Multi-Stereorekonstruktion nicht-lambert'scher Oberflächen mittels Graph-Cut Optimierung

Art Tevs
Universität des Saarlandes,
Saarbrücken, Deutschland

Bachelorthesis zur Erlangung des Grades
Bachelor of Science (B.Sc.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes
Eingereicht am 23. Februar 2006 in Saarbrücken.

Betreuender Hochschullehrer - Supervisor

Prof. Dr.-Ing. Marcus Magnor, MPI für Informatik, Saarbrücken, Deutschland

Gutachter - Reviewers

Prof. Dr.-Ing. Marcus Magnor, MPI für Informatik, Saarbrücken, Deutschland

Prof. Dr. Joachim Weickert, Fakultät der Mathematik und Informatik, Saarbrücken, Deutschland

Autor:	Art Tevs
Matrikelnummer:	2500415
Anschrift:	Breslauer Str. 18
	66121 Saarbrücken

Eidesstattliche Erklärung zur Bachelorarbeit

Name: _____

Vorname: _____

Ich versichere, die Bachelorarbeit selbständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben.

Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Saarbrücken, den 23.02.2006 _____

(Unterschrift)

Kurzfassung

In den letzten Jahren hat die Bestimmung der Tiefeninformationen eines Bildes sich zu einem großen Forschungszweig in der Computervision entwickelt. Viele der heutigen Algorithmen versuchen anhand zweier oder mehr Bilder jedem Pixel eine bestimmte Tiefe zuzuweisen. Die Voraussetzung dafür ist oftmals, dass die Werte korrespondierender Pixel sich nicht wesentlich unterscheiden.

In dieser Arbeit versuchen wir, anhand mehrerer gegebener Bilder die Tiefeninformationen zu rekonstruieren, ohne dabei bestimmte Voraussetzungen, wie z.B. eine diffuse Oberfläche, an die zu rekonstruierenden Objekte zu stellen. Dabei postulieren wir das Problem als ein Energieminimierungsproblem und benutzen das relativ junge Konzept der Graph-Cuts. Wir lösen einen Teil der Berechnung auf der GPU, was einen signifikanten Leistungsschub nach sich zieht. Am Schluss führen wir die Tiefenrekonstruktion auch auf “realen” Bildern aus, um die Robustheit des Algorithmus in der realen Umgebung zu verdeutlichen.

Abstract

In the last years computer vision problems like dense depth map reconstruction from given images became more and more important. There exists a lot of algorithms that are trying to assign a depth value to each pixel of the input image. Most of them have a strong assumption on the pixels of the object e.g. the color value constancy.

In this work we are trying to create a dense depth map for each input image without setting any strong assumptions, like diffuse surface, on the objects. We postulate that problem as energy minimization problem and try to solve it by using graph-cut approach. We also utilize the GPU to compute some parts of the energy function that does increase the whole computation speed immensely. At the end we show the stability of our algorithm by computing the depth maps on “real world” images.

INHALTSVERZEICHNIS

EINLEITUNG

In einem Buch steht geschrieben: “Am Anfang war das Licht.” Die Menschen haben von Beginn an festgestellt, dass das Licht für unsere Wahrnehmung unentbehrlich ist. Das Licht bzw. die Lichtteilchen verlassen die Lichtquelle, entfernen sich mit ungeheurer Geschwindigkeit ehe sie von Objekten reflektiert oder absorbiert werden. Die Lichtteilchen, die unsere Augen erreichen, projizieren eine Welt voller Farben auf unsere Netzhaut. Das Gehirn ist in der Lage, anhand der so entstandenen Bilder die Umgebung in unserem Geiste zu rekonstruieren. Dabei spielt die Tiefenwahrnehmung, die uns von Natur aus gegeben ist, eine wichtige Rolle. Wie oft müssen wir beispielsweise im Alltag die Entfernung eines herankommenden Autos abschätzen, um sicher die Straße zu überqueren. Die Fotografen benutzen Kameras, um die Welt in all ihrer Vielfalt festzuhalten. Dabei geht das Licht den gleichen Weg, wie es in unsere Augen gelangt. Das ankommende Licht projiziert die Umgebung auf eine flache Ebene, den Film in der Kamera.

Durch diese Projektion geht die Tiefe der abgebildeten Objekte verloren. Um das Problem zu lösen hat die Natur uns zwei Augen gegeben. Dadurch ist das menschliche Gehirn in der Lage, die Tiefe der wahrgenommenen Objekte zu berechnen. In der Forschung beschäftigt sich seit einiger Zeit ein Gebiet damit, den Computern das menschliche Sehen beizubringen. Ein Bereich der Computervision arbeitet an Algorithmen und Verfahren, die anhand gegebener Bilder die Tiefe der Objekte rekonstruieren. Dabei entstanden sehr viele und vor allem erfolgversprechende Ansätze. Viele der Algorithmen, wie in [SvMG04] vorgestellt, werden heute in der Robotik (siehe Abb. 1.1) und seit kurzem auch in der Autoindustrie eingesetzt, damit die autonom handelnden Maschinen entsprechend ihrer Umgebung auf die Umwelt reagieren können.

Diese Algorithmen arbeiten mit Stereo-Bildern und berechnen die Tiefe meistens in Echtzeit. Doch sie haben auch Nachteile: viele der Algorithmen haben Schwierigkeiten, stark glänzenden Objekten entsprechende Tiefe zuzuweisen. Der Grund dafür ist, dass der gleiche Punkt in beiden Bildern verschiedene Farbwerte aufweist, da die Blickrichtung unterschiedlich ist. Die Algorithmen setzen voraus, dass die zu untersuchenden Objekte eine *lambert’sche* bzw. diffuse Oberfläche aufweisen.

Es gibt Ansätze, die Stereo-Rekonstruktion zu generalisieren. Anhand mehrere Bilder, aufgenommen von mehreren Kameras, ist man in der Lage, die Szene im Computer zu rekonstruieren. Ein solcher Ansatz findet sich in [KZG03] und auch in [KZ02]. Doch

Abbildung 1.1: Stereo-Vision-System bei einem Roboter

auch hier ist eine wesentlicher Anspruch an die Oberfläche gestellt. Haben die korrespondierenden Pixel verschiedene Graustufen, so würden sie nicht als solche erkannt werden. Eine Metall- bzw. Plastikoberfläche verletzt diese Annahme und führt zu Fehlern im errechneten Tiefenbild.

1.1 Algorithmus

Die von uns vorgestellte intuitive Definition der Tiefenrekonstruktion ermöglicht es, auch mit nicht lambert'schen Oberflächen zu arbeiten. Dadurch können stark glänzende Oberfläche, wie die vom Metall oder Plastik (siehe Abb. 1.2) gut rekonstruiert werden.

Zudem spezifizieren wir den Algorithmus als ein Energieminimierungsproblem. Um das Problem zu lösen, verwenden wir das relativ junge Konzept der *Graph-Cuts*. Dieses Konzept wurde bereits in [KZG03] und in [BVZ01] erfolgreich angewendet. Die Graph-Cuts erlauben uns relativ schnell und robust, eine Energiefunktion zu minimieren. Ein Teil der Implementierung wurde auf die *GPU* (Graphic Processing Unit) verlagert. Durch die Verwendung eines programmierbaren Graphikprozessors konnten wir die Zeit für die Rekonstruktion der Tiefe verkürzen.

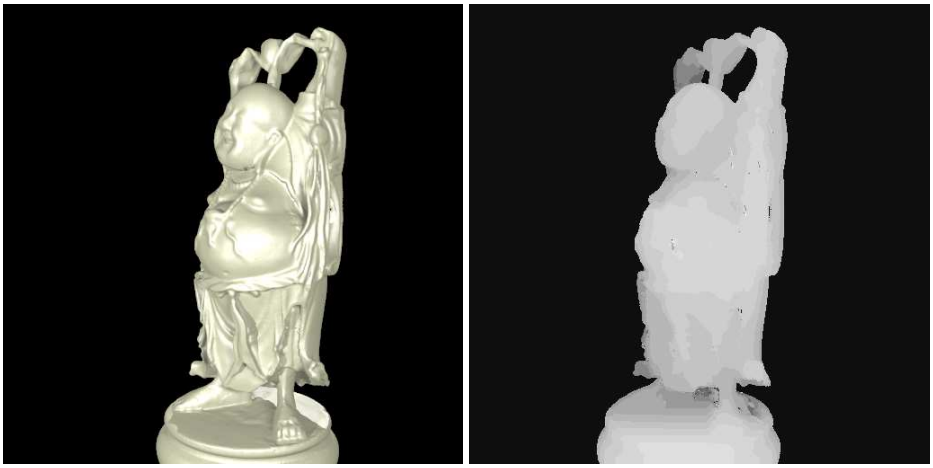

Abbildung 1.2: Das linke Bild zeigt das Happy Buddha Modell [Uni] mit stark glänzender Oberfläche aus Plastik. Das rechte Bild ist die Tiefenrekonstruktion mit Hilfe unseres Algorithmus. Das Modell wurde mit drei Lichtquellen illuminiert, die das starke Glänzen verursachen. Die Tiefenkarte enthält verschwindend wenige Artefakte.

1.2 Einsatzfeld

Die Einsatzmöglichkeiten sind weit und beschränken sich nicht nur auf das Theoretische. Da der Algorithmus, anders als beim Stereo-Matching, mit mehr als zwei Bildern arbeitet, sind die Berechnungen meistens präziser und liefern somit einen genauere Tiefenkarte. Zudem treffen wir nur wenige Annahmen an die Oberflächen der Objekten in der Szene. Im Vergleich zu vielen anderen Algorithmen kommt unserer auch mit nicht lambert'schen Oberflächen zurecht.

Durch die verbesserte Rechengeschwindigkeit und die Vorteile der Parallelisierung, die eine GPU bietet, sind wir in der Lage, die Zeit bis zur Fertigstellung der Tiefenkarte zu reduzieren. Bei kleineren Auflösungen und genügend Rechenkraft ist es sogar vorstellbar, dass der hier vorgestellte Algorithmus in Echtzeit ausführbar wäre. Durch die Berechnung in Echtzeit kann der Algorithmus auch in der Industrie eingesetzt werden, wo die Bestimmung einer Tiefenkarte von den aufgenommenen Bildern wichtig ist, wie z.B in der Autoindustrie oder Robotik.

Die Abbildung 1.3 zeigt den Gewinner des DARPA Grand Challenge im Jahr 2005. Solche Wettstreite der Technologien werden vom Militär gesponsort und dienen dazu, die Entwicklung der autonom fahrenden Vehikel voranzutreiben. Solche Vehikel sind schon jetzt in der Lage, den Weg bis zum Ziel automatisch zu finden. Die Tiefenwahrnehmung ermöglicht es dem OnBoard-Computer, auf die gegebene Umwelt entsprechend zu

Abbildung 1.3: Volkswagen Touareg namens Stanley von der Stanford University gewann das DARPA Grand Challenge im Jahr 2005. Die Aufgabe bestand darin, selbständig durch eine Wüste den Weg zum Ziel zu finden. Stanley hatte mehrere Kameras am Bord, um eine Rundumsicht sowie Tiefenwahrnehmung zu ermöglichen

reagieren, zum Beispiel Hindernissen auszuweichen.

KAPITEL 2

VERWANDTE ARBEITEN

In diesem Kapitel geben wir einen kleinen Überblick über die Arbeiten, die auf dem Gebiet der Computervision, insbesondere der Tiefenrekonstruktion, gemacht wurden. Wir zeigen einige grundlegende Ansätze, die mit Hilfe von Stereo-Bildern die Tiefe zu rekonstruieren versuchen. Zudem zeigen wir einen Ansatz, der die Tiefe anhand mehrerer gegebener Bilder rekonstruiert.

2.1 Stereo-Matching

Stereo-Matching ist ein Problem, dass bereits seit mehreren Jahren studiert wird. Viele unabhängige Forschergruppen entwickelten Algorithmen, um das Problem zu lösen. Die so entwickelten Algorithmen kann man in zwei Gruppen unterteilen: *eigenschafts-* und *flächenbasierte* (feature- and area-based). Eigenschaftsbasierte Algorithmen versuchen Eigenschaften, wie Linien oder Konturen, innerhalb der Bilder zu finden. Die flächenbasierten oder auch *intensitätsbasierten* Algorithmen suchen hingegen nach korrespondierenden Pixeln und benutzen dabei die Eigenschaft, dass die Intensität der Pixel in den Stereobildern ungefähr gleich ist.

Ist man in der Lage zwei “gleiche” Pixel innerhalb der Stereobilder zu ermitteln, so ist die Bestimmung der zugehörigen Tiefe mit Hilfe der *Triangulationsgeometrie* [HZ00] möglich (siehe Abb: 2.1). Wir werden hier nicht näher darauf eingehen, da es nicht relevant für unsere Arbeit ist.

2.1.1 Feature-based Stereo Matching

Bei diesem Ansatz werden die beiden Bilder zuerst von einem *Operator* vorbereitet. Als Operator fungiert ein Algorithmus, der bestimmte Eigenschaften, wie zum Beispiel Kanten oder Ecken, im Bild hervorhebt. Da die Eigenschaften in den Stereobildern selten verschieden sind, kann man nun anhand des so errechneten “gemeinsamen Nenners” die Bilder “matchen”.

Die Laufzeit solcher Algorithmen ist meistens sehr klein, da nur ein geringer Teil der gemeinsamen Eigenschaften geprüft wird. Viele der Algorithmen, die auf diesem Prinzip basieren, arbeiten mit Ecken, Linien- oder Kurvensegmenten innerhalb der Bilder. Diese Elemente sind sehr robust gegenüber der Veränderung der Perspektive, wie sie bei

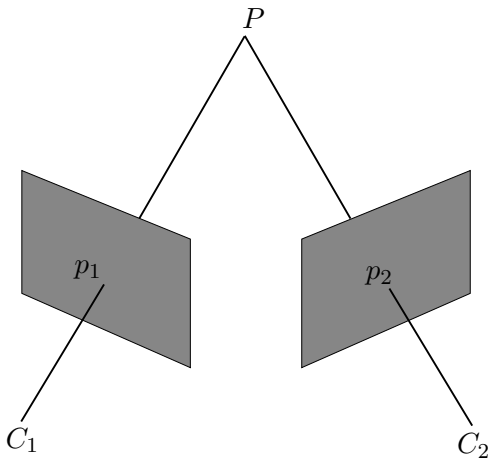

Abbildung 2.1: Der 3D-Punkt P wird auf die beiden Stereobilder von den Kameras C_1 und C_2 in verschiedenen Positionen p_1 und p_2 projiziert. Sind die beiden Positionen p_1 und p_2 bekannt, so kann man mit Hilfe der gegebenen Daten für C_1 und C_2 die Tiefe des Punktes P bestimmen.

Stereobildern vorhanden ist. Ecken und Kanten können schnell gefunden werden, haben aber den Nachteil, dass sie durch Verdeckung oft nicht mehr auffindbar sind. Linien und Kurven sind schwieriger zu finden, dafür aber robuster gegenüber Verdeckungen.

Die Vorteile des feature-based stereo matching liegen in seiner Geschwindigkeit und Robustheit gegen perspektivische Veränderungen. Auch Verdeckungen können mit Hilfe solcher Algorithmen leicht erkannt werden. Aber im Großen und Ganzen erzeugen diese Algorithmen sehr ungenaue Tiefenkarten, da sie nur auf den vorhandenen Eigenschaften arbeiten. Die so entstandenen *mageren* Tiefenkarten können nicht für realistische 3D-Rekonstruktionen verwendet werden, da sie nur einem kleinen Teil der Szene entsprechende Tiefen zuweisen können.

2.1.2 Area-based Stereo Matching

Flächenbasierte Methoden suchen zwei ähnlichste Flächen innerhalb der Stereobilder um einen Pixel. Um die Suche zu beschleunigen, beschränkt man sie auf bestimmte Bereiche, die man mit Hilfe der *Epipolargeometrie* eingrenzt. In der Abbildung 2.1 spannen die Punkte P_1, P_2 und P eine Ebene auf, auch *Epipolarebene* genannt. Die Ebene schneidet die beiden Bildebenen in zwei Schnittgeraden. Solche Geraden innerhalb der Bildflächen nennt man auch *Epipolarlinien*. Sind die beiden Kameras parallel ausgerichtet, verlaufen die Epipolarlinien horizontal in der Bildebene.

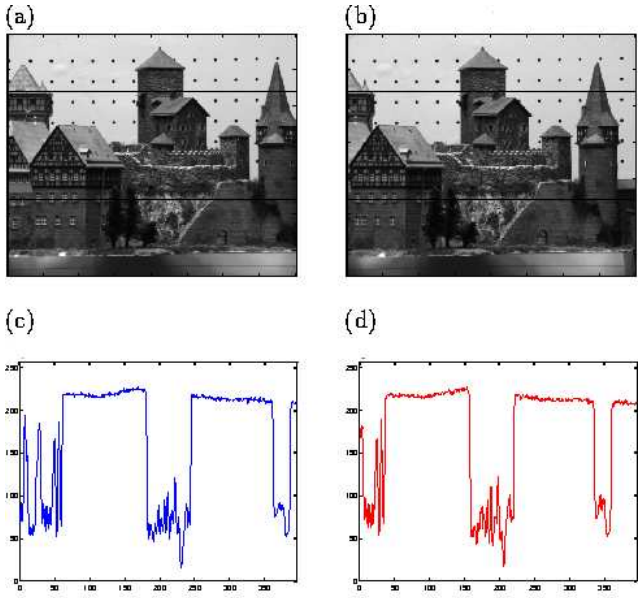

Abbildung 2.2: (a) Linkes Bild, (b) Rechtes Bild, (c) Graustufenverteilung des linken Bildes in der 80-ten Zeile, (d) Graustufenverteilung des rechten Bildes

Durch die Suche innerhalb der Epipolarlinie verkleinert sich das Problem von zwei Dimensionen auf nur eine. Das Verteilungsprofil der Graustufen auf der Epipolarlinie unterscheidet sich in horizontaler Verschiebung sowie lokaler Perspektive. Die Abbildung 2.2 zeigt eine solche Verteilung innerhalb gleicher Epipolarlinie.

2.2 Multi-camera scene reconstruction

Ein weiterer interessanter Ansatz, der mit Hilfe mehrerer Bilder die Tiefeninformationen rekonstruiert, ist *multi-camera scene reconstruction* [KZ02]. Der Ansatz benutzt zwar mehr als zwei Bilder einer Szene, um die Tiefe zu rekonstruieren, dennoch ist er sehr ähnlich zum (zwei Kamera) Stereo-Matching-Problem. Man benutzt Bildpaare benachbarter Kameras, um die Tiefe zu rekonstruieren. Die Ähnlichkeit zwischen den Pixeln der Bilderpaaren wird anhand der Intensitäten verglichen und ist somit sehr ähnlich zu dem intensitätsbasierten Stereo-Matching, das wiederum problematisch bei glänzenden Oberflächen ist. Weitere Arbeiten, die sich mit dem Problem der glänzenden bzw. nicht lambert'schen Oberflächen beschäftigt haben sind [JSY03] und [BN95].

Der Vorteil dieser Methode, sowie der aus Abschnitt 2.1.2, liegt darin, dass als Ergebnis eine *dichte* Tiefenkarte errechnet werden kann. Jedem Pixel auf dem Bild kann

eine Tiefe zugeordnet werden. Nachteil ist die Notwendigkeit ein Optimierungsproblem zu lösen, das nicht unbedingt zu einem globalen Minimum konvergieren muss.

Ein solches Problem kann mit Hilfe der dynamischen Programmierung, wie in [GY03] vorgestellt, genetischen Algorithmen [GY01] oder mit *Graph-Cuts* gelöst werden. Für die Lösung unseres Problems arbeiten wir mit Hilfe von Graph-Cuts. Der ausgewählte Ansatz ist neu und erlaubt eine Reduzierung vieler Rekonstruktionsfehler, wie sie bei glänzenden Oberflächen auftreten.

TIEFENREKONSTRUKTION

In diesem Kapitel zeigen wir, wie wir mit Hilfe mehrerer Bilder parallel ausgerichteter Kameras die Tiefe rekonstruieren bzw. jedem Pixel einen bestimmten Tiefenwert λ zuweisen. Zudem zeigen wir, dass unser Ansatz auch mit glänzenden Oberflächen hervorragend umgehen kann. Das Problem der Tiefenrekonstruierung werden wir als ein Energieminimierungsproblem postulieren und die entsprechende Energiefunktion herleiten. Im nächsten Kapitel werden wir genauer darauf eingehen, wie man die hier vorgestellte Energiefunktion mit Hilfe des Graph-Cut Ansatzes minimiert.

3.1 Problemdefinition

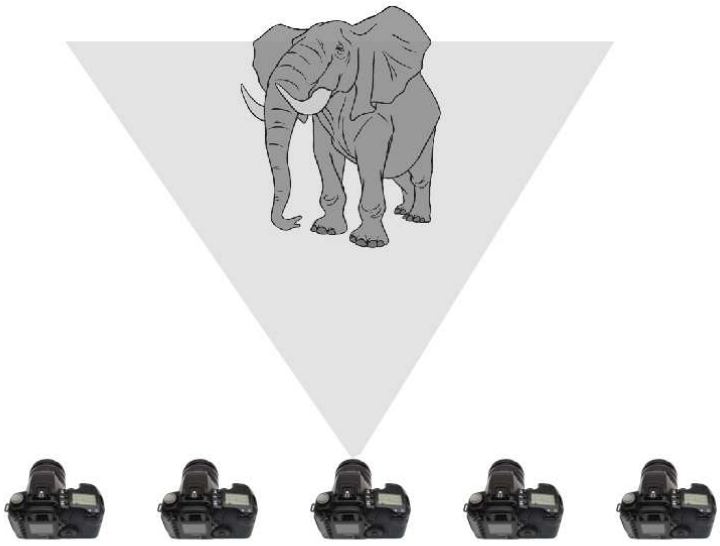

Abbildung 3.1: Eine parallele Anordnung von 5 Kameras. Die einzelnen Sichtbereiche können sich überlagern (hier nur einer eingezeichnet). Die Szene ist statisch.

Um die Tiefe der Szene zu ermitteln, benutzen wir mehrere Bilder parallel ausgerichteter Kameras. Abbildung 3.1 zeigt eine solche Anordnung. Die Bilder werden in ein Volumen der Größe $W \times H \times D$ eingeordnet, wobei W und H die Auflösung und D die Anzahl der Bilder repräsentiert. Ein solches Volumen nennen wir *Epipolarvolumen*.

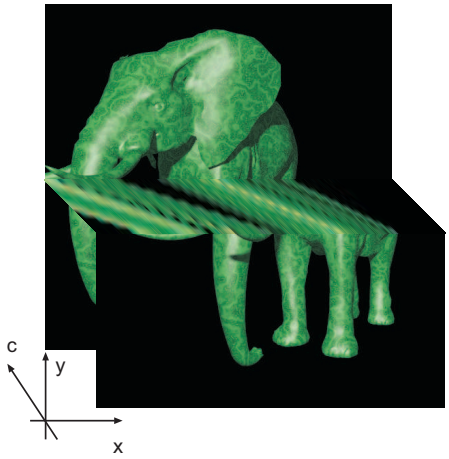

Abbildung 3.2: Schnitt des Epipolarvolumens an der xz-Ebene. Die xy-Ebene bei $c = i$ ist das Bild I_i der Kamera C_i für $i \in \{1...D\}$

Da die Kameras parallel ausgerichtet sind, sind deren Epipolarlinien die horizontalen Linien innerhalb der Bilder. Es ist auch eine beliebige Anordnung der Kameras möglich, sofern man die Bilder entsprechend der parallelen Kameraanordnung *rektifiziert* (siehe auch [HZ00]).

Durch die unterschiedlichen Positionen korrespondierender Pixel innerhalb der Bilder lässt sich nun die entsprechende Tiefe feststellen. Punkte, die näher an der Kamera sind, erfahren eine größere Positionsabweichung als solche, welche weiter von der Kamera entfernt sind. Die Positionsabweichungen bzw. *Disparität* nutzt man im Stereo-Matching-Verfahren, um die Tiefe der Objekte anhand der Triangulationsgeometrie [HZ00] zu errechnen (siehe Abb. 2.1)

Betrachtet man nun einen Schnitt an der xz-Ebene durch das Epipolarvolumen, so kann man einzelne Linien beobachten. Die Linien entstehen durch die oben erwähnten unterschiedlichen Positionen einzelner Pixel innerhalb der Bilder. Die Steigung der Linien entspricht der Entfernung der zugehörigen Punkte von den Kameras. Abbildung 3.2 zeigt einen solchen Schnitt des Epipolarvolumens.

Das Problem können wir nun als ein Energieminimierungsproblem innerhalb des Epipolarvolumens definieren. Ist ein Punkt P in der Szene gegeben, so können wir ihn an der Linie l_P in dem Epipolarvolumen V finden. Eine wichtige Beobachtung ist, dass die Steigung λ_P der Linie l_P umgekehrt proportional zu der Tiefe des Punktes P ist. So ist für $\lambda_P = 0$ die Tiefe $P_z = \infty$. Für den Fall, dass wir nur zwei Bilder haben ($D = 2$), spricht man für λ_P von einer Disparität.

Die Farbwerte innerhalb einer solchen Linie sind durch die *BRDF* (Bidirectional Reflectance Distribution Function, zu deutsch: bidirektionale Reflektanzverteilungsfunktion) f_r definiert (siehe Kapitel 3.3). Da die Anordnung der Lichter und die zugehörige BRDF für jeden Punkt P konstant sind, verändert sich folglich der Farbwert des Pixels nur unter dem Blickwinkel. Ist die BRDF f_r^P des Punktes P bekannt, so kann man das Problem als eine Energiefunktion der Form $E(p, \lambda) = d(l_P, f_r^P)$ auffassen, wobei $d(f, g)$ der Abstand der beiden Vektoren f und g ist. Eine Minimierung der Funktion

$$\lambda^* = \arg\min_{\lambda} E(p, \lambda) \tag{3.1}$$

für jeden Pixel P führt zur Bestimmung des Parameters λ^* und erlaubt somit die Berechnung der Tiefe P_z.

3.2 Epipolarvolumen

Für einen gegebenen Punkt x ist die beobachtete Lichtstrahlung in eine bestimmte Richtung r definiert durch die plenoptische Funktion $O(x, r)$; $O : \mathbb{R}^3 \times \mathbb{S}^2 \rightarrow \mathbb{R}^d$ (siehe [AB91]), wobei S^2 die Einheitskugel aller Richtungen in $\mathbb{R}^3$ ist. Für $d = 1$ spricht man von *Intensität* für $d = 3$ von Farbwerten. Für unsere Berechnungen gehen wir von einem *Lochkamera-Modell* aus. Somit lässt sich jedes aufgenommene Bild definieren durch $I(c) = O(c, r), \forall r \in \mathbb{S}_c^2$ für eine Kamera an der Position c und eine Untermenge $\mathbb{S}_c^2 \subseteq \mathbb{S}^2$, die durch die Kameraparameter, wie Öffnungswinkel (*field-of-view*) und Blickrichtung v, definiert ist.

Durch die parallele Anordnung der Kameras auf einer Linie $l_c = c_0 + zd, z \in \{0...D-1\}$, wobei d der Abstand zwischen den Kameras ist und $d \perp v$, können wir nun das Epipolarvolumen $V \in \mathbb{R}^2 \times \mathbb{N}$ wie folgt definieren:

$$V_z = I(c_z), z \in \{0...D-1\} \tag{3.2}$$

Mit anderen Worten ist das Volumen durch die Bilder der einzelnen Kameras definiert, die entlang der diskreten z-Achse bzw. c-Achse aufeinander in ihrer Reihenfolge gestapelt sind. In der Abbildung 3.2 ist ein solches Epipolarvolumen gezeigt.

3.3 Oberflächen BRDF

Das Neue an unserem Ansatz ist, dass wir in der Lage sind, die Tiefenkarten auch für glänzende Oberflächenstrukturen, wie Plastik oder Metall, zu berechnen.

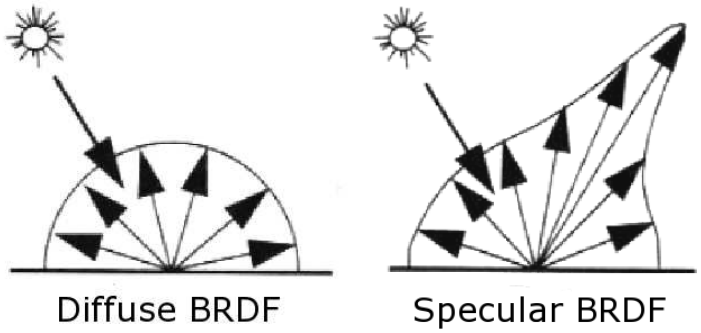

Abbildung 3.3: *Links*: BRDF einer diffusen Oberfläche. *Rechts*: BRDF einer nicht lambert'schen Oberfläche

Eine BRDF ist eine Funktion, die entsprechend dem Einfallswinkel und dem Ausfallswinkel die Intensität des reflektierten Lichtes beschreibt:

$$f_r(\theta_i, \phi_i, \theta_o, \phi_o) \tag{3.3}$$

wobei θ_i,ϕ_i den Einfallsvektor und θ_o,ϕ_o den Sichtvektor durch die Polarkoordinaten definieren.

Lambert'sche Oberflächen haben eine BRDF, die durch eine Halbkugel repräsentiert ist. Oberflächen wie Metall hingegen reflektieren das Licht in bestimmte Richtungen stärker, was eine ungleichmäßige, aber dennoch glatte BRDF zur Folge hat (siehe Abb. 3.3).

Die Intensitätsverteilung der Linie l_P für den Punkt P innerhalb der xz-Ebene des Epipolarvolumens V entspricht der für diesen Punkt spezifischen BRDF. Da die gegebene Szene statisch ist und somit die Lichtpositionen sowie die BRDF sich nicht ändern, variiert die Intensitätsverteilung nur in Abhängigkeit der Blickrichtung v.

Für unsere Tiefenrekonstruktion stellen wir die Annahme auf, dass die BRDF glatt und sprunglos innerhalb von l_P verläuft. In einer realen Szene ist dies in der Regel der Fall, vorausgesetzt man verwendet keine perfekt spiegelnden Oberflächen. Ein Energiefunktional, dass die "Unglattheit" eines solchen Intensitätsverlaufes ermittelt, kann dazu verwendet werden, die entsprechende Linie l_P für einen Punkt P zu finden. Durch die Minimierung dieser Energiefunktion suchen wir quasi nach der "bestglatten" Linie innerhalb aller Epipolarlinien.

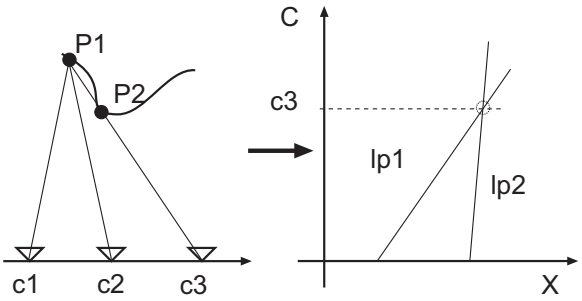

Abbildung 3.4: Der Punkt P_1 wird vom Punkt P_2 verdeckt. Die Verdeckung ist erst im Bild der Kamera c_3 erkennbar. Die Steigung der Epipolar-Linie l_{P_1} ist kleiner als die der l_{P_2}.

3.4 Verdeckte Pixel

Da die Szene, deren Tiefe wir zu rekonstruieren versuchen, aus mehreren Kamerapositionen aufgenommen wurde, können einige Punkte verdeckt sein.

Verdeckte Pixel verletzen die Annahme, dass jeder Punkt einer Objektoberfläche auf der Linie l_P zu finden ist. Dies geschieht nach festen Regeln, die innerhalb des Epipolarvolumens gelten. Ist der Punkt P_1 durch den Punkt P_2 verdeckt, dann gilt folgendes:

1 P_2 ist näher an der Kamera als P_1

2 Die zugehörigen Linien l_{P_1} und l_{P_2} schneiden sich

Wir definieren die Menge der verdeckten Pixel innerhalb der Linie l_P als P_v. Die neue Definition der Linie innerhalb des Epipolarvolumens kann dann wie folgt aufgefasst werden:

$$\hat{l}_P = l_P \setminus P_v \tag{3.4}$$

Außerdem stellen wir fest, dass die zugehörige Steigung der beiden Linien verschieden ist, genauer ist $\lambda_1 < \lambda_2$. D.h. die Linie l_{P_1} hat kleinere Steigung, da der zugehöriger Punkt weiter von der Kamera entfernt ist. Der Sachverhalt ist zudem im Bild 3.4 verdeutlicht.

Diese Eigenschaft nutzen wir, um die Verdeckungen aufzulösen. Im Gegensatz zu den bisherigen Verfahren, die einen gesonderten Term, wie $E_{occlusion}(f)$, in die Energiefunktion mitaufnehmen, behandeln wir die Verdeckung innerhalb des Datenterms. Von der errechneten Tiefenkarte können wir die einzelnen Tiefen der benachbarten Pixel

auslesen und entscheiden, ob der Pixel verdeckt ist. In der Implementierung im Kapitel 5.1.1 gehen wir genauer darauf ein, wie wir mit den verdeckten Pixeln umgehen.

3.5 Energiefunktion

Wir fassen das Problem der Tiefenbestimmung unter dem Gesichtspunkt des Labeling-Problems auf. Hierzu müssen wir jedem Pixel ein bestimmtes Label λ zuweisen, dass seine Tiefe repräsentiert. Das Problem kann somit wie folgt formuliert werden: Finde eine Funktion f, die jedem Pixel $p \in \mathcal{P}$ ein Label $f_p \in \mathcal{L}$ zuordnet, wobei f stückweise glatt und konsistent mit den beobachteten Daten ist.

Solch ein Problem kann mit Hilfe des Energieminimierungsansatzes spezifiziert werden. Das heißt, wir suchen nach einer Funktion bzw. einem Labeling, dass den Term:

$$E(f) = E_{data}(f) + E_{smooth}(f) \tag{3.5}$$

minimiert. Dabei stellt $E_{data}(f)$ die Unstimmigkeit zwischen f und den beobachteten Daten fest und $E_{smooth}(f)$ misst das Ausmaß der Glattheit der Funktion f. Typischerweise ist

$$E_{data}(f) = \sum_{p \in \mathcal{P}} D_p(f_p) \tag{3.6}$$

und wird auch als Datenterm bezeichnet. Der Glattheitsterm E_{smooth} wird durch

$$E_{smooth}(f) = \sum_{(p,q) \in N} V_{(p,q)}(f_p, f_q) \tag{3.7}$$

angegeben. Manchmal bezeichnet man ihn als Nachbarschaftsterm, da er nur für die Pixel berechnet wird, die in der definierten Nachbarschaft N liegen.

Die beiden Terme im Kontext der Energieminimierung haben quasi Wächterfunktion, welche die falsch gewählten Vermutungen über die zugrunde liegenden Daten bzw. Nachbarschaften mit einem höheren Energiewert bestraft und die richtige Wahl begünstigt. Dieses Konzept der Bestrafung und Begünstigung wird eine große Rolle im nächsten Kapitel spielen, wo wir darauf eingehen werden, wie eine Energiefunktion dieser Form minimiert werden kann.

3.5.1 Nachbarschaftsterm

Die Wahl für den Nachbarschaftsterm ist entscheidend dafür, wie stark die Korrelation der Nachbarschaftspixel in der Berechnung berücksichtigt wird. Ist die Gewichtung des Term im Vergleich zum Datenterm zu klein, so enthalten die errechneten Tiefenkarten

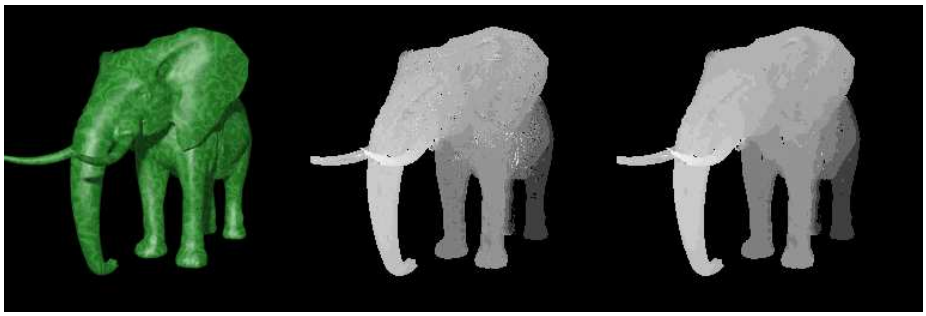

Abbildung 3.5: *Links*: Originalbild. *Mitte*: rekonstruierte Tiefenkarte ohne die Nachbarschaftsbeziehungen. *Rechts*: Tiefenkarte mit Nachbarschaftsterm

wenige Nachbarschaftsinformationen, was zu deutlichen Artefakten führen kann. Ist er im Gegensatz zu groß gewichtet, so wirken die Tiefenkarten sehr glatt, was wiederum eine ungenauere Tiefenkarte nach sich zieht. Siehe Abbildung 3.5

Um die beste Wahl für den Nachbarschaftsterm zu treffen, wählten wir einen Term der auf dem Pott's Modell, wie er in [BVZ01] vorgestellt wurde, basiert:

$$V_{p,q}(f_p, f_q) = \begin{cases} 0 & |f_p - f_q| < d_1 \\ 2K & d_1 \leq |f_p - f_q| < d_2 \\ K & \text{sonst} \end{cases} \tag{3.8}$$

wobei K eine Konstante ist, welche die Stärke der Nachbarschaftskorrelation repräsentiert. Die Werte d_1 und d_2 sind empirisch gewählt. Für unseren Fall haben wir $d_1 = \max(\frac{|\mathcal{L}|}{3}, 2)$ und $d_2 = d_1 * 2$.

3.5.2 Datenterm

Im Abschnitt 3.3 haben wir festgestellt, dass die zugehörige Linie l_P des Punktes P durch die Minimierung einer bestimmten Energiefunktion gefunden werden kann. Solch eine Energiefunktion muss die Glattheit einer Funktion $f(l_P)$ ermitteln.

Eine Funktion ist glatt in unserem Sinne, wenn sie differenzierbar ist, d.h. $f \in C^1$. Durch die Ableitung der Funktion f an einer Stelle x bestimmen wir die Steigung an dieser Stelle. Sehr glatte Funktionen in unserem Sinne haben keine große Veränderungen in ihrer Steigung. Um die verschiedenen Änderungen der Differenzwerte $\frac{\delta}{\delta x} f(x) = f'(x)$ zu ermitteln, greifen wir zur *Varianz*-Funktion, bekannt aus der Wahrscheinlichkeitsrechnung. Die Varianz ist ein Maß für die Abweichung von einem bestimmten Erwar-

tungswert, was in unserem Fall die mittlere Steigung ist. Somit definieren wir den Energieterm als:

$$D_p = Var(f'(l_P^\lambda)) \tag{3.9}$$

wobei l_P^λ die Linie innerhalb des Volumens V mit einer bestimmten Steigung λ ist. Die Varianz ist durch den Term:

$$Var = \frac{1}{D} \sum_{i=0}^{D-1} \left(f'(l_P^\lambda[i]) - m\right)^2 \tag{3.10}$$

gegeben. Wobei der Erwartungswert m als

$$m = \frac{1}{D} \sum_{i=0}^{D-1} f'(l_P^\lambda[i]) \tag{3.11}$$

definiert ist.

Anschaulich gesprochen bestraft der Datenterm die Intensitätsänderungen $f'(l_P^\lambda)$, die sehr große Abweichungen gegenüber dem Mittelwert haben. Kleinere bis keine Abweichungen hingegen werden bevorzugt. Somit bestraft der Datenterm indirekt die Wahl für die Steigung λ der Linie l_P. Falls die Intensitäten innerhalb der Linie zu stark von der "Ideallinie" abweichen, ist der Term groß, für kleinere Abweichungen ist der Term klein.

3.5.3 Energiemodell

Wir haben nun die Energiefunktion hergeleitet, die, minimiert, unser Problem der Labelzuweisung lösen soll:

$$E(f) = \sum_{(p,q)\in N} V_{(p,q)} + \sum_{p\in\mathcal{P}} D_p(f_p) \tag{3.12}$$

wobei $V_{(p,q)}$ der Nachbarschaftsterm aus dem Abschnitt 3.5.1 und $D_p(f_p)$ der Datenterm aus 3.5.2 sind. Die Energiefunktionen dieser Art werden auch als Pott's Energiefunktion bezeichnet. Durch die Arbeit von Boykov, Veksler und Zabih in [BVZ01] wissen wir, dass die Minimierung einer Pott's Energiefunktion *multiway-cut*-Problem [DJP$^+$92] entspricht. Dieses Problem ist NP-hart und nach dem heutigen Stand der Wissenschaft nicht in polynomieller Zeit lösbar. Der im nächsten Kapitel präsentierte Ansatz von Graph-Cuts ermöglicht es, dieses Problem in polynomieller Zeit zu approximieren.

3.6 Zusammenfassung

In diesem Kapitel haben wir uns mit der Frage beschäftigt, wie man auch nicht diffus reflektierende Oberflächen in der Tiefenrekonstruktion einbezieht. Die Minimierung der Energie erfolgt für die jeweiligen Labels, welche die Steigung der Epipolarlinien innerhalb des Epipolarvolumens definieren. Für die Berechnung der Glattheit der BRDF haben wir die Varianz verwendet, die aus der Wahrscheinlichkeitsrechnung bekannt ist. Dabei bestraft der Datenterm eine falsche Wahl für das Label λ und begünstigt diese, falls die Epipolar-Linie keine allzu große Schwankungen in ihrer Glattheit aufweist. Um nun die Energiefunktion zu minimieren, d.h. die günstigste Zuweisung der Labels zu den Pixeln zu finden, wird der Ansatz der Graph-Cuts verwendet, dessen nähere Beschreibung im nächsten Kapitel zu finden ist.

KAPITEL 4

ENERGIEMINIMIERUNG

Im letzten Kapitel haben wir das Problem der Bestimmung einer Tiefenkarte für das gegebene Epipolarvolumen V als ein Energieminimierungsproblem spezifiziert. Es gibt viele Ansätze, wie man eine Energiefunktion entsprechend minimiert um ein annähend globales Minimum zu finden. In unserem Fall nutzen wir eine Methode namens Graph-Cut. Die hier eingesetzte Variante der Minimierung, auch als α-Expansion-Move bekannt, ist in der Lage, eine Lösung nahe eines globalen Minimums zu finden.

4.1 Graph-Cut

Graph-Cut ist einer Methode, um ein Energieminimierungsproblem zu lösen. Graph-Cuts werden ausgiebig im *Computer Vision* Bereich genutzt, wo man das *Pixel Labeling*-Problem mit Hilfe solcher Methoden zu lösen versucht. Unser Algorithmus der Tiefenrekonstruktion ist auch in diese Problemgruppe einzuordnen, denn wir versuchen ebenso anhand gegebener Daten bestimmte *Labels* einem Pixel zuzuordnen. In unserem Fall entsprechen Labels den Tiefenwerten.

4.2 Geschichte

Die Idee der Graphen ist sehr alt und reicht bis ins Jahr 1736, wo sie zum ersten mal von Leonhard Euler ins Leben gerufen wurde. Euler fragte sich, ob es möglich ist, Königsberg zu bewandern ohne dabei mehr als einmal die gleiche Brücke zu überqueren. Das eigentliche Geburtsjahr der Graphentheorie ist jedoch 1976, als Kenneth Appel und Wolfgang Haken das Vier-Farben-Problem [AHK89] lösten. Dabei ging es um die Möglichkeit, alle Länder auf einer Karte mit nur vier Farben auszufüllen, ohne dass zwei benachbarte Länder die gleiche Farbe haben. Während der Lösungsentwicklung wurden viele fundamentale Konzepte der Graphentheorie entwickelt.

Im Jahre 1956 präsentierten Delber Ray Fulkerson und Lester Randoplh eine Arbeit, in der es um das so genannte Maximalfluss-Minimalschnitt-Problem ging [FF62]. Dabei stellten Fulkerson und Randolph unabhängig voneinander fest, dass ein minimaler Schnitt (minimal cut) in einem Graph äquivalent zu dem maximalen Fluss (maximal flow) ist. 1970 präsentierte ein russischer Wissenschaftler namens Dinic [Din70] einen

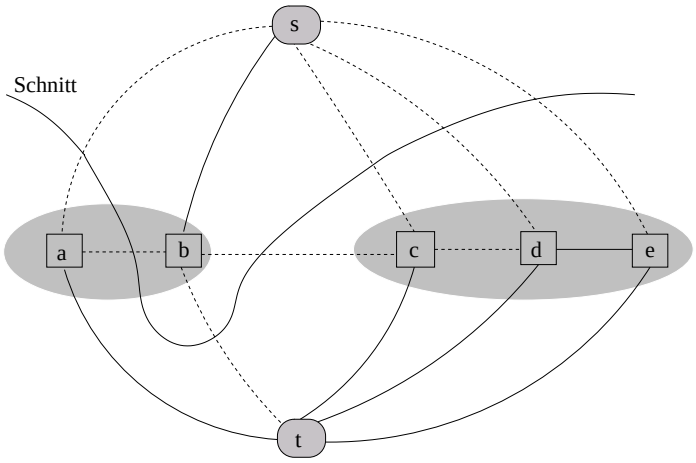

Abbildung 4.1: Schnitt eines Graphen. Die Kosten eines Schnittes werden durch die Kantengewichten der geschnittenen Kanten berechnet.

Algorithmus, der das Berechnen des maximalen Flusses in einem Graphen/Netzwerk ermöglichte.

4.3 Minimaler Schnitt

Erstens definieren wir, was ein Schnitt eines Graphen überhaupt ist. Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein gerichteter Graph mit nicht negativen Kantengewichten $c(u, v), (u, v) \in \mathcal{E}$. Der Schnitt eines Graphen G ist eine Partition der Knoten aus $\mathcal{V}$ in zwei *disjunkte* Mengen $\mathcal{S}$ und $\mathcal{T}$. Die Kosten eines solchen Schnittes definiert man als die Summe aller Gewichte der Kanten, die einen Knoten aus $\mathcal{S}$ mit einem Knoten aus $\mathcal{T}$ verbinden:

$$C(\mathcal{S}, \mathcal{T}) = \sum_{u \in \mathcal{S}, v \in \mathcal{T}, (u,v) \in \mathcal{E}} c(u, v) \tag{4.1}$$

Ein Minimal-Schnitt-Problem bedeutet, den Schnitt $\mathcal{C}(\mathcal{S}, \mathcal{T})$ mit den kleinsten Kosten zu finden. Wir führen nun zwei zusätzliche Terminalknoten s und t ein, *Quelle* und *Senke*. Ein Schnitt $\mathcal{C}$ des Graphen $\mathcal{G}$ erzeugt zwei disjunkte Mengen $\mathcal{S}$ und $\mathcal{T}$, sodass $s \in \mathcal{S}$ und $t \in \mathcal{T}$. Einen solchen Schnitt bezeichnet man auch als *s-t-Schnitt*. Nach dem Theorem von Ford and Fulkerson [FF62] entspricht das Minimal-Schnitt-Problem dem Maximal-Fluss-Problem. Es gibt bereits viele Ansätze, die das Problem des maximalen Flusses in polynomieller Zeit lösen.

4.4 Maximaler Fluss

Um besser zu verstehen, was ein Maximal-Fluss-Problem ist, stellt man sich den maximalen Fluss als die maximale Menge an Wasser vor, dass von der Quelle bis zur Senke fließen kann. Dabei stellen die Kanten die Rohre dar, die Kantengewichte stellen die Kapazität bzw. die Menge an Wasser dar, die durch das Rohr fließen kann. Ein maximaler Fluss ist demnach die größte Wassermenge, die man durch das gesamte Rohrsystem(Graph) von der Quelle bis zur Senke pumpen kann. Als minimalen Schnitt bezeichnet man die kleinstmögliche Menge der Engstellen, die man dicht machen muss, damit kein Tropfen Wasser mehr von der Quelle zur Senke fließen kann.

Der Brute-Force-Ansatz für die Lösung des Problems hat eine Komplexität von $O(2^{|\mathcal{V}|-2}|\mathcal{E}|)$, da man alle mögliche Schnitte ausprobiert, die die Quelle von der Senke trennen, bis man den besten gefunden hat. Für die praktische Arbeit nutzt man schnellere Algorithmen, wie den von Dinic [Din70]. Die Idee hinter dem Algorithmus von Dinic ist relativ einfach. Der Algorithmus benutzt den Ansatz von *breadth-first-search* (Breite-Zuerst-Suche), um die kürzesten vergrößerungsfähigen (*augmenting*) Pfade zu finden. Einen solchen Pfad $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_k$, wobei $u_1 = s$ und $u_k = t$, bezeichnet man als vergrößerungsfähig, wenn die Kantenrestkapazitäten größer als Null sind. Anschaulich bedeutet das, dass man zusätzlich mehr Wasser durch diesen Pfad pumpen kann, als es bereits geschehen ist. Ist solch ein Pfad gefunden, so vergrößert man den Fluss durch den Pfad um d_f bis eine der Kanten innerhalb des Pfades *gesättigt*, d.h. keine Vergrößerung des Flusses mehr möglich ist. Der Gesamtfluss des Graphen errechnet sich als $f = f + d_f$. Sobald alle kürzesten Pfade der fixen Länge k *gesättigt* sind, startet man die Suche mit den Pfaden der Länge $k + 1$. Die Laufzeit des Algorithmus lässt sich mit $O(|\mathcal{V}||\mathcal{E}|^2)$ angeben.

In unserer Implementation der Tiefenrekonstruktion nutzen wir einen neuen Ansatz aus [BK04], der im Grunde genommen ähnlich zu dem bereits erwähnten Ansatz von Dinic arbeitet. Eine genauere Erklärung kann man in [BK04] finden. Der Algorithmus wurde zudem von den Autoren auch als eine Bibliothek implementiert [BK]. Diese hervorragende Bibliothek nutzen wir, um das Problem des maximalen Flusses zu berechnen und in unserer Implementation des Algorithmus zu verwenden.

4.5 Energieminimierung

Der Einsatz von Graph-Cuts ermöglicht es, die Energiefunktion, die wir in vorherigem Kapitel hergeleitet haben, zu minimieren. Um eine Energiefunktion mit Hilfe von Graph-

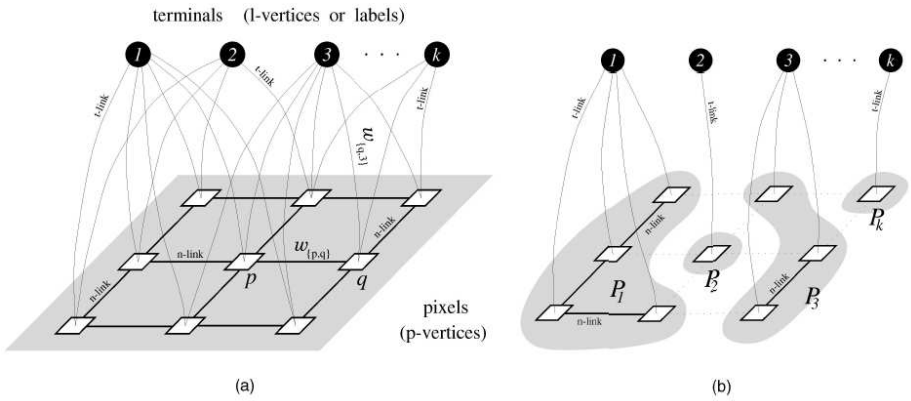

Abbildung 4.2: (a) Beispiel eines Graphen mit mehreren Terminalknoten. Jeder der Terminalknoten steht für ein Label. Die anderen Knoten repräsentieren die Pixel eines Bildes. (b)Multiway-cut auf dem Graph. Die Knoten sind nun partitioniert und sind mit dem günstigeren Label verbunden. Quelle [BVZ01]

Cuts zu minimieren, müssen wir einen Graph erstellen, in welchem die Berechnung eines minimalen Schnittes auch die Energiefunktion minimiert. Dabei werden die Pixel durch Knoten repräsentiert und die Energiefunktionsterme, wie der Nachbarschaftsterm und der Datenterm, werden als Gewichte zwischen den Kanten gespeichert.

Abbildung 4.2 zeigt einen solchen Graphen sowie einen Minimalschnitt. Die so errechnete Partitionierung minimiert global die Energiefunktion und ordnet jedem Pixel $p \in \mathcal{P}$ ein Label $l \in \mathcal{L}$ zu. Das Problem ist jedoch NP-hart und nach dem aktuellem Stand der Wissenschaft nicht in polynomieller Zeit lösbar. Den Beweis dafür findet man in [BVZ01], das Problem der Multiway-Cuts findet man in [DJP$^+$92].

Um das Problem nun zu approximieren, wenden wir einen Trick an. Wir iterieren über alle Labels und versuchen diese den Pixeln zuzuordnen. Ist die Wahl des Labels für ein Pixel günstig, d.h. die Energieterme sind am niedrigsten, so wird diesem Pixel das Label zugewiesen. In jedem Schritt lösen wir somit nur eine Frage: Wird Pixel p das Label l zugewiesen? Das Problem könnte man auch als ein Multiway-Cut-Problem mit nur zwei Terminalknoten bezeichnen. Für die Lösung eines solchen Problems sind Graph-Cuts bestens geeignet und erlauben uns, es in polynomieller Zeit zu lösen.

Diese intuitive Überlegung führt nun zu folgendem Algorithmus (siehe auch [BVZ01]):

1. Starte mit beliebigem Labeling, d.h. $f_p \in \mathcal{L},\ \forall p \in \mathcal{P}$

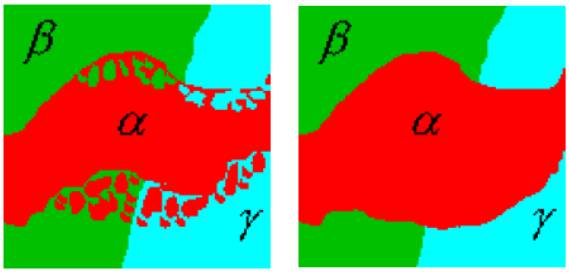

Abbildung 4.3: Einigen der Pixel wird das neue Label α zugewiesen, da dies die Energieminimierung begünstigt. Quelle [BVZ01]

2. Setze gefunden = 0
3. Für jedes Label $l \in \mathcal{L}$ tue:
   - 3.1. Finde g = argmin $E(f')$, wobei f' durch α-Erweiterung aus f berechnet wird
   - 3.2. Falls $E(g) < E(f)$ dann setze $f := g$ und gefunden = 1
4. Falls gefunden = 1 dann gehe zu 2.

Eine solche Approximierung führt nicht immer zu einem globalen Minimum der Energiefunktion. In der Arbeit von Olga Veksler [Vek99] wurde gezeigt, dass dieser Ansatz ein lokales Minimum berechnet, dass nahe dem globalen liegt. D.h:

$$E(f^*) \leq E(f) \leq 2k \cdot E(f^*) \tag{4.2}$$

wobei f^* das globale Minimum ist und $k = \frac{\max\{V(\alpha,l):\alpha \neq l\}}{\min\{V(\alpha,l):\alpha \neq l\}}$. Für uns heißt es, dass wir nur eine approximierte Lösung des globalen Minimums berechnen können, was sich in der Realität fast nicht bemerkbar macht.

4.5.1 α-Erweiterung

Der α-Erweiterungsschritt (engl: α-expansion-step) ist der Knackpunkt der Energieminimierung. Wie wir bereits festgestellt haben, können wir das Problem der Energieminimierung zur Zeit nicht polynomiell lösen. Dennoch ist das Problem der Zuweisung von zwei Labels zu den Pixeln mit Hilfe der Graph-Cuts lösbar. Diese Feststellung nutzen wir nun, um eine approximierte Lösung herzuleiten.

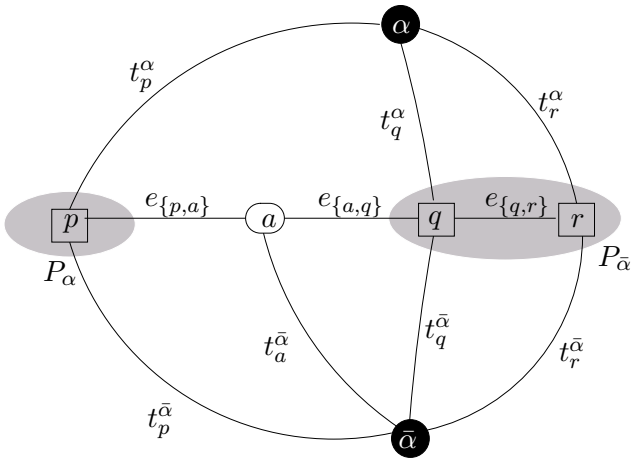

Abbildung 4.4: Konstruktion des Graphen für das Labelingproblem. Die Menge der Pixel ist $\mathcal{P} = \{p, q, r\}$. Die Pixel sind in zwei Mengen unterteilt in $\mathcal{P}_\alpha = \{p\}$ und $\mathcal{P}_{\bar{\alpha}} = \{q, r\}$

Die α-Erweiterung sieht vor, dass man mit Hilfe eines minimalen Schnittes feststellt, welchen Pixeln das neue Label α zugewiesen wird (siehe Abb. 4.3). Der Algorithmus aus dem Abschnitt 4.5 führt den α-Erweiterungsschritt jeweils für jedes Label $l \in \mathcal{L}$ aus. Dadurch wird iterativ für jedes Pixel gefragt, ob die Zuweisung eines anderen Labels $l = \alpha$ die Minimierung der Funktion begünstigt.

Was nun fehlt, ist ein Graph, dessen Schnitt die Energiefunktion minimiert bzw. die Wahl des neuen Labels begünstigt.

4.5.2 Konstruktion des Graphen

Um das Problem der Labelzuweisung mit Hilfe der Graph-Cuts zu lösen, müssen wir einen geeigneten Graph konstruieren. Wie wir bereits festgestellt haben, enthält ein solcher Graph nur zwei Terminalknoten: die Quelle s und die Senke t. Die Berechnung eines minimalen Schnittes ist bei einem solchen Graphen in polynomieller Zeit möglich.

Wir konstruieren den Graphen, wie er in [BVZ01] beschrieben wurde. Die Abbildung 4.4 verdeutlicht die Konstruktion. Die Quelle, in unserem Fall der Knoten α, wird mit jedem Pixel-Knoten (P-Knoten) verbunden. Der Graph wird in zwei Bereiche unterteilt $\mathcal{P}_\alpha$ und $\mathcal{P}_{\bar{\alpha}}$. $\mathcal{P}_\alpha$ enthält alle P-Knoten der Pixel, für die gilt: $f_p = \alpha$, $\mathcal{P}_{\bar{\alpha}}$ hingegen enthält alle übrigen Pixel, für die gilt: $f_p \neq \alpha$. Stammen zwei Pixel aus der gleichen Nachbarschaft ($(p, q) \in N$) und sind diese in verschiedenen Partitionen (d.h. $f_p \neq f_q$), so wird ein neuer Knoten a zwischen p und q hinzugefügt. Zudem werden alle Knoten

Kante	Gewicht	gilt für:
$t_p^{\bar{\alpha}}$	∞	$p \in \mathcal{P}_\alpha$
$t_p^{\bar{\alpha}}$	$D_p(f_p)$	$p \notin \mathcal{P}_\alpha$
t_p^{α}	$D_p(\alpha)$	$p \in \mathcal{P}$
$t_a^{\bar{\alpha}}$	$V(f_p, f_q)$	$(p,q) \in N, f_p \neq f_q$
$e_{\{p,a\}}$	$V(f_p, \alpha)$	
$e_{\{a,q\}}$	$V(\alpha, f_q)$	
$e_{\{q,r\}}$	$V(f_p, \alpha)$	$(q,r) \in N, f_p = f_q$

Tabelle 4.1: Gewichte für die Kanten innerhalb des Graphen für das 2-Labeling-Problem. Quelle [BVZ01]

mit der Senke, in unserem Fall $\bar{\alpha}$, verbunden. Die einzelnen Gewichte für die Kanten zwischen den Knoten sind in der Tabelle aufgeführt.

Sei $\mathcal{C}$ ein minimaler Schnitt des Graphen $\mathcal{G}$. Wir definieren $\mathcal{C}$ als die Menge aller Kanten, die der Schnitt schneidet, d.h. $\mathcal{C} \subset \mathcal{E}$. Die Berechnung eines maximalen Flusses bzw. eines minimalen Schnittes trennt die P-Knoten von den jeweiligen Terminalknoten. Die Zuweisung der Labels $f^{\mathcal{C}}$ nach dem Schnitt definieren wir nun wie folgt:

$$f_p^{\mathcal{C}} = \begin{cases} \alpha & \text{falls } t_p^{\alpha} \in \mathcal{C} \\ f_p & \text{sonst} \end{cases} \quad \forall p \in \mathcal{P} \tag{4.3}$$

Mit anderen Worten wird einem Pixel p das Label α zugewiesen, falls der Schnitt $\mathcal{C}$ den P-Knoten p von dem Terminalknoten α trennt. Dies entspricht in der Tat auch der intuitiven Zuweisung neuer Labels. Das heißt, dass das Gewicht der Kante t_p^{α} kleiner war, als das Gewicht der Kante $t_p^{\bar{\alpha}}$. Somit ist es für dieses Pixel günstiger, das neue Label $l = \alpha$ anzunehmen, weil so die Energiefunktion insgesamt verkleinert wird.

Ist der Schnitt auf dem ganzen Graphen ausgeführt, so bekommen mehrere Pixel gleichzeitig ein neues Label zugewiesen. Im nächsten Schritt wird der gleiche Prozess für das nächste Label durchgeführt. Eventuelle neue Änderungen der Labelzuweisung können erfolgen, was sich nach und nach in der Minimierung der gesamten Energiefunktion bemerkbar macht.

4.6 Zusammenfassung

Der hier vorgestellte Ansatz der Energieminimierung ist in der Lage, ein annähend globales Minimum der Energiefunktion zu berechnen. Die α-Erweiterung erlaubt uns die

NP-Härte des Problems von Multiway-Cuts zu umgehen und die Lösung zu approximieren. Wir benutzen einen Graph mit nur zwei Labels α und $\bar{\alpha}$, dessen minimaler Schnitt bzw. maximaler Fluss in polynomieller Zeit berechenbar ist. Ein Beweis über die Richtigkeit der Konstruktion des Graphen sowie eine genauere Beschreibung findet man in [BVZ01]. Eine Implementation das Maximal-Fluss-Problems findet man in [BK]. Diese Implementierung werden wir im nächsten Kapitel nutzen, um den hier vorgestellten Algorithmus in Programmcode umzusetzen.

IMPLEMENTIERUNG

In diesem Kapitel werden wir näher darauf eingehen, wie die Implementierung des Tiefenrekonstruktionsalgorithmus mittels Graph-Cut umgesetzt wurde. Zudem zeigen wir eine Möglichkeit, wie man die Berechnung beschleunigen kann, indem man einen Teil der zu errechnenden Daten von der *GPU* (General Processing Unit) berechnen lässt. Am Schluss werden wir einen kleinen Vergleich zwischen der Berechnung mit Hilfe der CPU und der GPU anstellen, um die Effizienz der gemischten Implementation zu verdeutlichen.

5.1 Programmaufbau

Eine wesentliche Hilfe bei der Implementierung war die Programmbibliothek von Yuri Boykov und Vladimir Kolmogorov [BK], die den maximalen Fluss in einem Graph in polynomieller Zeit berechnet. Die Zeiten für diese Berechnung liegen in Millisekunden auf der von uns eingesetzten CPU (Pentium 4 mit 2.4GHz).

Die Energiefunktion, die wir zu minimieren versuchen, ist

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{(p,q) \in N} V_{p,q}(f_p, f_q) \tag{5.1}$$

wobei $D_p(f_p)$ der Datenterm aus Kapitel 3.5.2 und $V_{p,q}(f_p, f_q)$ der Nachbarschaftsterm aus Kapitel 3.5.1 sind.

Da die Nachbarschaftsindizes sich nicht verändern, d.h. die einzelnen Pixel bleiben in der gleichen Relation zueinander, war es möglich, diese in einer Datei zu speichern. Für die Nachbarschaft wählten wir die 8-er Nachbarschaft, d.h. für jedes Pixel $p_{i,j}$ ist das Pixel $q_{k,n}$ mit $i - 1 \leq k \leq i + 1$ und $j - 1 \leq n \leq j + 1$ Nachbar.

Jedes Bild wurde in einem 2D-Array gespeichert und beim Zugriff auf die einzelnen Werte innerhalb der x-Achse linear zwischen den Pixeln interpoliert. Die Bilder wurden in einem Array abgelegt, sodass ein Epipolarvolumen enstand. Dadurch wurde die Energiefunktion nicht für ein einzelnes Bild, stattdessen aber auf das gesamte Volumen angewendet. Da keine Nachbarschaftsbeziehungen zwischen den einzelnen Bildern bestanden, konnte ein gemeinsamer Graph für das ganze Epipolarvolumen erstellt werden. Auf diesem Graph wurde der minimale Schnitt berechnet und die Zuweisung des neuen Labels α ermittelt. Somit wurde eine Labelingzuweisung für das gesamte Volumen,

statt für ein einzelnes Bild errechnet. Dies machte sich selbstverständlich in der Laufzeit bemerkbar, da man nun einen Graphen für ein 3D-Bild und nicht für ein flaches 2D Gebilde erstellt. Insgesamt ist die Erstellung des Graphen ein speicherintensives Problem, sodass dies der Flaschenhals in der Geschwindigkeit des Algorithmus war.

Die Minimierung der Energiefunktion erfolgte in mehreren Iterationen (siehe Kapitel 4.5). Nach drei bis vier Iterationen wurde keine deutliche Verbesserung erzielt, sodass wir die Minimierung auf etwa drei Iterationen begrenzten.

5.1.1 Verdeckte Pixel

Durch die verdeckten Pixel entstehen in der errechneten Tiefenkarte Fehler. Diese Fehler kommen dadurch zustande, dass die sich zugehörigen Epipolarlinien überlagern (siehe Kapitel 3.4). Um das Problem zu behandeln, wenden wir ein leicht veränderten Datenterm an. Wir benutzen die Labelingfunktion $\hat{f}$, die wir zuvor errechnet haben und gehen nun folgendermaßen vor:

- Starte den Rekonstruktionsprozess erneut, aber nun mit $f_p = \lambda_{max}, \forall p \in \mathcal{P}$
- Der Schritt 3. aus dem α-Erweiterungsalgorithmus im Abschnitt 4.5 wird nun für jedes Label λ in umgekehrter Reihenfolge durchgeführt, d.h. $\alpha = \lambda_{max}, \lambda_{max-1}, ..., \lambda_{min}$
- Bei der Berechnung der Linie l_P im α-Erweiterungsschritt mit der Steigung λ_α werden die Pixel ausgenommen, welche die Gleichung $\hat{f}_p > \lambda_\alpha + M$ nicht erfüllen
- Berechne $D_p(l_P)$ wie zuvor, jedoch sind die verdeckten Pixel nun von der Berechnung ausgenommen

Die Konstante M ist wählbar und wird dazu benötigt, um den Rekonstruktionsprozess robuster gegenüber Ausschweifungen innerhalb des errechneten Tiefenvolumens zu gestalten. In unserem Fall ist M empirisch gewählt und entspricht 2. Der Wert sollte aber dennoch in einer Korrelation zu der Kardinalität von $\mathcal{L}$ stehen.

Durch die vorhergehende Berechnung der Tiefenkarte bzw. des Tiefenvolumens für jede Kamera können wir annehmen, dass die Energiefunktion bereits minimiert wurde und dass jedes Pixel die niedrigste Energiestufe angenommen hat. In der Berechnung des Datenterms der neuen Energiefunktion ignorieren wir die Pixel, deren berechnetes Label größer ist als das des zu berechnenden Pixels. Ist das Label $\hat{f}_p$ größer als λ_α, so schneiden sich die beiden Epipolarlinien $l_p^{\hat{f}_p}$ und $l_q^{\lambda_\alpha}$, was einer Verdeckung des Pixels q durch das Pixel p gleich kommt.

Für die hier vorgestellten Ergebnisse ist eine solche Annahme über die verdeckten Pixel völlig ausreichend. Die Minimierung der veränderten Energiefunktion nimmt im Durchschnitt ähnlich viel Zeit in Anspruch wie die Minimierung der unveränderten Funktion. Dafür werden aber die verdeckten Pixel nicht einfach aus der Berechnung entnommen, sondern in die Berechnung miteinbezogen und sofern sie wenigstens zwei mal innerhalb der Epipolarlinie auftauchen, ist die Berechnung von deren Tiefe auch möglich.

5.2 GPU Optimierung

Jeder heutige Computer verfügt über weit mehr Rechenkraft, als man auf Anhieb vermuten kann. Die aktuelle Generation der Grafikkarten erlaubt uns, deren Grafikprozessor für unsere Zwecke zu entfremden. Somit haben wir zusätzliche Rechenkraft, die trotz niedriger Taktraten im Vergleich zu der CPU, sehr schnell rechnen kann. Leider sind die GPUs nicht dazu gedacht, Berechnungen wie auf der CPU auszuführen. Stattdessen muss man immer in der Lage sein, die zu berechnenden Daten entsprechend vorzubereiten.

5.2.1 Programmierbare Grafikhardware

Die aktuelle Generation der Grafikkarten ist mit einer programmierbaren Pipeline für die Berechnung der Pixelwerte ausgestattet. Die Hersteller ermöglichen uns, Teile dieser Pipeline umzuschreiben (siehe auch [Mar]). Dies wird ausgiebig in den aktuellen Computerspielen benutzt, um grafisch anspruchsvolle Effekte herzuzaubern.

Durch das OpenGL-Interface sind wir auch in der Lage, die so berechneten Pixelwerte abzufragen und die Daten somit von der GPU in den Systemspeicher zu laden. Zudem können wir nicht nur 8-Bit Werte, sondern auch Fließkommazahlen für die einzelnen Farbwerte der Pixel zu verwenden. Das macht man sich zunutze, um die GPU für die eigenen, immer öfter auch wissenschaftlichen, Zwecke zu entfremden.

5.2.2 Energieberechnung

Die zuvor vorgestellte Energiefunktion wird durch die Erstellung eines entsprechenden Graphen und die Berechnung des maximalen Flusses minimiert. Schaut man sich die Konstruktion des Graphen genauer an, so fällt es einem auf, dass der Datenterm $D_p(f_p)$ bei jedem α-Erweiterungsschritt unabhängig von der Nachbarschaft für jeden Pixel des

Bildes nur einmal berechnet wird. Der Wert dieser Berechnung bestimmt das Gewicht der entsprechenden Kante t_p^{α} bzw. $t_p^{\bar{\alpha}}$.

Dies nutzen wir, um die Berechnung der Datenterme für jeden Pixel auf der GPU ausführen zu lassen. Wir erstellen eine 3D-Textur, die das Epipolarvolumen enthält, sowie zwei weitere 3D-Texturen, welche die aktuelle Labelingfunktion und die aktuelle Partitionierung der Pixel in $\mathcal{P}_{\alpha}$ und $\mathcal{P}_{\bar{\alpha}}$ enthalten (siehe auch 4.5.2). Wir füttern die GPU mit diesen Daten und lassen sie für jeden Pixel $p \in \mathcal{P}$ innerhalb des ganzen Volumens V den Datenterm D_p berechnen.

Da die Anzahl der Kanten mit dem Nachbarschaftsterm $V_{p,q}$ in jedem α-Erweiterungsschritt nicht konstant ist, wird die Berechnung der Terme weiterhin auf der CPU ausgeführt. Die Vorbereitung der Daten für die GPU würde gleich viel bzw. mehr Zeit in Anspruch nehmen und steht in keiner Relation zu der Einfachheit der Berechnung des Nachbarschaftstermes für jedes Pixelpaar $(p, q) \in N$.

5.2.3 GPU vs CPU

Durch die Verwendung von GPU konnten wir eine immense Geschwindigkeitssteigerung erzielen. Im Durchschnitt lag sie auf unserem System bei Faktor drei. Für die Berechnung der Tiefenkarte des gesamten Volumens verwendeten wir einen Intel Pentium Prozessor mit 2.4 GHz und 2GB RAM. Als Grafikkarte kam die AGP Version von Nvidia's GeForce 6800 GT (NV45) zum Einsatz. Der Geschwindigkeitszuwachs durch die Verwendung von GPU ist in der unterstehenden Tabelle ersichtlich.

Auflösung	*dT CPU*	*dT GPU & CPU*
128x128x10	55 sek	22 sek
256x256x10	217 sek	71 sek
512x512x10	749 sek	233 sek

Tabelle 5.1: Laufzeiten des Algorithmus im Vergleich. Die aufgeführte Zeit entspricht einer Iteration mit 16 Labels für die Berechnung des Tiefenvolumens der Elefantenszene mit 10 Bildern.

5.3 Zusammenfassung

In diesem Kapitel haben wir uns damit befasst, wie die Implementation des zuvor vorgestellten Algorithmus auf unserer Hardware vollbracht wurde. Durch die Verwendung

eines leicht geänderten Datenterms der Energiefunktion waren wir in der Lage, mit Hilfe der bereits berechneten Labelingfunktion auch die Pixel, die verdeckt sind, zu rekonstruieren. Die Verwendung der programmierbaren Pixeleinheit der GPU erlaubt uns eine Beschleunigung der Berechnung, die merkbar ist. Am Schluss zeigen wir mehrere Ergebnisbilder, die mit Hilfe unseres Algorithmus berechnet wurden. Für die Bilder, die künstlich erzeugt wurden, wird auch ein Ground-Truth-Vergleich durchgeführt, um die Abweichungen der errechneten Tiefenwerte zu bestimmen.

5.4 Ergebnisse

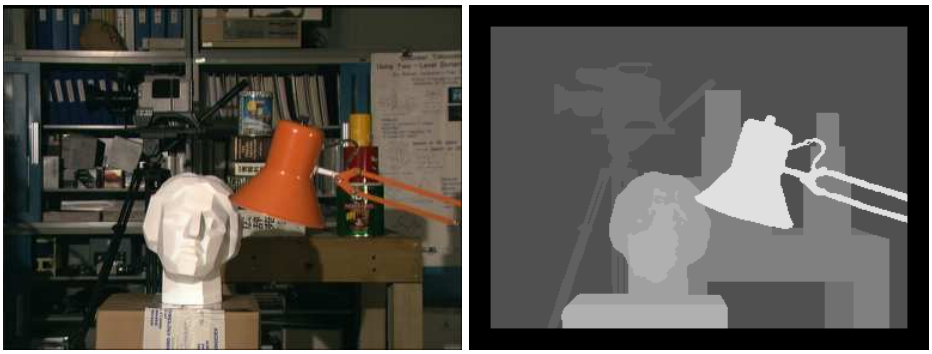

Abbildung 5.1: *Links*: Eines der Bilder für die Rekonstruktion (aus dem Tsukuba Datensatz). *Rechts*: Ground Truth des Bildes.

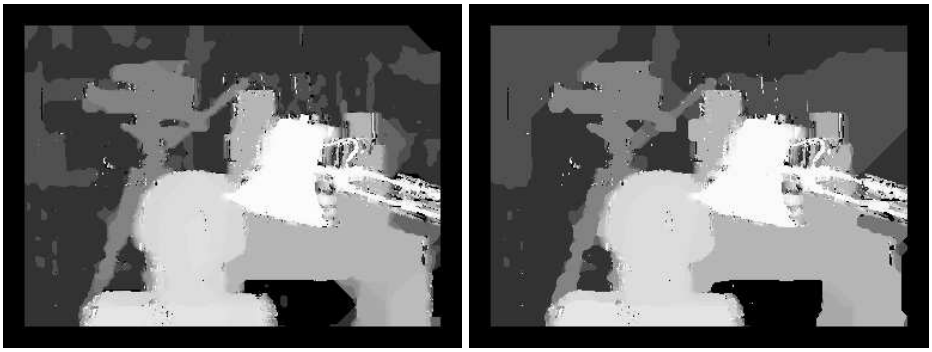

Abbildung 5.2: Rekonstruktion mittels unseren Algorithmus mit 16 Labels. Man beachte, sogar die Tiefe der Bücher im Regal wurde zum Teil rekonstruiert. *Links*: schwacher Nachbarschaftsterm. *Rechts*: starker Nachbarschaftsterm führt zu weniger detaillierten Ergebnissen.

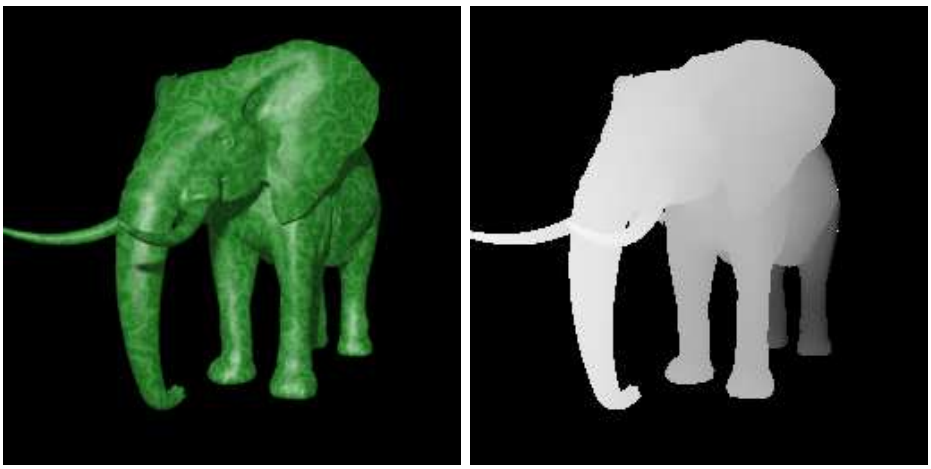

Abbildung 5.3: *Links*: Ein Bild aus der Elefantszene. *Rechts*: Ground Truth der Szene.

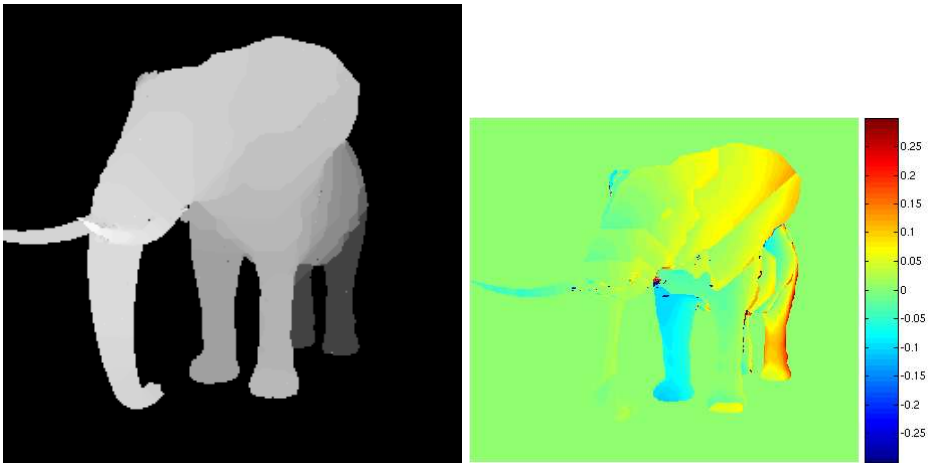

Abbildung 5.4: *Links*: Rekonstruktion mittels unseren Algorithmus mit 16 Labels. *Rechts*: Fehlertoleranz zwischen Ground Truth und rekonstruiertem Werten.

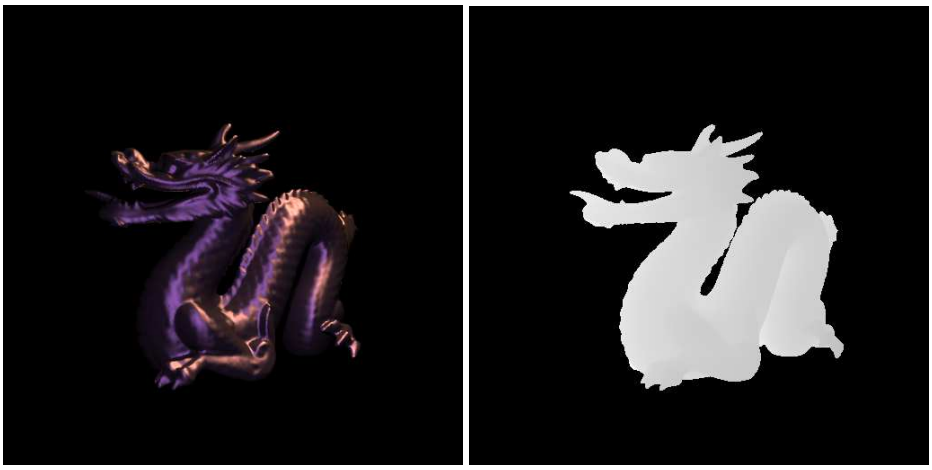

Abbildung 5.5: *Links*: Originalbild mit einem Drachen. *Rechts*: Ground Truth der Szene.

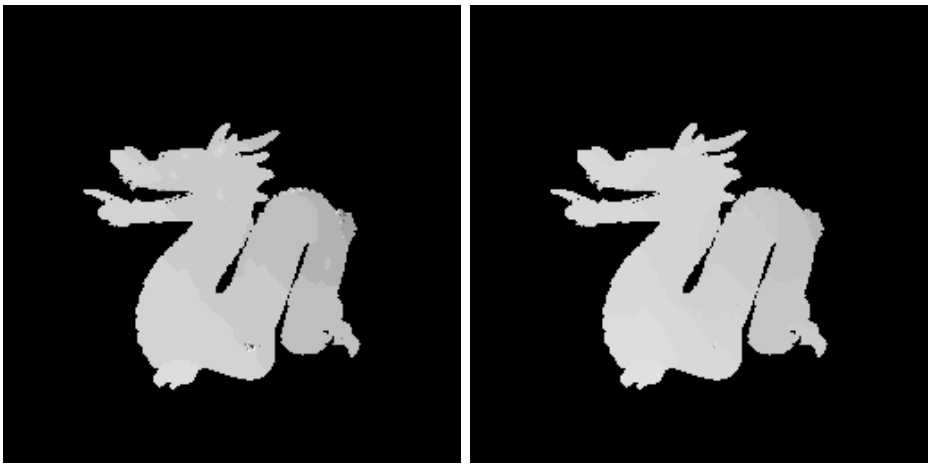

Abbildung 5.6: Rekonstruktion mittels unseren Algorithmus. *Links*: Rekonstruktion mit nur 16 Labels. *Rechts*: Rekonstruktion mit 32 möglichen Tiefenwerten.

KAPITEL 6

ZUSAMMENFASSUNG

In dieser Arbeit haben wir gezeigt, wie man mit Hilfe des Graph-Cut-Ansatzes aus mehreren Bildern die Tiefeninformationen berechnen kann. Unser Ansatz kann im Vergleich zu vielen anderen aktuellen Ansätzen auch mit nicht lambert'schen Oberflächen, wie Metall oder Plastik, hervorragend umgehen. Die Ergebnisse, die wir im Kapitel 5.4 präsentiert haben, zeigen den Algorithmus in Arbeit. Außer den künstlichen Szenen, wie Buddha oder Elefant, haben wir die realen, zur Verfügung gestellten Datensätze, in die Tiefe hin rekonstruieren lassen.

Die Verwendung der aktuellen Grafikhardware in der Implementierung des Algorithmus lieferte eine bis zu dreifache Geschwindigkeitssteigerung gegenüber rein CPU basierter Rechnung. Berechnet man die Tiefenkarte für eines der Bilder innerhalb des Epipolarvolumens statt für das ganze Volumen, so ist eine weitere Geschwindigkeitssteigerung zu erwarten. Dies beruht auf der verwendeten Bibliothek [BK], dessen Laufzeit linear mit der Auflösung des Volumens bzw. der Größe des Graphen steigt.

In einer der zukünftigen Arbeiten kann zudem, gestützt auf den hier beschriebenen Ansatz, die Abschätzung der Materialien untersucht werden. Der Verlauf der Intensitätswerte innerhalb einer Epipolarlinie ermöglicht uns die BRDF des jeweiligen Oberflächenpunktes abzuschätzen, sofern die Position der Lichtquellen relativ zu den Kameras bekannt ist. Dies eröffnet uns eine Möglichkeit die Materialeigenschaften der Objekte zu bestimmen. Die zuvor errechnete Tiefenkarte ermöglicht die Bestimmung der Normalen. Dadurch wäre man in der Lage, die Szene samt Geometrie und Materialien zu rekonstruieren.

Wir beenden unsere Arbeit mit den Worten, mit den wir sie angefangen haben: "Am Anfang war das Licht." Nun wissen wir, dass die Wissenschaft von heute immer näher an das erklärte Ziel heranrückt, die Sehfähigkeit des Menschen nachzuahmen, ja sogar diese zu übertreffen. Das Licht hingegen ist und bleibt unentbehrlich zur Wahrnehmung unserer Welt.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

[AB91] E. H. Adelson and J. R. Bergen. *Computational Models of Visual Processing*, chapter The Plenoptic Function and the Elements of Early Vision, pages 3–20. MIT Press, 1991.

[AHK89] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. *Illinois: Journal of Mathematics: vol.21*, pages 439–567, 1989.

[BK] Y. Boykov and V. Kolmogorov. The MAXFLOW algorithm. http://www.cs.cornell.edu/People/vnk/software.html.

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of mincut/max-flow algorithms for energy minimization in vision. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 1124–1137, 2004.

[BN95] D. Bhat and S Nayar. Stereo in the Presence of Specular Reflection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1086–1092, 1995.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, pages 1222–1239, 2001.

[Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, pages 1277–1280, 1970.

[DJP+92] E. Dahlhaus, S.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiway cuts. *ACM Symp. Theory of Computing (STOC)*, pages 241–251, 1992.

[FF62] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[GY01] M. Gong and Y.H. Yang. Multi-resolution stereo matching using genetic algorithm. In *IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV), Kauai Marriott, Kauai, Hawaii*, pages 21–29, 2001.

[GY03] M. Gong and Y.-H. Yang. Fast stereo matching using reliability-based dynamic programming and consistency constraints. In *IEEE International Conference on Computer Vision (ICCV)*, pages 610–617, 2003.

[HZ00] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[JSY03] H. Jin, S. Soatto, and A. Yezzi. Multi-View Stereo Beyond Lambert. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 171–178, 2003.

[KZ02] V. Kolmogorov and R. Zabih. Multi-camera Scene Reconstruction via Graph Cuts. In *European Conference on Computer Vision (ECCV)*, pages 82–96, 2002.

[KZG03] V. Kolmogorov, R. Zabih, and S. J. Gortler. Generalized multi-camera scene reconstruction using graph cuts. In *Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, pages 501–516, 2003.

[Mar] B. Mark. NVIDIA Programmable Graphics Technology. http://developer.nvidia.com/attach/6554.

[SvMG04] H. Sunyoto, W. v.d. Mark, and D. M. Gabrila. A Comparative Study of Fast Dense Stereo Vision Algorithms. *IEEE Intelligent Vehicle Symposium (IV), Parma, Italy*, pages 319–324, 2004.

[Uni] Stanford University. The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep.

[Vek99] O. Veksler. *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University, 1999.