



HOCHSCHULE LANDSHUT
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN

FAKULTÄT INFORMATIK

Studienarbeit Bildverstehen

IMPLEMENTIERUNG EINES BINÄREN STEREO-MATCHING
ALGORITHMUS

Andreas Filinger

Inhaltsverzeichnis

1 Einleitung	2
1.1 Problemstellung	2
1.2 BRIEF-Deskriptor	2
2 Implementierung	3
2.1 Sprache und Umgebung	3
2.2 Parameter	3
2.3 Datenstruktur für den Deskriptor	3
2.4 Deskriptorberechnung	4
2.5 Versatzbestimmung	4
2.6 Parallelisierung	4
2.7 Ergebnisaufbereitung	5
3 Ergebnisse	6
3.1 Vergleich	6
3.2 Laufzeit	6
Abbildungsverzeichnis	14
Tabellenverzeichnis	15

1 Einleitung

1.1 Problemstellung

Zhang et al. [ZLL⁺12] entwickelten einen Algorithmus zur Lösung des Korrespondenzproblems bei rektifizierten Stereobildern. Das Ziel dabei ist es, 3D-Tiefeninformationen aus den Bildern zu erhalten, indem korrespondierende Pixelpaare aus beiden Bildern gefunden werden. Dazu verwendeten sie einen Binären Deskriptor, den BRIEF-Deskriptor [CLSF10]. Im Rahmen dieser Studienarbeit sollte ein Teil des Algorithmus implementiert werden (d.h. weitere Verfeinerungen der Ergebnisse, wie sie im Paper beschrieben sind, wurden nicht verlangt).

1.2 BRIEF-Deskriptor

Ein Deskriptor der sich nur aus Bits zusammensetzt, statt aus Zahlen. Dies ist vorteilhaft in Hinblick auf Speicherbedarf und Geschwindigkeit der Kostenberechnung. Es werden sowohl bei der Berechnung als auch beim Vergleich nur Operationen auf binäre Strings benötigt. Bei der Berechnung vergleicht man eine Menge an zufällig gewählten Pixeln aus der Nachbarschaft: Jeder Vergleich entspricht einem Bit des Deskriptors. Für die Kostenberechnung nimmt man die Hammingdistanz her: Hierfür braucht man ausschließlich XOR-und POPCOUNT-Operationen. Dies ist relativ schnell, besonders wenn entsprechende CPU-Befehle ausgenutzt werden können.

2 Implementierung

2.1 Sprache und Umgebung

Um XOR und POPCOUNT Funktionen für die Kostenberechnung möglichst schnell durchführen zu können wurde die Implementierung in C++ durchgeführt. Weiterhin wurde die openCV-Bibliothek für grundlegende Operationen auf Bilddateien verwendet (öffnen, speichern, bearbeiten).

2.2 Parameter

Zusätzlich zu den Eingabebildern können folgende Parameter frei gewählt werden:

Parameter	Im Paper	Beschreibung
DESCRIPTORBITS	4096	Größe eines Deskriptors in Bits
WINDOWSIZE	26	Größe des Fensters bei der Berechnung des Deskriptors
GAUSS_SIGMA	4	Gaussparameter bei der Berechnung zufälliger Nachbarpunkte

2.3 Datenstruktur für den Deskriptor

Der Deskriptor kann wegen seiner Größe nicht als einzelne Zahl dargestellt werden (im Paper beträgt die Größe 4096 Bits). Er kann jedoch in eine Menge aus Integern zerlegt werden, die Hammingdistanz zweier Deskriptoren ist dann die Summe der Hammingdistanzen der einzelnen Teilzahlen. Um möglichst wenige XOR/POPCOUNT-Operationen zu brauchen wurde der größte unsigned-Integertyp verwendet, der Deskriptor wird dann dargestellt als:

```
struct brief {
    uint64_t elems[ELEMS_PER_DESCRIPTOR];
};
```

2.4 Deskriptorberechnung

Wie in Abschnitt 1.2 beschrieben wird der Deskriptor gebildet, indem eine Menge an Pixelpaaren um den aktuellen Pixel herum verglichen wird. Bevor also Deskriptoren berechnet werden können, muss eine Menge an zufälligen Punkten um das Fensterzentrum bestimmt werden. Dabei wird im Paper eine Isotropische Gaussverteilung verwendet, um die Gewichte der zufälligen Punkte festzulegen. Weil das Fenster quadratisch ist, wurde in dieser Implementierung eine 1D-Gaussverteilung verwendet, um eine Menge zufälliger 1D-Koordinaten zu berechnen, die der Anzahl benötigter Punktpaare entspricht (also vier Koordinaten je Punktpaar). Sind die Punkte vorhanden (Samplepoints), kann der Deskriptor für beliebige Pixel berechnet werden.

2.5 Versatzbestimmung

Ziel des Algorithmus ist es zu jedem Pixel des linken Bildes den entsprechenden Pixel im rechten Bild zu finden und ihren Versatz zu bestimmen. Dazu berechnet man die Kosten zwischen zwei Pixeln, welche der Hammingdistanz ihrer Deskriptoren entspricht. Es besteht dann eine Korrespondez zwischen zwei Pixeln, wenn ihre Kosten minimal sind. Da die Bilder metrisch rektifiziert sind, reicht es aus nur Pixel in der selben Zeile zu vergleichen. Zusätzlich ist bekannt, welches das Linke und welches das Rechte Bild ist, daher muss auch nicht immer die ganze Zeile des rechten Bildes abgesucht werden. Wie in Abschnitt 2.3 beschrieben setzt sich die Hammingdistanz zweier Deskriptoren zusammen aus der Summe der Hammingdistanzen ihrer Teilelemente. Die Bestimmung der Hammingdistanz zweier Zahlen benötigt zwei Operationen: XOR und POPCOUNT (= Anzahl der 1-Bits). Ist die minimale Hammingdistanz eines Pixels des linken Bildes gefunden, so ist der Versatz die Differenz der Spaltenindizes beider Pixel. Dieser wird in das Ergebnisarray eingetragen, welches zu einem Bild zusammengefügt eine Depthmap und damit das Ergebnis des Algorithmus bildet.

2.6 Parallelisierung

Um auch größere Bilder verarbeiten zu können bietet es sich an mehrere Threads zu verwenden. Dazu wurden Funktionen implementiert, die den Algorithmus jeweils

für eine Zeile durchführen. Jeder Thread kann dann eine eigene Menge an Zeilen abarbeiten.

2.7 Ergebnisaufbereitung

Das Array aus den Pixelabständen wird zum Schluss als PNG-Bilddatei abgespeichert. Anschließend wird das Bild mit der opencv-Funktion 'equalizeHist' für Menschliche Betrachter aufbereitet und als eine zweite PNG-Bilddatei gespeichert.

3 Ergebnisse

3.1 Vergleich

Die Autoren des Papers haben ihren Algorithmus an Daten der 'Middlebury Stereo Vision'-Seite [SB] getestet und dort ihre Ergebnisse hochgeladen. Außerdem sind dort auch die verwendeten Eingabebilder mit den entsprechenden Ground-Truth Depthmaps zu finden. In den folgenden Abbildungen werden diese Bilder und die Ergebnisse der eigenen Implementierung nebeneinandergestellt. Anzumerken ist hierbei dass die eigene Implementierung nicht den gesamten Algorithmus umfasst, nur zu dem Testbild 'cones' ist ein vergleichbares Teilergebnis der Autoren vorhanden. Die verwendeten Parameter entsprechen denen der Autoren (soweit möglich).

3.2 Laufzeit

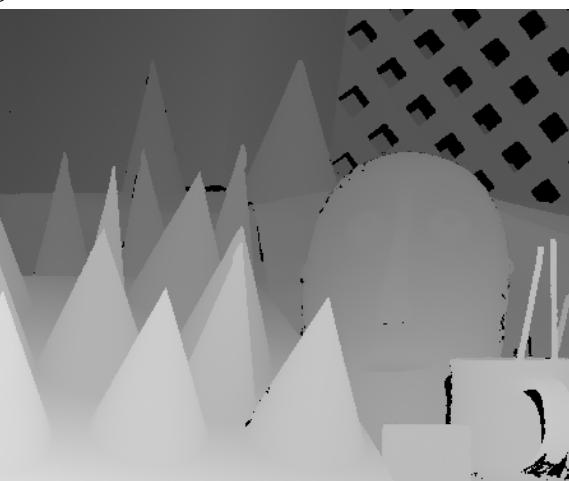
Im Paper ist die Laufzeit des ganzen Algorithmus für das Bild 'teddy' (450x375) angegeben als 50s auf einem einzigen Thread mit einer CPU-Frequenz von 2.67 GHz. Folgende Tabelle listet die Laufzeiten der eigenen (teilweisen) Implementierung an Bildern unterschiedlicher Auflösungen auf. Die CPU-Frequenz beträgt 2.8 GHz, bei der Verwendung mehrerer Threads werden 8 logische Kerne genutzt.

Bild	Auflösung	Laufzeit in Sekunden	Laufzeit in Sekunden (Multithreaded)
monopoly	443x370	13.7	3.6
teddy	450x375	14.6	3.9
shopvac	2356x1996	-	383
classroom	3000x1920	-	582

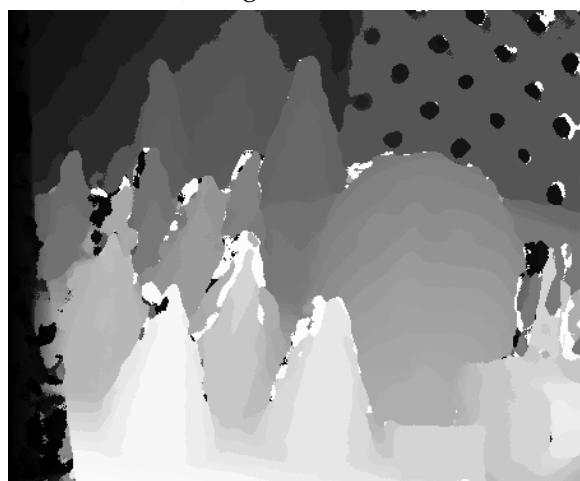
Abb. 3.1: Vergleich 'cones'



(a) Eingabebild Links



(b) Ground Truth



(c) Eigenes Ergebnis



(d) Teilergebnis im Paper

Abb. 3.2: Vergleich 'teddy'

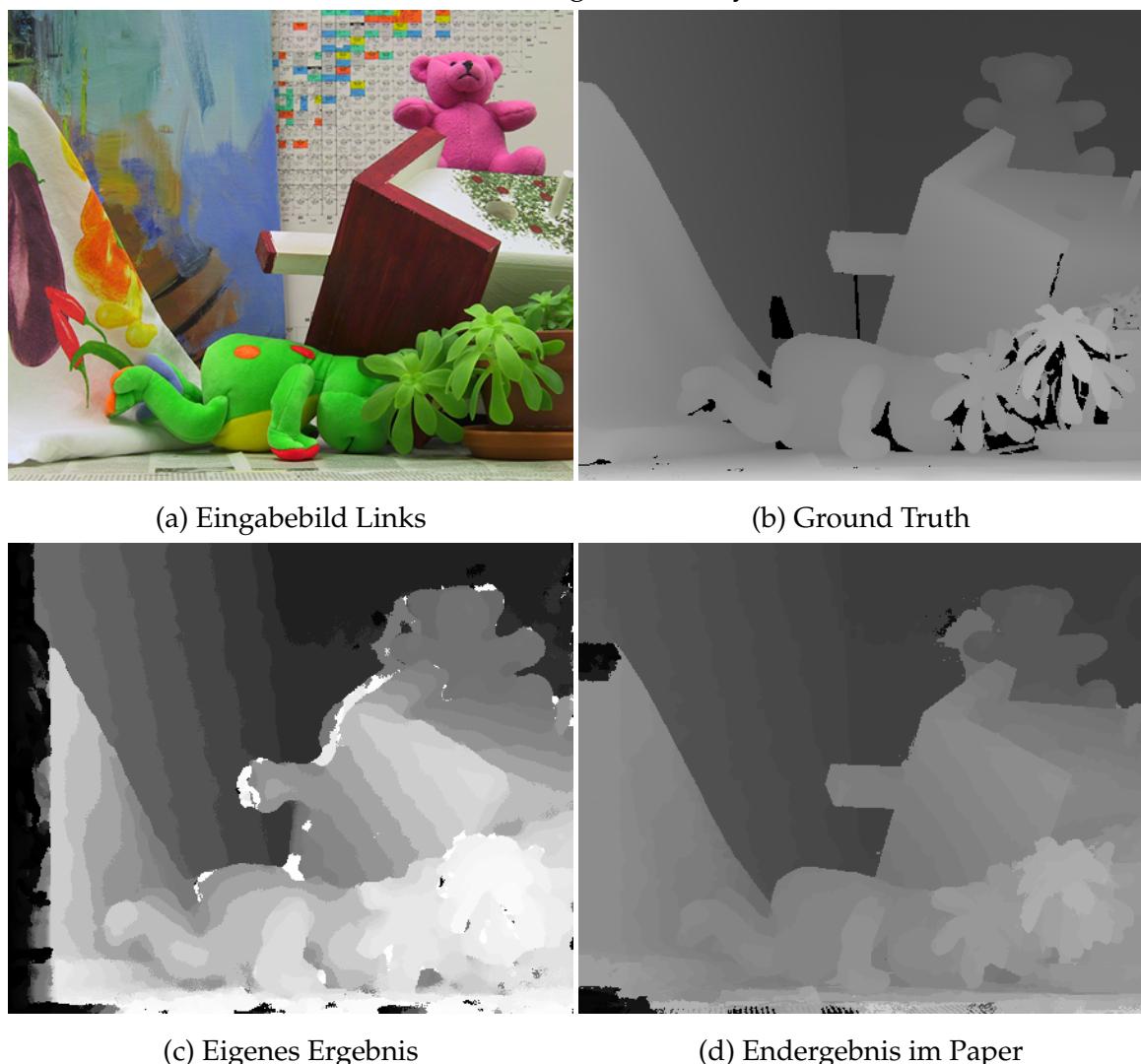


Abb. 3.3: Vergleich 'tsukuba'

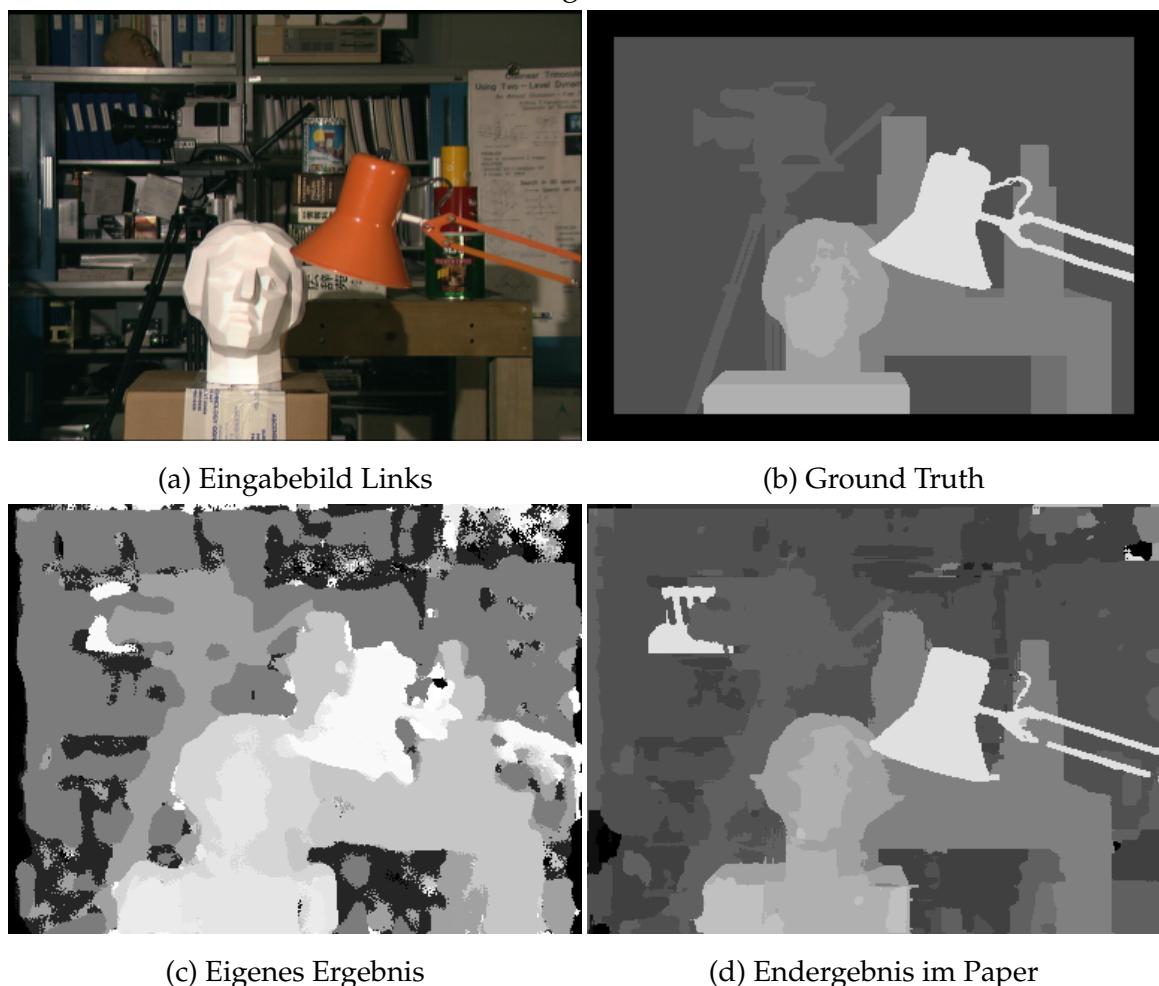


Abb. 3.4: Vergleich 'venus'

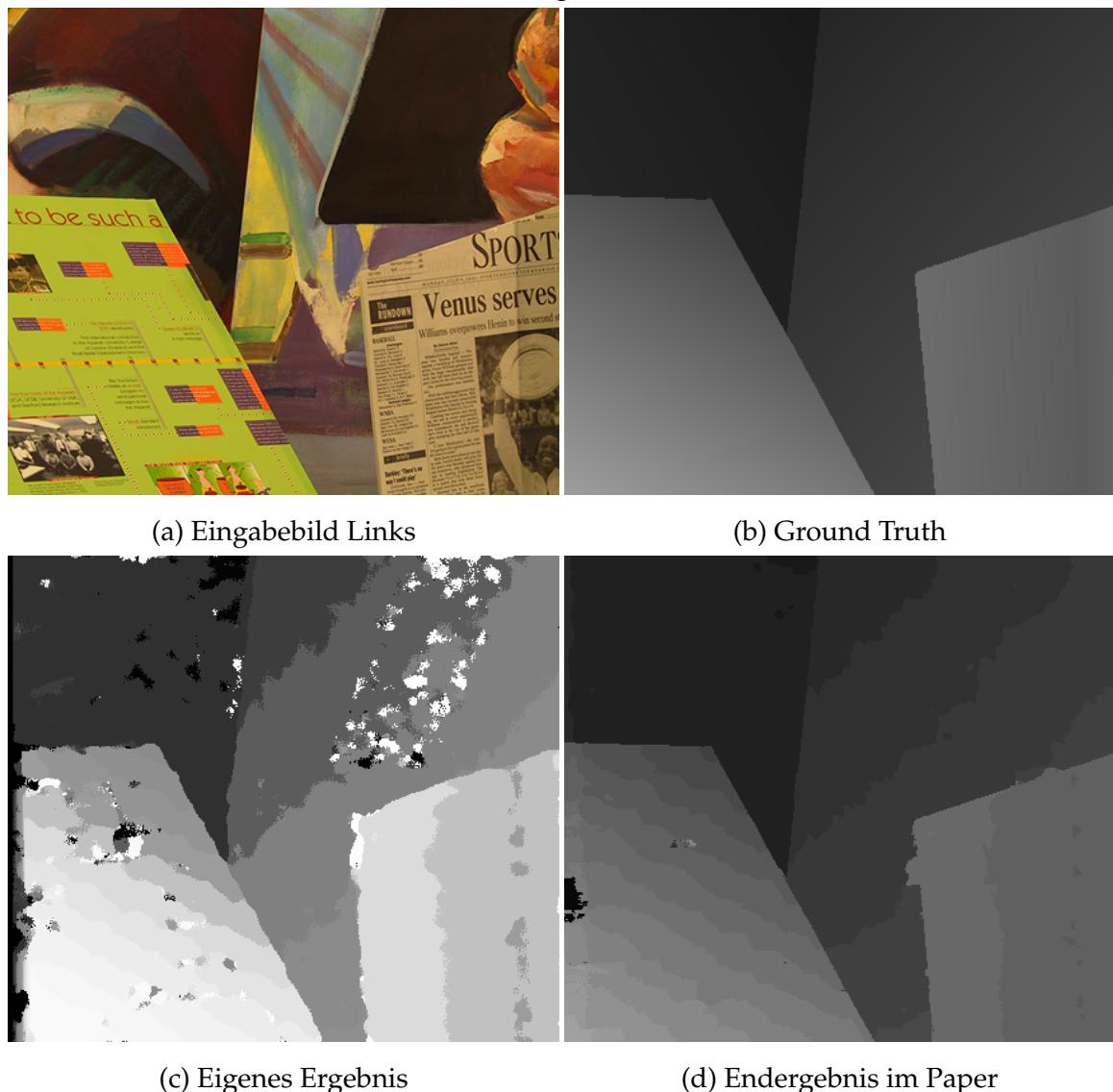


Abb. 3.5: Ergebnis 'shopvac'



(a) Eingabebild Links



(b) Eigenes Ergebnis

Abb. 3.6: Ergebnis 'classroom'



(a) Eingabebild Links



(b) Eigenes Ergebnis

Literaturverzeichnis

- [CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua.
Brief: Binary robust independent elementary features. In *ECCV*, 2010.
- [SB] Daniel Scharstein and Anna Blasik. Middlebury stereo vision, zuletzt
zugegriffen: 17.01.2020. <http://vision.middlebury.edu/stereo/>.
- [ZLL⁺12] Kang Zhang, Jiyang Li, Yijing Li, Weidong Hu, Lifeng Sun, and Shiqiang Yang. Binary stereo matching. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 356–359, 2012.

Abbildungsverzeichnis

3.1	Vergleich 'cones'	7
3.2	Vergleich 'teddy'	8
3.3	Vergleich 'tsukuba'	9
3.4	Vergleich 'venus'	10
3.5	Ergebnis 'shopvac'	11
3.6	Ergebnis 'classroom'	12

Tabellenverzeichnis