Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2441

Ein segmentierungsgestützter Variationsansatz zur Stereorekonstruktion

Fabian Andres

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. Andrés Bruhn
Betreuer/in:	Prof. Dr.-Ing. Andrés Bruhn
Beginn am:	11. November 2013
Beendet am:	13. Mai 2014
CR-Nummer:	I.4.3, I.4.6, I.4.8

Kurzfassung

Eine der wichtigsten Aufgabenstellungen in der Computer-Vision ist die Berechnung des Optischen Flusses und das damit nah verwandte Stereo Matching. Das ultimative Ziel dieser beiden Techniken ist die Bewegung der Objekte in einer Bildsequenz zu schätzen und diese anschließend dreidimensional zu rekonstruieren. Viele Algorithmen setzen dabei auf einen pixelbasierten Ansatz. Sie berechnen die Korrespondenzen der Bilder auf Pixelebene und weisen deshalb eine hohe Laufzeit und Komplexität auf. Die Algorithmen sind somit nicht geeignet, wenn eine schnelle Berechnung benötigt wird. Das Ziel dieser Arbeit ist die Entwicklung eines Algorithmus, welcher auf einem segmentbasierten Ansatz aufbaut. Hierbei werden die Pixel des Bildes zu einzelnen Objekten zusammenfasst. Durch die einstellbare Segmentierung wird die Komplexität des zu lösenden Problems stark verringert und somit die Laufzeit verbessert. Zudem erhöht sich die Robustheit durch die segmentweise Zusammenfassung der Bildinformation.

Abstract

One of the most important tasks in computer-vision is the computation of the optical flow and the closely related stereo matching. The ultimate goal of these techniques is to estimate the movement of objects in an image sequence and subsequently to reconstruct a three-dimensional scene. Many algorithms rely on a pixel-based approach. They calculate the correspondence for each pixel and thus have a high runtime and complexity. Therefore, these algorithms are not suitable when fast calculations are required. The aim of this work is to develop an algorithm which is based on a segment-based approach, which merges the pixels of the image into individual objects. Due to the adjustable segmentation, the complexity of the problem to be solved is greatly reduced, thus improving the runtime. In addition, the robustness is increased by the segment-wise Summary of image information.

Inhaltsverzeichnis

1 Einleitung

Mit zunehmendem Fortschritt der Technik, werden immer mehr Bereiche des Alltags automatisiert und es wird versucht die Handhabung von Aufgaben zu vereinfachen. Was anfangs nur der Industrie in der Fertigungstechnik und späteren Qualitätssicherung vorbehalten war, hat sich in den letzten Jahren stark in den Alltag der Menschen integriert. Was früher noch manuell und mit viel Aufwand oder Gefahren zu bewältigen war, kann heute automatisiert und ohne Einfluss eines Menschen durchgeführt werden, was zum einen auch an der Leistung der Hardware lag, welche vor 1-2 Jahrzehnten nicht die nötige Leistung aufbrachte um solch rechenintensive Vorgänge in Echtzeit zu berechnen. Gerade die Automobilindustrie, Medizin oder Robotik setzen immer mehr auf computerassistierende Elemente, um die Fahrt sicherer zu machen und eine Operation oder Diagnose einfacher durchführen zu können. Die Computer-Vision, welche sich damit beschäftigt, Computern das Sehen mithilfe von Sensoren oder Kameras beizubringen, befasst sich mit zwei der interessantesten Aufgabenstellungen in diesem Zusammenhang. Zum einen gibt es den Optischen Fluss, welcher versucht, die Pixel eines Bildes mit denen eines anderen, welches zeitlich nach dem Ersten spielt, zu verbinden und so die Bewegung einer Videosequenz oder Bilderfolge berechnet. Dies ist zum Beispiel in den Luxusklassen vieler Automobilhersteller integriert, um bei Auffahrunfällen oder gefährlichen Situationen mit Passanten automatisch den Bremsvorgang einleiten zu können. Was sich aus der Sicht eines Menschen leicht anhört, ist für eine Maschine jedoch schwer zu bewältigen, da für sie lediglich Bildpunkte mit einem bestimmten Wert vorliegen, aus welcher sich keinerlei Bedeutung erkennen lässt. Nah verwandt damit ist das sogenannte Stereo Matching, welches nicht eine Bilderfolge verwendet, sondern zwei gleichzeitig geschossene Fotos desselben Objekts zeigen und nun versucht, die Szene räumlich darzustellen. Diese Problemstellung ist oft in der Robotik zu finden, um Robotern die Tiefenwahrnehmung der Umgebung beizubringen. Viele bisherige Algorithmen setzten jedoch auf einen pixelbasierten Ansatz, welcher jeden Pixel des Bildes zuordnet, was viel Zeit beansprucht und zu Ungenauigkeiten führen kann. Aus diesem Grund geht die Entwicklung solcher Algorithmen in eine Richtung, welche die Bilder zuvor segmentiert, also versucht die Objekte des Bildes zusammenzufassen und anschließend die Verschiebung des kompletten Objektes berechnet. Das Ziel dieser Arbeit ist es nun einen Algorithmus vorzustellen, welcher zuerst die Bilder mit einem Segmentierungsalgorithmus bearbeitet, wodurch die Pixel eines Objektes zusammengefasst werden und anschließend durch einen Optischen Fluss Ansatz die Tiefe der Szene berechnet. Durch die kleinere Anzahl an sich zu verschiebenden Elementen verringert sich die Laufzeit, je nach der stärke der Segmentierung, und es entsteht eine genauere Schätzung des Bildes.

1.1 Aufbau der Arbeit

Die Arbeit ist in 6 Kapitel unterteilt. In der Einleitung wird eine Einführung und eine kleine Motivation beschrieben, gefolgt von verwandten Arbeiten, welche sich mit ähnlichen Verfahren beschäftigen. Ka-

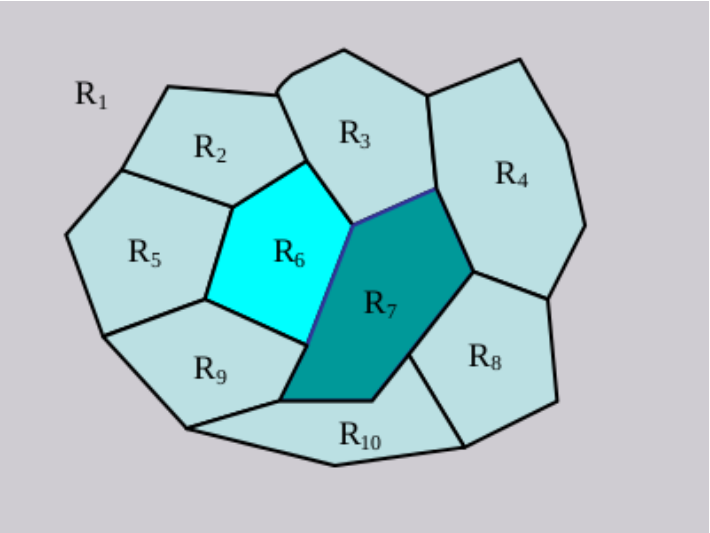
pitel 2 ist eine Einführung in die Mathematischen Grundlagen, welche zum Verständnis des späteren Algorithmus benötigt werden. Kapitel 3 handelt von der Herleitung und vollständigen Beschreibung der einzelnen Schritte eines pixelbasierten Ansatzes, welcher als Grundlage für die Entwicklung einer segmentierten Variante dient. Kapitel 4 behandelt den Kern dieser Arbeit, in welchem die wichtigsten Schritte der segmentierten Variante vorgestellt und bis zum finalen Algorithmus beschrieben werden. Anschließend folgt eine Evaluation der Ergebnisse und ein Vergleich mit anderen Arbeiten. Die Arbeit endet schließlich mit einer Zusammenfassung und einem Ausblick.

1.2 Verwandte Arbeiten

Der folgende Abschnitt handelt über Arbeiten, welche einen Algorithmus vorstellen der ebenfalls auf einer Segmentierung basiert und das Problem auf eine andere Art und Weise angehen als diese Arbeit. Diese weisen zu einem gewissen Grad Parallelen auf, welche auf die aktuelle Arbeit Einfluss haben.

1.2.1 Cooperative Optimization

Das erste Paper von Wang und Zheng mit dem Titel „A Region Based Stereo Matching Algorithm Using Cooperative Optimization“[WZ08] benutzt einen Algorithmus, welcher auf einem lokalen Ansatz basiert. Das Bild wird zuerst mit dem Mean-Shift Algorithmus segmentiert und danach mit einem lokalen Korrelationsalgorithmus gelöst, welches eine Grundlage der Lösung ergibt und später durch weitere Berechnungen gegen Ausreißer korrigert werden soll. Danach wird ein affiner Ansatz implementiert, der eine Ebene für jedes Segment berechnet basierend auf einer initialen Schätzung eines Korrelationsalgorithmus. Danach wird die Normale zu jeder Ebene mittels eines Eigenwertproblems berechnet, wobei die Lösung der Eigenvektor ist, welcher zu dem kleinsten Eigenwert gehört . Als nächstes wird ein lokaler Glattheitsterm aufgestellt, welche das zu berechnende Segment und all seine Nachbarn enthält, auch Subtargets genannt. Danach werden alle Subtargets alternierend gelöst.



.

Abbildung 1.1: Abbildung der lokalen Lösung der Segmente mit ihren Nachbarn des Cooperative Optimmization Algorithmus [WZ08]

1.2.2 Global Framework for Stereo Computation

Das Paper von Tao et al. mit dem Titel „A Global Matching Framework for Stereo Computation "[TSK01] stellt ein Framework zur berechnung der Tiefe vor, welches das Bild zuerst segmentiert und anschließend mit einem lokalen Fenster bestimmte Stellen des Bildes matched. Danach wird versucht den berechneten Fluss für jedes Segment mithilfe der benachbarten Segmente zu verbessern, indem Nachbarn in der Tiefe verändert werden und je nach Verbesserung oder Verschlechterung der Lösung dieser Wert übernommen wird. Da durch die aufwändige Suche sehr viele Vergleiche durchgeführt werden müssen, wird eine Warping Strategie eingeführt, welche das Referenzbild auf das zweite Bild registriert. Nun wird eine 3-Schichten-Segmentierung eingefügt, in welcher das zu berechnenden Segment der Richtigen Tiefe zugeordnet wird. Dies geschieht dadurch, dass das aktuelle Segment gelöscht wird und die jeweiligen Nachbarn auf verschiedene Tiefen gesetzt werden. Als nächstes wird ermittelt, zu welcher Tiefe das Segment besser passt, indem alle Möglichkeiten durchprobiert werden und die beste Lösung schließlich ausgewählt wird.

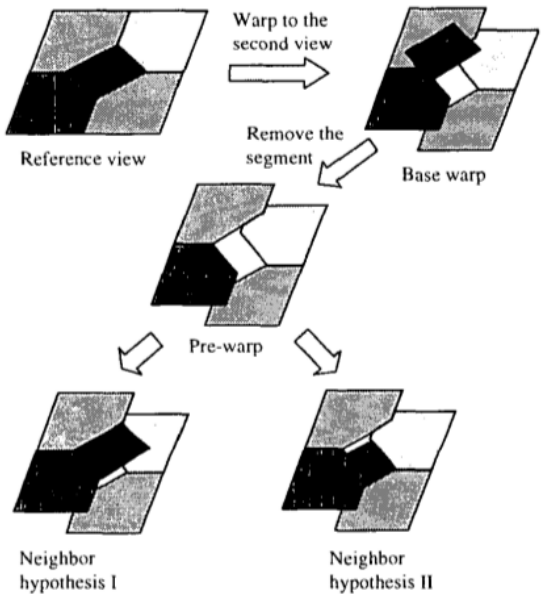


Abbildung 1.2: Abbildung des Warping Vorgangs. Das Bild wird gewarped, das Segment gelöscht und mehrere verschiedene Tiefen durchprobiert [TSK01].

2 Grundlagen

2.1 Mathematisches Grundwissen

Zuerst folgt eine Erklärung der wichtigsten Begriffe und Operatoren, welche für das spätere Verständnis der Algorithmen und verschiedenen Berechnungen benötigt werden. Zur besseren Veranschaulichung werden Beispiele für Funktionen mit einer sowie mit mehreren Veränderlichen angegeben.

2.1.1 Funktion

Bei Funktionen wird zwischen Skalarwertigen Funktionen und Vektorwertigen Funktionen unterschieden.

Skalarwertige Funktionen sind Funktionen, welche von einer Menge $\mathbb{R}^n \to \mathbb{R}$ abbilden. Die Dimension n steht für die Anzahl der Veränderlichen, jedoch wird auch bei mehreren Variablen auf ein Skalar abgebildet. Solche Funktionen sind zum Beispiel:

$$f(x) = x^2 + 2x + 2 \tag{2.1}$$

$$f(x, y, z) = x^2 + y^2 + z^2 \tag{2.2}$$

Vektorwertige Funktionen unterscheiden sich in der Hinsicht, dass von $\mathbb{R}^n \to \mathbb{R}^m$ abgebildet wird, also nicht mehr wie zuvor auf ein Skalar sondern stattdessen auf einen Vektor. Beispiele hierfür sind:

$$f(x) = \begin{pmatrix} 3x \\ cos(x) \end{pmatrix} \tag{2.3}$$

$$f(x, y, z) = \begin{pmatrix} sin(x) \\ cos(y) \\ cos(z) \end{pmatrix} \tag{2.4}$$

2.1.2 Differential

Das Differential beschreibt die Veränderung einer Funktion über eine bestimme Variable x. Also im Allgemeinen die Steigung an jedem Punkt der Funktion. Unterschieden wird zum einen zwischen dem Differenzenquotienten und dem Differentialquotient.

Differenzenquotient:

$$\varphi(x_0, x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{2.5}$$

Wobei x_0 und x_1 für zwei Punkte stehen und f für die abzuleitende Funktion. Dabei beschreibt $x_1 - x_0$ den Intervall von welchem die Steigung berechnet werden soll, was besonders im diskreten Bereich wichtig ist, da nicht nur kontinuierliche Funktionen betrachtet werden, sondern auch Diskrete, welche nur in bestimmten Abständen definiert sind.

Differentialquotient:

$$\lim_{x \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{2.6}$$

Wird nun $x_1 - x_0$ durch h ersetzt und lässt h gegen 0 laufen, also den Intervall unendlich klein setzt und somit die Steigung an jedem Punkt der Funktion berechnet, erhält man den Differentialquotienten, welcher allgemein als Ableitung einer Funktion bezeichnet und mit $\frac{d}{dx}$ abgekürzt wird, wobei x die Variable darstellt nach welcher man ableitet.

2.1.3 Integral

Das Integral ist neben dem Differential einer der wichtigsten Bereiche der Analysis. Es beschreibt die Fläche welche von der Funktion und der x-Achse eingeschlossen wird. Unterschieden wird zum einen zwischen dem bestimmten Integral, welches einem Intervall die jeweils dazugehörige Fläche zuordnet und dem unbestimmten Integral, welches aus der Menge aller Stammfunktionen besteht.

Bestimmtes Integral:

$$\int_a^b f(x)dx \tag{2.7}$$

wobei die Punkte a und b die Grenzen der Fläche markieren und f(x) die zu integrierende Funktion ist, dx steht für die Variable nach der integriert wird.

Unbestimmtes Integral:

$$\int f(x)dx = F(x) + C \tag{2.8}$$

wobei $F(x)$ für eine Stammfunktion von $f(x)$ steht. Leitet man sie ab, erhält man wieder die ursprüngliche Funktion $f(x)$, somit existieren unendlich viele Stammfunktionen, da das Absolutglied C durch das differenzieren wegfällt.

2.1.4 Nabla Operator

Der Nabla Operator ∇ ist ein Symbol um bestimmte Differentialoperatoren wie den Gradienten oder den Laplace Operator zu beschreiben, auf welchen später noch etwas eingegangen wird. Er ist definiert als:

$$\nabla = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} \tag{2.9}$$

und besteht aus den partiellen Ableitungen $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, $\frac{\partial}{\partial z}$. Für Funktionen mit mehreren Variablen erweitert sich der Vektor entsprechend.

2.1.5 Gradient

Der Gradient ist ein Vektor, welcher für jeden Punkt einer Funktion f in die Richtung des stärksten Anstiegs zeigt. Berechnet wird er durch das Produkt des Nabla Operators und f.

$$\nabla f(x,y,z) = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} f(x,y,z) = \begin{pmatrix} \frac{\partial}{\partial x} f(x,y,z) \\ \frac{\partial}{\partial y} f(x,y,z) \\ \frac{\partial}{\partial z} f(x,y,z) \end{pmatrix} = \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} \tag{2.10}$$

Betrachten man nun die Funktion $f(x,y,z) = x^2y^2z^2$ so ist der Gradient von f:

$$\nabla f(x,y,z) = \begin{pmatrix} \frac{\partial}{\partial x}(x^2y^2z^2) \\ \frac{\partial}{\partial y}(x^2y^2z^2) \\ \frac{\partial}{\partial z}(x^2y^2z^2) \end{pmatrix} = \begin{pmatrix} 2xy^2z^2 \\ x^2 \cdot 2yz^2 \\ x^2y^2 \cdot 2z \end{pmatrix} \tag{2.11}$$

2.1.6 Hesse-Matrix

Da spätere Grundlagen auf die Hesse-Matrix zurückgreifen und sie zum besseren Verständnis beiträgt, wird nun etwas näher auf sie eingegangen. Die Hesse Matrix ist eine Beschreibung aller zweiten partiellen Ableitungen einer Funktion f:

$$H(f(x,y,z)) = \begin{pmatrix} \frac{\partial^2}{\partial x^2} f(x,y,z) & \frac{\partial^2}{\partial xy} f(x,y,z) & \frac{\partial^2}{\partial xz} f(x,y,z) \\ \frac{\partial^2}{\partial yx} f(x,y,z) & \frac{\partial^2}{\partial y^2} f(x,y,z) & \frac{\partial^2}{\partial yz} f(x,y,z) \\ \frac{\partial^2}{\partial zx} f(x,y,z) & \frac{\partial^2}{\partial zy} f(x,y,z) & \frac{\partial^2}{\partial z^2} f(x,y,z) \end{pmatrix} = \begin{pmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{xz} & f_{zy} & f_{zz} \end{pmatrix} \tag{2.12}$$

wobei $\frac{\partial^2}{\partial xy}$ für die Ableitung nach x und danach nach y steht. Da die Ableitung $\frac{\partial^2}{\partial xy}$ und $\frac{\partial^2}{\partial yx}$ und alle anderen gemischten Ableitungen dem Gleichen entsprechen und die Matrix somit in der Diagonalen gespiegelt ist, kommt es vor, dass der untere Teil manchmal weggelassen wird.

2.1.7 Spur einer Matrix

Als die Spur einer Matrix bezeichnet man die Summe der Elemente der Hauptdiagonalen, welche gleichzeitig der Summe der Eigenwerte λ entspricht.

$$Spur(A) = \sum_{j=1}^{n} a_{jj} = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{j=1}^{n} \lambda_j \tag{2.13}$$

2.1.8 Laplace Operator

Bei dem Laplace Operator Δ handelt es sich um einen mathematischen Operator welcher gerade in der Physik eine große Rolle spielt. Dort dient er der Berechnung von physikalischen Feldern, beispielsweise der Ausbreitung der Wärme in einem geschlossenem Raum.

Zur Berechnung wird zum einen die Hesse Matrix benötigt

$$H(f) = \begin{pmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{pmatrix} \tag{2.14}$$

von welcher dann die Summe der Hauptdiagonalen berechnet wird. Der Laplace Operator ist somit die Spur der Hesse Matrix.

$$\Delta f = Spur(H(f)) = f_{xx} + f_{yy} + f_{zz} \tag{2.15}$$

2.1.9 Funktional

Funktionale unterscheiden sich zu Funktionen in der Hinsicht, dass die Parameter nicht wie bisher Elemente aus einem Zahlenraum $\mathbb{R}$ oder $\mathbb{Q}$ sind, sondern aus einem Funktionenraum V, in welchem jedes Element selbst eine Funktion ist.

$$G(f(x)) = \int_{\Omega} f(x)dx \tag{2.16}$$

Funktionale sind also Funktionen von Funktionen, welche auf einen skalaren Wert abbilden. Ein Teilgebiet der Mathematik, die Variationsrechnung, beschäftigt sich mit Extremale wie der Minimierung solcher Funktionale, welche auch in dieser Arbeit eine zentrale Rollen spielt. Der wohl wichtigste Begriff daraus sind die Euler-Lagrange Gleichungen, welche hier nur zur Vollständigkeit erwähnt und in einem späteren Kapitel näher erläutert werden.

2.1.10 Differentialgleichung

Differentialgleichungen sind mathematische Gleichungen, welche eine oder mehrere Variablen enthalten, sowie beliebig viele Ableitungen derselben Funktion auftreten können. Berechnet wird also nicht wie bisher ein Skalar oder ein Vektor, sondern eine Funktion f, welche die Differentialgleichung erfüllt.

Um dies zu verdeutlichen, ein kleines Beispiel:

$$y = y' \tag{2.17}$$

Gesucht ist eine Funktion y, welcher ihrer Ableitung entspricht. Durch Trennung der Variablen kann diese Funktion nun berechnet werden

$$\begin{aligned} y &= \frac{dy}{dx} \\ dx &= \frac{dy}{y} \\ 1 \cdot dx &= \frac{1}{y} dy \\ x &= ln(y) \\ e^x &= y \end{aligned} \tag{2.18}$$

Die gesuchte Funktion ist also die e-Funktion.

2.1.11 Taylorreihe

Die Taylorreihe ist ein Näherungsverfahren, mit welchem Funktionen um einen beliebigen Punkt herum entwickelt werden können. Das sogenannte Taylorpolynom besteht aus der Funktion selbst und ihrer Potenzreihe. Somit können komplizierte Funktionen schon mithilfe weniger Ableitungen hinreichend angenähert werden.

Das Taylorpolynom n-ten Grades sieht wie folgt aus:

$$T(f) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k \tag{2.19}$$

wobei a der Entwicklungspunkt, $k!$ die Fakultät ist und $f^{(k)}$ für die k-te Ableitung steht. Das Taylorpolynom ist für mehrere Variablen erweiterbar

$$T(f) = f(a,b) + \frac{f^{(1)}(a)}{1}(x-a) + \frac{f^{(1)}(b)}{1}(y-b) + ... \tag{2.20}$$

2.1.12 Linearisierung

Mithilfe der Linearisierung können nichtlineare Funktionen durch eine lineare Funktion approximiert werden. Dies wird benötigt, da im späteren Verlauf des Verfahrens so ein lineares Gleichungssystem entsteht, anstatt eines Nichtlinearen, welches um ein vielfaches schwieriger zu berechnen wäre. Erzielt wird dies mit der zuvor besprochenen Taylorreihe, mit dem Unterschied, dass nur die linearen Terme verwendet werden.

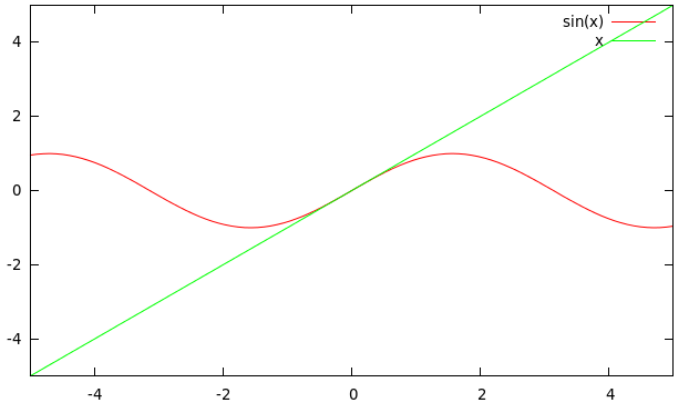


Abbildung 2.1: Rot: sin(x), **Grün:** lineare Annäherung an sin(x) mit Entwicklungspunkt 0

2.1.13 Gauß Kern

Die Gauß-Funktion ist eine Dichtefunktion, welche die Normalverteilung zu einer gegebenen Situation beschreibt. Sie ist unendlich oft differenzierbar zudem betrifft die Fläche unter der Funktion genau 1, was dem Mittelwert der Funktion nicht verändert. Dies ist besonders nützlich, da die Faltung einer Funktion mit dem Gauß Kern diese Eigenschaft überträgt und somit Bilder beliebig oft abgeleitet werden können.

Für den 2-dimensionalen Fall, sieht der Kern wie folgt aus:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.21}$$

wobei x und y für Koordinaten stehen und die Standardabweichung σ die Stelle des Wendepunkts markiert. Dadurch ist es möglich den Kern mit σ auf das Problem anzupassen, wobei ein großer Wert das Gewicht auf eine breitere Fläche verteilt und ein kleiner Wert den Fokus auf die mittleren Koordinaten legt. Hinzukommt, dass etwa 99.7% der Fläche in dem Intervall -3σ und 3σ liegen und deshalb der Rest meist vernachlässigt werden kann.

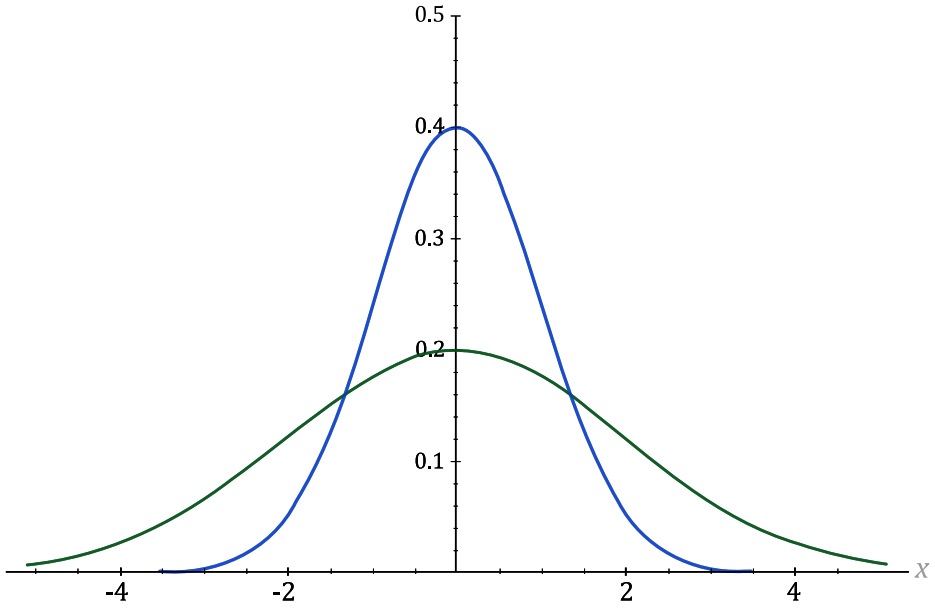


Abbildung 2.2: Eindimensionale Gauß-Funktionen mit unterschiedlichen Standardabweichungen. **Blau:** σ=1, **Grün:** σ=4 [Hem].

In Abbildung 2.2 lässt sich dies gut an der blauen Kurve erkennen. Die Standardabweichung beträgt 1, somit ändert die Kurve an 1 und -1 ihre Richtung und bei 3 und -3 beträgt der Wert schon beinahe 0. Im Zentrum wird stark Gewichtet wogegen schon die Werte > 3 und < 3 keine Rolle mehr spielen.

In der grünen Kurve, welche ein Standardabweichung von 4 hat, lässt sich erkennen, dass die Werte um den Hochpunkt weit weniger gewichtet werden und auch Werte, welche um den Bereich von -5 und 5 liegen, noch einen gewissen Einfluss auf das Ergebnis haben.

2.2 Bilder

Da dies ein Verfahren ist, welches hauptsächlich in Videosequenzen oder Bildern seine Anwendung findet, wird nun etwas näher auf die Bilder an sich und deren Aufbau eingegangen.

Bis ins späte 20. Jahrhundert wurde die Fotografieszene noch hauptsächlich von analogen Kameras dominiert, welche einen Film benötigten auf welches das fotografierte Objekt abgelichtet wurde. Erst Mitte der 70er Jahre änderte sich dies mit der Erfindung der Digitalkamera. Fotos wurden nicht länger mittels chemischen Prozessen gespeichert, sondern durch Halbleitersensoren in Rasteranordnung eingefangen.

2.2.1 Digitale Bilder

Da Computer nicht mit den kontinuierlichen Daten umgehen konnten, welche durch die analoge Fotografie entstanden, mussten die Fotos zuerst in eine Form gebracht werden mit welcher gerechnet werden konnte. Dies entstand durch die Abstufung der Fotos, also die Diskretisierung auf einzelne Bildpunkte, welche von Monitoren oder dem Fernseher bekannt sind. Dies wird auch oft als Digitalisierung bezeichnet.

Ein Digitales Bild wird durch verschiedene Elemente beschrieben:

Auflösung

Die Auflösung eines Bildes wird durch die Anzahl an Bildpunkten bestimmt. Jedes Foto hat eine bestimmte Anzahl an Bildpunkten in der Breite sowie in der Höhe. Multipliziert man diese Werte erhält man die Auflösung des Bildes. Je mehr Bildelemente vorhanden sind, desto feiner und detaillierter wird das Objekt auf dem Foto abgebildet. Die Auflösung gibt also die Abtastrate des analogen Bildes an. Oft wird auch der Begriff PPI (Pixel per inch) verwendet, welcher die Punktdichte eines Bildes oder Bildschirmes beschreibt.

Pixel

Jedes Bildelement, auch Pixel genannt, besteht aus den Farben rot, grün und blau. Diese werden in 3 oder manchmal auch 4 Subpixel unterteilt, also 4 Unterelemente eines Pixels, wobei die Farbe grün öfter vertreten ist, da das menschliche Auge am sensitivsten gegenüber grün ist. Durch die hohe Dichte und Größe der Subpixel verschwimmen so die einzelnen Farbkanäle, sodass das Auge dieses als einen Farbwert wahrnimmt. In der Regel haben Pixel eine quadratische oder rechteckige Form, es existieren jedoch Varianten, welche einer Raute oder einem Parallelogramm ähneln.

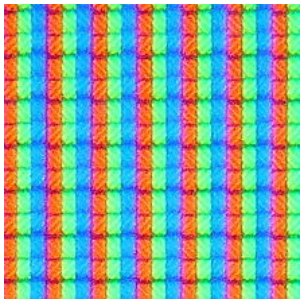


Abbildung 2.3: Stark vergrößertes Bild eines Monitors, zu erkennen sind die Einzelnen Pixel und deren rote, grüne und blaue Subpixel [Jö09].

Farbtiefe

Die Farbtiefe bestimmt die Abstufung (Quantisierung) innerhalb eines Farbkanals. Je mehr Bit der Kanal besitzt, desto mehr verschiedene Farben können dargestellt werden. In einem Binärbild, welches nur aus den Farben schwarz oder weiß besteht, ist nur 1 Bit notwendig, da nur 2 verschiedene Farben existieren. Ein Bild welches aus vielen verschiedenen Grauwerten besteht, benötigt deshalb mehr Bit um die Grautöne und Verläufe zwischen den Farben detailliert darzustellen.

2.3 Farbraum

Die Farbe eines Bildes wird durch den entsprechenden Farbraum bestimmt. Dieser wird durch die verschiedenen Farbkanäle wie rot, grün und blau erzeugt und kann durch einen 3-dimensionalen geometrischen Körper dargestellt werden, sodass jede Farbe einen einzigartigen Punkt in diesem Körper erhält.

2.3.1 RGB

Der RGB Farbraum ist vom Prinzip her wie das menschliche Auge aufgebaut und besteht aus den 3 Farbkanälen Rot, Grün und Blau, welche den Zapfen des Auges nachempfunden sind. Jeder Kanal besteht aus 8 Bit, somit sind 256 verschiedene Werte Pro Kanal darstellbar was in einer Menge aus 16.777.216 Farben resultiert. Wobei schwarz mit den Werten 0,0,0 und weiß mit 255,255,255 dargestellt wird. Durch additives Hinzufügen der Kanäle können so alle verschiedenen Farben erreicht werden. Bei dem gleichen Wert aller Kanäle, erhält man die Graustufen von schwarz bis weiß.

In Abbildung 2.4 ist eine mögliche Darstellung des RGB Farbraums zu sehen. Jeder Kanal wird auf eine Achse des Koordinatensystems verteilt, welche Werte von 0 bis 255 annehmen kann. Der Ursprung, welcher schwarz entspricht, ist im hinteren Bereich unten links zu sehen und weiß im Vordergrund oben rechts. Durch die Mischung der Kanäle können so bis zu 16.7 Millionen verschiedene Farben erzeugt werden. Grauwerte liegen in den Diagonalen zwischen schwarz und weiß. Schön zu sehen ist die Gruppierung der Farben, jede nimmt einen bestimmten Platz des Würfels

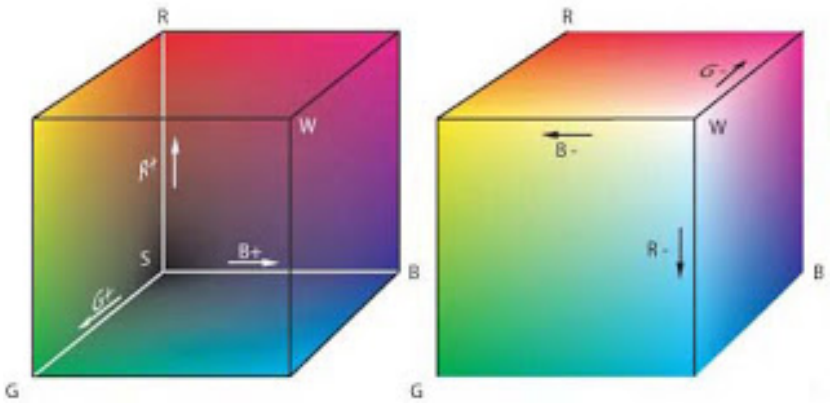


Abbildung 2.4: Räumliche Darstellung des RGB Farbraums [Fra]

ein, so dass beispielsweise Orange und alle Verläufe davon, ob hell oder dunkel, in einem Bereich zusammengefasst werden können. Dies hat den Vorteil dass bei späteren Berechnungen oder der Identifizierung einer bestimmten Farbe direkt ein Bereich festgelegt werden kann.

2.4 Kamera-Aufbau

Um die Tiefe einer Szene zu berechnen, reicht ein einziges Bild nicht aus. Man kann sich dies wie bei einer normalen Fotokamera vorstellen. Das zu fotografierende Objekt wird anvisiert und nach dem Abzug durch die Linse auf ein 2-dimensionales Bild übertragen. Durch diesen Schritt geht allerdings die komplette Tiefeninformation verloren, da jeder Punkt zwischen der Linse und dem Objekt, im Prinzip auch alle Objekte dahinter, auf ein und denselben Punkt abgebildet werden. Die logische Schlussfolgerung ist, eine zweite Kamera zu benutzen, die die Szene aus einem anderen Blickwinkel aufnimmt und so das Mehrdeutigkeitsproblem löst. Für die Berechnung der Tiefe werden also 2 Bilder der gleichen Szene benötigt, welche von verschiedenen Kameras zur gleichen Zeit aufgenommen werden und in einem bestimmten Abstand nebeneinander aufgestellt sind. In diesem Format gibt es 2 Varianten, einmal eine konvergierende Kameraanordnung, wobei die Kameras auf das Objekt geneigt sind und eine orthoparallele Anordnung, in welchem die Kameras den exakt gleichen Winkel zum Objekt haben und nur ein Unterschied durch die Position zustande kommt. Der Algorithmus dieser Arbeit geht von einer Aufnahme in orthoparallelen Anordnung aus, weshalb dieser nun etwas näher erläutert wird.

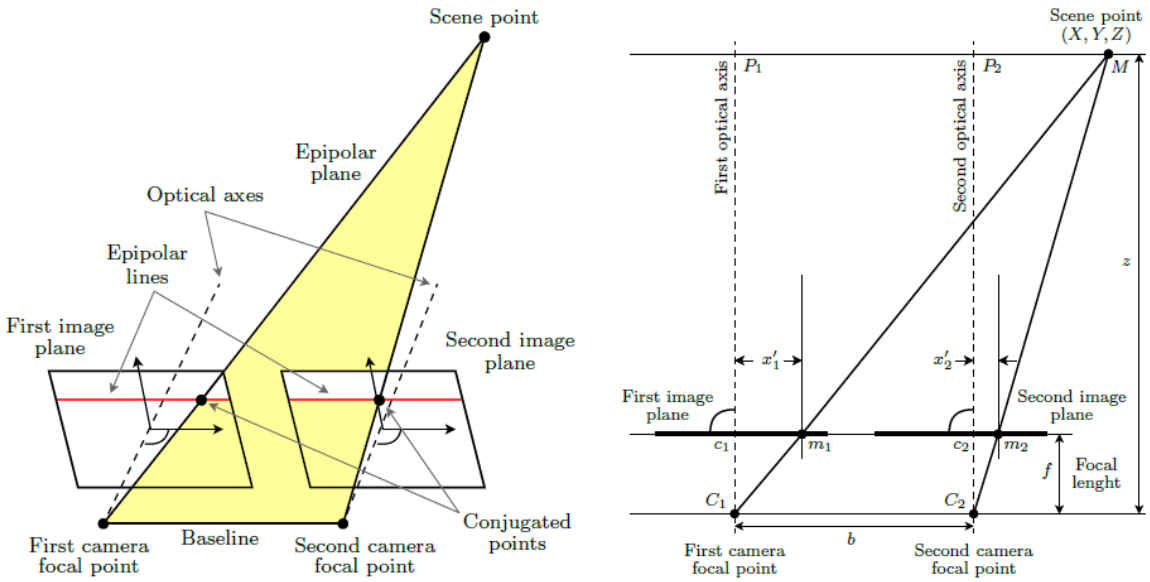


Abbildung 2.5: Prinzipieller Aufbau einer orthoparallelen Anordnung der Kameras [Bru13a]

Links ist nun ein gelbes Dreieck zu sehen, welches zwischen dem Szenenpunkt oben und den beiden Kameras unten aufgespannt wird. Gleich darüber liegen die jeweiligen Bildbereiche der Kameras, auf welche das spätere Objekt abgebildet wird und zwar an den Punkten, wo sich Bildebenen und die Linien des Dreiecks schneiden. Die roten Linien sind die sogenannten Epipolarlinien. Sie bilden den direkten Weg der benachbarten Kamera zum zu fotografierenden Objekt. Sie sind somit ein Abbild der Information, welche zwischen Szenenpunkt und der anderen Kamera stattfindet. Durch die parallele Anordnung sind hier die Epipolar Linien jedoch horizontal, was auch völlig Sinn macht, da sich Kamera 1 niemals in dem Blickfeld von Kamera 2 befindet oder umgekehrt und sie sich somit im unendlichen befinden. Dadurch kann eine Verschiebung der Szene nur auf diesen beiden Linien stattfinden, weshalb der vertikale Fall außen vor gelassen werden kann, was im späteren Verlauf die mathematische Berechnung ein wenig erleichtert.

Zunächst wird anhand eines kleines Beispiels gezeigt, wieso eine Verschiebung nur dort stattfinden kann. Dies kann mittels Strahlensatzes gezeigt werden. Betrachtet man das rechte Bild, lassen sich bei genauerem Blick vier Dreiecke erkennen, jeweils zwei Große und zwei Kleine. Zum einen das Dreieck zwischen dem Szenenpunkt M, der Kamera C_1 und dem Punkt P_1, wobei die Linie zwischen C_1 und M für die Entfernung der ersten Kamera zum Objekt steht. In diesem integriert ist nun ein kleineres Dreieck, welches durch die Punkte C_1, c_1 und m_1 definiert ist, wobei der Abstand C_1 und c_1 die Brennweite ist. Anhand diesen beiden Dreiecken lässt sich erkennen, dass sie im selben Verhältnis zueinander stehen.

Dieses Verhalten lässt sich durch den Strahlensatz mittels folgender Gleichung ausdrücken:

$$\frac{x}{z} = \frac{x_1^{'}}{f} \tag{2.22}$$

Exakt das Gleiche gilt für die zweite Kamera. Auch hier gibt es ein großes und ein kleines Dreieck, nämlich an den Punkten M, P_2, C_2 und m_2, c_2, C_2. Dies kann wieder mittels Strahlensatz beschrieben werden:

$$\frac{x - b}{z} = \frac{x_2^{'}}{f} \tag{2.23}$$

Stellt man diese zwei Gleichungen nun nach x um und setzt sie ein, erhält man:

$$z = \frac{b \cdot f}{x_1^{'} - x_2^{'}} \tag{2.24}$$

Ist der Abstand b der beiden Kameras, sowie die Brennweite bekannt, ergibt der Abstand der Punkte $x_1^{'}$ und $x_2^{'}$ die gewünschte Tiefe z. Somit lässt sich alleine durch die Länge der Verschiebung, die Tiefe rekonstruieren, wobei die Tiefe invers proportional zur Länge der Verschiebung ist.

3 Optischer Fluss

Der Optische Fluss ist in der Computer Vision ein weit verbreitetes und tief erforschtes Gebiet, da er in vielen Bereichen seine Anwendung findet. Von der Komprimierung von Videodateien, über Sicherheitssysteme der Automobilindustrie, bis hin zur Robotik. Alle nutzen die Vorteile der Bildverarbeitungstechnik. Der Optische Fluss beschreibt auf den ersten Blick eine für den Menschen recht einfache Aufgabe, nämlich das Wiederfinden von bestimmten Positionen in einer Bilderfolge. Genauer gesagt wird versucht, jeden Pixel aus einem Bild in einem zweiten Bild wiederzufinden, welches sich allerdings durch Zeit, Bewegung oder einen anderen Blickwinkel an einem anderen Ort befindet. Was für den Menschen jedoch allgegenwärtig ist, ist für den PC nicht ganz so einfach zu bewerkstelligen, da der Computer aus den Bildern alleine, keinerlei Beziehung zwischen verschiedenen Pixeln oder Objekten herstellen kann.

Um dies zu ändern wurden Ansätze zur Berechnung entwickelt. Zum einen lokale Ansätze, welche vom Ausgangspixel, jeden anderen Pixel in einer gewissen Nachbarschaft absuchen und miteinander vergleichen. Unterschieden wird hier in diskrete Modelle wie dem Blockmatching, welches eine Region absucht und kontinuierliche Modelle wie dem Lucas Kanade Verfahren [LK81], welches zwar auch nur eine gewissen Bereich abdeckt, die Lösung aber explizit berechnet. Beide Varianten hatten jedoch gewisse Nachteile, auf welche später noch etwas näher eingegangen wird, welche erst durch die globalen Methoden elegant gelöst wurden.

Dies war schließlich 1981 der Fall als die Horn und Schunck Methode [HS81] entwickelt wurde, welche die Grundlage für alle zukünftigen Methoden darstellt und Grundlage dieser Arbeit ist. Zugehörige Pixel wurden nun nicht mehr in einer Nachbarschaft berechnet, sondern global bestimmt.

3.1 Grundlagen

Zur Berechnung des Optischen Flusses werden Constraints benötigt, mit welchen das bisherige Wissen in den Ansatz einfließt. Dies sind kleine Gleichungen und Funktionen, welche das Vorhaben beschreiben. Zusätzlich folgt eine Erklärung zur Interpretation des Ergebnisses.

3.1.1 Grauwertkonstanz

Der erste Constraint ist die Grauwertkonstanz (BCCE), sie ist eine Gleichung, welche die Beziehung zwischen dem Grauwert eines Pixels aus Bild 1 mit dem Grauwert eines anderen Pixels aus Bild 2 vergleicht. Bei einem kurzen zeitlichen Intervall oder im Stereofall, nur durch eine andere Perspektive,

kann davon ausgegangen werden, dass sich der Grauwert hinsichtlich Reflexion, Schattierung oder Beleuchtung gar nicht oder nur sehr leicht ändert. Die BCCE sieht demnach wie folgt aus:

$$f(x+u, y+v, t+1) - f(x, y, t) = 0 \tag{3.1}$$

Die Gleichung besteht aus einer Funktion f, welche zu einer gegebenen Koordinate x und y und Zeitpunkt t den Grauwert des Pixels berechnet. Für das zweite Bild, befindet sich dieser Pixel an einer anderen Stelle, nämlich an einer in x-Richtung um u und in y-Richtung um v verschobenen Stelle, wobei u und v keine Variablen sondern Funktionen der Form $u(x, y)$ und $v(x, y)$ sind, welche die Verschiebung im Punkt x, y berechnen. $t + 1$ bedeutet, dass es sich hierbei um das direkt folgende Bild in einer Bilderfolge handelt.

3.1.2 Glattheitsannahme

Ein weiterer Constraint, welcher diese Methode erst von den lokalen Varianten abhebt, ist die Glattheitsannahme. Die Glattheit sorgt für einen gleichmäßigen Fluss benachbarter Pixel, wie es beispielsweise bei einer Wand oder gleichfarbigen Fläche auftritt. Berechnet wird sie allgemein durch:

$$|\nabla u|^2 + |\nabla v|^2 = 0 \tag{3.2}$$

Jeder Pixel hat im späteren Fluss einen Wert u und v, welcher die Verschiebung des Pixels beschreibt. Berechnet man von dieser Map nun den Betrag des Gradienten (Siehe Abschnitt 2.1.4), erhält man die Steigung des Flusses zu jedem Pixel. Da eine Fläche oder ein Objekt in der Regel in die gleiche Richtung verschoben wird, also zwei benachbarte Pixel nach der Verschiebung im Optimalfall wieder benachbart sein sollten, haben beide Pixel den gleichen Fluss, was im Gradienten dem Wert 0 entspricht. Verdeutlicht wird dies ebenfalls in Abbildung 3.1.

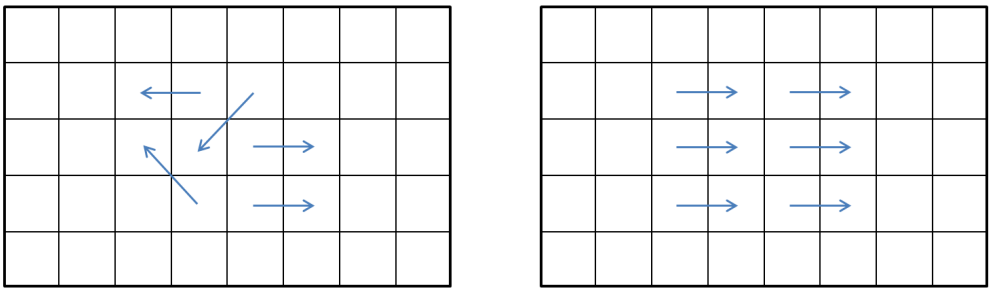


Abbildung 3.1: Links: Flussfeld ohne Glattheit, benachbarte Pixel verschieben sich kreuz und quer. **Rechts:** Flussfeld mit Glattheit, benachbarte Pixel sind nach der Verschiebung wieder nebeneinander

Natürlich sollte dies nur an Stellen beachtet werden, welche zum gleichen Objekt gehören und nicht an Kanten, welche unterschiedliche Objekte markieren.

3.1.3 Vektordarstellung

Um die berechneten Flussfelder darzustellen, haben sich mehrere Varianten etabliert. Zum einen ist es möglich die Verschiebungen in ein Vektorfeld zu zeichnen. So ist es sehr einfach den Ursprung, sowie die Position im zweiten Bild zu erkennen, wie in Abbildung 3.2 zu sehen ist. Es gibt allerdings einen großen Nachteil dieser Variante, nämlich dass die Vektoren viel Platz benötigen und so mehrere andere Pixel überdecken. Würde jeder Pixel einen Vektor bekommen, wäre das Ergebnis demnach komplett schwarz und nicht zu gebrauchen.

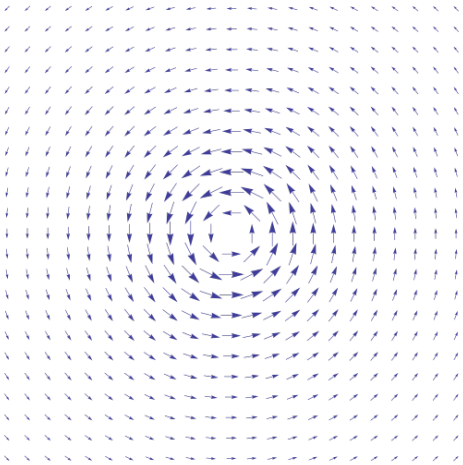


Abbildung 3.2: Bild eines Vektorfelds. Jeder Vektor steht für die Verschiebung eines Pixels. Gut zu erkennen ist die Richtung, jedoch auch der große Abstand zwischen den Vektoren [All08].

3.1.4 Farbdarstellung

Eine weitere Methode wäre die Darstellung mittels Farbkreis. Hierzu wird ein Template eines Farbkreises benötigt, welcher als Kodierung der Richtung sowie der Länge des Vektors dient. Die Vorteile dieser Methode liegen klar auf der Hand, denn es werden keine Vektoren eingezeichnet, sondern jedem Pixel wird eine Farbe zugeordnet, welche vom Mittelpunkt des Kreises aus gesehen dem eigentlich Vektor entspricht. So ist die Verschiebung jedes einzelnen Pixels zu erkennen. Hinzu kommt, dass die Länge des Vektors gleich mitkodiert wird, nämlich durch die Intensität der Farbe. Schwarz steht für keine Verschiebung und ist in der Mitte des Kreises platziert, je weiter man sich jedoch dem Rand nähert, also je länger der Vektor ist, desto kräftiger wird die entsprechende Farbe.

Wie in Abbildung 3.3 zu erkennen ist, wird im Gegensatz zum Vektorfeld jeder Pixel mit einer Verschiebung versehen und ein Bild erzeugt, welches dem originalen Bild recht ähnlich sieht. Man

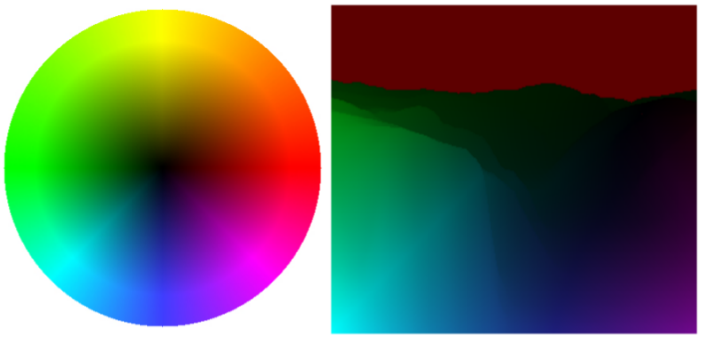


Abbildung 3.3: Links: Template des Farbkreises. **Rechts:** Yosemite Testsequenz mit eingefärbter Verschiebung [Bru13b].

sieht sofort welche Regionen in welche Richtung verschoben wurden, jedoch hat diese Methode nicht nur Vorteile, denn anhand der Farbe ist es nicht ersichtlich, wo der eigentliche Pixel nach der Verschiebung nun platziert ist. Es ist nur möglich die ungefähre Position in Relation zu den anderen Pixeln zu bestimmen. In dieser Arbeit fällt die Wahl allerdings trotzdem auf die Farbdarstellung.

3.2 Ansatz von Horn und Schunck

Der Ansatz, der nun für die Berechnung des Optischen Flusses erstellt werden soll, besteht aus den Constraints, welche weiter oben besprochen wurden. Diese werden in das Funktional (siehe Abschnitt 2.1.9) eingesetzt und quadratische Abweichungen werden über den gesamten Bildbereich integriert. Die Lösung ist dann der Fluss, der die Abweichung minimiert. Das allgemeine Optische-Fluss-Funktional, welches noch ein wenig modifiziert muss, jedoch zum besseren Verständnis erklärt wird, sieht folgendermaßen aus:

$$E(u,v) = \int_\Omega \underbrace{(f(x+u,y+v,t+1) - f(x,y,t))^2}_{Datenterm} + \alpha \underbrace{(|\nabla u|^2 + |\nabla v|^2)}_{Glattheitsterm} dxdy \tag{3.3}$$

wobei $\int_\Omega$ für das bestimmte Integral über den Bildbereich Ω steht. Es wird also nun versucht für das Energiefunktional E das kleinstmögliche Volumen zu finden, welches von E und den Achsen des Koordinatensystems eingeschlossen wird. Was wiederum der kleinsten Energie und der besten allgemeinen Lösung für alle Pixel des Bildes entspricht. Erzielt wird dies durch den Datenterm, welcher Grauwerte miteinander vergleicht und ihnen eine Energie zuordnet. Je größer der Unterschied der beiden Werte desto größer oder auch schlechter die Energie. Da jedoch nicht bekannt ist, welcher der beiden Grauwerte der Größere ist und es deshalb vorkommen kann, dass man ein negatives Ergebnis des Datenterms erhält, wird dieser noch quadriert um sicher zu stellen, dass zur Energie nur hinzugefügt und nichts abgezogen wird. Im hinteren Teil steht der bereits bekannte Glattheitsterm, welcher allerdings mit α multipliziert wird. Der Parameter α steht für die Gewichtung der beiden

Terme zur Energie. Ein kleiner Wert für α legt den Fokus auf den Datenterm, es wird hauptsächlich versucht gleiche Pixel zu finden und die Glattheit wird eher vernachlässigt, da ein großer Unterschied der Farbe direkt in einer großen Energie resultieren würde. Ein hoher Wert für αlegt den Fokus dagegen auf ein gleichmäßiges Resultat.

Anhand dieser Grundlage kann das Funktional auf das Problem angepasst werden. Zum einen verändern sich Datenterm und Glattheitsterm in der Hinsicht, dass aufgrund des Aufbaus (siehe Abschnitt 2.4), mit welchem die Fotos gemacht wurden, keine Verschiebung in y-Richtung möglich ist und deshalb dieser Fall auch nicht beachtet werden muss. Zum anderen werden in dieser Arbeit keine Grauwertbilder verwendet, sondern es dienen Farbbilder als Quelle (siehe Abschnitt 2.3.1). Wird das allgemeine Optische-Fluss-Funktional nun auf die Änderungen angepasst, entsteht:

$$E(u) = \int_\Omega \sum_{i=1}^{3} (f_i(x+u, y, t+1) - f_i(x, y, t))^2 + \alpha(|\nabla u|^2) dxdy \tag{3.4}$$

Was sofort auffällt ist, dass alle Argumente mit v verschwunden sind und sich der Datenterm ein wenig geändert hat. Da nun Farbbilder betrachtet werden und nicht mehr Graubilder wie bisher, haben sich auch die Anzahl der Kanäle geändert. $\sum_{i=1}^{3}$ bedeutet, dass der Farbwert des Bildes in jedem Kanal einzeln verglichen, aufsummiert und in ein gemeinsames Flussfeld übertragen wird.

3.3 Aperturproblem

Das Aperturproblem ist ein häufig auftretendes Problem der optischen Bildverarbeitung. Es beschreibt die Problematik zweier Pixel, welche nicht eindeutig identifizierbar sind. Um einen Pixel mit seinem Äquivalent zu verbinden werden Informationen benötigt, welche ihn von seiner Umgebung unterscheiden.

In Abbildung 3.4 sind 2 Rechtecke zu sehen, ein schwarzes, welches Bild 1 entspricht und ein graues, welches Bild 2 zugeordnet wird. Die kleinen roten Quadrate stehen für die Bereiche, welche vom Optischen Fluss betrachtet werden. Sieht man sich nun die einzelnen Bereiche etwas genauer an, lässt sich erkennen, dass es bei Quadrat 1 und 2 Schwierigkeiten gibt. Der Algorithmus versucht die Pixel der schwarzen Linie mit denen der grauen zu verbinden. Da jedoch in Ausschnitt 1 nicht ersichtlich ist, ob eine Verschiebung nach rechts oder nach links stattgefunden hat, kann hier keine Lösung berechnet werden. Gleiches gilt für das zweite Quadrat, bei der Verschiebung in y-Richtung. Quadrat 3 hingegen zeigt eine Ecke des Rechtecks, somit sind Informationen in x- und y-Richtung vorhanden und es kann deshalb exakt wiedergefunden werden.

Was in den lokalen Methoden noch zu Problemen geführt hat, wird in der Horn und Schunck Methode durch den Glattheitsterm gelöst. Sollte der Datenterm keine Lösung liefern, übernimmt der Glattheitsterm, sodass zumindest die Glattheit erfüllt ist. Die kleinste mögliche Energie wäre demnach eine glatte Fläche mit Gradient = 0. Dies geschieht durch Propagation bekannter Pixel auf nicht berechnete Nachbarn, bis alle Bereiche einen Fluss zugeordnet bekommen. Somit könnte alleine durch Quadrat 3 und die Propagation auf die anderen Bereiche der gesamte Fluss des Bildes berechnet werden.

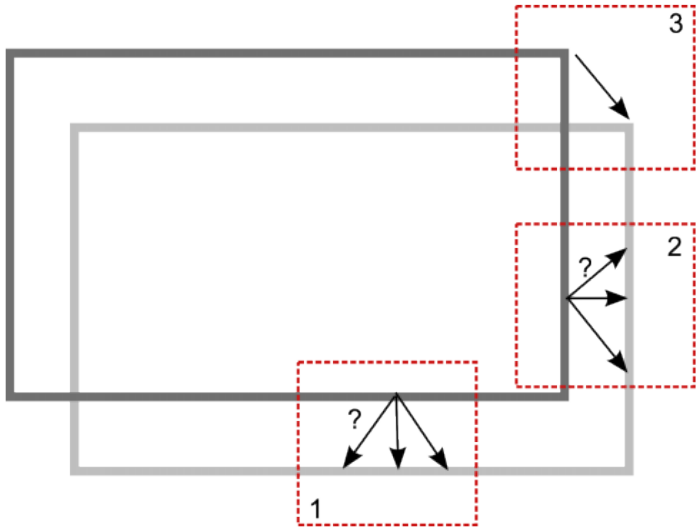


Abbildung 3.4: Zu sehen sind zwei Rechtecke, einmal zum Zeitpunkt t und t+1. Bei Quadrat 1 und 2 entsteht das Aperturproblem, Quadrat 3 hingegen kann berechnet werden und zeigt den wahren Fluss des Rechtecks. [Ape]

3.4 Minimierung

Um die Lösung des Funktionals zu berechnen, muss dies minimiert werden, da nur die allgemein kleinste Energie für den Fluss relevant ist. Dies ist vergleichbar mit der Minimierung normaler Funktionen. Die Funktion wird abgeleitet, gleich 0 gesetzt und anschließend nach dem globalen Minimum gesucht. Da es sich aber um eine Energiefunktional handelt, kann dies nicht 1 zu 1 übertragen werden. Hier fließt die Variationsrechnung mit ein, welche sich mit der Minimierung genau solcher Funktionale beschäftigt. Der wohl bekannteste Begriff der Variationsrechnung ist die Euler-Lagrange-Gleichung.

3.5 Euler-Lagrange-Gleichung

Zuerst muss die Form des Funktionals identifiziert werden. Bei der typischen Form des Optischen Flusses besteht es aus einer Funktion F, welche eine weitere Funktion u beinhaltet, wobei u differenzierbar ist, u' was für die Ableitung von u steht, und x, y was den Extremwert des Funktionals darstellt.

Das Funktional hat also die Form:

$$E(u) = \int_{\Omega} F(x, y, u, u_x, u_y) dxdy \tag{3.5}$$

Für diese Art des Funktionals existiert ein Euler-Lagrange-Framework, welche alle Funktionale der obigen Form ableitet:

$$F_u - \frac{d}{dx}F_{u_x} - \frac{d}{dy}F_{u_y} = 0 \tag{3.6}$$

wobei F_u für die Ableitung von F nach u steht und F_{u_x} und F_{u_y} die Ableitungen nach u_x sowie u_y sind. Wird dieses Framework jetzt auf das Funktional angewendet, erhält man die fertige Euler-Lagrange-Gleichung.

Als erstes kann durch Umformulierung des Glattheitsterms der Gradient auseinander gezogen werden und man erhält:

$$F = \sum_{i=1}^{3}(f_i(x+u,y,t+1) - f_i(x,y,t))^2 + \alpha(u_x^2 + u_y^2)dxdy \tag{3.7}$$

Danach werden die benötigten Komponenten des Frameworks berechnet, wobei u_x und u_y hier als eigenständige Variablen gewertet werden.

$$F_u = 2\sum_{i=1}^{3}(f_i(x+u,y,t+1) - f_i(x,y,t)) \cdot f_{ix}(x+u,y,t+1) \tag{3.8}$$

$$F_{u_x} = 2\alpha u_x \qquad\qquad F_{u_y} = 2\alpha u_y \tag{3.9}$$

und eingesetzt

$$0 = 2\sum_{i=1}^{3} f_{ix}(x+u,y,t+1) \cdot (f_i(x+u,y,t+1) - f_i(x,y,t)) - \frac{d}{dx}2\alpha u_x - \frac{d}{dy}2\alpha u_y \tag{3.10}$$

durch Umformulierung und Berechnung der partiellen Ableitungen erhält man schließlich:

$$0 = \sum_{i=1}^{3} f_{ix}(x+u,y,t+1) \cdot (f_i(x+u,y,t+1) - f_i(x,y,t)) - \alpha \underbrace{(u_{xx} + u_{yy})}_{\Delta u} \tag{3.11}$$

Nun kann der Glattheitsterm wieder zusammengefasst werden, was dem Laplace Operator (siehe Abschnitt 2.1.8) von u entspricht.

3.5.1 Neumann Randbedingung

Da Bilder nicht unendlich sind, sondern nur in einem bestimmten Bereichen definiert sind, ist nicht klar was genau an den Rändern der Funktion passiert. Genau um dieses Problem kümmern sich die Neumann Randbedingungen, welche Bestandteil der Euler-Lagrange-Gleichungen. Sie entscheidet, welchen Wert die Ableitung der Differentialgleichung an den Rändern annimmt. Definiert ist sie durch:

$$n^T \begin{pmatrix} F_{u_x} \\ F_{u_y} \end{pmatrix} = 0 \tag{3.12}$$

Hierbei steht n^T für die transponierte Normale und F_{u_x}, F_{u_y} für die bereits bekannten Ableitungen. Jede Differentialgleichung hat also ihre eigenen individuellen Randbedingungen, welche die Werte der Ableitung in Richtung der Normale bestimmen. Für dieses Funktional sieht sie folgendermaßen aus:

$$n^T \begin{pmatrix} 2\alpha u_x \\ 2\alpha u_y \end{pmatrix} = n^T \nabla u = 0 \tag{3.13}$$

wobei 2α herausgezogen werden kann und durch die Division weg fällt. Einfach gesagt, werden die Pixel der Ränder gespiegelt, sodass ein zusätzlich eingefügter Rahmen entsteht, welcher die gleichen Werte annimmt wie die Randpixel. So entsteht eine Ableitung welche in Richtung der Normale einen Wert von 0 hat.

3.6 Warping

Im herkömmlichen Optischen Fluss Verfahren wird normalerweise direkt die Grauwertkonstanz-Annahme linearisiert (siehe Abschnitt 2.1.12). Dies ist nötig, da nur durch die linearisierte Funktion und der quadratischen Bestrafung, nach der Ableitung eine strikt konvexe Funktion entsteht. Dies führt allerdings dazu, dass größere Verschiebungen nicht mehr berechnet werden können, da eine lineare Funktion nur innerhalb eines bestimmten Abstandes um den Entwicklungspunkt brauchbare Ergebnisse liefert.

In Abbildung 3.5 wird dieser Fall etwas verdeutlicht. Zu sehen sind 2 Funktionen, $-x^3 + x^2$ in rot und die lineare Annäherung in grün. In dem Intervall von -2 bis 2 ist die Approximation noch in einer bestimmten Nähe, welche der Funktion im Groben recht nahe kommt. Macht man jedoch einen etwas größeren Sprung in eine Richtung, so haben die beiden Funktionen nicht mehr viel miteinander gemein.

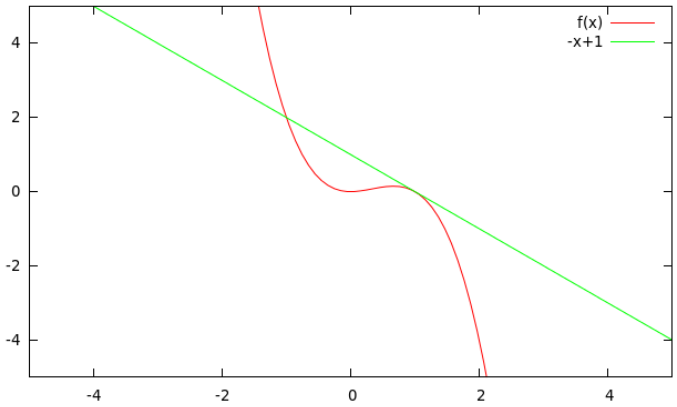


Abbildung 3.5: Rot: $-x^3 + x^2$, **Grün:** Linearisierte Funktion um Entwicklungspunkt 1

Mit dem Warping wird der Algorithmus nun so abgeändert, dass er sich iterativ der Lösung durch viele kleinere Verschiebungen annähert und so das Problem der großen Sprünge umgangen wird. Zuerst muss allerdings das Funktional angepasst werden, beginnend mit der Ursprünglichen Gleichung:

$$0 = \sum_{i=1}^{3} f_{ix}(x+u, y, t+1) \cdot (f_i(x+u, y, t+1) - f_i(x, y, t)) - \alpha \Delta u \tag{3.14}$$

Zuerst wird eine Fixpunktiteration eingefügt, welche die gesuchte Lösung u schrittweise berechnet. Dazu muss das Funktional in zwei verschiedene Zeitschritte unterteilt werden, sodass mit einem Startwert von u, einen neues u berechnet werden kann und sich so dem richtigen Wert immer mehr angenähert wird:

$$0 = \sum_{i=1}^{3} f_{ix}(x+u^k, y, t+1) \cdot (f_i(x+u^{k+1}, y, t+1) - f_i(x, y, t)) - \alpha \Delta u^{k+1} \tag{3.15}$$

Nun folgt die Einführung eines Inkrementes, in welchem die Lösung in mehrere kleine Verschiebungen zerlegt wird. Um u^{k+1} zu berechnen, wird es mithilfe des Inkrements du^k und der vorigen Lösung u^k ausgewählt:

$$u^{k+1} = u^k + du^k \tag{3.16}$$

und eingesetzt:

$$0 = \sum_{i=1}^{3} f_{ix}(x+u^k, y, t+1) \cdot (f_i(x+(u^k+du^k), y, t+1) - f_i(x, y, t)) - \alpha \Delta (u^k + du^k) \tag{3.17}$$

wobei u^k für den bisherigen Fluss aus den alten Zeitschritten steht und du^k für das Inkrement, welches bei jeder neuen Iteration dazu addiert wird. Da durch das Warping jedoch die Linearisierung nur hinausgezögert wurde und nicht gänzlich umgangen werden kann, ist dies der nächste Schritt.

Zuerst wird die Taylorreihe (siehe Abschnitt 2.1.11) auf die entsprechende Form für Vektoren angepasst:

$$f(x) \approx f(a) + (x-a)^T \nabla_3 f(a) \tag{3.18}$$

und anschließend um den Punkt $a = (x + u^k, y, t + 1)$ nach du linearisiert:

$$f_i(x+(u^k+du^k), y, t+1) \approx f_i(x+u^k, y, t+1) + \begin{pmatrix} x+u^k+du^k-(x+u^k) \\ y-y \\ t+1-(t+1) \end{pmatrix}^T \begin{pmatrix} f_{ix}(x+u^k, y, t+1) \\ f_{iy}(x+u^k, y, t+1) \\ f_{it}(x+u^k, y, t+1) \end{pmatrix} \tag{3.19}$$

was folgendem entspricht:

$$f_i(x+u^k, y, t+1) + \begin{pmatrix} du^k \\ 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} f_{ix}(x+u^k, y, t+1) \\ f_{iy}(x+u^k, y, t+1) \\ f_{it}(x+u^k, y, t+1) \end{pmatrix} \tag{3.20}$$

Werden die Vektoren nun ausmultipliziert erhält man:

$$f_i(x+u^k, y, t+1) + f_{ix}(x+u^k, y, t+1)du^k \tag{3.21}$$

Danach wird die linearisierte Version wieder in die Gleichung eingesetzt und ausmultipliziert:

$$0 = \sum_{i=1}^{3} f_{ix}(x+u^k, y, t+1)(f_{ix}(x+u^k, y, t+1)du^k + \underbrace{f_i(x+u^k, y, t+1) - f_i(x, y, t))}_{\approx f_{it}} - \alpha\Delta(u^k+du^k) \tag{3.22}$$

$$0 = \sum_{i=1}^{3} f_{ix}(x+u^k, y, t+1)^2 du^k + f_{ix}(x+u^k, y, t+1) \cdot (f_{it}(x+u^k, y, t+1) - \alpha\Delta u^k - \alpha\Delta du^k \tag{3.23}$$

Zum Schluss wird noch in die Bewegungstensor-Notation umgeformt:

$$0 = \sum_{i=1}^{3} J_{11i} du^k + J_{12i} - \alpha \Delta u^k - \alpha \Delta du^k \tag{3.24}$$

Hierbei steht J_{11} und J_{12} für den jeweils ersten und zweiten Eintrag der Matrix $J = \begin{pmatrix} f_{ix}^2 & f_{ix} f_{it} \\ f_{it} f_{ix} & f_{it}^2 \end{pmatrix}$ welche auch als Motion Tensor bekannt ist.

Zur besseren Übersicht kann die Summe auch in den Bewegungstensor gezogen werden:

$$0 = J_{11} du^k + J_{12} - \alpha \Delta u^k - \alpha \Delta du^k \tag{3.25}$$

mit $J = \begin{pmatrix} \sum_{i=1}^{3} f_{ix}^2 & \sum_{i=1}^{3} f_{ix} f_{it} \\ \sum_{i=1}^{3} f_{it} f_{ix} & \sum_{i=1}^{3} f_{it}^2 \end{pmatrix}$

3.6.1 Hierarchische Minimierung

Diese Fixpunktiteration wird nun in ein Coarse-to-Fine Schema eingefügt, welches bei der Lösung der größeren Verschiebungen hilft. Durch das Weglassen der Linearisierung im Modell ändert sich das Funktional von einem konvexen zu einem nicht-konvexen Funktional und es existiert deshalb mehr als ein Minimum. Als Folge dessen wird die Auflösung des Bildes schrittweise gesenkt, wodurch Details verschwinden und eine simplere Variante des Bildes entsteht. Oft wird dieser Vorgang auch Coarse-to-Fine-Pyramide genannt, da jedes Level eine Ebene einer Pyramide darstellt, auf welcher die Nächste aufbaut, wobei die unterste Ebene der gröbsten Auflösung entspricht.

Das Ziel ist nun, der Fixpunktiteration von ihrem Startpunkt aus, den Weg zu dem globalen Minimum zu ebnen. In der originalen Auflösung existieren viele lokale Minima, sodass es sehr wahrscheinlich ist, dass die Iteration in ein lokales anstatt des globalen Minimums konvergiert, was in Abbildung 3.6 an dem roten Pfeil zu erkennen ist und wiederum in einer falschen Verschiebung resultieren würde.

Das Schema beginnt also mit einer Initialisierung des Flusses u, welcher auf 0 gesetzt wird. Nun werden die verschiedenen Ebenen erstellt und mit der gröbsten begonnen. Als nächstes folgt die Berechnung des Inkrements du, was dem globalen Minimum dieses Levels entspricht und anschließend auf den bisherigen Gesamtfluss u addiert wird. Dieser Wert dient nun als Startwert für das nächst feinere Level. Der Vorgang wird bis zur originalen Auflösung wiederholt, bis dort schließlich der finale Fluss berechnet wird.

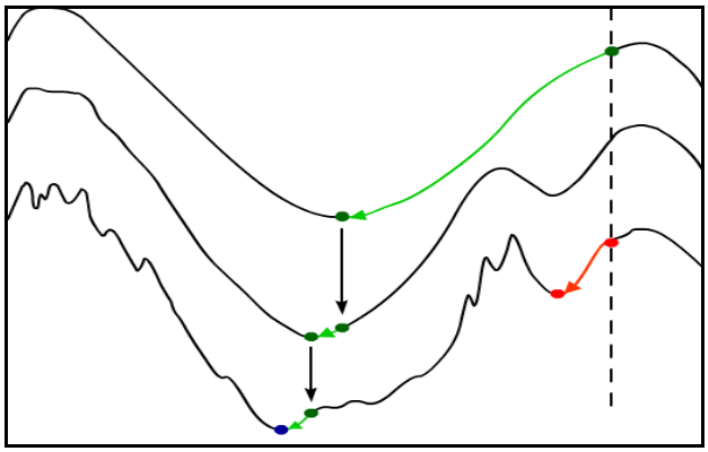


Abbildung 3.6: Coarse to Fine Pyramide mit drei Leveln. **Unten:** Originales Bild. Zu sehen sind viele lokale Minima, in welche die Fixpunktiteration stecken bleiben kann. **Mitte:** Herunterskalierte Version des Bildes. Viele Minima sind bereits verschwunden. **Oben:** Gröbste Version. Alle Minima bis auf eines sind verschwunden. Die optimale Schätzung dieser Ebene ist nun möglich [Bru13b].

3.6.2 Rückwärtsregistrierung

Der nächste Schritt ist nun das eigentliche Warping des Bildes. Nachdem der Fluss des aktuellen Levels berechnet wurde, muss er anschließend auf die Auflösung des nächsten Levels hochskaliert werden. Hinzu kommt, dass das zweite Bild um genau diesen Fluss verschoben werden muss, sodass sich Bild 1 und Bild 2 theoretisch in genau diesem Bereich überschneiden würden. Dies hat den Grund, dass anschließend nur die kleinere Verschiebung berechnet werden soll, welche durch die Skalierung des Bildes auf die nächstfeinere Ebene entstanden ist. Ohne diesen Schritt, würde auf jedem Level der komplette Fluss noch einmal berechnet berechnet werden, was den Sinn des Inkrements zunichte macht.

Betrachtet man Abbildung 3.7 sind kleine Ausschnitte von Bild 1 und 2 zu sehen. Der interessante Teil ist nun das kleine Quadrat mit den Werten 10, welcher sich im 1. Bild links und im 2. Bild rechts befindet. Der aktuelle Fluss wäre demnach für $u = 2$. Als nächstes folgt das Warping von Bild 2. Realisiert wird dies durch die Summe der Position der Pixel und der Verschiebung u. Für all die Pixel, welche nach der Verschiebung außerhalb des Bildbereichs landen würden, erhält das verschobene Bild $f2\_w$ die Werte aus Bild 1, somit gilt $f2\_w_{i,j} = f1\_w_{i,j}$. Und für alle anderen Pixel, welche innerhalb des Bildbereichs sind, gilt $f2\_w_{i,j} = f2\_w_{i,j} + u$.

Der gesamte Vorgang des Coarse to Fine Warpings sieht also wie folgt aus:

1. Level der Coarse to Fine Pyramide erstellen.
2. Startwert u mit 0 initialisieren und mit gröbstem Level beginnen.
3. Für jedes Level:

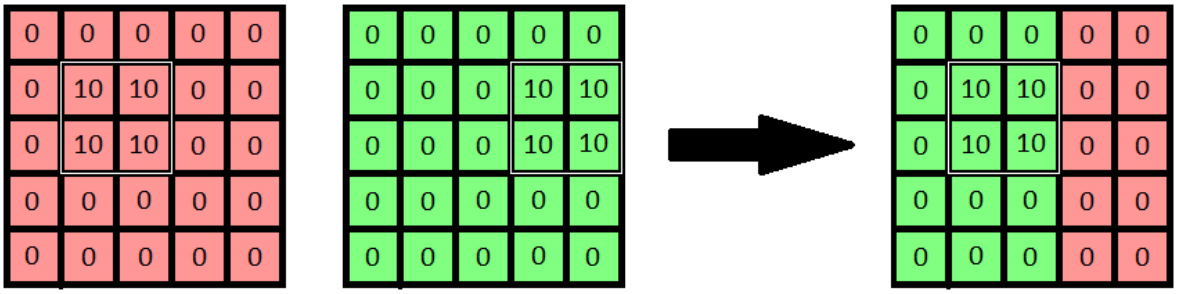


Abbildung 3.7: Backward Registration. **Links:** Ausschnitt aus Bild 1. **Mitte:** Ausschnitt aus Bild 2 mit verschobenem Quadrat um $u = 2$. **Rechts:** Zusammenstellung von gewarptem Bild f2_w. Pixel welche aus dem Bild gewarped werden, kommen aus Bild 1, die restlichen aus Bild 2.

   a) Inkrement du berechnen.

   b) Inkrement du auf Gesamtfluss u addieren.

   c) Fluss auf nächsthöheres Level skalieren.

   d) Zweites Bild durch Rückwärtsregistrierung um du warpen.

4. Schritt 3 wiederholen bis originales Level erreicht ist.
5. Fluss u ist berechnet.

3.7 Diskretisierung

Digitale Bilder (siehe Abschnitt 2.2.1) sind immer nur abschnittsweise definiert, nämlich an den jeweiligen Stellen der Pixel. Daher können die kontinuierlichen Funktionen dort nicht ohne weiteres angewendet werden. Diese müssen nun mithilfe finiter Differenzen diskretisiert werden. Dazu werden allgemein drei verschiedene Varianten verwendet.

Vorwärtsdifferenz:

$$u_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{h_x} \tag{3.26}$$

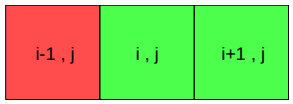


Abbildung 3.8: $u_{i,j}$ steht hier für den zu diskretisierenden Bereich an der Stelle i, j, welcher nun mit dem Differenzen-Quotienten approximiert wird. Dazu wird die Differenz des rechts neben ihm liegenden (vorwärts) und des zentralen Pixel berechnet und durch die Länge des Intervalls h_x geteilt, wobei x für die x-Richtung steht. Rechts daneben die Maske mit den relevanten Pixel in grün, sowie die unwichtigen in rot.

Rückwärtsdifferenz:

$$u_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{h_x} \tag{3.27}$$

Abbildung 3.9: Bei der Rückwärtsifference läuft der Vorgang genau gleich ab, mit dem Unterschied, dass nun der Pixel links daneben (rückwärts) mit einfließt. Rechts daneben die Maske mit den relevanten Pixel in grün, sowie die unwichtigen in rot.

Zentrale Differenz:

$$u_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} \tag{3.28}$$

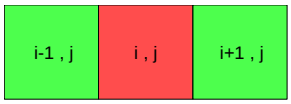


Abbildung 3.10: Die Zentrale Differenz ist im Prinzip eine Mischung aus beiden vorherigen Methoden, hier werden jeweils der linke und der rechte Pixel mit einbezogen, mit dem Unterschied dass nun die Länge des Intervalls doppelt so groß ist. Rechts daneben die Maske mit den relevanten Pixel in grün, sowie die unwichtigen in rot.

Benötigt wird hier allerdings nur die Zentrale Differenz, die die genauste Approximation liefert. Als Ausgangspunkt wird von der bereits bekannten Euler-Lagrange-Gleichung gestartet:

$$J_{11}du^k + J_{12} - \alpha\Delta u^k - \alpha\Delta du^k = 0 \tag{3.29}$$

Anhand des Funktionals lässt sich erkennen, dass zum einen der Bewegungstensor J, die Flussvariable du, sowie der Laplace von u und du diskretisiert werden muss, wobei das Vorgehen bei Δu und Δdu identisch ist und deshalb nur für Δdu veranschaulicht wird.

Diskretisierung von du:

$$du_{i,j} = du(i \cdot h_x, j \cdot h_y) \tag{3.30}$$

Die Diskretisierung von du erfordert nicht viel Aufwand und ist der trivialste Teil, denn die Funktion muss nur an das Gitter der Funktion angepasst werden, was mit der Multiplikation des Intervalls erreicht wird.

Diskretisierung von J:

Hier müssen einzeln alle Einträge der Matrix J diskretisiert werden. Da diese allerdings aus f_x und f_t bestehen, reicht es diese auszurechnen.

$$[f_x]_{i,j} = \frac{1}{2}\left(\frac{f1_{i+1,j} - f1_{i-1,j}}{2h_x} + \frac{f2_{i+1,j} - f2_{i-1,j}}{2h_x}\right) \tag{3.31}$$

wobei hier der Durchschnitt beider Zentraler Differenzen aus Bild $f1$ sowie $f2$ genommen wurde.

Die Zeit wird durch die Differenz beider Bilder errechnet.

$$[f_t]_{i,j} = (f2_{i,j} - f1_{i,j}) \tag{3.32}$$

Diskretisierung von Δdu:

Um den Laplace zu berechnen, wird dieser zuerst in eine andere Darstellung umgeformt.

$$\Delta du = (du_x)_x + (du_y)_y \tag{3.33}$$

Wie zu erkennen ist, bestehen die Terme jeweils aus der zweiten Ableitung von du, weshalb die Berechnung der Zentralen Differenz zwei mal ausgeführt werden muss. Aus diesem Grund wird ein Intervall von $\frac{1}{2}h_x$ festgelegt, um später wieder ein Intervall der Länge 1 zu erhalten, welche jeweils nur die benachbarten Pixel mit einfließen lässt.

$$\begin{aligned}
(du_x)_x + (du_y)_y &\approx \frac{(du_x)_{i+\frac{1}{2},j} - (du_x)_{i-\frac{1}{2},j}}{2(\frac{1}{2}h_x)} + \frac{(du_y)_{i,j+\frac{1}{2}} - (du_y)_{i,j-\frac{1}{2}}}{2(\frac{1}{2}h_y)} \\
&\approx \frac{\frac{du_{i+1,j}-du_{i,j}}{2(\frac{1}{2})h_x} - \frac{du_{i,j}-du_{i-1,j}}{2(\frac{1}{2})h_x}}{2(\frac{1}{2})h_x} + \frac{\frac{du_{i,j+1}-du_{i,j}}{2(\frac{1}{2})h_y} - \frac{du_{i,j}-du_{i,j-1}}{2(\frac{1}{2})h_y}}{2(\frac{1}{2})h_y} \\
&= \frac{du_{i+1,j} - du_{i,j}}{h_x^2} - \frac{du_{i,j} - du_{i-1,j}}{h_x^2} + \frac{du_{i,j+1} - du_{i,j}}{h_y^2} - \frac{du_{i,j} - du_{i,j-1}}{h_y^2} \\
&= \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{du_{\bar{i},\bar{j}} - du_{i,j}}{h_l^2}
\end{aligned} \tag{3.34}$$

Der Laplace ist somit die Summe der Differenzen der Nachbarn zu ihrem zentralen Pixel.

Werden nun alle Approximationen zusammengesetzt erhält man die diskrete Euler-Lagrange-Gleichung.

$$0 = [J_{11}]du + [J_{12}] - \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{u_{\bar{i},\bar{j}} - u_{i,j}}{h_l^2} - \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{du_{\bar{i},\bar{j}} - du_{i,j}}{h_l^2} \tag{3.35}$$

wobei $\sum_{l \in x,y}$ für die Richtung des Intevalls und $\sum_{(\bar{i},\bar{j}) \in N_l(i,j)}$ für die jeweiligen Nachbarn des Pixel i, j steht.

3.8 Lösung

Das Problem ist aufgestellt, nun muss es gelöst werden. Für jeden Pixel liegt eine Gleichung vor, welche alle gemeinsam und unter Beachtung der anderen Gleichungen berechnet werden müssen. Dies entspricht der Lösung eines linearen Gleichungssystems in der Größe der Auflösung der Bilder. Es gibt verschiedene Ansätze für Gleichungssysteme, doch einige sind nur bis zu einem gewissen Grad brauchbar und können bei dieser Größenordnung nicht mehr eingesetzt werden, da sie schlicht zu lange brauchen und zu viel Speicher benötigen. Ein Beispiel dafür wäre das Gauß'sche Eliminationsverfahren, welches eine Laufzeit von $O(n^3)$ hat, also die Laufzeit kubisch ansteigt mit der Anzahl der Zeilen. Aus diesem Grund werden in der Regel Iterationsverfahren verwendet, welche sich der Lösung langsam annähern und auf bereits berechneten Ergebnissen aufbauen.

Ein Beispiel dafür sind die Splitting-Verfahren, welche durch geschicktes Trennen der Matrix mit einem Anfangsvektor schrittweise zu dem Ergebnis konvergieren und nach ausreichender Genauigkeit abgebrochen werden.

Um von einem linearen Gleichungssystem der Form $Ax = b$ auf das gewünschte Iterationsschema zu kommen, muss die Matrix wie bereits erwähnt aufgeteilt werden:

$$A = A_1 + A_2 \tag{3.36}$$

Wird dies nun in die alte Form eingesetzt, kann die Gleichung nach dem Lösungsvektor x umgestellt werden.

$$(A_1 + A_2)x = b \rightarrow A_1x + A_2x = b \rightarrow A_1x = b - A_2x \rightarrow x = A_1^{-1}(b - A_2x) \tag{3.37}$$

wobei A_1^{-1} für die invertierte Matrix von A_1 steht.

Zum Schluss wird noch die Fixpunktiteration eingeführt:

$$x^{k+1} = A_1^{-1}(b - A_2x^k) \tag{3.38}$$

3.8.1 Jacobi-Methode

Angefangen wird mit der Jacobi Methode, welche allerdings nur als Grundlage dient und später noch durch einige Verbesserungen aufgewertet wird. Die Jacobi Methode trennt die Matrix A in zwei Teile auf, nämlich der Hauptdiagonale A_1 und den restlichen Einträgen A_2. Dies hat den Grund, dass A_1 invertiert werden muss und Diagonalmatrizen sehr einfach zu invertieren sind.

Um die nächsten Schritte besser zu veranschaulichen, ist es sinnvoll die Form des Gleichungssystems in geringer Größe etwas näher zu betrachten. Hierzu werden die Gleichungen umgeformt und alle

Komponenten, welche nichts mit du zu tun haben auf die rechte Seite gebracht. Das folgende Beispiel ist ein lineares Gleichungssystem, welches durch ein 2x2 Pixel großes Bild entstehen könnte:

(3.39)
$$\underbrace{\underbrace{\begin{pmatrix} J_{11} & & & \\ & J_{11} & & \\ & & J_{11} & \\ & & & J_{11} \end{pmatrix}}_{J_{11}} - \alpha \underbrace{\begin{pmatrix} -2 & 1 & 1 & \\ 1 & -2 & & 1 \\ 1 & & -2 & 1 \\ & 1 & 1 & -2 \end{pmatrix}}_{\Delta du} \underbrace{\begin{pmatrix} du \\ du \\ du \\ du \end{pmatrix}}_{Fluss}}_{A}_{x} = \underbrace{\underbrace{\begin{pmatrix} -J_{12} \\ -J_{12} \\ -J_{12} \\ -J_{12} \end{pmatrix}}_{J_{12}} + \alpha \underbrace{\begin{pmatrix} -2 & 1 & 1 & \\ 1 & -2 & & 1 \\ 1 & & -2 & 1 \\ & 1 & 1 & -2 \end{pmatrix}}_{\Delta u} \begin{pmatrix} u \\ u \\ u \\ u \end{pmatrix}}_{b}$$

Im nächsten Schritt wird A in den diagonalen Teil und den Rest aufgeteilt und anschließend A_2 auf die andere Seite gebracht.

(3.40)
$$\begin{aligned} &\left(\begin{pmatrix} J_{11} & & & \\ & J_{11} & & \\ & & J_{11} & \\ & & & J_{11} \end{pmatrix} - \alpha \begin{pmatrix} -2 & & & \\ & -2 & & \\ & & -2 & \\ & & & -2 \end{pmatrix} \right) \begin{pmatrix} du^{k+1} \\ du^{k+1} \\ du^{k+1} \\ du^{k+1} \end{pmatrix} \\ &= \begin{pmatrix} -J_{12} \\ -J_{12} \\ -J_{12} \\ -J_{12} \end{pmatrix} + \alpha \begin{pmatrix} -2 & 1 & 1 & \\ 1 & -2 & & 1 \\ 1 & & -2 & 1 \\ & 1 & 1 & -2 \end{pmatrix} \begin{pmatrix} u \\ u \\ u \\ u \end{pmatrix} + \alpha \begin{pmatrix} & 1 & 1 & \\ 1 & & & 1 \\ 1 & & & 1 \\ & 1 & 1 & \end{pmatrix} \begin{pmatrix} du^{k} \\ du^{k} \\ du^{k} \\ du^{k} \end{pmatrix} \end{aligned}$$

Zuletzt wird noch nach du umgestellt indem durch A_1 geteilt wird, was äquivalent zur Multiplikation der invertierten Matrix ist.

(3.41)
$$\begin{aligned} \begin{pmatrix} du^{k+1} \\ du^{k+1} \\ du^{k+1} \\ du^{k+1} \end{pmatrix} = &\left(\begin{pmatrix} J_{11} & & & \\ & J_{11} & & \\ & & J_{11} & \\ & & & J_{11} \end{pmatrix} - \alpha \begin{pmatrix} -2 & & & \\ & -2 & & \\ & & -2 & \\ & & & -2 \end{pmatrix} \right)^{-1} \\ &\cdot \left(\begin{pmatrix} -J_{12} \\ -J_{12} \\ -J_{12} \\ -J_{12} \end{pmatrix} + \alpha \left(\begin{pmatrix} -2 & 1 & 1 & \\ 1 & -2 & & 1 \\ 1 & & -2 & 1 \\ & 1 & 1 & -2 \end{pmatrix} \begin{pmatrix} u \\ u \\ u \\ u \end{pmatrix} + \begin{pmatrix} & 1 & 1 & \\ 1 & & & 1 \\ 1 & & & 1 \\ & 1 & 1 & \end{pmatrix} \begin{pmatrix} du^{k} \\ du^{k} \\ du^{k} \\ du^{k} \end{pmatrix} \right) \right) \end{aligned}$$

Der allgemeine Jacobi Löser für die oben angesprochene Gleichung lautet somit:

$$du^{k+1} = \frac{-J_{12} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{u_{\bar{i},\bar{j}} + du^k_{\bar{i},\bar{j}} - u_{i,j}}{h_l^2}}{J_{11} + \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{1}{h_l^2}} \tag{3.42}$$

3.8.2 Gauß-Seidel-Methode

Die Gauß-SeidelMethode ist eine Erweiterung des Jacobi-Verfahrens. Die Matrix A wird hier etwas anders aufgeteilt als es noch zuvor der Fall war. Da der diagonale Teil nur eine sehr vage Approximation der ursprünglichen Matrix ist, kommt hier noch der untere Teil des Restes dazu, womit eine Dreiecksmatrix und eine bessere Näherung von A entsteht. Die Vorteile dieser Methode sind eine schnellere Konvergenz der Lösung, etwa um den Faktor 50 und eine Einsparung des Speichers, weil nicht wie bei der Jacobi Methode zwei Vektoren gespeichert werden müssen, einmal für k und $k+1$, sondern die bereits berechneten Werte $k+1$ direkt in die aktuelle Iteration mit einfließen.

(3.43)

$$du^{k+1} = \frac{-J_{12} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l^-(i,j)} \frac{u^{k+1}_{\bar{i},\bar{j}} + du^{k+1}_{\bar{i},\bar{j}} - u^{k+1}_{i,j}}{h_l^2} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l^+(i,j)} \frac{u^{k}_{\bar{i},\bar{j}} + du^{k}_{\bar{i},\bar{j}} - u^{k}_{i,j}}{h_l^2}}{J_{11} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{1}{h_l^2}}$$

wobei N_l^- und N_l^+ für die untere sowie obere Dreicksmatrix stehen, welche durch die Gauß-Seidel-Erweiterung dazu kam.

3.8.3 Successive Over-Relaxation

Der Successive Over-Relaxation Löser ist wiederum eine Erweiterung des Gauß Seidel Solvers.

(3.44)

$$du^{k+1} = (1-\omega) \cdot du^k + \omega \cdot \left(\frac{-J_{12} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l^-(i,j)} \frac{u^{k+1}_{\bar{i},\bar{j}} + du^{k+1}_{\bar{i},\bar{j}} - u^{k+1}_{i,j}}{h_l^2} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l^+(i,j)} \frac{u^{k}_{\bar{i},\bar{j}} + du^{k}_{\bar{i},\bar{j}} - u^{k}_{i,j}}{h_l^2}}{J_{11} + \alpha \sum_{l \in x,y} \sum_{(\bar{i},\bar{j}) \in N_l(i,j)} \frac{1}{h_l^2}} \right)$$

In der SOR-Methode wird zusätzlich ein Überrelaxationsparameter ω eingeführt, welcher dafür sorgt, dass das Iterationsschema noch schneller zu der Lösung konvergiert, etwa um den Faktor 2. Generell wird ein Wert zwischen 1.50 und < 2.0 gewählt, je nach Matrix, wobei das Verfahren nur für Werte zwischen 0 und < 2 überhaupt konvergiert.

4 Segmentationsgestützte Stereorekonstruktion

Der Algorithmus des letzten Kapitels (siehe Sektion 3.2) dient nun als Grundlage für die weiteren Schritte dieser Arbeit. Das Ziel ist es, den zuvor angesprochenen Algorithmus von einer pixelbasierten Vorgehensweise mithilfe einer Segmentierung zu beschleunigen und gegebenenfalls zu verbessern. Doch bevor die einzelnen Schritte erläutert werden, ist es sinnvoll die eigentliche Segmentierung zu erklären.

4.1 Segmentation

Die Aufgabenstellung der Segmentierung an sich ist recht einfach zu beschreiben, sie ist der Vorgang einzelne Pixel zu Gruppen zusammenzufassen, im besten Fall zu den wahren Objekten des Bildes. Im optimalen Fall bestünde das segmentierte Bild also, nach der Anwendung eines Segmentationsalgorithmus, nur noch aus gruppierten Pixel welche jeweils einem Objekt zugeordnet werden.

Abbildung 4.1: Links: Zweites Bild der Venus Testsequenz [SS02], **Rechts:** Segmentiertes bild mit $\sigma = 4$.

In Abbildung 4.1 sind zum einen das unveränderte Bild und daneben ein segmentiertes Bild zu sehen. Gut zu erkennen ist zum Beispiel die Integration der Schrift in die Zeitung, welche bei einem noch höheren σ völlig miteinander verschmelzen würden.

Zur Segmentierung gibt es zahlreiche verschiedene Ansätze. Zum einen gibt es die pixelorientierten Verfahren, welche für jeden Pixel selbst entscheiden, ob er zum Hintergrund oder einem Objekt gehört. Hierzu wird lediglich ein Parameter benötigt, welcher anhand des Grauwertes oder der Farbe entscheidet, zu welcher Region der Pixel zugeordnet wird. Ein weiterer Ansatz sind die regionsbasierten Verfahren. Diese erstellen die Segmente anhand der Information innerhalb des Segmentes, beispielsweise durch eine Energiefunktion, welche die Farbwerte der Segmentierung mit dem Originalbild vergleicht und die geringste Abweichung davon wählt. Schließlich gibt es noch die kantenorientierten Verfahren, welche mit der Kanteninformation versuchen die Objekte einzuschließen. In dieser Arbeit wird die Segmentierung mittels Kantenerkennung vorgenommen. Der verwendete Algorithmus ist die Wasserscheidentransformation Segmentierung, welche nun genauer erklärt wird.

4.1.1 Wasserscheidentransformation

Wie bereits erwähnt, werden zur Segmentierung Kanten benötigt, weshalb das Bild zuerst durch eine Faltung mit dem Gauß-Kern (siehe Abschnitt 2.1.13) etwas geglättet wird. Dies spielt eine wichtige Rolle, da sich der Grad der Glättung, eingestellt durch die Standardabweichung σ des Gauß-Kerns, sehr stark auf die Anzahl und Größe der späteren Segmente auswirkt, worauf gleich noch etwas näher eingegangen wird. Danach müssen Stellen identifiziert werden, welche mögliche Kanten darstellen. Dazu wird zum einen für jeden Pixel die Ableitung in x- sowie in y-Richtung benötigt. Der Betrag des Gradienten (siehe Abschnitt 2.1.5), welcher die Steigung an jedem Punkt angibt, lässt sich dann nach folgendem Schema berechnen [Beu92]:

$$|\nabla f| = \sqrt{f_x^2 + f_y^2} \tag{4.1}$$

Zur Veranschaulichung lassen sich die jeweiligen Werte auch in ein Bild eintragen wie es in Abbildung 4.2 der Fall ist, wobei die weißen Stellen einen hohen Gradienten angeben und schwarze einen kleinen Wert haben oder gar 0 sind.

Abbildung 4.2: Gradient Magnitude des zweiten Bildes der Venus Testsequenz

Hier lässt sich gut die bereits angesprochene Problematik der Glättung erkennen. Da durch ein großes σ viele Pixel der Nachbarschaft in die Berechnung mit einfließen und somit eine Ähnlichkeit der umliegenden Pixel entsteht. Dies hat zur Folge, dass der Gradient kleiner und somit die Segmente größer werden, da bestimmte Kanten nicht mehr als Trennung der Segmente ausreichen. Der Algorithmus durchläuft nun jeden Pixel des Bildes, angefangen am linken oberen Rand, welcher zuerst als besucht markiert wird. Danach werden die Gradienten der 8 umliegenden Nachbarpixel ermittelt. Anschließend wird der Nachbar mit dem kleinsten Gradienten als nächster Pixel der Reihe festgelegt und der Vorgang wiederholt. So entsteht ein Pfad vom Ausgangspixel bis hin zu einem Minimum, an welchem alle Nachbarpixel einen größeren oder gleichen Gradienten besitzen und der Pfad sein Ende findet. Alle Pixel des Pfades gehören nun zu dem selben Segment. Man kann sich dies wie einen Wassertropfen vorstellen, welcher am Rand einer Schale beginnt und den Weg bis zum Boden herunterfließt. Dies wird so lange wiederholt, bis schließlich jeder Pixel einmal besucht wurde. Sollte der Tropfen auf einen Pixel treffen, welcher bereits besucht wurde, kann die Iteration sofort gestoppt werden, da der Pfad dieses Pixels bereits bekannt ist.

Während der Arbeit wurden zwei verschiedene Varianten der Segmentierung getestet. In der ersten Methode wurde jede Ebene der Coarse-to-fine-Pyramide einzeln segmentiert, was allerdings ein paar Probleme bei der Zuordnung der Segmente verursachte. Die Idee der zweiten Methode war das ursprüngliche Bild zu segmentieren und es mittels eines Nearest Neighbour Algorithmus auf die benötigten Auflösungen anzupassen. Im Nachhinein hat sich die Methode der Nearest Neighbour Skalierung als die bessere herausgestellt, welche von nun an in den verschiedenen Algorithmen ihren Einsatz findet.

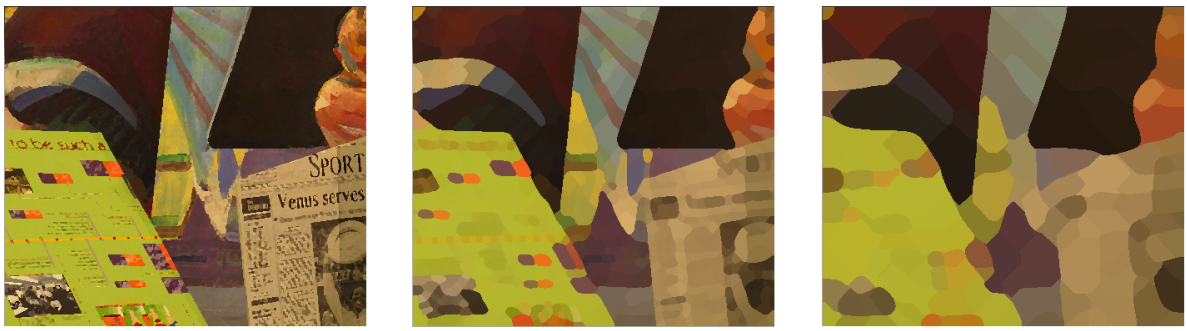


Abbildung 4.3: Verschieden starke Segmentierungen. **Links: σ: 0.3, Mitte: σ: 4, Rechts: σ: 10**

4.2 Algorithmen

Die Anpassung auf ein segmentationsgesützten Ansatz ist ein längerer Prozess, indem mehrere Erweiterungen den Weg in das Funktional gefunden haben und deshalb schrittweise die wichtigsten Änderungen aufgeführt und ähnlich der Baseline beschrieben werden. Die drei großen Änderungen waren zum einen die Anpassung des Funktionals auf eine segmentbasierte Variante. Anschließend wurde Daten- und Glattheitsterm mithilfe einer subquadratischen Funktion abgeschwächt und robuster gegen Ausreißer gemacht. Der letzte Schritt war die Erweiterung auf ein affines Flussmodell.

4.3 Segmentierter Ansatz

Im ersten Schritt ist es das Ziel, den bereits bekannten Baseline-Ansatz (siehe Abschnitt 3.2) auf ein Modell umzustellen, welches den Fluss mithilfe einer Segmentierung des Bildes berechnet. Im Gegensatz zur Baseline wird die Funktion diskret formuliert um bestimmte Bereiche etwas leichter darzustellen. Dies erfordert eine Änderung des Bewegungstensors, welcher nicht mehr aus den Pixel selbst besteht, sondern aus der Summe derer, welche innerhalb des entsprechenden Segments liegen. Zusätzlich wird der Bewegungstensor normiert, sodass größere Segmente nicht mehr Einfluss auf das Ergebnis haben als kleine. Ein weiterer Schritt ist die Anpassung des Glattheitsterms. Da Segmente keine feste Position oder Anordnung haben, können für jedes Segment unterschiedlich viele Nachbarn entstehen. In der Pixelbasierten Variante konnten drei Fälle auftreten. Es gab Eckpixel mit 2 Nachbarn, Kantenpixel mit 3 Nachbarn und Pixel, welche mitten im Bild lagen, und demnach 4 benachbarte Pixel hatten. In der segmentierten Variante können dagegen theoretisch beliebig viele Nachbarn entstehen. Auch hier wird durch die Anzahl der Nachbarn geteilt, aus dem selben Grund wie bei dem Bewegungstensor. All dies führt schließlich zu folgender Funktion:

$$E(u) = \sum_{l=1}^{m} \frac{1}{|\Omega_l|} \int_{\Omega_l} \sum_{i=1}^{3} (f_{2i}(x+u_l, y) - f_{1i}(x,y))^2 dxdy + \alpha \sum_{l=1}^{m} \frac{1}{|N(l)|} \sum_{j \in N(l)} (u_l - u_j)^2 \tag{4.2}$$

Das Funktional besteht wie bisher aus einer Energiefunktion, welche es zu minimieren gilt. Wobei m für die Anzahl der Segmente steht und l für das aktuelle Segment. Der Datenterm wird wie bereits erwähnt durch $\frac{1}{|\Omega l|}$ normiert und über den Bereich $\int_{\Omega l}$ aufsummiert, wobei jeder Pixel x innerhalb des Segments durch den gleichen Segmentfluss u_l verschoben wird. Gleiches beim Glattheitsterm, m steht für die Segmente und $\frac{1}{|N(l)|}$ für die Normierung über die Anzahl der Nachbarn von Segment l. Die Differenz der Nachbarn zu ihrem zentralen Segment bilden schlussendlich den Glattheitsterm.

4.3.1 Minimierung

Die Minimierung läuft ähnlich ab wie noch bei der Baseline (siehe Abschnitt 3.4), mit dem Unterschied, dass es nun diskret formuliert wurde und deshalb die Euler-Lagrange-Gleichung nicht länger benötigt wird, sondern einfach differenziert werden kann. Was zu folgender Ableitung führt:

$$0 = \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + u_l, y) - f_{1i}(x, y)) f_{2i_x}(x + u_l)\, dxdy + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l - u_j) \tag{4.3}$$

wobei der Faktor $\frac{N_l+N_j}{N_l \cdot N_j}$ die Normalisierung ist, welche den Glattheitsterm mit steigender Anzahl an Nachbarn heruntergewichtet. Durch die Kettenregel entsteht das Produkt der äußeren Ableitung des Datenterms, welcher mit der inneren Ableitung multipliziert wird. Gleiches gilt für den Glattheitsterm.

4.3.2 Warping

Als nächstes wird die Gleichung für das Warping vorbereitet, indem zuerst ein Iterationsschema eingeführt wird und danach mit der Regel $u^{k+1} = u^k + du^k$ ein Inkrement eingeführt wird, welches die kleinen Verschiebungen bestimmt. Am Ende wird alles in eine Coarse to fine Pyramide integriert (siehe Abschnitt 3.6):

$$0 = \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + u_l^{k+1}, y) - f_{1i}(x, y)) f_{2i_x}(x + u_l^k, y)\, dxdy + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^{k+1} - u_j^{k+1}) \tag{4.4}$$

Durch Anwendung der bereits angesprochenen Regel wird u^{k+1} durch die Inkrementelle Berechnung ersetzt:

$$\begin{aligned} 0 = {} & \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (u_l^k + du_l^k), y) - f_{1i}(x, y)) f_{2i_x}(x + u_l^k, y)\, dxdy \\ & + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^k + du_l^k - u_j^k - du_j^k) \end{aligned} \tag{4.5}$$

Nun bestehen alle Verschiebungen wieder komplett aus dem alten Zeitschritt und können so zur Berechnung des neuen u^{k+1} benutzt werden. Zuvor muss allerdings nach du_k linearisiert werden. Dies geschieht durch die bereits bekannte Taylorreihe, welche um den Punkt $(x + u_l^k, y)$ entwickelt wird:

$$f_2(x + (u_l^k + du_l^k), y) \approx f_2(x + u_l, y) + \begin{pmatrix} x + u_l^k + du_l^k - x - u_l^k \\ y - y \end{pmatrix} \begin{pmatrix} f_{2_x}(x + u_l^k, y) \\ f_{2_y}(x + u_l^k, y) \end{pmatrix} \tag{4.6}$$

was ausmultipliziert und in die obige Gleichung eingesetzt folgendem entspricht:

$$\begin{aligned} 0 = &\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i_x}(x + u_l^k, y) du_l^k + \underbrace{f_{2i}(x + u_l^k, y) - f_{1i}(x, y))}_{\approx f_{it}} f_{2i_x}(x + u_l^k, y)\, dxdy \\ &+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^k + du_l^k - u_j^k - du_j^k) \end{aligned} \tag{4.7}$$

Im letzten Schritt kann die Gleichung auf die Bewegungstensor-Notation umgeformt werden:

$$\begin{aligned} 0 = &\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i_x}(x + u_l^k, y)^2 du_l^k + f_{2i_x}(x + u_l^k, y) f_{2i_t}(x + u_l^k, y))\, dxdy \\ &+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^k + du_l^k - u_j^k - du_j^k) \end{aligned} \tag{4.8}$$

$$0 = \frac{1}{|\Omega l|} \int_{\Omega l} (\sum_{i=1}^{3} J_{11i} du_l^k + J_{12i})\, dxdy + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^k + du_l^k - u_j^k - du_j^k) \tag{4.9}$$

wobei das Integral durch die Summe ersetzt und mit der Farbe in den Bewegungstensor gezogen werden kann:

$$0 = \frac{1}{|\Omega l|} \hat{J_{11}} du_l^k + \frac{1}{|\Omega l|} \hat{J_{12}} + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l^k + du_l^k - u_j^k - du_j^k) \tag{4.10}$$

mit dem Bewegungstensor Tensor:

$$\hat{J} = \begin{pmatrix} \sum_{j \in l} (\sum_{i=1}^{3} f_{2ij_x}^2) & \sum_{j \in l} (\sum_{i=1}^{3} f_{2ij_x} f_{2ij_t}) \\ \sum_{j \in l} (\sum_{i=1}^{3} f_{2ij_t} f_{2ij_x}) & \sum_{j \in l} (\sum_{i=1}^{3} f_{2ij_t}^2) \end{pmatrix} \tag{4.11}$$

Die hier erstellte Funktion ist bereits zuvor diskret formuliert worden (siehe Abschnitt 4.3), weshalb an dieser Stelle auf eine Diskretisierung, wie es noch in der Baseline das Fall war, verzichtet und sie direkt gelöst werden kann.

4.3.3 Lösung

Die Lösung des linearen Gleichungssystem läuft im Prinzip ähnlich ab, wie die pixelbasierte Variante. Zuerst müssen alle Terme, welche nichts mit du_l^k zu tun haben auf die andere Seite gebracht werden, dass die Form $Ax = b$ entsteht. Danach wird die Hauptdiagonale vom Rest getrennt und nach du_l^k umgestellt.

$$0 = \frac{1}{|\Omega l|}\hat{J_{11}}du_l^k + \frac{1}{|\Omega l|}\hat{J_{12}} + \alpha\frac{N_l + N_j}{N_l \cdot N_j}\sum_{j\in N(l)}(u_l^k + du_l^k - u_j^k - du_j^k) \tag{4.12}$$

ist äquivalent zu

$$\frac{1}{|\Omega l|}\hat{J_{11}}du_l^k + \alpha\frac{N_l + N_j}{N_l \cdot N_j}\sum_{N(l)}du_l^k + \alpha\frac{N_l + N_j}{N_l \cdot N_j}\sum_{j\in N(l)}du_j^k = -\frac{1}{|\Omega l|}\hat{J_{12}} - \alpha\frac{N_l + N_j}{N_l \cdot N_j}\sum_{j\in N(l)}(u_l - u_j) \tag{4.13}$$

Anschließen wird noch nach du und auf den SOR-Löser umgestellt:

$$\begin{aligned} du_l^{k+1} &= (1-\omega)du_l^k \\ &\quad + \omega\left(\frac{-\frac{1}{|\Omega l|}\hat{J_{12}} + \alpha\frac{N_l+N_j}{N_l\cdot N_j}\sum_{j\in N_l^-}(u_j^{k+1} + du_j^{k+1} - u_l^{k+1}) + \alpha\frac{N_l+N_j}{N_l\cdot N_j}\sum_{j\in N_l^+}(u_j^k + du_j^k - u_l^k)}{\frac{1}{|\Omega l|}\hat{J_{11}} + \alpha\frac{N_l+N_j}{N_l\cdot N_j}\sum_{N(l)}1}\right) \end{aligned} \tag{4.14}$$

4.3.4 Ergebnisse

Im folgenden Abschnitt sind die Ergebnisse für zwei Bilder, jeweils aus der Middlebury Testsequenz Venus und Cones, welche mit dem bisherigen Algorithmus berechnet wurden.

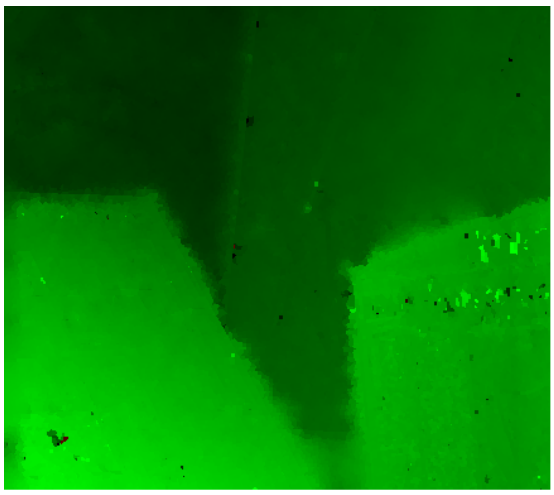
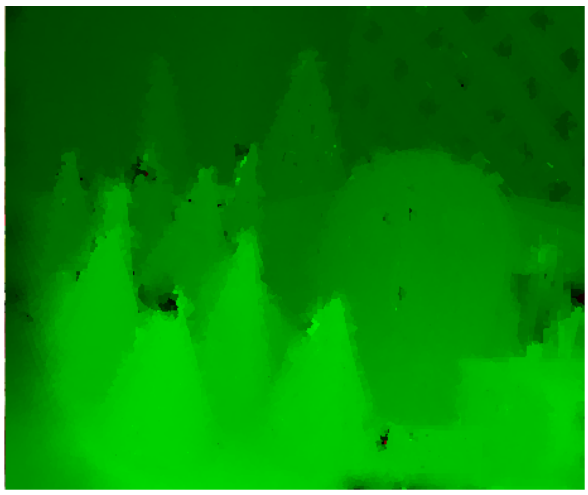



Abbildung 4.4: Links: Venus, **Rechts:** Cones

4.4 Subquadratischer Ansatz

Die Idee hinter dem subquadratischen Ansatz ist, den Einfluss von Ausreißern zu verringern und die Kanten besser zu erhalten. Wie in Abbildung 4.4 zu sehen ist, sind vereinzelt kleine schwarze Pixel oder sehr helle zu erkennen, welche vom Algorithmus falsch zugeordnet wurden und eine zu kleine sowieso zu große Verschiebung bedeuten. Um dies besser zu verhindern und ein gleichmäßigeres Bild zu erzeugen, wird nun eine Funktion eingeführt, welche das Energiefunktional nicht länger quadratisch, sondern subquadratisch bestraft. Dies ist vergleichbar mit einer Funktion welche stärker als linear, jedoch langsamer als eine quadratische Funktion ansteigt.

(4.15)
$$E(u) = \sum_{l=1}^{m} \Psi_D \left(\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x+u_l, y) - f_{1i}(x,y))^2 dxdy \right) + \alpha \sum_{l=1}^{m} \frac{1}{|N(l)|} \sum_{j \in N(l)} \Psi_S((u_l - u_j)^2)$$

4.4.1 Funktion Ψ

Die Funktion Ψ hat die Aufgabe, den Datenterm, sowie den Glattheitsterm subquadratisch zu bestrafen. Aus diesem Grund werden beide als Argument an Ψ übergeben, welche folgende Form hat:

(4.16) $$\Psi(s^2) = 2\epsilon^2 \cdot \sqrt{1 + \frac{s^2}{\epsilon^2}}$$

wobei ϵ ein weiterer Parameter in Daten- und Glattheitsterm ist. Ψ wird zwar auf beide Terme angewendet, beide besitzen jedoch unterschiedliche ϵ, da die Größenordnung der Argumente nicht gleich sein muss.

Der große Unterschied zu der bisherigen Variante ist nun, dass bei Abweichung des Daten- oder Glattheitsterm, die Energie direkt in die Höhe geschossen wäre und all diese Pixel als mögliche Treffer ausscheiden würden , da der jeweils andere Term, nicht die Möglichkeit hat diese auszugleichen. Durch den Faktor, welcher mit Ψ' vorangestellt wird, werden diese hohen Energien herunter gewichtet, sodass jeweils der andere Term dominieren kann.

4.4.2 Minimierung

Die Minimierung mit subquadratischer Bestrafungsfunktion verhält sich fast identisch zur bereits bekannten Minimierung des segmentbasierten Ansatzes. Zuerst wird mit der Kettenregel Ψ abgeleitet, wobei die äußere Ableitung Ψ' als Faktor vorangestellt und danach das Argument von Ψ wie in Abschnitt 4.3.1 differenziert wird.

(4.17)
$$0 = \Psi'_D \cdot \left(\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x+u_l, y) - f_{1i}(x,y)) f_{2i_x}(x+u_l, y)\, dxdy \right) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (u_l - u_j)$$

mit

(4.18)
$$\Psi'(s^2) = \frac{1}{\sqrt{1 + \frac{s^2}{\epsilon^2}}}$$

4.4.3 Warping

Auch das Warping ist mit der segmentbasierten Version identisch, wobei nur Ψ' jeweils zum Zeitschritt k mitberechnet werden muss. Die Linearisierung und Integration in die Coarse-to-fine-Pyramide ist hingegen genau wie in Abschnitt 4.3.2 zu berechnen.

(4.19)
$$\begin{aligned} 0 = {} & [\Psi'_D]^k \cdot \left(\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x+u_l^{k+1}, y) - f_{1i}(x,y)) f_{2i_x}(x+u_l^{k}, y)\, dxdy \right) \\ & + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} [\Psi'_S]^k \cdot (u_l^{k+1} - u_j^{k+1}) \end{aligned}$$

$$
\begin{aligned}
0 = [\Psi'_D]^k \cdot \left(\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (u_l^k + du_l^k), y) - f_{1i}(x, y)) f_{2i_x}(x + u_l^k, y)\, dxdy \right) \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} [\Psi'_S]^k \cdot (u_l^k + du_l^k - u_j^{k+1} - du_l^k)
\end{aligned} \tag{4.20}
$$

Nun kann nach du_l^k linearisiert und die Farbe mit dem Integral in den Bewegungstensor gezogen werden.

$$
\begin{aligned}
0 = [\Psi'_D]^k \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i_x}(x + u_l^k, y)^2 du_l^k + f_{2i_x}(x + u_l^k, y) f_{2i_t}(x + u_l^k, y)\, dxdy \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} [\Psi'_S]^k (u_l^k + du_l^k - u_j^k - du_j^k)
\end{aligned} \tag{4.21}
$$

$$
0 = [\Psi'_D]^k \left(\frac{1}{|\Omega l|} \hat{J_{11}} du_l^k + \frac{1}{|\Omega l|} \hat{J_{12}} \right) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} [\Psi'_S]^k (u_l^k + du_l^k - u_j^k - du_j^k) \tag{4.22}
$$

4.4.4 Lösung

Die Lösung ist analog zu der segmentierten Variante, einzig Ψ' muss ausmultipliziert und dann mit den entsprechenden Termen auf die andere Seite gebracht werden.

$$
0 = [\Psi'_D]^k \left(\frac{1}{|\Omega l|} \hat{J_{11}} du_l^k + \frac{1}{|\Omega l|} \hat{J_{12}} \right) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} [\Psi'_S]^k (u_l^k + du_l^k - u_j^k - du_j^k) \tag{4.23}
$$

$$
\begin{aligned}
\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} du_l^k + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{N(l)} du_l^k + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} du_j^k \\
= -\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{12}} - \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (u_l - u_j)
\end{aligned} \tag{4.24}
$$

$$
\begin{aligned}
du_l^{k+1} = (1 - \omega) du_l^k \\
+ \omega \left(\frac{-\Psi'^k_S \frac{1}{|\Omega l|} \hat{J_{12}} + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N_l^-} (u_j^{k+1} + du_j^{k+1} - u_l^{k+1})}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right) \\
+ \omega \left(\frac{\Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N_l^+} (u_j^k + du_j^k - u_l^k)}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right)
\end{aligned} \tag{4.25}
$$

4.4.5 Lagged-Nonlinearity-Ansatz

Da bei der Funktion Ψ durch die Ableitung die Flussvariable du_l in den Nenner rückt, ist nicht länger ein lineares Gleichungssystem, sondern ein nicht-lineares Gleichungssytem vorhanden, welches auf die herkömmliche Weise nicht mehr gelöst werden kann. Deshalb wird das System als Serie von linearen Gleichungssystemen gelöst. Dies bedeutet, dass Ψ' jeweils zum alten Zeitpunkt k ausgewertet wird, welches zwar nur eine ungenaue, jedoch ausreichende Lösung ist, damit das Gleichungssystem konvergiert. Ψ' hängt also immer etwas hinterher (lagged) und wird erst nach jeder Iteration auf den neusten Stand gebracht. Erste Ergebnisse sind im folgenden Abschnitt zu finden.

4.4.6 Ergebnisse

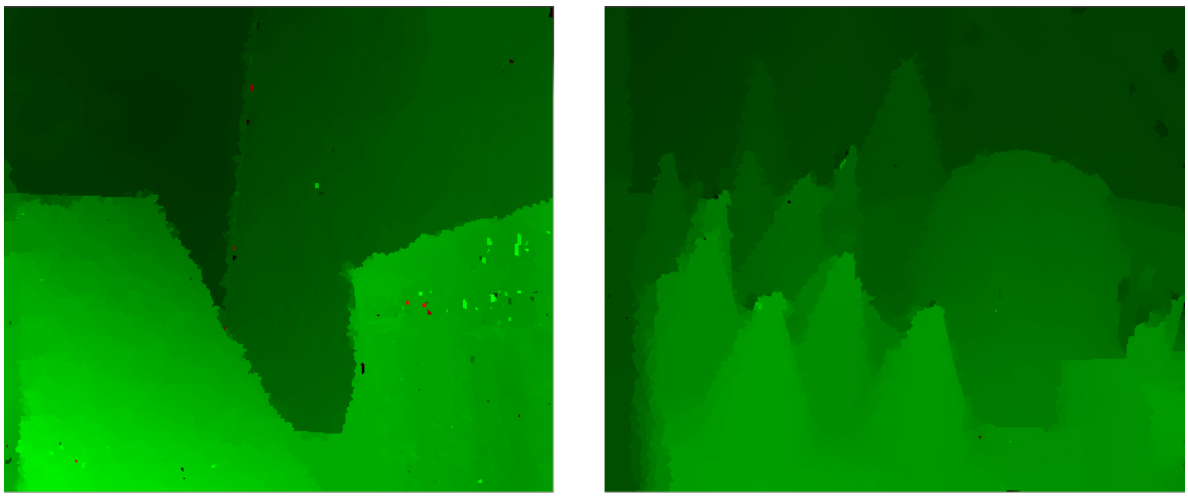


Abbildung 4.5: Links: Venus, **Rechts:** Cones

4.5 Affiner Parametrisierungs Ansatz

Die nächste Idee war es eine Parametrisierung einzuführen indem die Funktion $u(x, y)$ auf eine affine Form umgestellt wird. Realisiert wird dies durch eine Ebene der Form $ax + by + c$, welche für jedes Segment erzeugt wird. Dies formuliert den Fluss allgemeiner und hat den Vorteil dass nicht länger von einem konstanten Fluss innerhalb der Segmente ausgegangen wird, sondern von einem affinen.

4.5.1 Affine Parametrisierung

Um den Vorgang etwas genauer zu beschreiben, wird der Vorgang von der Bewegungstensor-Notation umgeformt. Als erstes wird der Flussvektor $w = (u, 1)^T$ auf die affine Form umgestellt, indem u

durch $ax + by + c$ ersetzt wird. Danach kann w in zwei Vektoren aufgespalten werden, wobei p die Flussvariablen beinhaltet.

$$w = \begin{pmatrix} u \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by + c \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} x & y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M} \cdot \underbrace{\begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}}_{p} \tag{4.26}$$

Dies kann nun in die Bewegunstensor-Notation eingesetzt werden. Um schließlich wieder die bekannte Form zu erhalten, wird der vordere Teil transponiert und anschließend die beiden Vektoren M^T und M in den Tensor hinein multipliziert.

$$w^T J w = (Mp)^T J (Mp) = (p^T M^T) J (Mp) = p^T J p \tag{4.27}$$

wobei J für folgende Matrix steht:

$$J = \begin{pmatrix} f_x^2 x^2 & f_x^2 yx & f_x^2 x & f_x f_t x \\ f_x^2 xy & f_x^2 y^2 & f_x^2 y & f_x f_t y \\ f_x^2 x & f_x^2 y & f_x^2 & f_x f_t \\ f_t f_x x & f_t f_x y & f_t f_x & f_t^2 \end{pmatrix} \tag{4.28}$$

Die Besonderheit im Gegensatz zu den bisherigen Ansätzen ist der, dass nun die Stelle des Pixel x, y, Einfluss auf das Ergebnis hat. Durch Anwendung der Parametrisierung wird schlussendlich der finale Ansatz erzeugt, welcher die letzte Stufe dieser Arbeit darstellt.

$$\begin{aligned} E(a,b,c) = &\sum_{l=1}^{m} \Psi_D \left(\frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (ax_l + by_l + c), y) - f_{1i}(x,y))^2 dxdy \right) \\ &+ \alpha \sum_{l=1}^{m} \frac{1}{|N(l)|} \sum_{j \in N(l)} \Psi_S((a_l - a_j)^2 + (b_l - b_j)^2 + (c_l - c_j)^2) \end{aligned} \tag{4.29}$$

4.5.2 Minimierung

Die Minimierung hat sich in der Hinsicht geändert, dass nicht länger nur eine Gleichung vorhanden ist, sondern gleich drei. Dies hat den Grund, dass der Fluss nicht mehr ausschließlich durch u beschrieben wird, sondern durch die Parameter a, b, c.

Ableitung nach a:

$$
\begin{aligned}
0 = \Psi'_D \cdot \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (ax_l + by_l + c), y)\, dxdy - f_{1i}(x, y)) \\
\cdot \int_{\Omega l} \sum_{i=1}^{3} f_{2i_x}(x + (ax_l + by_l + c), y)\, dxdy \cdot x \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (a_l - a_j)
\end{aligned} \tag{4.30}
$$

Ableitung nach b:

$$
\begin{aligned}
0 = \Psi'_D \cdot \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (ax_l + by_l + c), y)\, dxdy - f_{1i}(x, y)) \\
\cdot \int_{\Omega l} \sum_{i=1}^{3} f_{2i_x}(x + (ax_l + by_l + c), y)\, dxdy \cdot y \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (b_l - b_j)
\end{aligned} \tag{4.31}
$$

Ableitung nach c:

$$
\begin{aligned}
0 = \Psi'_D \cdot \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (ax_l + by_l + c), y)\, dxdy - f_{1i}(x, y)) \\
\cdot \int_{\Omega l} \sum_{i=1}^{3} f_{2i_x}(x + (ax_l + by_l + c), y)\, dxdy \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (c_l - c_j)
\end{aligned} \tag{4.32}
$$

mit

$$
\Psi'_D = \Psi'(\hat{J_{11}}a^2 + \hat{J_{22}}b^2 + \hat{J_{33}}c^2 + 2\hat{J_{12}}ab + 2\hat{J_{13}}ac + 2\hat{J_{23}}bc + 2\hat{J_{14}}a + 2\hat{J_{24}}b + 2\hat{J_{34}}c + \hat{J_{44}}) \tag{4.33}
$$

$$
\Psi'_S = \Psi'((a_l - a_j)^2 + (b_l - b_j)^2 + (c_l - c_j)^2) \tag{4.34}
$$

Zu beachten ist hier, dass im Datenterm drei mal die Kettenregel angewendet werden muss, wobei zuerst Ψ abgeleitet wird, dann f_2, sowie f_{2_x}, welche die Faktoren x und y jeweils erzeugt. Bei den beiden Ψ' Funktionen wird das Argument jeweils in die Bewegungstensor-Notation umgeformt und anschließend die affine Parametrisierung eingesetzt. Die Einträge von J entsprechend dann der weiter oben beschriebenen Art.

4.5.3 Warping

Das Warping hat sich in der Hinsicht ein wenig geändert, dass nun bei 3 Gleichungen ein Inkrement eingeführt wird und zwar für alle 3 Variablen a, b, c. Da sich die Ableitungen jedoch nur in dem Faktor x, y und 1, sowie dem Glattheitsterm unterschieden, wird hier nur die Variante für a erläutert. Die beiden Anderen können dazu analog berechnet werden.

Im ersten Schritt wird ein Iterationsschema eingeführt, welches anschließend in die Coarse-to-Fine-Pyramide eingesetzt wird, indem die Inkremente da, db und dc eingeführt werden.

$$
\begin{aligned}
0 = \Psi'_D \cdot \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i}(x + (ax_l + by_l + c) + (dax_l + dby_l + dc), y)\, dxdy - f_1(x, y)) \\
\cdot \int_{\Omega l} f_{2i_x}(x + (ax_l + by_l + c), y)\, dxdy \cdot x \\
+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (a_l + da_l - a_j - da_j)
\end{aligned}
\tag{4.35}
$$

Nun kann mit der Taylorreihe um den Punkt $(x + (ax + by + c))$ nach da, db und dc linearisiert werden.

(4.36)

$$
\begin{aligned}
f_2(x + (ax_l + by_l + c) + (dax_l + dby_l + dc), y) \approx f_2(x + (ax + by + c), y) \\
+ \begin{pmatrix} x + (ax + by + c) + (dax + day + dc) - x - (ax + by + c) \\ y - y \end{pmatrix} \cdot \begin{pmatrix} f_{2_x}(x + (ax + by + c), y) \\ f_{2_y}(x + (ax + by + c), y) \end{pmatrix}
\end{aligned}
$$

was ausmultipliziert und in die alte Gleichung eingesetzt folgendes ergibt:

(4.37)

$$0 = \Psi'_D \cdot \frac{1}{|\Omega l|} \int_{\Omega l} \sum_{i=1}^{3} (f_{2i_x} x \cdot da + f_{2i_x} y \cdot db + f_{2i_x} \cdot dc + \underbrace{f_{2i}(x + (ax + by + c), y) - f_{1i}(x, y)}_{\approx f_{it}}\, dxdy)$$
$$\cdot \int_{\Omega l} \sum_{i=1}^{3} f_{2i_x}(x + (ax_l + by_l + c), y)\, dxdy \cdot x$$
$$+ \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (a_l + da_l - a_j - da_j)$$

Multipliziert man die Gleichung aus und zieht das Integral mit der Farbe in den Bewegungstensor erhält man die finale Form der Ableitungen welche anschließend gelöst werden kann:

Für da:

(4.38)
$$0 = \Psi'_D \cdot \frac{1}{|\Omega l|} (\hat{J_{11}} da + \hat{J_{12}} db + \hat{J_{13}} dc + \hat{J_{14}}) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (a_l + da_l - a_j - da_j)$$

Für db:

(4.39)
$$0 = \Psi'_D \cdot \frac{1}{|\Omega l|} (\hat{J_{12}} da + \hat{J_{22}} db + \hat{J_{23}} dc + \hat{J_{24}}) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (b_l + db_l - b_j - db_j)$$

Für dc:

(4.40)
$$0 = \Psi'_D \cdot \frac{1}{|\Omega l|} (\hat{J_{13}} da + \hat{J_{23}} db + \hat{J_{33}} dc + \hat{J_{34}}) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (c_l + dc_l - c_j - dc_j)$$

4.5.4 Lösung

Alle drei Gleichungen werden nun gleichzeitig gelöst und nach jedem Level geupdated. Auf der letzten Ebene angekommen wird der Fluss wieder auf u mit $u = ax + by + c$ zurückgemappt.

(4.41)
$$0 = \Psi'_D \cdot \frac{1}{|\Omega l|} (\hat{J_{11}} da + \hat{J_{12}} db + \hat{J_{13}} dc + \hat{J_{14}}) + \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} \Psi'_S \cdot (a_l + da_l - a_j - da_j)$$

$$
\begin{aligned}
&\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} da_l^k + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{N(l)} da_l^k + \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} da_j^k \\
&= \Psi'^k_D \frac{1}{|\Omega l|} (-\hat{J_{12}} db_l^k - \hat{J_{13}} dc_l^k - \hat{J_{14}}) - \Psi'^k_S \alpha \frac{N_l + N_j}{N_l \cdot N_j} \sum_{j \in N(l)} (a_l - a_j)
\end{aligned} \tag{4.42}
$$

(4.43)

$$
\begin{aligned}
da_l^{k+1} &= (1-\omega) da_l^k \\
&+ \omega \left(\frac{\Psi'^k_S \frac{1}{|\Omega l|} (-\hat{J_{12}} db_l^k - \hat{J_{13}} dc_l^k - \hat{J_{14}}) + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^-} (a_j^{k+1} + da_j^{k+1} - a_l^{k+1})}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right) \\
&+ \omega \left(\frac{\Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^+} (a_j^k + da_j^k - a_l^k)}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{11}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right)
\end{aligned}
$$

Die anderen beiden Gleichung sind analog zu da:

(4.44)

$$
\begin{aligned}
db_l^{k+1} &= (1-\omega) db_l^k \\
&+ \omega \left(\frac{\Psi'^k_S \frac{1}{|\Omega l|} (-\hat{J_{12}} da_l^k - \hat{J_{23}} dc_l^k - \hat{J_{24}}) + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^-} (b_j^{k+1} + db_j^{k+1} - b_l^{k+1})}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{22}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right) \\
&+ \omega \left(\frac{\Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^+} (b_j^k + db_j^k - b_l^k)}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{22}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right)
\end{aligned}
$$

(4.45)

$$
\begin{aligned}
dc_l^{k+1} &= (1-\omega) dc_l^k \\
&+ \omega \left(\frac{\Psi'^k_S \frac{1}{|\Omega l|} (-\hat{J_{13}} da_l^k - \hat{J_{23}} db_l^k - \hat{J_{34}}) + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^-} (c_j^{k+1} + dc_j^{k+1} - c_l^{k+1})}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{33}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right) \\
&+ \omega \left(\frac{\Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{j \in N_l^+} (c_j^k + dc_j^k - c_l^k)}{\Psi'^k_D \frac{1}{|\Omega l|} \hat{J_{33}} + \Psi'^k_S \alpha \frac{N_l+N_j}{N_l \cdot N_j} \sum_{N(l)} 1} \right)
\end{aligned}
$$

Erste Ergebnisse sind im folgenden abschnitt zu sehen.

4.5.5 Ergebnisse

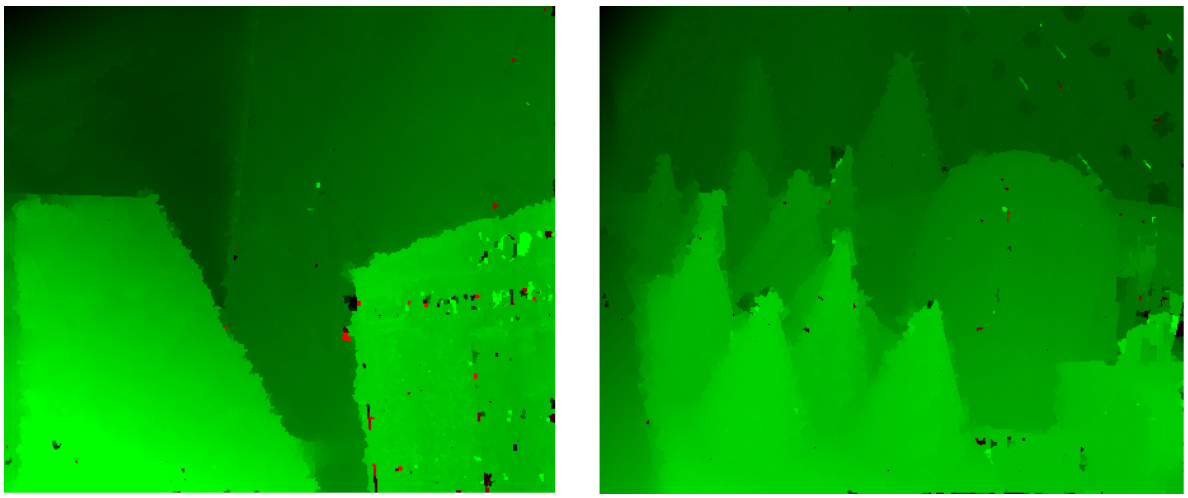


Abbildung 4.6: Links: Venus, **Rechts:** Cones

5 Evaluation

In der Evaluation werden die verschiedenen Entwicklungsschritte miteinander verglichen und analysiert. Als Test wurden Bilder des Middlebury Benchmarks verwendet, genauer die Bildersequenzen von Venus, Tsukuba, Teddy und Cones. Dies sind computergenerierte Bilder, welche extra für den Test eines Stereo Algorithmus entworfen wurden. Aus diesem Grund kann eine Ground Truth, welche gleich etwas näher beschrieben wird, erstellt werden, welche die exakte Lösung darstellt und mit welcher der errechnete Fluss verglichen werden kann. Zunächst folgt jedoch eine kleine Einführung der Parameter, welche zur Erzeugung der Bilder verwendet wurden.

5.0.6 Alpha

Dieser Parameter ist bereits aus dem Funktional bekannt, er gewichtet den Glattheitsterm im Verhältnis zum Datenterm. So kann entschieden werden, ob mehr auf exakte Farbwerte der Segmente geachtet werden oder eher ein glattes Ergebnis vorliegen soll.

5.0.7 Sigma

Sigma steht für die Standardabweichung des Gauß-Kern, mit welchem der Grad der Segmentierung geregelt wird und gleichzeitig für die Vorverarbeitung der Glätte des Bildes steht. Anhand dieser Segmentierung, wird der Startwert des Algorithmus festgelegt.

5.0.8 Iterationen

Dieser Wert steht für die Anzahl an Iterationen, bei welchem der Algorithmus zum Stillstand kommt und die bisher errechnete Lösung an die nächste Ebene weitergegeben wird. Zum einen wurde ein Parameter für die äußere Iteration definiert, welcher für den Lagged-Nonlinearity-Ansatz zuständig ist. Nach jeder äußeren Iteration wird Ψ' angepasst und an den Löser übergeben. Der zweite Parameter steht für die innere Iteration, welcher die Anzahl der Schritte des SOR-Lösers bestimmt.

5.0.9 Omega

Omega gibt den Relaxationsparameter für den SOR-Solver an. Dieser wurde bei allen Lösungen auf 1.95 gesetzt.

5.0.10 Epsilon

Epsilon steht für die beiden Parameter, mit welcher Ψ eingestellt werden kann. Durch ein kleines Epsilon kann das Segment jeweils heruntergewichtet werden, wobei ein großer Wert mehr Fokus darauf legt. Im Prinzip kann so neben Alpha noch etwas feiner gewichtet werden.

5.1 Ground Truth

Die Ground Truth ist ein separates Bild, welches bei dem Prozess mit eingelesen wird um die berechnete Lösung mit ihr zu vergleichen. Sie ist ein im voraus berechnetes Bild, welches zu 100% der perfekten Lösung entspricht. Da die Ursprünglichen Bilder computergeneriert sind, ist für jeden Pixel der Fluss bekannt und kann in der Ground Truth durch einen Wert beschrieben werden. Durch die Differenz dieses Wertes mit der berechneten Lösung kann so schnell erkannt werden, ob der Fluss stimmt oder nicht.

5.2 Okklusions Map

Die Okklusions Map ist ein binäres Bild, welche alle Pixel schwarz markiert, welche vom Algorithmus nicht berechnet werden können. Durch den anderen Blickwinkel der beiden Kameras auf die Szene, sind Bereiche nur auf einem der Bilder wiederzufinden, deshalb werden alle Pixel, welche nicht gematched werden können, bei der Evaluation des Fehler nicht berücksichtigt.

5.3 Bad Pixel Error

Als Testverfahren um die verschiedenen Lösungen zu evaluieren wurde der Bad Pixel Error verwendet. Dieser nimmt die berechnete Lösung und vergleicht diese Pixel für Pixel mit der Ground Truth, wobei die Differenz kleiner als ein vorher bestimmter Threshold sein, muss um als korrekter Pixel erkannt zu werden. Als Threshold wurde in allen Bilder der Wert 1 verwendet, was bedeutet, dass alle Pixel die um mehr als 1 Pixel an der eigentliche Lösung verschieden sind, als falsche Pixel markiert werden. Der finale Fehler entsteht dann durch eine Prozentuale Angabe der falschen Pixel wobei 0% ein in allen Belangen korrektes Bild angibt.

5.4 Vergleich

Im Folgenden Abschnitt werden alle Schritte, welche bis hin zum finalen Algorithmus benötigt wurden, einzeln evaluiert und miteinander verglichen. Dazu wurden alle 4 Testsequenzen mit den Algorithmen berechnet, welche anschließend mit dem Bad Pixel Error bewertet wurden. Anschließend wird jeweils auf die Vor- und Nachteile der jeweiligen Varianten näher eingegangen.

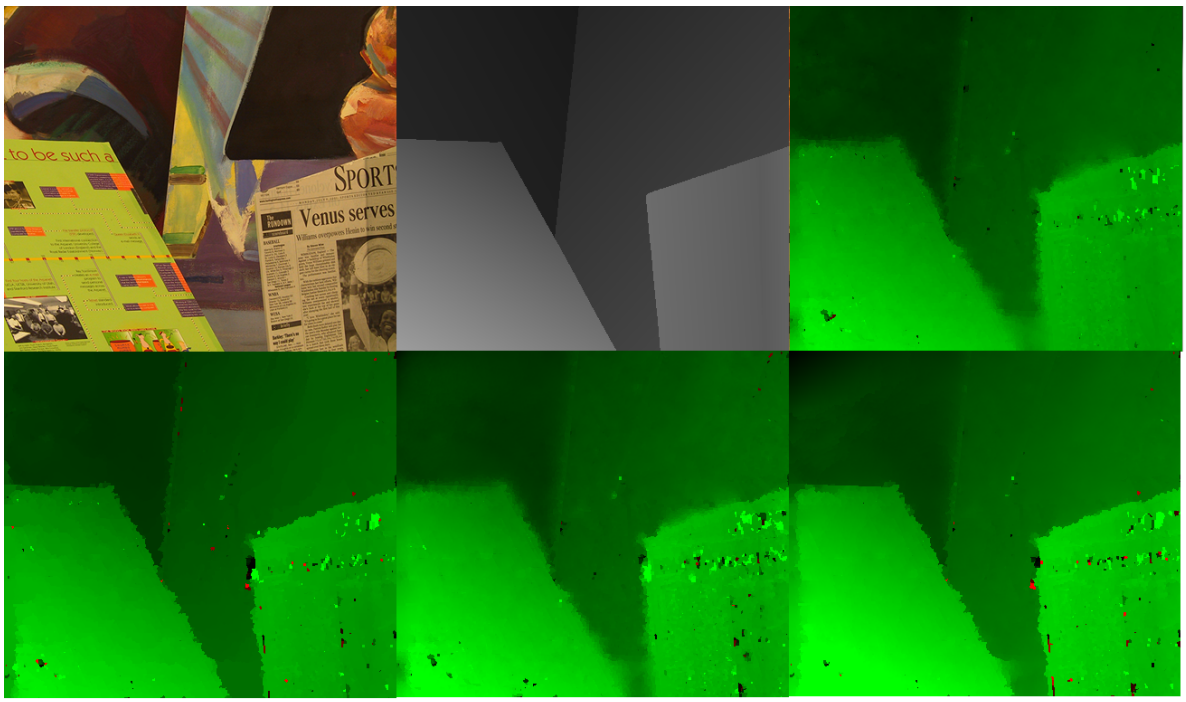


Abbildung 5.1: Oben Links: Venus Testsequenz, **Oben Mitte:** Grount Truth, **Oben Rechts:** Konstanter Ansatz mit α = 75, σ = 0.3 und einem Bad Pixel Error von **7.8%**, **Unten Links:** Robuster konstanter Ansatz mit α = 177, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.0017 und einem Error von **3.39%**, **Unten Mitte:** Affiner Ansatz mit α = 6493814, σ = 0.3 und einem Error von **8.7%**, **Unten Rechts:** Robuster affiner Ansatz mit α = 237137, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.00056 und einem Error von **5.6%**

Bei der Venus Sequenz war vor allem der Bereich der Kanten und die Fläche der Zeitungen zu beachten. Zu erkennen ist, dass mit der konstanten Methode die Form der Zeitungen gut getroffen wurde, jedoch die Kanten ein recht unscharfes Bild ergeben und zum Teil weiter unten auch miteinander verschmelzen. Innerhalb der Zeitung gab es hauptsächlich bei der Schrift Probleme und die Fläche an sich war nicht so glatt wie erhofft. Durch den robusten konstanten Ansatz wurde das Problem der unscharfen Kanten größtenteils behoben. Das Bild ist allgemein schärfer, die Brücke zwischen beiden Zeitungen ist verschwunden und auch die Fläche besteht aus einem gleichmäßigerem Grün. Der affine Ansatz hat sich im Vergleich mit der konstanten Variante nur marginal auf die Qualität ausgewirkt und ist an bestimmten Stellen etwas schlechter. Gerade bei dünnen Objekten wie der Schrift kamen einige Fehler hinzu. Diese konnten allerdings durch die robuste Ψ Funktion wieder zum Teil korrigiert werden.

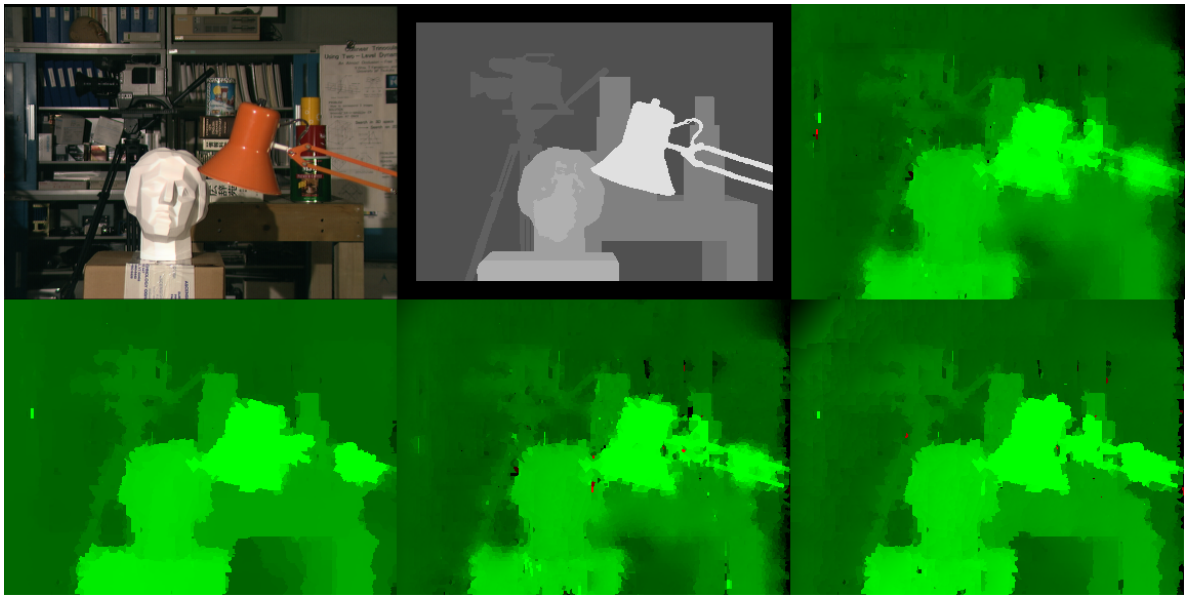


Abbildung 5.2: Oben Links: Tsukuba Testsequenz, **Oben Mitte:** Grount Truth, **Oben Rechts:** Konstanter Ansatz mit α = 153, σ = 0.3 und einem Bad Pixel Error von **13.2%**, **Unten Links:** Robuster konstanter Ansatz mit α = 649, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.0017 und einem Error von **7.4%**, **Unten Mitte:** Affiner Ansatz mit α = 4869674, σ = 0.3 und einem Error von **11.88%**, **Unten Rechts:** Robuster affiner Ansatz mit α = 865964, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.00017 und einem Error von **7.2%**

Bei der Tsukuba Sequenz bestand die Schwierigkeit darin, den gezeigten Objekten die richtige Tiefe zuzuordnen, da viele Objekte durch ein anderes oder gleich mehrere Objekte verdeckt wurden. Gerade der Bereich der Lampe mit der dünnen Halterung wurde oft nicht richtig erkannt, da dort gleich drei verschiedene Tiefen innerhalb weniger Pixel auftreten. So hat der konstante Ansatz zwar die Form der Lampe gut wiedergegeben, dabei aber die Tiefe an den Rändern falsch zugeordnet. Durch die Robustheit wurden die Kanten wieder glatter und die Flächen gleichmäßiger, dabei gingen allerdings dünne Teile der Halterung verloren. Die affine Methode hat auch hier die etwas feineren Objekte zurückgebracht, leidet jedoch im Großen und Ganzen an den gleichen Fehlern wie bereits die konstante Variante. Kombiniert mit der robusten Ψ Funktion, erhält man schließlich wieder die Glattheit aus dem robusten konstanten Ansatz mit dem Vorteil dass nun die dünnen Objekte besser erhalten wurden. Teile die zuvor noch dem Tisch zugeordnet wurden, werden nun korrekt an der Lampe platziert.

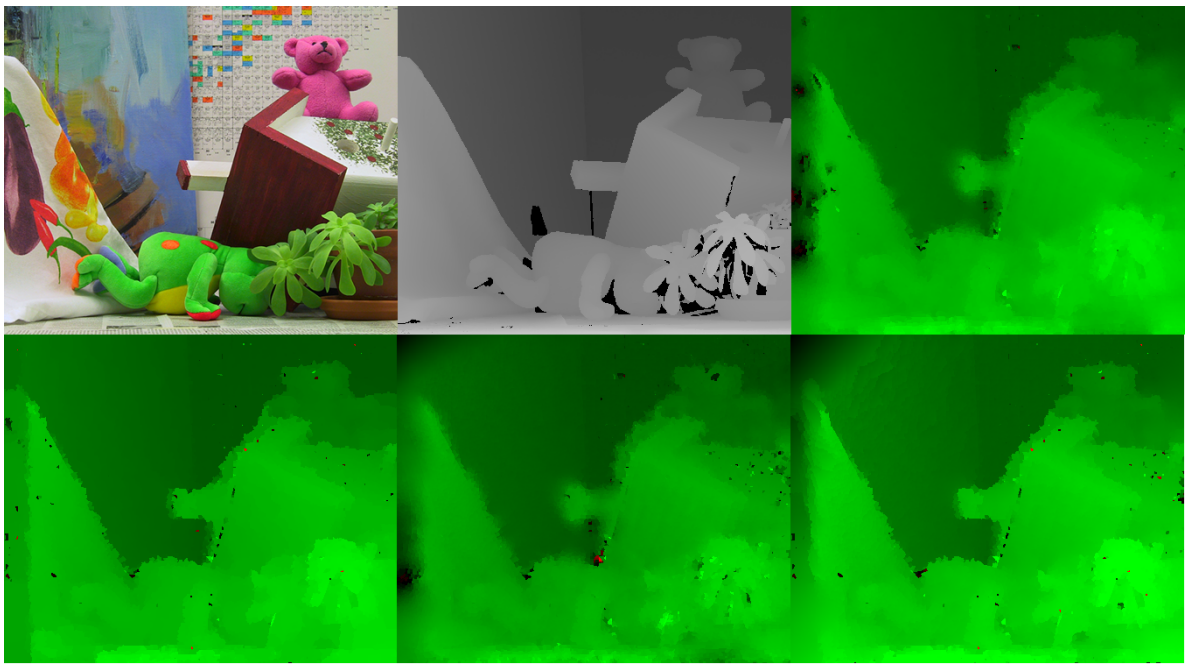


Abbildung 5.3: Oben Links: Teddy Testsequenz, **Oben Mitte:** Grount Truth, **Oben Rechts:** Konstanter Ansatz mit $\alpha = 86$, $\sigma = 0.3$ und einem Bad Pixel Error von **20.6%**, **Unten Links:** Robuster konstanter Ansatz mit $\alpha = 273$, $\sigma = 0.3$, $\epsilon_D = 0.05$, $\epsilon_S = 0.0017$ und einem Error von **14.3%**, **Unten Mitte:** Affiner Ansatz mit $\alpha = 6493815$, $\sigma = 0.3$ und einem Error von **22.2%**, **Unten Rechts:** Robuster affiner Ansatz mit $\alpha = 273841$, $\sigma = 0.3$, $\epsilon_D = 0.05$, $\epsilon_S = 0.00056$ und einem Error von **18.5%** [SS03].

Die Teddy Testsequenz ist von den 4 evaluierten Bildern diejenige, welche allgemein den größten Fehler hatte. Dies liegt daran, dass gerade im vorderen Bereich eine große Anzahl an okkludierten Pixel zu finden sind, welche durch die schwarzen Bereiche der Ground Truth dargestellt werden. Der konstante Ansatz hinterlässt dort ein sehr unscharfes Bild, wobei die Konturen des Bären und Teile der Pflanzen noch zu erkennen sind. Die Kanten werden jedoch erst durch die robuste Variante wieder etwas schärfer, wodurch der Kamin des Hauses besser dargestellt wird. Auch der Teddy kommt hier besser zum Vorschein. Die affine Methode hat auch hier Einfluss auf die etwas kleineren Bereiche, welche besser erhalten werden. Der Teddy der im konstanten Ansatz noch mit dem rechten Rand verschmolzen war, ist nun deutlich im Bereich des Armes von ihm getrennt. Im finalen Ansatz wird dies noch etwas fortgeführt. Hier werden Teile der Pflanze, welche zuvor noch ein großer Fleck waren, nun besser dargestellt und besser vom Hintergrund getrennt.

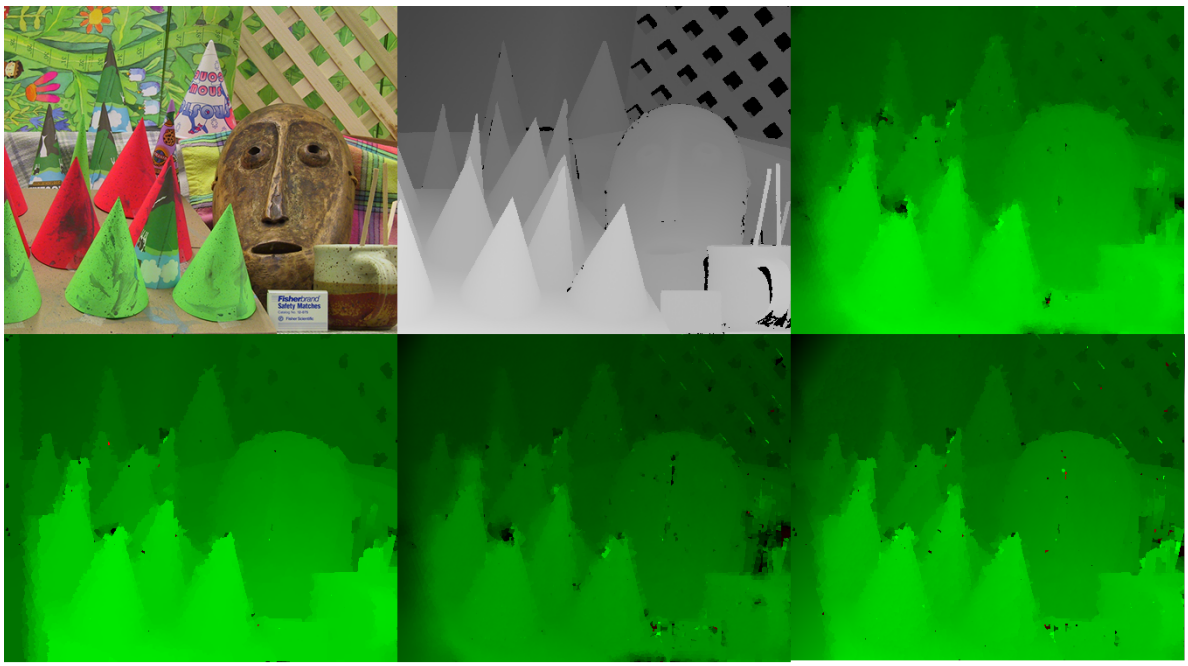


Abbildung 5.4: Oben Links: Cones Testsequenz, **Oben Mitte:** Grount Truth, **Oben Rechts:** Konstanter Ansatz mit α = 100, σ = 0.3 und einem Bad Pixel Error von **13.9%**, **Unten Links:**Robuster konstanter Ansatz mit α = 421, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.0017 und einem Error von **8.8%**, **Unten Mitte:** Affiner Ansatz mit α = 6493815, σ = 0.3 und einem Error von **15.34%**, **Unten Rechts:** Robuster affiner Ansatz mit α = 273841, σ = 0.3, ϵ_D = 0.05, ϵ_S = 0.00017 und einem Error von **10.7%** [SS03].

Cones besteht aus einer Menge kleiner Kegel, welche in vielen verschiedenen Tiefen platziert sind und zum Teil sich gegenseitig verdecken. Eines der Probleme war die Tasse im rechten unteren Bereich des Bildes. Die dünnen Bleistifte gingen leicht verloren und konnten von keinem der Ansätze korrekt zugeordnet werden. Der konstante Ansatz konnte jedoch bereits viele der Kegel der richtigen Tiefe zuordnen, leidet aber wie schon zuvor an unscharfen Kanten. Dies konnte durch den robusten Ansatz zum Teil behoben werden, wobei besonders die Maske sehr gut getroffen wurde. Die beiden affinen Ansätze bringen wieder die bereits Bekannten Vorteile mit sich, welche im hinteren Bereich des Zaunes zu erkennen sind. Viele Holzlatten welche zuvor miteinander verschmolzen waren, sind nun wieder zu erkennen.

Im Allgemeinen lässt sich über die einzelnen Varianten sagen, dass die subquadratische Bestrafungsfunktion einen sehr starken Einfluss auf das spätere Ergebnis hat. Die Kanten werden um ein vielfaches schärfer und die Flächen gleichmäßiger. Die affine Parametrisierung hat nur unter bestimmten Voraussetzungen ein positiven Einfluss, zum einen wenn es um dünne und kleine Objekte geht, die dadurch besser erhalten werden.

5.5 Middlebury Benchmark

Das Middlebury Benchmark [mid] ist eine Homepage der Computer Vision, welche es sich zur Aufgabe gemacht hat, verschiedene entwickelte Algorithmen im Bereich des Stereo Matchings oder des Optischen Flusses miteinander zu vergleichen. Zum Testen wurden mehrere auf das Problem angepasste Testsequenzen entwickelt, welche die Schwierigkeiten des Alltags repräsentieren sollen. Dort wird mithilfe der Ground Truth, Okklusions Map und anderen Statistiken eine Rangliste der dort hochgeladenen Algorithmen erstellt. Um für ein faires Ergebnis zu sorgen, müssen allerdings alle 4 Bilder mit den gleichen Parametern berechnet werden und dürfen nicht auf jedes Bild einzeln angepasst sein.

LCDM+AdaptWgt [68]	146.9	5.98 148	7.84 148	22.2 145	14.5 152	15.4 152	35.9 151	20.8 152	27.3 150	38.3 151	8.90 137	17.2 138	20.0 139	19.5
STICA [15]	147.0	7.70 151	9.63 152	27.8 149	8.19 148	9.58 148	40.3 152	15.8 142	23.2 144	37.7 150	9.80 139	17.8 139	28.7 150	19.7
YOUR METHOD	147.8	7.47 150	9.58 151	31.7 152	5.24 145	6.55 145	33.2 149	18.3 149	26.6 149	36.8 148	10.8 144	20.2 146	25.8 146	19.3
Infection [10]	148.4	7.95 152	9.54 150	28.9 151	4.41 143	5.53 143	31.7 148	17.7 146	25.1 148	44.4 152	14.3 149	21.3 148	38.0 151	20.7

Abbildung 5.5: Vergleich mit anderen Algorithmen der Middlebury Benchmark Homepage **Von links nach rechts:** Tsukuba, Venus, Teddy, Cones

Bild 5.5 zeigt nun einen kleinen Ausschnitt dieser Rangliste, in welcher dieser Algorithmus gelb gekennzeichnet ist. Die Wahl des Ansatzes fiel auf den robusten affinen Ansatz, da dieser das meiste Potential der Algorithmen hat und allgemein zufriedenstellende Ergebnisse erzielt. Die erste Spalte von links steht für den Rang, daneben ist die Analyse der Tsukuba Sequenz, gefolgt von Venus, Teddy und Cones. Die letzte Spalte steht für den durchschnittlichen Bad Pixel Error aller Tests. Es wurden 3 Tests für jedes der Bilder durchgeführt. Von links nach rechts ist dies zum einen der Vergleich mit einer Okklusions Map, bei welchem die Okkludierten Pixel nicht in den Fehler mit einfließen. Zum anderen ein Vergleich ohne Okklusions Map, bei der jeder Pixel Einfluss auf den Fehler hat, okkludiert oder nicht. Der letzte Test war auf die Kanten der jeweiligen Bilder fokussiert, hier wurden nur umliegende Bereich der Kanten gewertet. Zu erkennen ist, dass gerade bei den Kanten ein sehr hoher Fehler erzielt wird, welcher auch die Hauptproblematik vieler anderen Verfahren ist. Durch eine bessere Erhaltung der kanten, lässt sich somit ein weit besseres Ergebnis erzielen.

6 Zusammenfassung und Ausblick

Im letzten Kapitel dieser Arbeit werden die einzelnen Schritte des Algorithmus, sowie die Ergebnisse noch einmal zusammengefasst. Darüber hinaus wird im Ausblick etwas näher auf die zukünftige Entwicklung eingegangen, sowie ein paar Möglichkeiten angesprochen, welche den Algorithmus erweitern.

6.1 Zusammenfassung

Das Ziel dieser Arbeit war einen segmentationsgestützen Algorithmus zu entwerfen, welcher die räumliche Tiefe mithilfe eines Optischen Fluss Ansatzes berechnet. Angefangen wurde mit einem Orthoparallelen Kameraaufbau, welcher einzig Verschiebungen in der x-Achse zulässt, worauf zuerst der allgemeine Optische-Fluss-Ansatz auf Farbe und die angesprochene Stereoform umgestellt wurde, welche als Baseline für die nächsten Schritte diente. Diese konnte darauf in einen segmentierten Ansatz umgestellt werden, indem die Bilder zuvor mit der Wasserscheidentransformation segmentiert und die Objekte geclustert wurden. Da es sich um ein Minimierungsproblem handelt, musste das Funktional mithilfe des Euler-Lagrange Frameworks nach dem Fluss abgeleitet werden. Um jedoch größere Verschiebungen berechnen zu können, wurde die Lineariserung bis in den numerischen Teil hinausgezögert und alles in eine Warping-Strategie eingebaut, genauer gesagt in eine Coarse-to-Fine-Pyramide integriert. Anschließend wurde die Lösung schrittweise für verschieden große Auflösungen berechnet, wobei mit der gröbsten Auflösung begonnen wird und diese dann in die nächsthöhere Auflösung übertragen wird. Dadurch wurden die großen Verschiebung in viele Kleine aufgeteilt. Danach wurde alles durch einen iterativen SOR-Löser berechnet. Dieser Ansatz konnte mithilfe weiterer Konzepte allerdings verbessert werden. Das erste bestand darin, den Ansatz robuster gegen Ausreißer zu machen, dies geschah durch die Funktion Ψ, welche Fehler in der Zuordnung nicht länger quadratisch sondern subquadratisch bestraft, worauf Ausreißer an Einfluss verloren. Dies kam besonders der Erhaltung der Kanten zugute und erzeugte ein gleichmäßigeres Bild. Der letzte Schritt war die Umwandlung auf ein affines Modell, welches für jedes Segment eine Ebene mit den Parametern a, b und c berechnete und so den Fluss allgemeiner darstellte.

6.2 Ausblick

Die Richtung in die sich die Verfahren entwickeln geht klar zu den segmentierten Varianten, da viele Einsatzgebiete solcher Algorithmen einen echtzeitfähigen Algorithmus voraussetzen, also eine schnelle Berechnung erfordern, worauf zur Not eine nicht zu 100% korrekte Lösung in Kauf genommen wird.

Nichtsdestotrotz bestehen durchaus weitere Möglichkeiten, die Ergebnisse zu verbessern und trotzdem eine schnellen Algorithmus zu bekommen. Der affine Ansatz kann durch weitere Gewichtungen des Glattheitsterm besser auf die jeweiligen Parameter a, b, c angepasst werden, wodurch ein besseres Ergebnis entsteht. Zudem können Modifikationen am Datenterm vorgenommen werden, indem er durch eine Gradientenkonstanz erweitert wird, welche Pixel trotz Änderung der Beleuchtung richtig zuordnet. Eine weitere Möglichkeit wäre einen besseren Segmentierungsalgorithmus zu verwenden, wie den der Mean-Shift Segmentierung [CM02], welche nicht kantenbasiert, sondern regionsbasiert ist. Der größte Schritt wird allerdings durch die Erhaltung der Kanten erzielt, welche in diesem Algorithmus die größten Schwachstellen sind. Der hier entwickelte Algorithmus bietet aber eine solide Grundlage, welche in Zukunft weiter ausgebaut und auf verschiedene Anwendungsgebiete angepasst werden kann.

Literaturverzeichnis

[All08] M. Allen. Irrotationalfield. http://upload.wikimedia.org/wikipedia/commons/6/6d/Irrotationalfield.svg, 2008. (Zitiert auf Seite 25)

[Ape] Aperturproblem. http://www.offminor.de/files/pics/diplom/aperturproblem.png. (Zitiert auf Seite 28)

[Beu92] S. Beucher. The watershed transformation applied to image segmentation. *Scanning Microscopy-Supplement-*, S. 299–299, 1992. (Zitiert auf Seite 42)

[Bru13a] A. Bruhn. Vorlesungsscript Computer Vision. Universität Stuttgart, 2013. (Zitiert auf Seite 21)

[Bru13b] A. Bruhn. Vorlesungsscript Correspondence Problems in Computer Vision. Universität Suttgart, 2013. (Zitiert auf den Seiten 26 und 34)

[CM02] D. Comaniciu, P. Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002. (Zitiert auf Seite 68)

[Fra] H. Frank. RGB-Cube. http://upload.wikimedia.org/wikipedia/commons/0/03/RGB_farbwuerfel.jpg. (Zitiert auf Seite 20)

[Hem] W. A. Hemmerich. Normalverteilung. http://commons.wikimedia.org/wiki/File:Normalverteilung.svg. (Zitiert auf Seite 17)

[HS81] B. K. Horn, B. G. Schunck. Determining optical flow. In *Artificial Intelligence*, Band 17, S. 185–203. 1981. (Zitiert auf Seite 23)

[Jö09] F. Jörn. Subpixel. http://upload.wikimedia.org/wikipedia/commons/thumb/6/65/TVPixel.jpg/220px-TVPixel.jpg, 2009. (Zitiert auf Seite 19)

[LK81] B. D. Lucas, T. Kanade. An iterative image registration technique with an application to stereo vision. In *International journal of computer Vision*, Band 81, S. 674–679. 1981. (Zitiert auf Seite 23)

[mid] Middlebury Stereo Vision Page. vision.middlebury.edu/stereo/. (Zitiert auf Seite 65)

[SS02] D. Scharstein, 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. (Zitiert auf Seite 41)

[SS03] D. Scharstein, R. Szeliski. High-accuracy stereo depth maps using structured light. *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, 1:195–202, 2003. (Zitiert auf den Seiten 63 und 64)

[TSK01] H. Tao, H. S. Sawhney, R. Kumar. A global matching framework for stereo computation. In *IEEE International Conference on Computer Vision*, Band 1, S. 532–539. IEEE, 2001. (Zitiert auf den Seiten 9 und 10)

[WZ08] Z.-F. Wang, Z.-G. Zheng. A region based stereo matching algorithm using cooperative optimization. In *Computer Society, Conference on Computer Vision and Pattern Recognition*, S. 1–8. IEEE, 2008. (Zitiert auf den Seiten 8 und 9)

Alle URLs wurden zuletzt am 13. 05. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift