchschnittlich etwa zehn Iterationen nötig.

4.5 Validierung / Verifikation

Die Neon&Chrome Szene wurde mit *Autodesk Maya* erstellt. Hier sind die Modelldaten der Szene bekannt und können mit der Rekonstruktion verglichen werden. Da die rekonstruierten Vertices der Dreiecke mit sehr großer Wahrscheinlichkeit nicht mit denen der Neon&Chrome Szene übereinstimmen, wird dieser Vergleich lediglich auf die Tiefenwerte der Dreiecke durchgeführt. Die Tiefenkarten der Stereobilder konnten direkt in *Maya* gerendert werden.

Verglichen werden diese Ground-Truth Daten mit den geschätzten Disparitätskarten aus den Stereobildern sowie mit den fusionierten Tiefenkarten (siehe Kapitel 3.5.3). Das leichte Rauschen in den Ground-Truth Daten ist nicht beabsichtigt und stammt direkt aus dem *Maya*-Renderer. Die Konturen sind jedoch ausreichend scharf für einen Vergleich mit den Disparitätskarten.

Als Vergleichswerte wurden hier die Pixel gezählt, bei denen sich der Disparitätswert in der Disparitätskarte um mehr als 5 vom Ground-Truth Tiefenbild unterschieden hat. Ebenso wurde die Standardabweichung zum Ground-Truth errechnet. Da die Disparitätskarten am Rand nicht gut zu errechnen sind, wurden bei der Berechnung die ersten und letzten 30 Pixel auf der x-Achse ignoriert, ebenso wie Pixel, für die in einem der Bilder keine Disparität errechnet werden konnte, bzw. wo kein Tiefenwert im Ground-Truth vorhanden war.

Es sollte zusätzlich berücksichtigt werden, dass mit dem Vergleich der Tiefenkarte ebenfalls die Position mitgeschätzt wird. Die fusionierte Tiefenkarte ist ein 3-D-Modell aus Punkten, welche in eine geschätzte Kameraposition projiziert wird. Eine falsch geschätzte Position würde ebenfalls zu einer falschen Projektion der fusionierten Tiefenkarte führen.

Die Ergebnisse der Disparitätskarten aus Bild 4.11 wurden in Tabelle 4.4 zusammengefasst.

Vergleicht man nun diese Ergebnisse, so merkt man, dass sich die Disparitätskarten nicht gravierend von der fusionierten Tiefenkarte unterscheiden. Die fusionierte Karte nimmt die Tiefenwerte, die in den meisten Disparitätskarten vorkommen. Aus diesem Grund kann auch in einigen Fällen eine einfache Disparitätskarte besser sein, als eine fusionierte Kar-

Bild	Einfache Disparitätskarte		Fusionierte Disparitätskarte	
	σ	Ausreißer in %	σ	Ausreißer in %
1	8,47791	11,8176	9,98649	17,8026
2	7,22874	12,6807	8,21590	14,9900
3	8,72140	21,1668	6,32882	14,5609
4	7,91117	21,8093	6,26415	16,1099
5	7,91712	22,1705	6,03341	16,1141
6	12,48450	30,7220	6,67817	19,7719
1 bis 6	8,79014	20,0612	7,25116	16,5582

Tabelle 4.4: Errechnung der Standardabweichungen (zum Ground-Truth) und Anzahl der Tiefenwerte, die mehr als fünf Disparitätspunkte zum Ground-Truth abweichen. Es wurden Pixel mit der Disparität 0 (keine errechenbare Tiefe in den Pixeln) sowie die ersten und letzten 30 Pixel der x-Achse in den Bildern ignoriert. Die letzte Zeile der Tabelle ist die durchschnittliche Standardabweichung und die durchschnittliche Prozentzahl der Ausreißer von allen Tiefenkarten.

te, nämlich genau dann, wenn in den meisten Disparitätskarten ein falscher Tiefenwert häufiger vorkommt als ein Richtiger.

Vergleicht man die gemittelten Werte in der unteren Spalte von Tabelle 4.4, so sieht man, dass die fusionierte Tiefenkarte doch leicht besser ist, als die einzelnen Tiefenkarten.

Viel wichtiger in der fusionierten Tiefenkarte jedoch ist, dass sich die Fehler der einzelnen Tiefenkarten in der fusionierten Tiefenkarte nicht addiert haben. Ohne die Fusion der Tiefenkarten könnten falsche Werte bei der Mesh Generierung korrekt geschätzte Werte verdecken. Dadurch würde die Rekonstruktion sich mit jeder zusätzlichen Tiefenkarte verschlechtern. Tabelle 4.4 hat jedoch gezeigt, dass dies nicht der Fall ist und sich die Fehler sogar verringern.

Da zur Meshausdünnung eine bereits vorhandene Implementierung verwendet wurde, können hierzu die Ergebnisse des Memoryless-Simplification Algorithmus unter [LT98] nachgelesen werden.

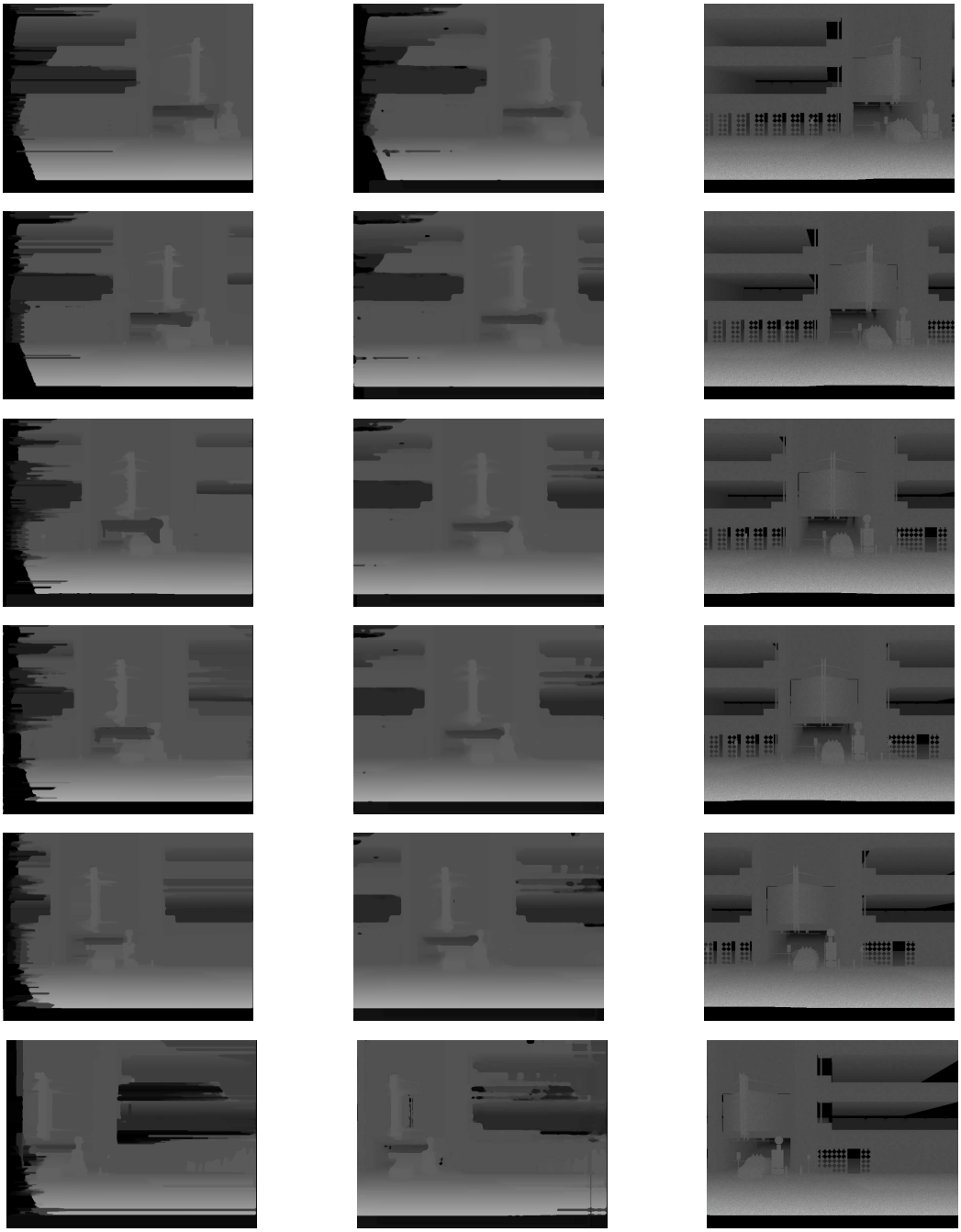


Bild 4.11: Die Tiefenbilder aus Tabelle 4.4. Links: Disparitätskarten aus *OpenCV*. Mitte: Die fusionierte Karte in verschiedene Ansichten projiziert und mit einem Median Filter geglättet. Rechts: Ground-Truth Disparitätskarten.

4.6 Grenzen des Verfahrens

Um aus einer Szene ein Mesh zu generieren, müssen mehrere Faktoren berücksichtigt werden:

- Die Distanz der Kameras muss der Szene angepasst sein. Ein zu großer Abstand führt dazu, dass nahe Objekte aufgrund der maximalen Disparität von 255 in der *OpenCV* Implementation nicht korrekt geschätzt werden können. Ein zu kleiner Abstand führt dazu, das weit entfernte Objekte eine zu kleine Disparität erhalten und keine Tiefe geschätzt werden kann.
- Die Szene sollte ausreichend texturiert sein, damit Korrespondenzen zwischen den Stereobildern zustande kommen.
- Die Stereobilder müssen absolut synchron aufgenommen werden, da ansonsten falsche Tiefenkarten geschätzt werden.
- Die Sequenz der Bilder muss sich großflächig überschneiden, um ähnliche SIFT Bildmerkmale in den Bildern zu finden.
- Können keine zuverlässigen Tiefenkarten geschätzt werden, ist auch keine zuverlässige Positionsschätzung möglich.
- Aufnahmen aus stark unterschiedlichen Positionen führen zu extrem verzerrten Dreiecken, da die Dreiecke aus der Sicht des zuerst sichtbaren Stereobildes rekonstruiert werden.
- Die Szene muss immer gleich beleuchtet sein. Beleuchtungsunterschiede führen zu einer ungleichmäßigen Texturierung.

Dies sind sehr viele Einschränkungen. Die Rekonstruktion einer beliebigen zufälligen Szene ist schwierig und schlägt oft fehl. Werden aber alle diese Faktoren berücksichtigt, so ist eine zuverlässige Rekonstruktion des Meshs möglich.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

In dieser Diplomarbeit wurde ein Verfahren zur Generierung eines texturierten 3-D Meshs aus Stereobildsequenzen entwickelt.

Der eigentliche Schwerpunkt der Arbeit, die Ausdünnung des Meshs wurde verschoben, da die Ergebnisse der Tiefenkarten keine konsistente Mesh Generierung zugelassen hätten. Der Schwerpunkt liegt somit bei der Fusionierung der Tiefenkarten und bei der Erstellung eines texturierten Meshs aus Tiefenkarten.

Die Bilder für dieses Verfahren müssen aus einem kalibrierten Stereokamerasystem stammen. Um die Kameras zu kalibrieren wurden Methoden aus *OpenCV* verwendet. Die Kalibrierung errechnet Daten, welche für die radiale Entzerrung nach Zhang und die Rektifizierung mit einer Homographiematrix benötigt werden. Die Kalibrierung sowie die Rektifizierung werden mit dem *CvCalibFilter* aus *OpenCV* durchgeführt.

Aus den rektifizierten Bildern wird mit dem Birchfield Algorithmus aus *OpenCV* für jedes Stereobildpaar eine Disparitätskarte erstellt, die mit einer Modifikation des *CvCalibFilters* entrektifiziert wird, um daraus in weiteren Schritten die 3-D Koordinaten der Disparitätskarte zu errechnen.

Um die 3-D Koordinaten von weiteren Stereobildern zu registrieren, werden aus dem lin-

ken Bild der radial entzerrten Stereobilder (welches das Ursprungsbild der Disparitätskarte ist) SIFT Merkmale mit der Implementierung von Lowe extrahiert. Die Korrespondenzen der SIFT Merkmale werden mit dem Nearest Neighbour Verfahren von Peter Decker [Dec06] gefunden. Mit den 3-D Koordinaten an den SIFT Merkmalen, welche aus der Disparitätskarte errechnet wurden, wird dann mit einer selbst entwickelten Variante des ICP Algorithmus die Kameraposition errechnet. Outlier werden effizient durch den ICP Algorithmus entfernt. Die 3-D Position von bisher gefundenen Bildmerkmalen wird zusätzlich mit deren Mittelwert gefiltert, um den inkrementellen Fehler bei der Positionsbestimmung möglichst gering zu halten. Der Algorithmus basiert nicht auf Zufallsgeneratoren und führt daher garantiert bei gleichen Eingangsdaten zu einer gleichen Ausführungszeit und exakt den selben Ergebnissen.

Mit der errechneten Kameraposition können die 3-D Koordinaten der Disparitätskarten ins Weltkoordinatensystem eingetragen werden, wodurch eine 3-D Punktewolke erzeugt wird. Um redundante Daten zu reduzieren und gleichzeitig Rückschlüsse auf falsch geschätzte Tiefenwerte zu erhalten, wird eine selbst entwickelter Bayesfilter verwendet, welcher als starke Vereinfachung des Confidence-Based-Fusion Algorithmus [MAW$^+$07] angesehen werden kann. Die Modifikation ermöglicht die Fusionierung der Tiefenkarten ohne vorher errechnete Konfidenz der Disparitätskarten, die Ignorierung von Okklusion Konflikten und eine effiziente Implementierung auf der CPU durch eine geringe Anzahl an Projektionen der 3-D Punkte in die Tiefenkarte.

Für die Generierung des Meshs wurde eine sehr einfache Methode gewählt. Aus den Ansichten der fusionierten Tiefenkarte wird aus den vorher errechneten Kamerapositionen jeweils ein Dreiecksnetz in der Bildebene erstellt und in die 3-D Weltkoordinaten projiziert. Doppelte Dreiecke, sowie solche, die aus der nicht eindeutigen Delaunay Triangulierung die selben Dreiecke darstellen, werden wieder gelöscht. Für die Delaunay Triangulierung wurde die *Triangle* Bibliothek verwendet. Die Reduktion von Rubber-Sheet Artefakten wird erreicht, indem Dreiecke mit großen Tiefenunterschieden wieder aus dem Mesh entfernt werden.

Gleichzeitig werden bei der Meshgenerierung Texturkoordinaten ermittelt und festgestellt, wann ein Dreieck zuerst in einer Ansicht sichtbar ist. Die Texturierung erfolgt aus dieser ermittelten Ansicht und den errechneten Texturkoordinaten mittels Texture-Mapping in

OpenGL.

Um die Anzahl der Dreiecke zu reduzieren wurde die Implementierung des Memoryless-Simplification Algorithmus aus der *GNU Triangulate Surface Library* verwendet. Die Anzahl der Dreiecke kann somit auf einen Bruchteil reduziert werden, ohne dass gravierende Fehler im Mesh entstehen.

Weiter wurde eine Methode implementiert, die eine Median Textur teilweise auf der GPU errechnet und aus den Texturen die nächste zum Median heraussucht. Hierdurch werden sich bewegende Objekte, leichte Spiegelungen und Objekte mit falsch geschätztem Tiefenwert entfernt.

Um das gesamte Verfahren zu testen wurde die Neon&Chrome *Maya* Szene aus [Dec06] bearbeitet, indem Texturen, Bump-Maps und Lichtquellen hinzugefügt wurden. Diese sollten die Szene realistischer machen und das Erstellen von Tiefenkarten mit dem Birchfield Algorithmus erleichtern, da dieser auf Texturen angewiesen ist. Aus der *Maya* Szene wurden dann Stereobilder und deren Ground-Truth Tiefenkarten gerendert. Mit diesen Stereobildern wurde ein Modell mit dem Rekonstruktionsalgorithmus dieser Diplomarbeit erstellt. Verglichen wurden die Ground-Truth Tiefenkarten mit Disparitätskarten aus *OpenCV* sowie mit der fusionierten Tiefenkarte, die in die entsprechenden Ansichten projiziert wurde..

Die Ergebnisse haben gezeigt, dass durch die Fusion der Tiefenkarten der Fehler in der Tiefenschätzung reduziert wurde und nur durchschnittlich 16% um mehr als 5 Tiefenwerte abweichen.

5.2 Ausblick

Das hier gewählte Verfahren ist sehr einfach gehalten und darauf ausgerichtet, komplizierte und unübersichtliche Algorithmen zu vermeiden. Die Ergebnisse sind dementsprechend nicht optimal. Was optisch beim rekonstruierten Modell zuerst ins Auge fällt, sind stark verzerrende Dreiecke, die durch eine nicht optimale Wahl der Ansicht bei der Meshgenerierung hervorgerufen werden. Das Verwenden von Masken, welche die Rekonstruktion von Flächen aus der optimalen Ansicht ermöglicht, könnte dieses Problem reduzieren.

Es könnten auch weitere Tiefenkarten errechnet werden, indem aus den bekannten Kamerapositionen der Einzelbilder weitere Stereobildpaare mit unterschiedlichen Basislinien gewählt werden. Ebenso wäre eine Erweiterung des Fusionsalgorithmus um eine Konfidenz möglich, die sich aus der visuellen Konsistenz der texturierten Tiefenkarte zum korrespondierenden Eingangsbilder oder aus der Gewichtung der Stereokorrespondenzen ergibt.

Die Geschwindigkeit wurde ebenfalls in diesem Verfahren kaum optimiert. Eine Geschwindigkeitssteigerung würde man durch Auslagerung der Algorithmen auf die GPU erreichen, bzw. indem man bereits vorhandene Implementierungen der Algorithmen auf der GPU verwendet, wie dies z. B. beim SIFT-Algorithmus möglich ist. Die Korrespondenzsuche der SIFT Merkmale könnte ebenfalls sehr viel schneller durch einen BBF-Algorithmus erfolgen [BL97].

Anhang A

Mathematische Bezeichner und Symbole

Im Folgenden werden die verwendeten Variablen und Bezeichner aufgelistet und kurz erläutert.

Variable/Bezeichner	Bedeutung
d	Disparität zweier Korrespondierender Pixel
p_x^p	X-Koordinate von Punkt p (linkes Bild) in Pixelkoordinaten
q_x^p	X-Koordinate von Punkt q (rechtes Bild) in Pixelkoordinaten
p_z^w	Z-Koordinate von Punkt p (linkes Bild) in Weltkoordinaten
F	Fokus der Kamera (Tiefenkarten)
b	Abstand zweier Kameras
F	Fundamental Matrix
$\mathrm{E}(R,t)$	Fehlerfunktion, die für den ICP Algorithmus minimiert wird
	Fortgesetzt auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Variable/Bezeichner	Bedeutung
R	Rotationsmatrix
t	Translationsvektor
M	Menge von Punkten in einem Modell (Modellmenge)
D	Menge von Punkten in einer Messung (Datenmenge)
m_i	Ein Punkt der Modellmenge M
d_i	Ein Punkt der Datenmenge D
c_m	Zentroid von allen $\mathsf{m}_\mathsf{i} \in M$
c_d	Zentroid von allen $\mathsf{d}_\mathsf{i} \in D$
H	Korrelationsmatrix (ICP)
$\hat{F}(x)$	Schätzung eines 3-D Punktes in der Referenzansicht
D_t	Tiefenkarte, aus einer Disparitätskarte errechnet
D_{ref}	Tiefenkarte, in die Referenzansicht projiziert
$P_t(x)$	Bildkoordinaten des 3-D Punktes x, projeziert nach Ansicht t
$D_t(x)$	Vereinfacht, entspricht $D_t(P_t(x))$, also dem Tiefenwert eines Punktes aus einer Disparitätskarte
$R_t(x)$	Die Distanz zwischen Punkt x und dem Kamerazentrum
$\hat{f}(x)$	Entspricht $R_{ref}(\hat{F}(x))$, also der Distanz des geschätzten Tiefenwertes für die Referenzansicht
$O(x_t)$	Tiefenwert, gerendert aus Ansicht t
$S(x)$	Stabilität eines Punktes
$N(x)$	Alternativer Tiefenwert eines Punktes
$C_i(x)$	Konfidenz eines Tiefenwertes aus einer Tiefenkarte
$\hat{C}(x)$	Geschätzte Konfidenz eines Punktes
z_i	z-Koordinate eines Punktes
t	(Rubber Sheets threshold) Schwellwert

Tabelle A.1: Übersicht über die verwendeten Variablen und Bezeichner.

Anhang B

Implementationsdetails

Der Quellcode ist in verschiedene Bereiche aufgeteilt. Im Ordner Programmcode finden sich folgende Ordner:

```
\Algorithms
\DepthMapper
\PoseEstimator
\SiftLowe
\MeshGenerator
\Triangle
\PumaGrabber

\Calibration
\ImageStereoSaver
\Stereo3DViewer
```

Die ersten 6 Ordner erzeugen jeweils eine Bibliothek, die letzten 3 Ordner beinhalten Programme, die diese Bibliotheken nutzen.

Die `Algorithms` Bibliothek beinhaltet diverse statische Funktionen und *typedefs* die von mehreren anderen Bibliotheken benutzt werden, wie z. B. die Transformation einer Punktemenge mit einer Matrix.

Die `DepthMapper` Bibliothek ist zuständig für Rektifikation, Linsenentzerrung, Erzeugung von Tiefenkarten und Umrechnung von Tiefenkarten in 3-D Koordinaten.

In `PoseEstimator` wird die Position mittels ICP geschätzt. Hierfür werden Tiefenkarten aus dem `DepthMapper` und SIFT Merkmale aus der `SiftLowe` Bibliothek verwendet.

Die `SiftLowe` Bibliothek enthält den kaum modifizierten Code aus [Dec06] zum Extrahieren und Matchen von SIFT Merkmalen. Die Nearest Neighbour Methode zum Vergleich zweier Deskriptoren wurde von Absoluter Distanz auf Euklidische Distanz verändert, einige Strukturen wurden verändert um die Dateien unter *Windows* kompilierbar zu machen und es wurde ein *QMake* Projekt erstellt, welches die nötigen Klassen zu einer Bibliothek kompiliert.

`MeshGenerator` beinhaltet den Bayes Filter zur Fusion mehrerer Tiefenkarten und Funktionen zum Erstellen und Zusammensetzen von Dreiecksnetzen.

Im `Triangle` Ordner ist die *Triangle* Bibliothek zur Delaunay Triangulation aus 2-D Punktmengen enthalten. Hier wurde das Makefile durch ein *QMake* Projekt ersetzt.

Im `PumaGrabber` sind Funktionen zum Einlesen von Bildern aus Dateien und zur Aufnahme von Bildern von Firewire Stereo Kameras enthalten. Die Ansteuerung der Stereo Kameras wurde aus *Puma 1* und Anfängen des *Robbie* Projekts entnommen, welche die *libdc1394* benutzen. Diese *libdc1394* ist allerdings nicht für *Microsoft Windows* verfügbar. Aus diesem Grund kompiliert die Bibliothek unter *Windows* automatisch ohne Firewire Ansteuerung und kann Bilder nur aus Dateien einlesen.

B.1 Stereo3DViewer

Im Ordner `Stereo3DViewer` befindet sich der Quellcode des Hauptprogramms dieser Diplomarbeit. Die ausführbare Datei `mainApp` befindet sich im Ordner `Programmcode`. Das Programm liest Bilder aus dem Unterverzeichnis `Stereo3DViewer/Tmp` ein, bzw. aus dem Verzeichnis mit dem es aufgerufen wurde (`./mainApp $Verzeichnis$/`). Es entzerrt und rektifiziert die Bilder anhand den Kalibrierungsinformationen aus der Datei `cameras.txt`, welche sich im selben Ordner wie dei Bilder befindet. Aus diesen Bildern wird die Tiefeninformation gewonnen, in 3-D Koordinaten umgerechnet und mit den Bildinformationen eingefärbt.

In der Mitte befindet sich ein *OpenGL* Fenster, welches eingefärbte 3-D Punkte anzeigt. Mit der Maus und den Pfeiltasten der Tastatur lässt sich dieses Modell bewegen. Die linke Maustaste rotiert die Ansicht, die rechte Maustaste zoomt die Kamera heran, genau wie das Mausrad. Die Pfeiltasten dienen der Translation und mit der Leertaste lässt sich das Modell wieder auf die Anfangsansicht zurücksetzen. An der rechten Seite befindet sich eine Kontrollleiste mit mehreren Buttons. `Next Image` lädt das nächste Stereobild, schätzt die neue Position und fügt die 3-D Punkte ins Weltkoordinatensystem. `Create Mesh` erzeugt aus den fusionierten Punktewolken ein 3-D Dreiecksmesh mit Textur. Aus dieser lässt sich dann mittels `Thinout Mesh` ein reduziertes Mesh erstellen. Dieser Vorgang kann jedoch je nach Modell mehrere Minuten dauern und lässt sich nur bei installierter *GNU Triangulated Surface Library* ausführen. `Select Texture` errechnet (teilweise auf der Grafikkarte) eine Median Textur für das ausgedünnte Mesh und weist einem Dreieck die Textur, mit dem geringsten Abstand zur Median Textur zu.

Im unteren `View` Bereich lassen sich weitere Einstellmöglichkeiten zur Ansicht vornehmen. Mit `Show Features` lässt sich auswählen, ob die Bildmerkmale sichtbar sind und mit `Toggle View Depth Map` lässt sich die Ansicht von Textur auf Tiefenwerte umschalten.

B.2 ImageStereoSaver

Das Unterprogramm `ImageStereoSaver` dient der einfachen Aufnahme von Stereobildern. Es verwendet lediglich die `PumaGrabber` Bibliothek. Mit `./saveImages 1000` wird von der angeschlossenen Kamera jede Sekunde ein Stereobild aufgenommen und angezeigt. Mit `./saveImages` (ohne Parameter) wird bei beliebigen Tastendruck jeweils ein Stereobild aufgenommen. Dieses Programm ist unter *Windows* aufgrund der fehlenden *libdc1394* nicht funktionstüchtig.

B.3 Calibration

Mit dem `Calibration` Unterprogramm lässt sich aus den mit Kalibriermuster aufgenommenen Bildern eine Kalibrierungsdatei erstellen. Die Bilder müssen im Unterordner `Tmp` liegen. Ist eine Kamera angeschlossen, werden direkt die Bilder aus der Kamera benutzt und im Ordner `Tmp` gespeichert. Aus den aufgenommenen Bildern wird eine mit dem *OpenCV CvCalibFilter* kompatible Datei (`cameras.txt`) erstellt, welche auch die Linsenkrümmung und die Homographie Matrix für die Rektifikation beider Kameras enthält.

B.4 Benötigte Bibliotheken, Abhängigkeiten und Systemvorraussetzungen

Als *build*-Tool wurde das plattformübergreifende `QMake` in der Version `2.01a`, als Bestandteil von `Qt 4.3.4`, verwendet.

Die Oberfläche wurde ebenfalls mit *Qt* erstellt. Als Bildverarbeitungsbibliothek wird `OpenCV 1.0.0`, zum Ansteuern der Kameras die `libdc1394` und zum Ausdünnen des Meshs wird die `GNU Triangulated Surface Library` (libgts) verwendet.

Der Quelltext wurde sowohl unter *Linux* (*Ubuntu 8.04 + gcc 4.2.3*) als auch unter *Windows* (*Vista Professional + Visual Studio 2008 + technical release 1 patch*) kompiliert und ge-

testet. Den vollen Funktionsumfang und eine komfortable Geschwindigkeit erreicht man allerdings nur unter *Linux*. Unter *Windows* fehlen sowohl die Funktionen zum Ansteuern der Stereokameras, als auch die Unterstützung für die *GNU Triangulated Surface Library* (welche aber nachträglich noch hinzugefügt werden könnte). Die Geschwindigkeit ist unter *Windows* um ein vielfaches geringer, was mit der Verwendung von *Hashmaps* aus dem *C++ Technical Release 1* oder den Zugriffen auf Elemente eines *STL Vectors* zusammenhängen könnte, die beide unter *Windows* anders implementiert sind.

Durch die Verwendung von *OpenGL* zur Ansicht des 3-D Meshs und der Textur ist unter *Linux* eine hardwarebeschleunigte Grafikkarte nötig.

Anhang C

Aufbau der CD

Der Ausarbeitung liegt eine CD bei, auf der die wichtigsten Quelldaten gespeichert sind. Der Aufbau ist wie folgt:

```
Ausarbeitung/
  Arbeit/

Programmcode/
  Stereo3DViewer/
  Calibration/
  ImageStereoSaver/

Sonstiges/
  NeonChrome/
  Videos/
  Images/
```

Im Verzeichnis `Ausarbeitung` befindet sich dieses Dokument. In den Unterverzeichnissen `Arbeit` liegen die LaTeX-Quelldateien zu diesem Dokument.

Die Quelldateien für die Programme liegen unter `Programmcode`. Mit dem Befehl `qmake && make` lassen sich die Programme kompilieren. Mit

`qmake && make release` wird das Programm mit Kompiler Optimierung übersetzt und läuft wesentlich schneller. Unter *Windows* wird in der *Visual Studio Konsole* der Befehl `nmake` anstelle von `make` verwendet. Man beachte, dass die benötigten Bibliotheken aus Anhang B.4 installiert sind. Das Hauptprogramm befindet sich im Verzeichnis `Stereo3DViewer`. Die Unterprogramme `Calibration` und `ImageStereoSaver` befinden sich ebenfalls im Ordner `Programmcode`.

Unter `Sonstiges` ist die `Neon&Chrome` *Maya* Datei, unter `Images` befinden sich die Stereobilder aus Kapitel 4 und unter `Videos` befinden sich Videos von den erstellten Rekonstruktionen im *Theora* Format.

Literaturverzeichnis

[AFM+06] AKBARZADEH, A. ; FRAHM, JM ; MORDOHAI, P. ; CLIPP, B. ; ENGELS, C. ; GALLUP, D. ; MERRELL, P. ; PHELPS, M. ; SINHA, S. ; TALTON, B. u. a.: Towards Urban 3D Reconstruction From Video. In: *Proc. 3DPVT* Bd. 4, 2006

[BBH08] BRADLEY, Derek ; BOUBEKEUR, Tamy ; HEIDRICH, Wolfgang: Accurate Multi-View Reconstruction Using Robust Binocular Stereo and Surface Meshing. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2008). `http://www.cs.ubc.ca/labs/imager/tr/2008/Bradley_AccurateMultiView_CVPR/cvpr08.pdf`

[BL97] BEIS, J. ; LOWE, David G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In: *Conference on Computer Vision and Pattern Recognition*. Puerto Rico, 1997, S. 1000–1006

[Bou02] BOUGUET, Jean-Yves: *Pyramidal Implementation of the Lucas Kanade Feature Tracker*. 2002

[BT98] BIRCHFIELD, Stan ; TOMASI, Carlo: Depth Discontinuities by Pixel-to-Pixel Stereo. In: *ICCV*, 1998, S. 1073–1080

[BVZ01] BOYKOV, Yuri ; VEKSLER, Olga ; ZABIH, Ramin: Fast Approximate Energy Minimization via Graph Cuts. In: *IEEE* 23 (2001), Nr. 11, 1222-1239. `http://www.cs.cornell.edu/rdz/Papers/BVZ-pami01-final.pdf`

[CL96] CURLESS, Brian ; LEVOY, Marc: A volumetric method for building complex models from range images. In: *Proc. SIGGRAPH* Bd. 96, 1996, S. 303–312

[Dec06] DECKER, Peter: *Erstellung einer Tiefenkarte aus Stereobildern für den RoboCup Rescue Wettbewerb*, Universität Koblenz-Landau, Studienarbeit, 2006

[Dec07] DECKER, Peter: *Bildbasierte Bewegungsschätzung aus Kamerafahrten anhand prägnanter Merkmale*, Universität Koblenz-Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualistik, Diplomarbeit, 9 2007. `http://www.uni-koblenz.de/~agas/DocsPubl/Decker2007BBA.pdf`

[Del08] *Delaunay Triangulation* . `http://commons.wikimedia.org/wiki/Image:Delaunay_Triangulation_(25_Points).svg`. Version: 2008

[FB81] FISCHLER, Martin A. ; BOLLES, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM, http://cacm.acm.org/* 24 (1981), Nr. 6, 381-395. `file:///lab/as/Docs/Articles/Fischler1981RSC.pdf`. ISBN 0001–0782

[FTV97] FUSIELLO, A. ; TRUCCO, E. ; VERRI, A.: Rectification with unconstrained stereo geometry. In: *British Machine Vision Conference*, 1997, 400-409

[GCS06] GOESELE, Michael ; CURLESS, Brian ; SEITZ, Steven M.: Multi-view stereo revisited. In: *CVPR* Bd. 2, 2006, S. 2402–2409

[Har97] HARTLEY, Richard I.: In Defense of the Eight-Point Algorithm. In: *Pattern Analysis and Machine Intelligence* 19 (1997), 6, Nr. 6, S. 580–593

[HIG02] HIRSCHMÜLLER, Heiko ; INNOCENT, P.R. ; GARIBALDI, J.: Real-Time Correlation-Based Stereo Vision with Reduced Border Errors. In: *International Journal of Computer Vision* 47 (2002), Nr. 1, S. 229–246

[KL81] KANADE, Takeo ; LUCAS, Bruce: An Iterative Image Registration Technique with an Application to Stereo Vision. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, 1981

[Low04] LOWE, David G.: Distinctive Image Features from Scale-Invariant Keypoints. In: *International Journal of Computer Vision* 60 (2004), Nr. 2, 91-110. `http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf`

[LT98] LINDSTROM, P. ; TURK, G.: Fast and memory efficient polygonal simplification. In: *Visualization'98. Proceedings*, 1998, S. 279–286

[MAW+07] MERRELL, P. ; AKBARZADEH, A. ; WANG, L. ; MORDOHAI, P. ; FRAHM, J.M. ; YANG, R. ; NISTÉR, David ; POLLEFEYS, M.: Real-Time Visibility-Based Fusion of Depth Maps. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, 2007, S. 1–8

[MFA+07] MORDOHAI, P. ; FRAHM, J. m. ; AKBARZADEH, A. ; ENGELS, C. ; GALLUP, D. ; MERRELL, P. ; SALMI, C. ; SINHA, S. ; TALTON, B. ; WANG, L. ; YANG, Q. ; STEWÉNIUS, H. ; TOWLES, H. ; WELCH, Greg ; YANG, R. ; POLLEFEYS, M. ; NISTÉR, David: Real-Time Video-Based Reconstruction of Urban Environments. In: *Proceedings of 3DARCH: 3D Virtual Reconstruction and Visualization of Complex Architectures*, 2007

[NLH07] NÜCHTER, Andreas ; LINGEMANN, Kai ; HERTZBERG, Joachim: Cached kd tree search for ICP algorithms. In: *Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007)-Volume 00* IEEE Computer Society Washington, DC, USA, 2007, S. 419–426

[NS01] NISTÉR, David ; (STOCKHOLM, Institutionen för numerisk analys och d.: *Automatic dense reconstruction from uncalibrated video sequences*. Tekniska högsk., 2001

[Nüc06] NÜCHTER, Andreas: *Semantische dreidimensionale Karten für autonome mobile Roboter*, Rheinische Friedrich-Wilhelms-Universität Bonn, Diss., 2006. `https://www.uni-koblenz.de/~agas/Pool/Nuechter2006SDK.pdf`

[Paj02] PAJAROLA, R.: *Overview of Quadtree-based Terrain Triangulation and Visualization*. Dept. of Information & Computer Science, University of California, Irvine, 2002

[PGV+04] POLLEFEYS, Marc ; GOOL, Luc van ; VERGAUWEN, Marten ; VERBIEST, Frank ; CORNELIS, Kurt ; TOPS, Jan ; KOCH, Reinhard: Visual modeling with a hand-held camera. In: *International Journal of Computer Vision 59(3)*, 2004, S. 207–232

[PKG99] POLLEFEYS, Marc ; KOCH, Reinhard ; GOOL, Luc J. V.: A Simple and Efficient Rectification Method for General Motion. In: *ICCV*, 1999, 496–501

[PNF+08] POLLEFEYS, M. ; NISTÉR, David ; FRAHM, J.M. ; AKBARZADEH, A. ; MORDOHAI, P. ; CLIPP, B. ; ENGELS, C. ; GALLUP, D. ; KIM, S.J. ; MERRELL, P. u. a.: Detailed Real-Time Urban 3D Reconstruction from Video. In: *International Journal of Computer Vision* 78 (2008), Nr. 2, S. 143–167

[PSM04] PAJAROLA, R. ; SAINZ, M. ; MENG, Y.: DMESH: FAST DEPTH-IMAGE MESHING AND WARPING. In: *International Journal of Image and Graphics* 4 (2004), Nr. 4, S. 653–681

[Ran08] *RANSAC*. `http://en.wikipedia.org/wiki/RANSAC`. Version: 2008

[RL01] RUSINKIEWICZ, Szymon ; LEVOY, Marc: Efficient Variants of the ICP Algorithm. In: *3rd International Conference on 3D Digital Imaging and Modeling (3DIM)*. Quebec City, Canada : IEEE Computer Society, 5 2001, 145-152

[SCD+06] SEITZ, S.M. ; CURLESS, Brian ; DIEBEL, J. ; SCHARSTEIN, D. ; SZELISKI, Richard: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: *Int. Conf. on Computer Vision and Pattern Recognition*, 2006, S. 519–528

[SIF08] *SIFT Features* . `http://en.wikipedia.org/wiki/Scale-invariant_feature_transform`. Version: 2008

[SS02] SCHARSTEIN, Daniel ; SZELISKI, Richard: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. In: *International Journal of Computer Vision* 47 (2002), Nr. 1-3, 7-42. `http://cat.middlebury.edu/stereo/taxonomy-IJCV.pdf`

[ST94] SHI, Jianbo ; TOMASI, Carlo: Good Features to Track. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*. Seattle, 6 1994, 593-600

[WB01] WELCH, Greg ; BISHOP, Gary: An Introduction to the Kalman Filter / University of North Carolina at Chapel Hill. Version: 2001. `http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf`. 2001. – Forschungsbericht

[Zha92] ZHANG, Z. Y.: Iterative Point Matching for Registration of Free-Form Curves. In: *INRIA*, 1992

[Zha00] ZHANG, Zhengyou: A flexible new technique for camera calibration. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 11, 1330-1334. `http://research.microsoft.com/~zhang/calib/`