FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Echtzeitansätze in der Stereoanalyse

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur/in**

im Rahmen des Studiums

**Medieninformatik**

eingereicht von

**Bakk. techn. Clemens Czermak**
Matrikelnummer 0126610

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuerin: Ao.Univ.Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz
Mitwirkung: Dipl.-Ing. Dr. Michael Bleyer

Wien, 18.11.2011 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

**Erklärung zur Verfassung der Arbeit**

Clemens Czermak

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 18.11.2011 _____

(Unterschrift Verfasser/in)

## Kurzfassung

Die Diplomarbeit beschäftigt sich mit der Stereoanalyse zur Gewinnung von Tiefeninformation aus Bildern. Es werden Anwendungsgebiete und Grundlagen der Stereoanalyse beschrieben und häufig gemachte Annahmen, welche die Korrespondenzsuche erleichtern, diskutiert. Bei den Verfahren zur Lösung des Korrespondenzproblems werden lokale Methoden sowie globale Methoden erörtert. Neben der theoretischen Erläuterung von Verfahren ist das Ziel der Arbeit die Implementierung eines Stereoalgorithmus auf einer Graphikkarte, um mithilfe der starken Parallelisierungsfähigkeiten der Karte einen Geschwindigkeitsvorteil gegenüber einer CPU-Implementierung zu erreichen.

Bei der theoretischen Aufbereitung werden vor allem jene Stereoverfahren berücksichtigt, welche eine Laufzeit besitzen, die für Echtzeitanwendungen geeignet ist und dennoch Resultate vergleichbar mit aufwendigeren Verfahren vorweisen. Zu diesen Methoden zählen bei den lokalen Ansätzen jene mit adaptiver Anpassung der Fenster bzw. der Gewichte an die Gegebenheiten der Bilder. Bei den globalen Verfahren wird der Ansatz der dynamischen Programmierung geschildert, da sich Verfahren mit dynamischer Programmierung effizient auf einer Graphikkarte implementieren lassen und gute Resultate liefern. Eigens erläutert wird jener auf dynamischer Programmierung basierende Stereoalgorithmus, welcher dann auf einer GPU implementiert wird.

Im praktischen Teil werden Aufbau und Architektur von programmierbaren Graphikprozessoren beschrieben und das Vorgehen bei der realisierten GPU-Implementierung wird erläutert. Die erzielten Ergebnisse der GPU-Implementierung werden anschließend diskutiert sowie mit den Ergebnissen der CPU-Implementierung verglichen. Das implementierte Stereoverfahren läuft mit 16 Bildern pro Sekunde bei Bildern der Größe 384x288 und 16 Disparitätsstufen, was für viele Echtzeitanwendungen ausreichend ist.

## Abstract

This work deals with stereo vision for obtaining depth information from image pairs. It describes applications and basic principles of stereo vision as well as common assumptions that simplify the correspondence search. Among the methods for solving the correspondence problem, local and global methods are discussed. Beside the theoretical explanation of efficient methods, the aim of the work is the implementation of a stereo algorithm on a graphics card to reach a speed advantage compared to a CPU implementation.

In the theoretical treatment we consider especially those stereo methods which are suitable for real-time applications while showing results that are comparable to those of more complex algorithms. In the field of local approaches, this includes the adaptive adjustment of the window size or a shape by suitable adjustment of weights. In the field of global approaches, dynamic programming (DP) is described in detail, since it produces good results and can be efficiently implemented on graphics cards. In particular, we explain a tree-based dynamic programming algorithm which is then implemented on a GPU.

The design and architecture of programmable graphics processors are examined and the steps of the realized GPU implementation are described. The results of the GPU implementation are analysed and compared against the results of the CPU implementation and ground truth data. The implemented stereo method runs at 16 frames per second, which is sufficient for many real-time applications.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Wir Menschen nehmen unsere Umwelt räumlich wahr und können uns dementsprechend gut in ihr bewegen und mit ihr interagieren. Unser Verstand verarbeitet für diese Wahrnehmung eine Vielzahl von Hinweisen und Reizen. Diese stammen in erster Linie von den Sinneseindrücken der Augen. Es werden allerdings auch von uns erworbene Kenntnisse über die Welt und ihre Beschaffenheit berücksichtigt. Der Gesamtprozess des räumlichen Wahrnehmens erstreckt sich von der Erfassung der Bilder im Auge über die Weiterleitung zum Gehirn zu dessen Interpretation. Er ist bis heute aufgrund der Komplexität noch nicht vollständig entschlüsselt. Das erschwert den Versuch, die menschliche Wahrnehmung nachzubauen oder zu simulieren, um Maschinen räumliches Sehen beizubringen. Abgesehen von einer subjektiven Wahrnehmung, die sich von Mensch zu Mensch unterscheidet und durch Kulturkreis und persönliche Erfahrungen geprägt ist, gibt es eine Menge von objektiven Hinweisen, die wir für die räumliche Wahrnehmung verwenden. Diese können sowohl monokular wie auch binokular sein. Monokulare Indizien für Räumlichkeit sind unter anderem die Perspektive, die Verdeckung oder atmosphärische Unschärfe. Bei binokularen Hinweisen spielen vor allem die Differenzen in den beiden von den Augen erfassten Bildern eine bedeutende Rolle. Wohl eine der wichtigsten dieser Differenzen ist die binokulare Disparität. Sie bezeichnet den Lageunterschied von Objekten im linken und rechten Auge, hervorgerufen durch die leicht verschobenen Positionen der Augen. Vom Betrachter weit entfernte Gegenstände befinden sich an fast exakt derselben Stelle im anderen Bild, nahe Gegenstände erfahren hingegen eine stärkere horizontale Deplatzierung [1].

Diese binokulare Disparität verwendet man bei der statischen Stereoanalyse[1] für die Gewinnung von Tiefeninformation. Die statische Stereoanalyse ist ein Teilgebiet des Maschinellen Sehens und der digitalen Bildverarbeitung. Für die Analyse benötigt man mindestens zwei Bilder, welche die gleiche Szene aus leicht unterschiedlichen Standpunkten im selben Moment zeigen, wie es auch beim menschlichen Sehen der Fall ist. Zur Ermittlung der Disparitäten in den Bildern ordnet man Pixel im linken Bild den korrespondierenden Bildpunkten im

1 Bei der dynamischen Stereoanalyse [2] werden hingegen mögliche Bewegungen miteinbezogen. In dieser Arbeit wird nur die statische Stereoanalyse betrachtet.

rechten Bild zu. Man möchte jene Pixel in Verbindung bringen, welche die Projektion derselben Oberflächenregion aus der realen Welt darstellen. Anschließend betrachtet man den Lageunterschied der korrespondierenden Pixel. Dieser Lageunterschied ist proportional zum Abstand der Oberflächenregion von der Stereokamera und erlaubt eine Tiefenmessung.

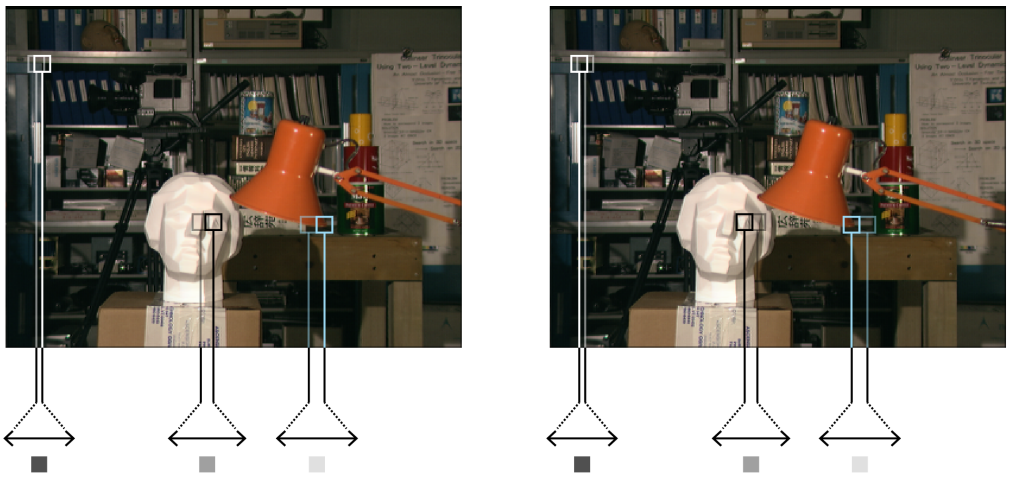
Dass räumliches Sehen alleinig unter Zuhilfenahme der binokularen Disparität möglich ist, zeigen die Experimente von Julesz [3]. Er versuchte durch computergenerierte Zufallsbilder, sogenannte Zufallsmusterstereogramme, monokulare Hinweise weitestgehend zu eliminieren, damit nur die binokulare Disparität für die räumliche Wahrnehmung zum Tragen kommt. Betrachtet man diese erzeugten Bilder stereoskopisch, ist ein Tiefeneindruck erkennbar. Einzig die Wahrnehmung des Eindruckes verzögert sich mitunter etwas. Daraus schloss Jualesz, dass monokulare Indizien für die Tiefenwahrnehmung zwar hilfreich sind, aber nicht essentiell. Nichtsdestoweniger werden bei Versuchen, räumliche Information nur mithilfe der Disparität abzuleiten, eine Vielzahl von anderen Indizien beiseitegelassen, deren Kombination bei uns Menschen zur Wahrnehmung der Tiefe verwendet wird [4][5][6]. Allerdings sind monokulare Hinweise für sich alleine weniger exakt und oft mehrdeutig [7].

Das Problem des Auffindens von korrespondierenden Elementen während der Stereoanalyse heißt Korrespondenzproblem. Es zählt zu den schwierigsten Teilen, da meistens keine eindeutige Lösung existiert. Die Korrespondenzsuche involviert zwei wichtige Aspekte, die geklärt werden müssen [8]:

(1) Welche Elemente sollen zugeordnet werden? Verlässt man sich auf Pixelintensitäten, Kanten, Regionen oder andere Merkmale?

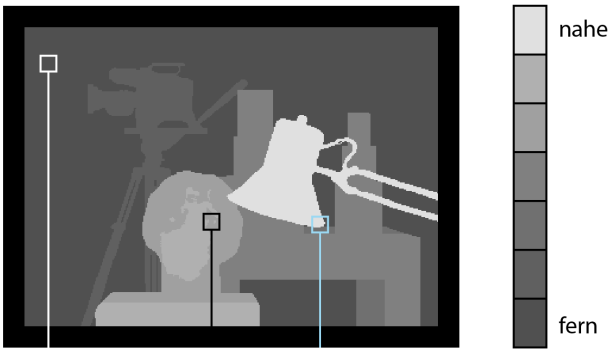
(2) Wie ordnet man Elemente in einem Bild Elementen im anderen Bild zu? Welche Indikatoren geben die Gleichheit zweier Merkmale am besten wieder? Welche Zuordnungsverfahren sind für einen bestimmten Einsatzzweck am geeignetsten?

Um überhaupt Lösungsansätze formulieren zu können, müssen bestimmte Annahmen gemacht werden, welche die Problemstellung vereinfachen, aber nicht immer zutreffen. In Kapitel 2 wird auf diese Annahmen noch genauer eingegangen. Das Ergebnis der Stereoanalyse ist eine sogenannte Disparitätskarte (oder Tiefenbild), die für jeden Bildpunkt dessen Disparität angibt. Disparitätskarten werden üblicherweise als Graustufenbilder codiert, wobei der Grauwert die Disparität widerspiegelt (Abb. 1). In der Regel verwendet man zwei Bilder für die Analyse, es können aber auch mehrere Bilder eingesetzt werden.



a) linkes Stereobild

b) rechtes Stereobild




c) Disparitätskarte

Abbildung 1: Bei der Korrespondenzsuche ordnet man Pixel im linken Bild den korrespondierenden Pixeln im rechten Bild zu und umgekehrt. Die zugeordneten Bildelemente sollten das Abbild des gleichen 3-D-Punktes einer Szene sein. In a) und b) werden drei ausgewählte Pixel aus Gründen der Übersicht durch ein Quadrat repräsentiert, wobei jeweils das Pixel im Zentrum des Quadrates gemeint ist. Die leicht transparenten Quadrate signalisieren die Position des Pixels im anderen Bild. Die Verschiebung der Lage aufgrund der unterschiedlichen Perspektiven bezeichnet man als binokulare Disparität. Sie lässt Rückschlüsse auf die Tiefe eines Pixels zu. Die Disparitäten visualisiert man üblicherweise in einem Grauwertbild (siehe c)).

## 1.2 Zielsetzung

Der theoretische Teil der Arbeit soll einen fundierten Einblick in den aktuellen Stand der Technik auf dem Gebiet der Stereoanalyse liefern, wobei der Fokus auf potentiell schnellen Verfahren liegt. Es werden gegebenenfalls wichtige Vorgängermethoden präsentiert, um die

Entwicklung gewisser Verfahren nachvollziehbar zu machen. Auch der Einsatzbereich derartiger Verfahren möchte analysiert sein. Zum Erreichen dieser Zielsetzung wurden State of the Art Artikel auf dem genannten Gebiet studiert und analysiert. Ein vollständiges Studium aller Veröffentlichungen war aufgrund der überwältigenden Menge nicht möglich, es werden aber die bedeutendsten Strömungen und Richtungen erfasst und diskutiert.

Um der Entwicklung der letzten Jahre hin zur parallelen Programmierung Rechnung zu tragen, beschäftigt sich der praktische Teil der Arbeit mit der Implementierung eines Stereoverfahrens auf Graphikhardware. Im Ressort der Informatik vollzog sich sowohl im Bereich der Hardware als auch im Softwarebereich ein kleiner Paradigmenwechsel. Rechnete früher ein Rechenkern in einem Computer und wurden die Programme für die Ausführung auf diesem einem Kern optimiert, befinden sich heute mehrere Rechenkerne in einem Computer und Programmierer müssen auf die Parallelisierbarkeit ihrer Programme achten. Dieser Trend verschärfte sich noch durch die Einführung programmierbarer Graphikhardware, welche per se schon auf parallele Ausführung ausgelegt ist. Vor allem bei laufzeitkritischen Anwendungen wird versucht, das Potential paralleler Verarbeitung auszuschöpfen. Damit soll ein deutlicher Geschwindigkeitsgewinn verbunden sein. Für die Implementierung wird CUDA[2] verwendet. Die Ergebnisse finden sich am Ende der Arbeit präsentiert.

## 1.3 Organisation

Die Arbeit beginnt mit der Untersuchung einiger Anwendungsgebiete der Stereoanalyse in Kapitel 1.4. Hierbei werden auch alternative Technologien zum Erlangen von Disparitätskarten vorgestellt. Anschließend werden die grundlegenden Begriffe und Sachverhalte der Stereoanalyse in Kapitel 2 näher erläutert. Es wird geklärt, wie sich aus der Disparität die Tiefe bestimmen lässt und welche Annahmen man bei Stereoverfahren üblicherweise macht. Die allgemeinen Schritte bei Lösungsverfahren für das Korrespondenzproblem werden ebenfalls vorgestellt. Kapitel 3 beschreibt lokale Methoden, weil die Mehrzahl der bisher realisierten Echtzeitsysteme lokale Stereoalgorithmen einsetzen. Sowohl die adaptive Anpassung von Fenstern wie auch der Gebrauch von Gewichten werden erklärt. Kapitel 4 beschäftigt sich genauer mit dem Optimierungsverfahren der dynamischen Programmierung. Den Ansatz von dynamischer Programmierung entlang von Zeilen sowie den Ansatz von dynamischer Programmierung auf einem Baum werden hier diskutiert. Der Grund dafür ist, dass der in der Arbeit implementierte Stereoalgorithmus ([9]) beide Ansätze nutzt und sie miteinander kombiniert. Die An-

2 CUDA (Compute Unified Device Architecture) bezeichnet das GPGPU-Konzept von Nvidia.

sätze sind demnach für das Verständnis des Algorithmus wichtig. Diesen Algorithmus beschreibt anschließend Kapitel 5 ausführlich. In Kapitel 6 wird das Vorgehen bei der Implementierung präsentiert und auch über den Gebrauch von Graphikhardware für die Beschleunigung von Algorithmen wird referiert. Die Ergebnisse der GPU-Implementierung diskutiert Kapitel 7. Mit einer Zusammenfassung der wichtigsten Punkte und Resultate (Kap. 8) schließt die Arbeit ab.

## 1.4 Anwendungen

Es existieren vielfältige Anwendungsmöglichkeiten für maschinelles Stereosehen mittels Stereoanalyse. Jede Anwendung besitzt dabei ihre eigenen Anforderungen an einen Stereoalgorithmus, welcher zur Berechnung der Disparitäten herangezogen wird. Die Berechnung der Disparitätskarten bildet bei vielen Anwendungen nur einen Zwischenschritt für die Erreichung des gewünschten Endergebnisses. Im Folgenden werden einige wichtige Anwendungsgebiete vorgestellt.

#### Dreidimensionale Rekonstruktion

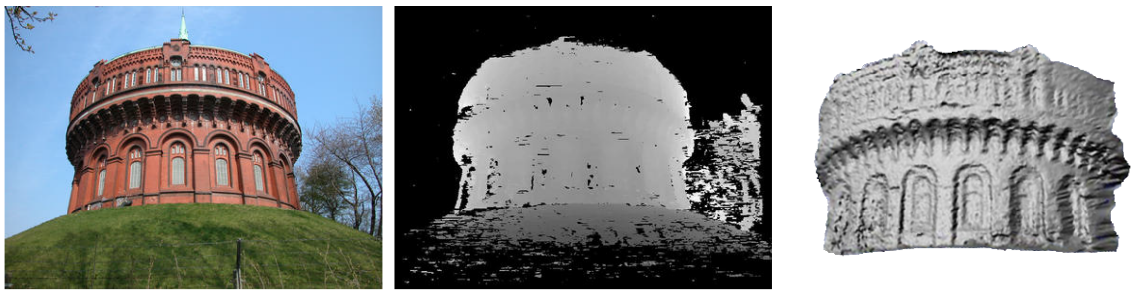
Das Gebiet der dreidimensionalen Rekonstruktion aus Bildern erfuhr in den letzten Jahren starke Aufmerksamkeit, weil die Nachfrage an wirklichkeitsgetreuen 3-D Modellen anstieg. Die dreidimensionale Rekonstruktion ist ein traditionelles Gebiet des Maschinellen Sehens und der Photogrammetrie. Ziel ist es, mithilfe der aus den Stereobildern gewonnenen Tiefeninformation die abgelichtete Szene nachzubilden. Für viele Stereoanalysesysteme wünschenswert wäre eine vollständige dreidimensionale Wiederherstellung der abgebildeten Szene. Eine zumindest teilweise Wiederherstellung wird bei vielen Systemen erreicht. Dafür ist bei komplexen Objekten oder Szenen mehr als ein Stereobildpaar eines Standpunktes erforderlich. Man verwendet dafür ganze Stereobildsequenzen, welche mehrere Stereobilder der Szene von unterschiedlichen Standpunkten beinhalten. Koch zeigt in [10], dass diese Sequenzen zum Beispiel aus Filmaufnahmen mit einer handelsüblichen Kamera gewonnen werden können. Die dazu notwendigen Kameraparameter lassen sich schätzen. Für eine Rekonstruktion wertet man die Disparitätskarten der einzelnen Ansichten aus und kombiniert die gewonnenen Tiefeninformationen miteinander. Auf diese Weise erhält man ein dreidimensionales Modell. In der Regel wird dabei eine statische Szene vorausgesetzt, um ein konsistentes 3-D Modell der Szene zu erlangen [11][12].

### Rekonstruktion von (urbanen) Gebieten

Der Bedarf an realistischen 3-D-Modellen von urbanen Gebieten ist seit Applikationen wie Google Earth und Bing Maps[3] kräftig gestiegen. Diese Applikationen haben sich zum Ziel gesetzt, die reale Welt virtuell nachzubauen. Neben der Verwendung in virtuellen Realitäten finden sich noch weitere Einsatzmöglichkeiten von urbanen 3-D-Modellen unter anderem in der Stadtplanung, dem Tourismus und der Unterhaltungsindustrie. Zur Unterhaltungsindustrie zählen der Film- und Videospielsektor. Der Mobilfunkindustrie dienen 3-D-Modelle beispielsweise zur Ausbreitungssimulation von Radiowellen. Mit den Modellen lassen sich ebenfalls Feuersimulationen und andere Sicherheitsstudien realisieren[13][14][15]. Die Autoren von [16] verwenden sie für die Verbesserung der Visualisierung von Navigationsgeräten. Die Rekonstruktion kann auch zur dreidimensionalen Erfassung von archäologischen Ausgrabungsstätten herangezogen werden. Sie unterstützt hierbei nicht nur das Darstellen der Stätten selbst, sondern kann auch zur Dokumentation des Ausgrabevorganges verwendet werden [17]. Hogue et al. nutzen die dreidimensionale Rekonstruktion zur Nachbildung von Unterwasserumgebungen, wie bei gesunkenen Schiffswracks oder Korallenriffen. Dadurch kann zum Beispiel der Gesundheitszustand von Riffen automatisch registriert und analysiert werden [18].

Grundsätzlich können für die Rekonstruktion Bodenaufnahmen, Luft- und Satellitenbilder eingesetzt werden. Luftbilder liefern genaue Gebäudegrundrisse und gute Höhenschätzungen von Gebäuden, es fehlt ihnen aber an Fassadeninformation. Bodenaufnahmen liefern sehr genaue Fassadendetails der Gebäude, zeigen jedoch keine Information von Dächern. Folglich erstellen Verfahren, welche Luftbilder und Bodenaufnahmen miteinander kombinieren, die genauesten Modelle. Abbildung 2 a) zeigt die Aufnahme eines Turmes aus Backstein, die dazu passende Disparitätskarte und das rekonstruierte 3-D Modell ohne Textur.

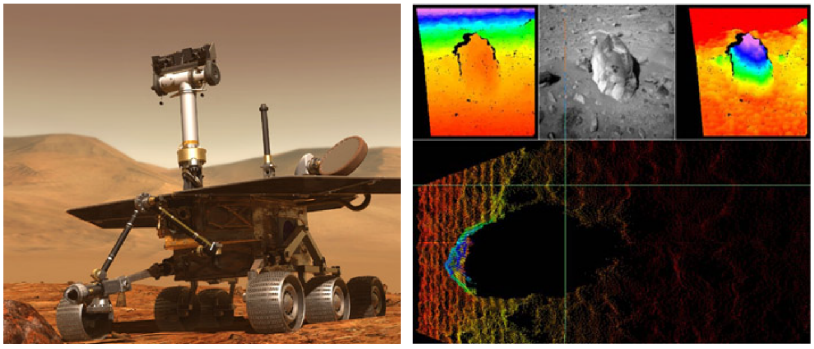
3 Bing Maps hieß bis Juni 2009 Microsoft Virtual Earth.



a) 3-D-Rekonstruktion

b) Generierung virtueller Ansichten

c) Automatische Kartographie



d) Navigation von Robotern und autonomen Vehikeln

Abbildung 2: Tiefeninformation, gewonnen mithilfe der Stereoanalyse, kann für unterschiedliche Anwendungen und Einsatzgebiete genutzt werden. Quellen: a) [19][20] b)[25] c)[22] d)[21][34].

### Generierung virtueller Ansichten

Eine bestimmte Szene aus verschiedenen Blickwinkeln betrachten zu können, war lange Zeit nur bei computergenerierten Szenen möglich. Benutzer können diese virtuellen Szenen nach Belieben durchschreiten und begutachten. Doch neue Applikationen und Techniken, wie die videobasierte Telekonferenz, Videos mit frei wählbarem Standort (engl. *free viewpoint videos*) oder 3-D-Videos, erfordern es, künstlich generierte Ansichten von realen Szenen zu erzeugen. Die reale Szene soll dabei so erscheinen, als hätte man sie von einem bestimmten virtuellen Standpunkt aus aufgenommen. Hierfür existieren zwei Ansätze: die modellbasierte Synthese und die bildbasierte Synthese.

Bei beiden Ansätzen erfasst man die Szene zuerst mit Stereokameras und kalkuliert die Disparitätskarten. Die modellbasierte Synthese erstellt anschließend mittels Algorithmen der dreidimensionalen Rekonstruktion ein explizites 3-D-Modell der Szene. Mithilfe des erstellten Modells können weitere neue Ansichten generiert werden. Allerdings ist der Aufwand für die Erstellung eines expliziten Modells nicht immer notwendig. Es reicht aus, wenn die Geometrie des abgebildeten Szenariums nur implizit via Disparitätskarten gegeben ist. Diese implizit gegebene Geometrie benutzt die bildbasierte Synthese zur Erstellung neuer Ansichten und spart sich dadurch die Konstruktion expliziter Modelle [23][24][25].

Die Generierung virtueller Ansichten funktioniert heutzutage mitunter in Echtzeit dank der Leistungsfähigkeit heutiger Computer. Eine Anwendung, welche Echtzeitverarbeitung benötigt, ist die videobasierte Telekonferenz. Weil Kameras nur am Rande des Bildschirmes positioniert werden können, blicken Konferenzteilnehmer nicht in die Kamera, sondern üblicherweise auf den Bildschirm. Folglich kommt es zu keinem Augenkontakt zwischen den Konversationspartnern. Der nicht vorhandene Augenkontakt stört den natürlichen Eindruck der Konversation per Video. Auch andere wichtige Interaktionshinweise wie Gesten und Kopfhaltung werden perspektivisch verzerrt und nicht korrekt angezeigt. In [26] und [27] verwenden die Autoren Verfahren der bildbasierten Synthese zur Erzeugung von Ansichten, die Teilnehmer so zeigen, als würden sie direkt in die Kamera blicken ergo auf den anderen Teilnehmer. Gesten und Kopfhaltung werden bei dem Verfahren ebenfalls korrigiert.

Bei Videos mit frei wählbarem Standort [28] benutzt man bei der Erstellung eines Filmes mehrere Kameras auf verschiedenen Positionen. Mithilfe dieser Positionen lassen sich später neue virtuelle Ansichten interpolieren. Das ermöglicht es, beispielsweise bei der DVD einer Oper oder eines Konzertes, seinen Standort frei zu wählen, von dem aus man die Aufführung verfolgen möchte.

In Abbildung 2 b) sind die Schritte zur Generierung virtueller Ansichten visuell dargestellt. Ganz links befindet sich das Referenzbild, daneben das Tiefenbild. Das dritte Bild zeigt die erstellte virtuelle Ansicht ohne Auffüllen verdeckter Regionen. Ganz rechts ist die neue Ansicht mit aufgefüllten verdeckten Regionen zu sehen.

### Automatische Kartographie

Es existiert eine große Anzahl von Luftaufnahmen vieler Gebiete unseres Planeten, größtenteils von Flugzeugen oder von Satelliten aufgenommen. Anhand der aus der Stereoanalyse gewonnenen Disparitätskarten können topographische Karten oder digitale Geländemodelle erstellt werden. Das Arbeitsfeld der automatischen Kartographie gehört zur Photogrammetrie, welche sich mit der Rekonstruktion der Lage und Form von Objekten vor allem aus Luftbildern beschäftigt. Es ist ein ideales Einsatzgebiet für Stereoalgorithmen. Zur Aufnahme der Gebiete werden Spezialkameras, auch Messkameras genannt, eingesetzt, damit die photogrammetrische Auswertung der Bilder verhältnismäßig einfach bleibt [29]. In Abbildung 2 c) werden generierte Geländemodelle von einer ländlichen und einer urbanen Umgebung gezeigt. Links befindet sich jeweils die Referenzluftaufnahme, gefolgt von dem digitalen Oberflächenmodell (DOM) und dem digitalen Geländemodell (DGM). Das DOM zeigt die Oberfläche unter Einbeziehung von Bauten und Vegetation. Das DGM zeigt die Oberfläche ohne Bauten und Vegetation [29].

Die automatische Kartographie stellt keine hohen Anforderungen an Stereoalgorithmen. Luftaufnahmen sind generell stark texturiert. Sie beinhalten kaum Verdeckungen und eine Abarbeitung in Echtzeit spielt keine Rolle. Diese Umstände erleichtern die Korrespondenzanalyse. Das führt zu exakten Karten und Geländemodellen, deren Genauigkeit im Bereich von einigen Dezimetern und noch besser liegt, abhängig von den Aufnahmebedingungen und dem Aufnahmebereich. Trotz großer Fortschritte seit Beginn der Photogrammetrie ist eine vollständige Automatisierung der Erstellung von Karten ganz ohne menschliches Mitwirken noch nicht möglich. Vor allem die automatische Merkmalsextraktion für die Korrespondenzsuche stellt ein Problem dar [24][30][31].

### Navigation von Robotern und autonomen Vehikeln

Tiefeninformation verwendet man ebenfalls für die selbständige Navigation bei Robotern und autonomen Vehikeln. Die Navigation besteht in der Robotik aus drei Gebieten: Lokalisierung, Pfadplanung und Robotersteuerung. Der Roboter muss also wissen, wo er sich befindet, wie

er weiteres Vorgehen plant und welche Gelenke oder Räder er bewegen muss, um ein bestimmtes Ziel zu erreichen. Mithilfe der Tiefenbilder können Roboter Hindernisse erkennen, diese in einen Plan eintragen und sie umfahren. Die Tiefenbilder berechnet man aus Stereobildern, welche von einem am Roboter montierten Stereokamerasystem stammen [32].

Dank der Tiefenbilder ist auch eine simultane Generierung von 3-D-Geländemodellen der erkundeten Umgebung realisierbar. Hierfür benutzt man Verfahren der dreidimensionalen Rekonstruktion. Planetare Erkundungsroboter könnten so umgehend ein Modell der erforschten Oberfläche fremder Planeten liefern und nicht nur Fotos oder Videoaufnahmen. Für die Marsroboter Spirit und Opportunity der NASA, die 2004 auf der Marsoberfläche landeten, war dreidimensionales Sehen mithilfe von Stereoanalyse der einzige 3-D-Wahrnehmungssensor. Er wurde zur Hinderniserkennung und für visuelle Odometrie verwendet (siehe Abb. 2 d)). Aufgrund des Erfolges des Systems wird es auch für künftige Erkundungsroboter ein fixer Bestandteil sein [33][34].

Weiters kann Stereoanalyse bei Robotern zur Objekterkennung herangezogen werden. Objekterkennung eröffnet Robotern und anderen autark agierenden Maschinen die Möglichkeit zur Interaktion mit Gegenständen. Ein wichtiger Einsatzbereich findet sich in der Industrie im Bereich der Produktionsautomatisierung. Hier werden Maschinen zur automatischen Inspektion oder bei der Herstellung von Artikeln eingesetzt. Aber auch humanoide Roboter benötigen eine Objekterkennung für ein besseres Zusammenspiel mit Menschen. Ein bekanntes Beispiel ist der Roboter ASIMO von Honda. Er kann unter anderem Gesten und Körperhaltung von Personen interpretieren, um ihnen die Hand zu schütteln oder ein Zuwinken ebenfalls mit einem Winken zu beantworten. Dies ist alles infolge der Tiefenwahrnehmung möglich [35][36].

Gesichtserkennungssysteme sind robuster, wenn sie Tiefeninformation miteinbeziehen. Das ist sowohl für humanoide Roboter als auch für biometrische Sicherheits- und Überwachungssysteme von Bedeutung. Die zusätzliche Information der dritten Dimension reduziert falsche Erkennungen [37].

### Tiefenbasierte Bildfreistellung

Ein Erfordernis in der Videoverarbeitung ist das fehlerfreie Zusammenfügen zweier Videosignale zu einem neuen, bei welchem sich die Objekte der beiden Eingangssignale korrekt überschneiden. Meistens ist eines der Videosignale synthetisch, wie etwa bei Wetterberichten im Fernsehen. Dafür ist es notwendig, Objekte oder Zonen in den Videobildern akkurat freistellen zu können. Als Standardverfahren für solch einen Zweck etablierte sich das Freistellen via

Farbmaske (engl. *chroma keying*). Objekte oder Personen befinden sich vor einer Leinwand mit konstanter Farbe, in der Regel entweder blau oder grün. Beim Mischen der Signale, welche sich aus einer Folge von Bildern zusammensetzen, substituiert man jedes Pixel der Hintergrundfarbe in den Bildern des ersten Signals durch den Farbwert der Bilder des zweiten Signals. Diese Technik hat allerdings ihre Schwächen, wie die Notwendigkeit eines speziellen Hintergrundes, die Schwierigkeit des homogenen Ausleuchtens der Fläche und der Hintergrund wird immer komplett ersetzt.

Im Zuge der Erschließung von Tiefeninformation aus Bildern, entwickelte man ein neues Verfahren für das Mischen zweier Videos oder im engeren Sinne zweier Bilder, die tiefenbasierte Bildfreistellung (engl. *z-keying* oder *depth keying*). Die Methode trennt Objekte eines Bildes in Vordergrund und Hintergrund anhand ihrer relativen Distanz von der Kamera und ist nicht auf Farbinformation angewiesen. Unterschiedliche Videosignale können auf diese Weise flexibler miteinander vermischt werden. Für die Erstellung einer Maske verwendet man die Tiefenwerte der einzelnen Bildpunkte. Für jeden Bildpunkt des neu erzeugten Bildes werden die Tiefenwerte der Bildpunkte der Eingabebilder verglichen und der Farbwert des Bildes, welches am nächsten ist, übernommen. Als Folge dessen können sich Objekte in den neu erzeugten Bildern gegenseitig korrekt verdecken, abhängig von ihrer geometrischen Beziehung. Das führt zu einer Aufhebung der starren Segmentierung in Vordergrund und Hintergrund der farbbasierten Freistellung. Ein spezieller Hintergrund ist nicht mehr erforderlich und das Verfahren kann bei Aufnahmen in natürlicher Umgebung angewandt werden. Es besteht die Möglichkeit, mehrere Ebenen einzuführen, indem man den Raum in verschiedene Abschnitte einteilt: als Vordergrund einen Moderator, als Zwischenebene eine eingeblendete halbtransparente Graphik und als Hintergrund den natürlichen Hintergrund, von wo aus die Szene aufgenommen wird. Dieses Szenario ist nur eines der möglichen Szenarien, welches sich mit der tiefenbasierten Bildfreistellung realisieren lässt [38][39].

#### Alternative Technologien

Die in den beschriebenen Anwendungen genutzte Tiefeninformation lässt sich nicht immer nur mit der Stereoanalyse aus Bildern gewinnen. Es gibt eine Reihe von Möglichkeiten, unter Zuhilfenahme verschiedener Technologien, dreidimensionale Daten der zu untersuchenden Umgebung zu gewinnen. Vor allem der Einsatz von aktiven Sensoren hat in den letzten Jahren stark zugenommen. Zu den aktiven Sensoren zählen unter anderem Laserabtastung (exempli gratia (e.g.) LIDAR), Mikrowellensensoren, Ultraschallsensoren und Röntgenstrahlgeräte. Gemein haben sie alle, dass sie selbst Energie aussenden und die zurückgeworfene Energie

analysieren. Bei der Laserabtastung zum Beispiel ermittelt man die Zeit, die es braucht, bis der emittierte Laserstrahl durch Reflexion wieder an der Quelle ankommt. Aus der Laufzeit des Strahles kann eine Tiefenmessung abgeleitet werden. Bei passiven Sensoren, wie bei Kameras für die Stereoanalyse, wird nur ausgestrahlte Energie der Sonne, von Lampen oder anderen externen Energiequellen aufgefangen. Sie sind auf das Vorhandensein einer solchen Energiequelle angewiesen, weil sie selbst keine Energie emittieren. Im Folgenden werden einige Schwächen und Stärken von aktiven und passiven Sensoren diskutiert.

Als eine der Stärken der Stereoanalyse gilt die dem Verfahren inhärente Abstimmung von Tiefeninformation mit visueller Information. Bei Laserscans erhält man nur ein Tiefenbild ohne eine dazu passende Farbaufnahme. Normale Kamerabilder sind für uns Menschen leicht interpretierbar und spielen nicht nur bei Missionen auf fremden Planeten eine große Rolle. Auch bei der Erstellung von realistischen 3-D-Rekonstruktionen benötigt man Bilder der Szene zur Texturierung der Modelle. Die einfache und günstige Anschaffung der Hardware spricht ebenfalls für die Stereoanalyse. Eine Kamera oder zwei reichen im Normalfall aus. Das spezielle Equipment für aktive Sensoren ist in der Anschaffung teurer. Aktuelle aktive Nahbereichssensoren haben auch meistens eine geringere Reichweite als die bildbasierte Tiefengewinnung, bei welcher der Tiefenbereich nur von der Position der Kameras und der damit verbundenen Basislinie beschränkt ist. Aktive Sensoren besitzen generell eine geringe Auflösung. Die Messungen sind dafür präziser als bei der passiven Technik. Für eine Messung müssen aktive Sensoren Energiestrahlen aussenden, die für den Menschen potentiell gefährlich sind. Bei der Aufnahme via Kameras werden keinerlei Strahlen ausgesendet. Daher ist der Einsatz von Kameras prädestiniert für eine Anwendung bei humanoiden Robotern. Nachteilig wirkt sich jedoch aus, dass Kameras sehr stark vom Umgebungslicht abhängig sind und es zu starken Intensitätsunterschieden zwischen linkem und rechtem Stereobild kommen kann. Das erschwert das Finden von Korrespondenzen. Auch Bildrauschen sowie fehlende Textur kann zu falschen Korrespondenzen führen. Fehlende Textur stellt für aktive Sensoren wiederum keine Schwierigkeit dar, aber bei spiegelnden Oberflächen und Oberflächen mit geringem Reflexionsgrad, wie klarem Wasser, kommt kein Energiestrahl zum Empfänger zurück. An diesen Stellen liefern die Sensoren daher keine Entfernungsangaben [10][29][34].

Betrachtet man die unterschiedlichen Vor- und Nachteile der beiden Sensortechniken, so ist der Trend hin zu Systemen, welche Daten aus verschiedenen Sensoren beziehen, verständlich. Man erhöht damit die Robustheit des Systems und verbessert die Resultate, weil die Kombination diverser sensorieller Datenquellen die Mängel einzelner Sensorentypen relativieren.

Mit der zunehmenden Datenintegration lassen sich auch a priori Informationen, beispielsweise von GIS-Systemen, einbeziehen. Damit einher geht auch die Verbindung von Fachgebieten wie Maschinelles Sehen, Robotik, Computergraphik, Photogrammetrie und Fernerkundung. Die zunehmende Verlässlichkeit und Vielfalt der Multisenor-Daten hilft außerdem, den Automatisierungsgrad für das Ableiten von Metainformation wie Objekterkennung zu erhöhen [40].

# 2 Grundlagen der Stereoanalyse

Das Ziel der Stereoanalyse ist die Bestimmung von Tiefeninformation anhand der im Vergleich von zwei Bildern vorhandenen binokularen Disparität. Grundsätzlich besteht ein Stereosystem aus den folgenden Phasen [2][41]:

- **Bilderfassung**: Hier sind die Auflösung der Kamera sowie das Umgebungslicht zu berücksichtigen.
- **Kalibrierung der Kamera:** Für eine Tiefenbestimmung bedarf es der Kenntnis der inneren und externen Geometrie der Kamera (oder Kameras, wenn mehr als eine Kamera verwendet wird). Abhängig vom eingesetzten Kamerasystem kann die Kalibrierung a priori stattfinden oder wenn dies nicht möglich ist, erst nach der Bildaufnahme.
- **Merkmalsextraktion**: Je nachdem welcher Stereoalgorithmus zum Einsatz kommt, werden entsprechende Merkmale für die Korrespondenzanalyse in den Bildern ausfindig gemacht.
- **Korrespondenzanalyse**: Sie stellt den schwierigsten und wichtigsten Teil der Stereoanalyse dar. Der Begriff bezeichnet das automatische Auffinden von korrespondierenden Merkmalen in beiden Bildern zur Disparitätsbestimmung.
- **Verfeinerung der Disparitäten:** Je nach Notwendigkeit der Anwendung kann die resultierende Disparitätskarte mit Nachbearbeitung etwas verbessert werden.
- **Tiefenrekonstruktion via Triangulation:** Kennt man die Disparitäten, lässt sich damit die Tiefe von Szenepunkten mit Triangulation bestimmen.

Kapitel 2.5 behandelt die Punkte Merkmalsextraktion, Korrespondenzanalyse und Verfeinerung der Disparitäten näher. Auf die Punkte Bilderfassung und Kalibrierung der Kamera wird nicht genauer eingegangen. Kapitel 2.2 erläutert die Tiefenrekonstruktion via Triangulation. Die hier beschriebenen Algorithmen benötigen als Eingabe zwei Stereobilder und liefern als Ausgabe die Disparitäten für die Bilder. Das Erfassen der Bilder und die Kalibrierung der Kamera sind hierfür bereits beendet. Weiters müssen die Stereobilder epipolar begradigt vorliegen, damit die Epipolarlinien mit den Bildzeilen übereinstimmen. Die dazugehörige Geometrie bezeichnet man als Epipolargeometrie (auch Kernstrahlgeometrie).

## 2.1 Epipolargeometrie

Stereoaufnahmen können auf unterschiedliche Weise generiert werden. Verwendet man nur eine Kamera, werden aufeinanderfolgende Bilder einer Sequenz als Stereobilder herangezogen, gewissermaßen ein virtuelles Stereokamerasystem. Das bringt den Vorteil, dass die Basislänge – das ist der Abstand zwischen den Projektionszentren der Aufnahmen – je nach Genauigkeitsanforderung angepasst werden kann [29]. Bei Festbasisstereokameras mit zwei Kameras ist eine Veränderung der Basislänge nicht möglich. Sie verfügen indes über den positiven Aspekt, Bilder in achsenparalleler Anordnung zu produzieren. Dabei stehen die Aufnahmerichtungen normal zur Basislinie und sind zueinander parallel. In diesem Fall gestaltet sich die Stereoanalyse einfacher. Es ist die Zielsetzung von Stereokamerasystemen, virtuell oder echt, Bilder in dieser achsenparallelen Anordnung zu liefern. Eine Menge von Stereoalgorithmen setzen diese Geometrie voraus. Leider lässt sich dieser Fall insbesondere bei Luftaufnahmen schwer einhalten und die Kameras besitzen eine konvergente Anordnung. Diese Konvergenz kann allerdings in einem Nachbearbeitungsschritt durch eine virtuelle Drehung der Bilder ausgeglichen werden. Diesen Prozess bezeichnet man als (epipolare) Begradigung oder Rektifikation.

Der Grund für die Begradigung und die sich daraus ergebende Vereinfachung wird deutlich, betrachtet man die Geometrie von Stereokamerasystemen näher. Abbildung 3 zeigt schematisch ein konvergentes Stereosystem. Die drei Punkte $O_1$, $O_2$ und $P$ spannen die sogenannte Epipolarebene auf. Beide Kameras besitzen jeweils eine Bildfläche. Sie definiert die Bildebene einer Kamera, indem man die Fläche in alle Richtungen hin verlängert. Aus den Schnittgeraden zwischen der Epipolarebene und den Bildebenen der Kameras resultieren die Epipolarlinien. Ändert sich der Szenepunkt *P*, entsteht eine andere Epipolarebene und damit ebenfalls andere Epipolarlinien. Zu jeder Epipolarebene gibt es zwei Epipolarlinien, eine in der Bildebene der linken Kamera und eine in der Bildebene der rechten Kamera. Alle Epipolarlinien für eine Bildebene schneiden sich im sogenannten Epipol. Das ist jener Punkt, an dem die Gerade, welche aus der Verbindung der beiden Projektionszentren entsteht, auf die Bildebene trifft. Der Epipol ist das Abbild des anderen Projektionszentrums oder anders gesagt, der Ort, an dem die eine Kamera von der anderen gesehen wird [42].

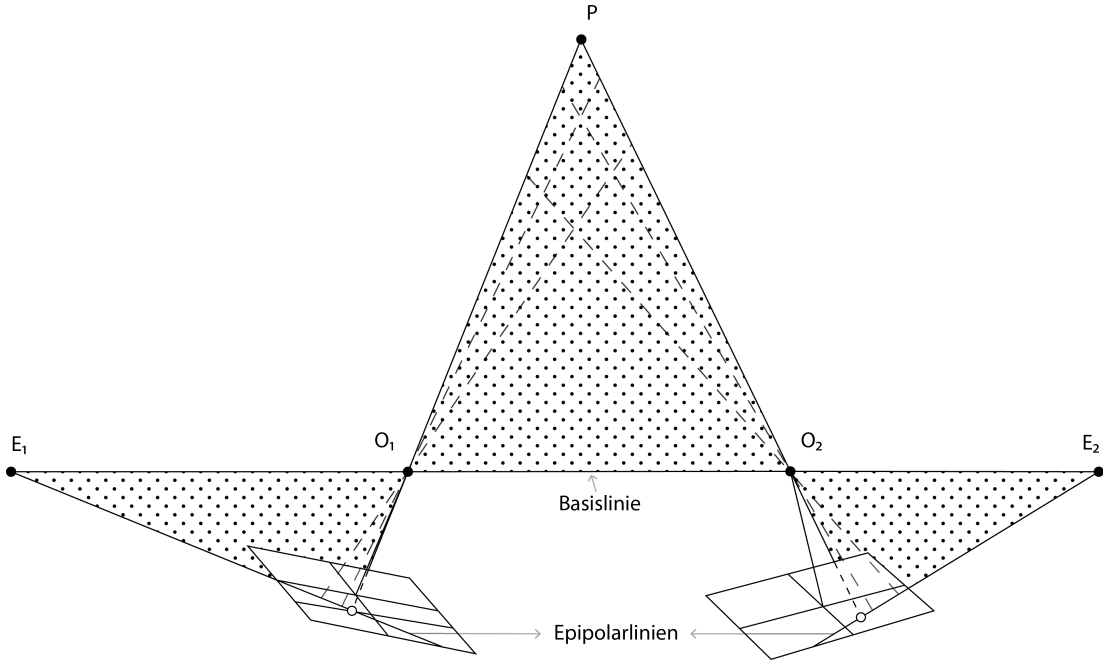



Abbildung 3: Bei der konvergenten Kameraanordnung verlaufen die Epipolarlinien quer durch die Bildfläche der Kamera. Die Epipolarlinien ergeben sich aus der Schnittfläche der Epipolarebene mit den Bildebenen der Kameras. Die Epipolarebene wird durch die zwei Projektionszentren der Kameras $O_1$ und $O_2$, sowie einem Szenepunkt $P$ definiert. Alle Epipolarlinien einer Bildebene schneiden sich im Epipol. $E_1$ und $E_2$ beschreiben je den Epipol für die Bildebene der linken und rechten Kamera. Der korrespondierende Bildpunkt eines Szenepunktes $P$ liegt immer auf der dazugehörigen Epipolarlinie im anderen Bild [42]. Quelle: Vorlage aus [29].

Der springende Punkt ist nun, dass mithilfe der Epipolarlinien die Suche nach korrespondierenden Merkmalen von einem zweidimensionalen Problem auf ein eindimensionales reduziert werden kann. Zusammengehörige Merkmale finden sich entlang übereinstimmender Epipolarlinien. Somit muss nicht mehr das ganze Bild abgesucht werden, sondern nur mehr die entsprechende Epipolarlinie im anderen Bild. Das bringt einen Gewinn an Rechenzeit, reduziert die Komplexität der Suche und erhöht die Zuverlässigkeit der Korrelationen. Weil das Bestimmen der Epipolarlinien allerdings Zeit kostet, begradigt man die Epipolarlinien derart, dass sie mit den horizontalen Zeilen der Bilder übereinstimmen. Bei der Begradigung wird die konvergente Kameraanordnung zu einem achsenparallelen Stereosystem umgewandelt, wie in Abbildung 4 zu sehen. Die Korrespondenzsuche reduziert sich auf die Suche zwischen einer Zeile im linken Bild und der entsprechenden Zeile im rechten Bild [23][29].

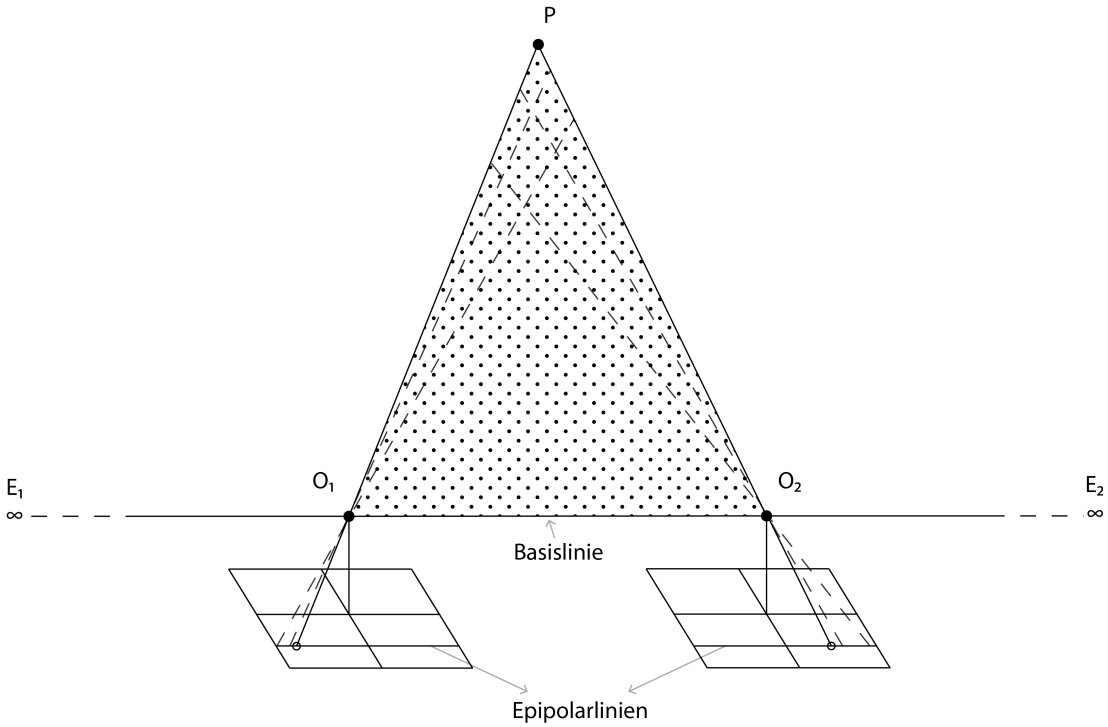



Abbildung 4: Bei der achsenparallelen Kameraanordnung decken sich die Epipolarlinien mit den Zeilen der Bilder. Daher entfällt eine Bestimmung der Epipolarlinien. Der korrespondierende Bildpunkt eines Szenepunktes $P$ liegt nun immer auf der korrespondierenden Zeile im anderen Bild. Durch die Begradigung wandern die Epipole $E_1$ und $E_2$ ins Unendliche, weil sich die Epipolarlinien nicht mehr schneiden. $O_1$ und $O_2$ sind die Projektionszentren der Kameras. Quelle: Vorlage aus [29].

Angenommen die Bilder wurden begradigt und eine Disparitätskarte mithilfe eines Stereoalgorithmus wurde erstellt, dann kann damit relativ leicht die Tiefe von Szenepunkten eruiert werden.

## 2.2 Tiefenbestimmung

Ein Stereoverfahren generiert eine Disparitätskarte, in der für jedes Pixel die Disparität verzeichnet ist. Diese bezeichnet den Lageunterschied korrespondierender Elemente in Pixelanzahl. Unter Kenntnis der inneren und äußeren Parameter der Kameras und durch Anwendung der Strahlensätze kann der Abstand vom Aufnahmeort bestimmt werden [23][29].

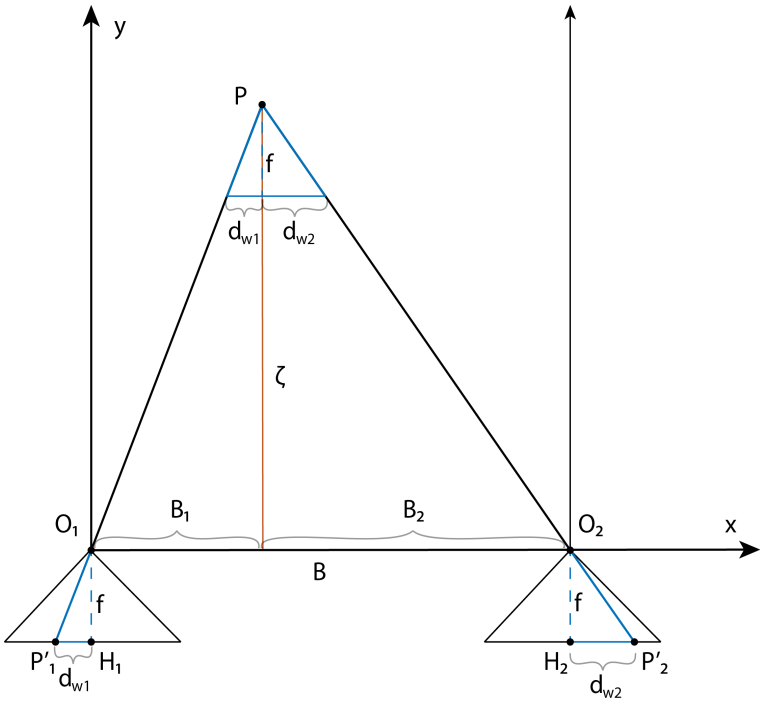



Abbildung 5: Prinzip der Tiefenbestimmung mithilfe von Strahlensätzen für den horizontalen Normalfall. Es zeigt sich, dass die Disparität umgekehrt proportional zur Raumtiefe eines Punktes ist. Je kleiner die Disparität ist, desto größer ist die Entfernung von den Projektionszentren der Kameras. Quelle: [29].

Die Einheit der Disparität $d$ ist Pixel, weil sie in Bildkoordinaten berechnet wird:

$$d = d_1 - d_2 \tag{2.1}$$

$d_1$ und $d_2$ beschreiben die Position des Pixels von Interesse im linken und rechten Bild.
Für die Ermittlung der Tiefe ist eine Umrechnung der Disparität in eine Einheit der Weltkoordinaten notwendig, e.g. Millimeter. Dazu multipliziert man die Einheit mit dem Skalierungsfaktor $b_{CCD}$, welcher die Breite eines CCD-Elements der eingesetzten Kameras angibt:

$$d_w = d_{w1} - d_{w2} = (d_1 - d_2)\, b_{CCD} \tag{2.2}$$

Abbildung 5 zeigt die relevanten geometrischen Verhältnisse zur Bestimmung der Tiefe. $d_{w1}$ und $d_{w2}$ werden aus Gründen der Übersicht von der Bildhälfte weg definiert. $d_{w2}$ ist in diesem Fall negativ. $O_1$ und $O_2$ sind die Projektionszentren der Kameras, $f$ die Brennweite und $B$ die Basislänge.

Der dritte Strahlensatz führt zu folgenden Beziehungen:

$$\frac{d_{w1}}{d_w} = \frac{B_1}{B} \quad und \quad \frac{d_{w2}}{d_w} = \frac{B_2}{B} \tag{2.3}$$

Damit können die Unbekannten $B_1$ und $B_2$ durch bekannte Größen ausgedrückt werden.

Gemäß dem zweiten Strahlensatz ergeben sich die Relationen:

$$\frac{f}{\zeta} = \frac{d_{w1}}{B_1} \quad und \quad \frac{f}{\zeta} = \frac{d_{w2}}{B_2} \tag{2.4}$$

Setzt man für $B_1$ und $B_2$ die aus Formel (2.3) erhaltenen Termini in (2.4) ein, resultiert daraus die Gleichung zur Tiefenbestimmung:

$$\zeta = \frac{f \cdot B}{d_W} \tag{2.5}$$

Die Disparität $d_w$ bzw. $d$ (in Einheit Pixel) ist somit umgekehrt proportional zur Raumtiefe $\zeta$ eines Punktes. Der Wert der Brennweite $f$ der beiden Kameras und die Basislänge $B$ sind konstant. Mit größer werdender Entfernung von der Kamera konvergiert die Disparität somit gegen null.

## 2.3 Bedingungen und Annahmen

Schwierigkeiten in der Stereoanalyse bereitet insbesondere die Ambiguität der Bildaufnahme. Oberflächen unterschiedlicher Materialien und Eigenschaften können zu denselben Abbildungen in den Aufnahmen führen. D.h. der Abbildungsprozess während der Aufnahme repräsentiert eine *n-zu-1* Beziehung, bei dem *n* Gegebenheiten auf *1* reduziert werden.

Für die umgekehrte Aufgabe, aus Bilddaten wieder auf Oberflächen, deren Materialien und geometrischen Merkmale zu schließen, gibt es daher keine eindeutige Lösung. Man kennt zwei mögliche Auswege aus dieser Situation [43]. Zum einen können zusätzliche Daten respektive Bilder, gesammelt werden, zum anderen hilft es, Annahmen über die Welt und ihre Beschaffenheit zu machen. Bei der Stereoanalyse trifft de facto beides zu. Zur Schätzung der Disparität benötigt man mindestens zwei Bilder. Manche Algorithmen nutzen sogar mehr als zwei Bilder. Des Weiteren müssen für das Auffinden korrespondierender Merkmale bestimmte Annahmen gemacht oder Bedingungen eingehalten werden, um zu einer realisierbaren Lö-

sung zu gelangen. Annahmen oder Bedingungen reduzieren die Ambiguität bei der Korrespondenzfindung und erleichtern damit die Analyse. Die Bedingungen sind einerseits von geometrischen Eigenschaften bei der Bildentstehung ableitbar andererseits von Eigenschaften der Objekte in der natürlichen Welt. Bedingungen müssen aus verschiedenen Gründen nicht immer zutreffen. In diesem Fall werden bei Forcierung der Bedingungen als Konsequenz inkorrekte Zuordnungen in Kauf genommen [23][44]. Unterschiedliche Stereoalgorithmen bedienen sich unterschiedlicher Annahmen, je nach Art und Weise des Algorithmus. Nachfolgend werden einige häufig gebrauchte Bedingungen und Annahmen vorgestellt [2][45][46].

#### Epipolarbedingung

Das zu einem Pixel im linken Bild korrespondierende Pixel im rechten Bild kann nur auf der entsprechenden Epipolarlinie des rechten Bildes liegen und umgekehrt. Damit reduziert sich die Suche nach Korrespondenzen von einem zweidimensionalen Problem auf ein eindimensionales. Ein Sonderfall ergibt sich nach der epipolaren Begradigung der Bilder. Hier stimmen die Epipolarlinien mit den horizontalen Pixelreihen der Bilder überein. Es entfällt die Bestimmung der Epipolarlinien.

#### Eindeutigkeitsbedingung

Die Eindeutigkeitsbedingung besagt, dass jedes Pixel eines Bildes nur mit genau einem Pixel des anderen Bildes korrespondieren kann. Das bedeutet, die Projektion eines Punktes der realen Welt führt nur jeweils zu einer Abbildung im linken und rechten Bild. Die Annahme trifft nicht zu, wenn zwei oder mehrere Szenepunkte auf dem Sichtstrahl einer Kamera liegen (siehe Abb. 6 a)). In diesem Fall kommt es bei dieser Kamera nur zu einer Abbildung, aber zu verschiedenen in der anderen Kamera. Dies tritt bei der Rekonstruktion geneigter Oberflächen auf. Auch bei Verdeckungen kommt es zu einer Verletzung der Bedingung. In diesem Fall existiert kein übereinstimmendes Element im anderen Bild. Bestimmte Pixel tauchen lediglich in einem Bild auf, weil sie im anderen Bild durch ein Objekt geringerer Tiefe verdeckt werden.

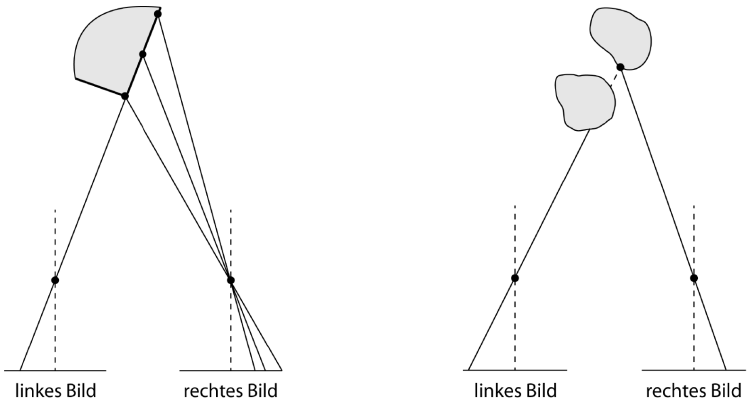



Abbildung 6: a) Wenn mehrere Szenepunkte auf dem Sichtstrahl einer Kamera liegen, wird die Eindeutigkeitsbedingung verletzt. b) Bei Verdeckungen kommt es ebenfalls zu einer Verletzung der Bedingung, da manche Pixel nur in einem Bild auftauchen. Quelle: [45].

#### Symmetriebedingung

Unabhängig von der Suchrichtung sollen die gleichen Korrespondenzen gefunden werden. Würde man also linkes und rechtes Bild vertauschen, müssten die gleichen Bildpunkte korrespondieren. Diese Bedingung gebraucht man häufig zum Auffinden fehlerhafter Zuordnungen.

#### Fotokonsistenzannahme

Die Bedingung setzt voraus, dass sich die Intensitätswerte korrespondierender Bildpunkte nur leicht unterscheiden. Seien $(x_l, y_l)$ und $(x_r, y_r)$ übereinstimmende Pixel im linken und rechten Bild, so müsste demnach die absolute Differenz der Intensitäten $|I_l(x_l, y_l) - I_r(x_r, y_r)|$ klein sein. Eine exakte Übereinstimmung ist auf Grund der unterschiedlichen Perspektiven der Kameras unwahrscheinlich, weil sich die Lichtbedingungen und die Reflexionen der Oberflächen ändern. Die Annahme ist auf Farbwerte erweiterbar, wenn sie anstatt der Intensitätswerte genutzt werden.

#### Geometrische Ähnlichkeitsbedingung

Geometrische Formen wie Liniensegmente korrespondieren nur dann mit Formen im anderen Bild, wenn sie eine geometrische Ähnlichkeit aufweisen. Bei Liniensegmenten entspricht dies

einer ähnlichen Orientierung oder Länge der Segmente in beiden Bildern. Die Bedingung basiert auf der Beobachtung, dass sich die geometrischen Eigenschaften von Objekten im linken und rechten Bild zueinander nicht stark ändern.

#### Kontinuitätsannahme für Disparitäten

Bei der Kontinuitätsbedingung nimmt man an, dass die Disparitäten fast überall im Bild nur stetig variieren. Das setzt eine kontinuierliche Veränderung der Tiefe von 3-D-Punkten voraus. Entsprechend darf sich auch die Disparität zwischen benachbarten Pixeln in den Bildern nur innerhalb einer bestimmten Toleranz ändern. Seien $(x_l, y_l)$ und $(x_r, y_r)$ korrespondierende Pixel im linken und rechten Bild, so kann ein zu $(x_l, y_l)$ benachbarter Pixel $(x_{l2}, y_{l2})$ nur mit einem Pixel $(x_{r2}, y_{r2})$ im rechten Bild korrespondieren, wenn die absolute Differenz der Disparitäten gering ist:

$$|\sqrt{(x_l - x_r)^2 - (y_l - y_r)^2} - \sqrt{(x_{l2} - x_{r2})^2 - (y_{l2} - y_{r2})^2}| \tag{2.6}$$

An den Objekträndern ist diese Bedingung üblicherweise nicht erfüllt. Hier kommt es zu Tiefensprüngen, welche wiederum Diskontinuitäten in der Disparität hervorrufen. Die Kontinuitätsbedingung wird auch als Glattheitsbedingung bezeichnet.

#### Disparitätslimit

Unter Kenntnis des Mindestabstandes der Objekte von den Kameras ist es möglich, bei einem Stereosystem ein Disparitätslimit $d_{max}$ anzugeben. Dadurch reduziert sich der Suchaufwand für mögliche Kandidaten bei der Korrespondenzanalyse. Auch beim Menschen gibt es eine Form des Disparitätslimits. Wenn zwei Übereinstimmungen im linken und rechten Auge eine zu hohe Disparität aufweisen, werden sie nicht zu einem Bild fusioniert, sondern doppelt wahrgenommen [45].

#### Reihenfolgebedingung

Diese Annahme drückt aus, dass die Reihenfolge der auf einer Epipolarlinie liegenden Bildpunkte in einem Bild entlang der entsprechenden Epipolarlinie im zweiten Bild erhalten bleibt. Bei Szenen mit kleinen Objekten im Vordergrund oder bei Szenen mit transparenten Oberflächen trifft diese Bedingung nicht zu. Abbildung 7 illustriert einen solchen Ausnahmefall. Die Reihenfolgebedingung kann ebenfalls als eine Form der Glattheitsbedingung angese-

hen werden. Sie setzt voraus, dass sich alle Objekte annähernd im selben Abstand zur Kamera befinden, sodass keine Invertierung der Reihenfolge auftritt. Die Bedingung wird gerne in Verbindung mit dynamischer Programmierung eingesetzt, um die Komplexität des Algorithmus zu reduzieren.

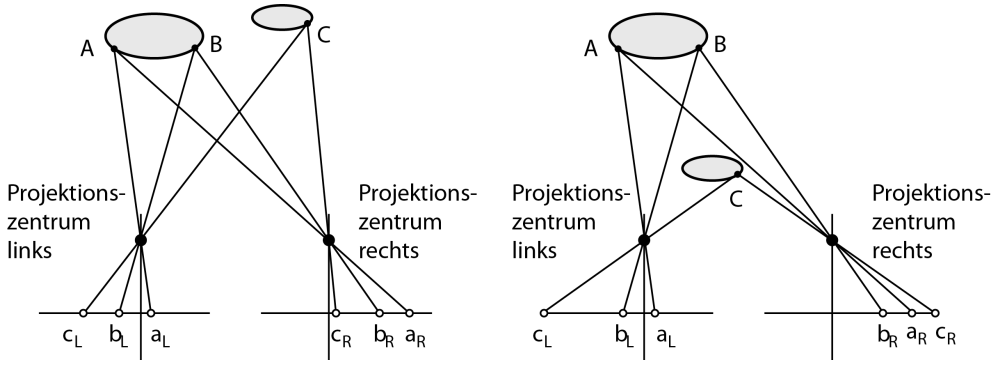



Abbildung 7: Kleine Objekte im Vordergrund bewirken eine Verletzung der Reihenfolgebedingung. Links besitzen die Objekte annähernd denselben Abstand von den Bildebenen. Die Projektionen der Punkte treffen in der Reihenfolge c, b, a auf die Bildebenen. Rechts befinden sich die Objekte unterschiedlich weit von den Ebenen entfernt. Auf die rechte Bildebene treffen die Projektionen der Punkte nun in der veränderten Reihenfolge $b_R$, $a_R$, $c_R$ auf. Quelle: [2][46].

## 2.4 Herausforderungen der Stereoanalyse

Wenngleich auf dem Gebiet der Stereoanalyse schon seit Jahrzehnten intensiv geforscht wird, kann das Korrespondenzproblem, ein wesentlicher Bestandteil eines Stereoverfahrens, noch nicht als gelöst betrachtet werden. Die Gründe dafür sind mannigfaltig:

- **Ambiguität von Merkmalen**: Oberflächen verschiedener Materialien und Eigenschaften können zu denselben Intensitäts- oder Farbwerten führen. Dies führt wiederum bei Merkmalen, welche auf diesen Werten basieren, zu Ambiguitäten in der Korrespondenzsuche.
- **Homogene Regionen:** Viele Bilder beinhalten homogene Regionen, in denen Intensitäts- oder Farbwerte nur leicht schwanken. Eine Zuordnung auf Pixelebene ist in diesen Regionen aussichtslos, weil sich für verschiedene Disparitäten fast identische Zuordnungskosten ergeben.

- **Wiederkehrende Textur:** Als ähnlich problematisch gelten Regionen mit wiederkehrender Textur. Wiederholt sich die Textur, wiederholen sich auch die Merkmalsausprägungen.

- **Heterogene Merkmalsausprägungen**: Bei Ambiguität von Merkmalen stimmen verschiedene Bereiche überein, obwohl sie das nicht sollten. Bei heterogener Merkmalsausprägung hingegen stimmen Bereiche nicht überein, obwohl sie es sollten. Der Grund dafür ist, dass die Abbildung eines Punktes der realen Welt zu unterschiedlichen Intensitäts- oder Farbwerten in den beiden Kameras eines Stereosystems führen kann. Mehrere Faktoren spielen dabei eine Rolle [47]. So sind zwei Kameras nie völlig identisch in ihren Eigenschaften und liefern unterschiedliche Werte. Die abgelichteten Oberflächen der Szene sind in der Regel keine Lambert-Flächen und reflektieren je nach Betrachtungswinkel stärker oder schwächer. Des Weiteren verursachen Kameratechnik und Quantisierungsfehler Bildrauschen und durch die versetzte Kameraposition entstehen perspektivische Verzerrungen zwischen den Bildern. Ein kaum beachtetes Problem sind darüber hinaus transparente Oberflächen. Weil die Intensitätswerte bei transparenten Oberflächen von mehreren Objekten unterschiedlicher Tiefe stammen, bewirkt eine Versetzung der Kamera abweichende Werte. Kaum ein Stereosystem berücksichtigt bisher transparente Oberflächen.

- **Verdeckungen:** Von der Kameraposition unterschiedlich weit entfernte Objekte erzeugen Verdeckungen, welche in den Bildern halb verdeckte Pixel erzeugen. Das sind jene Pixel, welche nur in einem Bild vorkommen, weil sie im anderen Bild aufgrund der verschobenen Perspektive hinter einem Objekt verborgen sind. Solche Halbverdeckungen treten an Disparitätssprüngen auf. Für derartige Pixel existieren keine Korrespondenzen im anderen Bild. Eine korrekte Disparitätszuweisung gestaltet sich darum besonders schwierig. Eine verbreitete Technik besteht darin, die halb verdeckten Pixel zu identifizieren (e.g. mithilfe eines Konsistenztests) und sie anschließend mit Disparitäten verlässlicher Schätzung der Nachbarschaft aufzufüllen.

- **Bewahrung hervorstechender Anhaltspunkte:** Viele Szenen weisen für uns Menschen hervorstechende Anhaltspunkte in ihrer Struktur auf. Dazu zählen Tiefendiskontinuitäten an Objektgrenzen, steil abfallende Oberflächen oder schmale Objekte im Vordergrund [7]. Aufgrund eingesetzter Bedingungen (e.g. Kontinuitätsbedingung) oder lokaler Aggregation besteht die Gefahr, über diese Punkte hinweg zu glätten. Für eine akkurate Rekonstruktion sollten diese Merkmale jedoch erhalten bleiben.

## 2.5 Lösungsverfahren für das Korrespondenzproblem

Mittlerweile existiert im Bereich des maschinellen Stereosehens ein weites Spektrum an unterschiedlichen Verfahren. Alle zu behandeln, würde den Rahmen dieser Arbeit sprengen. Manche Artikel beschäftigen sich allerdings damit, die wichtigsten Strömungen und Methodiken zu identifizieren und zusammenzufassen. Dazu gehören die Artikel von [48][49][46][41] [50]. Besondere Erwähnung gebührt dem Artikel [41]. Es gelingt Scharstein und Szeliski eine Taxonomie für bestehende Stereoverfahren zu etablieren, welche es ermöglicht, verschiedene Komponenten der Verfahren individuell zu vergleichen. Ferner begannen sie, eine Testumgebung für Stereoalgorithmen aufzubauen, die eine quantitative Evaluierung ermöglicht und so die Gelegenheit bietet, Ergebnisse von Stereoalgorithmen gegenüberzustellen. Ihre Testumgebung hat sich mittlerweile als Standardtestumgebung im Bereich der Stereoanalyse [51] profiliert. Scharstein und Szeliski beschränken sich auf Algorithmen, die mit begradigten Stereobildpaaren arbeiten und eine dichte Disparitätskarte[4] liefern. Zusätzlich zu diesen Einschränkungen, wird hier in dieser Arbeit vor allem auf Verfahren eingegangen, die ein Echtzeitpotential besitzen.

Nach der Taxonomie von [41] führen Stereoalgorithmen generell vier Schritte aus:

(1) Berechnung der Zuordnungskosten

(2) Kostenaggregation

(3) Bestimmung und Optimierung der Disparitäten

(4) Verfeinerung der Disparitäten

Ad (1): Für die Berechnung der Zuordnungskosten bestehen verschiedene Ähnlichkeitsmetriken. Die Kosten speichert man für gewöhnlich in ein sogenanntes Disparitätsraumvolumen (DRV, engl. *disparity space volume*). Das DRV ist eine dreidimensionale Struktur, welche jedes Pixel im Referenzbild mit einer Reihe möglicher Korrespondenzkandidaten im anderen Bild in Verbindung bringt. Die Dimensionen des DRVs ergeben sich aus der Bildbreite $m$, der Bildhöhe $n$ und dem Suchbereich für die Disparitäten $d$. Die Zuordnungskosten für ein

4 Dichte Disparitätskarten bezeichnen Disparitätskarten, welche für jedes Pixel eine Disparitätsschätzung besitzen. Lokale Methoden und pixelbasierte globale Verfahren produzieren für gewöhnlich dichte Disparitätskarten. Merkmalsbasierte Ansätze hingegen bringen spärlich besetzte Disparitätskarten hervor, weil sie nur für die zugeordneten Merkmale über Disparitätswerte verfügen. Die restlichen Disparitäten müssen in einem Nachbearbeitungsschritt interpoliert werden.

Pixel $(x, y)$ im Referenzbild zu dem Pixel $(x-d, y)$ im Zielbild stehen im DRV an der Stelle $(x, y, d)$. Für eine bestimmte Bildzeile $y_i$ reduzieren sich die Dimensionen des DRVs auf zwei. Solche zweidimensionalen Kostenmatrizen mit den Dimensionen *x* und *d* deklariert man als Disparitätsraumbilder (DRB, engl. *disparity space image*). Legt man die DRBs jeder Zeile schichtweise übereinander, erhält man wieder das DRV.

Ad (2): Über die Aggregation von Kosten innerhalb bestimmter Bereiche berichtet Kapitel 3. Die Aggregation definiert den Grundgedanken lokaler Methoden und die unterschiedlichen Formen der Aggregation differenzieren die Verfahren. Lokale Ansätze bestimmen die Disparitäten für jeden Bildpunkt nach ausgeführter Kostenaggregation normalerweise mit einer simplen Gewinner-nimmt-alles-Strategie. Dabei wird die Disparität mit den geringsten Zuordnungskosten ausgewählt.

Ad (3): Die Optimierung der Disparitäten ist der Schlüssel zum Erfolg bei globalen Methoden. Sie werden in Kapitel 4 erörtert. Globale Ansätze berücksichtigen bei der Tiefenbestimmung für ein Pixel im Idealfall die Kosten für alle anderen, indem sie explizite Glattheitsbedingungen involvieren und die Minimierung einer globalen Energiefunktion anstreben.

Ad (4): Diverse Algorithmen führen Nachbearbeitungsschritte aus, um die Schätzung der Disparitäten noch etwas zu verbessern. Dazu zählen Subpixelverfeinerung, Verdeckungsbehandlung oder der Gebrauch bestimmter Filter. Eine simple Art der Subpixelverfeinerung ist das Einpassen einer Kurve in die Umgebung der diskreten Zuordnungskosten. Damit lässt sich die Auflösung relativ mühelos erhöhen. Eine beliebte Technik zur Verdeckungserkennung stellt der Konsistenztest dar. Hierbei berechnet man Disparitätskarten für linkes und rechtes Stereobild, vergleicht die Resultate und eliminiert inkorrekte Zuordnungen unter dem Licht bestimmter Bedingungen. Bei der starken Konsistenzbedingung beispielsweise müssen die Disparitäten $d_l$ und $d_r$ der Stellen $(x_l, y_l)$ im linken Bild und $(x_r-d_l, y_r)$ im rechten Bild dieselben Werte aufweisen. Bei der schwachen Konsistenzbedingung sollte $d_r$ gleich oder größer als $d_l$ sein, andernfalls wird das Pixel als verdeckt markiert [52]. Verdeckte Pixel können anschließend mit Disparitäten aus der Umgebung aufgefüllt werden.

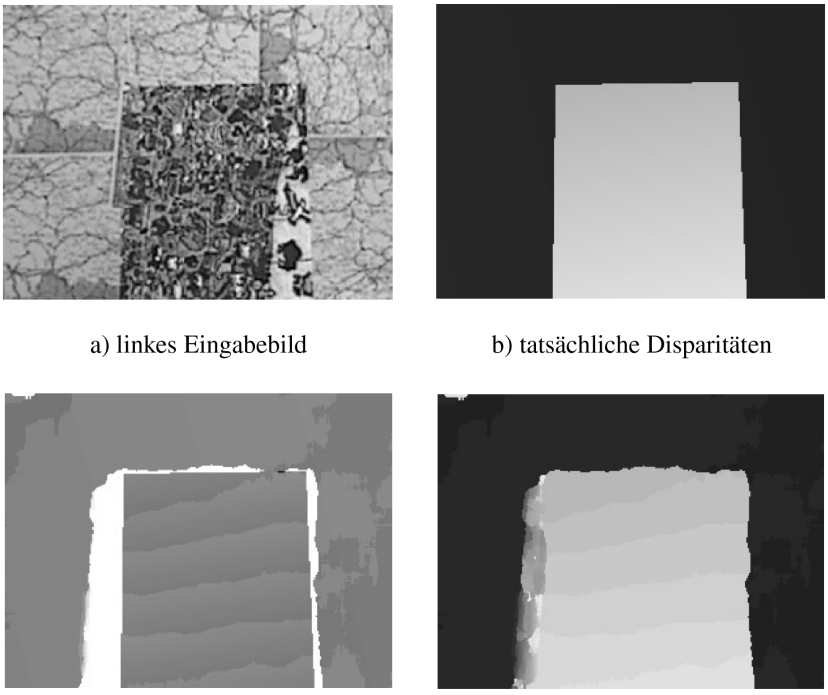
# 3 Lokale Echtzeitverfahren

Bei Stereoalgorithmen, welche eine dichte Disparitätskarte erzeugen, muss im Wesentlichen für jedes Pixel in einem Bild das korrespondierende Pixel im anderen Stereobild gefunden werden. Weil aber einzelne Pixel für eine Zuordnung nicht diskriminierend genug sind, konstruiert man bei lokalen Methoden Bereiche um die Pixel von Interesse, meistens in Form eines Fensters. Das Fenster wird mit einem zweiten Fenster im anderen Bild anhand einer Ähnlichkeitsmetrik verglichen. Je nach Disparität wird das zweite Fenster leicht verschoben. Wie weit das Fenster verschoben wird, kann je nach Szene variieren, weil die Anzahl der möglichen Disparitäten für jede Szene differiert. Die Position mit der größten Übereinstimmung determiniert schließlich die Schätzung für die Disparität (Gewinner-nimmt-alles-Strategie). Größe und Form des Fensters, sowie die eingesetzte Ähnlichkeitsmetrik sind dabei die bestimmenden Faktoren für die Leistungsfähigkeit lokaler Methoden.

## 3.1 Probleme lokaler Methoden

Der Einsatz von Fenstern wird durch die Kontinuitätsannahme gerechtfertigt. Sie setzt voraus, dass benachbarte Bildpunkte fast überall im Bild zu einer kontinuierlich verlaufenden Oberfläche im Raum gehören und daher eine ähnliche Disparität aufweisen. An Objektgrenzen, wo es zu Diskontinuitäten in der Tiefe kommt, wird diese Annahme verletzt. In diesem Fall beinhalten die zu vergleichenden Fenster Bildpunkte von zwei oder mehreren unterschiedlichen Oberflächen mit verschiedenen Disparitäten. Während in texturlosen Bereichen ein möglichst großes Fenster wünschenswert ist, um inkorrekte Zuordnungen zu vermeiden, möchte man in der Nähe von Objektgrenzen ein Fenster, welches nur Pixel jenes Objektes enthält, auf dem sich das momentan zu korrespondierende Pixel befindet. Welche Disparitätspopulation sich bei einem Fenster, das mehrere Objekte unterschiedlicher Tiefe einschließt, letztlich durchsetzt, hängt von der Texturierung der Objekte ab. Als Beispiel zeigt Abbildung 8 ein interessantes Resultat für das Stereobildpaar Map, welches als Erstes in [53] vorgestellt wird und mittlerweile Teil des Middlebury-Datenbestandes ist. In der Abbildung zu sehen sind das linke Bild, die tatsächlichen Disparitätswerte und die mittels eines 21-mal-21 Pixel großen Fensters rekonstruierten Disparitätswerte. Als Metrik kam die Summe der absoluten Differenzen (SAD) zum Einsatz. Ein Fehlerbild, welches inkorrekte Zuordnungen als weiße Bildpunkte repräsentiert, verdeutlicht, dass falsche Zuordnungen vermehrt nahe Diskontinuitäten auftreten. Betrachtet man die obere und rechte Kante des Vordergrundobjektes im Bild der berech-

neten Disparitäten (Abb.8 d), so zeigt sich, dass diese etwas in den Hintergrund hineingewachsen sind. Das liegt an der stärkeren Textur des Vordergrundes gegenüber dem Hintergrund. Es hat sich daher im Zweifelsfall an den Grenzen des Objektes die Subpopulation des Vordergrundes gegenüber dem Hintergrund durchgesetzt. Dieses Phänomen wird bei lokalen Verfahren weithin als Verdickung des Vordergrundes (engl. *foreground fattening*) bezeichnet. Die Fehler, welche an der linken Kante zu sehen sind, ergeben sich aufgrund von Verdeckungen infolge der unterschiedlichen Perspektiven [41].



a) linkes Eingabebild b) tatsächliche Disparitäten

c) Fehlerbild d) rekonstruierte Disparitäten

Abbildung 8: Bei Fenstermethoden mit starrem Fenster kommt es in der Nähe von Disparitätsdiskontinuitäten zu einer Verdickung des Vordergrundes. In diesen Regionen befinden sich mehrere Disparitätspopulationen innerhalb desselben Fensters, und weil der Vordergrund häufig stärker texturiert ist als der Hintergrund, setzt sich in der Regel die Disparitätspopulation des Vordergrundes durch. Zur Berechnung der Disparitäten setzte man ein starres 21x21 Fenster ein und nutzte die SAD als Ähnlichkeitssmaß. Quelle: [41].

Aus Effizienzgründen verwenden viele lokale Stereoalgorithmen feste Fenster mit fixer Größe. Neben der erwähnten Vordergrundverdickung führt das zu weiteren Problemen wie das Abrunden von Ecken und das Verschwinden von dünnen Objekten und filigranen Details, ab-

hängig von der Größe des Fensters [54]. Kommt nahe Disparitätskontinuitäten, wie bei der linken Kante des Vordergrundobjektes in Abbildung 8, zusätzlich noch die Verdeckung erschwerend hinzu, beruht die Zuordnung auf der Ähnlichkeit der halb verdeckten Region zum Hintergrund bzw. zum Vordergrund [55]. In der Regel kommt es auch in diesem Fall zu einer Verdickung des Vordergrundobjektes. Die Situation stellt sich ähnlich wie bei der Vordergrundverdickung dar. Bei der Vordergrundverdickung, wie oben erwähnt, existieren allerdings keine halb verdeckten Regionen im Referenzbild. Eine Situation mit einer halb verdeckten Region zeigt Abbildung 9.

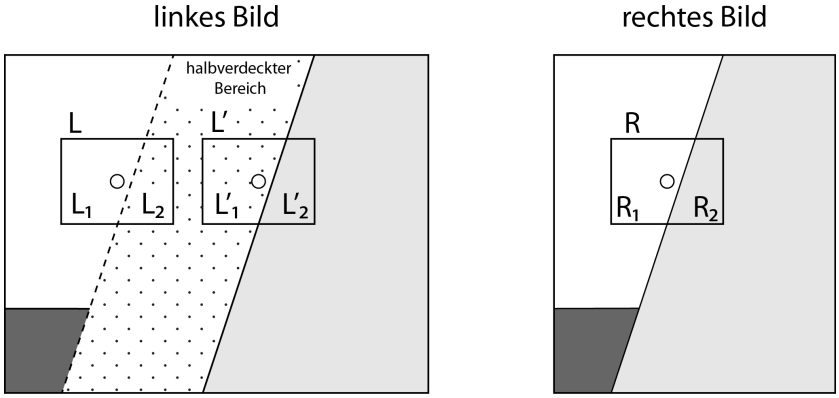



Abbildung 9: In der Nähe von Diskontinuitäten mit halb verdeckten Regionen tritt ein Entscheidungskonflikt auf. Welche Fenster übereinstimmen, hängt von der Beschaffenheit der halb verdeckten Region ab. Quelle: [55].

Die richtige Zuordnung des Korrelationsfensters $R$ wäre $L$. Dennoch kommt es öfters zu der inkorrekten Korrespondenz von $R$ und $L'$. Für eine genaue Betrachtung des Sachverhaltes werden die Fenster in zwei Hälften geteilt. Sei nun $k(a,b)$ eine beliebige Ähnlichkeitsfunktion, welche die Ähnlichkeit der zu vergleichenden Regionen $a$ und $b$ numerisch widerspiegelt mit einem niedrigen Wert für hohe Ähnlichkeit. Die Werte für $k(L_{1,}R_1)$ und $k(L'_{2,}R_2)$ sollten demnach gering sein, weil sie dieselbe Region im Raum beinhalten. Folglich ergibt sich die Zuordnung aus der Ähnlichkeit zwischen $L_2$ und $R_2$ respektive $L'_1$ und $R_1$. Liefert $k(L'_{1,}R_1)$ ein niedrigeres Ergebnis als $k(L_{2,}R_2)$, resultiert daraus eine falsche Zuordnung. Gleicht die halb verdeckte Region also eher dem Hintergrund, kongruiert das Korrelationsfenster $R$ mehr mit $L'$ als mit $L$. Dieser Fall tritt häufiger ein und das Vordergrundobjekt wächst dann in die halb verdeckte Region hinein.

Um den Problemen, die starre Fenster mit fixer Größe verursachen, beizukommen, versucht man Form und Größe der Fenster adaptiv anzupassen.

## 3.2 Formen der Aggregation

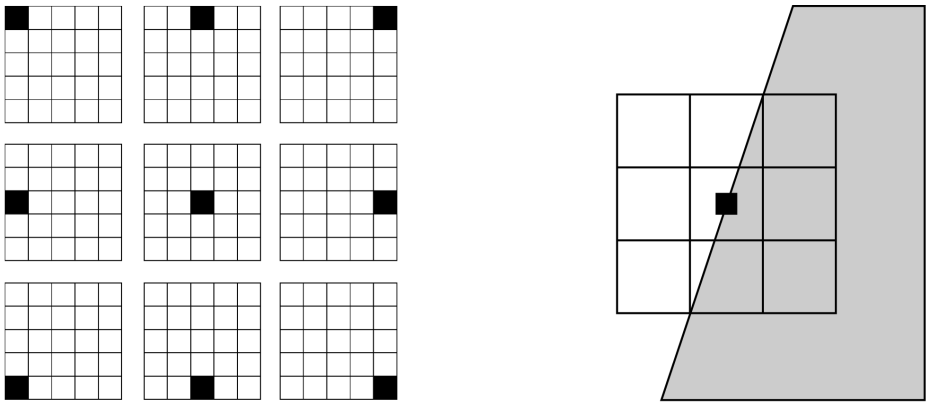
Bei dem traditionellen Ansatz für die Aggregation von Übereinstimmungskosten verwendet man gleichmäßig große Bereiche, welche das Pixel von Interesse umgeben. Die Bereiche besitzen meistens wegen der Möglichkeit einer schnellen rekursiven Implementierung die Form eines einfachen rechteckigen Fensters [56][48]. Die rigide Struktur der Bereiche verursacht die oben erwähnten Komplikationen. Als adäquater erweisen sich Bereiche variabler Größe und Form. Die Idee dahinter ist, für jedes Pixel die für einen Ähnlichkeitsvergleich am besten geeigneten Pixel in seiner Nachbarschaft herauszufinden. Der Unterstützungsbereich soll pro Disparität variiert werden, um sich den lokalen Gegebenheiten anzupassen. Damit können Zuordnungsambiguitäten in leicht texturierten Regionen vermieden und die Genauigkeit in der Nähe von Objektgrenzen erhöht werden [57].

### 3.2.1 Ansatz mit adaptiven Fenstern

Kanade und Okutomi [58][59] realisieren einen Algorithmus für die automatische Selektion der Fenstergröße, abhängig von lokalen Eigenschaften der Bilder. Sie erstellen ein statistisches Modell für Intensitäts- und Disparitätsvariation innerhalb eines Fensters. Dieses Modell erlaubt es ihnen, abzuschätzen, wie stark solche Variationen in einem bestimmten Bereich die Schätzung der Disparitäten beeinflussen. Mit Stichproben aus einem Fenster, unter Verwendung einer initialen Disparitätsverteilung, versuchen die Autoren mit Hilfe des Bayes-Theorems die tatsächliche Verteilung zu schätzen. Sie bestimmen Mittelwert und Varianz einer bedingten Verteilung, welche den Unterschied zwischen der initialen und der tatsächlichen Verteilung widerspiegelt. Den Mittelwert verwenden Kanade und Okutomi als Korrektur für die anfängliche Schätzung und die Varianz als Grad für die Unsicherheit der Korrektur. In einem iterativen Prozess können sie sich der tatsächlichen Disparitätsfunktion annähern, indem sie ihre Schätzung jedes Mal um den Mittelwert korrigieren. Die Varianz ermöglicht ihnen, nun auch das Fenster entsprechend der lokalen Umstände zu adaptieren. Beginnend mit einem kleinen Fenster erweitern Kanade und Okutomi in [59] dieses in jede Richtung, solange sich die Varianz nicht erhöht (und damit die Unsicherheit der Korrektur), oder bis ein vorher gesetztes Limit erreicht wird. Das Fenster passt sich dadurch lokal an. Es ist in texturfreien Regionen groß genug, um Mehrdeutigkeiten zu verhindern und nahe Tiefendiskontinuitäten ge-

rade ausreichend kompakt, um möglichst nur eine Disparitätspopulation zu beinhalten. In [58] wird schlicht aus verschieden großen rechteckigen Fenstern jenes mit der geringsten Unsicherheit ausgewählt. Nachteile des Algorithmus sind dessen Abhängigkeit von der initialen Disparitätsschätzung, welche das Endergebnis stark beeinflussen kann, und der hohe rechnerische Aufwand aufgrund des aufwendigen Modells [60][56].

Eine simple adaptive Fenstermethode wie die in [58] wird in [61][62][63] genutzt. Anstatt jedoch aus unterschiedlich großen Fenstern das geeignetste auszuwählen, bedient man sich mehrerer Fenster der gleichen Größe aber mit jeweils verschobener Position. In [61][63] setzt man dafür insgesamt neun Fenster ein. Sie sind in Abbildung 10 a) demonstriert. Selektiert wird das Fenster mit den niedrigsten Zuordnungskosten, womit man erreicht, dass möglichst nur eine Disparitätspopulation innerhalb des gewählten Bereiches liegt. Das Disparitätsprofil selbst bestimmt quasi die Auswahl eines geeigneten Fensters.



a) b)

Abbildung 10: Der Sinn von variablen oder multiplen Fenstern ist, dass lediglich jener Bereich zur Bestimmung der Ähnlichkeit eingesetzt wird, welcher die Disparitätspopulation des Pixels von Interesse enthält. a) In [61] gebraucht man neun in ihrer Position leicht verschobene Fenster und selektiert das Fenster, welches die niedrigsten Kosten verursacht. b) In [64] unterteilen Hirschmüller et al. einen Bereich um das zentrale Pixel in neun Sektoren. Zur Korrelation werden die vier besten und der mittlere Sektor eingesetzt. Quellen: a) [61] b) [64].

Diese Vorgehensweise erweitern Hirschmüller et al. in [64] (siehe Abbildung 10 b)). Sie unterteilen einen Bereich um das Pixel von Interesse in 9 Sektoren und wählen nicht nur den Sektor (Fenster) mit den geringsten Zuordnungskosten für die Korrespondenzanalyse, sondern

die besten 4 plus den mittleren Sektor. Damit erzielen sie bessere Ergebnisse als nur mit dem besten Fenster. Die Anordnung der Sektoren ist jedoch etwas anders, weil außer dem mittleren Sektor kein anderer Sektor mehr das zentrale Pixel enthält und sich die Sektoren nicht überlappen. Diese Idee lässt sich auf mehr oder auf weniger als 9 Sektoren übertragen. So testen sie auch Konfigurationen, wo sie aus 25 Sektoren die besten 12 auswählen plus den zentralen Sektor oder aus 5 die besten 2 und den zentralen Sektor. Bei der Konfiguration mit 5 Sektoren werden die Sektoren allerdings so unterteilt, dass sie sich wieder überlappen.

Chan et al. [65] komplettieren die Multiple-Fenster-Methode von [61] mit einer Methode zur Anpassung der Fenstergröße. Nachdem sie das beste der 9 Fenster bestimmt haben, reduzieren sie die Dimensionen des Fensters iterativ solange bis entweder die relativen Zuordnungskosten höher werden oder das kleinstmögliche Fenster erreicht wurde.

Einen ähnlichen Ansatz wählen Demoulin und Droogenbroeck bei ihrem Verfahren [66]. Aber statt 9 Fenster nutzen sie nur 4 Fenster mit unterschiedlichen Positionen. Noch vor der Selektion des besten Fensters werden die Dimensionen jedes Fensters individuell erweitert. Dafür verwenden sie zwei Kriterien: (i) Den Wert einer definierten horizontalen und vertikalen Varianz, (ii) den Wert der Zuordnungskosten. Bei niedriger horizontaler Varianz wird das Fenster horizontal expandiert und vertikal bei niedriger vertikaler Varianz. Die Expansion wird nur dann ausgeführt, wenn sich die Kosten nicht erhöhen. Eine getrennte Erweiterung in horizontale und vertikale Richtung ermöglicht eine beliebig rechteckige Form des Korrelationsbereiches.

Bereiche beliebiger Form lässt Veksler in [60] zu. Der Bereich um das Pixel von Interesse wird dabei als Graph repräsentiert. Die Autorin versucht innerhalb dieses Graphen, den Zyklus mit einem bestimmten minimalen Verhältnis zu finden. Das Verhältnis bezieht sich auf eine vorher definierte Ähnlichkeitsmetrik und der Zyklus soll dieses minimieren. Das Verfahren stammt aus der Graphentheorie. Der gefundene Zyklus umschließt den für die Zuordnung verwendeten Bereich und ist nicht auf Rechtecke limitiert. Mit der Methode kann ein in Bezug auf die Metrik optimales Fenster gefunden werden. Der Aufwand ist aber etwas höher als für andere Verfahren mit variablen Fenstern.

In [67] bedient sich Veksler der Integral-Bild-Technik, um damit effizient Korrelationskosten unterschiedlich großer Quadrate berechnen und vergleichen zu können. Dies ermöglicht eine Echtzeitabarbeitung des Algorithmus. Für jedes Pixel werden pro Disparität mehrere verschieden große Quadrate revidiert, wobei das mit den niedrigsten Kosten gewinnt und beibehalten wird. Um Fenster unterschiedlicher Größe zu vergleichen, werden die Kosten unter Einbezie-

hung der Fenstergröße normiert. Ein Pixel taucht nun in seinem eigenen quadratischen Fenster sowie in den Fenstern der Nachbarschaft auf. Mittels dynamischer Programmierung sucht die Autorin pro Pixel das beste Fenster, in dem ein Pixel auftaucht, welches schließlich das Fenster zur Berechnung der Korrelationskosten darstellt. Das hilft nahe Diskontinuitäten, Fenster auszuwählen, bei denen das Pixel von Interesse am Rande liegt und nicht in der Mitte. Aus Effizienzgründen beschränkt sie die Revision der Fenstergrößen auf etwa 6 für jedes Pixel, unter Berücksichtigung der bereits ermittelten optimalen Fenstergröße des Nachbarpixels.

Eine andere Strategie für eine adaptive Fenstergröße verfolgen Kang et al. in [68]. Sie berechnen nicht immer für jede Fenstergröße alle möglichen Disparitäten. Beginnend bei einem kleinen Fenster werden für alle Pixel nur die Disparitätshypothesen akzeptiert, welche einen gewissen Grad an Sicherheit aufweisen. Anschließend vergrößert man das Fenster. Nun müssen nur mehr für jene Pixel Hypothesen berechnet werden, die nicht bereits eine zugewiesene Disparität besitzen. Erneut werden lediglich die sichersten Hypothesen übernommen. Diesen Vorgang wiederholen Kang et al. genau zwölf Mal. Die Sicherheit einer Disparitätshypothese berechnen sie über die lokale Varianz der Kostenfunktion in der Nähe dieser Disparität. Je höher die Varianz, umso verlässlicher ist die Hypothese.

Statt das optimale Fenster pro Pixel und Disparität anhand der niedrigsten Korrelationskosten auszusuchen, entscheiden sich Adhyapak et al. [56] das optimale Fenster mithilfe der Kostenfunktion zu bestimmen. Zu diesem Zweck implementieren die Autoren einen Zuverlässigkeitstest, welcher die gesamte Kostenfunktion analysiert und deren Zuverlässigkeit bestimmt. Dabei berücksichtigen sie sowohl lokale Variationen der Funktion als auch die Deutlichkeit des globalen Minimums. Das Fenster mit dem zuverlässigsten Funktionsverlauf gewinnt, wobei auch die Varianz der Fenster in die Entscheidung einfließt. Mithilfe der Varianz und des Zuverlässigkeitstests lassen sich auch verdeckte Pixel identifizieren ganz ohne Links-rechts-Konsistenztest.

Eine andere Herangehensweise zum Erlangen von geeigneten Bereichen für die Korrespondenzanalyse besteht in der Segmentierung des Bildes beispielsweise anhand von Farbinformation. Die resultierenden Segmente werden anschließend als Korrelationsbereiche genutzt. Dabei wird implizit angenommen, dass die Disparität innerhalb der Segmente nur gering variiert. Trifft dies zu, erhält man eine gute Disparitätsbestimmung, wenn die Segmente nicht zu klein ausfallen. Dieser Ansatz benötigt allerdings a priori eine genaue Farbsegmentierung. Daher ist

er für eine Realisierung in einem Echtzeitstereosystem nicht geeignet, es sei denn, man benützt ähnlich wie [69] eine schnelle Farbsegmentierung mit dennoch akzeptablen Resultaten [70].

### 3.2.2 Ansatz mit adaptiver Gewichtung

Ebenfalls weite Verbreitung in der Stereoanalyse finden Verfahren, welche das Fenster an sich unverändert lassen und dafür innerhalb des Bereiches dieses Fensters die Bildpunkte mit adaptiven Gewichten versehen. Die Gewichte bestimmen den Einfluss der Bildpunkte auf die Korrelationskosten des zentralen Bildpunktes. Diese Methode besitzt den Vorteil einer höheren Genauigkeit gegenüber adaptiven Fenstern. Sie erlaubt jedoch kaum den Einsatz inkrementeller Berechnungsmodelle wie in [71][56], weil die Unterstützungsregionen im Allgemeinen zu unregelmäßig sind. Einige Algorithmen dieses Konzeptes spezifizieren die Unterstützungsregionen anhand der Informationen aus den Fenstern im linken und rechten Bild und erhöhen damit die Korrektheit [57].

Einer der Ersten, welcher adaptive Gewichtung einsetzt, ist Prazdny [8]. Die Unterstützungsfunktion zur Eruierung der Gewichte ist aber nicht explizit auf ein Fenster beschränkt. Sie ermöglicht quasi globalen Support. Prazdny stellt ein Kohärenzprinzip auf, dessen Eigenschaften sich in der Funktion zur Gewichtungsberechnung widerspiegeln und das auch transparenten Oberflächen Rechnung tragen soll. Der Algorithmus benötigt bereits initiale Disparitätshypothesen, um die Unterstützung für eine Disparität zu ermitteln, die ein Pixel von einem anderen Pixel erhält.

In [72][73] berechnet man die eingesetzten Gewichte ausgehend von einem zentralen Bildpunkt radial nach außen hin und setzt bereits Fenster zur Begrenzung der Unterstützungsfunktion ein. Beide Methoden verwenden Farbinformation. In [72] nimmt Darrell an, dass ausgehend vom Zentrum einem Strahl nach außen folgend, der erste Kontrast bereits eine Tiefendiskontinuität darstellt. Man verwendet dabei die Farbwerte der Pixel, um Kontrast festzustellen. Dementsprechend werden Informationen von Pixeln, die hinter dieser Kontrastgrenze liegen, kaum gewichtet. Dies entspricht in etwa einer Farbsegmentierung des Bildes und der Verwendung der Segmente für die Gewichtung. Das Verfahren benötigt keine initiale Disparitätsschätzung. Xu et al. [73] definieren den Grad des Supports entlang des Strahles mithilfe dreier Indikatoren. Der erste basiert auf initialen Disparitätsdichtefunktionen, bei denen das Gewicht eines Bildpunktes von der Gewichtung und Funktion des Vorgängers abhängt. Je weiter man

nach außen geht, desto weniger wird gewichtet. Der zweite beruht auf Differenz der initialen Disparitätsverteilungen. Der dritte Indikator wertet den Farbunterschied zum zentralen Pixel aus.

Die Idee von [72] wird später in [74] wieder aufgegriffen. Diesmal werden die Gewichte allerdings nicht entlang eines Strahles berechnet, sondern es wird versucht, den minimalen Pfad zwischen dem zentralen Pixel und jenem Pixel im Korrelationsfenster zu finden, für welches man die Gewichte berechnet. Dieser minimale Pfad kann wie ein Strahl die direkte Verbindung zwischen den Pixeln darstellen. Er muss es aber nicht, weil der Pfad auch Kurven beinhalten könnte.

Yoon und Kweon [75] berechnen die Gewichte innerhalb eines Fensters mit Farbvergleichen und anhand der räumlichen Distanz zum zentralen Pixel, wie es ansatzweise schon in [73] geschieht. Dies entspricht dem Grundgedanken des bilateralen Filterns [76]. Je näher sich ein Bildpunkt am Zentrum befindet und je höher die farbliche Ähnlichkeit zum zentralen Bildpunkt, desto größer die Gewichtung, sprich der Beitrag dieses Pixels zu den Zuordnungskosten des Fensters. Die Autoren benötigen keine anfängliche Disparitätsschätzungen, weil die Auswertung der Unterstützungsgewichte sich aus der verfügbaren Information innerhalb des Fensters herleitet. Die eingesetzten Gewichtungsfunktionen unterscheiden sich zum bilateralen Filtern dadurch, dass Yoon und Kweon zusätzlich die Gewichte aus beiden Bildern miteinander kombinieren.

Anstatt einer aufwendigen Farbsegmentierung setzen Gong und Yang [77] auf eine kantenbasierte Segmentierung, welche effizienter zu realisieren ist, ähnlich wie in [72]. Ihr Algorithmus ist für Echtzeitverarbeitung ausgelegt. Die Autoren aggregieren die Kosten innerhalb eines Fensters in einem iterativen Prozess mithilfe eines Filters. Diese filterbasierte Aggregation erleichtert die Implementierung auf Graphikhardware.

Ein Mean-Shift-Algorithmus segmentiert in [69] das Referenzbild und ermöglicht so eine adaptive Gewichtung innerhalb eines Fensters anhand der Segmente. Die German-McClure-Funktion, angewendet auf die Zuordnungskosten, reduziert den Einfluss von Ausreißern. Um die Laufzeit zu optimieren und unabhängig von der Fenstergröße zu machen, entwickeln Gerrits und Bekaert eine Variation der inkrementellen Berechnungsmodelle, welche normalerweise bei fensterbasierten Methoden ohne adaptive Gewichtung eingesetzt wird. Zur Eliminierung von Artefakten, resultierend aus einer möglicherweise falschen Segmentierung, kom-

binieren die Autoren abschließend die berechneten Disparitätskarten beider Stereobilder miteinander. Mit ihren eingesetzten Techniken erreichen sie Ergebnisse, die mit globalen Algorithmen vergleichbar sind, aber mit einem Laufzeitverhalten geeignet für Echtzeitsysteme.

Ähnliches gilt für [78]. Auch hier unterteilt ein Mean-Shift-Algorithmus das Bild in Segmente. Weil aber die Annahme, alle Pixel eines Segments haben gleiche Disparitäten, nicht immer zutrifft, verlassen sich Tombari et al. nicht nur auf die Kosten aggregiert über ein Segment, sondern kombinieren für die Aggregation zwei Bildbereiche. Den ersten Bereich erhalten sie über die Segmentierung, indem sie, wie schon andere, die Segmente als Aggregationsbereich gebrauchen. Der zweite Bereich besteht aus einem kleinen rechteckigen Fenster um das Pixel von Interesse. Der Bereich, resultierend aus der Segmentierung, ist besonders nützlich nahe Diskontinuitäten und bei nur leicht texturierten Gebieten, wohingegen das rechteckige Fenster den Pixeln in der Nähe des Pixels von Interesse eine stärkere Gewichtung verleiht. Das ist bei großen Segmenten mit leichter Neigung von Bedeutung, wo nicht alle Pixel die gleiche Disparität besitzen, sowie in stark texturierten Gebieten, wo sich Segmente üblicherweise nur aus einigen wenigen Pixeln zusammensetzen.

# 4 Dynamische Progammierung als globales Echtzeitverfahren

Als Beispiel für eine globale Methode wird die Optimierung einer Energiefunktion mit dynamischer Programmierung angeführt. Dynamische Programmierung eignet sich besonders gut für eine parallele Implementierung. Die Laufzeit kann damit drastisch gesenkt werden, wobei die guten Resultate der Methode erhalten bleiben. Hier werden die Ansätze dynamischer Programmierung entlang von Zeilen und dynamischer Programmierung auf einen Baum diskutiert. Sie sind für das Verständnis des nachfolgend implementierten Stereoalgorithmus wesentlich, weil dieser eine Kombination beider Ansätze verwendet.

## 4.1 Globale Methoden

Globale Stereoverfahren inkludieren bei der Tiefenbestimmung eines Pixels alle anderen Pixel im Referenzbild und deren Disparitäten. Lokale Methoden berücksichtigen im Gegensatz dazu immer nur einen Ausschnitt des Bildes. Um das Ziel des globalen Supports zu erreichen, minimieren die Verfahren in der Regel eine Energiefunktion. Diese Energiefunktion lässt sich aus dem Bayes'schen Theorem herleiten. Das Theorem und die Bayes-Statistik gelten als fundamentale Theorien in Schätzungs- und Entscheidungsprozessen [79]. Laut dem Theorem ist die A-posteriori-Verteilung $p(x \mid y)$ einer unbekannten Größe $x$ gegeben durch eine Anzahl von Beobachtungen $y$ mit der bedingten Verteilung $p(y \mid x)$, kombiniert mit der A-priori-Verteilung $p(x)$ der unbekannten Größe $x$:

$$p(x \mid y) = \frac{p(y \mid x) \cdot p(x)}{p(y)} \tag{4.1}$$

wobei $p(y) = \int_x p(y \mid x) \cdot p(x)$ konstant ist. Das bedeutet, $p(y)$ wird für einen Entscheidungsprozess nicht benötigt und kann weggelassen werden.
Es geht nun darum, eine Disparitätskarte zu finden, welche mit hoher Wahrscheinlichkeit der tatsächlichen Disparitätskarte entspricht. Sei $\Delta$ eine Disparitätskarte der in den Bildern $I_l$ und $I_r$ abgelichteten Szene, so ergibt sich das Bayes'sche Theorem zu:

$$p(\Delta \mid I_l I_r) = p(I_l I_r \mid \Delta) \cdot p(\Delta) \tag{4.2}$$

Auf diese Weise lässt sich ermitteln, wie wahrscheinlich eine Disparitätskarte $\Delta$ ist. Der erste Term auf der rechten Seite der Gleichung wird allgemein als Datenterm beschrieben und misst, wie gut die Disparitätskarte $\Delta$ mit den beobachteten Bildern übereinstimmt. Dazu eruiert man, wie stark die Pixel im Referenzbild den anhand der Disparitätskarte korrespondierenden Pixeln im anderen Bild entsprechen. Der zweite Term auf der rechten Seite verkörpert das A-priori-Modell und gibt an, wie wahrscheinlich a priori eine Disparitätskarte $\Delta$ ist, noch bevor die Bilder aufgenommen werden. Bei globalen Stereoverfahren nutzt man zumeist eine Glattheitsbedingung als A-priori-Modell.

Eine Disparitätskarte mit größtmöglicher Wahrscheinlichkeit erhält man schließlich durch eine maximale a posteriori (MAP) Verteilung $\hat{\Delta} = arg\,max\, p(\Delta \mid I_l I_r)$. Dieses $\hat{\Delta}$ beinhaltet die optimalen Korrespondenzen bezüglich des gewählten Modells [7].

Aus konventionellen Gründen und für eine bequemere Darstellung nimmt man den negativen Logarithmus beider Seiten von Gleichung (4.2):

$$-\log(p(\Delta \mid I_l I_r)) = -\log(p(I_l I_r \mid \Delta)) - \log(p(\Delta)) \tag{4.3}$$

Das Maximierungsproblem wird dadurch zu einem Minimierungsproblem, wobei die negative logarithmische Wahrscheinlichkeit als eine Art Energiefunktion angesehen werden kann, welche es zu minimieren gilt:

$$E = E_d + E_s \tag{4.4}$$

$E_d$ entspricht dem Datenterm und $E_s$ dem A-priori-Modell. Häufig stellt man $E_s$ einen Kontrollparameter $\lambda$ voran, um so den Einfluss der a priori Annahme zu steuern. Globale Verfahren, welche eine Energiefunktion minimieren, müssen beide Terme $E_d$ und $E_s$ spezifizieren. Den Datenterm $E_d$ verwirklicht man einfach als Summe der Zuordnungskosten von Pixel:

$$E_d = \sum_{p \in I_{ref}} m(p, d_p) \tag{4.5}$$

Gern setzt man bei $m(p, d_p)$ auf den absoluten Intensitätsunterschied $|I_{ref}(p) - I_{ziel}(p - d_p)|$ zwischen dem Pixel $p$ im Referenzbild und Pixel $p - d_p$ im Zielbild [80].

Die populärste Art das A-priori-Modell zu spezifizieren geht über *Markov Random Fields* (MRF, auch Markow-Netzwerke). Dank des Hammersely-Clifford Theorems [79] weiß man, dass für ein MRF die Wahrscheinlichkeitsverteilung $p(x)$ eine Gibbs- oder Boltzmann-Verteilung ist, deren negative logarithmische Wahrscheinlichkeit als eine Summe paarweise interagierender Potentiale geschrieben werden kann. Gewisse Sorten von Potentialen, wie das Potts-Modell oder Derivate davon, reduzieren die Komplexität des Minimierungsprozesses und werden deshalb häufig eingesetzt. Das Potts-Modell als Glattheitsbedingung im A-priori-Modell hat die Form:

$$s_p(d_p, d_q) = \begin{cases} 0 & wenn \quad d_p = d_q \\ (w_1 & wenn \quad |d_p - d_q| = 1 \ ) \\ w_2 & andernfalls \end{cases} \tag{4.6}$$

$E_s$ als Summe der paarweise interagierenden Potentiale lautet demnach:

$$E_s = \sum_{(p,q)\in N} s_p(d_p, d_q) \tag{4.7}$$

$d_p$ bezeichnet die Disparität des Pixels *p,* $d_q$ die Disparität des Pixels *q.* Zwischen beiden Pixeln existiert eine Nachbarschaftsbeziehung *N*. $w_1$ und $w_2$, wobei $w_1 < w_2$, sind Strafkosten, welche Disparitätssprünge bestrafen, aber nicht unmöglich machen. Der in Formel (4.6) in Klammern stehende Term wird oft als Erweiterung gegenüber dem ursprünglichen Potts-Modell hinzufügt, um schräg verlaufenden Oberflächen Rechnung zu tragen. Das Nachbarschaftssystem *N* beinhaltet die Pixelpaare, für welche die Kosten $s_p(d_p, d_q)$ ausgewertet werden.

Die Wahl des Nachbarschaftssystems hat Auswirkungen auf die Komplexität des Minimierungsproblems sowie auf die Korrektheit der Ergebnisse. So ist die 2-D-Optimierung von Gleichung (4.4) mit einer Vierer-Nachbarschaft im Allgemeinen ein NP-hartes Problem, dessen Approximationen jedoch sehr akkurate Resultate liefern. Die 1-D-Optimierung kann mittels dynamischer Programmierung effizient gelöst werden, allerdings mit etwas schlechteren Ergebnissen als die 2-D Optimierung [81].

Das Potts-Modell fungiert als Glattheitsbedingung, weil es die Zuweisung von Disparitäten zu Pixeln bestraft, welche sich von den Disparitäten der Nachbarschaftspixel unterscheiden. Nichtsdestoweniger ermöglicht das Modell Disparitätssprünge, falls diese durch den Datenterm unterstützt werden und die Strafkosten $w_1$ und $w_2$ nicht zu hoch sind.

Wie bei jedem Modell, welches die Wirklichkeit nachzuahmen versucht, kommt man bei einer Energiefunktion irgendwann an die Grenze, wo der Modellfehler eine weitere Verbesserung der Ergebnisse unmöglich macht. Mit einer bestimmten Wahl für den Datenterm und das A-priori-Modell macht man immer implizit gewisse Annahmen und setzt manche Sachverhalte voraus. Je nach Beschaffenheit einer Szene kann beispielsweise die Glattheitsbedingung mehr oder weniger stark zutreffen. Beim Datenterm können Ambiguitäten und Unterschiede von Farb- oder Intensitätswerten korrespondierender Elemente falsche Kosten verursachen (siehe Abschnitt 2.4). Dieser Modellfehler spielt in Bezug auf die Resultate eine wichtige Rolle. Heutige Optimierungsverfahren für die Gleichung (4.4) wie Belief Propagation [82] und Graph-Cuts [83] schaffen es, die Energie so weit zu minimieren, dass sie nahe am globalen Optimum liegt. Sie produzieren mitunter eine Lösung, deren Energie niedriger ist als die Energie der tatsächlichen Disparitäten (engl. *ground truth*).

Dieser Umstand verlangt folglich nach besseren Modellen, welche die Geometrie einer Szene geeigneter beschreiben. Hier befindet man sich in einer Konfliktsituation. Generell existieren zwei divergierende Ansätze. Entweder man verwendet sehr elaborierte Modelle mit vielen a priori Annahmen oder man versucht, mit so wenig Annahmen wie möglich auszukommen. Algorithmen mit anspruchsvollen Modellen schneiden bei Szenen, für welche sie zugeschnitten wurden, besser ab als Algorithmen mit wenigen Bedingungen. Nur bleibt ihr Einsatz auf diese Szenen beschränkt. Algorithmen mit wenigen Beschränkungen taugen zum Gebrauch bei unterschiedlichen Szenen, obwohl ihre Ergebnisse für einige Szenen hinter den der spezialisierten Algorithmen bleiben. Optimal oder minimal bedeutet in diesem Zusammenhang folglich immer optimal bezüglich einer eingesetzten Energiefunktion und der darin vorkommenden Annahmen [80][84].

## 4.2 Grundlagen dynamischer Programmierung

Dynamische Programmierung ist eine mathematische Methode, welche die rechnerische Komplexität von Optimierungsproblemen reduziert. So lässt sich die Optimierung der Energiefunktion unter bestimmten Umständen mittels dynamischer Programmierung in polynomieller Zeit bewerkstelligen. Die Methode fußt auf einer tabellarischen Herangehensweise zur Lösung vielschichtiger Probleme. Sie wurde erstmals von Richard Bellmann im Jahre 1957 vorgestellt [85] und folgt dem Teile- und Beherrsche-Prinzip, in dem eine komplexe Aufgabenstellung in kleinere, leichter lösbare Teilprobleme zerlegt wird. Durch die

Kombination der optimalen Lösungen der Teilprobleme gelangt man zur optimalen Lösung des Ganzen. Laut [86] besteht das Auffinden einer optimalen Lösung per dynamischer Programmierung aus vier Abschnitten:

(1) Beschreibung der Struktur einer optimalen Lösung

(2) Rekursive Definition des Wertes einer optimalen Lösung

(3) Ermittlung des Wertes einer optimalen Lösung durch sukzessives optimales Lösen der Teilprobleme

(4) Deduktion einer zugehörigen optimalen Lösung aus den bereits eruierten Daten

Dynamische Programmierung setzt man in unterschiedlichsten Aufgabengebieten ein, wie Texterkennung, Spracherkennung oder im Bereich der digitalen Kommunikation, wo Viterbi 1967 einen Algorithmus zur Decodierung von Faltungscodes entwickelte. Mit der Zeit fand eine Verallgemeinerung des Algorithmus von Viterbi auch in der Stereoanalyse Verwendung, weswegen in der Literatur statt dynamischer Programmierung auch der Begriff Viterbi-Algorithmus vorkommt [87].

Zu Beginn wurde dynamische Programmierung für kantenbasierte Zuordnungsverfahren benutzt, welche eine spärliche Disparitätskarte[5] kreieren. Kantenbasierte Verfahren wurden zu dieser Zeit unter anderem auf Grund des geringeren rechnerischen Aufwandes verwendet (e.g. [88][89][90]). So löst Baker mit dynamischer Programmierung in [88] zuerst das Korrespondenzproblem zwischen Kanten. Die gefundenen Korrespondenzen dienen in einem weiteren Schritt zur Begrenzung der Zuordnung auf Pixelebene, bei dem ebenfalls dynamische Programmierung die Korrespondenzen bestimmt. Vergleichbar verfahren später Sadeghi et al. [91]. Sie bestimmen mithilfe eines merkmalsbasierten Verfahrens zuerst initiale Tiefenschätzungen und füllen die Lücken zwischen den Schätzungen pixelbasiert mit dynamischer Programmierung.

In jüngerer Zeit konzentriert man sich auf intensitätsbasierte Verfahren, welche eine dichte Disparitätskarte hervorbringen. Dabei findet man mit dynamischer Programmierung Korrespondenzen entweder für Bildpunkte entlang einer Bildzeile oder für in einer Baumstruktur angeordnete Bildpunkte.

---

5 Spärliche Disparitätskarten beinhalten nicht für jedes Pixel eine Disparität, sondern nur für bestimmte Merkmale im Bild.

## 4.3 Dynamische Programmierung entlang von Zeilen

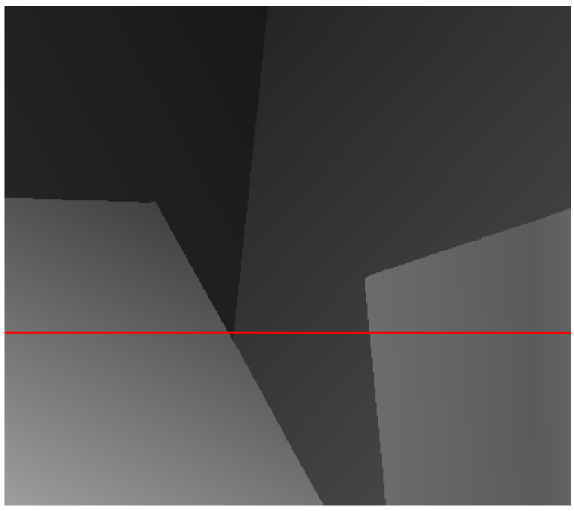
Weil in einem epipolar begradigten Stereobildpaar Korrespondenzen immer entlang von Bildzeilen auftreten, hat man anfänglich dynamische Programmierung im Suchprozess zum Auffinden der richtigen Korrespondenzen unabhängig für jede Bildzeile einzeln eingesetzt. Angenommen das DRV wurde bereits ermittelt, so operiert das Verfahren auf den einzelnen DRBs für jede Zeile. Das DRV ist, wie schon in Kapitel 2.5 beschrieben, eine dreidimensionale Struktur $(x, y, d)$, wobei $x$ Spalten, $y$ Bildzeilen und $d$ Disparitätsstufen repräsentieren. Im DSV werden alle Zuordnungskosten gespeichert. Als DRB bezeichnet man die 2-D-Matrix für eine bestimmte Bildzeile $y_i$ im DRV.

Abbildung 11 c) zeigt ein solches DRB für Zeile 250 des Stereobildpaares Venus aus dem Middlebury-Datensatz mit dem linken Bild als Referenzbild. Das Referenzbild selbst ist in Abbildung 11 a) dargestellt. Die dazugehörigen tatsächlichen Disparitäten werden in Abbildung 11 b) gezeigt. Zur Erstellung des DRBs wurde als Ähnlichkeitsmetrik die Summe der absoluten Differenzen der RGB-Werte zwischen zwei Pixel eingesetzt.

DRB sind 2-D-Bilder. Sie repräsentieren die Kostenmatrix für die Zuweisungen der Pixel einer Bildzeile im linken Stereobild zu dem Pixel der korrespondierenden Bildzeile im rechten Stereobild. Sei $x_p$ ein Pixel in der linken Bildzeile, so finden sich die Zuordnungskosten zu dem Pixel $x_{p-d}$ im rechten Bild an der Stelle $(x_p, d)$ in der Kostenmatrix. Für die hier gewählte Darstellung als 2-D-Bild gilt, je dunkler ein Pixel desto höher sind die Zuordnungskosten der entsprechenden Bildpunkte im linken und rechten Bild.

Alternativ zu der hier beschriebenen Anordnung des DRBs respektive des DRVs besteht noch eine zweite Möglichkeit der Anordnung, welche jedoch nicht so speichereffizient ist. Die Dimensionen der DRBs ergeben sich dabei aus der linken Bildzeile und der rechten Bildzeile. Bei der dynamischen Programmierung startet die Suche nach dem minimalen Pfad in der linken unteren Ecke und endet in der rechten oberen Ecke. Beispiele für eine solche Anordnung finden sich in den Artikeln [92][50][93].


a) linkes Bild des Stereobildpaares Venus
(Zeile 250 weiß markiert)



b) tatsächliche Disparitäten von a)
(Zeile 250 rot markiert)


c) DRB für Zeile 250 des Stereobildpaares Venus


d) DRB für Zeile 250 des Stereobildpaares Venus
(tatsächliche Disparitäten rot markiert)

Abbildung 11: Dynamische Programmierung kann bei epipolar begradigten Bildern zur effizienten Bestimmung der Disparitäten herangezogen werden. Die Methode arbeitet unabhängig auf den einzelnen DRBs jeder Zeile. c) zeigt ein DRB für die in a) markierte Zeile. Den in d) rot markierten Pfad gilt es, ausfindig zu machen.

Dementsprechend, wie zutreffend die Zuordnungskosten ermittelt wurden und wie angemessen die a priori Annahmen sind, sollte der Pfad mit den geringsten Kosten durch die Kostenmatrix jener Pfad sein, der die tatsächlichen Disparitäten widerspiegelt. In Abbildung 11 c) müsste ein solcher Pfad möglichst entlang weißer Pixel verlaufen, da diese niedrige Kosten bedeuten. Bei genauer Betrachtung lässt sich ein solcher weißer Pfad erahnen. Je ausgeprägter dieser Pfad desto besser die gewählte Ähnlichkeitsmetrik zur Erstellung des DRBs. Abbildung 11 d) zeigt den Pfad der tatsächlichen Disparitäten in Rot markiert, welchen man mit dynamischer Programmierung möglichst effizient zu eruieren versucht. Die Problematik der Gewinnung von Korrespondenzen zwischen Bildpunkten entlang von linker und rechter Bildzeile der Stereobilder kann somit durch ein Pfadfindungsproblem auf einem 2-D-Bild gelöst werden [90].

Der erste Schritt bei der dynamischen Programmierung ist, die Struktur einer optimalen Lösung zu beschreiben. Das geschieht mithilfe einer Energiefunktion. Die Energiefunktion besteht aus dem Datenterm und der Glattheitsbedingung als A-priori-Modell. Der Datenterm wird durch die Zuordnungskosten der Kostenmatrix repräsentiert. Bei der Glattheitsbedingung geht man davon aus, dass die Disparitäten fast überall im Bild nur stetig variieren. Es kann aber zu Tiefensprüngen innerhalb einer Szene kommen, hervorgerufen durch verschiedene Objekte mit unterschiedlicher Tiefe. Somit muss dem Pfad ebenfalls erlaubt werden, solche Sprünge zu vollziehen. Der rote Pfad in Abbildung 11 d) weist exemplarisch genau dieses Verhalten auf. Er ändert sich fast überall nur kontinuierlich außer bei Tiefendiskontinuitäten zwischen unterschiedlichen Objekten.

Die Pfadkosten für einen Punkt $(x_i, d_j)$ im DRB ergeben sich nun nach dem verwendeten Modell aus der Summe des Datenterms und der Glattheitsbedingung. Eine optimale Struktur lässt sich damit wie folgt definieren: Der minimale Pfad zu einem Knoten $(x_i, d_j)$ resultiert aus den Datenkosten von $(x_i, d_j)$, plus den Kosten zu jenem Vorgänger $(x_{i-1}, d_k)$ mit minimalen Kosten aus der Glattheitsbedingung und den niedrigsten Pfadkosten nach $(x_{i-1}, d_k)$. In anderen Worten, der Pfad nach $(x_i, d_j)$ über einen Vorgänger $(x_{i-1}, d_k)$ erschließt sich aus den Kosten des minimalen Pfades zu $(x_{i-1}, d_k)$, zuzüglich den Kosten nach $(x_i, d_j)$, welche unabhängig vom minimalen Pfad nach $(x_{i-1}, d_k)$ sind. Das heißt, um den minimalen Pfad nach $(x_i, d_j)$ zu finden, geht man einfach alle Vorgänger durch und wählt jenen, der in einen minimalen Pfad nach $(x_i, d_j)$ mündet. Wurde das Prinzip auch für die Vorgänger angewendet, wäre damit sichergestellt, dass dieser Pfad auch der minimale Pfad nach $(x_i, d_j)$ ist.

Indem jeweils die optimalen Pfadkosten zwischen zwei Knoten $(x_i, d_j)$ und $(x_{i-1}, d_k)$ bestimmt werden, erhält man eine optimale Lösung für den gesamten Pfad durch Kombination der optimalen Lösung der Teilprobleme. Cormen et al. [86] sprechen in diesem Zusammenhang von einer optimalen Teilstruktur. Sie stellt sicher, dass die gefundene globale Lösung eine optimale Lösung des gesamten Problems darstellt. Man könnte diese Form der Pfadfindung auch als einen Markow'schen Entscheidungsprozess bezeichnen. Einen Entscheidungsprozess bezeichnet man als Markow'schen Prozess, wenn in jeder Stufe das Verhalten des Prozesses nur von der aktuellen Stufe und den direkten Vorgängern abhängt [79].

Eine Struktur von Teilproblemen sollte ebenfalls, um dynamische Programmierung einsetzen zu können, eine geordnete Reihenfolge aufweisen, damit zum Lösen eines Teilproblems alle nötigen Ergebnisse bereits gelöster Probleme vorliegen. Im Fall der Korrespondenzsuche beginnt man die Suche nach dem minimalen Pfad üblicherweise an der linken Seite des DRBs und arbeitet sich dann zur rechten Seite durch. Auf der linken Seite im DRB kommt es bei den Spalten mit Index $x_0$ und $x_{dmax-1}$ zu einem schwarzen stufenweise kleiner werdenden Bereich in den Zuordnungskosten, weil hier keine korrespondierenden Pixel im rechten Bild existieren (siehe Abb. 11 c)).

Der zweite Schritt nach den zu Beginn des Kapitels postulierten Paradigmen dynamischer Programmierung besteht darin, rekursiv den Wert einer optimalen Lösung zu definieren. Die optimalen Kosten für das Erreichen eines jeden Punktes im DRB werden rekursiv wie folgt definiert [9]:

$$l(p,d) = m(p,d) + \min_{i \in D}\left(s(d,i) + l(q,i)\right) \tag{4.8}$$

*D* beschreibt die Menge der möglichen Disparitäten, *q* einen unmittelbaren Vorgänger und in diesem Fall den linken Nachbarn von *p*. $l(p,d)$ sind die minimalen Pfadkosten zum Punkt $(p,d)$ im DRB (von links kommend) und $m(p,d)$ die ursprünglichen Zuweisungskosten der Disparität *d* zum Pixel *p*.

Nachdem man mit Gleichung (4.8) zur rechten Seite des DRBs vorgedrungen ist und für alle Punkte ein $l(p,d)$ berechnet vorliegt, erhält man den Wert einer optimalen Lösung durch Selektion der minimalen Kosten in der letzten Spalte:

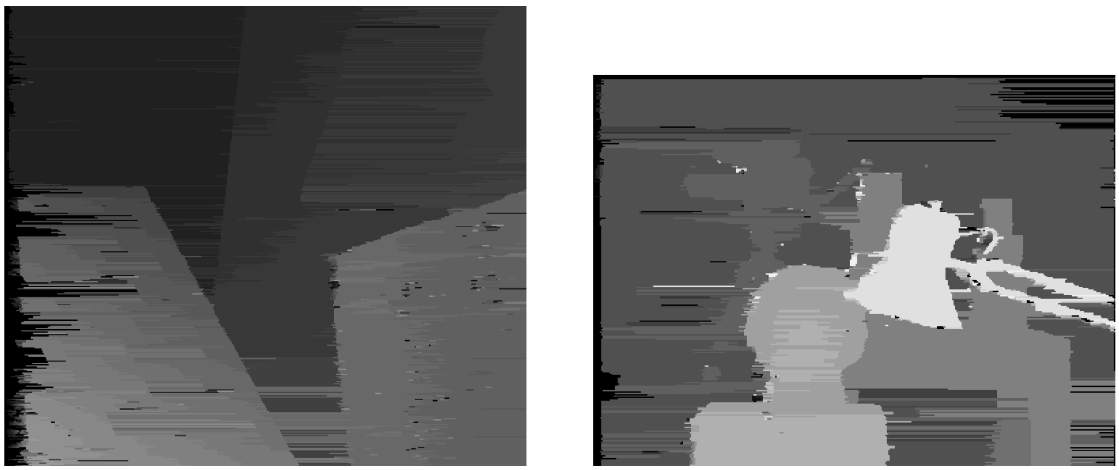
$$C_{opt} = \min_{d \in D}\left(l(p_{m-1}, d)\right) \tag{4.9}$$

$m-1$ bezeichnet den Index für die letzte Spalte im DRB.

Die optimale Disparitätszuweisung für die letzte Spalte erhält man danach mittels Ersetzen von *min* durch *argmin* in Gleichung (4.9):

$$d_{opt} = \arg\min_{d \in D} \left( l(p_{m-1}, d) \right) \tag{4.10}$$

Schließlich, als letzten Schritt, gilt es, mithilfe des minimalen Wertes den minimalen Pfad, der eine optimale Lösung des Problems repräsentiert, und die dazugehörigen Disparitäten abzuleiten. Das geschieht durch schrittweises Rückwärtsgehen mithilfe von gespeicherten Rückwärtszeigern. Beginnend an der Stelle des gefundenen optimalen Wertes folgt man den Zeigern so lange, bis man wieder an der linken Seite angekommen ist. Abbildung 12 zeigt die erzielten Ergebnisse für die Stereobildpaare Venus und Tsukuba. In Abbildung 13 werden die einzelnen Schritte der Methoden an einem simplifizierten Beispiel grafisch aufbereitet.



a) b)

Abbildung 12: Ergebnisse der Bildzeilenoptimierung mit dynamischer Programmierung für die Stereobildpaare a) Venus und b) Tsukuba. Weil die Zeilen unabhängig voneinander optimiert werden, besteht keine Konsistenz über unterschiedliche Bildzeilen hinweg. Dieser Umstand führt zu der deutlich sichtbaren Streifenbildung in den berechneten Disparitätskarten. Quelle: [51].

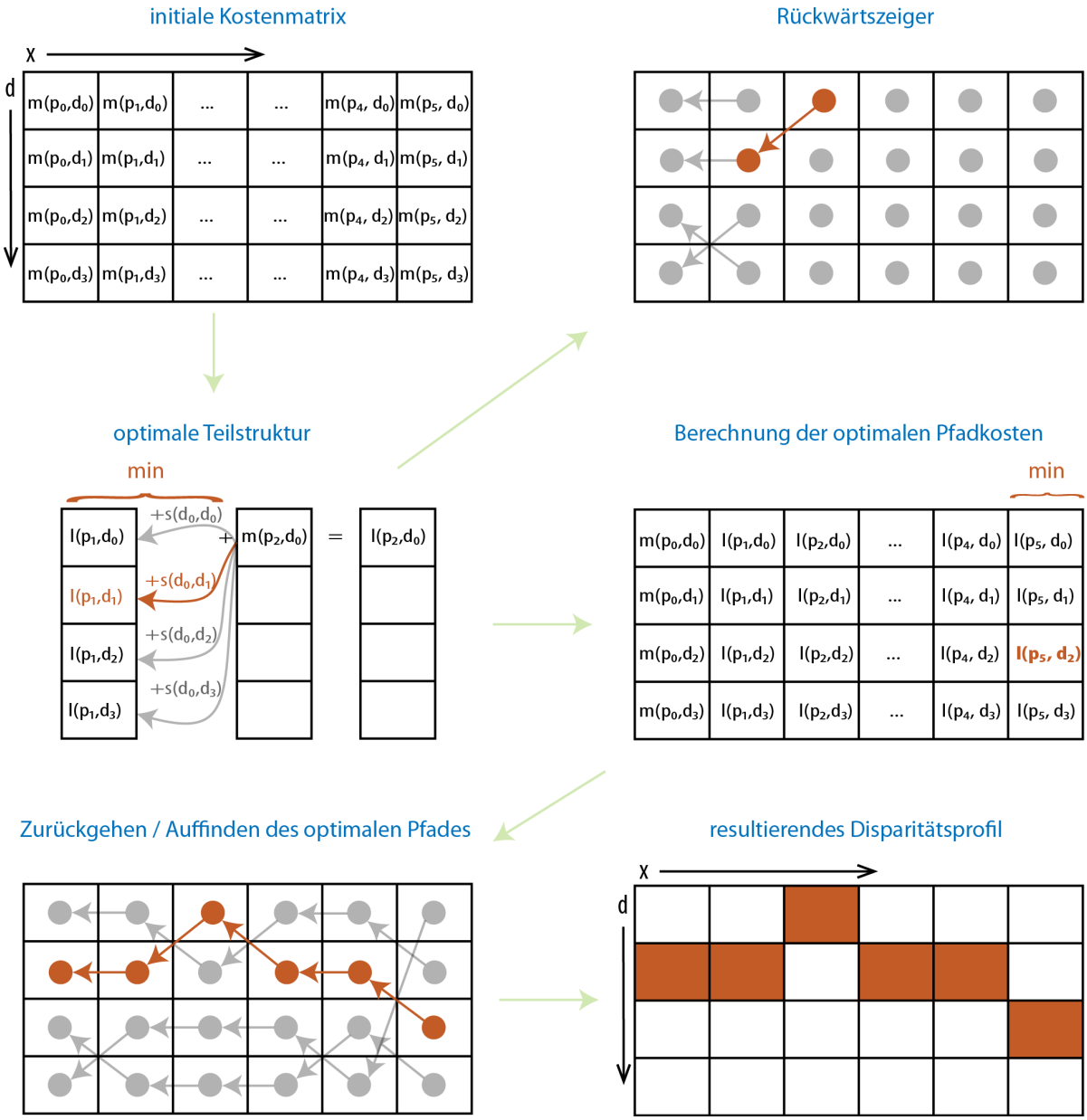



Abbildung 13: Die einzelnen Schritte zur Berechnung von Disparitäten durch Energieminimierung via dynamische Programmierung entlang von Bildzeilen. Das sukzessive optimale Lösen von Teilproblemen führt zu einer globalen optimalen Lösung.

Die hier beschriebene Methode verwendet die Eindeutigkeitsbedingung nicht aber die Reihenfolgebedingung. Somit entfällt der Parameter für die Verdeckungskosten und ohne die Glattheitsbedingung wäre das Verfahren eine simple Gewinner-nimmt-alles-Optimierung. Außerdem ist die Methode asymmetrisch. Erstmals wurde diese Form der Bildzeilenoptimierung in [41] vorgestellt. Sei $\delta$ die Anzahl der Menge der möglichen Disparitäten $D$, $m$ die Breite der Stereobilder und $n$ deren Höhe, dann beläuft sich die Komplexität des Verfahrens für eine

Bildzeile auf $O(m \cdot \delta^2)$ und für das ganze Bildpaar auf $O(m \cdot n \cdot \delta^2)$. Eine naive Evaluierung aller möglichen Kombinationen von $m$ Bildpunkten und $\delta$ Disparitätszuweisungen hätte eine exponentielle Laufzeit von $O(\delta^m)$ pro Zeile. Folglich verringert dynamische Programmierung den rechnerischen Aufwand für das Auffinden des optimalen Pfades beträchtlich. Zusätzlich zu der methodischen Verringerung der Laufzeit hat man verschiedene Techniken entwickelt, um die Laufzeit weiter zu senken.

#### Techniken zur Reduktion der Laufzeit

Spezielle Glattheitsbedingungen vermindern die Komplexität des Verfahrens noch weiter. Bei Einsatz des Potts-Modells reicht es, höchstens zwei mögliche Vorgänger zu betrachten [52]. Den Vorgänger mit gleicher Disparität, so dass keine Diskontinuitätskosten entstehen und den Vorgänger, wenn er sich bezüglich dem ersten unterscheidet, mit den geringsten akkumulierten Pfadkosten. Natürlich setzt das voraus, dass man sich immer den besten Vorgänger merkt. Die Laufzeit beträgt so $O(m \cdot n \cdot \delta)$.

Hierarchische Techniken versprechen eine zusätzliche Reduktion der Laufzeit. Meerbergen et al. stellen in [92] eine hierarchische Implementierung vor, deren Laufzeit unabhängig von der in den Bildern auftretenden Disparität ist und das ohne qualitative Beeinträchtigungen. Sie zeigen, dass die zweifache Ausführung des Algorithmus mit kleiner Disparität schneller ist als eine einzige komplette Suche über die volle Disparität, vor allem, wenn die Auflösung beim ersten Durchgang reduziert ist. Zuerst berechnen sie die Disparitäten auf dem Stereobildpaar mit geringerer Auflösung. Die Resultate benutzen sie für die Korrespondenzsuche in den Bildern mit hoher Auflösung. Dieser Vorgang lässt sich öfters wiederholen. Bei einer bestmöglichen Anzahl an Verfeinerungsschritten beträgt die Komplexität des Verfahrens $O(m \cdot n)$ für ein Stereobildpaar. Eine derartige Grob-zu-Fein-Strategie kommt auch in [94] und [95] zum Einsatz.

Etwas anders verfahren Forstmann et al. in [93]. Sie berechnen zuerst die Disparitäten für jede n-te Zeile und limitieren mit den Resultaten die Suche in den dazwischenliegenden Bildzeilen. Birchfield und Tomasi [96] verweigern, um Rechenzeit zu sparen, die Fortführung von schlechten Pfaden während der Suche, ohne dabei die Optimalität der Lösung zu opfern. Die Laufzeit ihres Algorithmus beträgt $O(m \cdot \delta \cdot \log(\delta))$ pro Zeile.

Eine Verminderung des benötigten rechnerischen Aufwandes kann sich ferner aus dem Einsatz der Reihenfolgebedingung ergeben. Die Bedingung erlaubt die Reduktion der Anzahl potentieller Übergange von einem Punkt entlang des Pfades zum nächsten. In [61] benutzen Bobick

und Intille die Verdeckungsbedingung, welche die Reihenfolgebedingung inkludiert. Dadurch erreichen sie eine Laufzeit von $O(m \cdot \delta)$ pro Zeile. Jeder Punkt im DRB befindet sich hierbei in einem von drei Zuständen: Übereinstimmung (Ü), vertikale Verdeckung (V) oder diagonale Verdeckung (D). Dementsprechend gibt es nur drei Richtungen, die ein Pfad einschlagen kann. Entweder er bewegt sich horizontal und wechselt in den Übereinstimmungszustand oder er wechselt nach oben in eine vertikale Verdeckung oder er zieht nach rechts unten in eine diagonale Verdeckung.

Diente das linke Stereobild als Referenzbild bei der Erstellung des DRVs, bedeutet eine vertikale Verdeckung, dass sich im rechten Bild halb verdeckte Pixel befinden, für die keine Korrespondenzen im linken Bild vorliegen. Bei einer diagonalen Verdeckung verhält es sich umgekehrt. Abbildung 14 zeigt einen Ausschnitt aus dem DRB von Abbildung 11 d), der den optimalen Pfad unter Gebrauch der Verdeckungsbedingung nach [61] darstellt. Die tatsächlichen Disparitäten sind rot und dunkelrot markiert, wobei dunkelrote Bereiche verdeckte Pixel repräsentieren. Grüne Pixel entsprechen einer vertikalen Verdeckung und blaue Pixel einer diagonalen Verdeckung.

#### Verdeckungskosten

Werden Verdeckungen wie in [61] explizit in die Pfadfindung eingebunden, taucht die Frage auf, welche Kosten die Punkte im verdeckten Zustand zugewiesen bekommen sollen. Sie müssen ungleich null sein, da sonst der optimale Pfad jener Pfad ist, der keine Übereinstimmungen findet, sondern stets zwischen vertikalen und diagonalen Sprüngen wechselt. Die Kosten dürfen auch nicht zu hoch angesetzt werden, um Disparitätssprünge nicht zu unterbinden und dadurch falsche Korrespondenzen zu favorisieren, weil sie günstiger sind als Disparitätssprünge.

Um die Sensibilität gegenüber den Verdeckungskosten zu mildern, bedienen sich Bobick und Intille [61] spezieller Punkte im DRB, sogenannter Kontrollpunkte, welche mit hoher Wahrscheinlichkeit eine Übereinstimmung verkörpern. Sie zwingen alle Pfade, durch diese Punkte zu gehen. Diese Strategie nötigt die Pfade Verdeckungssprünge zu vollziehen, welche unter Umständen durch zu hohe Verdeckungskosten ausgelassen worden wären. Die Kontrollpunkte bestimmen die Autoren heuristisch. Als zweite Maßnahme zur Desensibilisierung führen sie eine Kantenerkennung auf dem DRB aus. Die Information der gewonnenen Kanten beziehen sie in die Pfadfindung mit ein, indem sie Verdeckungskosten entlang der Kanten senken. Man

nimmt dabei an, dass sich jeder Tiefensprung in einer Szene als Intensitätsdiskontinuität in den Stereobildern manifestiert und daher Disparitätssprünge bei Intensitätsdiskontinuitäten auftreten.

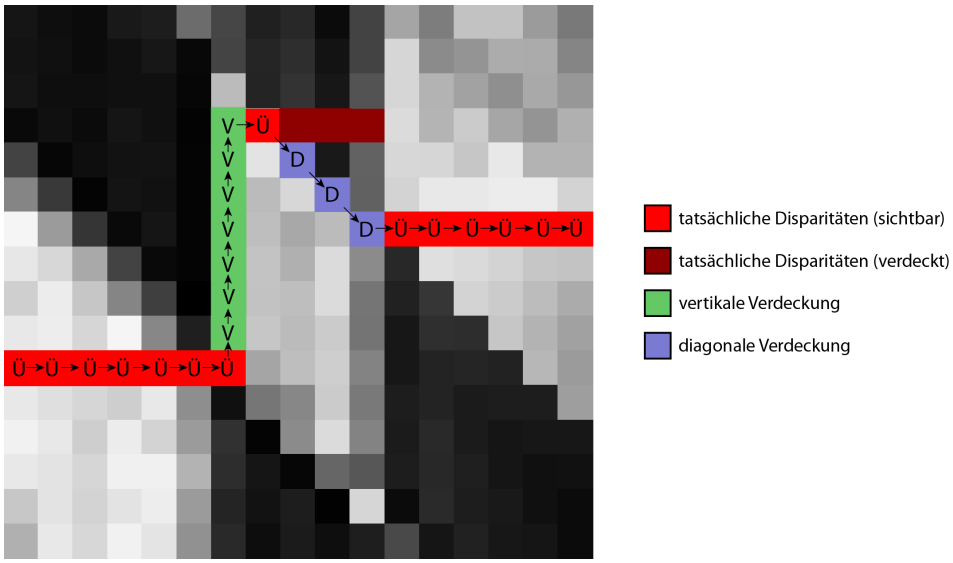


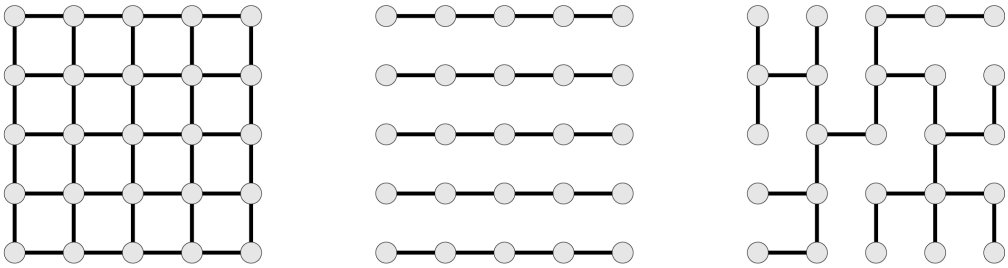

Abbildung 14: Forcierung der Reihenfolgebedingung nach [61]. Zu sehen ist ein Ausschnitt des DRBs aus Abbildung 11 d). Wählt man die Verdeckungskosten für vertikale und diagonale Sprünge zu hoch, findet man mit dynamischer Programmierung nicht den korrekten Pfad, weil ein Pfad mit falschen Korrespondenzen womöglich billiger ist. In der Abbildung würde der Pfad dann nicht zum einzelnen übereinstimmenden Pixel nach oben springen, sondern einen direkteren Weg zwischen den beiden Übereinstimmungssequenzen links und rechts ausmachen.

Kim et al. [97] verwenden ähnliche Schritte zur Desensibilisierung bezüglich der Verdeckungskosten. Sie erweitern allerdings das Konzept der Kontrollpunkte. Statt wie in [61] pro Spalte des DRBs nur einen sicheren Kontrollpunkt zuzulassen, eruieren sie die wahrscheinlichsten Disparitätskandidaten und nehmen diese als generalisierte Kontrollpunkte. Pro Spalte im DRB existieren nun mehrere Durchgänge für Pfade. Diese Praktik erhöht die Robustheit des Verfahrens dadurch, dass sie die Wahrscheinlichkeit eines falschen Kontrollpunktes verringert. Kanteninformationen binden die Autoren über eine Gewichtungsfunktion ein, deren Wert invers proportional zum Intensitätsgradienten ist. Bei einem hohen Intensitätsgradienten in der Nähe von Kanten werden die Kosten für eine Verdeckung gemindert.

### Streifenbildung

Wie Abbildung 12 demonstriert, zeigen Disparitätskarten, welche mit dynamischer Programmierung entlang einer Bildzeile erstellt wurden, eine starke horizontale Streifenbildung. Kontinuierlich verlaufende vertikale Kanten finden sich selten. Diese Situation resultiert aus der Optimierung der Energiefunktion in Gleichung (4.4) nur unter Berücksichtigung horizontaler Nachbarn pro Bildzeile. Für eine global optimale Lösung aller Disparitäten ist der limitierte Kontext einer einzigen Bildzeile suboptimal [89]. Es existieren zumeist mehrere optimale Lösungen pro Zeile, von denen man mit dynamischer Programmierung zwar eine davon findet, aber nicht notwendigerweise jeweils die zur Vorgängerlösung korrekt korrespondierende [84]. Bildrauschen kann die Auswahl einer Lösung ebenfalls ungünstig beeinflussen, wenn mehrere Pfade existieren, die nahe am Minimum liegen.

Eine Visualisierung der horizontalen Nachbarschaft wird in Abbildung 15 b) gezeigt. Aus rechentechnischer Hinsicht verfügt das unabhängige Abarbeiten der Bildzeilen über eine gewisse Attraktivität, weil es sich leicht formalisieren und parallelisieren lässt. Sinnvoller wäre es allerdings, die eindimensionale Kostenfunktion zu einer zweidimensionalen Kostenfunktion zu erweitern. Das entspräche einer Vierer-Nachbarschaft wie in Abbildung 15 a). Allerdings ist die 2-D-Optimierung der Gleichung (4.4) unter Berücksichtigung einer Vierer-Nachbarschaft ein NP-hartes Problem für die meisten Klassen von Glattheitsfunktionen [81]. Bevor heute übliche Verfahren zur effizienten Approximation einer Lösung der 2-D Optimierung wie Graph-cuts [83] oder Belief Propagation [82] aufgekommen sind, bediente man sich unterschiedlicher Modi Procedendi, um eine Konsistenz zwischen horizontalen Bildzeilen zu erreichen. Einige dieser Verfahrensweisen sind auch heute noch wegen ihrer Möglichkeit zur schnellen und effizienten Implementierung relevant.



a) Vierer-Nachbarschaft b) Zweier-Nachbarschaft (horizontal) c) Baumstruktur

Abbildung 15: Die Wahl des Nachbarschaftssystems für Gleichung (4.4) wirkt sich auf die Komplexität der Optimierung der Energiefunktion aus. a) Ein Vierer-Nachbarschaftssystem wäre wünschenswert, bringt jedoch ebenfalls den meisten Aufwand mit sich. Ein Zweier-Nachbarschaftssystem wie in b) kann mit dynamischer Programmierung effizient gelöst werden, birgt allerdings den Nachteil der fehlenden Konsistenz zwischen den Lösungen der einzelnen Zeilen. Diesen Nachteil versucht man in c) durch die Wahl einer Baumstruktur als Nachbarschaftssystem zu entkräften, ohne die Komplexität dabei zu erhöhen. Quelle: [9].

#### Ansätze zur Herstellung vertikaler Konsistenz

Baker und Binford [88] benutzen nach der Korrespondenzsuche von Kanten pro Bildzeile einen kooperativen Algorithmus zur Identifizierung und Entfernung von Zuordnungen, welche die Kontinuität von Oberflächen über Zeilen hinweg unterbrechen. Sie wollen erreichen, dass eine verbundene Sequenz von Kanten in einem Bild mit einer verbundenen Sequenz von Kanten im anderen Bild korreliert.

Ohta und Kanade [90] definieren einen 3-D Suchraum, in welchem sie schließlich die geringsten Kostenpfade zwischen 3-D Knoten ausmachen. Der 3-D Suchraum ist eine Aufschichtung jener 2-D-Ebenen, die zur Auffindung der optimalen Disparitäten je Bildzeile eingesetzt worden sind. Kanten in den Stereobildern verlaufen in diesem Raum dreidimensional durch mehrere Ebenen und treffen dabei auf mehrere 2-D-Knoten. Eine solche Aufreihung von 2-D Knoten entlang von Kanten formen einen 3-D-Knoten. Die optimalen Pfade zwischen zwei 3-D-Knoten finden die Autoren wieder mit dynamischer Programmierung. Diese zusätzliche Optimierung bewirkt eine gewisse Konsistenz zwischen den Zeilen.

Belhumeur und Mumford nutzen in [98] und [7] die gefundenen Lösungen für die Bildzeilen als Anfangszustand für die schrittweise Minimierung einer 2-D Version der eindimensionalen

Energiefunktion. In [98] bewerkstelligen sie das via simulierte Abkühlung (engl. *simulated annealing*). In [7] setzt Belhumeur auf eine heuristische Strategie, welche er selbst als iterative stochastische dynamische Programmierung bezeichnet

Cox et al. [84] modifizieren die Pfadfindung so, dass im Falle mehrerer globaler Optima der Pfad mit minimalen horizontalen Diskontinuitäten eruiert wird. Dies verringert zwar die Fehlerquote, stellt aber noch keine vollständige vertikale Konsistenz über Zeilen hinweg sicher. Hierfür vergleichen sie in einem zweiten Durchgang die Disparitäten einer Bildzeile mit den Disparitäten der Zeilen darüber und darunter, um vertikale Diskontinuitäten zwischen den Zeilen zu lokalisieren und zu minimieren. Die Autoren präferieren die Lösung mit der geringsten Anzahl an horizontalen und vertikalen Diskontinuitäten, weil sie damit keine Glattheitsbedingung forcieren. Sie argumentieren weiters, dass die Lösung mit minimalen Diskontinuitäten die glatteste Lösung darstellt und daher auch vertikale Konsistenz mit sich bringt.

Für vertikale Konsistenz setzen Birchfield und Tomasi in [96] auf einen Zuverlässigkeitsansatz. In einem Nachbearbeitungsschritt weisen sie jedem Pixel einen Zuverlässigkeitswert zu und propagieren längs vertikaler Spalten die Disparität der zuverlässigsten Pixel in Regionen mit unverlässlichen Disparitätswerten. Der Zuverlässigkeitswert eines Pixels ergibt sich aus der Summe benachbarter Pixel mit übereinstimmender Disparität. Ferner präsentieren Gong und Yang in [52] einen Zuverlässigkeitsansatz. Die Zuverlässigkeit einer vorgeschlagenen Zuordnung definieren sie als den Kostenunterschied zwischen der global besten Disparitätsbelegung, welche die vorgeschlagene Zuordnung enthält, und der global besten Disparitätsbelegung, welche die Zuordnung nicht enthält. In einem iterativen Prozess übernehmen die Verfasser zuerst nur die vertrauenswürdigsten Korrespondenzen und erst in weiteren Schritten, mit veränderten Parametern der zu optimierenden globalen Funktion, weitere Korrespondenzen. Neue Zuordnung gliedert man nur dann ein, wenn sie nicht aufgrund von Konsistenzbedingungen mit den bereits gefundenen Zuordnungen in Konflikt stehen. Eine solche Lösung entfernt viele der horizontalen Streifen, erzeugt aber keine dichte Disparitätskarte bzw. erst nach mehreren Iterationen. Forstmann et al. [93] verwirklichen eine Prozedur, die Farbunterschiede vertikal über Zeilen hinweg glättet.

Obwohl viele Verfahren mit dynamischer Programmierung auf einer Kostenmatrix arbeiten, die durch Vergleiche auf Pixelebene erhoben wurde, e.g. die absolute Intensitätsdifferenz zwischen zwei Pixeln, bietet sich dennoch die Möglichkeit vertikale Konsistenz schon über die Kostenmatrix implizit in den Algorithmus einzubinden. Das geschieht durch Verwendung einer Fenstermethode zur Kostenaggregation und Erstellung des DRVs. Danach optimiert man

die Disparitäten normal mit dynamischer Programmierung pro Bildzeile. Diese Taktik setzen unter anderen [99] und [100] ein. Dong et al. [99] benutzen einen auf Segmentierung basierten variablen Fensteransatz zur Aggregation. In [100] approximieren Wang et al. den wirkungsvollen, wenn auch teuren adaptiven Fensteransatz von [75]. Sie schaffen es allerdings, ihr System mit programmierbarer Graphikhardware auf Bildraten, geeignet für Echtzeitsysteme, zu beschleunigen. Durch den Aufwand der Kostenaggregation verringert sich die Streifenbildung deutlich.

#### Kombination mehrerer Durchläufe dynamischer Programmierung

Eine naheliegende Lösung zur Herstellung vertikaler Konsistenz über Zeilen hinweg ersinnt Sun in [95]. Er optimiert mit dynamischer Programmierung in einem zweistufigen Prozess sowohl entlang horizontaler Bildzeilen als auch entlang vertikaler Bildspalten und kombiniert die Ergebnisse. Der erste Durchgang berechnet die Pfadkosten für jeden Punkt im DRB längs der senkrechten Spalten, ohne jedoch am Ende einen Pfad zu ermitteln. Mit den aktualisierten Kosten aus dem ersten Durchgang kalkuliert Sun schließlich den optimalen Pfad entlang der waagrechten Bildzeilen. Die Reihenfolge der Durchgänge spielt grundsätzlich keine Rolle, wenn der Algorithmus eine entsprechende Anpassung erfährt. Dieses Prinzip der Kombination mehrerer Durchgänge wird aufgrund seiner Effizienz in vielen Methoden mit dynamischer Programmierung eingesetzt.

Desselben Verfahrens bedienen sich auch Kim et al. [97]. Beim ersten Durchgang berechnen sie hingegen die Pfadkosten für jedes Element im DRB nicht nur von einer Richtung kommend, sondern von zwei Richtungen kommend. Gong und Yang verbessern in [101] ihr Verfahren von [52] mit dieser Zweipassstrategie und beschleunigen ihren Algorithmus mithilfe von programmierbarer Graphikhardware. Ebenfalls dem Verfahren von [97] folgend, aktualisiert Mozerov in [102] die Kosten für jede Stelle im DRB von beiden Richtungen kommend. Er beschränkt sich allerdings nicht auf zwei Durchläufe für vertikale und horizontale Konsistenz. Um die Ergebnisse weiter zu verbessern, führt er zusätzlich zwei diagonale Durchläufe aus. Die finalen Disparitäten bestimmt er nicht wie üblich durch Rückwärtsgehen entlang des besten Pfades nach dem letzten Durchlauf. Stattdessen eruiert eine simple Gewinner-nimmt-alles-Strategie am Ende die finalen Disparitäten im aktualisierten DRV. Prinzipiell schlägt er in seinem Algorithmus vor, solange in verschiedene Richtungen zu optimieren, bis sich die resultierende Disparitätskarte nicht mehr weiter verfeinern lässt.

Diesen Gedanken, aus verschiedenen Richtungen kommend zu optimieren, nutzt auch Hirschmüller in seinem semiglobalen Ansatz [103]. Er schlägt vor, die Kosten für jedes Pixel von mindestens 8 Richtungen kommend zu aggregieren. Besser wären allerdings 16 Richtungen, welche von den Bildrändern sternförmig auf das Pixel von Interesse zulaufen.

Eine Kombination mehrerer Methoden zum Erreichen vertikaler Konsistenz benutzen Salmen et al. in [104]. Zum einen aggregieren sie die Kosten vertikal über benachbarte Bildzeilen zur Erstellung der Kostenmatrix für die Optimierung mit dynamischer Programmierung, zum anderen optimieren sie sowohl horizontal als auch vertikal und gebrauchen dabei ähnlich wie in [97] eine Art generalisierter Kontrollpunkte, mit denen sie die Komplexität reduzieren. Nur gewinnen sie die Kontrollpunkte nicht über lokale Korrespondenzsuche, sondern aus dem ersten Durchgang der dynamischen Programmierung, indem sie, wie in [52], mehrere optimale oder fast optimale Pfade eruieren. Durch verschiedene Pfade können sich mehrere Disparitätskandidaten pro Pixel ergeben, welche als generalisierte Kontrollpunkte für den vertikalen Durchlauf dienen. Die Autoren setzen mehr Parameter als üblich in ihrem Verfahren ein, um eine flexiblere Gestaltung zu ermöglichen. Die Schätzung der Parameter realisieren sie maschinell mit einem evolutionären Algorithmus.

Eine weitere Lösung für das Streifenproblem und für die Optimierung der Energiefunktion erarbeitet Veksler in [105]. Es geht dabei um die Optimierung eines Graphen, der als Knoten die Pixel des Bildes enthält und mit seinen Kanten Nachbarschaftsbeziehungen simuliert. Generell gilt, dass die Optimierung eines Graphen, der alle Kanten zwischen den Pixeln enthält, wie in Abbildung 15 a), mit polynomiellen Algorithmen nicht bewerkstelligt werden kann. Restringiert man hingegen die möglichen Graphen auf Bäume, wie Veksler (siehe Abb. 15 c)), befindet man sich in der Lage, den Graphen mit zugehöriger Energiefunktion effizient via dynamische Programmierung zu minimieren.

## 4.4 Dynamische Programmierung auf einem Baum

Eine Baumstruktur liefert eine merklich bessere Approximation des 2-D Gitters als eine eindimensionale Zeilenstruktur. Ihre Optimierung lässt sich ohne signifikante Geschwindigkeitseinbußen gegenüber der Optimierung pro Zeile bewerkstelligen. Veksler [105] macht insgesamt drei Vorteile aus, die für eine Baumstruktur sprechen.

- Der Baum beinhaltet $(m \cdot n) - 1$ Kanten des 2-D Gitters, eine verbundene Zeile hingegen immer nur $m - 1$ ( $m$ bezeichnet die Bildbreite, $n$ die Bildhöhe).

- Zumal jeder Knoten mit jedem anderen Knoten im Baum in Verbindung steht, hängt die Schätzung der Disparität eines Bildpunktes von der Schätzung aller anderen Bildpunkte ab. Somit bewirkt dynamische Programmierung auf einen Baum eine echte globale Optimierung.
- Laut Cayleys Formel für die Anzahl von Bäumen lassen sich mit n Knoten $n^{(n-2)}$ verschiedene Bäume aufspannen. Aus dieser großen Auswahl kann der Baum mit den wichtigsten Kanten des Gitters ausgesucht werden. Eine mögliche Wahl für die Kanten wäre zum Beispiel, die Kanten mit geringsten Intensitätsunterschieden zwischen den Pixeln, welche sie verbinden, zu selektieren. Die Pixel solcher Kanten besitzen vermutlich dieselbe Disparität.

Die Implementierung des Verfahrens stellt eine einfache Generalisierung der dynamischen Programmierung entlang von Zeilen dar und wird im Folgenden näher erläutert. Im Anschluss daran wird die Erstellung von geeigneten Bäumen diskutiert.

#### Verfahrensablauf

Gegeben sei der Baum $G(V, E)$, mit Knoten $V$ und Kanten $E$. $r \in V$ ist Wurzelknoten des Baumes. Die Wahl von $r$ darf beliebig getroffen werden. Aus der Baumstruktur resultiert, dass jeder Knoten $v$ außer $r$ über genau einen Elternknoten $p(v)$ verfügt. Die minimale Energie für einen Knoten $v$ wird wieder rekursiv angegeben, bestehend aus dem minimalen Unterbaum wurzelnd in $v$ und der Kante zwischen $v$ und $p(v)$. Sie wird als Funktion der Disparität des Elternknotens $d_{p(v)}$ festgelegt:

$$E_v(d_{p(v)}) = \min_{d_v \in D}\left( m(d_v) + s(d_v, d_{p(v)}) + \sum_{w \in C_v} E_w(d_v) \right) \tag{4.11}$$

wobei $C_v$ die Kinder von $v$ beschreibt.

Der analogen Gleichung für den Wurzelknoten fehlt lediglich der Term $s(d_v, d_{p(v)})$ und die Funktion wird als Funktion von $d_v$ definiert. Wie schon bei der dynamischen Programmierung entlang einer Bildzeile, bekommt man die optimale Disparitätszuweisung mittels Ersetzen von $min$ durch $arg\,min$ in Gleichung (4.11). Die Formel für die optimale Zuweisung der Wurzel lautet demnach:

$$L_r = \arg\min_{d_r \in D} \left( m(d_r) + \sum_{w \in C_v} E_w(d_r) \right) \tag{4.12}$$

Beginnend bei der Wurzel arbeitet sich die Rekursion bis zu den Blattknoten vor. Weil Blattknoten keine Kinderknoten haben und daher $C_v$ nur die leere Menge enthält, können $E_v$ und $L_v$ direkt evaluiert werden. Als nächsten Schritt berechnet man $E_v$ und $L_v$ für alle Elternknoten der Blattknoten und arbeitet sich derart wieder zu der Wurzel durch. Dort wird mittels Gleichung (4.12) die optimale Disparitätszuweisung für den Wurzelknoten berechnet. Mit dieser optimalen Korrespondenz für die Wurzel und den bereits berechneten Werten $L_v$ für die anderen Knoten, eruiert man durch schrittweises Hinabsinken zu den Blattknoten die optimalen Korrespondenzen für jeden Knoten im Baum respektive für jedes Pixel im Bild. Abbildung 16 verdeutlicht diese Vorgehensweise.

Sei δ die Anzahl der Menge der möglichen Disparitäten *D*, dann beträgt die Laufzeit des Verfahrens $O(n \cdot m \cdot \delta^2)$, wie schon bei der Bildzeilenoptimierung. Und wie bei der Bildzeilenoptimierung lässt sich die Laufzeit mithilfe spezieller Glattheitsbedingungen, e.g. dem Potts-Modell, auf $O(n \cdot m \cdot \delta)$ reduzieren.

#### Erstellung geeigneter Bäume

Für die Generierung von geeigneten Bäumen präsentiert Veksler [105] zwei Techniken. Beim Baum der minimalen Intensitätsdifferenzen (MID-Baum) werden alle Kanten des kompletten 2-D-Gitters mit den Intensitätsdifferenzen der Pixel gewichtet, welche sie verbinden. Anschließend konstruiert man den minimalen Spannbaum [86] und verwendet diesen als Eingabe für den Algorithmus. Dabei nimmt man implizit an, dass Tiefendiskontinuitäten mit Intensitätsdiskontinuitäten übereinstimmen und Pixel mit ähnlicher Intensität eine ähnliche Disparität aufweisen. In der Regel existieren mehrere minimal spannende Bäume. Damit die Auswahl eines Baumes nicht von der Implementierung für den minimalen Spannbaum abhängt und um den sinnvollsten unter ihnen auszuwählen, verwendet Veksler als zweite Technik eine Distanztransformation. Hiermit misst sie, wie tief ein Pixel *p* innerhalb einer homogenen Region liegt, und nutzt diese Information für die Wahl eines geeigneten Baumes. Zusätzlich setzt sie Gewichte ein, welche diese Information widerspiegeln. Eine Kante wird höher gewichtet, wenn sich die zwei Pixel, welche die Kante verbindet, nahe an Intensitätsdiskontinuitäten be-

finden, und niedriger gewichtet, wenn die Pixel innerhalb von homogenen Regionen liegen. Auf diese Weise lassen sich Ambiguitäten bei der Erstellung des minimal spannenden Baumes auflösen. Die Autorin bezeichnet den Baum mit Distanztransformation als MIDDT-Baum.

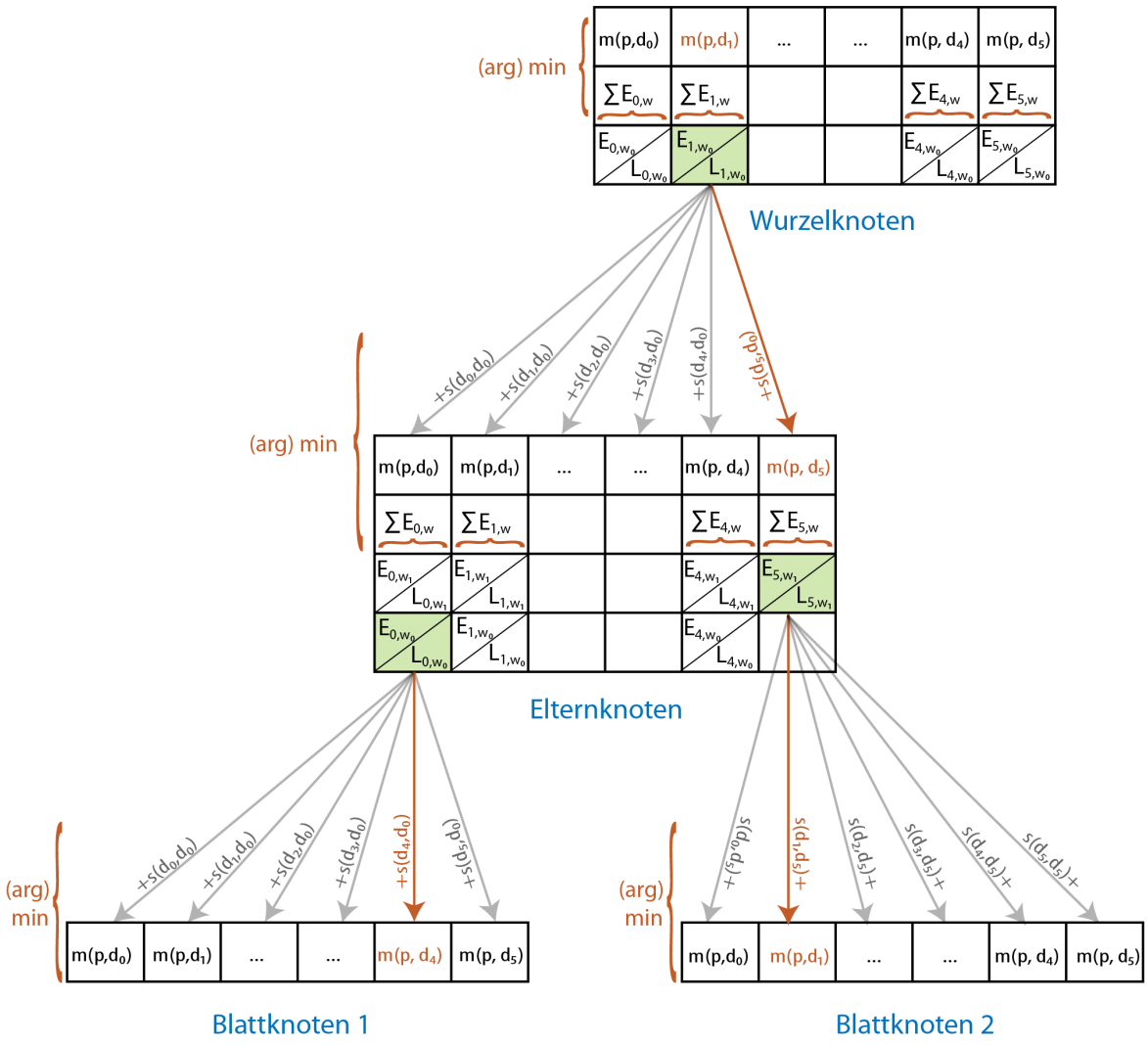
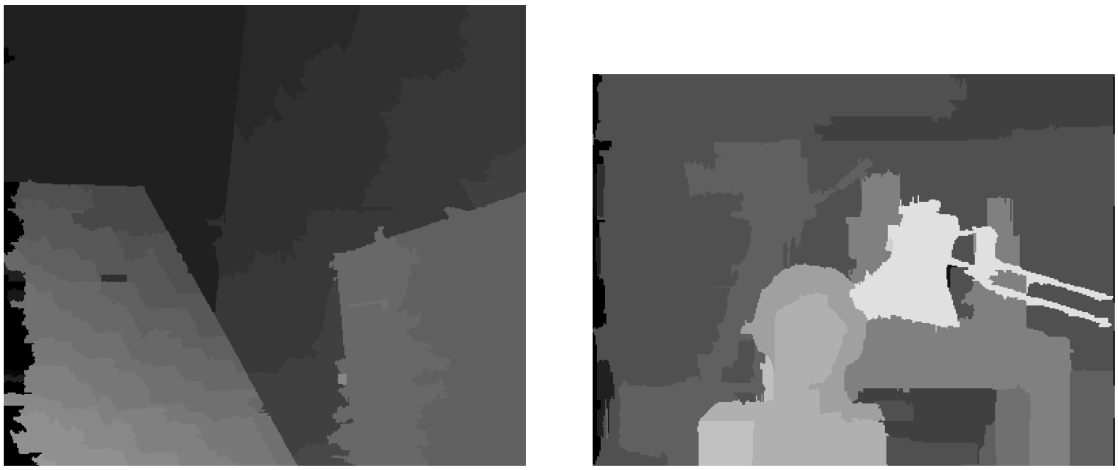



Abbildung 16: Die Bestimmung der optimalen Disparität für den Wurzelknoten mithilfe dynamischer Programmierung angewendet auf einen Baum. Die Vorgehensweise ähnelt der Vorgehensweise von dynamischer Programmierung entlang von Zeilen.

Abbildung 17 zeigt Tiefenbilder für die Stereobildpaare Tsukuba und Venus. Sie wurden mit dynamischer Programmierung angewendet auf einen Baum unter Einsatz des MIDDT-Baumes erstellt. Im Vergleich mit Abbildung 12 wird deutlich, dass das Problem der Streifenbildung behoben werden konnte und die Ergebnisse insgesamt besser ausfallen. Tabelle 1 stellt die

beiden Verfahren anhand der Evaluierung der Middlebury-Stereoseite [51] für die Bilder Tsukuba und Venus gegenüber. Die Tiefenbilder aus Abbildung 17 sind der ersten Version der Middlebury-Evaluierung entnommen, weil sie den Resultaten aus dem Artikel [105] entsprechen.



a) b)

Abbildung 17: Ergebnisse der Disparitätsoptimierung via dynamische Programmierung auf einem Baum, wie in [105] beschrieben, für das Stereobildpaar a) Venus und b) Tsukuba. Vergleicht man die Resultate mit den Resultaten dynamischer Programmierung entlang von Zeilen (Abb. 12), zeigt sich, dass keine Streifenbildung mehr auftritt und die Ergebnisse im Allgemeinen besser ausfallen. Quelle: [51].

| **Algorithmus** | **Tsukuba** | | | **Venus** | | | **Ø falscher Pixel** |
|---|---|---|---|---|---|---|---|
| | nicht verdeckt | alle | nahe Disk. | nicht verdeckt | alle | nahe Disk. | |
| **DP Baum** | 1.99 | 2.84 | 9.96 | 1.41 | 2.1 | 7.74 | 11.7 |
| **DP Zeile** | 5.08 | 7.22 | 12.2 | 9.44 | 10.9 | 21.9 | 16.6 |

Tabelle 1: Resultate der Evaluation der Middlebury-Stereoseite [51] für die Stereobilder Venus und Tsukuba der beiden Verfahren für dynamische Programmierung. Dynamische Programmierung auf einem Baum liefert die besseren Ergebnisse. Die Spalte ganz rechts zeigt den Prozentsatz falscher Pixel über alle Bilder der Evaluation von [51] (nicht nur der Bilder Venus und Tsukuba).

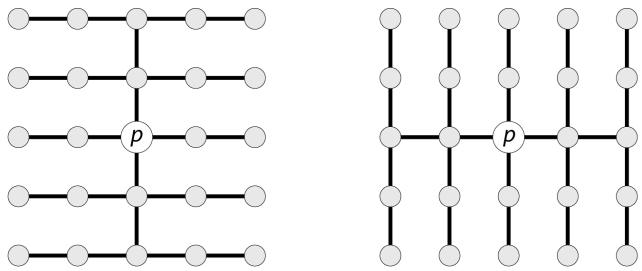
#### Erweiterungen der Methode

Deng und Lin [106] erhöhen die Genauigkeit der dynamischen Programmierung auf einem Baum. Statt einzelner Pixel setzen sie ganze Liniensegmente als Knoten für den Baum ein. Dazu führen sie zuerst eine Farbsegmentierung der Stereobilder durch, welche ähnliche Farbpixel pro Zeile in Liniensegmente zusammenfasst. Diese Segmente dienen anschließend als Knoten. Um schräge Ebenen in der Szene zu modellieren, verwenden sie einen 3-parametrischen Labelraum, der die möglichen Korrespondenzen zwischen Segmenten beschreibt. Zur Energieminimierung verfahren die Autoren, wie in [105] beschrieben. Lei et al. [107] benutzen gleichfalls die Idee, Segmente als Knoten für den Baum zu gebrauchen. Nur beschränken sie sich dabei nicht auf Liniensegmente, sondern nutzen ganze Regionen, die Pixel mehrerer Zeilen beinhalten können. Sin et al. [108] setzen zwar auch Segmentierung ein, verfahren mit den einzelnen Segmenten jedoch anders als in [106] und [107]. Sie konstruieren die Segmente mit Hilfe eines modifizierten Algorithmus von Kruskal [86], der ihnen einen minimal spannenden Wald liefert. Jeder Baum in diesem Wald repräsentiert ein Segment. Pixel in zu kleinen Segmenten bzw. Bäumen fügen die Autoren zu größeren Bäumen hinzu, anhand der geodätischen Distanz zwischen dem Pixel von Interesse und der benachbarten Segmente. Danach minimieren sie jeden Baum einzeln mit dynamischer Programmierung. Der Vorteil dieser Vorgehensweise liegt in ihrem Potential zur Parallelisierung, weil man die Disparitätsverteilung jedes Baumes parallel optimieren kann.

Eine Komposition aus dynamischer Programmierung entlang von Zeilen und dynamischer Programmierung auf einen Baum findet sich in [9]. Dieser Algorithmus wurde auch zur Implementierung auf einer Graphikkarte in dieser Arbeit ausgewählt. Er wird er im Folgenden genauer geschildert.

# 5 Dynamische Programmierung auf einer effizienten Baumstruktur

Die Autoren Bleyer und Gelautz setzen auf eine einfache und effizient zu implementierende Baumstruktur. Für jedes Pixel konstruieren sie zwei Bäume (Abb. 18). Ihre komplementäre Struktur soll den Nachteil kompensieren, dass bei einem Baum nicht alle Kanten des 2-D-Gitters Verwendung finden. Weil ein komplementärer Baum jeweils die Kanten beinhaltet, welche beim anderen Baum ausgelassen werden, approximiert die Kombination der Ergebnisse beider Bäume das 2-D-Gitter relativ gut. Im Gegensatz zu anderen Methoden mit dynamischer Programmierung auf einen Baum (e.g. [105] und [107]) bestimmt daher nicht nur ein einzelner Baum alle Disparitäten, sondern man verwendet pro Pixel zwei Bäume. Allerdings summieren sich zwei Bäume pro Pixel zu Myriaden von Bäumen für das komplette Referenzbild. Um die Energieminimierung aller Bäume in realistischer Zeit zu bewerkstelligen, nutzen die Autoren die spezielle Struktur der Bäume aus. Sie erlaubt es ihnen, die Optima aller Bäume eines Typs in vier Durchläufen mit zeilenbasierter dynamischer Programmierung zu berechnen.




a) Horizontaler Baum b) Vertikaler Baum

Abbildung 18: Komplementäre Bäume von [9]. Quelle [9].

Das Verfahren ist ein echtes globales Verfahren, weil jeder der eingesetzten Bäume alle Bildpunkte im Bild nutzt. Es werden somit keine, für eine Zuordnung womöglich entscheidende Regionen ausgelassen.

## 5.1 Energiefunktion

Die eingesetzte Energiefunktion besteht, wie für globale Verfahren typisch, aus einem Datenterm und einem Glattheitsterm:

$$E(\Delta) = \sum_{p \in I_{ref}} m(p, d_p) + \sum_{(p,q) \in N} s_p(d_p, d_q) \tag{5.1}$$

wobei $I_{ref}$ die Menge aller Bildpunkte des Referenzbildes repräsentiert. $\Delta$ beschreibt eine Lösung (Disparitätskarte), welche jedem Pixel $p$ eine Disparität $d_p \in D$ zuweist. $D$ bezeichnet die Menge der möglichen Disparitäten. Als Datenterm $E_d$ (linker Term auf der rechten Seite von Gleichung (5.1)) verwenden die Autoren die Metrik von Birchfield und Tomasi [47], welche unempfindlich gegenüber der Bildabtastung ist, auf den RGB-Werten der Bilder. Als Glattheitsannahme kommt ein erweitertes Potts-Modell zum Einsatz (siehe auch Gleichung (4.6)):

$$s_p(d_p, d_q) = \begin{cases} 0 & wenn \;\; d_p = d_q \\ P_1 & wenn \;\; |d_p - d_q| = 1 \\ P_2 & andernfalls \end{cases} \tag{5.2}$$

Die Aufgabe der Gewichte $P_1$ und $P_2$ ist das Bestrafen von Disparitätssprüngen. Kleine Sprünge im Bereich von einer Disparitätsstufe tauchen für gewöhnlich bei schräg verlaufenden Oberflächen auf und sollten demnach nicht stark bestraft werden. Deswegen ist $P_1$ kleiner als $P_2$. Bei größeren Disparitätssprüngen kommt das Gewicht $P_2$ zum Tragen, bei dem wiederum zwei Fälle unterschieden werden:

$$P_2 = \begin{cases} P_3 \cdot \acute{P}_2 & wenn \;\; |I_p - I_q| < T \\ \acute{P}_2 & andernfalls \end{cases} \tag{5.3}$$

mit $|I_p - I_q|$ als Summe der absoluten Differenzen der RGB-Werte zwischen dem Bildpunkt $p$ und dem Bildpunkt $q$. Die Autoren möchten damit erreichen, dass Disparitätssprünge mit den Diskontinuitäten in den Farbwerten übereinstimmen. Bei Farbdiskontinuitäten weist $|I_p - I_q|$ einen hohen Wert auf und liegt über dem Schwellwert $T$. Ein Disparitätssprung wird folglich nur mit $\acute{P}_2$ bestraft. Für ihre Implementierung setzen Bleyer und Gelautz die Parameter auf $P_1 = 20$, $\acute{P}_2 = 30$, $P_3 = 4$ und $T = 30$.

## 5.2 Berechnung der horizontalen Bäume

Obwohl eigentlich die Energie von Bäumen minimiert werden soll, berechnen die Autoren die Optima der Minimierung mit dynamischer Programmierung entlang von Zeilen. Pro Bildpunkt werden zwei Typen von Bäumen optimiert, ein horizontaler Baum und ein vertikaler Baum. Im Folgenden wird die Erstellung einer Matrix $H$ erläutert, welche die optimalen Werte der horizontalen Bäume enthält. Die Berechnung einer Matrix V für die Optima der vertikalen Bäume erfolgt äquivalent.

Als ersten Schritt optimieren Bleyer und Gelautz die horizontalen Bildzeilen. Dazu führen sie einen Vorwärts- und einen Rückwärtsdurchlauf von dynamischer Programmierung aus und kombinieren die Ergebnisse. Ziel ist es, die Zuordnungskosten $C[p,d]$ für jedes Pixel $p$ und jede Disparität $d$ zu eruieren, wobei die Matrix $C[p,d]$ die optimalen Kosten vom linken und rechten Bildrand kommend repräsentiert. Die Vorgehensweise ist in Abbildung 19 a)-c) zu sehen.

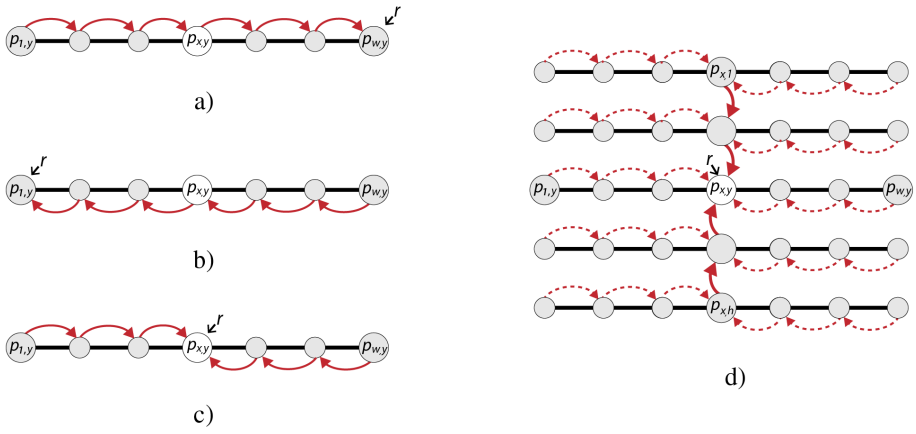



Abbildung 19: a) Vorwärtsdurchlauf mit dynamischer Programmierung in horizontaler Richtung b) Rückwärtsdurchlauf c) Durch die Kombination von Vorwärts- und Rückwärtsdurchlauf lassen sich jene Kosten ermitteln, welche die optimalen Kosten von linker und rechter Seite kommend darstellen. d) zeigt die Berechnung der minimalen Kosten eines horizontalen Baumes, wurzelnd im Pixel $p_{x,y}$. Dazu optimiert man in vertikaler Richtung unter Verwendung der Werte aus der horizontalen Optimierung.

Vorwärts- und Rückwärtsdurchlauf mit dynamischer Programmierung werden prinzipiell mit Gleichung (4.8) berechnet. Die hier verwendete spezielle Glattheitsbedingung erlaubt allerdings eine Reformulierung dieser Gleichung:

$$\dot{l}(p,d) = m(p,d) + \min\left(\dot{l}(q,d), \dot{l}(q,d-1)+P_1, \dot{l}(q,d+1)+P_1, \min_{i\in D} \dot{l}(q,i)+P_2\right) \tag{5.4}$$

mit $D$ als Menge der möglichen Disparitäten und $q$ als unmittelbarer Vorgänger von $p$. Im Falle des Vorwärtsdurchlaufes besteht der Vorgänger $q$ aus dem linken Nachbarn von $p$, beim Rückwärtsdurchlauf ist $q$ der rechte Nachbar.

Der Einsatz von Gleichung (5.4) hat den Vorteil, dass dadurch die Komplexität der Pfadfindung bei der dynamischen Programmierung reduziert wird. Die Laufzeit verringert sich von $O(m\cdot n\cdot \delta^2)$ auf $O(m\cdot n\cdot \delta)$, weil der Wert von $min_{i\in D}\, l'(q,i)$ für alle Disparitäten eines Pixels konstant ist und vorberechnet werden kann. Die ermittelten Kosten aus dem Vorwärtsdurchlauf speichert man in eine Matrix $F$ und die Kosten des Rückwärtsdurchlaufes in eine Matrix $B$. Beide Matrizen werden bei der Berechnung der Matrix C benötigt. Die Gleichung zur Berechnung der Matrix C resultiert aus der Gleichung zur Berechnung der minimalen Kosten mit dynamischer Programmierung angewendet auf eine Baumstruktur. Diese definieren Bleyer und Gelautz folgendermaßen:

$$l(p,d) = m(p,d) + \sum_{q\in v(p)} \min_{i\in D}\left(s(d,i)+l(q,i)\right) \tag{5.5}$$

wobei $v(p)$ die Kinderknoten von $p$ bezeichnet.

Die Definition von Gleichung (5.5) ist äquivalent zu der Definition in Gleichung (4.11). Bei Gleichung (5.5) lässt sich allerdings leichter erkennen, dass dynamische Programmierung entlang von Zeilen eigentlich nur einen Spezialfall von dynamischer Programmierung angewendet auf einen Baum darstellt[6].

Die Formel zur Kalkulation der Matrix $C$ lautet:

6 Befindet sich der Wurzelknoten an einer der beiden Zeilenenden (Abb. 19 a) und b)), hat jeder Knoten, inklusive des Wurzelknotens, immer nur einen Kinderknoten. Somit entfällt die Summe in Gleichung (5.5) und sie wird zur Gleichung (4.8).

$$C(p_{x,y},d) = m(p_{x,y},d) + \min_{i \in D}\left(s(d,i) + F[p_{x-1,y},i]\right) + \min_{i \in D}\left(s(d,i) + B[p_{x+1,y},i]\right) \tag{5.6}$$

Dies ist gleichbedeutend mit:

$$C(p_{x,y},d) = F[p_{x,y},d] + B[p_{x,y},d] - m(p_{x,y},d) \tag{5.7}$$

Nach der horizontalen Optimierung nutzt man dieselbe Strategie wieder, um in vertikaler Richtung zu optimieren. Das entspricht der Optimierung entlang der vertikalen Kanten der Bäume. Damit die bereits eruierten Ergebnisse der horizontalen Optimierung mitgenommen werden, gebraucht man bei den vertikalen Durchläufen mit dynamischer Programmierung die Werte, welche in der Matrix *C* gespeichert sind, als Datenkosten:

$$\ddot{l}(p_{x,y},d) = C[p_{x,y},d] + \sum_{q \in \dot{v}(p_{x,y})} \min_{i \in D}\left(s(d,i) + \ddot{l}(q,i)\right) \tag{5.8}$$

wobei hier die Funktion $\dot{v}(p_{x,y})$ nun die Kinderknoten von *p* zurückgibt, welche über eine vertikale Kante mit *p* verbunden sind. Abbildung 19 d) visualisiert die Berechnung. Gleichung (5.8) lässt sich wieder in dieselbe Form wie Gleichung (5.4) bringen.

Nach vertikalem Vorwärts- und Rückwärtsdurchlauf mit den Kosten der Matrix *C* als Datenkosten kombinieren die Autoren die Durchläufe wieder, wie in Gleichung (5.7) angegeben. Auch hier werden die ursprünglichen Datenkosten durch die Werte von *C* ersetzt. Daraus resultiert schließlich die Matrix *H*, welche die minimalen Kosten des horizontalen Baumes für jedes Pixel beinhaltet. Die Berechnung der Kosten für die vertikalen Bäume erfolgt äquivalent wie die Berechnung der Matrix *H*.

## 5.3 Kombination der komplementären Bäume

Bleyer und Gelautz bestimmen in ihrer Implementierung zuerst die Kosten der vertikalen Bäume und speichern die Werte in der Matrix *V*. Danach berechnen sie die Matrix *H*. Damit die in der Matrix *V* gefundenen Disparitäten und eruierten Kosten ihre Tendenzen in die Matrix *H* propagieren, aktualisieren sie die ursprünglichen Datenkosten, bevor sie die Matrix *H* ermitteln, auf folgende Weise:

$$\dot{m}(p,d) = m(p,d) + \lambda \cdot (V(p,d) - \min_{i \in D} V(p,i)) \tag{5.9}$$

wobei der Parameter $\lambda$ den Einfluss von $V$ reguliert.

Für jedes Pixel $p$ und einer bestimmten Disparität $d$ nimmt man zuerst einmal die ursprünglichen Kosten, welche die Disparität $d$ verursachen würde. Dazu addiert man die Differenz der Kosten einer optimalen Disparitätszuweisung, welche $d$ beinhaltet, und der Kosten der besten Lösung für dieses Pixel. Die Differenz zeigt an, wie nahe $d$ an der optimalen Lösung liegt. Bei einer hohen Differenz dürfte $d$ eine inkorrekte Disparitätszuweisung sein und daher sollten die Kosten für die Zuweisung von $d$ zu $p$ höher ausfallen. Im umgekehrten Fall könnte $d$ eine korrekte Zuweisung repräsentieren und die Kosten für die Zuweisung sollten demnach nicht merklich erhöht werden. So wird erst im weiteren Verlauf bei der Berechnung der horizontalen Bäume entschieden, welche Disparitätszuweisung schließlich gewinnt. Bei ihrer Implementierung setzen die Autoren $\lambda$ auf 0.025.

Die finalen Disparitäten bestimmen die Autoren schließlich nicht via Rückwärtsgehen beim letzten Durchgang dynamischer Programmierung, sondern durch eine Gewinner-nimmt-alles-Strategie auf den zuletzt aktualisierten Kosten:

$$d_p = \arg\min_{d \in D} H[p,d] \tag{5.10}$$

## 5.4 Verdeckungsbehandlung

Zum Verbessern der Ergebnisse wird eine explizite Verdeckungsbehandlung durchgeführt. Dazu eruieren die Verfasser zuerst die Disparitätskarte $\Delta_R$ für das rechte Bild, indem sie das rechte Bild als Referenzbild nehmen. Anschließend wird unter Verwendung von $\Delta_R$ die Geometrie des rechten Bildes auf die Geometrie des linken Bildes abgebildet. Entfällt auf ein Pixel im linken Bild kein Pixel des rechten Bildes, markiert man dieses Pixel als verdeckt. Als Resultat erhalten Bleyer und Gelautz ein Verdeckungsbild $O_L$ für das linke Bild, welches für jedes Pixel anführt, ob es verdeckt ist oder nicht. Bei geneigten Oberflächen kommt es aufgrund der perspektivischen Verzerrung öfters zu schmalen, ein Pixel breiten Verdeckungsregionen in $O_L$. Diese eliminiert man in einem Nachbearbeitungsschritt, weil die Pixel in diesen Regionen nicht wirklich verdeckt sind. Sie verletzen nur die Eindeutigkeitsannahme.

Von der Information des Verdeckungsbildes machen die Autoren bei der Berechnung der Disparitätskarte $\Delta_L$ für das linke Bild Gebrauch. Sie setzen dabei die Glattheitsbedingung für Bildpunkte, bei denen einer der beiden Nachbarn verdeckt ist, auf null. Dadurch verhindern sie, dass verdeckte Bildpunkte die Disparitätszuweisung ihrer Nachbarn durch mögliche Verdeckungskosten beeinflussen. Nach der Generierung der Disparitätskarte $\Delta_L$ mithilfe des Verdeckungsbildes, überschreiben die Autoren als letzten Schritt die Disparitäten verdeckter Regionen. Dort kommt es wegen der Verdeckung zu keinen sinnvollen Disparitätszuweisungen. Hierfür bestimmen sie für jeden verdeckten Bildpunkt die ersten nicht verdeckten Nachbarn in linker und rechter Richtung. Die kleinere Disparität der beiden Nachbarn wird schließlich als Disparität für den verdeckten Bildpunkt eingesetzt.

## 5.5 Erzielte Ergebnisse

Abbildung 20 dokumentiert die berechneten Disparitätskarten zusammen mit den tatsächlichen Disparitäten und den linken Bildern der Stereobildpaare Tsukuba, Venus, Teddy und Cones des Middlebury-Datenbestandes. Wie zu sehen ist, generiert das Verfahren Disparitätskarten mit einem glatten Verlauf der Disparitäten innerhalb von Objekten. Zudem gelingt, es die Disparitätsdiskontinuitäten an den Rändern von Objekten korrekt wiederzugeben. Das verdankt man der Verdeckungsbehandlung und dem Verzicht von lokaler Aggregation bei der Erstellung des DRVs. Weiters bleiben kleine Details erhalten, wie die Stäbchen im Vordergrund auf der rechten Seite des Stereobildpaares Cones. Ein Resultat, welches aufgrund der Glattheitsbedingung nicht selbstverständlich ist. Ferner ist auch so gut wie keine Streifenbildung beobachtbar, wie es eigentlich für dynamische Programmierung entlang von Zeilen typisch wäre (siehe Abb. 12). Dieser Umstand kann der Berechnung der komplementären Struktur der Bäume zugeschrieben werden. Die Berechnung der Bäume für ein Bild (ohne Verdeckungsbehandlung) mündet in acht Durchläufe dynamischer Programmierung, vier horizontale Durchläufe und vier vertikale Durchläufe. Die Kombination der Durchläufe stellt horizontale und vertikale Konsistenz sicher.

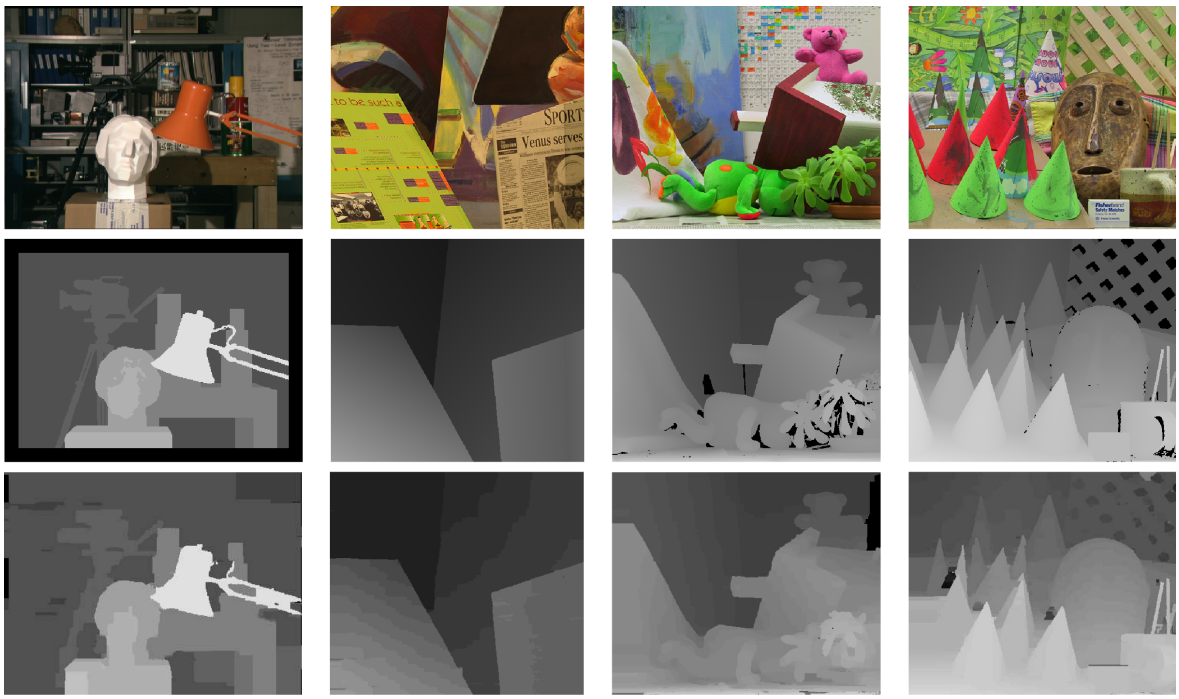



Abbildung 20: Die linken Bilder (oberste Reihe), die tatsächlichen Disparitäten (mittlere Reihe) und die nach [9] berechneten Disparitätskarten (unterste Reihe) der Stereobildpaare Tsukuba, Venus, Teddy und Cones des Middlebury-Datenbestandes. Die berechneten Disparitätskarten weisen einen glatten Verlauf innerhalb von Objekten auf und auch die Ränder von Objekten werden korrekt wiedergegeben. Trotz Glattheitsbedingung bleiben kleinere Details erhalten. Quellen: [9][51].

Um besser zu verstehen, wie effizient die dynamischen Durchgänge in dem beschriebenen Verfahren miteinander kombiniert werden, soll das Verfahren mit der Methode von Hirschmüller [103] verglichen werden. Hirschmüller setzt auf die gleiche Anzahl von Durchgängen dynamischer Programmierung, jedoch sind Richtung und Kombination der Durchgänge verschieden. Abbildung 21 verdeutlicht den Unterschied zwischen dem Verfahren von Bleyer und Gelautz und der Methode von Hirschmüller [103]. Die Disparitätskarten nach [9] enthalten kaum isolierte Pixel. Diese entstehen bei der Methode von [103], wenn die Pfade bei den unterschiedlichen Durchgängen dynamischer Programmierung nicht genug Textur beinhalten. Um die isolierten Pixel zu identifizieren und zu korrigieren, benötigt Hirschmüller einen zusätzlichen Nachbearbeitungsschritt, bei dem er die Disparitätskarte segmentiert und einzelne Segmente untersucht. Im Gegensatz dazu beinhaltet jeder Baum im Algorithmus von Bleyer und Gelautz alle Pixel. Es werden daher keine für die Disparitätsbestimmung unter Umständen wichtige Texturen oder andere Details ausgelassen [109].

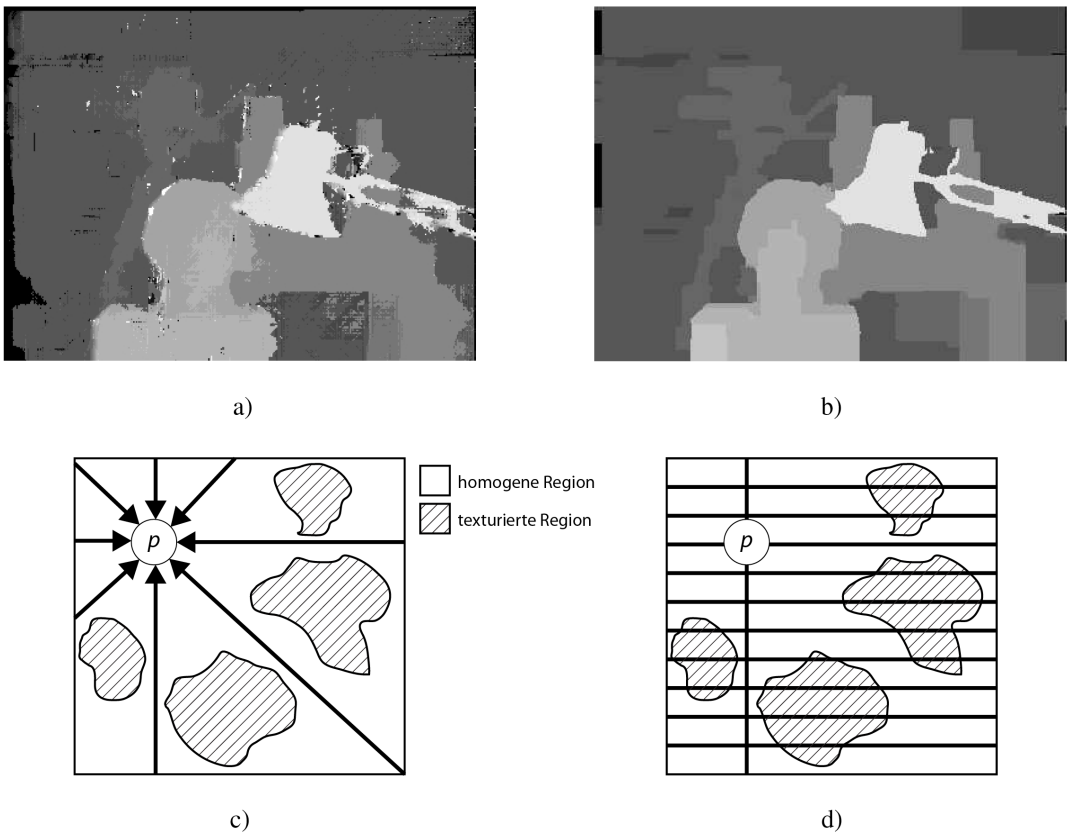



Abbildung 21: a) Disparitätskarte des Stereobildpaares Tsukuba berechnet mit einer Reimplementierung von [103] b) Disparitätskarte nach [9]. In a) tauchen vermehrt isolierte Pixel auf, die entstehen, wenn die Aggregationspfade bei den Durchgängen dynamischer Programmierung nicht ausreichend Textur enthalten (c)). In b) tritt dieses Problem nicht auf, weil jeder Baum, der zur Aggregation eingesetzt wird, alle Pixel des Referenzbildes enthält (d)). Quellen: a), c) und d) [109] b)[9].

Die Resultate der Middlebury-Evaluierung von [9] sind in Tabelle 2 angeführt. Die meisten Algorithmen, welche ein besseres Ranking erzielen (siehe [51]), gebrauchen aufwendigere Verfahren wie Graph-Cuts oder Belief Propagation. Die längeren Laufzeiten dieser Verfahren sind jedoch nicht für Echtzeitverarbeitung geeignet. Außerdem nutzen viele besser gereihte Algorithmen Segmentierung. Diese ist nicht nur zeitintensiv, sondern führt bei manchen stark texturierten Bildern zu suboptimalen Resultaten.

#### Performanz

Die Laufzeiten der Implementierung von Bleyer und Gelautz reichen von 0.18 Sekunden für das Stereobildpaar Tsukuba mit einer Größe von 384x288 Pixeln und 16 Disparitätsstufen, bis

zu 0.57 Sekunden für die Stereobildpaare Teddy und Cones mit einer Größe von 450x375 Pixeln und 60 Disparitätsstufen. Als Rechenmaschine kam ein Zweikern-AMD-Opteron mit 2.4 GHz zum Einsatz. Die Autoren versuchten die Vorteile der Mehrkernarchitektur auszunützen und parallelisierten die Berechnung der Durchläufe dynamischer Programmierung so gut wie möglich (unter anderem mit SIMD-Instruktionen).

| **Tsukuba** | | **Venus** | | **Teddy** | | **Cones** | |
|---|---|---|---|---|---|---|---|
| nicht verdeckt | alle | nicht verdeckt | alle | nicht verdeckt | alle | nicht verdeckt | alle |
| 1.86 | 2.56 | 0.42 | 0.76 | 7.31 | 12.70 | 4.00 | 9.74 |

Tabelle 2: Die Ergebnisse der Middlebury-Evaluation.

# 6 Implementierung eines Echtzeitverfahrens auf einer GPU

## 6.1 Einsatz von Graphikhardware

Aufgrund der leicht zugänglichen, weitverbreiteten und relativ billigen Graphikprozessoren (*GPUs*) und der Möglichkeit, diese Prozessoren für generelle Berechnungen zu verwenden, avancierte der Einsatz paralleler Programmierung in den letzten Jahren mehr und mehr zu einem Massenphänomen. Eine andere Ursache dafür ist die Einführung immer mehr Kerne bei Hauptprozessoren (*CPUs*). Dort kann wegen fundamentaler Begrenzungen bei der Herstellung integrierter Schaltkreise ein Leistungszuwachs nicht mehr in den Maßen wie bisher durch Erhöhung der Schaltfrequenz erzielt werden. Deswegen richtet man das Augenmerk auf die Integration mehrerer Kerne auf einen Chip. Das Konzept der parallelen Programmierung löst allmählich das herrschende Paradigma der sequentiellen Programmierung ab. In diesem Zusammenhang wird mitunter ebenfalls von der Mehrkern-Revolution gesprochen [110].

Hersteller von Graphikprozessoren wie die Firmen NVIDIA und AMD bieten SDKs und Bibliotheken an, um die Auslagerung genereller Berechnungen auf die Graphikkarte zu erleichtern. Für die Nutzung von Graphikprozessoren zum Lösen allgemeiner Aufgaben hat sich der Begriff GPGPU (*General Purpose Computation on Graphics Processing Unit*) etabliert. NVIDIA taufte sein GPGPU-Konzept CUDA (*Compute Unified Device Architecture*). Es findet im wissenschaftlichen Bereich vermehrt Anklang und wird auch hier zur Implementierung von [9] eingesetzt.

### Unterschiede zwischen CPU und GPU

Das Programmieren für eine GPU unterscheidet sich trotz des Einsatzes höherer Programmiersprachen wie CUDA C teilweise stark von der Programmierung für eine oder mehrere CPUs. Prof. Dr. Kun Zhou bemerkte zu der Frage der Portierbarkeit von Quellcode für CPUs auf GPUs: „Neu schreiben ist eher zutreffend als portieren. Um den Vorteil der hohen Parallelität von Grafikchips auszunutzen, muss man oftmals komplett neue Algorithmen schreiben“ [111]. Zhou ist Leiter der Abteilung für Graphik und parallele Systeme an der chinesischen Zhejing-Universität. Seine Forschung umfasst unter anderem parallele Programmierung für GPUs. Weiters bemerken die Autoren von [111], dass der begrenzende Faktor für die Laufzeit

von Programmen auf einer Graphikkarte bisher für gewöhnlich „die Programmierfähigkeit der Entwickler ist und nicht die Rechenleistung“ der Karte selbst. Schnellen Code zu schreiben, impliziert in der Regel, dass der Programmierer über detailliertes Wissen bezüglich des Aufbaues und der Funktionsweise einer GPU verfügt. Bei der Programmierung für CPUs ist dies nicht in diesem Ausmaße relevant. Der Grund hierfür liegt in der unterschiedlichen Architektur der beiden Prozessortypen.

GPUs wie auch CPUs fassen ihre Rechenwerke in Gruppen zusammen, welche man als SIMD-Einheiten bezeichnet. SIMD steht für „*Single Instruction, Multiple Data*“ und bedeutet, dass jede Einheit dasselbe Programm gleichzeitig abarbeitet, aber jeweils auf andere Daten anwendet. Diese SIMD-Einheiten sind typisch für Graphikkarten, weil in der Computergraphik oft für viele Pixel gleich verfahren wird. GPUs besitzen aus diesem Grund mehr SIMD-Einheiten als CPUs und verfügen zumindest theoretisch über eine höhere Rechenkraft. Beide Prozessortypen verwenden mehrstufige Caches. Diese sind allerdings bei GPUs kleiner und meistens nur für lesende Zugriffe optimiert. Der Datenzugriff auf den Graphikkartenspeicher bietet daher nicht dieselbe Flexibilität. Ferner funktioniert der Transport von Daten zum Hauptspeicher des Rechners nur mit beträchtlichen Verzögerungen und sollte daher auf ein Minimum reduziert werden [112].

Generell gilt, dass CPUs flexibler sind, weil sie mehr Transistoren für Datencaching, vorausschauende Strategien und Datenflusssteuerung nützen. Bei GPUs werden mehr Transistoren für die Datenverarbeitung aufgewendet, weshalb sie mehr SIMD-Einheiten aufweisen. Das geht auf Kosten der Flexibilität und GPUs haben daher eine streng vorgegebene Datenflussrichtung. Schafft man aber die SIMD-Einheiten einer Graphikkarte korrekt mit Daten zu versorgen, erreicht man eine schnellere Abarbeitung als auf CPUs [113].

## 6.2 CUDA

Mit der Einführung der Programmiersprache CUDA C durch NVIDIA anno 2007 änderten sich viele Beschränkungen, welche zuvor bei der GPGPU-Programmierung auftraten. Man legte CUDA C von Beginn an darauf aus, mit ihr allgemeine Berechnungen auf der Graphikkarte auszuführen. Selbst die Hardware wurde etwas angepasst, um derartige Berechnungen zu erleichtern [114].

### Architektur

CUDA-fähige GPUs sind in Reihen von Stream-Multiprozessoren (SM) angeordnet. Abbildung 22 zeigt die Architektur des G80-Graphikprozessors als Beispiel für eine CUDA-fähige Graphikeinheit. Er gilt als erster Prozessor, welcher CUDA unterstützt. Wie man sieht, formen zwei Stream-Multiprozessoren einen Verarbeitungsblock. Die Zahl der SMs pro Block kann je nach Prozessorgeneration variieren. Jeder SM besteht aus mehreren Stream-Prozessoren (SP). Diese SP teilen sich Kontrolllogik und Caches. Jede GPU besitzt des Weiteren einen globalen Speicher (DRAM). Zugriffe auf diesen Speicher sollten möglichst koordiniert erfolgen, um die hohe Bandbreite des Speichers auszunutzen und um die höhere Latenz, die der Speicher aufweist, wieder wettzumachen [114].

GPUs unterstützen bis zu 1024 aktive Threads pro SM. Bei Karten wie der Geforce GTX 280 mit 30 SM führt das zu mehr als 30 000 möglichen aktiven Threads. Bei CPUs bewegt man sich hingegen momentan noch im Bereich von 16 bis 64 aktiven Threads. Diese beträchtlichen Ressourcen an aktiven Threads sind einer der Hauptunterschiede zwischen CPU und GPU. Und die Möglichkeiten der Parallelisierung, sprich die Anzahl der aktiven Threads, werden mit neuen Graphikprozessoren noch weiter ansteigen. Der hohe Grad an Parallelisierung hat natürlich Auswirkungen auf das Programmiermodell von CUDA [115].

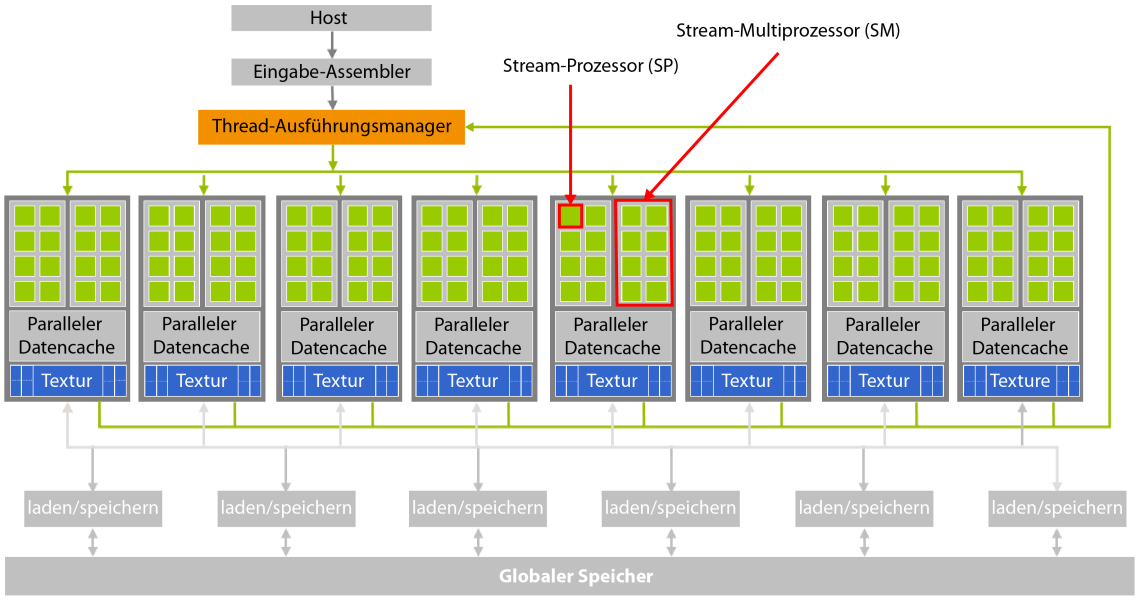
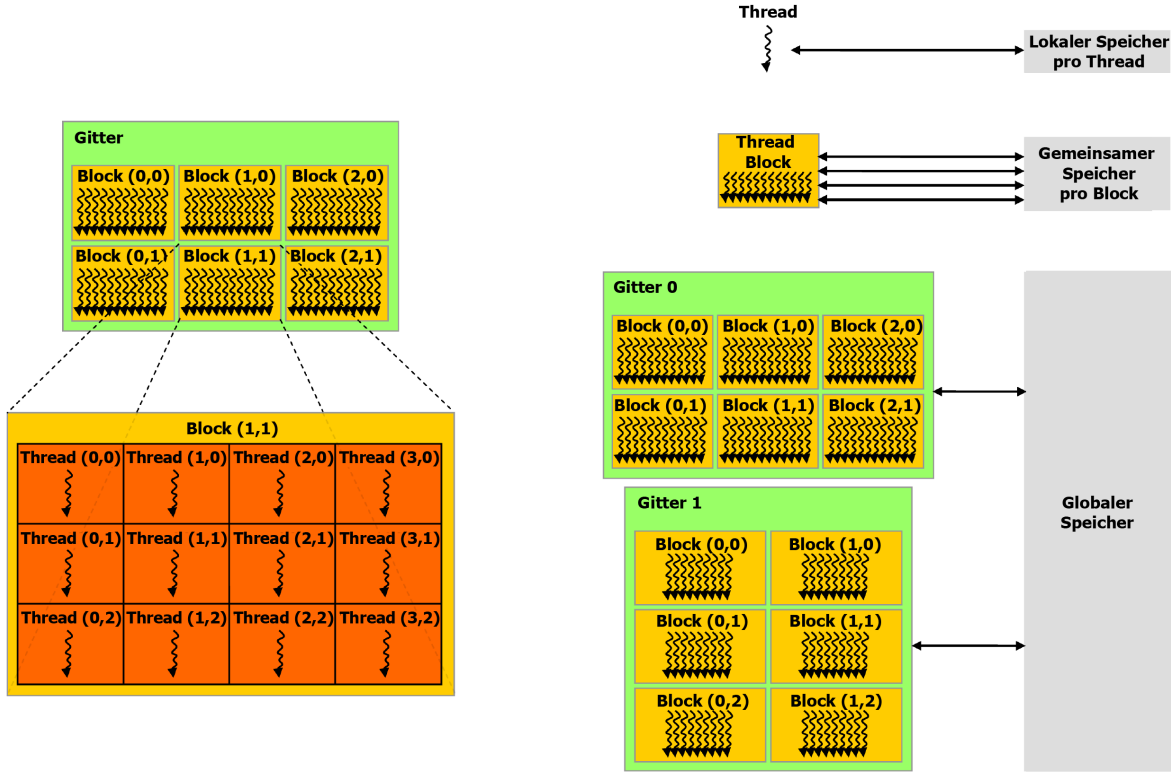



Abbildung 22: Aufbau des G80-Graphikprozessors als Beispiel für eine CUDA-fähige Architektur. Quelle: [116].

### Programmiermodell

Bei CUDA C wurde die Programmiersprache C um spezielle Funktionen, sogenannte Kernels (engl. für Kerne), erweitert. Diese lassen sich N-mal parallel auf N verschiedene CUDA-Threads ausführen [113]. Die Threads sind in Blöcke organisiert, die wiederum in einem Gitter angeordnet sind. Abbildung 23 a) veranschaulicht eine Organisation von zweidimensionalen Blöcken in einem zweidimensionalen Gitter. Sowohl die Blöcke als auch das Gitter können ein-, zwei- oder dreidimensional sein. Damit ermöglicht man je nach Aufgabenstellung eine intuitive Aufteilung der Threads.




a) Organisation der Threads

b) Speicherhierarchie

Abbildung 23: Zwei wichtige Komponenten des Programmiermodells von CUDA: a) die Organisation der Threads und b) die Speicherhierarchie. Quelle: [113].

Jeder Thread verfügt über einen privaten lokalen Speicher für private Variablen und Funktionsaufrufe. Der gemeinsame Speicher pro Block erfüllt die für parallele Berechnungen essentielle Aufgabe der Kommunikation zwischen Threads. Er speichert für parallele Algorithmen wichtige, gemeinsam genutzte Daten und Resultate. Weil sich dieser Speicher direkt auf dem

Chip des Prozessors befindet, weist er eine höhere Geschwindigkeit als der globale Speicher auf. Daher sollten benötigte Daten primär vom globalen Speicher in den gemeinsamen Speicher geladen werden und anschließend bei der Berechnung der gemeinsame Speicher genutzt werden. Zum Schluss werden die Ergebnisse wieder in den globalen Speicher geschrieben, dessen Lebensdauer über die ganze Applikation geht. Die Daten im lokalen Speicher und im gemeinsam genutzten Speicher gehen nach Terminierung eines Kernels verloren [113][115]. Abbildung 23 b) stellt diese beschriebene Speicherhierarchie dar.

Bei CUDA verfolgt man einen heterogenen Programmieransatz. Das bedeutet, CUDA C Programme involvieren das Ausführen von Code auf zwei verschiedenen Plattformen, einem Gastsystem mit einer oder mehreren CPUs und einem Hauptsystem mit einer oder mehreren GPUs. Etliche Problemstellungen beinhalten sowohl Aufgaben, welche nur sequentiell abgearbeitet werden können, als auch Aufgaben, welche sich zur massiven Parallelisierung eignen. Die sequentiellen Aufgaben führt man der heterogenen Idee folgend auf den CPUs aus, weil diese dafür optimiert sind. Die parallelen Kalkulationen verschiebt man auf die GPU. Damit versucht man, das Beste aus beiden Welten zu nutzen [115].

## 6.3 Verwandte Arbeiten

Zum Erreichen von Echtzeitverarbeitung in der Stereoanalyse wurden im Laufe der Zeit entweder besonders einfache Algorithmen eingesetzt, welche sich effizient implementieren lassen, oder es wurde auf spezielle Hardware zurückgegriffen wie ein FPGA. Einen guten Überblick von Echtzeitimplementierung ohne Einsatz von Graphikkarten liefert dabei [50]. Ein Vergleich von FPGA- und GPU-Implementierungen findet sich in [117].

Viele GPU Echtzeitimplementierungen nutzen einfache lokale Methoden [118]. Die Ergebnisse lokaler Methoden bleiben jedoch generell hinter den Ergebnissen globaler Methoden zurück. Die Komplexität globaler Methoden erschwert allerdings eine effiziente Implementierung, weshalb die Laufzeiten Größenordnungen hinter der lokaler Methoden liegen. Eine Ausnahme bildet hier dynamische Programmierung, weil das Verfahren relativ unkompliziert zu parallelisieren ist. Verwendet man beispielsweise dynamische Programmierung entlang von Zeilen, kann jede Zeile unabhängig von einem eigenen Thread abgearbeitet werden [119].

Gong und Yang [101] schaffen es beispielsweise, ihren Algorithmus mit zwei Durchgängen dynamischer Programmierung durch Einsatz von Graphikhardware auf Echtzeitabarbeitung zu beschleunigen. Ebenfalls zur parallelen Implementierung geeignet ist die Methode von [103], weil sie starken Gebrauch von dynamischer Programmierung macht. Rosenberg et al. [120]

realisieren diese Methode als Erste auf einer Graphikkarte. Als Metrik nutzen sie jedoch die absolute Differenz auf den RGB-Farbwerten. Gibson und Marques [121] implementieren die Methode unter Gebrauch von BT [47] als Ähnlichkeitsmetrik. Den kompletten Algorithmus von [103] auf einer Graphikkarte verwirklichen Ernst und Hirschmüller in [119]. Das Verfahren von [9] eignet sich ebenso zur Portierung auf eine Graphikkarte. Allerdings finden sich bis auf der hier vorgestellten Implementierung auf einer Graphikkarte noch keine anderen, dem Autor bekannten GPU-Implementierungen des Verfahrens.

Die Mehrheit der bisher veröffentlichten Implementierungen von Stereoanalysealgorithmen auf einer Graphikkarte gebrauchen hardwarenahe Shadersprachen in Kombination mit den Graphikschnittstellen OpenGL oder DirectX [122]. CUDA wird bisher nur in wenigen Veröffentlichungen verwendet. Das liegt daran, dass CUDA erst seit 2007 existiert. Seitdem steigt die Anzahl allerdings stetig (e.g. [123][121][124]).

Bei einem Vergleich von Laufzeiten zwischen CUDA C und Shadersprachen zeigt sich, dass Implementierungen mit Shadersprachen etwas effizienter laufen als jene mit CUDA. Das mag daran liegen, dass Shadersprachen den Programmierer eher dazu zwingen, die Daten korrekt an die Graphikkarte zu liefern und die Berechnungen möglichst effizient auszuführen, als dies bei CUDA C der Fall ist. In [119] erzielen Ernst und Hirschmüller mit Cg als Shadersprache und OpenGL eine Laufzeit von 8.8 Bildern pro Sekunde mit inkludiertem Konsistenztest und Mutual Information als Metrik auf Bildern der Größe 450x375 mit 64 Disparitätsstufen. Die CUDA-Implementierung der Autoren Gibson und Marques von [121] erreicht auf Bildern der selben Größe mit BT als Metrik und ohne Subpixelverfeinerung und Konsistenztest bei vergleichbarer Hardware nur eine Laufzeit von 5.9 Bilder pro Sekunde. Ähnlich die CUDA-Implementierung von [124]. Sie erreicht eine Bildrate von etwa 6-7 Hz[7] (bezogen auf Bilder der Größe 450x375 mit 64 Disparitätsstufen) unter Gebrauch der absoluten Differenz von Pixelintensitäten als Metrik. Allerdings setzen die Autoren eine schnellere Graphikkarte mit mehr Rechenkernen ein.

## 6.4 Details der Implementierung

Für die Implementierung von [9] wurde ausschließlich CUDA C eingesetzt. Auf den Einsatz spezieller Speicher der Graphikkarte, wie Texturspeicher, welche manche Leseoperation dank Caching beschleunigen können, wurde außer beim bilateralen Medianfilter verzichtet. Das

7 Die Laufzeiten wurden aus einem Diagramm geschätzt, deshalb die Ungenauigkeit.

kommt der Lesbarkeit des Quellcodes zugute. Außerdem ist es aufgrund des Algorithmus öfters notwendig, Daten zwischenzuspeichern, und der Texturspeicher kann nur zum Lesen von Daten genutzt werden. Die verwendete Graphikkarte ist eine Geforce GTX 460 mit 1 GB globalem Speicher, deren GF104 Graphikprozessor bereits die Fermi-Architektur besitzt. Der Testrechner besitzt einen Intel Core2 Quad Q6600 CPU und 2 GB Arbeitsspeicher.

Obwohl die Graphikkarte für Fließkommaoperationen optimiert ist, werden die Daten der Bilder und Zwischenergebnisse in 16 Bit *unsigned Short*-Arrays im globalen Speicher untergebracht. Dies brachte einen leichten Geschwindigkeitszuwachs von ein paar Millisekunden (abhängig von der Bildgröße) gegenüber der Speicherung in *Float*-Arrays. Außerdem benötigt man den damit gesparten Speicherplatz bei Stereobildern mit hoher Auflösung und einer größeren Menge an möglichen Disparitätsstufen. Die GPU-Implementierung schafft noch etwa eine Bildauflösung von 950x750 mit 96 Disparitätsstufen. Bei mehr Disparitätsstufen oder einer höheren Auflösung reicht jedoch der zur Verfügung stehende globale Speicher nicht mehr aus.

Bei Daten, welche für Berechnungen in den gemeinsamen Speicher geladen werden, zeigte sich, dass es generell besser ist, im gemeinsamen Speicher *float* zu nutzen. Das liegt daran, dass bei dem ausgeführten Zugriffsmuster auf den gemeinsamen Speicher mit *float* als Datentyp weniger Bankkonflikte entstehen [115]. Damit die Daten während der Aggregation mit dynamischer Programmierung nicht zu hohe Werte für 16 Bit große Speicher aufweisen, werden bei der Berechnung von Gleichung (5.4) die minimalen Pfadkosten des vorherigen Pixels abgezogen [103]. Diese Anpassung ändert nichts an den erzielten Ergebnissen, weil sich die Position des Minimums dadurch nicht verschiebt.

### 6.4.1 Erstellung des Disparitätsraumvolumens

In [9] wird die gegenüber Bildabtastung unempfindliche BT-Metrik [47] zur Berechnung der Zuordnungskosten herangezogen. Die Resultate von [125] demonstrieren allerdings, dass der Nutzen dieser Metrik beschränkt ist. Aus diesem Grund setzt die hier vorgestellte Implementierung auf die Ähnlichkeitsmetrik von [126], welche wie folgt lautet:

$$\begin{aligned} m(p,d) = (1-\alpha) \cdot min\left[\|I_{ziel}(p-d) - I_{ref}(p)\|,\ \tau_1\right] + \\ \alpha \cdot min\left[\|\nabla_x I_{ziel}(p-d) - \nabla_x I_{ref}(p)\|,\ \tau_2\right] \end{aligned} \tag{6.1}$$

$\nabla_x$ beschreibt den Gradienten in x-Richtung, $\tau_1$ und $\tau_2$ sind Begrenzungswerte und $\alpha$ regelt den Einfluss des Farb- respektive Gradiententerms. $\|I_{ziel}(p-d)-I_{ref}(p)\|$ bezeichnet die L1-Distanz zwischen zwei RGB-Farbvektoren.

Zur Generierung des DRVs wird pro Zeile ein eigener Block genützt, der so viele Threads beinhaltet, wie die Zeile Bildpunkte hat. Weil die Farbbilder drei Werte pro Bildpunkt aufweisen, muss eine ganze Farbzeile mit drei Durchgängen in den gemeinsamen Speicher geladen werden. Danach berechnet je ein Thread für ein Pixel im Referenzbild die L1-Distanzen zwischen zwei Farben iterativ für jede Disparitätsstufe. Das Ergebnis wird schließlich im DRV gespeichert. Der Gradient lässt sich vorberechnen und an der entsprechenden Stelle einlesen. Die benützte diskrete Approximation für den Gradienten in x-Richtung lautet [23]:

$$\nabla_x = \frac{dI}{dx} = I(x-1,y) - I(x+1,y) \tag{6.2}$$

Die Organisation des DRVs hat Auswirkungen auf die Laufzeiten der Durchgänge mit dynamischer Programmierung. Es werden für jeden Bildpunkt die Zuordnungskosten der einzelnen Disparitäten angrenzend in einer Reihenfolge gespeichert. Zuerst kommt der erste Bildpunkt mit seinen Zuordnungskosten. Daran schließt der benachbarte Bildpunkt mit seinen Kosten an. Abbildung 24 illustriert diese Organisation. Sie ermöglicht später ein möglichst effizientes Einlesen aller Kosten für einen Bildpunkt.

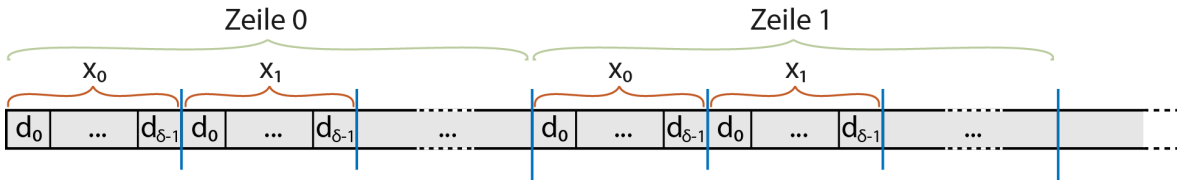



Abbildung 24: Organisation des DRVs. Die gewählte Anordnung erlaubt ein effizientes Lesen der Werte aus dem globalen Speicher zu einem späteren Zeitpunkt. Die Disparitätsstufen gehen von $d_0$ ... $d_{\delta-1}$ mit $\delta$ als Anzahl der möglichen Disparitätsstufen.

### 6.4.2 Durchgänge mit dynamischer Programmierung

Für die Erstellung der finalen Disparitätskarte benötigt man mit Verdeckungsbehandlung insgesamt sechzehn Durchläufe dynamischer Programmierung. Daher hängt die Laufzeit des ge-

samten Programmes stark davon ab, wie effizient ein Durchgang auf der Graphikkarte läuft. Die verschiedenen Durchgänge (vorwärts und rückwärts in horizontaler und vertikaler Richtung) unterscheiden sich kaum in ihrer Implementierung.

Bei den Durchgängen definiert δ, die Anzahl der Menge der möglichen Disparitäten, die Blockgröße pro Zeile (bzw. pro Spalte bei einem vertikalen Durchlauf). Die Gittergröße wird bei horizontalen Durchläufen durch die Bildhöhe bestimmt, bei vertikalen Durchläufen durch die Bildbreite. Wie die Organisation des DRVs und die Wahl von Block- und Gittergröße das Einlesen der Zuordnungskosten aus dem globalen Speicher erleichtert, verdeutlicht Abbildung 25 anhand eines vertikalen Durchganges. Die Speicherzugriffe erfolgen gut koordiniert, womit die hohe Bandbreite zum globalen Speicher gut ausgeschöpft werden kann. Bei horizontalen Durchgängen erfolgen die Speicherzugriffe nicht ganz so wirkungsvoll. Die Laufzeit erhöht sich dadurch aber nur kaum, weil innerhalb eines Blockes die Daten weiterhin effizient einlesbar sind.

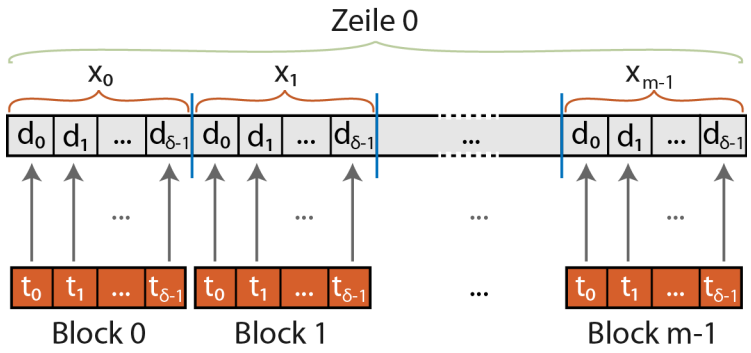



Abbildung 25: Bei den Durchläufen dynamischer Programmierung liest je ein Block gut koordiniert alle Disparitätskosten für ein Pixel. Dafür besitzt jeder Block so viel Threads, wie es Disparitätsstufen gibt. So schafft man es, die hohe Bandbreite zum globalen Speicher effizient zu nutzen.

Nach dem Einlesen der Daten vom globalen Speicher in den gemeinsamen Speicher wird zur Berechnung von Gleichung (5.4) nur mehr der gemeinsame Speicher eingesetzt. Das spart Zugriffe auf den weit entfernten globalen Speicher. Pro Disparitätsstufe in einer Zeile gibt es je einen Thread, welcher iterativ nach Gleichung (5.4) die Kosten berechnet. Somit lassen sich alle Zuweisungskosten für ein Pixel pro Zeile gleichzeitig berechnen. Zwischen den Iterationen werden alle Threads synchronisiert, um sicherzustellen, dass das Ergebnis jedes einzelnen

Threads in der nächsten Iteration zur Verfügung steht. Zudem wird nach der Synchronisation auch $min_{i \in D}\, l'(q,i)$ mittels paralleler Reduktion ermittelt. Das Schreiben der Resultate erfolgt nach demselben Schema wie das Laden der Kosten, um abermals eine hohe Bandbreite zu erzielen.

#### Parallele Reduktion

In der Berechnung von $min_{i \in D}\, l'(q,i)$ aus Gleichung (5.4) liegt ein beträchtliches Potential zur Laufzeitreduzierung. NVIDIA hat für diese Art von Problemstellung eigens ein Beispiel in ihrem SDK (Version 4.0) integriert mit Whitepaper [127]. Bei einer naiven Implementierung würde ein einzelner Thread Schritt für Schritt alle Werte durchgehen und das Minimum bestimmen mit einer Laufzeit von $O(\delta)$. Alle übrigen eingesetzten Threads müssten in diesem Fall warten. Es besteht aber die Möglichkeit einer parallelen Implementierung. Abbildung 26 visualisiert die Grundidee. Die Hälfte der Threads berechnet in jeder Iteration das Minimum zweier Einträge. Damit reduziert sich die Laufzeit auf $O(\log(\delta))$.

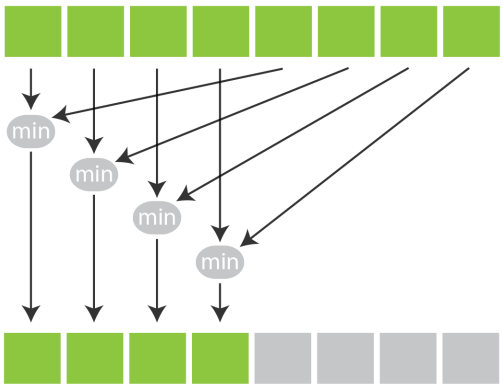



Abbildung 26: Grundgedanke der parallelen Reduktion. Quelle: [110].

Mit einem zusätzlichen Kunstgriff lässt sich die Laufzeit weiter senken. Theoretisch müsste man nach jeder Iteration alle Threads synchronisieren, weil das Resultat jedes einzelnen Threads in der nächsten Iteration benötigt wird. Mit einer expliziten Synchronisation geht jedoch immer ein kleiner Zeitverlust einher. Die Architektur von CUDA verwaltet die bereitgestellten Threads in Gruppen zu 32 parallelen Threads. Diese Gruppen werden als Warps bezeichnet. Instruktionen innerhalb eines Warps sind SIMD-synchron. Das bedeutet für die parallele Reduktion, dass man sich die explizite Synchronisation spart, wenn sich die Einträge,

über welche das Minimum eruiert werden soll, auf eine Anzahl kleiner oder gleich 64 belaufen[8]. Dies kommt in der Stereoanalyse öfters vor, weil die Anzahl der möglichen Disparitätsstufen nicht zu groß wird. Für eine effiziente parallele Implementierung sollte die Anzahl der Einträge zudem ein Faktor von 2 sein [127]. In der Implementierung werden daher die Einträge immer auf die nächste Länge, welche einen Faktor von 2 darstellt, künstlich ergänzt.

### 6.4.3 Nachbearbeitung

Die Nachbearbeitungsschritte von [9] finden sich ebenfalls in der GPU-Implementierung. Dazu gehören die Verdeckungsbehandlung und das anschließende Auffüllen verdeckter Regionen. Bei den Durchgängen dynamischer Programmierung mit Verdeckungsbehandlung wird wie bei den anderen Durchgängen davor verfahren, außer, dass für jedes Pixel überprüft wird, ob es verdeckt ist oder nicht. Das geschieht mit einem zuvor erstellten Verdeckungsbild. Für verdeckte Pixel werden alle Strafkosten der Glattheitsbedingung auf null gesetzt.

Das Auffüllen verdeckter Regionen folgt allerdings einem einfachen Prinzip und erzeugt streifenähnliche Artefakte. Zur Entfernung dieser Artefakte wird, wie in [126] vorgeschlagen, nach dem Auffüllen ein bilateraler Medianfilter auf die resultierende Disparitätskarte angewendet.

#### Bilateraler Medianfilter

Für die Ermittlung des bilateralen Medians innerhalb eines Fensters bestimmt man zuerst die bilateralen Gewichte im Fensterbereich. Die Größe des Fensters wird durch einen Radius bestimmt, welcher in der Implementierung auf sieben gesetzt ist. Jedes berechnete Gewicht wird danach mit dem Disparitätswert derselben Stelle kombiniert und als Schlüssel-Wert-Paar gespeichert. Die resultierenden Paare sortiert man anschließend aufsteigend nach der Disparität. Das Endergebnis der Filterung besteht aus jenem Disparitätswert, bei dem die Summe der bilateralen Gewichte zum ersten Mal größer oder gleich der Hälfte der Gesamtsumme der Gewichte ist. Die Kalkulation der Summe beginnt beim niedrigsten Disparitätswert. Auf diese Weise wird sichergestellt, dass der Median möglichst nur über Disparitätswerte ermittelt wird, die eine gewisse Signifikanz besitzen, ausgedrückt durch eine höhere Gewichtung.

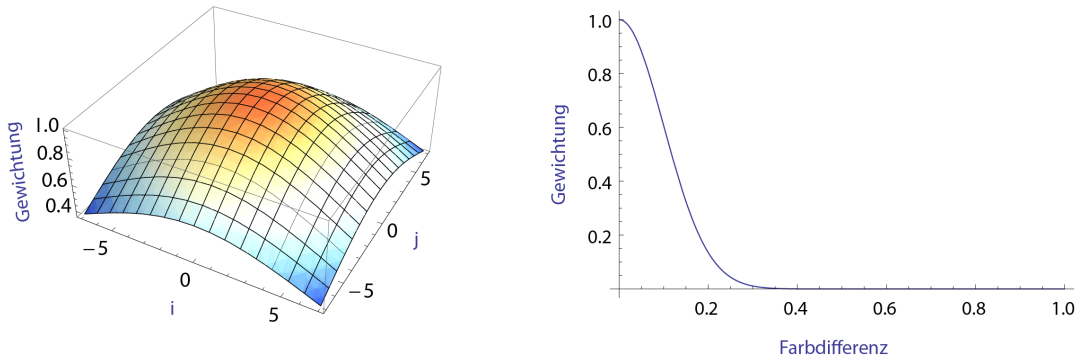
---

8 Mit 32 Threads können bis zu 64 Werte parallel minimiert werden, weil je ein Thread das Minimum zweier Werte bestimmt.

Gaußsche Funktionen legen die bilateralen Gewichte fest, sowohl beim eingesetzten räumlichen Filter als auch beim Similaritätsfilter [76]. Sie finden sich in Abbildung 27 visualisiert. Die Gewichte als Kombination der beiden Filter lauten:

$$W_{i,j}^{bf} = \mathrm{e}^{-\frac{1}{2}\left(\frac{i^2+j^2}{\sigma_s^2}\right)} \cdot \mathrm{e}^{-\frac{1}{2}\left(\frac{|I_{i,j}-I_{0,0}|}{\sigma_c^2}\right)} \tag{6.3}$$

wobei $i^2+j^2$ den quadrierten Abstand vom Fensterzentrum $(0,0)$ repräsentiert und $|I_{i,j}-I_{0,0}|$ die Summe der quadrierten Differenzen der RGB-Farbwerte von den Farben an den Stellen $(i,j)$ und $(0,0)$. Ersteres misst demnach die geometrische Distanz vom Zentrum und Letzteres die Farbsimilarität. Die Parameter $\sigma_s^2$ und $\sigma_c^2$ regulieren das Streuverhalten der Funktionen.




a) räumlicher Filter

b) Similaritätsfilter

Abbildung 27: Die gaußschen Funktionen für den bilateralen Medianfilter. Beim Similaritätsfilter wird die Farbdifferenz zwischen zwei RGB-Farben auf normalisierten Farbkanäle zwischen 0 und 1 ermittelt, deshalb die niedrigen Werte.

Ein bitonischer Sortierer [86], von dem bereits eine Implementierung im CUDA SDK existiert, bringt die Schlüssel-Wert-Paare in eine aufsteigende Reihenfolge. Der Sortierer repräsentiert ein paralleles Sortierverfahren und die genutzte Implementierung eignet sich speziell für Einträge, welche im gemeinsamen Speicher Platz finden. Die Gesamtsumme der Gewichte ist ebenfalls effizient parallel ermittelbar. Für das Auffinden des Medians, bei dem

die Summe der Gewichte größer oder gleich der Hälfte der Gesamtsumme ist, muss allerdings ein Kompromiss zwischen iterativer und paralleler Kalkulation eingegangen werden, zumal er sich nicht vollständig parallel eruieren lässt.

Bei einer Fenstergröße von 15x15 ergeben sich beispielsweise 225 bilaterale Gewichte. Das Aufrunden auf den nächsten Faktor von 2 führt zu 256 Werten. Es wird nun mit paralleler Berechnung in zwei Durchgängen die Summe von vier nebeneinanderstehenden Werten berechnet. Der erste Durchgang summiert jeweils zwei Nachbarn und der zweite Durchgang jeweils zwei Summen aus dem ersten Durchgang. Danach bleiben 64 Elemente übrig. Iterativ wird nun die Stelle eruiert, bei der die Summe der Gewichte über die Hälfte der Gesamtsumme ansteigt. Somit weiß man auf vier Stellen genau, wo sich der gesuchte Index befindet. Zu guter Letzt muss man noch von dem gefundenen Index ausgehend die nächsten vier Werte überprüfen, um genau festzustellen, bei welchem von den vier in Frage kommenden Elementen die Summe der Gewichte größer als die Hälfte der Gesamtsumme wird. Die Kombination aus paralleler und iterativer Berechnung ist schneller als eine reine iterative Berechnung.

# 7 Ergebnisse der Implementierung

Abbildung 28 illustriert die Ergebnisse der GPU-Implementierung (ohne und mit bilateraler Medianfilterung) zusammen mit den Ergebnissen der CPU-Implementierung. Tabelle 3 dokumentiert die Resultate anhand der Middlebury-Evaluation. Die Laufzeiten finden sich in Tabelle 4.

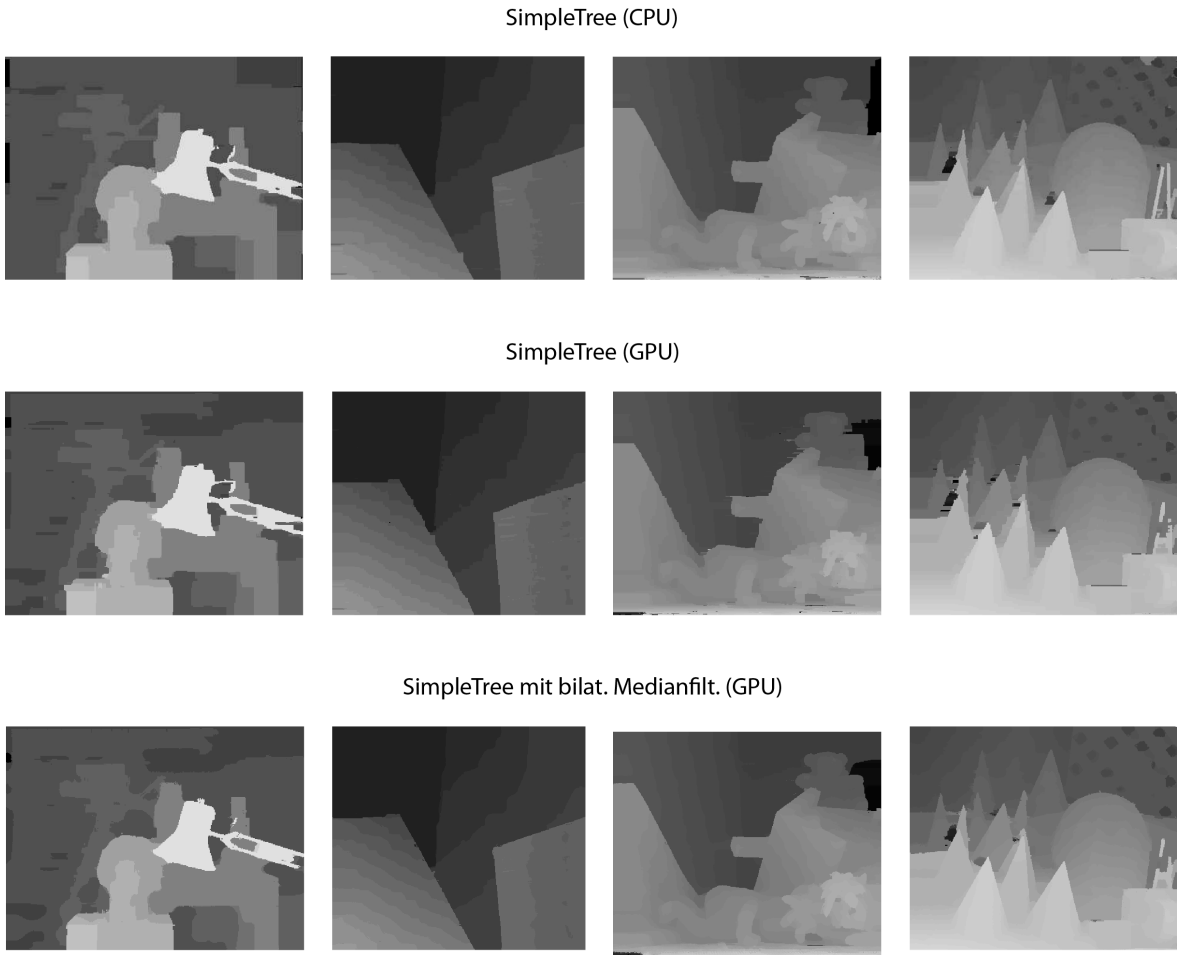



Abbildung 28: Die Ergebnisse der GPU-Implementierung von [9] mit und ohne bilateraler Medianfilterung sowie die Ergebnisse der CPU-Implementierung. Quelle: [9].

| | **Tsukuba** | | **Venus** | | **Teddy** | | **Cones** | | **Rang** |
|---|---|---|---|---|---|---|---|---|---|
| | n. v.[+] | alle | n. v.[+] | alle | n. v.[+] | alle | n. v.[+] | alle | Ø |
| SimpleTree (CPU) | 1.86 | 2.56 | 0.42 | 0.76 | 7.31 | 12.70 | 4.00 | 9.74 | 45.3 |
| SimpleTree (GPU) | 2.29 | 3.30 | 1.24 | 1.82 | 8.03 | 13.80 | 4.33 | 10.50 | 62.7 |
| SimpleTree (GPU) mit bilat. Medianfilt. | 2.15 | 2.75 | 0.84 | 1.19 | 8.22 | 13.50 | 3.77 | 9.49 | 54.8 |

Tabelle 3: Die Resultate der CPU- und GPU-Implementierung anhand der Middlebury-Evaluierung. [+](n.v. = nicht verdeckt).

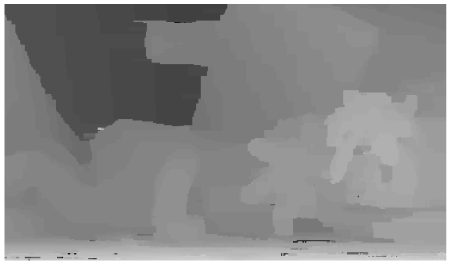
| | Bildgröße: 384x288<br>Disparitäten: 16 | | Bildgröße: 450x375<br>Disparitäten: 60 | |
|---|---|---|---|---|
| | Laufzeit (ms) | Beschleunigungsfaktor | Laufzeit (ms) | Beschleunigungsfaktor |
| SimpleTree (CPU) | 180 | - | 570 | - |
| SimpleTree (GPU) | 60 | 3x | 133 | 4.3x |
| SimpleTree (GPU) mit bilat. Medianfilt. | 117 | 1.5x | 219 | 2.6x |

Tabelle 4: Die Laufzeiten der CPU-Implementierung und der GPU-Implementierungen für die Stereobildpaare Tsukuba und Cones bzw. Teddy. Obwohl die CPU-Implementierung teilweise schon parallelisiert wurde, läuft die GPU-Implementierung (ohne bilateralen Medianfilter) dank zusätzlicher Parallelisierung um den Faktor 3 bis 4 schneller.

Die eingesetzten Parameter blieben für alle Bilder unverändert und wurden heuristisch ermittelt. Im Folgenden werden die Parameter nach Einsatzgebieten aufgelistet:

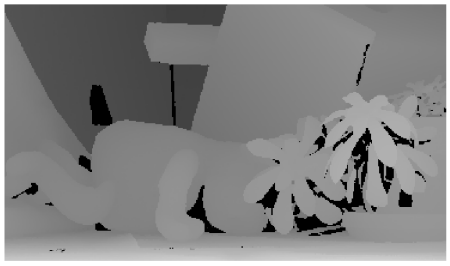
- SimpleTree: $P1 = 30, P2 = 65, T = 65, P3 \cdot P2 = 110$ und $\lambda = 0.025$
- Gradientenberechnung: $\alpha = 0.2, \tau_1 = 100$ und $\tau_2 = 25$
- Bilaterale Medianfilterung: $\sigma_s = 7, \sigma_c = 0.1$ und $r = 7$

Die Resultate der GPU-Implementierung ohne bilaterale Medianfilterung reichen visuell und quantitativ nicht ganz an die CPU-Implementierung heran. Sie liegen etwa einen Prozentpunkt hinter den Originalwerten. Der Grund hierfür liegt vermutlich in der unterschiedlichen Ähnlichkeitsmetrik. So müssen die Glattheitskosten für bessere Resultate etwas erhöht werden, was wiederum dazu führt, dass über kleinere Details mehr geglättet wird. Daher bleiben beispielsweise die kleinen Stäbchen rechts vorne im Stereobildpaar Cones nicht vollständig erhalten. Die bilaterale Medianfilterung verbessert die Resultate deutlich, visuell wie quantitativ. Sie schafft es bei Cones sogar die Originalwerte zu übertreffen. Beim Bildpaar Teddy kommt es allerdings kaum zu einer quantitativen Reduktion der Fehler. Das fußt wohl darin, dass die Bereiche, wo das Filtern die Disparitäten korrigiert, als Bereiche mit unbekannter Disparität gelten (siehe Abb. 29). Optisch verbessert sich die Disparitätskarte merklich.



a) ohne bilat. Medianfilt.

b) mit bilat. Medianfilt.



c) tatsächliche Disparitäten (schwarz unbekannt)

d) entsprechender Ausschnitt im Referenzbild

Abbildung 29: Betrachtet man die Resultate ohne und mit Filterung bei einem Ausschnitt des Stereobildpaares Teddy, fällt auf, dass das Filtern die Resultate rein visuell deutlich verbessert. Allerdings spiegelt sich dieser Umstand nicht quantitativ wider. Dies dürfte an den unbekannten Regionen liegen. Dort korrigiert das Filtern die Disparitäten zwar, dies wird aber nicht erfasst.

Weil die bilaterale Medianfilterung keinen Bestandteil des Originalalgorithmus darstellt, wird hier sowohl bei der Laufzeit als auch bei den anderen Ergebnissen zwischen Implementierung mit und ohne Filterung unterschieden. Für kleine Bilder mit wenig Disparitätsstufen ergibt sich ohne bilaterales Filtern eine Beschleunigung um das Dreifache gegenüber der CPU-Implementierung. Bilder mittlerer Größe und ungefähr 60 Disparitätsstufen beschleunigt die GPU sogar um das Vierfache. Eine Ursache für den höheren Beschleunigungsfaktor bei mehr Disparitätsstufen findet sich in der parallelen Berechnung von $min_{i \in D}\, l'(q,i)$, welche erst ab einer größeren Anzahl möglicher Disparitäten einen relevanten Geschwindigkeitsvorteil mit sich bringt. Es sei noch einmal darauf verwiesen, dass bei der CPU-Implementierung die Rechenlast bereits auf zwei Kerne aufgeteilt worden ist und SIMD-Instruktionen die parallelen Fähigkeiten der CPU ausschöpfen. Der Geschwindigkeitsgewinn gegenüber einer rein seriellen Implementierung wäre sicherlich höher. Das bilaterale Filtern dauert 57 ms beim Stereobildpaar Tsukuba und 86 ms bei Cones und Teddy, weshalb sich der Tempogewinn erheblich vermindert.

## 7.1 Effekt des Gradiententerms

Es wurde zusätzlich der Nutzen des Gradiententerms von Gleichung (6.1) überprüft. Er hält sich nach den hier durchgeführten Experimenten in Grenzen. Die Ergebnisse verbessern sich nur marginal gegenüber derselben Metrik, welche die Gradienteninformation nicht beachtet. Für die Experimente ohne Gradiententerm wurde der Parameter $\alpha$ auf null gesetzt. Tabelle 5 stellt die Resultate mit und ohne Gradiententerm zahlenmäßig gegenüber. Der Prozentsatz der falschen Zuordnung verringert sich teilweise nur im Promillebereich, teilweise erhöhen sich die Werte. Zumal allerdings die Berechnung des Gradienten kaum die Laufzeit beeinflusst, kostet der Einsatz des Gradiententerms nicht viel.

Der Grund für das schlechte Abschneiden des Gradiententerms liegt wohl daran, dass er kaum Information einfließen lässt, welche nicht schon durch den Datenterm erfasst wird. Abbildung 30 zeigt das linke und rechte Gradientenbild für das Stereobildpaar Teddy. In homogenen Regionen liefert der Gradient per Definition keine Information, weshalb große Bereiche des Bildes einfach nur grau sind. Und die Information von Kanten dürfte schon im ausreichenden Maße durch die Farbdifferenzen in den Datenkosten gegeben sein.

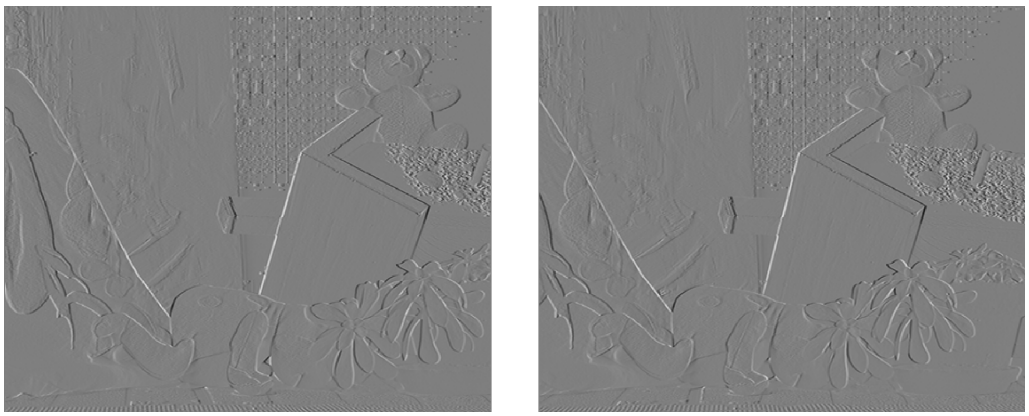


Abbildung 30: Linkes und rechtes Gradientenbild für das Stereobildpaar Teddy. Weiß repräsentiert einen positiven Gradienten, schwarz einen negativen Gradienten und grau bedeutet der Gradient ist null.

| | **Tsukuba** | | **Venus** | | **Teddy** | | **Cones** | |
|---|---|---|---|---|---|---|---|---|
| | n. v. | alle | n. v. | alle | n. v. | alle | n. v. | alle |
| SimpleTree_GPU (ohne Gradientent., ohne bilat. Medianfilt.) | 2.33 | 3.31 | 1.21 | 1.79 | 8.36 | 13.90 | 4.40 | 10.40 |
| SimpleTree_GPU (mit Gradientent., ohne bilat. Medianfilt.) | 2.29 | 3.30 | 1.24 | 1.82 | 8.03 | 13.80 | 4.33 | 10.50 |
| SimpleTree_GPU (ohne Gradientent., mit bilat. Medianfilt.) | 2.28 | 2.89 | 0.63 | 0.99 | 8.53 | 13.70 | 3.93 | 9.50 |
| SimpleTree_GPU (mit Gradientent., mit bilat. Medianfilt.) | 2.15 | 2.75 | 0.84 | 1.19 | 8.22 | 13.50 | 3.77 | 9.49 |

Tabelle 5: Der Nutzeffekt des Gradiententerms von Gleichung (6.1) ist nach den hier durchgeführten Versuchen limitiert.

In der nächsten Tabelle (6) werden die Laufzeiten einzelner Schritte der GPU-Implementierung für das Stereobildpaar Teddy aufgelistet; zusätzlich dazu die Zeit für die Verdeckungsbehandlung. Wie zu erkennen ist, entfallen auf die Verdeckungsbehandlung 55 Prozent der gesamten Laufzeit. Mehr als die Hälfte der Laufzeit wird hiefür gebraucht, weil zuerst ein Verdeckungsbild erstellt werden muss und die Durchgänge dynamischer Programmierung, welche die Kosten anhand des Verdeckungsbildes anpassen, etwas langsamer laufen.

| | **Laufzeiten (ms)** |
|---|---|
| Gradientenberechnung | 0.2 |
| DRV-Erstellung (mit Grad.) | 4.4 |
| 1x Durchlauf dyn. Prog. | ~6.0 |
| Erstellung d. Verdeckungsbildes | 0.2 |
| Disparitätsselektion (Gewinner-nimmt-Alles) | 4.2 |
| Verdeckungsbehandlung (insgesamt) | ~73 |

Tabelle 6: Laufzeiten einzelner Schritte der GPU-Implementierung, wie auch der gesamten Verdeckungsbehandlung

## 7.2 Verdeckungsbehandlung

Aufgrund des hohen Aufwandes für die Verdeckungsbehandlung stellt sich die Frage nach ihrem Benefit. Abbildung 31 und Tabelle 7 demonstrieren den Unterschied zwischen Disparitätskarten mit und ohne Verdeckungsbehandlung für die Stereobilder Cones und Teddy.

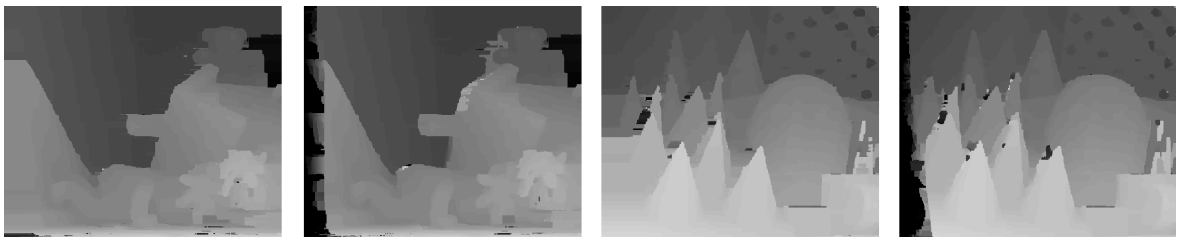


Abbildung 31: Die Verdeckungsbehandlung benötigt 55 Prozent der gesamten Laufzeit, verbessert dafür die Ergebnisse deutlich. Die linken Bilder der beiden Stereobildpaare zeigen jeweils die Resultate mit Verdeckungsbehandlung. Bei den rechten Bildern wurde die Verdeckungsbehandlung weggelassen.

| | **Teddy** | | **Cones** | |
|---|---|---|---|---|
| | n. v. | alle | n. v. | alle |
| SimpleTree_GPU (mit Verdeckungsb.) | 8.18 | 13.90 | 4.33 | 10.50 |
| SimpleTree_GPU (ohne Verdeckungsb.) | 9.05 | 18.40 | 4.38 | 14.90 |

Tabelle 7: Quantitative Erfassung der Resultate mit und ohne Verdeckungsbehandlung.

Die Fehler reduzieren sich mit Verdeckungsbehandlung um annähernd 4.5 Prozent. Dieser Gewinn spricht für den zeitlichen Aufwand. Ein möglicher Kompromiss zwischen Genauigkeit und Geschwindigkeit wäre der Einsatz einer weniger präzisen Verdeckungsberücksichtigung, e.g. ein einfacher Links-rechts-Konsistenztest. Die Disparität des Zielbildes könnte man dabei nach der Methode wie in [119] erstellen. Die Autoren eruieren in diesem Artikel die Disparitätskarte für das Zielbild aus der Kostenmatrix des Referenzbildes. Dazu versetzen sie das Pixel $p$ in Gleichung (5.10) (mit $H[p,d]$ als Kostenmatrix) um die an dieser Stelle ermittelte Disparität $d_p$ und bestimmen dort die Disparität des Zielbildes. Derart erspart man sich die explizite Kalkulation einer Kostenmatrix mit dem ehemaligen Zielbild als Referenzbild und

umgekehrt, verliert allerdings an Genauigkeit. Welche Methode schließlich für die Verdeckungsbehandlung herangezogen wird, dürfte von den Anforderungen des Anwendungsgebietes abhängen.

Die Qualität der eingesetzten Verdeckungsbehandlung bzw. des gesamten Stereoalgorithmus lässt sich auch an dem erzeugten Verdeckungsbild (engl. *occlusion map*) ermessen. In Abbildung 32 ist das Verdeckungsbild vor und nach dem Nachbearbeitungsschritt zu sehen, welches mit der in der Implementierung eingesetzten Verdeckungsbehandlung von [9] erstellt wurde. Der Nachbearbeitungsschritt eliminiert einzelne Pixel, welche beispielsweise durch schräg verlaufende Oberflächen entstehen. Das nachbearbeitete Verdeckungsbild beinhaltet fast alle tatsächlichen Verdeckungen. Das spricht für die Güte der Verdeckungsbehandlung.

Abbildung 32: Links das Verdeckungsbild vor dem Nachbearbeitungsschritt; in der Mitte das Bild danach. Ganz rechts die tatsächlichen Verdeckungen. Quelle: [51].

### 7.2.1 Einfluss des Verdeckungsbildes

Der Einsatz des Verdeckungsbildes soll verhindern, dass verdeckte Pixel bei den Durchgängen mit dynamischer Programmierung falsche Disparitäten in nicht verdeckte Regionen propagieren. Eine korrekte Disparitätszuweisung in verdeckten Regionen mithilfe der binokularen Disparität zu gewinnen, ist nicht möglich, weil in diesen Regionen keine korrespondierenden Elemente im anderen Bild existieren. Die Zuweisungskosten weisen in der Regel willkürliche Werte auf, abhängig von der Beschaffenheit der verdeckten Region. Aus diesem Grund möchte man diese Kosten bei den Durchgängen nicht berücksichtigen. Dazu setzt man die Strafkosten der Glattheitsbedingung bei den verdeckten Pixeln auf null.

Der Einfluss dieses Vorgehens ist in Abbildung 33 illustriert. Zu sehen sind die Disparitätskarten von Rückwärtsdurchgängen mit dynamischer Programmierung in horizontaler Richtung

mit und ohne Einsatz des Verdeckungsbildes[9]. An der linken Kante des Kopfes im Vordergrund zeigt sich, dass mithilfe des Verdeckungsbildes das Propagieren der Disparität des Vordergrundes in Bereiche des Hintergrundes deutlich unterbunden werden kann. Ähnliches passiert auch an der linken Kante der Lampe. Das Verdeckungsbild erfüllt demnach seine Funktion.

Abbildung 33: Einfluss des Verdeckungsbildes. Links die Disparitätskarte nach einem Rückwärtsdurchgang ohne Gebrauch des Verdeckungsbildes, rechts dis Disparitätskarte mit Gebrauch des Verdeckungsbildes. Das Ausbreiten falscher Disparitäten über verdeckte Regionen hinweg wurde beim rechten Bild verhindert.

### 7.2.2 Resultate auf anderen Stereobildern

Bisher wurden nur für die vier Bilder der Middlebury-Evaluierung (Tsukuba, Venus, Teddy und Cones) die berechneten Disparitätskarten präsentiert. Diese Bilder decken zwar schon einige interessante Situationen ab, allerdings noch nicht alle. Im Folgenden werden deshalb Ergebnisse weiterer Bilder vorgestellt. Abbildung 34 veranschaulicht die Ergebnisse der GPU-Implementierung mit und ohne bilateraler Medianfilterung für vier Stereobildpaare des Middlebury-Datensatzes 2005. Die Anzahl der Disparitätsstufen beträgt 80. Das ist bei Bildern von etwa 430 Pixel Breite schon fast ein Fünftel des Bildes. Mit Schwierigkeiten verbunden sind vor allem homogene Regionen und Verdeckungen. Beim Wäschekorb im Bild Laundry beispielsweise dürfte sowohl die sich wiederholende Textur als auch das homogene Weiß des Korbes Probleme bereiten.

9 Die Durchgänge arbeiten auf den ursprünglichen Datenkosten und die Disparitätskarten werden mittels einer Gewinner-nimmt-alles-Strategie auf den aktualisierten Kosten bestimmt.

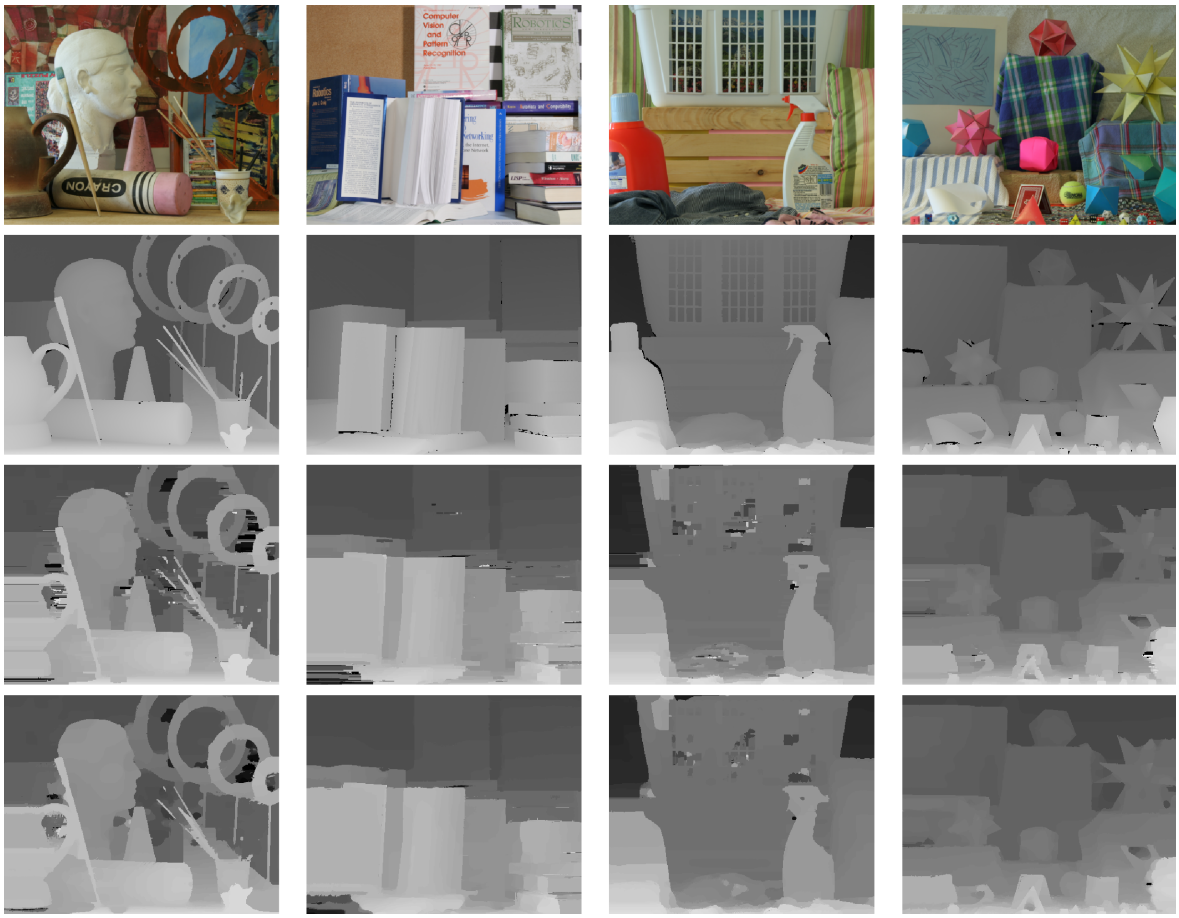



Abbildung 34: Die Ergebnisse der GPU-Implementierung für die Stereobildpaare Art, Books, Laundry und Moebius des Middlebury-Datensatzes 2005. Die erste Reihe (von oben) zeigt die Referenzbilder, die zweite die tatsächlichen Disparitäten, die dritte die Disparitätskarten der GPU-Implementierung ohne bilateraler Medianfilterung und die letzte die Disparitätskarten nach der Filterung.

In Abbildung 35 werden die Resultate für vier Bilder des Middlebury-Datensatzes 2006 gezeigt. Beim Stereobildpaar Cloth3 ist erkennbar, dass bei idealen Voraussetzungen, wie ausreichend Textur und nicht zu großen Tiefensprüngen, eine Rekonstruktion der Tiefe sehr gut gelingt.

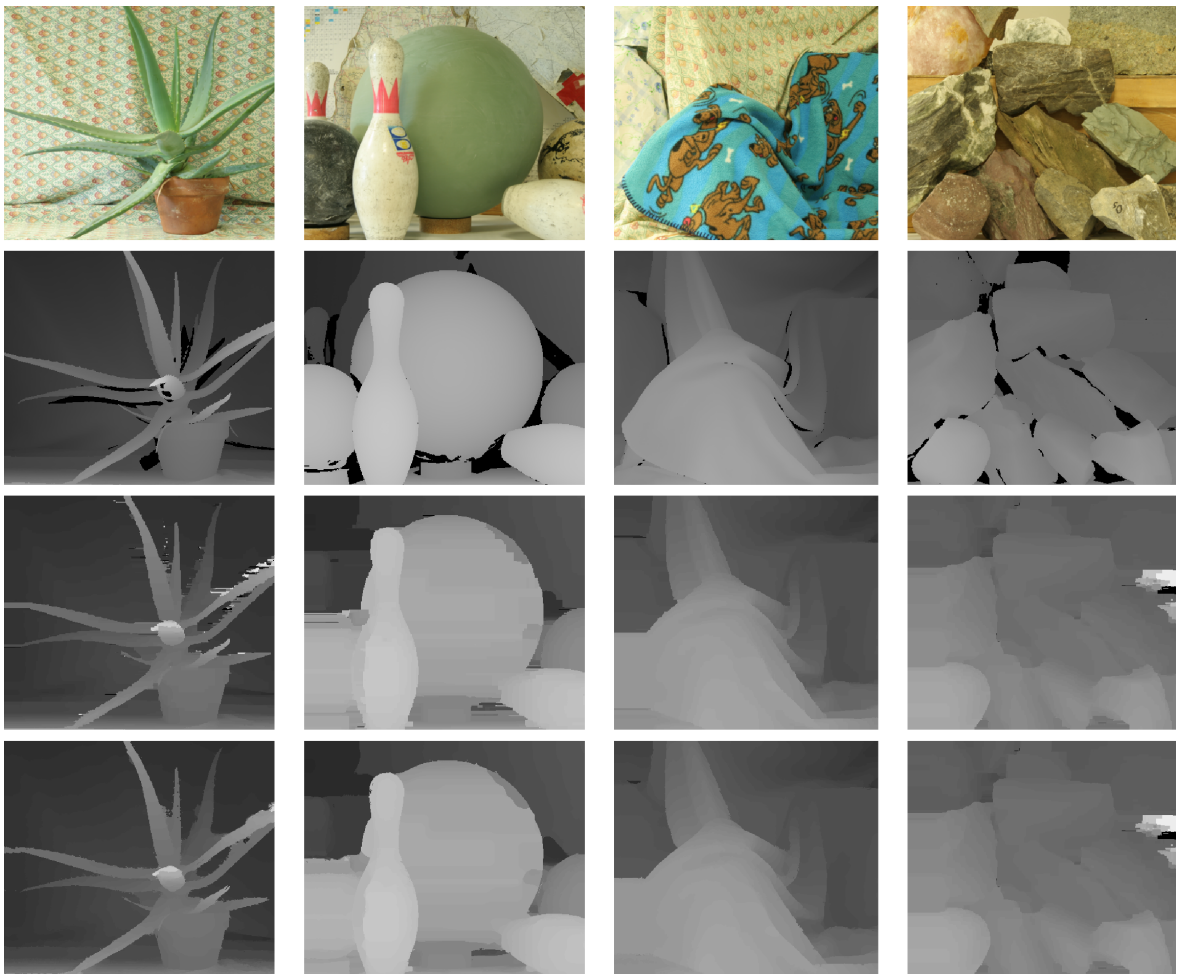


Abbildung 35: Die Ergebnisse der GPU-Implementierung für die Stereobildpaare Aloe, Bowling2, Cloth3 und Rocks2 des Middlebury-Datensatzes 2006. Die erste Reihe (von oben) zeigt die Referenzbilder, die zweite die tatsächlichen Disparitäten, die dritte die Disparitätskarten der GPU-Implementierung ohne bilateraler Medianfilterung und die letzte die Disparitätskarten nach der Filterung.

### 7.2.3 Ergebnisse bei radiometrischen Unterschieden

Die Ähnlichkeitsmetrik SAD (Summe der absoluten Differenzen) und die darauf basierende Metrik von [126] (Gleichung (6.1)), welche in dieser Arbeit zum Einsatz kommt, eignen sich laut [125] nicht für Bilder mit radiometrischen Differenzen. Das konnte hier experimentell bestätigt werden. In Abbildung 36 sind die Ergebnisse für zwei Stereobildpaare mit radiometrischen Unterschieden zu sehen. Beide Stereobilder basieren auf dem Bildpaar Cloth3. Bei dem Stereobild in der ersten Reihe von Abbildung 36 wurden die Bilder mit verschiedenen Belichtungszeiten aufgenommen und das linke Bild ist deutlich dunkler als das rechte. Die fehler-

hafte Disparitätskarte verdeutlicht, dass die eingesetzte Metrik nicht für diese Art von Unterschied in den Bildern geeignet ist. Etwas besser die Resultate für die zweite Reihe in Abbildung 36, wo die Bilder gleiche Belichtungszeiten aber eine unterschiedliche Beleuchtung aufweisen. Die Resultate dürften hier auf der rechten Seite der Disparitätskarte deswegen gut ausfallen, weil der Beleuchtungsunterschied zu keinen großen Farbunterschieden in den beiden Bildern führt.

Abbildung 36: Radiometrische Unterschiede im linken und rechten Bild können von der hier eingesetzten Metrik kaum kompensiert werden. Die Ergebnisse verschlechtern sich signifikant. Die erste Reihe zeigt linkes und rechtes Bild des Bildpaars Cloth3 aufgenommen mit verschiedenen Belichtungszeiten und die damit berechnete Disparitätskarte. Die zweite Reihe zeigt linkes und rechtes Bild bei einer unterschiedlichen Beleuchtung (mit gleichen Belichtungszeiten) und die berechnete Disparitätskarte.

## 7.3 Auswirkung der Optimierung

Zum Schluss sei noch die gesamte Auswirkung der verwendeten Optimierung anhand des Stereobildpaares Cones verdeutlicht. Abbildung 37 zeigt dazu zum einen die Disparitätskarte für das DRV[10] (i.e. vor der Optimierung) und zum anderen die Disparitätskarte nach der letzten Nachbearbeitung. Es ist erstaunlich zu sehen, wie die Anwendung von ein wenig Wissen über

10 Die Disparitätskarte des DRVs wurde mit einer Gewinner-nimmt-alles-Strategie auf den ursprünglichen Datenkosten erstellt.

die Beschaffenheit der uns umgebenden Welt, verbunden mit einer guten Optimierung, die Resultate verbessert. Das a priori Wissen ist in diesem Fall durch die Glattheitsbedingung gegeben. D.h. allein das Wissen, dass sich die Tiefe in einer Szene außer an Objektgrenzen immer nur kontinuierlich ändert, bewirkt diese immense Verbesserung der Ergebnisse.

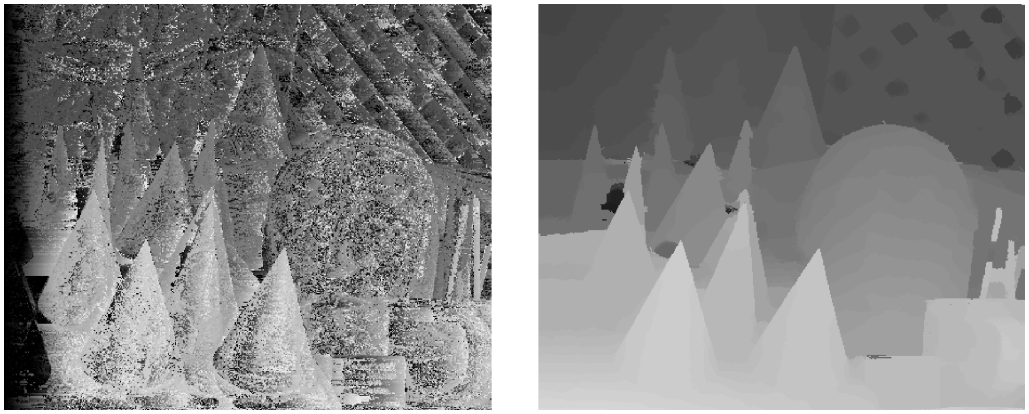


Abbildung 37: Die Wirkung der eingesetzten Optimierung veranschaulicht anhand des Stereobildpaares Cones. Links die Disparitätskarte vor der Optimierung und rechts die Disparitätskarte nach der Optimierung.

# 8 Zusammenfassung

Die statische Stereoanalyse, ein Bereich des Maschinellen Sehens, beschäftigt sich damit, Computern räumliches Sehen beizubringen. Unzählige Anwendungsgebiete bekräftigen den Nutzen von und erhöhen die Nachfrage nach schnellen und effizienten Verfahren in diesem Bereich. Als Vorbild für die Verfahren dienen die Natur und unter anderem wir Menschen. Mithilfe der binokularen Disparität, dem Lageunterschied von Objekten in zwei in ihrer Position leicht versetzten Aufnahmen, lässt sich die Tiefe einer Szene relativ leicht eruieren. Ein Grund dafür, weshalb man bei der statischen Stereoanalyse diese binokulare Disparität verwendet. Die größte Herausforderung auf diesem Gebiet besteht im Identifizieren und Zuordnen gleichartiger Elemente in den Bildern, welche in der Literatur als Korrespondenzproblem Bezeichnung findet. Für die Korrespondenzsuche erschwerend sind die Ambiguität von Merkmalen, homogene Regionen, wiederkehrende Textur und heterogene Merkmalsausprägungen.

Globale und lokale Verfahren versuchen, das Korrespondenzproblem auf unterschiedliche Weise unter Zuhilfenahme verschiedener Annahmen und Bedingungen zu meistern. Wegen der großen Anzahl an Veröffentlichungen konnte ich nur eine Auswahl an Methoden zur Lösung des Korrespondenzproblems vorstellen. Mein Augenmerk lag besonders auf schnellen Algorithmen mit guten Ergebnissen.

Lokale Verfahren setzen auf den Vergleich von Regionen um das Pixel von Interesse. Das korrekte adaptive Anpassen des Bereiches an den Bildinhalt, sei es via veränderbares Fenster oder via Gewichtung, stellt die primäre Herausforderung bei lokalen Methoden dar. Viele realisierte Echtzeitverfahren setzen lokale Verfahren ein, weil sie nicht zu komplex sind und effizient implementiert werden können.

Zur Optimierung der Disparitätsverteilung minimieren globale Verfahren eine Energiefunktion. Eine besonders wichtige Optimierungsmethode repräsentiert die dynamische Programmierung, weil sie einen guten Kompromiss zwischen Geschwindigkeit und Genauigkeit darstellt. Diese Methode wird am ausführlichsten von allen beschrieben. Mehrere Durchläufe dynamischer Programmierung lassen sich ohne große Anstrengung parallelisieren. Und das Potential der Parallelisierung von Algorithmen ist dank der Mehrkern-Revolution der letzten Jahre enorm. Vor allem programmierbare Graphikkarten bieten aufgrund ihrer Konzeption einen hohen Grad an Parallelisierung.

Die Zielsetzung für den praktischen Teil der Diplomarbeit lautete daher, einen effizienten, leicht parallelisierbaren Stereoalgorithmus auf einer Graphikkarte zu implementieren. Das Vorhaben, damit einen Geschwindigkeitsgewinn zu erreichen, konnte umgesetzt werden. Der implementierte Algorithmus läuft um bis das Vierfache schneller. Bei dem Stereobildpaar Tsukuba läuft die Implementierung mit 16 Bildern pro Sekunde und bei den Bildern Teddy und Cones mit 7 Bildern pro Sekunde. Solche Bildraten genügen für viele Echtzeit- oder Nahe-Echtzeit-Anwendungen.

Die Resultate der GPU-Implementierung blieben nur einen Prozentpunkt hinter den Originalwerten. Die leicht unterschiedlichen Ergebnisse dürften der anderen Ähnlichkeitsmetrik zuzuschreiben sein. Bilaterale Medianfilterung in einem Nachbearbeitungsschritt verbesserte die Ergebnisse merklich, allerdings verbunden mit einer deutlich erhöhten Laufzeit.

# Literaturverzeichnis


[1] E. B. Goldstein. Wahrnehmungspsychologie - Der Grundkurs. 7th ed., *Spektrum Akad. Verl.*, 2008.

[2] R. Klette, A. Koschan and K. Schlüns. Computer Vision - Räumliche Information aus digitalen Bildern. *Vieweg*, 1996.

[3] B. Julesz. Binocular depth perception of computer-generated patterns. *The Bell System Technical Journal*, 39(5):1125-1161, 1960.

[4] B. Jähne, H. Haußecker and P. Geißle, (eds.). Handbook of computer vision and applications - Volume 2 Signal processing and pattern recognition. *Academic Press*, 1999.

[5] B. L. Anderson and K. Nakayama. Toward a general theory of stereopsis: Binocular matching, occluding contours, and fusion. *Psychological Review*, 101(3):414-445, 1994.

[6] H. A. Mallot, P. A. Arndt and H. H. Bülthoff. A psychophysical and computational analysis of intensity-based stereo. *Biological Cybernetics*, 75(3):187-198, 1996.

[7] P. N. Belhumeur. A Bayesian approach to binocular stereopsis. *International Journal of Computer Vision*, 19(3):237-260, 1996.

[8] K. Prazdny. Detection of binocular disparities. *Biological Cybernetics*, 52(2):93-99, 1985.

[9] M. Bleyer and M. Gelautz. Simple but effective tree structures for dynamic programming-based stereo matching. *International Conference on Computer Vision Theory and Applications*, 415-422, 2008.

[10] R. Koch. Three-dimensional modeling of objects from image sequences. *in: B. Jähne, H. Haußecker and P. Geißle, (eds.): Handbook of computer vision and applications - Volume 3 Systems and applications*, Academic Press, 1999.

[11] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *Conference on Computer Vision and Pattern Recognition*, 1: 519-528, 2006.

[12] D. Bradley, T. Boubekeur and W. Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. *Conference on Computer Vision and Pattern Recognition*, 1-8, 2008.

[13] W. Ribarsky, T. Wasilewski and N. Faust. From urban terrain models to visible cities. *IEEE Computer Graphics and Applications*, 22(4):10-15, 2002.

[14] J. Hu, S. You and U. Neumann. Approaches to large-scale urban modeling. *IEEE Computer Graphics and Applications*, 23(6):62-69, 2003.

[15] M. Pollefeys, D. Nistér, J. -M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. -J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch and H. Towles. Detailed real-time urban 3D reconstruction from video. *International Journal of Computer Vision*, 78(2/3):143-167, 2008.

[16] N. Cornelis, K. Cornelis and L. V. Gool. Fast compact city modeling for navigation pre-visualization. *Conference on Computer Vision and Pattern Recognition*, 2: 1339-1344, 2006.

[17] M. Pollefeys, L. V. Gool, M. Vergauwen, K. Cornelis, F. Verbiest and J. Tops. 3D recording for archaeological fieldwork. *IEEE Computer Graphics and Applications*, 23(3):20-27, 2003.

[18] A. Hogue, A. German and M. Jenkin. Underwater environment reconstruction using stereo and inertial data. *International Conference on Systems, Man and Cybernetics*, 2372-2377, 2007.

[19] Christian-Albrechts-University of Kiel, Institute of Computer Science, Department Multimedia Information Processing. Structure from motion with uncalibrated image sequences. Images taken from the website, *URL: www.mip.informatik.uni-kiel.de/tiki-index.php?page=3D+reconstruction+from+images*, last visited: 10.09.2011.

[20] Christian-Albrechts-University of Kiel, Institute of Computer Science, Department Multimedia Information Processing. 3D-Modelle aus Bildern. Flyer for CeBIT 2003, *URL: www.mip.informatik.uni-kiel.de/tiki-download_file.php?fileId=76*, last visited: 10.09.2011.

[21] Jet Propulsion Laboratory, Mars Exploration Rover Mission, Press Release 22.01.2004,. Mars exploration rover mission status. Images taken from the website, *URL: marsrover.nasa.gov/newsroom/pressreleases/20040122a.html*, last visited: 10.09.2011.

[22] GeoPerspectives. Flyer of GeoPerspectives product portfolio (2006). *URL: www.bluesky-world.com/geoperspectives/*, last visited: 15.04.2010.

[25] Y.-M. Feng, D.-X. Li, K. Luo and M. Zhang. Asymmetric bidirectional view synthesis for free viewpoint and three-dimensional video. *IEEE Transactions on Consumer Electronics*, 55(4):2349-2355, 2009.

[34] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Wilson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein and A. Angelova. Computer vision on mars. *International Journal of Computer Vision*, 75(1):67-92, 2007.

[23] O. Schreer. Stereoanalyse und Bildsynthese. *Springer*, 2005.

[24] D. Scharstein. View synthesis using stereo vision. *Lecture Notes in Computer Science*, 1583, 1999.

[26] F. Isgró, E. Trucco and L.-Q. Xu. Towards teleconferencing by view synthesis and large-baseline stereo. *International Conference on Image Analysis and Processing*, 198-203, 2001.

[27] R. Yang and Z. Zhang. Eye gaze correction with stereovision for video-teleconferencing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):956-960, 2004.

[28] A. Smolic, K. Mueller, P. Merkle, C. Fehn, P. Kauff, P. Eisert and T. Wiegand. 3D video and free viewpoint video - technologies, applications and MPEG standards. *International Conference on Multimedia and Expo*, 2161-2164, 2006.

[29] K. Kraus. Photogrammetrie Band 1 - Geometrische Informationen aus Photographien und Laserscanneraufnahmen. 7th ed., *de Gruyter Lehrbuch*, 2004.

[30] E. Gülch. Digital systems for automated cartographic feature extraction. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 33(B2):241-255, 2000.

[31] S. Kicherer, J. A. Malpica and M. C. Alonso. An interactive technique for cartographic feature extraction from aerial and satellite image sensors. *Sensors 2008*, 8(8):4786-4799, 2008.

[32] R. Sim and J. J. Little. Autonomous vision-based exploration and mapping using hybrid maps and Rao-Blackwellised particle filters. *International Conference on Intelligent Robots and Systems*, 2082-2089, 2006.

[33] D. Klimentjew, A. Stroh, S. Jockel and J. Zhang. Real-time 3D environment perception: An application for small humanoid robots. *International Conference on Robotics and Biomimetics*, 354-359, 2008.

[35] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki and K. Fujimura. The intelligent ASIMO: System overview and integration. *International Conference on Intelligent Robots and Systems*, 3:2478-2483, 2002.

[36] H. van Dijck. Object recognition with stereo vision and geometric hashing. Dissertation, *University of Twente*, 1999.

[37] S. Kosov, K. Scherbaum, K. Faber, T. Thormählen and H.-P. Seidel. Rapid stereo-vision enhanced face detection. *International Conference on Image Processing*, 1221-1224, 2009.

[38] T. Kanade, K. Oda, A. Yoshida, M. Tanaka and H. Kano. Video-rate z keying: A new method for merging images. Technical Report CMU-RI-TR-95-38, *The Robotics Institute, Carnegie Mellon University*, 1995.

[39] R. Gvili, A. Kaplan, E. Ofek and G. Yahav. Depth keying. *Proceedings of SPIE*, 5006:564-574, 2003.

[40] A. Gruen. Scientific-technological developments in photogrammetry and remote sensing between 2004 and 2008. *in: Z. Li, J. Chen and E. Baltsavias, (eds.): Advances in photogrammetry, remote sensing and spatial information sciences - 2008 ISPRS congress book*, CRC Press, 2008.

[41] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1/2/3):7-42, 2002.

[42] M. Bleyer. Segmentation-based stereo and motion with occlusions. Dissertation, *Vienna University of Technology*, 2006.

[43] V. S. Nalwa. A guided tour of computer vision. *Addison-Wesley*, 1993.

[44] X. Jiang and H. Bunke. Dreidimensionales Computersehen - Gewinnung und Analyse von Tiefenbildern. *Springer-Verlag*, 1997.

[45] M. Sonka, V. Hlavac and R. Boyle. Image processing, analysis, and machine vision. 3rd ed., *Thomson*, 2008.

[46] A. Koschan. What is new in computational stereo since 1989 - A survey on current stereo papers. Technical Report 93-22, *Technical University of Berlin*, 1993.

[47] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401-406, 1998.

[48] S. T. Barnhard and M. A. Fischler. Computational stereo. *ACM Computing Surveys*, 14(4):553-572, 1982.

[49] U. R. Dhond and J. K. Aggarwal. Structure from stereo - A review. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1489–1510, 1989.

[50] M. Z. Brown, D. Burschka and G. D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):993-1008, 2003.

[51] D. Scharstein and R. Szeliski. The middlebury stereo vision page. *URL: vision.middlebury.edu/stereo/*, last visited: 10.09.2011.

[52] M. Gong and Y.-H. Yang. Fast stereo matching using reliability-based dynamic programming and consistency constraints. *Proceedings of the Ninth International Conference on Computer Vision*, 2:610-617, 2003.

[53] R. Szeliski and R. Zabih. An experimental comparison of stereo algorithms. *Proceedings of the International Workshop on Vision Algorithms - Theory and Practice*, 1-19, 1999.

[54] Y. Boykov, O. Veksler and R. Zabih. A variable window approach to early vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1283-1294, 1998.

[55] H. Hirschmüller. Improvements in real-time correlation-based stereo vision. *Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision*, 141-148, 2001.

[56] S. A. Adhyapak, N. Kehtarnavaz and M. Nadin. Stereo matching via selective multiple windows. *Journal of Electronic Imaging*, 16(1):013012, 2007.

[57] F. Tombari, S. Mattoccia, L. Di Stefano and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. *Conference on Computer Vision and Pattern Recognition*, 1-8, 2008.

[58] M. Okutomi and T. Kanade. A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2):143-162, 1992.

[59] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window - theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920-932, 1994.

[60] O. Veksler. Stereo correspondence with compact windows via minimum ratio cycle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1654-1660, 2002.

[61] A. F. Bobick and S. S. Intille. Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181-200, 1999.

[62] D. Geiger, B. Ladendorf and A. Yuille. Occlusions and binocular stereo. *International Journal of Computer Vision*, 14(3):211-226, 1995.

[63] A. Fusiello, V. Roberto and E. Trucco. Efficient stereo with multiple windowing. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 858–863, 1997.

[64] H. Hirschmüller, P. R. Innocent and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47((1/2/3)):229-246, 2002.

[65] S. O.-Y. Chan, Y.-P. Wong and J. K. Daniel. Dense stereo correspondence based on recursive adaptive size multi-windowing. *Proceedings of the Conference on Image and Vision Computing New Zealand*, 1:256–260, 2003.

[66] C. Demoulin and M. Van Droogenbroeck. Disparity map determination by multiple adaptive windows. *Proceedings of the Workshop on Circuit, Systems and Signal Processing*, 615-618, 2005.

[67] O. Veksler. Fast variable window for stereo correspondence using integral images. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1:556–561, 2003.

[68] S. B. Kang, R. Szeliski and J. Chai. Handling occlusions in dense multi-view stereo. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1:103–110, 2001.

[69] M. Gerrits and P. Bekaert. Local stereo matching with segmentation-based outlier rejection. *Proceedings of the Conference on Computer and Robot Vision*, 66-72, 2006.

[70] M. Gong, R. Yang, L. Wang and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *International Journal of Computer Vision*, 75(2):283-296, 2007.

[71] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin and C. Proy. Real time correlation-based stereo: algorithm, implementations and applications. Technical Report 2013, *INRIA*, 1993.

[72] T. Darrell. A radial cumulative similarity transform for robust image correspondence. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 656-662, 1998.

[73] Y. Xu, D. Wang, T. Feng and H.-Y. Shum. Stereo computation using radial adaptive windows. *Proceedings of the Conference on Pattern Recognition*, 3:595-598, 2002.

[74] A. Hosni, M. Bleyer, M. Gelautz and C. Rhemann. Local stereo matching using geodesic support weights. *International Conference on Image Processing*, 2093-2096, 2009.

[75] K.-J. Yoon and I. S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650-656, 2006.

[76] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. *Proceedings of the Sixth International Conference on Computer Vision*, 839-846, 1998.

[77] M. Gong and R. Yang. Image-gradient-guided real-time stereo on graphics hardware. *Proceedings of the Conference on 3-D Digital Imaging and Modeling*, 548–555, 2005.

[78] F. Tombari, S. Mattoccia, L. Di Stefano and E. Addimanda. Near real-time stereo based on effective cost aggregation. *Conference on Pattern Recognition*, 1-4, 2008.

[79] S. Z. Li. Markov random field modeling in image analysis. 3rd ed., *Springer-Verlag*, 2009.

[80] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068-1080, 2008.

[81] R. Szeliski. Computer vision - algorithms and applications. draft September 3, *Springer-Verlag*, 2010.

[82] J. Sun, N.-N. Zheng and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787-800, 2003.

[83] Y. Boykov, O. Veksler and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222-1239, 2001.

[84] I. J. Cox, S. L. Hingorani, S. B. Rao and B. M. Maggs. A maximum likelihood stereo algorithm. *Computer Vision and Image Understanding*, 63(3):542-567, 1996.

[85] R. Bellman. Dynamic programming. 6th ed., *Princeton University Press*, 1972.

[86] T. H. Cormen, C. E. Leiserson, R. Rivest and C. Stein. Algorithmen - Eine Einführung. 2nd ed., *Oldenbourg Wissenschaftsverlag*, 2007.

[87] G. D. Forney Jr.. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268-278, 1973.

[88] H. H. Baker and T. O. Binford. Depth from edge and intensity based stereo. *International Joint Conference on Artificial Intelligence*, 2:631-636, 1981.

[89] R. D. Arnold. Automated stereo perception. Dissertation, *Stanford University*, 1983.

[90] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(2):139-154, 1985.

[91] H. Sadeghi, P. Moallem and S. A. Monadjemi. Feature based dense stereo matching using dynamic programming and color. *International Journal of Information and Mathematical Sciences*, 4(3):179-186, 2008.

[92] G. Van Meerbergen, M. Vergauwen, M. Pollefeys and L. Van Gool. A hierarchical symmetric stereo algorithm using dynamic programming. *International Journal of Computer Vision*, 47(1/2/3):275-285, 2002.

[93] S. Forstmann, Y. Kanou, J. Ohya, S. Thuering and A. Schmitt. Real-time stereo by using dynamic programming. *Conference on Computer Vision and Pattern Recognition Workshops*, 3:29-36, 2004.

[94] C. Sun. A fast stereo matching method. *International Conference on Digital Image Computing: Techniques and Applications*, 95-100, 1997.

[95] C. Sun. Fast stereo matching using rectangular subregioning and 3D maximum-surface techniques. *International Journal of Computer Vision*, 47(1/2/3):99-117, 2002.

[96] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Conference on Computer Vision*, 1073-1080, 1998.

[97] J. C. Kim, K. M. Lee, B. T. Choi and S. U. Lee. A dense stereo matching using two-pass dynamic programming with generalized ground control points. *Conference on Computer Vision and Pattern Recognition*, 2:1075-1082, 2005.

[98] P. N. Belhumeur and D. Mumford. A bayesian treatment of the stereo correspondence problem using half-occluded regions. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 506-512, 1992.

[99] W.-P. Dong, Y.-S. Lee and C.-S. Jeong. A stereo matching using variable windows and dynamic programming. *Australian Conference on Artificial Intelligence*, 1277-1280, 2005.

[100] L. Wang, M. Liao, M. Gong, R. Yang and D. Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. *Third International Symposium on 3D Data Processing, Visualization, and Transmission*, 798-805, 2006.

[101] M. Gong and Y.-H. Yang. Real-time stereo matching using orthogonal reliability-based dynamic programming. *IEEE Transactions on Image Processing*, 16(3):879-884, 2007.

[102] M. Mozerov. An effective stereo matching algorithm with optimal path cost aggregation. *Lecture Notes in Computer Science*, 4174:617-626, 2006.

[103] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328-341, 2008.

[104] J. Salmen, M. Schlipsing, J. Edelbrunner, S. Hegemann and S. Lüke. Real-time stereo vision - making more out of dynamic programming. *International Conference on Computer Analysis of Images and Patterns*, 1096-1103, 2009.

[105] O. Veksler. Stereo correspondence by dynamic programming on a tree. *Conference on Computer Vision and Pattern Recognition*, 2:384-390, 2005.

[106] Y. Deng and X. Lin. A fast line segment based dense stereo algorithm using tree dynamic programming. *European Conference on Computer Vision*, 201-212, 2006.

[107] C. Lei, J. Selzer and Y.-H. Yang. Region-tree based stereo using dynamic programming optimization. *Conference on Computer Vision and Pattern Recognition*, 2378-2385, 2006.

[108] C.-H. Sin, C.-M. Cheng, S.-H. Lai and S.-Y. Yang. Geodesic tree-based dynamic programming for fast stereo reconstruction. *International Conference on Computer Vision Workshops*, 801-807, 2009.

[109] M. Bleyer and M. Gelautz. Simple but effective tree structures for dynamic programming-based stereo matching. *International Conference on Computer Vision Theory and Applications*, presentation slides, 2008.

[110] J. Sanders and E. Kandrot. CUDA by example - An introduction to general-purpose GPU programming. *Addison-Wesley Professional*, 2010.

[111] M. Fischer and A. Stiller. Feurige Tesla-Strömungen - GPGPUs im High Performance Computing. *Magazin für Computertechnik c't*, 09(11):148-149, 2009.

[112] M. Bertuch, H. Gieselmann, A. Trinkwalder and C. Windeck. Supercomputer zu Hause - Ungenutzte PC-Rechenleistung ausschöpfen. *Magazin für Computertechnik c't*, 09(7):88-95, 2009.

[113] NVIDIA. CUDA C programming guide (version 4.0). *NVIDIA Corporation*, 2011.

[114] D. B. Kirk and W.-m. W. Hwu. Programming massively parallel processors - A hands-on approach. *Morgan Kaufmann Publishers*, 2010.

[115] NVIDIA. CUDA C best practices guide (version 4.0). *NVIDIA Corporation*, 2011.

[116] D. B. Kirk and W.-m. W. Hwu. Lecture slides for Programming massively parallel processors (Chapter 3). *URL: www.elsevierdirect.com/companion.jsp?ISBN=9780123814722*, last visited: 10.09.2011.

[117] R. Kalarot and J. Morris. Comparison of FPGA and GPU implementations of real-time stereo vision. *Conference on Computer Vision and Pattern Recognition Workshops*, 9-15, 2010.

[118] L. Wang, M. Gong, M. Gong and R. Yang. How far can we go with local optimization in real-time stereo matching. *International Symposium on 3D Data Processing, Visualization, and Transmission*, 129-136, 2006.

[119] I. Ernst and H. Hirschmüller. Mutual information based semi-global stereo matching on the GPU. *International Symposium on Advances in Visual Computing*, 228-239, 2008.

[120] I. D. Rosenberg, P. L. Davidson, C. M. R. Muller and J. Y. Han. Real-time stereo vision using semi-global matching on programmable graphics hardware. *ACM SIGGRAPH 2006 Sketches*, Article 89, 2006.

[121] J. Gibson and O. Marques. Stereo depth with a unified architecture GPU. *Conference on Computer Vision and Pattern Recognition Workshops*, 1-6, 2008.

[122] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80-113, 2007.

[123] J. Congote, J. Barandiaran, I. Barandiaran and O. Ruiz. Realtime dense stereo matching with dynamic programming in CUDA. *Proceedings of the 19th Spanish Congress of Graphical Informatics*, 231-234, 2009.

[124] K. Zhu, M. Butenuth and P. d'Angelo. Comparison of dense stereo using CUDA. *European Conference on Computer Vision: Workshop on Computer Vision on GPUs*, 2010.

[125] H. Hirschmüller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1582-1599, 2009.

[126] C. Rhemann, A. Hosni, M. Bleyer, C. Rother and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *Conference on Computer Vision and Pattern Recognition*, 3017-3024, 2011.

[127] M. Harris. Optimizing parallel reduction in CUDA. *NVIDIA Developer Technology*, Whitepaper, 2008.