RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN
INSTITUT FÜR INFORMATIK III
COMPUTER VISION AND PATTERN RECOGNITION GROUP

**Diplomarbeit**

# Shape Matching mittels Graph Cuts

**Eno Töppe**

Erstgutachter: Prof. Dr. Daniel Cremers
Zweitgutachter: Prof. Dr. Reinhard Klein
Betreuer: Frank Schmidt

# Danksagung

Ich danke Prof. Dr. Daniel Cremers für meine Einbeziehung in aktuelle Forschungsarbeiten und die Möglichkeit, deren Thematik im Rahmen dieser Arbeit behandeln zu können sowie für seine wertvollen Hinweise zur schriftlichen Ausarbeitung. Ich danke Frank Schmidt für die kooperative, interessante und produktive Zusammenarbeit sowie seine äußerst hilfreiche Betreuung beim Entstehen dieser Arbeit. Des Weiteren danke ich Prof. Dr. Reinhard Klein für seine Bereitschaft, die Arbeit als Zweitgutachter zu betreuen.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Diese Arbeit beschäftigt sich mit Shape Matching. Es wird ein Graph Cut Verfahren vorgestellt, welches punktweise Korrespondenzen auf zwei Formen findet. Des Weiteren werden Verfahren zur Berechnung des minimalen Cuts auf dem konstruierten Graphen getestet und gezeigt, dass die Laufzeiten vergleichbare Verfahren mittels dynamischer Programmierung übertreffen.

## 1.1 Shape Matching und Graph Cuts

Das Shape Matching gehört zum Gebiet der Formenanalyse. Die Formenanalyse hat die systematische Beschreibung von Formen zum Gegenstand. Ziel des Shape Matching ist es, auf zwei verschiedenen Formen korrespondierende Stellen oder Teile zu identifizieren. Das Problem ist als sehr schwierig einzustufen. Die Schwierigkeit beginnt bereits bei der Definition des Formbegriffs. Die Antwort auf die Frage nach der Sinnhaftigkeit von Korrespondenzen zwischen Formteilen fußt auf vielen unterschiedlichen und schwer subsumierbaren Kriterien. Diese setzen eine Interpretation der Formen voraus, die beim Menschen stark auf Vorwissen beruht. Im überwiegenden Teil der Fälle - sowie in dieser Arbeit - werden daher generische Vergleichsmethoden betrachtet, die lediglich die geometrische Beschaffenheit von Formen in die Berechnung einfließen lassen. Eng verbunden mit dem Problem der Korrespondenzfindung ist das der Ähnlichkeit von Formen. Dabei soll ein Maß dafür gefunden werden, wie verschieden Formen voneinander sind.
Die Anwendungen von Shape Matching und Formendistanz sind zahlreich. Die wohl wichtigsten Bereiche sind die Objekterkennung in Bilddaten, Anfragen in Formendatenbanken, die Berechnung von Formklassen, sowie Tracking in Bildsequenzen und die Berechnung glatter Übergänge zwischen Formen.

Viele Probleme in der Bildverarbeitung werden als Optimierungsproblem formuliert. Um Lösungen zu finden werden daher geeignete Optimierungsverfahren benötigt. Ein in der Bildverarbeitung mittlerweile sehr beliebtes Verfahren zur dis-

kreten, globalen Optimierung stellen die Graph Cuts dar. Berechnungsmethoden für Graph Cuts wurden bereits in den sechziger Jahren von Ford und Fulkerson entwickelt [11]; das bahnbrechende Max-Flow / Min-Cut Theorem wurde sogar - sowohl von Ford und Fulkerson als auch von Kotzig - bereits 1956 entdeckt. Seitdem gab es viele Anstrengungen, die Berechnung des maximalen Flusses auf einem Graphen zu beschleunigen. Um die Jahrtausendwende etablierten sich Graph Cuts im Kontext der Bildverarbeitung hauptsächlich durch die Arbeiten von Y. Boykov, V. Kolmogorov und Mitarbeitern [6], [5], [19], nachdem bereits zehn Jahre zuvor Pionierarbeit geleistet worden war [15]. Graph Cuts wurden unter anderem im Kontext von Bildrestaurierung, Bildsegmentierung, Bewegungssegmentierung und 3D-Rekonstruktion eingesetzt, bisher jedoch noch nicht für Shape Matching.

## 1.2 Shape Matching - Stand der Forschung

In der Bildverarbeitung wurden bisher sehr viele unterschiedliche Ansätze zum Formenvergleich entwickelt. Eine große Gruppe bilden solche Verfahren, die Korrespondenzen zwischen Punkten oder Konturabschnitten zweier Formen herstellen z.B. [7], [14], [23]. Daneben gibt es andere Ansätze, wie z.B. den teilbasierten [31] oder skelettbasierten Verfahren [24], die jedoch in dieser Arbeit nicht betrachtet werden.

Die meisten punktweisen Shape Matching Algorithmen definieren eine spezielle Energiefunktion, die jeder Korrespondenzabbildung zwischen den Formen einen Wert zuweist. Die Korrespondenzabbildung mit der kleinsten Energie wird als optimal angesehen. Die Optimierung über die Korrespondenzabbildungen wird entweder global über dynamische Programmierung [21] oder über Gradientenabstiegsmethoden [7] durchgeführt, die allerdings nicht notwendig in einem globalen Minimum enden.

Um die Berechnung von Korrespondenzen beim Formenvergleich beherrschbar zu halten, wird oft davon ausgegangen, dass die Korrespondenzabbildung die Ordnung der Punkte entlang der Konturen erhält. Bei geschlossenen Konturen entsteht jedoch dabei das Problem, dass eine feste Anfangskorrespondenz gefunden werden muss, von der die Berechnung ausgeht. Viele Algorithmen versuchen dieses Problem durch Heuristiken zu umgehen, eine ganzheitliche, effiziente Lösung existiert jedoch bisher nicht.

Eng verbunden mit diesem Problem ist ein weiterer Forschungsschwerpunkt: die Laufzeit. Für viele Anwendungen von Shape Matching Methoden ist eine hohe Effizienz unabdingbar. Das betrifft sowohl Datenbankanwendungen, die auch in Hinblick auf das Internet immer wichtiger werden, als auch den Einsatz in Echtzeitsystemen. Der Vergleich geschlossener Konturen benötigt bei einem Großteil der Verfahren immer noch kubische Laufzeit, weil eine Anfangskorrespondenz ermittelt werden muss.

## 1.3 Beitrag dieser Arbeit

Die wesentlichen Beträge dieser Arbeit sind zum einen, Shape Matching über Graph Cuts durchzuführen; hauptsächlich wird damit die Suche nach der Anfangskorrespondenz obsolet. Zum anderen werden effiziente Methoden zur Berechnung des Minimalen Cuts getestet.

Ausgangspunkt ist ein neu vorgestelltes punktweises Shape Matching Verfahren, das über dynamische Programmierung global optimiert wird. Der Algorithmus besitzt für geschlossene Konturen eine kubische Laufzeit. Das Problem wurde anschließend zu einem Graph Cut-Problem transformiert. Dazu wurde ein planarer Graph konstruiert, dessen minimaler Cut die Punktkorrespondenzen kodiert und dessen maximaler Fluss die Distanz der Formen wiedergibt.

Anschließend wurde die Laufzeit des Graph Cut Verfahrens untersucht. Diese hängt vom unterliegenden Algorithmus zur Berechnung des maximalen Flusses ab. Es wurden zwei Algorithmen getestet: der in der Bildverarbeitung beliebte Algorithmus von Boykov und Kolmogorov [6], der im Mittel zu einer kubischen Laufzeit führte und ein $O(n \log(n))$-Verfahren speziell für planare Graphen von G. Borradaile und P. N. Klein [4]. Dieser ergab eine Laufzeit von $O(n^2 \log(n))$ in der Anzahl der Abtastpunkte entlang der Konturen. Außerdem resultiert die Graph Cut-Formulierung in einer Abhängigkeit von Berechnungsdauer und Ähnlichkeit der Formen. Im Falle von Boykov und Komlogorov war diese wesentlich ausgeprägter.

Ein weiterer Beitrag dieser Arbeit ist die Verbesserung der Krümmungsapproximation der Integralinvarianten von Manay et.al. [20].

## 1.4 Gliederung dieser Arbeit

In **Kapitel 2** wird zunächst eine Definition von Form gegeben, wie sie im Kontext dieser Arbeit verwendet wird. Außerdem werden Repräsentationsmethoden besprochen, die invariant gegenüber bestimmten Gruppentransformationen sind. Dies bietet die Grundlage für die Konstruktion von Shape Matching Algorithmen. Im **Kapitel 3** werden grundlegende Verfahrensweisen zur Berechnung von punktweisen Korrespondenzen zwischen Formen anhand existierender Arbeiten eingeführt und insbesondere die Vorteile lokaler Ansätze als auch die Nachteile von Gradientenabstiegsmethoden in diesem Kontext aufgezeigt. In **Kapitel 4** werden daher Shape Matching Algorithmen mittels dynamischer Programmierung eingeführt. Diese bilden den Ausgangspunkt zur Vorstellung eines neuen Vergleichsmodells in **Kapitel 5**. Laufzeittests des neuen Verfahrens schließen das Kapitel ab. Der vorgestellte Ansatz wird in **Kapitel 6** in eine Graph Cut-Formulierung transformiert

und deren beider Äquivalenz bewiesen. **Kapitel 7** beschäftigt sich schließlich mit der Frage nach geeigneten Methoden zur Berechnung des maximalen Flusses auf dem Graphen. Das Augenmerk liegt dabei auf Algorithmen für planare Graphen. Laufzeittests untermauern die theoretischen Überlegungen.

# Kapitel 2

# Form: Definition und Repräsentation

Dieses Kapitel legt die Grundlagen für alle weiteren Ausführungen. In Abschnitt 2.1 wird definiert, was im Kontext dieser Arbeit unter Form verstanden wird. Des Weiteren werden sowohl kontinuierliche als auch diskrete Repräsentationsformen hergeleitet. In Abschnitt 2.2 wird schließlich auf die praktische Realisierung der vorgestellten Repräsentationsformen eingegangen, die auch im weiteren Verlauf der Arbeit Verwendung findet. Die vorgestellte Methode stammt von Manay et. al. [20], sie wird in dieser Arbeit jedoch verfeinert.

## 2.1 Formbegriff

Die Definition von Form hängt von der Zielsetzung ab, und es sind sicherlich verschiedene Ansätze sinnvoll. Nach Kendall (1977) ist Form die gesamte geometrische Information, die übrig bleibt, wenn man Orts-, Skalierungs- und Rotationseffekte aus einem Objekt herausfiltert (vgl. [9]). An dieser Definition orientiert sich auch die vorliegende Arbeit. Im Folgenden werden zunächst sämtliche Erscheinungen einer Form getrennt betrachtet. Im zweiten Unterabschnitt werden solche Formen identifiziert, die sich durch rigide Transformationen ineinander überführen lassen.

### 2.1.1 Beschreibung als Kurve

In der Bildverarbeitung gehen Formen gemeinhin aus einer Bildsegmentierung hervor, die ein Objekt vom Hintergrund trennen soll. Das Resultat ist meist eine geschlossene, einfache Kontur. Kontinuierlich betrachtet sind konkrete Formen in dieser Arbeit daher stets zweimal stetig differenzierbare, geschlossene, einfache Kurven in der Ebene

$$c : [0, L] \rightarrow \mathbb{R}^2$$

$L$ ist dabei die Bogenlänge der Kurve. *Einfach* ist gleichbedeutend mit $c(s) \neq c(t)$, $\forall s, t \in [0, L]$ geschlossen ist eine Kurve, falls $c(0) = c(L)$ gilt. Die erste Ableitung von $c$ beschreibt die Funktion der *Tangentenrichtungen*, die zweite die der *Kurvenkrümmungen* $\kappa(s)$. Die Menge sämtlicher Kurven dargelegter Art, wird hier mit $C$ bezeichnet.
Meist beschreibt die Kurvenfunktion die äußere Silhouette eines Objektes und es wird davon ausgegangen, dass diese im Uhrzeigersinn abgetastet wird. Um Probleme zu umgehen, die bei der Einführung von Matchingmodellen auf *geschlossenen* Konturen auftreten, wird oft stattdessen zunächst der Raum der *offenen* Kurven betrachtet, der jedoch bis auf die Geschlossenheit äquivalent definiert ist.

### 2.1.2 Abstraktion von rigiden Transformationen

Zwei Kurven, von denen die erste eine translierte, skalierte und rotierte Version der anderen ist, sind unterschiedliche Elemente der Menge $C$. Menschen hingegen würden beide Kurven als identische Formen auffassen.
Die Gruppe der *Ähnlichkeitstransformationen* $S(2)$ bezeichnet Abbildungen der Form:

$$f(x) = sRx + t \text{ mit } R \in O(2),\ x, t \in \mathbb{R}^2\ s \in \mathbb{R}$$

$R$ beschreibt die Rotation, $t$ den Translationsvektor und $s$ den Skalierungsfaktor. Für eine gegebene Kurve $c \in C$ ist die zugehörige Form nun definiert als die Menge

$$[c] := \left\{ A(c) \;\middle|\; A \in S(2) \right\} \tag{2.1}$$

und umfasst damit sämtliche transformierte Versionen von sich selbst. Für eine *offene* Kurve ist die zugehörige Form analog definiert.

### 2.1.3 Repräsentationsformen

Die formale Definition einer Form ist unhandlich. Eine kompakte Repräsentationsform wird für die Definition von geeigneten Energiefunktionalen sowie die Berechnung von Lösungen benötigt.

#### Kontinuierlich

In mathematischen Formulierungen wird daher die *Krümmungsfunktion*

$$\kappa : [0, 1] \to \mathbb{R} \tag{2.2}$$

für die Repräsentation einer Form verwendet. Die Krümmung an einem Punkt einer Kurve ist unabhängig von deren Rotation und Position. Um Skalierung auszublenden wird von einer Kurvenlänge von 1 ausgegangen und die Kurve nach Bogenlänge parametrisiert. Durch $\kappa$ wird die Kurve im Sinne einer Form *eindeutig* beschrieben.

**Diskret**

Für die praktische Berechnung von Korrespondenzabbildungen wird eine diskrete Kurvenrepräsentation verwendet. Je nach Vergleichsmodell kommen verschiedene Repräsentationsformen in Frage. Eine Möglichkeit ist, die Kurve durch ein Polygon zu approximieren. Stellen der Kurve mit hoher Krümmung werden durch die Polygonecken repräsentiert und mit geraden Linien verbunden. Nachteilig ist vor allem die Rotations- und Skalierungsinvarianz einer solchen Repräsentation.

Für die Anwendungen dieser Arbeit werden Kurven in regelmäßigen Abständen abgetastet und die Krümmung an den entsprechenden Stellen verwendet. Formal wird die Kurve durch die Funktion

$$\kappa_d : [0..n] \rightarrow \mathbb{R}$$

repräsentiert, die den $n$ abgetasteten Punkten entlang der Kurve die Krümmungen zuordnet. Bleibt die Frage, wie die Krümmungen für jeden Punkt berechnet werden können. Eine Möglichkeit dafür wird im nächsten Abschnitt besprochen.

### 2.1.4 Invarianten

Man nennt Darstellungsformen einer Klasse von Objekten *Invarianten*, falls sie unveränderlich gegenüber bestimmten Transformationen sind. Meist wird Invarianz gegenüber einer Gruppe von Transformationen gefordert (beispielsweise der projektiven Gruppe). In der Bildverarbeitung entstammen Invarianten der Idee, geometrische Objekte auch dann noch wieder erkennbar zu machen, nachdem sie durch den komplexen Bildgenerierungsprozess verzerrt wurden. Invariante Repräsentation von Objekten kann ganzheitlich (*global*), oder auf der Basis von lokalen Merkmalen geschehen (*lokal*). Idealerweise sind Invarianten *kompakt* in der Darstellung, *diskriminativ* und *robust* gegenüber Störfaktoren. Ein Überblick über die Geschichte der Invarianten wird in [20] gegeben.
Die im letzten Abschnitt vorgestellte *Krümmungsfunktion* ist invariant gegenüber der Gruppe der Ähnlichkeitsabbildungen und - da sie die Kurve punktweise beschreibt - lokal.
Lokale Invarianten sind die Voraussetzung für elastische Matchingverfahren, wie sie in den folgenden Kapiteln besprochen werden. Daher wird im nächsten Abschnitt eine robuste Invariante eingeführt, die auch in den Experimenten dieser Arbeit Verwendung findet.

## 2.2 Stabile Krümmungsapproximation

Die Krümmung gehört als zweite Ableitung der Kurve zu den *differentiellen Invarianten*. Differentielle Invarianten sind sehr anfällig gegenüber verrauschten Daten.

Schon eine leichte Störung der Punktumgebung kann zu einer ganz anderen Krümmung am Punkt selbst führen. Daher werden in diesem Abschnitt lokale Invarianten eingeführt, die sowohl robust gegenüber Rauschen sind, als auch Aufschluss über die Krümmung geben können.

### 2.2.1 Integralinvarianten

Integralinvarianten wurden 2006 von Manay et. al. [20] ausführlich untersucht. Sie haben die allgemeine Form

$$I_c(p) = \int f(p,x)dx$$

wobei $p$ auf der geschlossenen Kurve $c$ liegt. Der Integrationsbereich kann beliebig gewählt werden; naheliegend ist die Integration über die Fläche der Kurve $c$ oder über die Kurve selbst. Für eine Integralinvariante muss gelten:

$$\int_A f(p,x)dx = \int_{gA} f(gp,x)dx \quad \forall g \in G$$

d.h. die Funktion an einem gegebenen Punkt $p$ auf der Kurve $c$ ist unveränderlich gegenüber Anwendung eines Elementes $g$ aus der Gruppe $G$ zulässiger Transformationen. Für $f$ kann in der kontinuierlichen Formulierung prinzipiell jede integrierbare Funktion verwendet werden. Die von Manay et. al. vorgeschlagene Integralinvariante ist die Schnittmenge des Kurveninneren und einer Kreisscheibe mit festem Radius um den Punkt $p$.

$$I_c^r(p) = \int_{int(c)} w(p,x) f_r(p,x)dx \quad \text{mit} \quad f_r(p,x) = \begin{cases} 1 & |p-x| \leq r \\ 0 & \text{sonst} \end{cases} \tag{2.3}$$

$int(c)$ bezeichnet das Innere der Kurve $c$. Die Gewichtungsfunktion $w(p,x)$ kann im simpelsten Fall die konstante Funktion $w(p,x) = 1$ sein; um die Kurvenfläche weiter entfernt von $p$, graduell weniger einfließen zu lassen, wird im Folgenden ein Gausskern $w(p,x) = \frac{1}{2\pi\sigma} \cdot exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{2\sigma^2}\right)$ gewählt. Dadurch wird der lokale Charakter der Invariante stärker betont.
Es ist klar, dass die Funktion $I_c^r$ per Konstruktion invariant gegenüber Rotation und Translation ist. Skalierungsinvarianz muss differenziert betrachtet werden. Zwar lässt sich die gemessene Fläche normieren, indem (2.3) durch $\int_{\mathbb{R}^2} w(p,x)dx$ geteilt wird, so dass die Integralinvariante im Intervall $[0,1]$ liegt. Dennoch hängt der Integralwert bei festem Radius von der Skalierung der Kurve $c$ ab. Streng genommen müssen Radius $r$ und Kurvenskalierung also aneinander angepasst werden. In der Praxis ist die normierte Invariante jedoch relativ robust gegen Skalierung, wie in Kapitel 5 zu sehen sein wird. Die Flächeninvariante beschreibt ihre lokale Umgebung wohlgemerkt bei weitem nicht eindeutig.

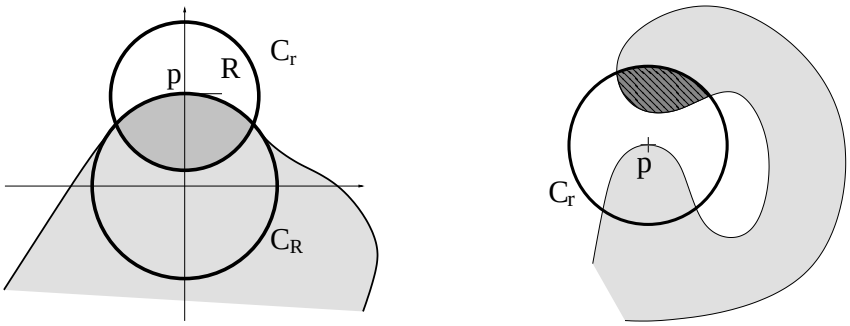



Abbildung 2.1: *links*: Ausgangssituation für die Krümmungsapproximation. Die Integralinvariante misst die Fläche $A_r$. *rechts*: Fehler bei der Berechnung der Integralinvariante. Der schraffierte Bereich geht fälschlich in die gemessene Fläche ein und verursacht einen zu kleinen Krümmungswert am Punkt $p$.

Durch die Wahl des Kreisradius $r$ kann bestimmt werden, wie viel von der Umgebung von $p$ in die Funktion eingeht. Im Allgemeinen ist die *Flächeninvariante* für einen sehr kleinen Radius anfälliger für verrauschte Daten, als für einen größeren. Durch die Mittelung über eine weite Umgebung entspricht der Wahl eines größeren Radius einer impliziten Glättung der Kurvenmerkmale.

### 2.2.2 Beziehung zur Kurvenkrümmung

Gegeben eine differenzierbare Kurve kann die Kurvenkrümmung an einem Punkt durch den Flächenbetrag approximiert werden, der durch die Integralinvariante von Manay et. al. gemessen wird. Dies soll in diesem Abschnitt gezeigt werden. In die Approximation geht die Annahme ein, dass das Integral am Rand einer glatten Kurve gemessen wird. Im Allgmeinen lassen sich natürlich keine Rückschlüsse vom gemessenen Integral am Punkt $p$ und einer Krümmung ziehen. Auf mögliche Problemfälle wird später eingegangen.

Gegeben sei ein Kurvenausschnitt. An einem Punkt $p$ soll die Krümmung $\kappa$ angenähert werden. In Abbildung 2.1 sind neben dem Kurvenstück der *Krümmungskreis* $C_R$ mit Radius $R$ und der *Messungsumkreis* $C_r$ der Integralinvariante mit dem *Radius* $r$ um den Punkt $p$ eingezeichnet. Der Krümmungskreis ist so definiert, dass dessen Mittelpunkt auf einer Linie mit der Kurvennormale liegt und für den Radius $R = \frac{1}{|\kappa|}$ gilt. Der Kreis berührt den Punkt $p$ und nähert die Kurve in dessen Umgebung an. Sei $A_r$ die von der Integralinvariante gemessene Fläche bei konstanter Gewichtsfunktion $w$.

**Satz.** *Unter den genannten Bedingungen lässt sich die Krümmung als Funktion der Invariantenfläche $A_r$ und dem Integralradius $r$ approximieren:*

$$\kappa \approx \frac{2}{r} \sin\left(\frac{3\pi}{2}\left[\frac{1}{2} - \frac{A_r}{\pi r^2}\right]\right) \tag{2.4}$$

*Anders betrachtet ist die Krümmungsapproximation von der normierten Integralinvariante $I_c^r := \frac{A_r}{\pi r^2}$ und deren Radius $r$ abhängig.*

*Beweis.* Wir legen ein Koordinatensystem über die beiden Kreise, so dass $C_R$ im Ursprung angesiedelt ist und $C_r$ den Mittelpunkt $(0, R)$ hat. Die von der Integralinvariante gemessene Fläche $A_r$ entspricht ungefähr der Fläche, die durch die Überlappung des Krümmungskreises mit $C_r$ um den Punkt $p$ entsteht. Die Überlappung kann exakt berechnet werden, indem die Fläche zwischen der Gleichung des unteren Halbkreises von $C_r$ und der des oberen Halbkreises von $C_R$ gemessen wird. Die Integrationsgrenzen des zugehörigen Integrals werden durch die zwei Schnittpunkte der beiden Halbkreise bestimmt: $a = \pm\sqrt{r^2 - (\frac{r^2}{2R}^2)}$. Es gilt also:

$$\begin{aligned}
A_r &\approx \int_{-a}^{a} \sqrt{R^2 - x^2} - [R - \sqrt{r^2 - x^2}]dx \\
&= R^2 \int_{-\frac{a}{R}}^{\frac{a}{R}} \sqrt{1 - x^2}dx + r^2 \int_{-\frac{a}{r}}^{\frac{a}{r}} \sqrt{1 - x^2}dx - 2Ra \\
&= R^2 \frac{1}{2}(\arcsin(x) + x\sqrt{1 - x^2})\Big|_{-\frac{a}{R}}^{\frac{a}{R}} + r^2 \frac{1}{2}(\arcsin(x) + x\sqrt{1 - x^2})\Big|_{-\frac{a}{r}}^{\frac{a}{r}} - 2Ra \\
&= R^2 \arcsin(\frac{a}{R}) + r^2 \arcsin(\frac{a}{r}) + a(\underbrace{\sqrt{R^2 - a^2} + \sqrt{r^2 - a^2}}_{R}) - 2Ra \\
&= R^2 \arcsin(\frac{a}{R}) + r^2 \arcsin(\frac{a}{r}) - Ra
\end{aligned} \tag{2.5}$$

Setzt man den Ausdruck $\phi := \arcsin(\frac{r}{2R})$, so erhält man nach Umformung und der Definition von $a$ die Gleichungen $\frac{r}{2R} = sin(\phi)$, $\frac{a}{r} = cos(\phi)$ und $\frac{a}{R} = sin(2\phi)$. Substituiert man diese in Gleichung (2.5), entsteht eine eine von $\phi$ abhängige Formulierung:

$$\begin{aligned}
\frac{A_r}{r^2} &\approx \frac{R^2}{r^2} \arcsin(\frac{a}{R}) - \frac{aR}{r^2} + \arcsin(\frac{a}{r}) \\
&= \frac{1}{4\sin(\phi)^2} \arcsin(\sin(2\phi)) - \frac{1}{2}\left(\frac{\cos(\phi)}{\sin(\phi)}\right) + \left(\frac{\pi}{2} - \arccos(cos(\phi))\right) \\
&= \frac{1}{2}\left(\frac{\phi}{\sin(\phi)^2} - \frac{\cos(\phi)}{\sin(\phi)}\right) + \frac{\pi}{2} - \phi \\
&=: f(\phi)
\end{aligned} \tag{2.6}$$

Nun soll die rechte Seite von Gleichung (2.6) durch eine lineare Taylorentwicklung mit Entwicklungspunkt $0$ angenähert werden. Da (2.6) für $\phi = 0$ nicht definiert ist, lässt man $\phi$ gegen $0$ gehen und wendet ggf. mehrfach die L'Hospitalsche Regel an (gekennzeichnet durch ein "H"), um den Grenzwert in Erfahrung zu bringen. Für $f(\phi)$ (Gleichung (2.6)) gilt:

$$\begin{aligned}\lim_{\phi\to 0} f(\phi) &= \frac{\pi}{2} + \lim_{\phi\to 0} \frac{\phi - \sin(\phi)\cos(\phi)}{2\sin(\phi)^2} \\ &\overset{\text{H}}{=} \frac{\pi}{2} + \lim_{\phi\to 0} \frac{1 - \cos(\phi)^2 + \sin(\phi)^2}{4\cos(\phi)\sin(\phi)} = \frac{\pi}{2} + \lim_{\phi\to 0} \frac{\sin(\phi)}{2\cos(\phi)} = \frac{\pi}{2}\end{aligned}$$

Für $\lim_{\phi\to 0} f'(\phi)$ gilt nach dem gleichen Prinzip:

$$\begin{aligned}\lim_{\phi\to 0} f'(\phi) &= \lim_{\phi\to 0} \frac{d}{d\phi}\left(\frac{\phi - \sin(\phi)\cos(\phi)}{2\sin(\phi)^2} - \phi\right) \\ &= \lim_{\phi\to 0}\left(1 - \frac{[\phi - \sin(\phi)\cos(\phi)]\cos(\phi)}{\sin(\phi)^3}\right) - 1 \\ &\overset{\text{H}}{=} \lim_{\phi\to 0}\left(\frac{\phi}{3\sin(\phi)\cos(\phi)} - 1\right) \\ &\overset{\text{H}}{=} \lim_{\phi\to 0} \frac{1}{3(\cos(\phi)^2 + \sin(\phi)^2)} - 1 = -\frac{2}{3}\end{aligned}$$

Insgesamt lautet die Taylorentwicklung erster Ordnung

$$\frac{A_r}{r^2} \approx \frac{\pi}{2} - \frac{2}{3}\phi = \frac{\pi}{2} - \frac{2}{3}\arcsin(\frac{r}{2R}) \tag{2.7}$$

Da für die Krümmung am Punkt $p$ gilt $\kappa(p) = \frac{1}{R}$, lässt sich Gleichung (2.7) zur Behauptung umformen. □

Die Krümmung wächst also erwartungsgemäß mit dem Integralwert an. Die gezeigte Approximation ist eine Verbesserung gegenüber der Krümmungsschätzung in [20], wie in den Experimenten zu sehen sein wird.

### 2.2.3 Approximationsfehler

#### Größenordnung des Approximationsfehlers

Das Restglied der Approximation enthält Terme quadratischer und höherer Ordnung in $\phi$. Nach der Definition von $\phi$ wird der Fehler demnach umso kleiner, je kleiner $r$ wird. Dies entspricht der Einengung der Integrationsumgebung. Geht $r$ gegen $0$, wird (2.4) zur Gleichung im strengen Sinn. Das ist auch anschaulich klar, denn als differentielle Größe ist die Krümmung nur eine Eigenschaft der infinitesimal kleinen Umgebung um den Punkt $p$.

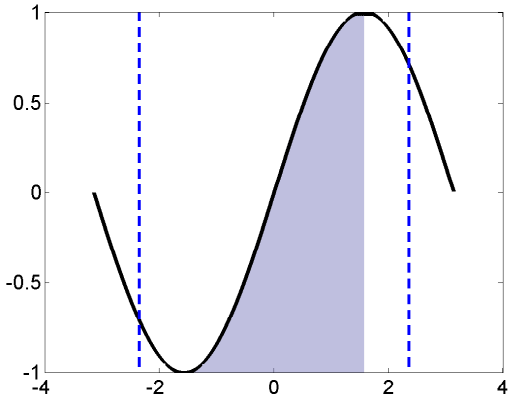



Abbildung 2.2: Der Sinusterm in der Krümmungsapproximation nimmt Werte zwischen $\sin(-\frac{3}{4}\pi)$ und $\sin(\frac{3}{4}\pi)$ an (gekennzeichnet durch die gestrichelten Grenzen). Für die Krümmungsschätzung brauchbare Werte liegen im schattierten Bereich für $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Am Rand des Bereiches fallen die Werte wieder ab.

Letztlich findet also bei der Verwendung der Integralinvarianten eine Abwägung statt zwischen impliziter Kurvenglättung und damit Stabilität der Krümmungsberechnung durch eine größer gewählte Integrationsumgebung auf der einen Seite und der Genauigkeit der Krümmungsschätzung auf der anderen. Das optimale Verhältnis wird in der Praxis experimentell bestimmt (siehe auch Kapitel 5 und 7). Der Einfluss des Integrationsradius auf die Krümmungsapproximation wird im nächsten Unterabschnitt näher untersucht.

### Randverhalten

Gleichung (2.4) hat folgende Form:

$$\kappa \approx \frac{2}{r} \sin\left(\frac{3\pi}{2} x\right) \text{ mit } x \in [-\frac{1}{2}, \frac{1}{2}]$$

An der Schreibweise wird deutlich, dass der Parameter der Sinusfunktion zwischen $-\frac{3}{4}\pi$ und $\frac{3}{4}\pi$ liegt (Abb. 2.2). Die Sinusfunktion ist auf dem schattierten Bereich in Abbildung 2.2 monoton steigend. Für Werte $> \frac{\pi}{2}$ nehmen die Funktionswerte jedoch wieder ab. Analog nehmen die Krümmungen für $x < -\frac{\pi}{2}$ zu. Eine kleine Umformung zeigt, dass dieser ungewünschte Effekt genau dann ausbleibt, falls $-\frac{1}{3} < x < \frac{1}{3}$. Dies hat zur Folge, dass Werte der Integralinvariante, die in die beiden äußeren Sechstel ihres Wertebereiches fallen, abgeschnitten werden sollten. Da dieser Fall in der Praxis recht selten auftritt, stellt er jedoch keine Einschränkung dar.

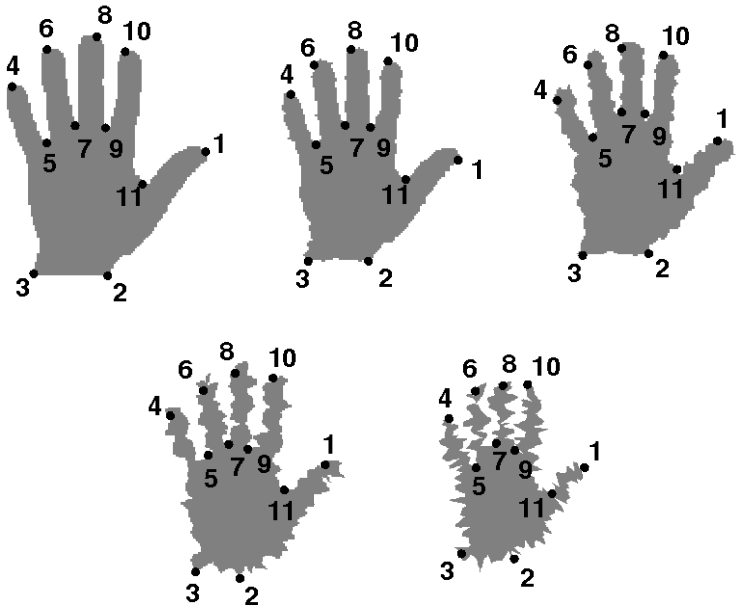



Abbildung 2.3: Test der Krümmungsapproximation in Anwesenheit von Rauschen: Die linke obere Hand wurde mit den verrauschten Versionen ($\sigma = 0.5, 1, 2$, und $3$) verglichen.

### 2.2.4 Experimente

Die Abbildungen 2.3 und 2.4 zeigen Experimente zur Rauschempfindlichkeit der Krümmungsapproximation und zur Beziehung zum Integralradius.
Um die geringe Sensibilität verrauschten Daten gegenüber zu zeigen, wurde in Abb. 2.3 die Originalform (links oben) mit verrauschten Versionen der Hand verglichen. Dabei wurde das Matchingverfahren aus Kapitel 5 verwendet. Es wurde eine gleichverteilte Störung mit Standardabweichung $\sigma \in [0.5, 3]$ in Normalenrichtung addiert. Es ist gut zu sehen, dass bis auf Punkt 2 alle Korrespondenzen sicher erkannt werden. Der Radius der Invariante darf dabei natürlich nicht zu klein gewählt werden und beträgt in diesem Beispiel $16$ bei einer Bildgröße von $230 \times 230$ Pixeln.
Der Effekt des Radius auf die Krümmungsapproximation ist zum Vergleich in Abb. 2.4 zu sehen. Entlang der Kontur einer Hand wurden 15 Punkte äquidistant abgetastet und die Integralinvariante mit Radien $15$, $9$ und $2$, sowie die zugehörige Krümmungsapproximation gemäß Gleichung 2.4 berechnet. Mit wachsendem Radius beschränkt sich der Wertebereich der gemessenen Krümmungen auf ein kleines Band um den Nullpunkt: Bei Radius $15$ ist die kleinste Krümmung $-0.133$, die größte beträgt $0.127$. Bei Radius 9 vergrößert sich der Bereich bereits auf $-0.222$ bis $0.222$, während beim kleinsten Radius Punktkrümmungen am differenziertesten erkannt werden. So wird der spitze Punkt links unten an der Hand (Punkt 3 in Abbildung 2.3) mit einer im Verhältnis entsprechend hohen Krümmung ausgestattet.

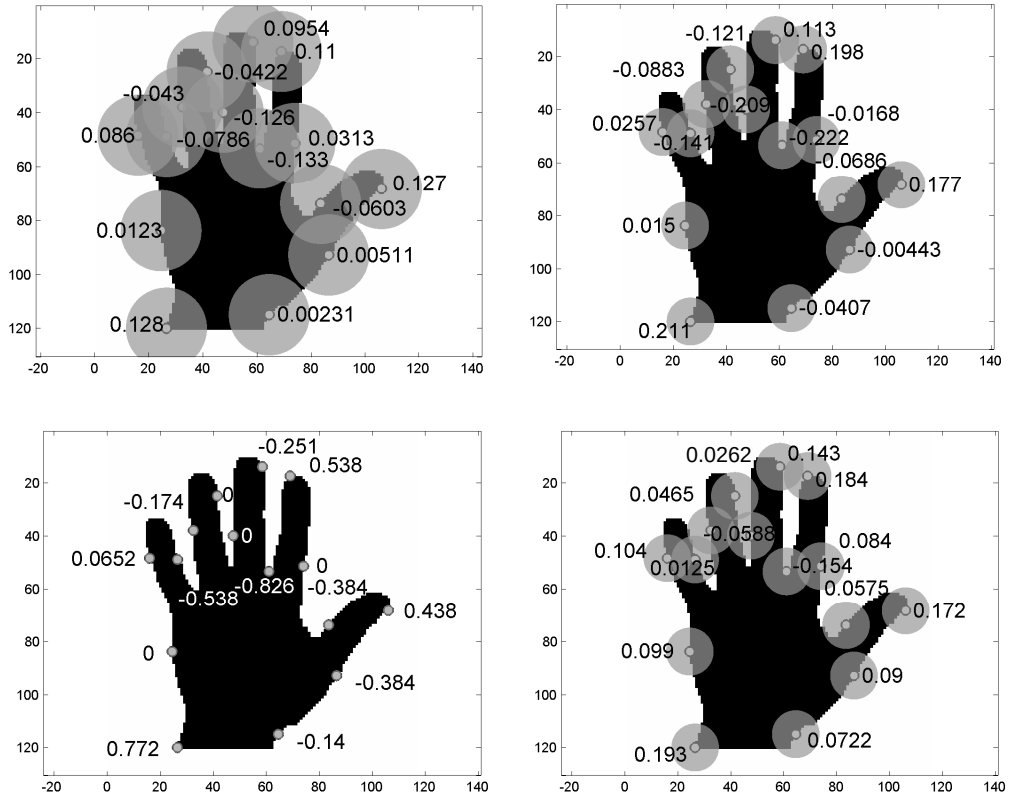



Abbildung 2.4: Krümmungsapproximation über die Integralinvarianten verschiedener Radien (15, 9 und 2). *recht unten:* Krümmungsapproximation von Manay et.al.

Analog wird der Punkt an der Verbindungsstelle von Zeige- und Mittelfinger mit der mit Abstand niedrigsten Krümmung versehen. Auf der anderen Seite verursachen kleine Pixelstörungen starke Krümmungsveränderungen (so z.B. der Punkt in der Mitte der Daumenaußenseite). Ein höherer Radius bringt insgesamt also einen stärkeren Glättungseffekt mit sich.
Schließlich wurde ein Vergleich mit der Krümmungsapproximation aus der Arbeit von Manay et.al. [20] durchgeführt (Abb. 2.4 rechts unten). An den lokalen Maxima der Krümmungen lässt sich ablesen, dass die Krümmungen durch die Taylorapproximation genauer angenähert werden. Einige der Experimente zum Shape Matching in Kapitel 5 ergaben mit der Krümmungsapproximation von Manay falsche Punktkorrespondenzen (so z.B. der Verdeckungstest).

### 2.2.5 Bewertung der Integralinvarianten

Integralinvarianten zeichnen sich durch eine robuste und - wie nachgewiesen werden konnte - diskriminative Eigenschaft aus, die beide allerdings im gegensätzli-

chen Verhältnis stehen. Ein wesentlicher Nachteil entsteht durch die Überlappung des Integrationsbereiches mit verschiedenen Kurventeilen (siehe Abbildung 2.1): Die Fläche entfernter Kurvenbereiche kann in den Wert des Integrals eingehen und hat zur Folge, dass die Krümmung als niedriger eingeschätzt wird, als sie am betrachteten Punkt tatsächlich ist (siehe auch Abb. 2.4). Im Zusammenhang mit Shape Matching kann dies falsche Korrespondenzen zur Folge haben. Der Effekt kann abgeschwächt werden, indem als Gewichtung eine Gaussfunktion gewählt wird. Weitere Eigenschaften der Integralinvarianten und der dargelegten Krümmungsapproximation werden in den Kapiteln 5 und 6 untersucht. Einige Nachteile der Integralinvarianten werden durch alternative Verfahren wie z.B. dem *Shape Context* [3] oder dem *Inner Shape Context* [17] ausgeglichen. Letzterer Ansatz ist insbesondere invariant gegenüber Artikulation von Formteilen, unterliegt jedoch einem nicht zu vernachlässigendem Rechenaufwand.

# Kapitel 3

# Shape Matching

Dieses Kapitel gibt einen Überblick über Shape Matching Verfahren, die Korrespondenzen zwischen den Punkten zweier Formen ermitteln. Abschnitt 3.1 definiert das Problem und geht auf Schwierigkeiten ein. Außerdem werden Kriterien brauchbarer Vergleichsverfahren eingeführt, die als Richtlinie für die Bewertung der in dieser Arbeit vorgestellten Algorithmen dienen. Anschließend werden in den Abschnitten 3.2 und 3.3 anhand bekannter Algorithmen Eigenschaften lokaler und globaler Shape Matching Methoden diskutiert.

## 3.1 Formenvergleich und Formendistanz

**Formenvergleich**

Das Shape Matching ist eines der für die Bildverarbeitung typischen Matching Probleme. Ziel ist es, die korrespondierenden Teile oder Punkte zweier gegebener Formen zu identifizieren. Dabei kann eine Form zwei- oder drei-dimensional sein.

Im Kontext dieser Arbeit werden Verfahren betrachtet, die Korrespondenzen von Punkten bzw. Abschnitten entlang zweier Konturen suchen; zur Definition und Repräsentation von Formen siehe Kapitel 2. In die Berechnung der gesuchten Korrespondenzabbildung zwischen den Formpunkten geht lediglich die Konturinformation ein; weder Farbe noch Textur oder gar inhaltliche Kriterien (z.B. was für ein Objekt die Kontur darstellt) werden als Bestimmungsmerkmal für den Vergleichsprozess genutzt.

Eine Schwierigkeit ist die Beurteilung der Qualität der berechneten Korrespondenzen. Letztes Maß der Dinge ist in diesem Punkt die menschliche Perzeption. Algorithmen verwenden als Qualitätsmaß in den meisten Fällen die *Formendistanz*.

### Formendistanz

Die *Formendistanz* ist ein Wert - in der Regel eine positive reelle Zahl - der angibt, wie ähnlich zwei verglichene Formen sind. Ein großer Wert steht für eine starke Unähnlichkeit, während eine Nulldistanz die Identität der beiden Formen angibt. In vielen Ansätzen ist die Berechnung der Korrespondenzen und der Distanz eng gekoppelt, indem der Distanzwert nur für gegebene Korrespondenzen berechnet werden kann, und er gleichzeitig deren Qualität bemisst. Die Korrespondenzabbildung mit dem geringsten Distanzwert wird gemeinhin als die optimale bezeichnet. Nicht immer steht ein niedriger Distanzwert jedoch für eine *sinnvolle* Korrespondenzabbildung. Das ist beispielsweise bei Mischformen der Fall.
Die Distanz von Formen kann - je nach zugrunde liegendem Shape Matching Modell - verschiedene Eigenschaften besitzen. Es ist wünschenswert, dass das Distanzmaß $d(x, y)$ für zwei Formen $x$ und $y$ eine *Metrik* darstellt, d.h. folgenden Bedingungen genügt:

- für alle Shapes $x \neq y$ gilt $d(x, y) > 0$ (Positivität)
- für eine Shape $x$ gilt $d(x, x) = 0$ (Identität)
- $d(x, y) = d(y, x)$ (Symmetrie)
- für alle $x, y, z$ gilt $d(x, z) \leq d(x, y) + d(y, z)$ (Dreiecksungleichung)

Allerdings weisen Basri et.al. darauf hin [1], dass das menschliche Distanzempfinden von Formen nicht unbedingt der Dreiecksungleichung genügt. Tatsächlich sind die meisten Distanzmaße für Formen *Semimetriken*. Die Gültigkeit der Dreiecksungleichung ist allerdings vorteilhaft, weil sich so die Menge der Formen in den euklidischen Raum einbetten lässt. Punkte stehen dann für einzelne Formen und es können Cluster und mittlere Formen berechnet werden.

## 3.1.1 Kriterien für Shape Matching-Verfahren

In diesem Unterabschnitt sollen einige Grundanforderungen an Verfahren zum Formenvergleich und zur Distanzberechnung aufgestellt werden. Die wenigsten Shape Matching Algorithmen erfüllen sämtliche der unten stehenden Kriterien. Dennoch dienen diese als Qualitätsmaß für die in der Arbeit vorgestellten Verfahren. Die Aufstellung orientiert sich an der menschlichen Wahrnehmung von Formen.

### Invarianz gegenüber Gruppentransformationen

Der in Kapitel 2 definierte Formen-Begriff sieht von Translation, Rotation und Skalierung als Unterscheidungskriterium ab. Folglich soll auch der Abstand einer geschlossenen Kontur zu einer verschobenen, rotierten und verkleinerten Version der-

selbe sein, wie der Abstand der Kontur zu sich selbst. Diese Abstraktion ist eine der grundlegendsten Eigenschaften menschlicher Wahrnehmung.

### Robustheit gegenüber Rauschen

Einer der Grundprobleme in der Bildverarbeitung ist der Umgang mit verrauschten Daten. Zwar kann man das Rauschen über Glättungsverfahren unterdrücken, jedoch meist nur unter Verlust von Information (siehe Kapitel 2). Im Falle von Konturen schlägt sich die Verrauschtheit der Daten in kleineren Ausbuchtungen entlang der Kontur nieder, die jedoch große Störungen der Krümmungsfunktion mit sich bringen. Ein guter Shape Matching Algorithmus soll Rauschen von formimmanenten Merkmalen filtern können.

### Robustheit bei Verdeckung, Artikulation und Verzerrung

Bei der Formenextraktion aus Bilddaten kommt es oft vor, dass durch unvollständige Sicht auf das Objekt oder Segmentierungsfehler entweder Mischkonturen oder unvollständige Formen entstehen. Charakteristisch für solche **Verdeckungen** ist, dass diese oft an einer einzelnen Stelle der Kontur auftreten. Die menschliche Wahrnehmung ist in der Lage, unvollständige Formen zu ergänzen und unpassende Informationen rauszufiltern. Daher ist das Ziel, entsprechend robuste Matching- und Distanzalgorithmen zu entwerfen.
Ebenfalls starke Änderungen der Kontur entstehen durch **Artikulation** von Formteilen, z.B. durch Bewegungen an Gelenken, wie dem Abspreizen eines Fingers. Typischerweise treten solche Erscheinungen an Stellen mit hoher Kurvenkrümmung auf. Für das menschliche Distanzempfinden sind artikulierte Formen sehr ähnlich. Auch vom Matchingverfahren wird Robustheit erwartet.
Das gleiche gilt für unwesentliche **Verzerrungen** (d.h. nicht Skalierung) der Kontur. Die Kontur eines schmal gebauten unterscheidet sich von der eines stark gebauten Menschen nur *relativ*. Trotzdem sind beide Konturen stark verwandt. Sie sollten daher eine geringe Distanz voneinander aufweisen, und korrespondierende Punkte sollten relativ eindeutig identifizierbar sein.

### Symmetrie

Obwohl nicht klar ist, dass die menschliche Wahrnehmung unbedingt symmetrisch agiert, mutet eine gewisse Unabhängigkeit von der Wahl der Ausgangskontur sinnvoll an. Daher wird von einem guten Vergleichsmaß gefordert, dass die Distanz einer Kontur A zu einer Kontur B dieselbe ist, wie die Distanz von B zu A. Ebenso soll das Matchingergebnis dasselbe sein, egal von welcher der beiden Konturen ausgegangen wird.

### 3.1.2 Shape Matching Methoden

Es gibt eine Vielzahl sehr unterschiedlicher Herangehensweisen an die Berechnung von Formkorrespondenzen und Distanzmaßen. Wie bereits angedeutet, werden in dieser Arbeit lediglich Verfahren betrachtet, die Punkte oder Abschnitte entlang zweier Konturen miteinander identifizieren. Innerhalb dieser Klasse von Shape Matching-Verfahren gibt es im wesentlichen zwei Typen von Algorithmen:

**Globale Ansätze** transfomieren die beiden Formen als Ganzes, so dass korrespondierende Konturteile zur Deckung kommen.

**Lokale Ansätze** vergleichen nur die unmittelbaren Umgebungen von Punkten oder Abschnitten entlang der Kontur und versuchen, möglichst ähnliche Teile zu identifizieren.

Im weiteren Verlauf des Kapitels wird auf jede der beiden Methoden näher eingegangen. Am Beispiel einschlägiger Verfahren werden die jeweiligen Vor- und Nachteile besprochen.

## 3.2 Globale Shape Matching Verfahren

### 3.2.1 Überblick über globale Verfahren

Globale Verfahren transformieren die zu vergleichenden Konturen so, dass sie möglichst ähnlich sind. Dabei wird auf jeden Punkt einer Form die gleiche Transformation angewendet. Die zulässigen Transformationen sind typischerweise klar eingegrenzt (z.B. Gruppentransformationen). Anhand des bleibenden Unterschiedes der transformierten Konturen wird die Formendistanz berechnet. Es kann auch das Ausmaß der verwendeten Transformation in die Berechnung eingehen. Im Folgenden wird ein beliebtes Vergleichsverfahren vorgestellt, hier in der Formulierung von [9]. Für die Transformation der Formen werden *Ähnlichkeitsabbildungen* verwendet. Für die Bestimmung der Punktkorrespondenzen siehe z.B. in [28].

### 3.2.2 Summe der quadratischen Abstände

#### Distanzmaß

Zunächst wird die Berechnung des *Distanzmaßes* zwischen zwei Formen dargelegt; es wird also davon ausgegangen, dass eine punktweise Korrespondenzabbildung bereits gegeben ist. Die Formen sind durch zwei Punktmengen $P_1 = \{x_1, x_2, ..., x_n\}$ und $P_2 = \{y_1, y_2, ..., y_n\}$ im $\mathbb{R}^2$ gegeben, wobei Punkt $x_i$ und $y_i$ korrespondieren. Die Punkte können abgetastete Kurven sein oder fixe Markierungen auf der Form (*Landmarks*).

Gesucht werden Translation, Rotation und Skalierung der Menge $P_1$, so dass korrespondierende Punkte beider Punktmengen bestmöglich zur Deckung kommen. Besonders elegant lässt sich das Problem im Komplexen formulieren, indem der zweidimensionale euklidische Raum $\mathbb{R}^2$ mit der Menge der komplexen Zahlen $\mathbb{C}$ identifiziert wird. Schreibt man nämlich $P_1$ und $P_2$ als komplexe Vektoren $x = (x_1, x_2, ..., x_n)^T$ und $y = (y_1, y_2, ...y_n)^T$, dann existiert für beliebige Transformationsparameter $t$, $\beta$ und $\phi$ ein $n \times 1$ Fehlervektor $\epsilon$, so dass gilt:

$$\begin{aligned} y &= t \cdot 1_n + \beta e^{i\phi} x + \epsilon \\ &= [1_n, x] \begin{pmatrix} t \\ \beta e^{i\phi} \end{pmatrix} + \epsilon \\ &= XA + \epsilon \end{aligned}$$

Dabei ist $t \in \mathbb{C}$ die Translation, $\beta$ die Skalierung und $\phi$ der Rotationswinkel der Menge $P_1$. $1_n$ bezeichnet den $n \times 1$ Vektor aus Einsen. Das *Distanzmaß* zwischen den beiden Punktmengen wird als Summe der quadratischen Abstände korrespondierender Punkte definiert, also nach Umformung:

$$d^2(x, y) = \min_{t,\beta,\phi}(\epsilon^* \epsilon) = \min_{t,\beta,\phi}(y - XA)^*(y - XA) \tag{3.1}$$

wobei $\epsilon^*$ die komplex konjugierte und transponierte Version von $\epsilon$ bezeichnet. Diese Formulierung entspricht linearer Regression.

### Berechnung der Korrespondenzen

Für die Bestimmung den optimalen Korrespondenzen zwischen den Punktmengen kann die Distanz als Gütemaß verwendet werden. Gesucht ist dann diejenige Korrespondenzabbildung von Punkten, für die die oben beschriebene Distanz minimal wird. Die Suche nach den Korrespondenzen kann unter verschiedenen Bedingungen erfolgen:

- Ein Punkt kann auf mehrere Zielpunkte abgebildet werden oder nur auf einen.
- Es werden Punkte zugelassen, die gar keinen Korrespondenzpartner besitzen oder jeder Punkt besitzt mindestens bzw. genau einen Korrespondenzpartner.
- Die Punkte können eine Ordnung haben, die bei der Abbildung berücksichtigt werden muss, z.B. können Abbildungen über Kreuz verboten werden: Korrespondenzen $(x_i, y_j)$ und $(x_{i+l}, y_{j-k})$ sind dann unvereinbar. Das entspricht einer Monotoniebedingung der Korrespondenzpartner.

Insbesondere die letzte Einschränkung kann den Vorgang wesentlich vereinfachen (siehe Kapitel 4), denn im Allgemeinen ist der globale Ansatz der punktweisen Korrespondenzfindung unter rigiden - und erst recht unter nicht-rigiden - Transformationen sehr rechenintensiv. Das Problem der ersten Annahme ist, dass sie einen

gegen Null gehenden Skalierungsfaktor begünstigt, der sämtliche Punkte der ersten Form auf genau einen Punkt der zweiten Form abbildet, so dass die Summe der Distanzen minimal wird.

### 3.2.3 Bewertung globaler Verfahren

Viele globalen Methoden zur Berechnung von punktweisen Korrespondenzen zwischen zwei Formen sind sehr rechenintensiv. Mit immer größer werdenden Punktmengen, wächst auch die Menge der Korrespondenzmöglichkeiten stark an. Das Hauptproblem ist jedoch die Unverträglichkeit solcher Verfahren mit Verdeckung und Artikulation. Weder können diese von globalen Ansätzen gut unterschieden, noch kann ihr Einfluß auf die Distanz gezielt gemindert werden, da sämtliche Punkte bei der Transformation gleich behandelt werden (Beispiel: gestrecktes und angewinkeltes Bein).
Auf der anderen Seite behalten globale Methoden die Gesamtform im Auge und können starke Ähnlichkeit von Punktmengen sicher nachweisen. Es gibt auch Anstrengungen, die globalen Vergleichsmethoden mit Artikulation verträglicher zu machen. In diesem Kontext wurde beispielsweise die *Hausdorff-Distanz* auf den Bereich der Bildverarbeitung übertragen und so definiert, dass Ausreißer nicht überbewertet werden (siehe [16]). Die Hausdorff-Distanz ist allerdings bekanntlich nicht symmetrisch.

## 3.3 Lokale Shape Matching Vefahren

### 3.3.1 Überblick über lokale Verfahren

Viele der oben gezeigten Nachteile globaler Matching-Verfahren, werden durch sogenannte lokale Ansätze vermieden. Ziel ist es auch hier, eine punkt- oder abschnittweise Korrespondenz zwischen zwei geschlossenen Konturen herzustellen.
Allen lokalen Verfahren liegt die Annahme zugrunde, dass Punkte oder Segmente mit möglichst ähnlichen lokalen Eigenschaften aufeinander abgebildet werden sollen. Durch die auf Umgebungen beschränkte Vergleichsweise sind lokale Matching-Verfahren gutmütig gegenüber zahlreichen nicht-rigiden Transformationen der Formen, wie beispielsweise der Artikulation von Formteilen. Daher werden sie auch oft als *elastische* Verfahren bezeichnet.
Die meisten lokalen Ansätze messen die Summe der Unähnlichkeit korrespondierender lokaler Stellen. Dazu kann ein Glattheitsterm kommen, der Verformungen bestraft. Oft wird der Raum der Korrespondenzabbildungen kombinatorisch beschränkt, indem die Ordnung der Punkte entlang der Konturen aufrecht erhalten wird (siehe Kapitel 4). Es gibt jedoch auch andere Varianten [3]. Punkte mit sehr ausgeprägten lokalen Eigenschaften bestimmen bei elastischen Verfahren oftmals

die grobe Richtung der Korrespondenzabbildung, da ihre Korrespondenzpartner nicht beliebig gewählt werden können, ohne das Datenkriterium zu stark zu verletzen. Die Idee punktweiser lokaler Methoden wird an einer der ersten krümmungsbasierten elastischen Verfahren von Cohen et. al. dargelegt.

### 3.3.2 Cohen et. al.

Cohen, Ayache und Sulger veröffentlichten 1992 ein elastisches Shape Matching Verfahren als Variationsansatz [7]. Ziel ist es, evolvierende Konturen in Bildsequenzen zu verfolgen. Die Annahme ist daher, dass die Krümmung markanter Kurvenpunkte aufeinanderfolgender Bilder annähernd konstant bleibt.

Gegeben zwei auf dem Einheitsintervall parametrisierte, geschlossene Kurven $C_1$ und $C_2$, soll die Korrespondenzfunktion $f : [0,1] \rightarrow [0,1]$ berechnet werden, die die Punkte der einen Kurve auf die der anderen abbildet. Neben der Krümmungskonsistenz soll gleichzeitig das Vektorfeld, das aus den Differenzenvektoren korrespondierender Punkte besteht, möglichst glatt sein. Formal wird also eine Korrespondenzabbildung $f$ gesucht, die folgendes Funktional minimiert:

$$E(f) = \int_{[0,1]} [\kappa_1(s) - \kappa_2(f(s))]^2 ds \; - \; \lambda \int_{[0,1]} \left|\left| \frac{\delta(C_1(s) - C_2(f(s)))}{\delta s} \right|\right|^2 ds$$

Wobei $\kappa_{1,2}$ die Krümmungen von $C_{1,2}$ beschreiben. Das Funktional ist sehr typisch für lokale Verfahren, auch wenn viele Variationen existieren. Die Randbedingungen für eine gültige Korrespondenzfunktion geschlossener Konturen lauten:

$$f(0) = 0, \; f(1) = 1$$

Der **Datenterm** (links) von $E$ bewirkt, dass *bevorzugt* Punkte mit besonders hoher bzw. niedriger Krümmung auf ähnliche Punkte der zweiten Kontur abgebildet werden, sich der Rest des Matchings also an der optimalen Korrespondenz von Punkten ausgeprägter Krümmung orientiert. Der **Regularisierungsterm** (rechts) sorgt für eine möglichst gleichmäßige Korrespondenzverteilung insbesondere an glatten Abschnitten der Kontur.

#### Minimierung

Cohen et.al. berechnen zum Funktional $E$ die Euler-Lagrange-Gleichungen und suchen das Minimum über die Gradientenabstiegsmethode. Der Gradientenabstieg führt allerdings nur bei konvexen Funktionalen notwendig zum globalen Minimum. Da der Datenterm des Funktionals $E$ jedoch nicht konvex ist, ist auch $E$ nicht konvex. Ob die Minimierungsmethode zum globalen Minimum führt, hängt von einer geeigneten Initialisierung $f_0$ der Korrespondenzfunktion für den Gradientenabstieg

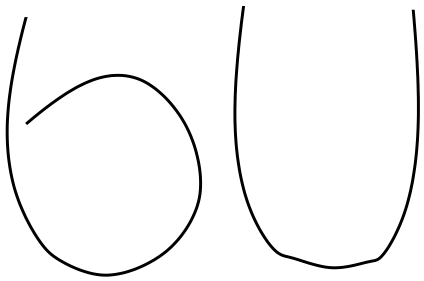


Abbildung 3.1: Beispiel für zwei Formen, deren Distanz von lokalen Verfahren nicht repräsentativ berechnet werden kann, da sich die Krümmungen lediglich um einen kleinen Bereich herum signifikant unterscheiden (Abbildung nach [23]).

ab. Diese wird durch eine Heuristik ermittelt, in die die Annahme einfliesst, dass die Ordnung der Punkte entlang der Kontur erhalten bleibt. Dies führt jedoch zu dem Problem, dass eine geeignete Initialkorrespondenz vorliegen muss, die nicht sicher bestimmt werden kann (vgl. zu diesem Problem Abschnitt 4.4).
In jedem Fall ist globale Minimierung des Funktionals ungewiss, auch wenn sie bei kleinen Veränderungen der Konturen zueinander stabil ist. Wie beim Gradientenabstieg üblich, hängt die Konvergenzgeschwindigkeit des Verfahrens stark von der Initialisierung ab.

### 3.3.3 Bewertung lokaler Verfahren

Elastische Shape Matching-Verfahren sind durch ihre primär lokalen Vergleichskriterien im Allgemeinen weniger anfällig gegenüber Artikulation und Verdeckung. Funktionale - wie das vorgestellte - sind sehr typisch für lokale Matching Probleme. Die globale Minimierung über Gradientenabstieg scheitert jedoch an der fehlenden Konvexität des Datenterms. Daher werden im nächsten Kapitel Verfahren vorgestellt, die ein globales Optimum garantieren. Am Beispiel von Cohen et. al. wurde außerdem gezeigt, dass die Beibehaltung der Punktreihenfolge bei der Abbildung auf die Zielkontur zu dem Problem der optimalen Initalkorrespondenz führt; mehr dazu in Kapitel 5.

Nachteilig bei lokalen Verfahren ist, dass sie durch die Konzentration auf kleine Umgebungen dazu tendieren, den größeren Formzusammenhang aus dem Auge zu verlieren. So werden z.B. die Formen in Abbildung 3.1 (insbesondere als Schriftzeichen) vom Menschen als sehr unterschiedlich empfunden, ihre Krümmung divergieren jedoch lediglich auf einem kleinen Abschnitt.

# Kapitel 4

# Elastisches Shapematching über Dynamische Programmierung

In Kapitel 2 wurden elastische Shape Matching Verfahren eingeführt. Es wurde gezeigt, dass die Energieminimierung über Gradientenabstieg oder ähnliche Methoden aufgrund der fehlenden Konvexität des Datenterms weder effizient ist noch notwendig zu global minimalen Lösungen führt. Daher soll in diesem Kapitel eine Gruppe von Ansätzen vorgestellt werden, die das Matching Problem als Sequenzvergleich betrachten, welches relativ effizient über dynamische Programmierung gelöst werden kann.
In Abschnitt 4.1 wird in Grundzügen der Ansatz der dynamischen Programmierung erläutert. Dies bildet die Basis für viele Algorithmen zum Sequenzvergleich, für die im Abschnitt 4.2 ein Überblick gegeben wird. Anschließend wird in Abschnitt 4.3 erklärt, wie Sequenzvergleiche für das Shape Matching eingesetzt werden können. Die Probleme, die dabei für den Vergleich geschlossener Konturen entstehen, werden in Abschnitt 4.4 näher untersucht.

## 4.1 Dynamische Programmierung

Dynamische Programmierung ist ein algorithmisches Prinzip zur globalen Lösung bestimmter Optimierungsprobleme. Der Begriff “Dynamische Programmierung” wurde von Richard Bellman eingeführt, der die Methode 1957 publizierte. Er entstammt nicht dem Kontext der Computerprogrammierung sondern vielmehr dem der “mathematischen Programmierung” oder Optimierung. Ursprünglich richtete sich das Verfahren auf mehrstufige, dynamische Entscheidungsprozesse.
Das zugrunde liegende Modell ist folgendes: Ein System ist zu jedem Zeitpunkt in einem bestimmten Zustand. Für jeden Zustand gibt es eine Reihe von möglichen Folgezuständen, bei deren Übergang bestimmte Kosten anfallen. Ziel ist es, von einem gegebenen Anfangszustand aus die Summe der Übergangskosten, die bis zur

Erreichung eines Endzustandes anfallen, möglichst gering zu halten. Die Strategie, in jedem Schritt den Folgezustand mit den geringsten Kosten zu wählen ("Greedy-Strategie"), führt im Allgemeinen *nicht* zum Ziel. Sämtliche mögliche Zustandsfolgen zu berechnen und diejenige mit den geringsten Kosten zu wählen, optimiert das Problem zwar global, führt jedoch zu einer exponentiellen Laufzeit. Dynamische Programmierung hingegen liefert unter bestimmten Voraussetzungen dieselbe (globale) Lösung, jedoch mit praktikablem Rechenaufwand. Jedes Optimierungsproblem, dass sich als ein solcher dynamischer, mehrstufiger Entscheidungsprozess formulieren lässt, kann über dynamische Programmierung gelöst werden, sofern es dem so genannten *Optimalitätsprinzip* genügt.

### 4.1.1 Optimalitätsprinzip

Um ein Optimierungsproblem über dynamische Programmierung lösen zu können, muss es folgendem Prinzip genügen:

**Optimalitätsprinzip** Für einen gegebenen Zustand $x_i$ des Systems ist die optimale Übergangsfolge bis zum Endzustand nicht abhängig von den Vorgängerentscheidungen, die das System in den aktuellen Zustand $x_i$ überführten.

Die optimale Folge von Übergängen mit den geringsten Kosten von $x_i$ aus, hängt also *nur* von $x_i$ selbst ab. Diese Voraussetzung führt zu einer rekursiven Berechnungsvorschrift: Seien $F_{end}(x_i)$ die Kosten einer optimalen (nicht unbedingt eindeutigen) Zustandsfolge vom Zustand $x_i$ in den Endzustand $x_{end}$ über zulässige Übergänge. Dann gilt:

$$F_{end}(x_i) = \min_{x_{i+1} \in U(x_i)} \{c(x_i, x_{i+1}) + F_{end}(x_{i+1})\} \tag{4.1}$$

Wobei $U(x_i)$ die Menge der zulässigen Folgezustände beschreibt und $c(x_i, x_{i+1})$ die Kosten des Überganges von Zustand $x_i$ nach $x_{i+1}$. Da $F_{end}(x_{end}) = 0$, ist die Rekursion berechenbar, sofern die Zustandsmenge endlich ist. Das Gesamtproblem $F_{end}(x_{start})$ wird also rekursiv durch Teilprobleme $F_{end}(x_i)$ ersetzt. Diese können bei der Berechnung der optimalen Zustandsfolge mehrmals auftreten, insbesondere bei *ein-dimensionalen* Problemformulierungen. Bei der dynamischen Programmierung werden daher die bereits berechneten Ergebnisse der Teilprobleme in einer Tabelle gespeichert und müssen bei Bedarf nicht erneut berechnet werden.

### 4.1.2 Kürzeste-Wege-Problem

Ein wohl bekanntes Problem, zu dessen Lösung dynamische Programmierung verwendet werden kann, ist das Finden kürzester Wege. In einem gewichteten Graphen soll von einem Startknoten $u$ aus zu jedem anderen Knoten der Pfad mit den geringsten Kosten berechnet werden. Die Kosten sind die Summe der Gewichte der auf dem Pfad liegenden Kanten.

Als Entscheidungsprozess betrachtet stellen die Knoten die Zustände dar. Der Startzustand ist der Quellknoten, der Zielknoten des gesuchten kürzesten Weges ist der Endzustand. Die Kantengewichte sind die Kosten, die beim Zustandswechsel anfallen. Die Rekursionsgleichung bekommt eine anschauliche Bedeutung:

$$F_u(w) = \min_{(v,w)\in E}\{F_u(v) + c(v,w)\} \tag{4.2}$$

wobei $F_u(x)$ den kürzesten Weg von Startknoten $u$ nach Knoten $x$ beschreibt und $c(v,w)$ das Kantengewicht der Kante zwischen den Knoten $v$ und $w$. Das Gesamtproblem wird also in die Berechnung kürzester Teilpfade zerlegt.
Bellman weist allerdings in [2] darauf hin, dass die Formulierung offen lässt, in welcher Reihenfolge die Zwischenergebnisse $F_u$ am besten berechnet werden können und schlägt zwei Iterationsschemata vor. Es ist die Leistung von *Dijkstras Algorithmus*, ein Schema zu formulieren, nach welchem der bisher bekannte kürzeste Weg für jeden Knoten nur *einmal* aktualisiert werden muss.

## 4.2 Sequenzvergleiche

Gegeben seien zwei Eingabesequenzen $a = a_1a_2...a_n$ und $b = b_1b_2...b_m$. Ziel ist es zum einen, eine *Relation* zu finden, welche die Elemente beider Sequenzen einander zuordnet. Dabei soll die Ordnung der Sequenzelemente bei der Abbildung aufrechterhalten werden, d.h. Korrespondenzen zwischen Elementen können sich nicht überkreuzen. Zum anderen soll über die Relation eine *Distanz* zwischen beiden Sequenzen berechnet werden, die ein Ähnlichkeitsmaß darstellt.

Motivation und Anwendungen für Sequenzvergleiche sind vielseitig. Grundsätzlich gibt es eine *kontinuierliche* und eine *diskrete* Interpretation. Ziel der kontinuierlichen Variante ist der Vergleich zweier Signale, von deren Ablaufgeschwindigkeit in einem gewissen Maße abstrahiert werden soll. Ein historisches Beispiel ist die *Spracherkennung*. Die *diskrete* Interpretation wird für *Stringvergleiche* genutzt, wie sie z.B. in der Molekularbiologie auf genetischen Codes durchgeführt werden.
In den siebziger Jahren kamen erstmals Verfahren auf, die mit Hilfe dynamischer Programmierung Sequenzvergleiche durchführten. Vintsyuk (1968), Velichko und Zagoruyko (1970) sowie Sakoe und Chiba (1970) setzten erstmals das so genannte *Dynamic Time Warping* zur Spracherkennung ein. Needleman und Wunsch (1970) entwickelten ähnliche Verfahren in der Molekularbiologie.

Im Folgenden soll das Prinzip der Sequenzvergleiche über dynamische Programmierung anhand einer Standardform erläutert werden. Anschließend werden kurz Abwandlungen beschrieben. Dabei geht es nicht um Vollständigkeit, sondern um Formen, die im Shape Matching Kontext relevant sind. Eine historische sowie in-

haltliche Übersicht über Algorithmen zu Sequenzvergleichen geben Sankoff und Kruskal [22].

### 4.2.1 Verfahren

Gesucht wird eine Relation zwischen den Sequenzen $a = a_1a_2...a_n$ und $b = b_1b_2...b_m$ als eine Folge von Zweiertupeln $r = (r_1, r_2, ..., r_p)$ mit $r_k = (a_{i_k}, b_{j_k})$. Die in Frage kommenden Sequenzabbildungen sollen dabei kombinatorisch beschränkt werden:

**Randbedingung** Die erste Zuordnung ist $r_0 = (a_0, b_0)$, die letzte $r_p = (a_n, b_m)$

**Monotonie** Lautet eine Zuordnung $r_x = (a_i, b_j)$, so gilt für eine folgende Zuordnung $r_{x+t} = (a_k, b_l)$ mit $k \geq i$ und $l \geq j$

**Stetigkeit** Für zwei unmittelbar aufeinander folgende Zuordnungen $r_x = (a_i, b_j)$ und $r_{x+1} = (a_k, a_l)$ ist $|k - i| \leq 1$ und $|l - j| \leq 1$

Die Monotonie-Bedingung stellt sicher, dass die Ordnung der Sequenzelemente bei der Abbildung nicht zerstört wird. Die Stetigkeit verhindert, dass Elemente aus $a$ oder $b$ in der Folge der Zuordnungen übersprungen werden; wohlgemerkt können jedoch Elemente doppelt zugewiesen werden. Monotonie und Stetigkeit zusammen beschränken für eine Korrespondenz $r_x = (a_i, b_j)$ die **Menge der möglichen Folgekorrespondenzen** $r_{x+1}$ auf

$$r_{t+1} \in \{(a_{i+1}, b_{j+1}), (a_{i+1}, b_j), (a_i, b_{j+1})\}.$$

Gesucht ist diejenige unter den im obigen Sinn zulässigen Relationen, welche möglichst ähnliche Bereiche der beiden Sequenzen aufeinander abbildet. Gegeben eine Zuordnungsfolge $r = (r_1, r_2, ..., r_p)$ und ein Distanzmaß $d(a_i, b_j)$ (meistens die *euklidische Distanz*), das die Ähnlichkeit zweier Elemente als reelle Zahl darstellt, lässt sich die Güte der Relation $r$ als Summe der Distanzen zugeordneter Sequenzelemente messen:

$$c(r) := \sum_{(a_i,b_j)\in r} d(a_i, b_j)$$

Diese Summe wird als die *Distanz* zwischen den Sequenzen definiert. Das Optimierungsproblem besteht darin, die nicht notwendigerweise eindeutige Relation mit den geringsten Kosten $c(r)$ zu finden. Ein häufig verwendetes Distanzmaß $d()$ ist der *euklidische Abstand*.

### 4.2.2 Repräsentation als Graph

Der Lösungsraum des Problems lässt sich intuitiv als Pfadmenge in einem Graphen darstellen. Der Graph besteht aus einem rechteckigen Knotengitter mit $n \times m$

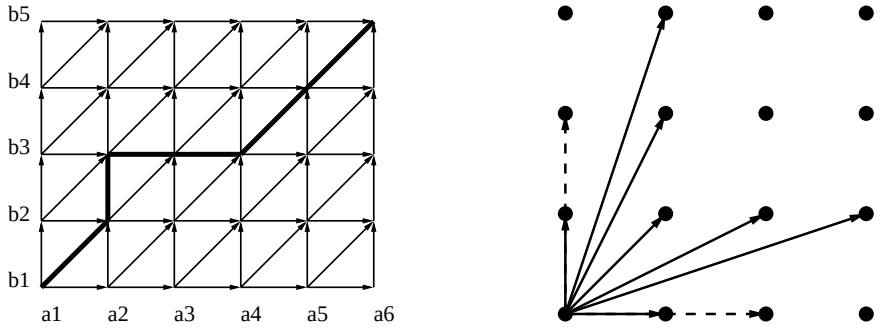



Abbildung 4.1: *links*: Beispiel eines Graphen zum Sequenzvergleich mit Warping-pfad. *rechts*: Beispiel für eine alternative Schrittkonfiguration zwischen den Knoten

Knoten (siehe Abb. 4.1). Knoten $(i, j) \in \{1, ..., n\} \times \{1, ..., m\}$ repräsentiert eine Zuordnung des Elements $a_i$ der Sequenz $a$ mit dem Element $b_j$ der Sequenz $b$. Jeder Knoten $(i, j)$ wird mittels *gerichteten* Kanten mit den drei möglichen Nachfolgezuordnungen $(i + 1, j + 1)$, $(i + 1, j)$ und $(i, j + 1)$ verbunden. Es entsteht ein regelmäßiges Kantengitter. Die Menge der Kanten wird mit $E$ bezeichnet.
Den drei Bedingungen der Monotonie, Stetigkeit und der Randbedinung zufolge entspricht jeder Pfad von Knoten $(0, 0)$ nach Knoten $(n, m)$ einer gültigen Relation zwischen den Elementen der beiden Sequenzen. Darüber hinaus gibt es für jede mögliche Relation einen Pfad. Eine Beispielrelation ist ebenfalls in Abbildung 4.1 gegeben.

#### Kantengewichte

Versieht man jede Kante $(i, j) \rightarrow (k, l)$ mit dem Gewicht $d(k, l)$, so kann man auf dem Graphen kürzeste Wege berechnen. Der kürzeste Pfad von Knoten $(0, 0)$ zum Knoten $(n, m)$ entspricht dann der gesuchten optimalen Relation zwischen den Sequenzen. Denn jeder Pfad $p \in E$ zwischen diesen beiden Knoten hat die Kosten

$$c(p) = \sum_{[(i,j)\rightarrow(k,l)]\in p} d(a_k, b_l)$$

die gleich den Kosten der zugehörigen Relation minus der Konstanten $d(a_0, b_0)$ sind. Die Kosten des kürzesten Weges sind also gleich dem Abstand der beiden Eingabesequenzen minus $d(a_0, b_0)$.

### 4.2.3 Optimierung

Die Reduktion auf das Kürzeste-Wege-Problem hat zur Folge, dass jeder Algorithmus zur Optimierung eingesetzt werden kann, der den kürzesten Weg zwischen zwei Knoten auf einem Graphen berechnet. *Dijkstras Algorithmus* benötigt Zeit

$O(nm\log(n))$. Die Monotoniebedingung erlaubt es hingegen, ohne Priority-Queue auszukommen: Zu jedem Knoten $(i, j)$ gibt es genau drei mögliche Vorgänger: $(i-1, j-1)$, $(i-1, j)$ und $(i, j-1)$. Die zu lösende Rekursionsgleichung lautet also

$$F(i,j) = d(a_i, b_j) + \min\{F(i-1, j-1), F(i-1, j), F(i, j-1)\} \tag{4.3}$$

mit $F(0,0) = 0$ und $F(i,j)$ die Kosten des kürzesten Weges von Knoten $(0,0)$ nach $(i,j)$. Werden in jedem Knoten $(i,j)$ die Kosten $F(i,j)$ gespeichert, reicht folgende

**Berechnungsvorschrift.** *Besuche die Knoten von unten nach oben und links nach rechts und aktualisiere die Kosten jedes Knotens entsprechend Formel (4.3).*

### 4.2.4 Varianten

Je nach Anforderung des Algorithmus kann das beschriebene Grundverfahren abgewandelt werden. Um die Berechnung für dynamische Programmierung effizient zu gestalten, muss allerdings stets die Monotoniebedingung aufrechterhalten werden. Die Modifikationen haben zum Ziel, den Raum der möglichen Relationen zwischen den Eingabesegmenten zu erweitern oder zu beschränken. Ein anderer Lösungsraum bringt auch ein anderes Distanzmaß mit sich.

#### Stetigkeit

Die Menge der möglichen Nachfolgerknoten (und damit die Kantenmenge im Graphen) kann erweitert werden, so dass es möglich wird, Sequenzelemente zu überspringen. Unweigerlich ändern sich dadurch auch die möglichen Vorgängerkorrespondenzen eines Knotens. Abbildung 4.1 zeigt ein Beispiel.

#### Randbedingung

Die Randbedingung lässt sich ebenfalls ändern. So können auch *Mengen* möglicher Anfangs- und Endkorrespondenzen definiert werden. Je nach möglichen Anfangskorrespondenzen muss der Graph erweitert oder anpasst werden, um die Menge aller gültigen Korrespondenzabbildungen abzudecken. Alternativ kann ein zyklischer Graph definiert werden.

#### Knoten- vs. kantenbasiert

Bisher wurde eine Kante mit den Kosten des Zielknotens versehen. Diese "knotenorientierte" Konstruktion macht keine Unterscheidung bezüglich des gegangenen Weges. Je nach Anwendung kann es daher nützlich sein, verschiedene Typen von Eingangskanten mit unterschiedlichen Kosten zu versehen. So kann beispielsweise in der kontinuierlichen Interpretation eine Stauchung oder Streckung einer der Sequenzen mit zusätzlichen Kosten belegt werden.

## 4.3 Shapematching über dynamische Programmierung

### 4.3.1 Überblick

In Abschnitt 3.3 wurden elastische Methoden vorgestellt, die lokale Formmerkmale vergleichen und einander zuordnen. Der Grundgedanke des Shape Matchings über dynamische Programmierung ist, den Raum der möglichen Zuordnungen einzuschränken. Dies wird erreicht, indem die Merkmale entlang der Kontur als String aufgefasst werden. Das Problem wird also auf einen Sequenzvergleich und damit auf ein *ein*-dimensionales Problem reduziert.
In den späten achtziger Jahren trat die dynamische Programmierung in Verbindung mit lokalen Shapematchingmethoden auf. Gorman, Mitchell und Kuhl [14] unterteilten 1988 geschlossene Konturen in Stücke - die über Fourierkoeffizienten angenähert wurden - und verglichen die Stückfolge als Strings. McConnell et. al. [21] nutzten 3 Jahre später ebenfalls String Matching Methoden, um kleine Kurvenstücke in größeren wieder zu finden und erweiterten dafür ein existierendes Verfahren. Neuere Methoden konzentrierten sich auf die Verbesserung der Vergleichseigenschaften. Basri et. al. [1] stellten 1995 eine Reihe von geforderten Charakteristiken lokaler Vergleichsmethoden auf und untersuchten, welche davon gleichzeitig erfüllbar sind. Genau wie Geiger et. al. [13] verglichen sie die Konturen punktweise und führten einen Glattheitsterm ein, der die Verformung von Formteilen bestraft. Sebastian et. al. [23] erweiterten 2003 die kontinuierliche Herangehensweise um bessere Symmetrieeigenschaften mittels einer *Alignmentcurve* (siehe unten). Manay et. al. [20] verwendeten 2006 ebenfalls eine solche Funktion und verbesserten die Robustheit bei Rauschen, minimierten jedoch das unterliegende Optimierungsproblem über Dijkstras Algorithmus auf einem zyklischen Graphen.

### 4.3.2 Verfahren

Im Folgenden wird das Grundverfahren der elastischen Algorithmen, die das Optimum über dynamische Programmierung berechnen, beschrieben.

#### Kurvenrepräsentation

Die genannten Verfahren repräsentieren die zu vergleichenden Kurven zunächst durch *Kurvenelemente*. Dies können Stücke sein, in die die Kurven unterteilt werden, oder Punkte, an denen Merkmale - wie z.B. die Krümmung - ausgewertet wurden. Der Unterschied ist, dass im ersten Fall die Kurvenstücke weder zu klein noch zu groß gewählt werden dürfen, da sonst Rauschen überbewertet wird, oder der Vergleich der Stücke keine Aussagekraft besitzt. Der zweite Fall ist eher als kontinuierliches Signal entlang der Kurve zu interpretieren, obgleich die Repräsentation

der Kurve im Endeffekt diskretisiert werden muss. Beispiele für stückweise Verfahren sind [14] und [12], punktweise Kurvenvergleiche werden in [23] oder [20] verfolgt.

### Vergleichsmodell

Im nächsten Schritt wird ein Vergleichsmodell für die Kurvenelemente festgelegt. Insbesondere beim Vergleich stückweiser Kurvenelemente muss zunächst definiert werden, was unter Distanz zu verstehen ist. Der Unterschied in der Kurvenkrümmung wird meistens über den quadratischen Abstand gemessen.
Oftmals wird ein *Glattheitsterm* in die theoretische Formulierung eingebracht. Er sorgt dafür, dass das Matching von Elementen möglichst ohne Stauchungen bzw. Streckungen entlang einer der beiden Konturen erfolgt. Diese können die Distanz zwischen den Formen zwar verringern, machen jedoch die Korrespondenzabbildung unwahrscheinlicher. Gemessen wird beispielweise die Glattheit des Vektorfeldes der Differenzenvektoren korrespondierender Punkte oder die Abtastgeschwindigkeit der Konturen. Beispiele für Glattheitsterme sind in [13], [1], [20] und [23] zu finden.

### Graphenkonstruktion und Optimierung

Schließlich wird ein Graph konstruiert, wie es im Abschnitt über Sequenzvergleiche beschrieben wurde. Dazu müssen Stetigkeits- und Randbedingungen festgelegt werden. Sie hängen von den gewünschten Eigenschaften des Vergleichsmodells ab. Aus dem Stetigkeitsmodell ergibt sich die Kantenstruktur. Die Randbedingung bestimmt die Ausmaße des Graphen und die Startknoten bei der Berechnung der kürzesten Wege. Die Kantengewichte werden über das Vergleichsmodell der Kurvenelemente und dem Glattheitsterm bestimmt. Die Optimierung erfolgt über einen geeigneten Kürzeste-Wege Algorithmus. Die Distanz wird meist als die Länge des kürzesten Weges definiert.

### Unterschiede

Die Verfahren über dynamische Programmierung unterscheiden sich zum Teil stark durch die Invarianz gegenüber Translation, Rotation, Skalierung und Toleranz gegenüber Artikulation und Verdeckung und Symmetrie des Abstandsmaßes. Dies hängt nicht zuletzt mit der sehr unterschiedlichen Art der Repräsentation von Kurven zusammen. Des Weiteren gibt es solche Verfahren, die nur auf offenen Kurven operieren [21]. Die meisten Verfahren lassen sich jedoch auf naive Weise stets auf geschlossene Konturen verallgemeinern (siehe Abschnitt 4.4).

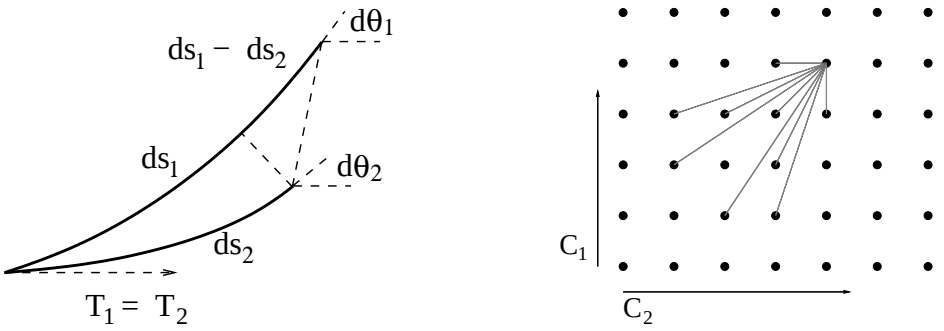



Abbildung 4.2: *links*: Die infinitesimal kleinen Kurvenstücke werden über Tangente und Ursprung ausgerichtet und Winkel- und Längendifferenz gemessen

### 4.3.3 Beispiel: Sebastian et. al.

Im Folgenden soll ein neueres Verfahren über dynamische Programmierung genauer betrachtet werden. Dieser Unterabschnitt soll einerseits die besprochenen Aspekte näher beleuchten, auf der anderen Seite werden Konzepte vorgestellt, die im nächsten Kapitel Verwendung finden. Insbesondere die kontinuierliche Handhabung und die Einführung der *Alignmentcurve* werden im nächsten Kapitel aufgegriffen.

Sebastian, Klein und Kimia [23] präsentierten 2003 ein punktweises Verfahren zum Vergleich von offenen und geschlossenen Kurven, das intrinsische Eigenschaften der Kurven zum Vergleichsgegenstand macht. Zunächst wird von offenen Kurven ausgegangen, deren Anfangs- und Endpunkte korrespondieren.

#### Ähnlichkeit infinitesimaler Kurvenstücke

Die theoretische Formulierung geht von einem Vergleichsmodell für infinitesimal kleine Kurvenstücke aus. Beim Vergleich wird die Verformungsenergie gemessen, die benötigt wird, um die Stücke anzugleichen. Die Anfangspunkte der Kurvenstücke und deren Tangenten werden zunächst zur Deckung gebracht und mit der x-Achse ausgerichtet (siehe Abb. 4.2). Die *Verformungenergie* besteht dann aus zwei Summanden :

**Streckung** Besteht aus dem Längenunterschied $|ds - ds'|$ der Kurvenstücke.

**Biegung** Wird durch den Krümmungsunterschied repräsentiert $\lambda|d\theta - d\theta'|$.

wobei $d\theta, d\theta'$ die Winkel zwischen den Endpunkttangenten und der x-Achse sind. $\lambda$ ist ein Gewichtungsfaktor.

### Alignmentcurve

Die zu vergleichenden Kurven $C_1$ und $C_2$ seien nun nach Kurvenlänge $L$ und $L'$ parametrisiert. Gesucht wird eine Kurve

$$f := (f_1, f_2) : [0, l] \rightarrow [0, L] \times [0, L'],$$

- von den Autoren *Alignmentcurve* genannt - die ebenfalls nach Bogenlänge $l$ parametrisiert ist. Die Kurve bildet den Punkt $C_1(f_1(t))$ auf den Punkt $C_2(f_2(t))$ ab für alle $t \in [0, l]$. Da die Komponenten von $f$ als *stetig* und *monoton steigend* angenommen werden, bleibt die Reihenfolge der Punkte auf den Kurven erhalten und das Vergleichsproblem wird auf ein eindimensionales reduziert. Eine *optimale Alignmentcurve* bildet entlang der Kurven möglichst ähnliche infinitesimale Stücke aufeinander ab - also Stücke mit möglichst kleiner Verformungsenergie.

### Energiefunktional

Die Grundannahme ist, dass die Ähnlichkeit der Kurven zueinander gleich der Summe der Ähnlichkeiten ihrer Teilstücke ist. Daher wird obige Verformungsenergie korrespondierender infinitesimaler Teilstücke aufintegriert:

$$\begin{aligned} F_{C_1}^{C_2}(f) &= \int_0^l \left| \frac{df_1(x)}{dx} - \frac{df_2(x)}{dx} \right| - \lambda \left| \frac{d\theta(f_1(x))}{dx} - \frac{d\theta'(f_2(x))}{dx} \right| dx \\ &= \int_0^l \left| \frac{df_1}{dx} - \frac{df_2}{dx} \right| - \lambda \left| \frac{d\theta(f_1(x))}{df_1} \frac{df_1}{dx} - \frac{d\theta'(f_2(x))}{df_2} \frac{df_2}{dx} \right| dx \\ &= \int_0^l |f_1(x)' - f_2(x)'| - \lambda |\kappa_1(f_1(x)) f_1(x)' - \kappa_2(f_2(x)) f_2(x)'| \, dx \end{aligned} \quad (4.4)$$

Der erste Term ist ist ein **Glattheitsterm** und bestraft ungleiche Abtastgeschwindigkeiten der beiden Kurven. Der **Datenterm** (zweiter Summand) ist umso kleiner, je ähnlicher die Kurvenkrümmung der verglichenen Stücke ist. Eine optimale Korrespondenzabbildung $f^*$ minimiert dieses Funktional. Der Wert des Funktionals für $f^*$ ist die **Distanz** der beiden Kurven zueinander.

### Optimierung des Funktionals

Für die diskrete Optimierung des Funktionals (4.4) werden die Kurven zunächst äquidistant abgetastet. Der Graph, auf dem die kürzesten Wege berechnet werden, entsteht durch ein rechteckiges Knotengitter (siehe Abb. 4.2). Dessen Spalten stehen von links nach rechts für die Punkte auf der Kontur $C_1$, die Zeilen von unten nach oben repräsentieren die Punkte der Kontur $C_2$. Das Kantenschema für jeden Knoten wird wie in Abbildung 4.2 gewählt. Entscheidend ist, dass jede Kante für die gegenseitige Abbildung der beiden durch sie repräsentierten Kurvenstücke steht und

nicht nur für die der Endpunkte. Das Kantengewicht ist folglich gleich der benötigten Verformungsenergie der Kurvenstücke zueinander. Gesucht wird der kürzeste Weg vom linken unteren zum rechten oberen Knoten.

#### Eigenschaften

Dadurch, dass lediglich die Krümmung in den Ansatz eingeht, ist das Verfahren sowohl translations- als auch rotationsinvariant. Über die optimalen Skalierungsfaktoren der beiden Konturen führen Sebastian et. al. allerdings einen Gradientenabstieg aus. Durch die Darstellung der Korrespondenzfunktion als Kurve (*Alignment-curve*), wird die Symmetrie des Verfahrens gewährleistet. Dadurch, dass stets Kurven*abschnitte* anstelle einzelner Punkte aufeinander abgebildet werden, ist das Berechnungsergebnis der Distanz zwischen zwei Kurven relativ robust gegenüber der Auflösung der Kurvenabtastung.

#### Nachteile

Nachteilig an der Umsetzung des Verfahrens ist die relativ große und recht beliebig gewählte Kantenmenge im Graphen, die eine Konsequenz der abschnittweisen Abbildungssicht ist. Vor allen Dingen leidet darunter die Effizienz, der Gewinn mehrerer Kantensteigungen ist hingegen fraglich. Zudem ist die praktische Bestimmung der Kantengewichte, sprich, der Verformungsenergie zwischen den Kurvenstücken, unklar. Die genannten Probleme sollen in Kapitel 5 mit einem neuen Ansatz ausgeglichen werden.

## 4.4 Vergleich geschlossener Konturen

Für den Kurvenvergleich über dynamische Programmierung wurde die Aufrechterhaltung der Punktordnung bei der Abbildung von einer Kurve auf die andere gefordert. Sollen geschlossene Konturen verglichen werden entsteht daraus eine Schwierigkeit. Problemkonstellation und bekannte Lösungsansätze werden in diesem Abschnitt besprochen.

### 4.4.1 Problemanalyse

Bei der Korrespondenzberechnung offener Kurvenstücke ging die *Randbedingung* ein, dass jeweils Anfangs- und Endpunkte der Kurven korrespondieren. Dadurch konnte die Optimierung über die Berechnung des kürzesten Weges von der Anfangs- zur Endkorrespondenz im Graphen formuliert werden. Beim Vergleich geschlossener Konturen kann hingegen keine sinnvolle Randbedingung festgelegt werden, da es weder Anfang noch Ende gibt. Um den kürzesten Weg im Graphen zu berechnen, braucht man jedoch einen Anfangs- und Endpunkt.

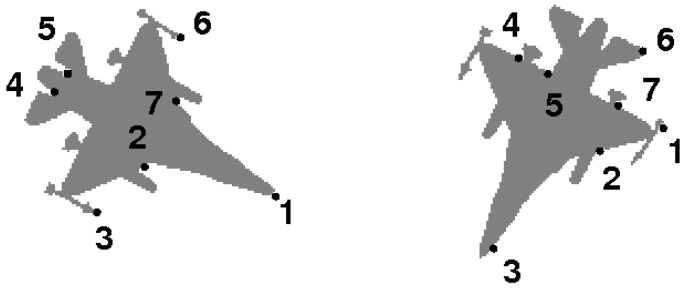



Abbildung 4.3: Wird beim Vergleich geschlossener Konturen die Initialkorrespondenz falsch gewählt, führt das selbst bei identischen Formen zu falschen Korrespondenzabbildungen.

### Initialkorrespondenz und optimale Korrespondenzabbildung

Zunächst ist klar, dass bei geschlossenen Kurven im Graphen die Anfangskorrespondenz gleich der Endkorrespondenz sein muss. Wir nennen diesen Knoten die *Initialkorrespondenz*. Legt man eine beliebige Initialkorrespondenz fest, so kann selbst bei identischen Kurven das Korrespondenz-Ergebnis unzureichend sein (siehe Abb. 4.3), weil die Voraussetzung des Algorithmus nicht stimmt und der kürzeste Weg zwischen den falschen Knoten berechnet wird. Die *optimale Korrespondenzabbildung* ist daher definiert als der kürzeste Weg von allen möglichen Wegen, die von einer beliebigen Korrespondenz ausgehen und dort auch enden.

### Suchstrategie

Da es - bei $n$ Kurvenelementen auf beiden Kurven - $n^2$ verschiedene Initialkorrespondenzen gibt, müssten - da die Berechnung jedes kürzesten Weges $O(n^2)$ Zeit benötigt - $n^4$ Rechenschritte ausgeführt werden, um die optimale Korrespondenzabbildung zu berechnen. Allerdings, durch die zyklische Struktur der Kurven, kommt als Initialkorrespondenz *jede* Korrespondenz in Frage, die in der optimalen Abbildung enthalten ist. Der Weg resultiert für jeden dieser Knoten in derselben Knotenfolge und denselben Kosten. Da jedoch jedes Kurvenelement auf $C_1$ mindestens einen Korrespondenzpartner auf der anderen Kurve besitzt, reicht es, einen Punkt auf $C_1$ fest zu wählen und den Korrespondenzpartner unter sämtlichen Kurvenelementen von $C_2$ zu suchen. Eine optimale Initalkorrespondenz muß unter ihnen sein. Es bleiben also noch $n$ Schritte für die Suche nach der Initialkorrespondenz.

### Naive Lösung

Die naive Lösung für den Vergleich geschlossener Konturen sieht wie folgt aus: Der Graph mit den Korrespondenzen wird derart erweitert, dass jeder zu berechnende

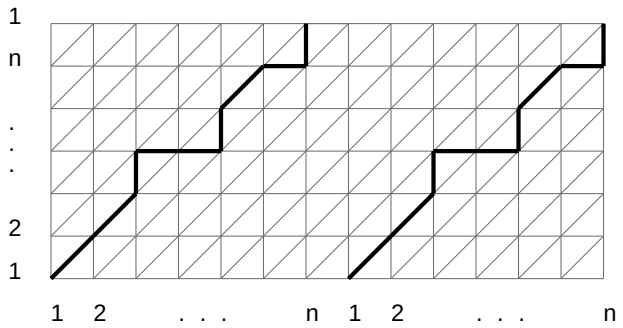



Abbildung 4.4: Der Graph elastischer Verfahren kann derart erweitert werden, dass er sämtliche kürzeste Wege enthält, die gültigen Korrespondenzabbildungen entsprechen. Nach Verdopplung der Graphengröße grenzen zwei identische kürzeste Wege (dunkle Linie) die Knoten für alle kürzesten Wege dazwischen ein.

kürzeste Weg in ihm enthalten ist. Dazu werden die Kurvenelemente von $C_1$ *zweimal* auf der x-Achse abgetragen und die *unterste* Zeile des resultierenden Graphen oben wiederholt (siehe Abb. 4.4). Die Zeilen sind von unten nach oben, die Spalten von links nach rechts durchnummeriert und Knoten $(i, j)$ steht in Spalte $i$ und Zeile $j$. Nun kann von jedem Knoten $(i, 1), i \in \{1, ..., n\}$ entlang der untersten Reihe der kürzeste Weg zum Knoten $(i + n, n)$ der obersten Reihe berechnet werden. Der Weg mit den geringsten Kosten unter ihnen entspricht der gesuchten global optimalen Korrespondenzabbildung. Dies führt wie bereits gezeigt zu einer **kubischen Laufzeit**.

### 4.4.2 Bekannte Lösungsansätze

Die Suche nach der Initalkorrespodenz ist rechenintensiv. Es werden dabei viele kürzeste Wege berechnet, obgleich nur *ein* kürzester Weg gesucht wird. Um dies zu verhindern wurden von verschiedenen Autoren unterschiedliche Ansätze verfolgt. Alle bekannten Ansätze sind entweder *effizientere Suchstrategien* oder *Heuristiken*, die jedoch das Problem der globalen Optimierung nicht in einem Schritt zusammenfassen.

**Sebastian et. al.** betrachten für die globale Optimierung einen erweiterten Graphen ähnlich dem in Abbildung 4.4, der sämtliche zu betrachtenden kürzesten Wege enthält. Für jede Berechnung des kürzesten Weges $\alpha_i$ zwischen den Knoten $(i, 1)$ und $(i+n, n)$ müssen die $n^2$ Knoten im eingegrenzten quadratischen Bereich betrachtet werden.
Sebastian nutzt nun die Beobachtung, dass sich zwei kürzeste Wege $\alpha_i$ und $\alpha_j$ nicht kreuzen können. Würden sie sich kreuzen, so müssten sie sich - um ihr jeweiligen Zielknoten erreichen zu können - topologisch bedingt *zweimal* kreuzen. Dies würde

jedoch implizieren, dass die Teilwege von $\alpha_i$ und $\alpha_j$ zwischen den Schnittpunkten keine kürzesten Wege sind, also $\alpha_i$ und $\alpha_j$ selbst keine optimalen Wege sein können. Die Knoten, die zur Berechnung des kürzesten Weges $\alpha_i$ betrachtet werden müssen, liegen daher zwischen den beiden Wegen $\alpha_j$ und $\alpha_k$ für alle $j < i < k$. Berechnet man die kürzesten Wege in einer Reihenfolge ähnlich der binären Suche, so halbiert sich aufgrund der bereits bekannten kürzesten Wege die Anzahl der zu betrachtenden Knoten nach jeder Berechnung. Daher ergibt sich die Laufzeit $O(n^2 \log(n))$ für die Berechnung des globalen Minimums beim Shape Matching geschlossener Konturen.

Wesentlich heuristischer ist die Reduktion des Suchraums von **Gdalyahu und Weinshall** [12]. Für jede mögliche Startkorrespondenz wird nur eine kleine, feste Anzahl von Berechnungsschritten durchgeführt. Werden die $k$ Initialkorrespondenzen mit den kleinsten Kosten auf einen Parameterraum projiziert, so stellt man fest, dass sie sich um einen Punkt häufen. Nur für solche Initialkorrespondenzen, die nicht zu weit vom Zentrum des Häufungspunkts entfernt sind, wird die Berechnung bis zum Ende durchgeführt.

Eine ähnlich heuristische Strategie verfolgen **Ling und Jacobs** [17]. Sie richten die zu vergleichenden Formen entsprechend ihrer Hauptmomente aus und führen die Berechnung nur mit $k$ Initialkorrespondenzen in der näheren Umgebung durch. Da $k$ eine Konstante mit $k \ll n$, bleibt die Laufzeit quadratisch; die globale Optimierung des Problems kann jedoch nicht garantiert werden.

Auch **Manay et.al.** [20] verweisen auf Strategien, bei denen die Ausrichtung der Formen zueinander geschätzt wird, um den Suchraum der Initialkorrespondenz zu verkleinern. Alternativ schlagen sie vor, lediglich Initialkorrespondenzen zwischen Punkten mit ausgeprägten Merkmalen auf beiden Konturen zu suchen, da es in der Regel wenige davon gibt. Beide Verfahren sind Heuristiken und schließen eine Garantie für das Finden eines globalen Optimums aus. Generell kommen Ausrichtungsschätzungen nicht gut mit Artikulationen zurecht.

Ein Hauptziel dieser Arbeit ist, ein Shape Matching Verfahren vorzustellen, welches die Suche nach der Initialkorrespondenz für geschlossene Konturen vermeidet. Dadurch können viele Berechnungsschritte für mögliche Initialkorrespondenzen gespart werden, die sich hinterher als nicht optimal herausstellen. Dafür wird im nächsten Kapitel ein Ansatz über dynamische Programmierung vorgestellt. In Kapitel 6 wird das Problem dann so umgeformt, dass auf Suchstrategien verzichtet wird.

# Kapitel 5

# Ein elastisches Vergleichsmodell

Als Grundlage weiterer Ausführungen soll ein elastisches Vergleichsmodell sowohl offener als auch geschlossener Kurven hergeleitet werden. Die Grundlage bildet ein Variationsansatz, der in Abschnitt 5.1 als kontinuierliches Energiefunktional formuliert wird. Das zugehörige Distanzmaß wird ebenfalls definiert. Anschließend wird in Abschnitt 5.2 das Problem diskretisiert und über dynamische Programmierung gelöst. Experimente mit dem eingeführten Verfahren beschließen das Kapitel.

## 5.1 Herleitung des Modells

Das Modell orientiert sich an bereits besprochenen lokalen Vergleichsmethoden. Zunächst wird festgelegt, welche Erwartungen an das Verfahren gestellt werden. Dabei muss zwischen dem Modell selbst und dessen algorithmische Umsetzung gegebenenfalls differenziert werden:

**Invarianz** Es wird vollständige Invarianz des Modells bezüglich Translation und Rotation erwartet. Gegenüber Skalierung soll sich der Algorithmus zumindest tolerant verhalten.

**Symmetrie** Der Vergleich von zwei Kurven soll in beide Richtungen dieselbe Korrespondenzabbildung und Distanz ergeben.

**Robustheit** Gegenüber Verdeckung, Artikulation und Rauschen soll der Algorithmus robust sein.

**Eindeutigkeit** Der Algorithmus soll von der Auflösung der Kurvenabtastung relativ unabhängig sein, indem das kontinuierliche Ergebnis angenähert wird.

**Universalität** Das Modell soll offen sein bezüglich der Kurvenmerkmale, die miteinander verglichen werden, d.h. es soll für andere lokale Formdeskriptoren außer der Krümmung wohldefiniert bleiben.

Trotz der geforderten Universalität soll zunächst die Kurvenkrümmung in die Formulierung eingehen. Außerdem sollen ohne Einschränkung - zumindest des Modells - zunächst *offene* Kurven verglichen werden, bei denen Anfangspunkt nicht gleich Endpunkt ist. Die Form der Kurven $C_1$ und $C_2$ wird über ihre Krümmungsfunktionen $\kappa_{1,2} : [0, 1] \rightarrow \mathbb{R}$ repräsentiert (siehe Abschnitt 2.1).

**Vergleichbare Verfahren**

Verbesserungen gegenüber vergleichbaren Verfahren sollen insbesondere in punkto Effizienz, Einfachheit und Eindeutigkeit der praktischen Berechnung geschehen. Auch wird eine einfache sowie leistungsfähige theoretische Formulierung angestrebt. Über die Effizienz zu Sebastian wurde bereits in Abschnitt 4.3 gesprochen; ebenso über die Unklarheit bezüglich der Berechnung der Verformungsenergie. Manay et. al. verwenden den relativ langsamen Algorithmus von Dijkstra zur Optimierung, da sie einen zyklischen Graphen konstruieren. Des Weiteren ist ihre theoretische Herleitung recht kompliziert.

### 5.1.1 Korrespondenzfunktion

Berechnet werden soll eine “optimale” Korrespondenzfunktion

$$m : [0, 1] \rightarrow [0, 1] \tag{5.1}$$

Sie bestimmt für jeden Punkt $s$ der Quellkurve, auf welchen Punkt $m(s)$ der Zielkurve er abgebildet wird. Folgende Bedingungen werden an die Funktion $m$ gestellt:

**Stetigkeit** Beliebige Teilstrecken der Quellkurve sollen auf der Zielkurve nicht zerrissen werden, daher muss $m$ stetig sein.

**Monotonie** Bei der Abbildung von $C_1$ auf $C_2$ soll die Ordnung der Punkte entlang der Quellkurve beibehalten werden: ist ein Punkt $a$ entlang der Quellkurve vor einem Punkt $b$, so soll $b$ auf der Zielkurve nicht vor dem Punkt $a$ sein. Formal also: $\forall s_1, s_2 \in [0, 1] : s_1 \leq s_2 \Rightarrow m(s_1) \leq m(s_2)$

**Randbedingung** Kurven werden immer als Ganzes verglichen, Die Quellkurve soll nicht auf einen Teil der Zielkurve abgebildet werden können. Daraus folgen die Randbedingungen $m(0) = 0$ und $m(1) = 1$. Die Endpunkte der Kurven korrespondieren damit grundsätzlich.

*Stetigkeit* und *Monotonie* zusammen ergeben für die Ableitung: $m'(s) \geq 0$. Die Ableitung der Korrespondenzfunktion $m'(s)$ gibt das Ausmaß von *Stauchung* ($m'(s) < 1$) und *Streckung* ($m'(s) > 1$) der Umgebung des Punktes $s$ bei der Abbildung an. Diese Verformung von Kurventeilen soll bei der Bewertung von Korrespondenzabbildungen bestraft werden.

Die Stauchung eines beliebigen Quellkurvenstückes auf einen einzelnen Punkt auf der Zielkurve ist äquivalent dazu, dass die Steigung von $m$ gleich 0 ist. Allerdings muss umgekehrt für die Streckung eines einzelnen Punktes auf ein Zielkurvenstück die Ableitung gegen unendlich gehen. Dies macht es schwierig, die Stauchung und Streckung symmetrisch zu bestrafen. Daher wird im Folgenden eine **symmetrische Form der Korrespondenzabbildung** gewählt:

$$\vec{m} := (m_1, m_2) : [0,1] \rightarrow [0,1]^2 \tag{5.2}$$

Für alle $s$ korrespondiert der Punkt $m_1(s)$ auf $C_1$ zum Punkt $m_2(s)$ auf $C_2(s)$. Die Bedingungen an $m$, Stetigkeit und Monotonie übertragen sich direkt auf die beiden Komponenten von $\vec{m}$. Die Randbedingungen lauten $\vec{m}(0) = (0,0)^T$ und $\vec{m}(1) = (1,1)^T$. Die Korrespondenzfunktion ist aufgrund der Stetigkeitsbedingung eine Kurve auf dem Einheitsquadrat, deren Anfangspunkt bei $(0,0)$ und deren Endpunkt bei $(1,1)$ ist. Sie ist auf dem Einheitsintervall parametrisiert.
Die Korrespondenzfunktion in dieser Form ist zu vergleichen mit der *Warping-function* oder *Alignmentcurve* diverser Autoren wie z.B. Sebastian et. al. [23] oder Manay et. al. [20] mit dem Unterschied, dass $\vec{m}$ nicht über die Länge parametrisiert wird. Zu beachten ist, dass die Korrespondenzkurve $\vec{m}$ die Korrespondenzabbildung zwischen den beiden Kurven nicht eindeutig beschreibt, da die genaue Kurvenparametrisierung offen ist.
Die Menge $K$ aller **gültigen Korrespondenzfunktionen** ist die Menge aller Funktionen der Form (5.2), die den genannten Bedingungen der Stetigkeit, der Monotonie und den Randbedingungen genügen.

### 5.1.2 Energiefunktional

Es wird ein Maß für die Güte einer Korrespondenzfunktion benötigt. Wir führen das folgende Energiefunktional ein:

$$E_{\kappa_1}^{\kappa_2}(m) = \int_0^1 \left[\, \kappa_1(m_1(s)) - \kappa_2(m_2(s))\right]^2 \, d\vec{m}(s) + \alpha \int_0^1 |m_1'(s) - m_2'(s)| ds \tag{5.3}$$

Das Funktional besteht aus zwei Termen:

#### Datenterm

Der erste Summand bestraft Unähnlichkeit der Krümmung korrespondierender Punkte entlang der Kurven quadratisch. Der Term entspricht dem Kurvenintegral über die Krümmungsdifferenz $\kappa_d : [0,1]^2 \rightarrow \mathbb{R}^+$ entlang der Korrespondenzkurve $\vec{m}$. Das Kurvenintegral über die Funktion $f$ entlang der Kurve $\vec{m}$ ist definiert durch

$$\int f(\vec{m}(s))\, d\vec{m}(s) = \int f(m_1(s), m_2(s)) \sqrt{m_1'(s)^2 + m_2'(s)^2}\, .$$

Der Normierungsfaktor unter der Wurzel macht den Integralwert unabhängig von der Kurvenparametrisierung. Ohne weiteres kann im Datenterm die Krümmung durch andere lokale Deskriptoren ersetzt werden.

#### Glattheitsterm

Der zweite Summand in (5.3) bestraft die Streckung oder Stauchung eines Kurvenabschnittes bei der Abbildung auf die Zielkurve. Je unterschiedlicher die Abtastgeschwindigkeiten der beiden Kurven sind, desto mehr Streckung bzw. Stauchung erfolgt bei der Abbildung und desto größer ist der Integralwert. Anschaulich sorgt der Term dafür, dass die Korrespondenzkurve $\vec{m}$ eine möglichst direkte Verbindung zwischen den Punkten $(0,0)$ und $(1,1)$ bildet. Der Parameter $\alpha$ bestimmt die Stärke der Glattheitsbestrafung. Geht $\alpha$ gegen unendlich, wird die Identitätsabbildung zwischen den beiden Kurven forciert. Nun kann bestimmt werden, wann eine Korrespondenzfunktion als *optimal* angesehen wird:

**Definition.** *Für eine **optimale Korrespondenzabbildung** $m_{opt}$ zwischen zwei Kurven $\kappa_1$ und $\kappa_2$ gilt:*

$$m_{opt} := \underset{m \in K}{\operatorname{argmin}} \, E_{\kappa_1}^{\kappa_2}(m)$$

### 5.1.3 Distanzmaß

Der Wert des Funktionals einer optimalen Korrespondenzfunktion soll als Distanz der beiden verglichenen Formen definiert werden. In diesem Unterabschnitt soll überprüft werden, wie sinnvoll ein solcher Distanzbegriff ist.

Das Funktional (5.3) misst sowohl Unterschiede in der Krümmung korrespondierender Punkte, als auch die Verformungsenergie (Streckung und Stauchung), die nötig ist, um die Kurven aufeinander abzubilden. Je kleiner der Wert des Funktionals einer gegebenen Korrespondenzfunktion ist, desto ähnlicher sind korrespondierende Punkte und desto weniger Energie ist nötig, diese Punkte aufeinander abzubilden. In diesem Sinn gilt folglich: je kleiner die Energie, desto ähnlicher die Kurven. Weiterhin ist ein gewisses Stetigkeitsverhalten der Energie bezüglich der verglichenen Formen zu erwarten. Kleine Veränderungen der Kurven haben demnach auch kleine Veränderungen des Funktionalwertes zur Folge, auch wenn Sprünge der Energie nicht ausgeschlossen sind.
Für einen sinnvollen Distanzbegriff sollte außerdem gelten, dass dieser den Eigenschaften einer Metrik oder zumindest denen einer Semimetrik genügt (siehe Kapitel 2). Durch die Formulierung des Funktionals (5.3) über die Korrespondenzkurve $\vec{m}$ ist die Symmetrie gegeben. Die Verwendung des quadratischen Abstandes im Datenterm und des Betrages im Glattheitsterm garantiert eine positive Energie. Fest steht auch, dass für zwei identische Kurven - identisch bis auf Translation, Rotation

und Skalierung - die Energie gleich 0 ist.

Insgesamt ist es also durchaus sinnvoll den Wert des Funktionals auf einer optimalen Korrespondenzfunktion als Distanzmaß zu definieren:

**Definition.** *Die Distanz zwischen zwei Kurven $\kappa_1$ und $\kappa_2$ ist definiert durch:*

$$D(\kappa_1, \kappa_2) := \min_{m \in K} E_{\kappa_1}^{\kappa_2}(m) \tag{5.4}$$

Für weitere Untersuchungen des Distanzmaßes siehe Abschnitt 5.3.

## 5.2 Globale Minimierung des Energiefunktionals

Die Minimierung des Funktionals (5.3) soll über dynamische Programmierung vorgenommen werden. Daher werden diskrete Repräsentationen der Korrespondenzfunktion und des Energiefunktionals benötigt.

### 5.2.1 Diskretisierung der Korrespondenzabbildung

Die Kurven werden beide äquidistant abgetastet. Der Einfachheit halber nehmen wir an, dass beide Kurven mit $n$ Punkten abgetastet werden. An jedem der $n$ Punkte wird die Kurvenkrümmung berechnet, so dass sich die diskretisierten Kurven als zwei Vektoren $\vec{\kappa}^1 = [\kappa_0^1, \kappa_1^1, ..., \kappa_{n-1}^1]^T$ und $\vec{\kappa}^2 = [\kappa_0^2, \kappa_1^2, ..., \kappa_{n-1}^2]^T$ darstellen lassen.
Die diskrete Version der Korrespondenzabbildung ist eine Abbildung

$$m_d := (m_d^1, m_d^2) : \{1, ..., L\} \rightarrow \{0, ..., n-1\}^2$$

$L$ bezeichnet dabei die Anzahl der durch die Abbildung festgelegten Korrespondenzen. Für ein $i \in \{1, ..., L\}$ schreibt $m_d(i) = (k, l)$ vor, dass der $k$-te Punkt auf der Quellkontur auf den $l$-ten Punkt der Zielkontur abgebildet wird. Die *Stetigkeit*, *Monotonie* und *Randbedingung* der kontinuierlichen Korrespondenzabbildung wird wie folgt auf die diskrete Version übertragen:

**Definition.** *Eine gültige, diskrete Korrespondenzfunktion zwischen zwei diskretisierten Kurven mit jeweils $n$ Punkten genügt folgenden Bedingungen (vgl. Abschnitt 5.1.1):*

1. $\forall i \in \{1, ..., L-1\} \Rightarrow m_d^l(i+1) - m_d^l(i) \in \{0, 1\},\ l \in \{1, 2\}$
2. $m_d(1) = (0, 0)$ *und* $m_d(L) = (n-1, n-1)$

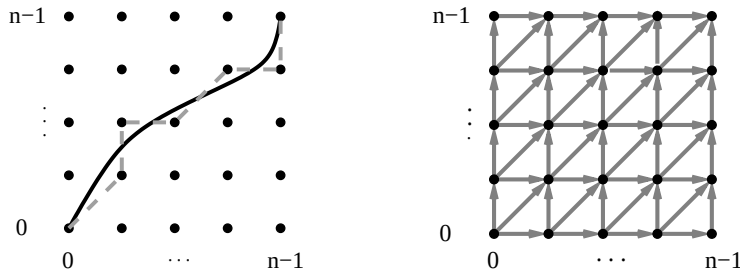



Abbildung 5.1: *links*: Der Bildraum der Korrespondenzkurve $\vec{m}$ (durchgezogene Linie) wird durch ein Punktgitter unterteilt und die Kurve durch einen Streckenzug angenähert (gestrichelte Linie).*rechts*: Der für die Optimierung verwendete Graph $G$ ergibt sich intuitiv aus der Diskretisierung des Bildraumes links.

Die erste Bedingung ist die intuitive Übertragung der Stetigkeit und der positiven Ableitung der kontinuierlichen Korrespondenzabbildung auf den diskreten Fall. Anschaulich bedeutet sie, dass die Korrespondenzabbildung auf den Konturen weder rückwärts gehen, noch Konturpunkte überspringen darf. Das zweite Kriterium entspricht den Randbedingungen der kontinuierlichen Version. Eigenschaft 1 schränkt den Vorgänger einer Korrespondenz auf drei Möglichkeiten ein:

$$m_d(i) = (k,l) \Rightarrow m_d(i-1) \in \{(k-1,l),(k,l-1),(k-1,l-1)\}$$

### 5.2.2 Diskretisierung des Energiefunktionals

Das Energiefunktional (5.3) integriert die Funktion der Krümmungsdifferenzen entlang der Korrespondenzkurve $\vec{m}$ auf. Die Kurve verläuft innerhalb des Einheitsquadrates in der Ebene $\mathbb{R}^2$.
Zunächst wird der Bildraum der Korrespondenzkurve diskretisiert: das Einheitsquadrat im $\mathbb{R}^2$ wird in ein $n \times n$-Punktgitter unterteilt (Abb. 5.1). Jeder Punkt entspricht einer Korrespondenz $(k,l)$ zwischen den zu vergleichenden Kurven. Eine Korrespondenzkurve $\vec{m}$ wird nun durch einen Streckenzug vom Knoten $(0,0)$ zum Knoten $(n-1,n-1)$ angenähert (Abb. 5.1). Die Bedingung dabei ist, dass die Knotenfolge des Streckenzugs eine gültige diskrete Korrespondenzabbildung repräsentieren muss, wie sie im vorherigen Unterabschnitt definiert wurde. Entlang des Streckenzuges werden Daten- und Glattheitsterm aufintegriert.

#### Datenterm

Der Datenterm am Korrespondenzpunkt $(k,l)$ ist

$$d_{k,l} := (\kappa_k^1 - \kappa_l^2)^2.$$

Da die Krümmungen zwischen den Diskretisierungspunkten unbekannt sind, wird entlang jedes Linienstücks zwischen den Datentermen der Endknoten interpoliert. Der Datenterm der diskretisierten Energie ergibt sich daraus wie folgt:

$$\begin{aligned} E_{data}(m_d) :&= \sum_{i=1}^{L-1} \int_0^1 \left( d_{(m_d^1(i),m_d^2(i))} \cdot s + d_{(m_d^1(i+1),m_d^2(i+1))} \cdot (1-s) \right) \\ &\qquad \cdot \frac{1}{n-1} \underbrace{\sqrt{m_d^1(i+1) - m_d^1(i) + m_d^2(i+1) - m_d^2(i)}}_{C_i} \, ds \\ &= \frac{1}{n-1} \sum_{i=0}^{L-1} C_i \cdot \left( \frac{d_{m_1(i),m_2(i)} + d_{m_1(i+1),m_2(i+1)}}{2} \right) \end{aligned}$$

Da es für eine Korrespondenz $m_d(i) = (k, l)$ nur eine von drei Vorgängerkorrespondenzen geben kann, gilt für $C_i$:

$$C_i = \begin{cases} 1 & \text{falls } \; m_d(i-1) = (k-1, l) \text{ oder } (k, l-1) \\ \sqrt{2} & \text{falls } \; m_d(i-1) = (k-1, l-1) \end{cases}$$

**Glattheitsterm**

Der Glattheitsterm wird analog entlang des Streckenzugs berechnet:

$$\begin{aligned} E_{smooth}(m_d) :&= \alpha \sum_{i=1}^{L-1} \int_0^1 \left| \left( \frac{m_d^1(i+1) - m_d^1(i)}{n-1} \right) - \left( \frac{m_d^2(i+1) - m_d^2(i)}{n-1} \right) \right| ds \\ &= \frac{\alpha}{n-1} \sum_{i=1}^{L-1} \underbrace{\int_0^1 \left| (m_d^1(i+1) - m_d^1(i)) - (m_d^2(i+1) - m_d^2(i)) \right| ds}_{D_i} \end{aligned}$$

Für die Summanden $D_i$ gilt:

$$D_i = \begin{cases} 1 & \text{falls } \; m_d(i-1) = (k-1, l) \text{ oder } (k, l-1) \\ 0 & \text{falls } \; m_d(i-1) = (k-1, l-1) \end{cases}$$

Die Summanden des Glättungsterms können also *zwei* verschiedene Werte annehmen. Falls das korrespondierende Linienstück des Streckenzugs vertikal oder horizontal ist, ist der Summand $1$, bei einem diagonalen Verbindungsstück ist er $0$.

Die diskretisierte Version des Funktionals (5.3) ist die Summe von Daten- und Glattheitsterm:

$$E_d(m_d) := \frac{1}{n-1} \sum_{i=1}^{L-1} C_i \left( \frac{d_{m_1(i),m_2(i)} + d_{m_1(i+1),m_2(i+1)}}{2} \right) + \frac{\alpha}{n-1} \sum_{i=1}^{L-1} D_i \quad (5.5)$$

Der Vorteil dieser Herleitung ist, dass die Energie nach der Diskretisierung des Bildbereichs weiterhin kontinuierlich berechnet wird. Das hat zur Folge, dass die Kosten für die Abbildung der Kurven*abschnitte* zwischen den Abtastpunkten angenähert werden und nicht nur die der Abtastpunkte selbst berechnet wird. Dadurch wird die Energie - obwohl für kleines $n$ ungenau - relativ unabhängig von der Abtastrate der Kurven. Des Weiteren ist das Kantenschema des Graphen elementar gehalten, was zu einer effizienten Optimierung über dynamische Programmierung beiträgt.

### 5.2.3 Minimierung über dynamische Programmierung

Die globale Minimierung des diskreten Funktionals wird über dynamische Programmierung gelöst. Dazu wird ein Graph $G$ konstruiert. Die Knotenmenge $V$ entspricht den Korrespondenzen zwischen Kurvenpunkten und ist ein regelmäßiges $n \times n$-Knotengitter, $V := \{(i,j),\ 0 \leq i,j < n\}$. Die Gitterspalten stehen für die Knoten $0, 1, ..., n-1$ auf der ersten Kontur $C_1$, die Zeilen für die entsprechenden Punkte auf der zweiten Kontur $C_2$. Ein Knoten $(i,j)$ entspricht der Korrespondenz des $i$-ten Punktes auf $C_1$ zum Punkt $j$ auf $C_2$. Zu jedem Knoten führen drei Kanten, die ihn mit den drei möglichen Vorgängerkorrespondenzen verbinden. Abbildung 5.1 zeigt einen Beispielgraphen. Formal ist die Kantenmenge wie folgt beschaffen

$$\begin{aligned} E := \quad & \{\,[(i,j),(i+1,j)] \mid 0 \leq i < n-1,\ 0 \leq j < n\} \cup && \text{horizontal} \\ & \{\,[(i,j),(i,j+1)] \mid 0 \leq i < n,\ 0 \leq j < n-1\} \cup && \text{vertikal} \\ & \{\,[(i,j),(i+1,j+1)] \mid 0 \leq i < n-1,\ 0 \leq j < n-1\} && \text{diagonal} \end{aligned}$$

Die Kosten einer Kante $e = [(i,j),(k,l)] \in E$ ergeben sich aus (5.5) zu

$$c(e) := c((i,j),(k,l)) = \begin{cases} \frac{1}{n-1} \cdot [\frac{1}{2}(d_{i,j} + d_{k,l}) + \alpha] & \text{falls } e \text{ horiz. oder vert.} \\ \frac{1}{n-1} \cdot \frac{1}{\sqrt{2}}(d_{i,j} + d_{k,l}) & \text{falls } e \text{ diagonal} \end{cases}$$

Jeder Pfad von Knoten $(0,0)$ bis Knoten $(n-1, n-1)$ entspricht einer gültigen Korrespondenzabbildung von $C_1$ auf $C_2$, wenn man die Knotenfolge als diskrete Abbildung interpretiert. Die Kosten des Pfades sind per Konstruktion gleich den Kosten des diskretisierten Energiefunktionals. Umgekehrt ist jede gültige, diskrete Korrespondenzfunktion im Graphen als kürzester Weg enthalten und besitzt dieselben Kosten. Der kürzeste Weg wird über dynamische Programmierung berechnet; siehe dazu Abschnitt 4.2. Die Laufzeit für die Berechnung des globalen Minimums der Funktion $E_d$ beträgt daher $O(n^2)$.

### 5.2.4 Vergleich geschlossener Konturen

Bisher wurde vom Vergleich offener Kurven ausgegangen, bei dem Anfangs- und Endkorrespondenz als Randbedingungen fest eingingen. Für geschlossene Kurven

muss der Begriff der gültigen Korrespondenzfunktion angepasst werden. Da wir geschlossene Konturen betrachten, werden die Krümmungsfunktionen $\kappa_{1,2}$ auf dem Einheitskreis $\mathbb{S}^1$ parametrisiert. Gesucht wird dann eine Korrespondenzfunktion

$$\vec{m} : \mathbb{S}^1 \rightarrow \mathbb{S}^1 \times \mathbb{S}^1$$

Dabei sei in der kontinuierlichen Formulierung angenommen, dass $\vec{m}$ ein orientierungserhaltender Diffeomorphismus ist. Daraus folgt automatisch die Geschlossenheit, d.h. Anfangskorrespondenz muss gleich Endkorrespondenz sein:

$$\vec{m}(0) = \vec{m}(2\pi)$$

Für den kürzesten Weg im Graphen bedeutet dies, dass Startknoten gleich Endknoten sein muss. Die Optimierungssituation beim Matching geschlossener Konturen wurde bereits in Abschnitt 4.4 besprochen. Übertragen auf das hier besprochene Matchingmodell sind folgende Erweiterungen des Algorithmus vorzunehmen:

**Graphenerweiterung** Für jedes $i \in \{0, ..., n-1\}$ wird ein Graph $G(i)$ definiert. $G(i)$ ist strukturell wie der oben beschriebene Graph $G$ beschaffen mit folgendem Unterschied:

- Die Spalten stehen für die Punkte $i, ..., n-1, 0, ..., i$ der Kontur $C_1$.
- Die Zeilen stehen für die Punkte $0, ..., n-1, 0$ der Kontur $C_2$

Die Graphen $G(i)$ für $0 \leq i < n$ sind also eine Spalte und eine Zeile größer als $G$ und die linke und rechte Spalte sowie obere und untere Zeile beschreiben dieselben Korrespondenzen. Die Graphen $G(i)$ sind *spaltenrotierte* Versionen voneinander.

**Optimierung** Um die optimale Korrespondenzabbildung zu finden muss auf jedem Graphen $G(i)$ der kürzeste Weg zwischen der linken unteren und der rechten oberen Ecke - d.h. zwischen den beiden Korrespondenzen $(i, 0)$ - berechnet werden. Der kürzeste unter allen diesen Wegen ist die optimale diskrete Korrespondenzabbildung. Dies geht unmittelbar aus den Argumenten in Abschnitt 4.4 hervor.

Bei einer Abtastrate von $n$ Punkten ergibt sich - da der kürzeste Weg auf $n$ Graphen berechnet werden muss - eine Gesamtlaufzeit von $O(n^3)$ zur Berechnung der optimalen Korrespondenzabbildung.

## 5.3 Ergebnisse

Die Tests für das vorgestellte Verfahren sollen zum einen bestimmte Eigenschaften des Algorithmus nachweisen, zum anderen die Laufzeit messen. Des Weiteren soll

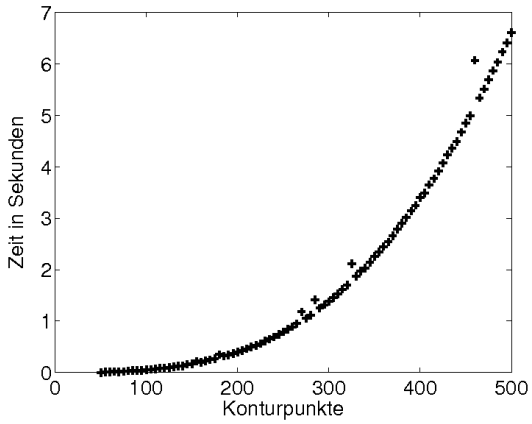



Abbildung 5.2: Laufzeit des Vergleichverfahrens über dynamische Programmierung.

das Verhalten des Distanzmaßes untersucht werden, insbesondere in Bezug auf die intuitiv empfundene Ähnlichkeit von Formen.

Die Eingabeshapes lagen in Form von schwarz-weiß Pixelbildern vor. Es wurden Bilder aus der LEMS-Datenbank der Brown-University verwendet. Die Formkonturen wurden berechnet und in regelmäßigen Abständen abgetastet. In den nachfolgenden Beispielen wurden sie jeweils mit 300 Punkten abgetastet. An den Punkten wurde die Integralinvariante gemäß Kapitel 2 berechnet und die Krümmung wie beschrieben über den gemessenen Integralwert approximiert. Die Vergleichsergebnisse sind durch korrespondierende Nummern gekennzeichnet, wobei Punkte mit sehr hoher bzw. niedriger Krümmung im Uhrzeigersinn entlang der Kontur ausgewählt wurden.

### 5.3.1 Eigenschaften

#### Rotation und Skalierung

In Bild 5.3 ist die Rotations- und Skalierungsinvarianz gezeigt ($\alpha = 0$). Durch das kreisförmige Integrationsgebiet der Integralinvariante aus Kapitel 2 bleibt die approximierte Krümmung auch nach der Rotation der Hand erhalten. Bemerkenswert ist allerdings, dass beim Vergleich die Hand rechts zwar um 100% vergrößert wurde (in der Abbildung ist sie verkleinert dargestellt), jedoch der Integrationsradius der Integralinvariante beibehalten wurde. Bei einer Skalierung um 150% unter Beibehaltung des Radius versagte der Vergleich; die Skalierungsninvarianz des Verfahrens ist daher - wie zu erwarten war - nur bedingt gegeben. Die Distanz der beiden Formen beträgt $0.2570$.

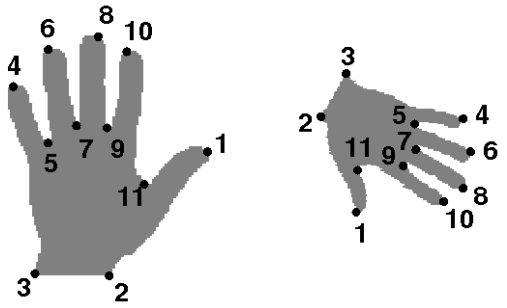



Abbildung 5.3: Rotation und Skalierung: Die Hand links im Vergleich mit der rotierten und um 100% vergrößerten Hand (im Bild verkleinert dargestellt)

### Artikulation von Shapeteilen

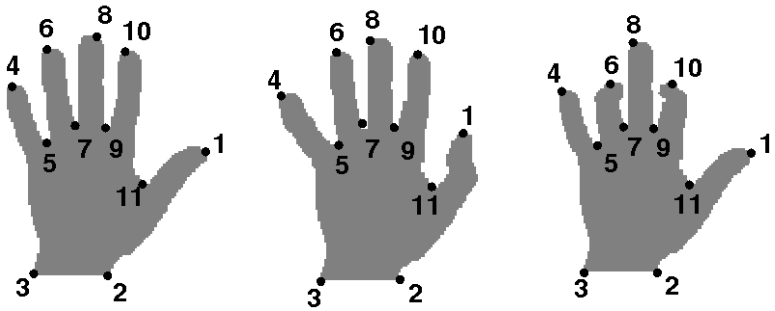



Abbildung 5.4: Artikulationstest: Die Musterform links wurde jeweils mit den zwei rechten Händen verglichen.

In Abbildung 5.4 sind zwei Beispiele einer Hand zu sehen, deren Finger auf verschiedene Weise abgeknickt wurden. Die Punkt-Korrespondenzen zwischen der linken Hand und den anderen beiden Formen wurden korrekt gefunden. Im zweiten Paar Hände sind Punkt 6 und 10 leicht verrutscht, was sich auf einen Messfehler der Integralinvariante zurückführen lässt, der durch die starke Nähe der Fingerkuppen zum Mittelfinger zustande kommt (vgl. Abschnitt 2.2.1). Die Distanzen der artikulierten Hände zur linken Hand betragen $0.0330$ und $0.0282$ ($\alpha = 0$).

### Verdeckung

Bei der Verdeckung wurden zwei Fälle untersucht: Es fehlen Teile der Kontur (Abb. 5.5) oder die Kontur enthält Ausbuchtungen, die nicht zum Objekt gehören (Abb. 5.6). In Abbildung 5.5 wird eine vollständige Hand im ersten Fall mit einer Hand mit fehlendem Mittelfinger und im zweiten Fall mit Fingern der halben Länge verglichen. Im ersten Fall wird der Mittelfinger (Punkt 7) dank des Glattheitsterms in

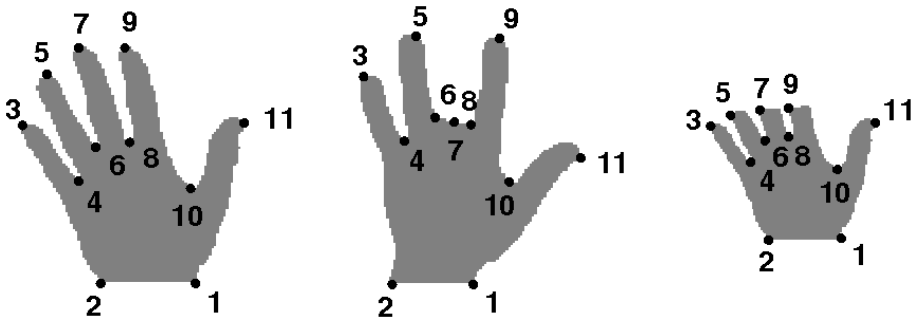



Abbildung 5.5: Verdeckungstest 1: Die linke Musterform wurde jeweils mit zwei Formen rechts daneben gematcht, bei denen Formteile fehlen.

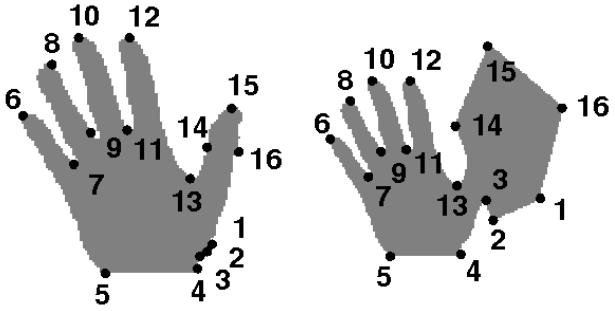



Abbildung 5.6: Verdeckungstest 2: Vergleich der linken Hand mit einer entstellten Hand rechts.

der Mitte der entstehenden Lücke vermutet ($\alpha = 0.01$). Im zweiten Fall werden folgerichtig die durch das Abschneiden entstehenden spitzen Ecken als Fingerkuppen gewertet, da sie lokale Maxima der Krümmungsfunktionen repräsentieren.
In Abbildung 5.6 wird eine Hand mit einer Mischform aus einer Hand und einem Fünfeck verglichen. Die Ecken stellen Punkte mit hoher Krümmung dar und erschweren das Finden der Korrespondenzen. Mit $\alpha = 0.01$ findet der Algorithmus die Korrespondenzen der Hand und verteilt die Punkte des Fünfecks entlang des Daumens.

### 5.3.2 Laufzeit

Der Laufzeittest ist in Abbildung 5.2 zu sehen. Es wurden zwei Formen verglichen, die mit 50-500 Punkten abgetastet wurden. Die Tests wurden mit einem Pentium IV Prozessor mit 3.4 GHz durchgeführt. Die kubische Laufzeit ist klar erkennbar und reicht von 0.001 Sekunden bei 50 Punkten bis 6.61 Sekunden bei 500. Die Laufzeit ist lediglich von der Punktzahl abhängig, nicht jedoch von den verglichenen Formen. In jedem Fall werden $n^3$ Berechnungschritte durchgeführt. Die Zeit für die Berechnung der Invarianten wurde nicht mit gemessen.

# Kapitel 6

# Shape Matching mittels Graph Cuts

Das in Kapitel 5 eingeführte diskrete Energiefunktional wurde durch die Berechnung von $n$ kürzesten Wegen - einer für jede mögliche Initialkorrespondenz - minimiert. Dies führte letztlich zu einer kubischen Laufzeit. Gesucht wird hingegen lediglich *ein* kürzester Weg, die restlichen Wege werden gar nicht gebraucht. Ziel dieses Abschnittes ist es daher, die Suche nach der optimalen Initalkorrespondenz zwischen den Konturen *implizit* zu lösen. Dies wird durch einen geeigneten Graph Cut-Ansatz ermöglicht.
Zu diesem Zweck werden in Abschnitt 6.1 die benötigten Grundlagen zu Graph Cuts besprochen. Abschnitt 6.2 beschäftigt sich mit der Konstruktion eines geeigneten Graphen. In Abschnitt 6.3 wird der Graph Cut Ansatz formalisiert, um die Äquivalenz des Verfahrens zu dem über dynamische Programmierung zu zeigen.

## 6.1 Graph Cuts

Graph Cuts gehören dem Bereich der kombinatorischen Optimierung an. Durch sie ist es möglich, bestimmte Energiefunktionale global und in polynomieller Zeit zu optimieren.

### 6.1.1 Graphen

Ein *Graph* $G$ besteht aus einer endlichen *Knotenmenge* $V$ und einer endlichen *Kantenmenge* $E$, kurz $G = (V, E)$. Jedes Element aus $E$ ist ein Zweiertupel $(v, w)$ mit $v, w \in V$. Sind die Knoten der Tupel geordnet - d.h. $(v, w) \neq (w, v)$ - so ist der Graph *gerichtet*, andernfalls *ungerichtet*. In beiden Fällen kann jeder Kante ein positives *Gewicht* zugeordnet werden. Dies führt zu einer Funktion $w : E \to \mathbb{R}^+$. Ein gewichteter Graph wird auch als Dreiertupel $G = (V, E, w)$ notiert. Jeder Graph

besitzt eine graphische Darstellung, in der jeder Knoten als Punkt und jede Kante $(v, w)$ als Linienverbindungen zwischen den Punkten $v$ und $w$ repräsentiert wird. Ist der Graph gerichtet, so wird die Linienverbindung durch einen Pfeil in die entsprechende Richtung ersetzt.

Ein Graph wird als *planar* bezeichnet, falls er sich derart in der Ebene oder auf einer Kugeloberfläche darstellen lässt, dass sich keine zwei Kanten kreuzen. Identifiziert man die Graphenknoten mit Punkten im $\mathbb{R}^2$ und jede Kante $e \in E$ mit der Kurve $c_e : [0, 1] \to \mathbb{R}^2$, so ist *planar* gleichzusetzen mit der Bedingung $c_{e_1}(s) \neq c_{e_2}(t)$ für alle $e_1, e_2 \in E$ und alle $s, t \in [0, 1]$. Ein planarer Graph teilt die Ebene, auf der er eingebettet ist, in Flächen auf, die durch einfache Kreispfade begrenzt werden. Lediglich eine dieser Flächen - die unendliche Fläche $f_\infty$ - ist nach außen hin unbegrenzt. Bei einer Einbettung auf der Kugeloberfläche, gibt es solch eine Fläche nicht. Stattdessen lässt sich eine *beliebige* Fläche als $f_\infty$ auszeichnen. Die Flächen $F$ sind für einen gegebenen planaren Graphen - bis auf die Reihenfolge paralleler Kanten - eindeutig bestimmt und man kann einen planaren Graphen $G$ schreiben als $G = (V, E, F, w)$. Die Fläche links von einer Kante $e$ in Kantenrichtung wird als $f_l(e)$ bezeichnet. Analog schreiben wir $f_r(e)$ für die rechte Fläche.

Zu einem planaren Graph $G = (V, E, F)$ lässt sich ein *dualer* Graph $G^* = (V^*, E^*, F^*)$ definieren, indem $V^* = F$, $F^* = V$ und $E^* = \{(f_l(e), f_r(e)) \mid e \in E\}$ gesetzt wird. Anschaulich entsteht der duale Graph also, indem Flächen in $G$ als Knoten in $G^*$ interpretiert werden. Zwei Knoten in $G^*$ werden genau dann über eine Kante verbunden, wenn die korrespondierenden Flächen in $G$ benachbart sind. Die Kante in $G^*$ kann mit der Kante in $G$ zwischen den benachbarten Flächen identifiziert werden, woraus $|E| = |E^*|$ folgt. Im gerichteten Fall entspricht die Dualisierung einer Drehung der Kante um 90° im Uhrzeigersinn.

### 6.1.2 Maximaler Fluss in einem Netzwerk

Ein Flussnetzwerk $G = (V, E, c, s, t)$ ist ein gerichteter Graph mit Knotenmenge $V$, Kantenmenge $E$ und einer *Kapazitätsfunktion* $c : E \to \mathbb{R}^+$, die jeder Kante eine Kapazität zuordnet. $s$ und $t$ sind zwei ausgezeichnete Knoten $\in V$, die *Quelle* und die *Senke*. Von der Senke gehen keine Kanten aus und zur Quelle führen keine hin. Ein *Fluss* ist eine Funktion $f : E \to \mathbb{R}^+$ mit folgenden Eigenschaften:

$$\forall v \in V \setminus \{s, t\} \text{ gilt } \sum_{(u,v)\in E} f((u, v)) - \sum_{(v,w)\in E} f((v, w)) = 0 \tag{6.1}$$

$$\forall e \in E \text{ gilt } \quad f(e) \leq c(e) \tag{6.2}$$

Bedingung (6.1) besagt, dass der Fluss, der den Knoten über eingehende Kanten betritt, in der Summe gleich dem ausströmenden Fluss über ausgehende Kanten ist.

Außerdem kann über eine Kante $e$ nicht mehr Fluss gehen als ihre Kapazität zulässt - das besagt (6.2). Der *maximale Fluss* durch das Netzwerk G ist diejenige Flussfunktion $f_{max}$, für die der Fluss zur Senke $c(f) = \sum_{\substack{(v,t)\in E \\ v\in V}} f((v,t))$ maximal ist unter allen Flussfunktionen $f$.

Sei $U \subset V$. Ein *Cut* $C_U$ besteht aus allen Kanten, die aus der Menge $U$ herausführen zu Knoten aus $V \setminus U$. Formal gilt:

$$C_U := \{(u,v) \in E \mid u \in U \text{ und } v \in V \setminus U\}$$

Die Kapazität eines Cuts ist die Summe der Kapazitäten der in $C_U$ enthaltenen Kanten, kurz $c(C_U) = \sum_{e\in C_U} c(e)$. Ein Cut heißt *(s,t)-separierend*, falls $s \in U$ und $t \notin U$. Da der maximale Fluss von der Quelle in die Senke von oben durch die Kapazität eines *beliebigen* ($s$,$t$)-separierenden Cuts begrenzt sein muss, ist ein Fluss $f$ genau dann maximal, falls ein Cut $C$ existiert, so dass $c(f) = c(C)$. Das **Theorem von Ford und Fulkerson** [11] besagt daher, dass der maximale Fluss in einem Netzwerk gleich der minimalen Kapazität eines (s-t)-separierenden Cuts ist.

### 6.1.3 Graph Cuts in der Bildverarbeitung

Viele Aufgaben im Bereich der Bildverarbeitung können als *Multilabeling-Probleme* formuliert werden. Jedem Pixel $p$ aus der Pixelmenge $P$ eines Bildes soll ein Label $f_p$ aus der endlichen Menge $L$ zugewiesen werden. Die Werte aus $L$ können je nach Problemstellung *Farben* bei der Bildrestaurierung, *Disparitäten* bei der Stereovision oder sogar *Ebenen* bei der 3D-Rekonstruktion kodieren, um nur einige Beispiele zu nennen. Die Energiefunktionale haben oft die Form

$$E(f) = \sum_{p\in P} D_p(f_p) + \sum_{(p,q)\in N} V_{p,q}(f_p, f_q) \tag{6.3}$$

Die erste Summe bewertet, wie wahrscheinlich für ein Pixel $p$ das Label $f_p$ ist. Die zweite Summe misst die Unterschiede der Labels benachbarter Pixel. $N$ bezeichnet die Nachbarschaftsbeziehungen der Pixel und kann verschieden gewählt werden.
Für binäre Bildrestaurierung setzten Greig et. al. 1989 erstmals Graph Cuts ein [15]. Sie konstruierten einen speziellen Graphen, in dem jeder Knoten einem Pixel entsprach und berechneten den minimalen Cut. Die Zugehörigkeit jedes Knotens zu entweder der Quelle oder der Senke bestimmten deren Farbe (schwarz oder weiss). Das Ergebnis ist global optimal und exakt. Auf diese Weise wurden erstmals laufzeitintensive und ungenaue Methoden zur globalen Energieminimierung, wie beispielsweise das *Simulated Annealing*, umgangen. Eine Verallgemeinerung auf mehrere Labels in polynomieller Zeit ist hingegen nicht möglich, was nicht zuletzt der

Grund dafür war, dass die Arbeit weitestgehend unbeachtet blieb. Erst ab 2001 trugen unter anderem Boykov und Kolmogorov dazu bei, das Potential der Graph Cuts für die Bildverarbeitung weiter zu erschließen.
In der 2001 publizierten Arbeit [5] zeigen Boykov, Veksler und Zabih, dass selbst einfachste *nicht-binäre*, diskontinuitätserhaltende Multilabel-Probleme der Form (6.3) NP-hart sind (das *Potts*-Modell). Zugleich entwickeln sie jedoch ein polynomielles, mehrschrittiges Graph Cut Verfahren, das für bestimmte Probleme der Form (6.3) in einem strengen Sinn das global optimale Labeling approximiert. Kern der Idee ist, das Multilabelproblem in jedem Schritt auf ein binäres Labelproblem zu reduzieren ($\alpha$-*Expansion,* $\alpha$-$\beta$-*Swap*), das seinerseits effizient über Graph Cuts zu lösen ist.
Die 2004 veröffentlichte Arbeit von Kolmogorov und Zabih [19] gab weiteren Aufschluss darüber, welchen Bedingungen ein Funktional einer Klasse binärer Energiefunktionale genügen muss, damit es über Graph Cuts optimierbar ist (*Submodularität*). Erstmals wurden außerdem Graphenkonstruktionen vorgestellt, mit denen derartige Funktionale unabhängig von der speziellen Problemstellung minimiert werden können.

Der in diesem Kapitel vorgestellte Ansatz unterscheidet sich von typischen Graph Cut Problemen in der Bildverarbeitung: Die Knoten repräsentieren keine Pixel und eine Energie der Form (6.3) ist nicht gegeben. Stattdessen entsteht der Graph aus einer bereits konstruierten Kürzesten-Wege-Formulierung, bei der die Positivität der Kantegewichte gewährleistet ist und von der bereits gezeigt wurde, dass sie das zugrundeliegende Funktional optimiert. Dies wird im Folgenden ausgenutzt, wodurch sich die Frage nach Submodularität einer binären Formulierung gar nicht erst stellt. In Kapitel 7 wird die Andersartigkeit von herkömmlichen Bildverabeitungsproblemen Konsequenzen für die Wahl des verwendeten Max-Flow-Algorithmus haben.

## 6.2 Graphenkonstruktion

In diesem Abschnitt wird der Graph konstruiert, dessen minimaler Cut eine Lösung des diskreten Vergleichsmodells aus Kapitel 5 liefert. Zunächst wird der dort konstruierte Graph zu einem äquivalenten Graph transformiert, dann eine intuitive Knotenbenennung eingeführt und schließlich wird der resultierende Graph zum Flussnetzwerk gewandelt.

### 6.2.1 Erste Vorstufe: Graphentransformation

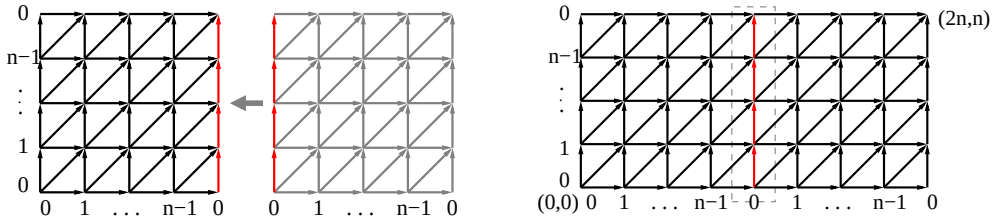
Zunächst soll erreicht werden, dass die $n$ verschiedenen Graphen des Ansatzes in Kapitel 5 in *einem* Graphen vereint sind und gültige Korrespondenzabbildungen geschlossene Pfade auf diesem Graphen darstellen.

Ausgangspunkt sind die Graphen $G(i),\ 0 \leq i < n$ mit der Initialkorrespondenz $(i, 0)$ (siehe Abschnitt 5.2). Die Graphen bestehen aus einem rechteckigen $n \times n$-Knotengitter. Die Gitterspalten repräsentieren von links nach rechts die Knoten $i, ..., n-1, 0, ..., i$ auf der ersten Kontur $C_1$, die Zeilen von unten nach oben die Punkte $0, ..., n-1, 0$ der zweiten Kontur $C_2$.

Eine gültige Korrespondenzfunktion mit Initialkorrespondenz $(i, 0)$ entspricht einem Pfad vom linken unteren zum rechten oberen Knoten in $G(i)$. Ein solcher Pfad soll im Weiteren als **gültiger Korrespondenzweg mit Initialkorrespondenz** $(\mathbf{i}, \mathbf{0})$ bezeichnet werden. Analoges gilt für den *kürzesten* Pfad zwischen den beiden Knoten, der analog als **kürzester Korrespondenzweg** bezeichnet wird. Derjenige unter den $n$ kürzesten Korrespondenzwegen mit den kleinsten Kosten entspricht der optimalen Korrespondenzabbildung zwischen den diskretisierten Kurven [1]. Die Graphen $G(i),\ 0 \leq i < n$ werden nun durch folgende Schritte ersetzt:

### Schritt 1

Der Graph $G(0)$ wird zweimal hintereinander *geklebt* und zwar derart, dass die letzte Spalte des ersten Graphen zur Deckung mit der ersten Spalte des zweiten Graphen kommt (s.u.). Der resultierende **Graph** $\tilde{\mathbf{G}}$ besteht aus einem $(2n + 1) \times (n + 1)$ Knotengitter.

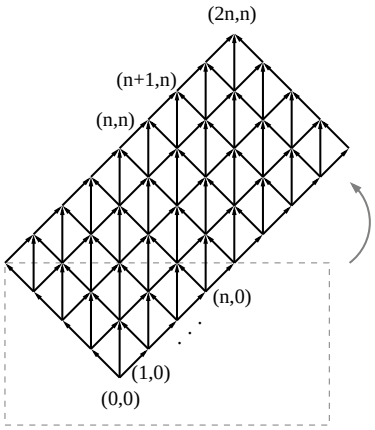



Die Knotenbenennung wird wie folgt gewählt: $(i, j)$ bezeichnet den Knoten der $i$-ten Spalte von links und der $j$-ten Zeile von unten. Die zugehörige Korrespondenz ist $\mathrm{cor}((i, j)) := (i \bmod n, j \bmod n)$.

[1] Es kann durchaus mehrere solcher kürzesten Wege geben, eine optimale Matchingfunktion ist also nicht eindeutig definiert.

### Schritt 2

Der Graph $\tilde{G}$ wird um den linken oberen Knoten um 45° im Gegenuhrzeigersinn gedreht, so dass der Knoten $(i, 0)$ auf einer vertikalen Linie mit dem Knoten $(i+n, n)$ steht für alle $0 \leq i < n$. Es gilt $\mathrm{cor}((i, 0)) = \mathrm{cor}((i + n, n))$.




### Schritt 3

Nun wird der Graph $G$ zu einem Zylinder gebogen, indem der Knoten $(i, 0)$ mit $(i + n, n)$ identifiziert wird für alle $0 \leq i < n$ (siehe Abbildung). Der resultierende **Graph G** enthält also $n$ Knoten weniger als $G$.

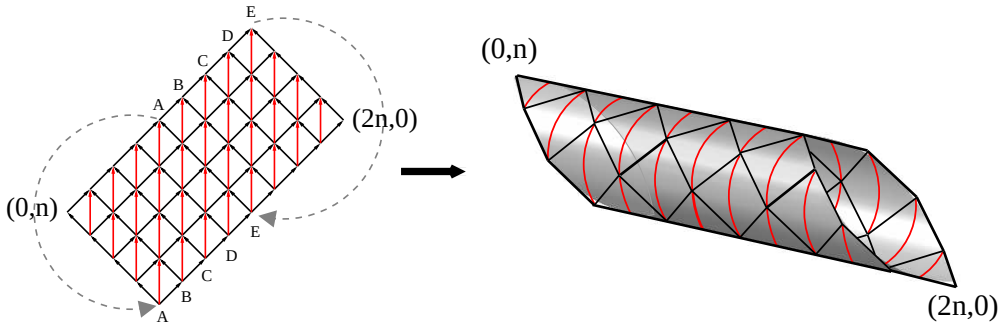



Abbildung 6.1: Der Graph $\tilde{G}$ (*links*) wird zu einem Zylinder gebogen (*rechts*), indem die Knoten $(i, 0)$ mit den Knoten $(i + n, n)$ identifiziert werden. Diagonale Kanten in $\tilde{G}$ (rot) werden zu vertikalen Kanten in $G$.

### Erläuterung

In Schritt 1 werden die $n$ Graphen $G(i)$ durch einen großen Graphen $\tilde{G}$ ersetzt. Das sieht man wie folgt: Statt auf dem Graphen $G(i)$ für ein $i$ mit $0 \leq i < n$, den kürzesten Weg von links unten nach rechts oben zu berechnen, kann auf dem Graphen $\tilde{G}$ der kürzeste Weg zwischen Knoten $(i, 0)$ und $(i + n, n)$ berechnet werden. Da jeder Graph $G(i)$ eine Untermenge des Graphen $\tilde{G}$ darstellt, muss der so berechnete kürzeste Weg derselbe sein, wie der im Graphen $G(i)$. Die Kosten der beiden Wege sind identisch. Somit kann die Berechnung der Korrespondenzabbildung zweier geschlossener Konturen auf einem *einzelnen* Graphen $G$ erfolgen.

Anfangs- und Endknoten eines kürzesten Korrespondenzweges in $G$ repräsentieren identische Korrespondenzen zwischen zwei Kurvenpunkten. Dies entspricht der Tatsache, dass geschlossene Konturen aufeinander abgebildet werden, d.h. die Anfangskorrespondenz ist gleich der Endkorrespondenz. Dieser Sachverhalt soll sich nun auch im Graphen selbst widerspiegeln, sprich **gültige Korrespondenzwege sollen geschlossen sein in $\tilde{\mathbf{G}}$**. Daher werden in Schritt 2 und 3 die Anfangs- und Endkorrespondenzen miteinander identifiziert. Schritt 2 hat dabei lediglich die Funktion, die Anschaulichkeit zu wahren. Er hat für das Ergebnis keinerlei Bedeutung.

### Eigenschaften

Der resultierende **Graph G** besitzt folgende Eigenschaften:

1. Der Graph ist immer noch planar und kann auf einer Kugeloberfläche eingebettet werden.

2. Die diagonalen Kanten von $\tilde{G}$ sind die vertikalen Kanten in $G$.

3. Jeder *gültige* und insbesondere *kürzeste* Korrespondenzweg in $\tilde{G}$ ist in $G$ ein *einfacher*, *geschlosser* Weg, der den Zylinder einmal umfasst.

4. Jeder *einfache*, *geschlossene* Weg in $G$ beschreibt eine gültige Korrespondenzabbildung.

**Eigenschaften 3 und 4 implizieren eine eins-zu-eins Beziehung zwischen gültigen Korrespondenzabbildungen und einfachen geschlossenen Wegen**. Die Korrektheit von 3 und 4 sieht man wie folgt: Die Endknoten jedes gültigen Korrespondenzweges wurden miteinander identifiziert; also ist klar, dass er geschlossen in $G$ sein muss. Außerdem ist jeder Korrespondenzweg in $\tilde{G}$ einfach, also ist er es auch in $G$.
Die Umkehrung gilt aus folgenden Gründen: Geschlossenheit eines Weges auf dem Zylinder $G$ impliziert, dass Anfangs- und Endknoten gleich sind. Also gilt das gleiche für die Korrespondenzen der zum Weg gehörigen Abbildung. Für die Einfachheit gilt folgendes: Generell gibt es auf einem Zylinder keinen geschlossenen Weg, der den Zylinder zweimal umwandert und sich dabei nicht selbst schneidet. Also muss ein einfacher geschlossener Weg den Zylinder genau einmal umwandern. Speziell in $G$ sind jedoch alle Wege, die den Zylinder genau einmal umlaufen notwendig einfach, da sämtliche Kanten auf dem Zylinder in eine Umlaufrichtung zeigen (siehe Abb. 6.1, links). Aus der Graphenstruktur von $\tilde{G}$ geht schließlich hervor, dass ein solcher Weg nur dann geschlossen sein kann, wenn er eine gültige Korrespondenzabbildung beschreibt.

Um die Übersicht auf dem komplexen Ergebnisgraphen $G$ zu behalten, ist eine andere Knotenbenennung sinnvoll. Dafür wird im nächsten Abschnitt eine alternative Repräsentation der Korrespondenzabbildung $m(s)$ aus Abschnitt 5.1.1 eingeführt.

### 6.2.2 Zweite Vorstufe: Knotenbenennung

#### Disparitätsfunktion

Die Korrespondenzfunktion $m$ für geschlossene Konturen ist eine Funktion $m : \mathbb{S}^1 \to \mathbb{S}^1$, die den Punkt $C_1(s)$ der ersten Kontur auf den Punkt $C_2(m(s))$ der zweiten abbildet.

Statt dieser absoluten kann nun auch eine äquivalente *relative* Darstellungsform der Korrespondenzabbildung gewählt werden. Die **Disparitätsfunktion**

$$d : \mathbb{S}^1 \to \mathbb{R},\ s \mapsto d(s) \tag{6.4}$$

ist implizit definiert über

$$m(s) = s - d(s)$$

Sie beschreibt also die Verschiebung des Parameters der Kurve $C_2$ relativ zu dem von $C_1$. Da geschlossene Kurven betrachtet werden, gilt

$$s - d(s) = m(s) = m(s) + n2\pi = s - d(s) + n2\pi$$

Die Disparitäten $d(s)$ und $d(s) + n2\pi,\ \forall n \in \mathbb{Z}\ s \in \mathbb{S}^1$ können also miteinander identifiziert werden. Ferner lässt sich jede Korrespondenzfunktion als Disparitätsfunktion darstellen. Die Umkehrung gilt ebenso. Daher heißt eine Funktion $d(s)$ genau dann eine *gültige Disparitätsfunktion*, wenn $s - d(s)$ eine gültige Korrespondenzabbildung darstellt.

Die *diskrete Version* der Disparitätsfunktion bei $n$ Diskretisierungspunkten auf beiden geschlossenen Kurven ist

$$d : [0..n-1] \to \mathbb{Z} \tag{6.5}$$

mit

$$m(i) = i - d(i),\ 0 \leq i < n$$

Analog zum kontinuierlichen Fall definieren wir $d(i) + k \cdot n \equiv d(i), \forall k \in \mathbb{Z}$.

#### Knotenbezeichnung in G

Der Zylinder $G$ besteht aus Knotenringen (siehe Abb. 6.1). **Sämtliche Knoten eines Ringes besitzen die gleiche Disparität**. Längs des Zylinders nimmt die Disparität der Ringe nach links hin ab, während sie für Ringe weiter rechts steigt.

| $\tilde{\mathbf{G}}$ | $\mathbf{G}$ |
|---|---|
| diagonal, $[(i,j),(i+1,j+1)]$ | vertikal, $[(d,i),(d,i+1)]$ |
| horizontal, $[(i,j),(i+1,j)]$ | diagonal, $[(d,i),(d+1,i+1)]$ |
| vertikal, $[(i,j),(i+1,j+1)]$ | diagonal, $[(d,i),(d-1,i)]$ |

Tabelle 6.1: Korrespondierende Kanten in $G$ und $G'$

Daher bietet sich die Disparität als intuitive Knotenmarkierung an. Der Knoten $(i,j)$ aus $G$, wird zu

$$(d,i) := (i-j,i),$$

wobei $d$ die Disparität bezeichnet. Die äußeren beiden (vollständigen) Knotenringe auf dem Zylinder besitzen die Disparität $0$ und entsprechen der Identitätsabbildung $m(i) = i$ zwischen den Kurvenpunkten.

Die Kanten des Graphen $\tilde{G}$ lassen sich folgendermaßen auf die Kanten in $G$ übersetzen. Eine diagonale Kante $[(i,j),(i+1,j+1)] \in E_{\tilde{G}}$ wird in $G$ durch die Drehung zu einer vertikalen Kante zwischen den Knoten $[(d,i),(d,i+1)]$. Eine horizontale $[(i,j),(i+1,j)]$ bzw. vertikale Kante $[(i,j),(i,j+1)]$ wird in $G$ zu $[(d,i),(d+1,i+1)]$ bzw. $[(d,i),(d-1,i)]$, also zu diagonalen Kanten (vgl. Abbildung 6.1 und Tabelle 6.1). Um die drei Typen von Kanten eindeutig zu bezeichnen, sollen sie auch in $G$ *horizontal*, *vertikal* und *diagonal* heißen.

Ein Pfad im Graphen $G$ ist ein *gültiger Disparitätsweg*, falls der Weg eine gültige Disparitätsfunktion repräsentiert.

### 6.2.3 Formulierung als Graph Cut-Problem

Bis jetzt ist folgende Situation erreicht:

1. Das Korrespondenzproblem wurde auf die Berechnung kürzester Wege auf *einem* planaren Graphen reduziert.

2. Jede gültige Disparitätsfunktion/gültige Korrespondenzfunktion ist in dem Graphen als geschlossener Weg enthalten, der den Zylinder einmal umwandert.

Gerade die zweite Eigenschaft ist die ideale Voraussetzung für einen Graph Cut-Ansatz. Denn wird $G$ auf Kugeloberfläche eingebettet, so dass die Zylinderringe des Graphen parallel zum Äquator liegen, so trennt aufgrund der Eigenschaft 2) jeder gültige Disparitätsweg die beiden Polflächen voneinander. Im Folgenden seien diese Polfächen mit $f_s$ und $f_t$ bezeichnet.

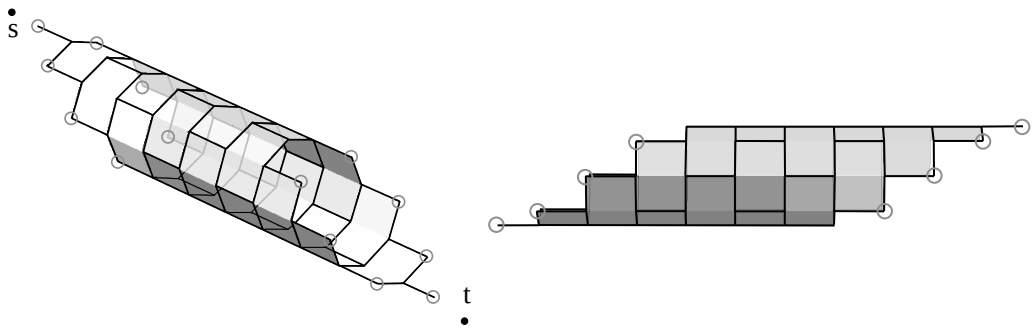



Abbildung 6.2: Der duale Graph $G^*$, der für die Graph Cut-Formulierung Verwendung findet. Die eingekreisten Knoten sind mit der Quelle bzw. Senke verbunden.

### Dualer Graph $G^*$

Betrachtet man statt $G$ den dualen Graphen $G^*$, **so beschreiben die Kanten eines gültigen Korrespondenzweges auf $\tilde{G}$ in G einen *Cut***. Dies sieht man leicht: Jeder Knoten auf einem geschlossenen Pfad besitzt genau eine Eingangs- und eine Ausgangskante. Im dualen Graphen werden die Knoten zu Flächen, die nach Definition des Dualen einen *Ring* bilden. Die Kanten, über die die Flächen sich berühren, zeigen alle in eine Richtung, da auch die Kanten im primalen Graphen in eine Richtung zeigen.
Die Flächen $f_s$ und $f_t$ von $G$ sind im dualen Graphen zwei Knoten, die dort als $s$ und $t$ bezeichnet werden. Diese beiden Knoten werden in der Graph Cut-Formulierung die Funktionen der Quelle und der Senke des Flussnetzwerkes einnehmen. In $G^*$ sind diese beiden Knoten mit dem äußeren linken respektive rechten Rand des Zylinders mittels Kanten verbunden (siehe Abb. 6.2).

Strukturell entspricht diese Graphenkonstruktion $G^*$ nun bereits der gewünschten Graph Cut-Formulierung. Lediglich die Kapazitäten der Kanten müssen noch angepasst werden, damit von einer sinnvollen Dualität von Cuts und Korrespondenzabbildungen gesprochen werden kann. Für jede Kante wird die Kapazität in Vorwärtsrichtung aus dem primalen Graphen übernommen. Dies ist sinnvoll, da die Korrespondenzpfade im primalen Graphen im dualen durch Cuts repräsentiert werden, deren Kosten identisch zu denen der Pfade sein sollen. Allerdings können in der Graph Cut-Formulierung Kanten auch rückwärts geschnitten werden. Die Kosten eines Rückwärtsschnittes betragen $0$. Dadurch können ungültige Korrespondenzabbildungen entstehen, bei denen die Monotonie-Bedingung verletzt ist (siehe Abb. 6.3). Solche Rückwärtsschnitte können jedoch verhindert werden, **indem für jede Kante die Rückwärtskapazität [2] auf $\infty$ gesetzt wird**.

[2]Zur Unterscheidung zwischen Vorwärts- und Rückwärtskapazität einer gerichteten Kante im Flussnetzwerk siehe Kapitel 7. Alternativ kann man eine parallele Kante in Gegenrichtung mit der gewünschten Kapazität in den Graphen einfügen.

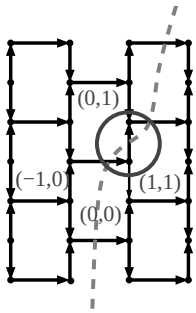



Abbildung 6.3: Ausschnitt des dualen Graphen $G^*$ mit einem Cut (gestrichelt). Die Flächen sind wie die Knoten in $G$ über die Disparität bezeichnet. Der Cut enthält einen zu verhindernden Rückwärtsschnitt (eingekreist), der in diesem Fall die ungültige Korrespondenzfolge $0-0, 1-1, 1-0, 2-1$ impliziert.

#### Interpretation des Cuts

Auf dem konstruierten Graphen $G^*$ kann über einen geeigneten Algorithmus der minimale Cut berechnet werden (siehe dazu Kapitel 7). Das Ergebnis ist eine Menge von Kanten. Jede Kante ist die duale Version einer Kante $[(d_1, i_1), (d_2, i_2)]$ im primalen Graphen, also einer Verbindung zwischen zwei Korrespondenzen. Daher wird festgelegt, dass die Korrespondenzen $(d_1, i_1)$ und $(d_2, i_2)$ Teil der Korrespondenzabbildung sind. Da die Kantenmenge eines Cuts im primalen Graphen einen geschlossenen Pfad bilden (siehe oben), ist die Anfangs- gleich der Endkorrespondenz. Der genaue Korrektheitsbeweis folgt im nächsten Abschnitt.

## 6.3 Formalisierung des Graph Cut-Ansatzes

In diesem Abschnitt soll die hergeleitete Graph Cut-Formulierung des Shape Matching Problems formalisiert werden. Dies bildet die Grundlage für den nachfolgenden Korrektheitsbeweis.
Die formale Beschreibung des Graphen wird durch die unregelmäßigen Enden des Zylinders $G$ (und damit $G^*$) erschwert. Daher wird der Graph zunächst ergänzt. Die Berechnung auf dem größeren Graphen liefert jedoch dieselbe Korrespondenzfunktion und die gleichen Kosten, da der Lösungsraum lediglich um redundante Knoten und Kanten erweitert wird.

#### Graphenvergrößerung - der Graph H

Der primale Graph $G$ besteht aus vollständigen Knotenringen im Zentrum und aus unvollständigen Ringen an den Enden des Zylinders (siehe Abb. 6.1). Der Graph $G$ wird also zu einem Graphen $H$ vervollständigt, indem die unvollständigen Disparitätsringe komplettiert und weitere Ringe links und rechts am Zylinder angehängt

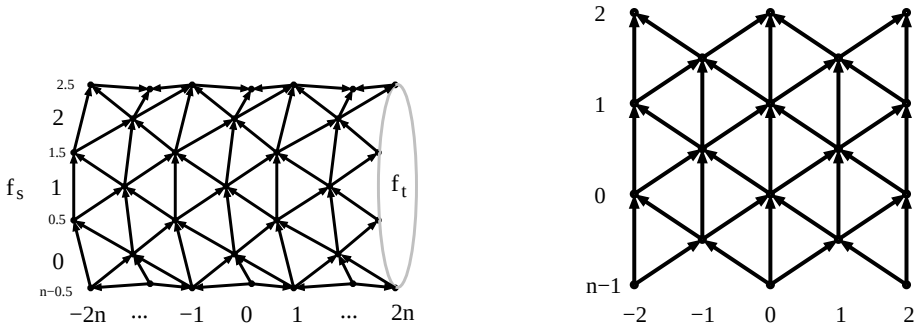



Abbildung 6.4: Der Graph $H$ als Ausgangspunkt für die Formalisierung ist eine Obermenge des Graphen $G$ und enthält daher ebenfalls alle Korrespondenzwege. *Links* ist der Zylinder, *rechts* ein Ausschnitt des Knotengitters zu sehen.

werden. $H$ besteht aus $4n + 1$ Disparitätsringen der Disparitäten $-2n$ bis $2n$ mit jeweils $n$ Knoten. [3] Knoten- und Kantenstruktur sind der des Graphen $G$ gleich: Die Ringe sind so gegeneinander verschoben, dass ein Knoten des Ringes $i$ in den Zwischenraum jeweils zweier Knoten eines Nachbarringes fällt (siehe Abb. 6.4).
Jeder Knoten bezeichnet wieder eine Korrespondenz zwischen zwei diskretisierten Kurvenpunkten und *ein* Knoten im Disparitätsring $0$ wird mit der Korrespondenz $(d, i) = (0, 0)$ ausgezeichnet. Die Markierung der restlichen Knoten ergibt sich dann aus dem Kantenschema des Graphen $G$, wie es in Abschnitt 6.2.2 beschrieben wurde.
In Abbildung 6.5 ist der duale Graph $H^*$ zu sehen. Er wird im Folgenden formalisiert:

**Knoten**

Sei

$$D = [-2n - 0.5, -2n + 0.5, \ ... \ - 0.5, 0.5, \ ... \ , 2n - 0.5, 2n + 0.5]$$

das Intervall der Disparitäten und

$$I = [0, 0.5, \ ... \ , n - 0.5]$$

Dann ist die Knotenmenge des Graphen $H^*$ als das kartesische Produkt der beiden Mengen definiert:

$$V = \{(v_d, v_x) \mid (v_d, v_x) \in D \times I\} \cup \{s, t\}$$

Dabei ist $I$ modulo $n$ definiert, d.h. für $i \in I$ gilt $i + n \equiv i$.

[3] Der Grund für die Wahl der spezifischen Disparitätsgrenzen ist, dass die Knoten von $H$ eine Obermenge der Knoten von $G$ bilden sollen.

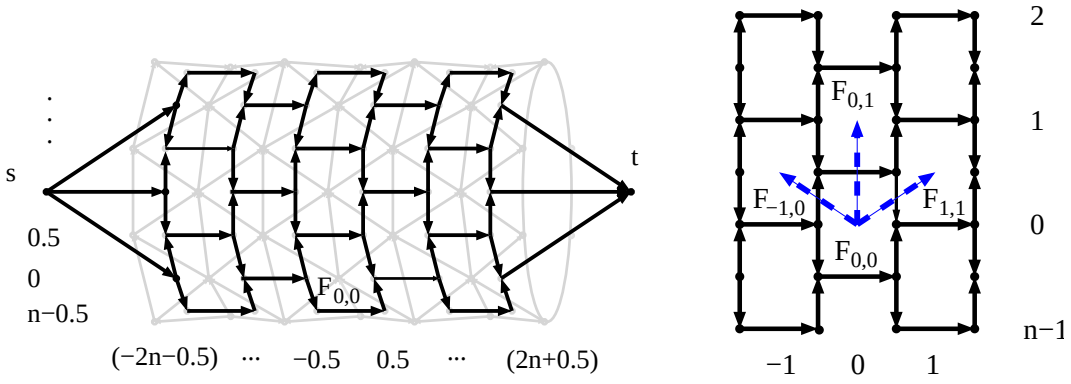



Abbildung 6.5: Der duale Graph $H^*$ dient als Beweisgegenstand der Korrektheit des Graph Cut Ansatzes. *Links* ist der Zylinder mitsamt Verbindungen zur Quelle und Senke angedeutet, *Rechts* ein Ausschnitt der Graphenstruktur mit den möglichen Wegen aus einer Fläche heraus.

### Kanten

Die Typenbezeichnungen *horizontal*, *vertikal* und *diagonal* der Kanten in $H$ (vgl. Graph $G$ in Abschnitt 6.2) werden im dualen Graphen $H^*$ direkt übernommen. Trägt man in jeder Kantengruppe der gegeneinander verschobenen Struktur von geraden und ungeraden Knotenringen Rechnung, ergeben sich folgende Kantenmengen (siehe auch Abb. 6.5):

$$\begin{aligned} E_d \quad = \quad & \{\ [a,b] := [(d,x) + (-0.5, 0.5)\ ,\ (d,x) + (0.5, 0.5)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}, \mathbb{Z}) \wedge [a,b] \in V \} \\ \cup \quad & \{\ [a,b] := [(d,x) + (-0.5, 0)\ ,\ (d,x) + (0.5, 0)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}+1, \mathbb{Z}) \wedge [a,b] \in V \} \end{aligned}$$

$$\begin{aligned} E_v \quad = \quad & \{\ [a,b] := [(d,x) + (-0.5, 0)\ ,\ (d,x) + (-0.5, 0.5)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}, \mathbb{Z}) \wedge [a,b] \in V \} \\ \cup \quad & \{\ [a,b] := [(d,x) + (-0.5, -0.5)\ ,\ (d,x) + (-0.5, 0)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}+1, \mathbb{Z}) \wedge [a,b] \in V \} \end{aligned}$$

$$\begin{aligned} E_h \quad = \quad & \{\ [a,b] := [(d,x) + (0.5, 0.5)\ ,\ (d,x) + (0.5, 0)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}, \mathbb{Z}) \wedge [a,b] \in V \} \\ \cup \quad & \{\ [a,b] := [(d,x) + (0.5, 0)\ ,\ (d,x) + (0.5, -0.5)] \quad | \\ & \quad \forall (d,x) \in (2\mathbb{Z}+1, \mathbb{Z}) \wedge [a,b] \in V \} \end{aligned}$$

Zusätzlich müssen *Quelle* $s$ und *Senke* $t$ mit dem Gitter verbunden werden. Dies geschieht an den Rändern des Zylinders:

$$\begin{aligned} E_{st} &= \{\, [s, (-2n - 0.5, x)] \quad | \quad \forall x \in I \setminus \mathbb{Z} \} \\ &\cup \{\, [(2n + 0.5, x), t] \quad | \quad \forall x \in I \setminus \mathbb{Z} \} \end{aligned}$$

Man beachte, dass sämtliche Kanten gerichtet sind. Die Kantenmenge des Graphen ist die Vereinigung dieser Mengen $E = E_d \cup E_v \cup E_h \cup E_{st}$.

### Flächen

Die entstandenen Flächen $F_{d,i}$ des Graphengitters sind die dualen Flächen zu den korrespondierenden Knoten $(d, i)$ im primalen Graph. Sie repräsentieren die *Korrespondenzen* zwischen Kurvenpunkten. Die durch die Eckknoten $(\pm 0.5, \pm 0.5)$ begrenzte Fläche ist die Fläche $F_{0,0}$. Aus der Knotennummerierung im primalen Graphen folgt, dass die Fläche im dualen Graphen mit den Eckknoten

$$(d \pm 0.5, i \; - \; 0.5d - 0.5), (d \pm 0.5, i \; - \; 0.5d + 0.5)$$

der Fläche $F_{d,i}$ entspricht.
Vom Knoten $(d, i)$ im primalen Graphen $H$ führen genau drei Kanten zu den Knoten $(d-1, i)$, $(d, i+1)$ und $(d+1, i+1)$. Der Cut soll ebenfalls exakt die drei Wege von der Fläche $F_{d,i}$ zu den Flächen $F_{d-1,i}$, $F_{d,i+1}$ und $F_{d+1,i+1}$ gehen können (siehe Abb. 6.5). Daher darf er nur die Kanten $[(d, i), (d-1, i)]$, $[(d, i), (d, i+1)]$ oder $[(d, i), (d+1, i+1)]$ schneiden. Sie dürfen jedoch nicht *rückwärts* geschnitten werden. Um genau dies zu verhindern, werden die Kapazitäten in Rückwärtsrichtung auf unendlich gesetzt.
Außerdem sollen die Kosten, die beim Passieren der Kanten entstehen, denen im primären Graphen gleichen. Folgende Kantenkapazitäten stellen diese Bedingungen sicher (vgl. Abschnitt 5.2):

$$c(e) = \begin{cases} \frac{m_{d,i} \, + \, m_{d-1,i}}{2(n-1)} \; / \; \infty & \text{falls } \; e \in E_v \\ \frac{m_{d,i} \, + \, m_{d+1,i+1}}{2(n-1)} \; / \; \infty & \text{falls } \; e \in E_h \\ \left( \frac{m_{d,i} \, + \, m_{d,i+1}}{\sqrt{2}(n-1)} \; + \frac{\alpha}{(n-1)} \right) \; / \; \infty & \text{falls } \; e \in E_d \\ \infty \; / \; \infty & \text{sonst} \end{cases}$$

wobei die Matrix $M$ mit den Einträgen $m_{d,i}$ die paarweisen, quadratischen Abstände der Invarianten enthält. Formal ist

$$m_{d,i} := (I_1(i) - I_2(i - d))^2$$

Wobei $I_{1,2}$ beispielsweise die Krümmungsfunktionen der Kurven sein können.

### 6.3.1 Korrektheit des Graph Cut-Ansatzes

Der Cut, den man durch die Berechnung des maximalen Flusses in dem konstruierten Graphen erhält, besteht aus einer Kantenmenge, die als Korrespondenzabbildung zwischen den beiden Konturen interpretiert werden kann. Denn wird die Kante $[(d_1, i_1), (d_2, i_2)]$ geschnitten, so sind die Korrespondenzen $(i_l, i_l - d_l),\ l = 1, 2$ in der Abbildung enthalten. Noch ist allerdings unklar, ob jeder Cut mit *endlichen* Kosten auch eine *gültige* Korrespondenzabbildung repräsentiert und umgekehrt, ob für jede gültige Korrespondenzabbildung auch ein möglicher Cut existiert. Weiterhin muss sichergestellt werden, dass die Distanz zwischen zwei Konturen in der Graph Cut- sowie in der Kürzeste-Wege-Formulierung gleich, beide Ansätze zur Berechnung der Distanz zwischen zwei Kurven also äquivalent sind. Das soll in diesem Unterabschnitt gezeigt werden.
Es steht fest, dass der Cut mit den kleinsten Kosten $< \infty$ sein muss, denn es genügt, einen Cut mit endlichen Kosten anzugeben:

$$C_{id} = \{e \in E_d \mid e = [(-0.5, x), (0.5, x)], \forall x \in I\}$$

entspricht der Identitätsabbildung, die jeden Punkt $i$ der ersten Kontur auf den Punkt $i$ der zweiten Kontur abbildet. Im Folgenden wird mit *endlicher Cut* ein Cut mit endlichen Kosten bezeichnet.

**Satz.** *Jede gültige Disparitätsfunktion wird durch einen endlichen Cut repräsentiert.*

*Beweis.* In Abschnitt 6.2.1 wurde bereits gezeigt, dass im Graphen $G$ für jede gültige Disparitätsfunktion ein geschlossener Weg enthalten ist, der den Zylinder einmal umwandert. Da der primäre Graph $H$ aber $G$ als Untermenge enthält, gilt diese Aussage erst recht für $H$. Folglich werden von jedem gültigen Disparitätsweg der linke und der rechte Rand des Zylinders voneinander getrennt. Der geschlossene Weg der Länge $l$ in $H$ besteht aus einer Knotenfolge $((d_1, i_1), ..., (d_l, i_l), (d_1, i_1))$. Wir betrachten im dualen Graphen $H^*$ die entsprechende Folge von Flächen: $(F_{d_1,i_1}, ..., F_{d_l,i_l}, F_{d_1,i_1})$. Jede Kante in $H$ von Knoten $(d_j, i_j)$ zum Knoten $(d_{j+1}, i_{j+1})$ für $j \in [1..l-1]$ (analog für die Kante von Knoten $(d_l, i_l)$ nach $(d_1, i_1)$) muss eine von drei Kantentypen sein: *horizontal*, *vertikal* oder *diagonal*. Die entsprechende Kante grenzt aber per Definition des dualen Graphen ebenfalls an den beiden Flächen $F_{d_j,i_j}$ und $F_{d_{j+1},i_{j+1}}$ an. Die Kosten der Kante in die entsprechende Richtung wurden gerade so gewählt, dass sie identisch mit jenen der Kante im primalen Graphen $H$ sind. Es gibt also einen endlichen Cut in $H^*$, und dessen Kosten sind identisch zum Korrespondenzweg in $H$. □

**Satz.** *Jeder endliche Cut beschreibt eine gültige Disparitätsfunktion.*

*Beweis.* Sei ein endlicher, $(s,t)$-separierender Cut des dualen Graphen $H^*$ gegeben. Ein Standardresultat aus der Graphentheorie besagt, dass die Kanten eines gerichteten Cuts im primalen Graphen einen *einfachen*, gerichteten Kreis beschreiben [30]. Nun muss noch gezeigt werden, dass die Kanten in diesem Kreis bezüglich der Graphenkanten in die richtige Richtung zeigen und damit eine gültige Korrespondenzabbildung beschreiben. Betrachten wir die Folge von Flächen $(F_{d_1,i_1}, ..., F_{d_l,i_l}, F_{d_1,i_1})$, die der Cut als Weg durchläuft. Für eine Fläche $F_{d_j,i_j} := F_{d,i}$ der Folge muss $F_{d_{j+1},i_{j+1}}$ eine der Flächen $F_{d-1,i}$, $F_{d,i+1}$ oder $F_{d+1,i+1}$ sein, denn laut Konstruktion von $H^*$ sind dies die einzigen benachbarten Flächen $F_{d,i}$, die über endliche Kosten erreichbar sind. Bezogen auf den primären Graphen $H$ bedeutet dies jedoch, dass vom Knoten $(d,i) := (d_j, i_j)$ aus zu einem der Knoten $(d-1,i)$, $(d,i+1)$ oder $(d+1,i+1)$ über eine *horizontale*, *vertikale* oder *diagonale* Kante gegangen wird, welche laut Definition die gleichen Kosten besitzt, wie die vom Cut geschnittene Kante in $H^*$. In Abschnitt 6.2 wurde bereits gezeigt, dass jeder einfache, geschlossene Pfad einen gültigen Disparitätsweg beschreibt. □

Die Frage nach der Äquivalenz von Graph Cut- und Kürzeste-Wege-Formulierung lässt sich mit diesem Ergebnis leicht beantworten. Denn nach den beiden bewiesenen Sätzen gibt es eine eins-zu-eins Korrespondenz zwischen endlichen Cuts und gültigen Korrespondenzwegen, deren Kosten einander gleichen. Der minimale Cut des Graphen $H^*$ muss folglich gleich dem kürzesten Weg in $\tilde{G}$ aus Abschnitt 6.2 sein.

Zu bemerken ist noch, dass aufgrund der nach oben hin willkürlich wählbaren Länge des Zylinders $H^*$ durchaus mehrere endliche Cuts denselben kürzesten Weg beschreiben können, in diesem Sinne also in der Tat keine Bijektion von endlichen Cuts und Korrespondenzabbildungen besteht. Dieser Umstand ist äquivalent zu der Tatsache, dass - wie in Abschnitt 6.2.2 bereits bemerkt - $d(i) \equiv d(i) + k \cdot n, \forall k \in \mathbb{Z}$ gilt. Da diese äquivalenten Cuts alle dieselben Korrespondenzabbildungen beschreiben, stellt diese Redundanz jedoch kein Problem dar.
Die Wahl der Zylinderlänge $4n+1$ in der Formalisierung des Graphen $H$ hatte den Sinn, dass der wesentlich kleinere Graph $G$ aus Abschnitt 6.2.3 als Untermenge enthalten ist, was im Korrektheitsbeweis Verwendung fand. In der Implementierung des Graph Cut-Ansatzes kann und sollte der Graph $G^*$ zur Berechnung verwendet werden, um Speicherplatz und Laufzeit zu sparen. Mit dieser Version des Graphen wurden auch alle Tests in vorliegender Arbeit durchgeführt.

# Kapitel 7

# Effizientes Shape Matching über Max-Flow / Min-Cut Algorithmen

Nachdem in Kapitel 6 eine Graph Cut Formulierung zum Shape Matching vorgestellt wurde, muss nun auf dem Graphen der minimale Cut berechnet werden. Der Schwerpunkt der Untersuchung liegt auf der Effizienz als wichtiges Kriterium für die Brauchbarkeit eines solchen Ansatzes.
In Abschnitt 7.1 wird ein Überblick über Max-Flow / Min-Cut Algorithmen gegeben und die Grundproblematik besprochen. Mit diesen Voraussetzungen wird in Abschnitt 7.2 zunächst ein in der Bildverarbeitung bekannter Max-Flow Algorithmus für allgemeine Flussnetzwerke eingeführt und auf das Shape Matching Problem aus Kapitel 6 angewandt. Daraufhin wird ein Algorithmus speziell für planare Graphen, der bisher noch gar nicht im Bildverarbeitungskontext zu sehen war, untersucht und angewendet. Die Ergebnisse werden schließlich verglichen.

## 7.1 Berechnung des maximalen Flusses

Es existiert eine Reihe von Algorithmen zur Berechnung des maximalen Flusses / minimalen Cuts für unterschiedliche Graphentypen. Neben der Laufzeitkomplexität eines Max-Flow Algorithmus hängt dessen Effizienz meistens von der speziellen Graphenbeschaffenheit ab. Das betrifft sowohl die Graphenstruktur als auch die Kapazitätsfunktion. Daher lassen sich die Laufzeiten von Netzwerkflussalgorithmen oft nicht pauschal vergleichen, Im Wesentlichen fallen die Algorithmen zur Flussberechnung in eine von *zwei* Grundkategorien: *Pfad-Augmentierung* und *Push-Relabel*-Verfahren.

### 7.1.1 Algorithmen auf Basis augmentierender Pfade

Bei der Pfad-Augmentierung wird der Fluss des Netzwerkes - angefangen bei Null - stückweise erhöht. Dabei wird die Invariante aufrechterhalten, dass jeder Schritt

des Algorithmus in einen gültigen Fluss resultiert. Kann der Fluss nicht mehr erhöht werden, so ist er maximal. Während des Verfahrens wird in jeder Kante $e$ der aktuelle Fluss $f(e)$ gespeichert und mit $0$ initialisiert.

### Augmentierende Pfade

Ein *Augmentierender Pfad* $P$ ist ein Pfad aus Rückwärts- und Vorwärtskanten von $s$ nach $t$. Vorwärtskanten zeigen längs des Pfades in Richtung Senke, während Rückwärtskanten zur Quelle zeigen. Für jede Kante $e$ in $P$ gilt:

- $e$ ist Vorwärtskante $\Rightarrow f(e) < c(e)$
- $e$ ist Rückwärtskante $\Rightarrow f(e) > 0$

Vortwärtskanten oder Rückwärtskanten mit den genannten Eigenschaften sind in der entsprechenden Richtung *unsaturiert*. Bezeichne $F$ die Menge der Vorwärtskanten und $R$ die Menge der Rückwärtskanten in $P$. Ist ein augmentierender Pfad gefunden, kann der Fluss auf diesem Pfad folgendermaßen erhöht werden:

$$f(e) = \begin{cases} f(e) + \delta & \text{falls } e \in F \\ f(e) - \delta & \text{falls } e \in R \end{cases}$$

wobei $\delta$ die *Restkapazität*

$$\delta := \min\left\{ \min\{c(e) - f(e) \mid e \in F\},\ \min\{f(e) \mid e \in R\}\right\}$$

bezeichnet. Die Wahl von $\delta$ stellt folgendes sicher:

- Keine Kante auf $P$ transportiert mehr Fluss, als ihre Kapazität erlaubt.
- $P$ ist *nach* der Erhöhung kein augmentierender Pfad mehr ($P$ ist *saturiert*).

Es ist leicht zu sehen, dass die Flusseigenschaft nach der Augmentierung erhalten bleibt und dass der Gesamtfluss von $s$ nach $t$ um $\delta$ erhöht wird. Die Einbeziehung von Rückwärtskanten in den Augmentierungsvorgang ist für die Erreichung des maximalen Flusses u.U. unerlässlich und kann als Umleiten bereits vorhandenen Flusses interpretiert werden.

### Algorithmen

Ford und Fulkerson stellten 1962 [11] den Grundalgorithmus auf Basis augmentierender Pfade auf: Durch Tiefensuche auf dem Graphen werden solange augmentierende Pfade gesucht und augmentiert, bis keiner mehr gefunden werden kann, sprich: keiner mehr existiert. Die maximale Laufzeit des Algorithmus hängt allerdings von der Summe der Kantenkapazitäten im Netzwerk ab. Bei ganzzahligen Kapazitäten ist die obere Laufzeitschranke durch $\Omega\left(\sum_{e \in E} c(e)\right)$ gegeben.

Anfang der siebziger Jahre stellten Dinic und Edmonds und Karp unabhängig voneinander fest, dass nach höchstens $|V| \cdot |E|$ Augmentierungsschritten der maximale Fluss erreicht ist, sofern stets kürzeste Pfade von $s$ nach $t$ augmentiert werden. *Edmonds und Karp* publizierten auf dieser Basis einen Algorithmus mit Laufzeit $O(|V| \cdot |E|^2)$ - der auf "dünnen" Graphen effizient läuft, während *Dinic* einen $O(|V|^2 \cdot |E|)$-Algorithmus vorschlug [10] [8].

Auf diesen klassischen Resultaten bauen auch die im Folgenden vorgestellten Algorithmen auf. Sie sind Ergebnisse aus der neueren Forschung. Zwei effiziente Methoden werden dann zur Graph Cut-Berechnung auf dem Shape Matching-Graphen aus Kapitel 5 herangezogen und die Ergebnisse verglichen.

### 7.1.2 Push-Relabel Algorithmen

Wird bei Pfad-Augmentierungs-Algorithmen stets ein Fluss aufrecht erhalten und ein saturierter Cut gesucht (der dann der minimale ist), so wird beim Push-Relabeling invers verfahren: Jeder Schritt des Algorithmus resultiert in einem *Preflow*, d.h. einem Fluss, bei dem das Erhaltungsgesetz nicht gelten muss. Sämtliche Knoten des Graphen erhalten außerdem ein *Labeling*, das bestimmte Voraussetzungen erfüllt. Das Labeling steuert den nächsten Schritt des Algorithmus. Es besitzt folgende Eigenschaften:

- Das Labeling gibt einen Cut an, dessen Kanten alle saturiert sind (solange der Algorithmus nicht terminiert, handelt es sich jedoch um einen Preflow).
- Das Labeling gibt für jeden Knoten eine untere Schranke der Distanz zur Senke im Residualgraphen an.

In jedem Schritt des Algorithmus wird der überschüssige Fluss eines Knotens in Richtung Senke geleitet (*Push*). Fluss wird immer zu Knoten mit kleinerer Distanz zur Senke geschoben. Ist die Distanz zur Senke unendlich (Fluss kann nicht weiter geleitet werden), wird der Überschuss zurück zur Quelle geleitet. Am Ende jedes Schrittes werden die Labels der Knoten aktualisiert (*Relabel*). Da Push-Relabel-Methoden in den weiteren Ausführungen keine Rolle spielen, wird hier nicht weiter auf das Verfahren eingegangen.

### 7.1.3 Dynamische Bäume

Das Kernproblem pfadbasierter Maxflow-Algorithmen ist, in geringer Zeit einen augmentierenden Pfad zu finden, insbesondere dann, wenn dieser bestimmte Eigenschaften besitzen soll (z.B. finde den *kürzesten* oder den am *weitesten links stehenden Pfad* etc.). Um nicht in jedem Schritt eine aufwendige Tiefensuche auf dem Residualgraphen ausführen zu müssen, verwalten die meisten Algorithmen einen

Wald von *Suchbäumen* (dynamische Bäume). Die Verwaltung eines *einzelnen* Suchbaumes kann als Spezialfall angesehen werden. Die Knotenmenge der Suchbäume ist eine Teilmenge der Graphenknoten. Knotenmengen verschiedener Bäume sind disjunkt. Eine Kante ist nur dann Teil eines Suchbaums, falls die beteiligten Knoten im Flussnetzwerk durch eine residuale Kante miteinander verbunden sind. Einer der Bäume zeichnet sich dadurch aus, dass seine Wurzel - je nach Algorithmus - entweder die Quelle oder die Senke ist. Ziel ist es, im Hauptbaum Pfade von der Quelle zur Senke zu suchen. Da die Baumkanten allesamt residuale Graphenkanten repräsentieren, ist auch der gesuchte Pfad residual.
Das Grundschema der Maxflow-Algorithmen mit Suchbaum ist das folgende. Solange es einen Pfad innerhalb des Suchbaumes von $s$ nach $t$ gibt, wiederhole folgende Schritte:

1. Augmentiere den Pfad von der Quelle zur Senke.
2. Ermittele saturierte Kanten.
3. Entferne diese Kanten.
4. Verbinde den abgetrennten Unterbaum entweder sofort oder zu einem späteren Zeitpunkt (Algorithmusabhängig) über eine residuale Kante mit dem Baum, dessen Wurzel die Quelle bzw. Senke ist.

Da diese Operationen in jedem Schritt des Maxflow-Algorithmus ausgeführt werden müssen, ist dessen Laufzeit stark von der Datenstruktur abhängig, die für die Verwaltung der dynamischen Bäume eingesetzt wird. 1982 entwickelten Sleator und Tarjan eine geeignete Datenstruktur [25], deren amortisierte Laufzeit in jeder der genannten Operation $O(\log(n))$ in der Anzahl der Knoten beträgt. Die Datenstruktur ist mit einem nicht unerheblichen Aufwand verbunden, verhindert jedoch in den meisten Fällen, dass alle Knoten entlang eines Pfades während der Augmentierung angefasst werden müssen.

## 7.2 Shape Matching mittels allgemeiner Graph Cut-Algorithmen

Der in Kapitel 6 konstruierte Graph dient dem Shape Matching und gehört damit in den Bereich der Bildverarbeitung. Es liegt daher zunächst nahe, Graph Cut Algorithmen anzuwenden, die in der Bildverarbeitung bereits etabliert sind. Einer davon ist der Algorithmus von Boykov und Kolmogorov aus dem Jahre 2004 [6]. Der Algorithmus berechnet den maximalen Fluss / minimalen Cut auf allgemeinen, gerichteten Graphen, wurde jedoch insbesondere in Hinblick auf typische Bildverarbeitungsprobleme - wie der Bildsegmentierung - optimiert. In diesem Abschnitt soll der Algorithmus getestet werden. Zunächst jedoch wird er näher untersucht, um das Laufzeitergebnis interpretieren zu können.

### 7.2.1 Beschreibung des Algorithmus

Der Algorithmus von Boykov und Kolmogorov berechnet den maximalen Fluss durch Pfad-Augmentierung. Dabei versucht er nach Möglichkeit stets kürzeste Pfade zu finden, um eine polynomielle Laufzeit zu erreichen. Eine Garantie dafür gibt der Algorithmus jedoch nicht.

#### Suchbäume

Zur Berechnung der augmentierenden Pfade werden exakt zwei disjunkte Suchbäume $S$ und $T$ mit Wurzel $s$ bzw. $t$ verwaltet. Dies geschieht *nicht* über die in Abschnitt 7.1 erwähnte Datenstruktur von Sleator und Tarjan. Graphenknoten gehören entweder zu $S$, zu $T$, oder sie sind *frei*. Die Bäume sind folgendermaßen definiert:

- Jeder Pfad in $S$ von der Wurzel $s$ zu jedem anderen Knoten besteht aus *unsaturierten Kanten*.
- Jeder Pfad von einem Knoten in $T$ zur Wurzel $t$ besteht vollständig aus *unsaturierten Kanten*.

Berühren sich die Bäume $S$ und $T$ über eine unsaturierte Kante von einem Knoten aus $S$ nach einem Knoten aus $T$, so folgt aus der Definition, dass ein augmentierender Pfad von $s$ nach $t$ gefunden wurde.

#### Algorithmus

Zunächst gilt $S = \{s\}$ und $T = \{t\}$. Der Algorithmus führt in jeder Phase drei Operationen durch:

1. **Baumerweiterung**: $S$ und $T$ werden um residuale Kanten erweitert, bis sich beide Bäume treffen.
2. **Augmentierung**: Der gefundene Pfad von $s$ nach $t$ wird augmentiert und saturierte Kanten gelöscht. Dabei entsteht im Allgemeinen ein Wald.
3. **Wiederherstellung**: Die abgespaltenen Unterbäume werden wieder an $S$ bzw. $T$ angegliedert oder, falls unmöglich, in freie Knoten umgewandelt.

Falls keine Baumerweiterung mehr möglich ist und sich $S$ und $T$ nicht berühren, terminiert der Algorithmus.

### 7.2.2 Eigenschaften und Laufzeit

Der Algorithmus von Boykov und Kolmogorov kann sowohl auf planaren, als auch auf nicht-planaren Graphen ausgeführt werden. Da die Datenstruktur für die Bäume

$S$ und $T$ äußerst simpel gehalten ist, benötigt die Augmentierung eines Pfades im Gegensatz zur Datenstruktur von Sleator und Tarjan lineare Zeit in der Knotenzahl. Es ist daher zu erwarten, dass das Verfahren im Verhältnis zu anderen pfadbasierten Algorithmen besonders effizient auf Graphen läuft, deren Wege von der Quelle zur Senke relativ klein sind.

### Laufzeit

Der Vorteil gegenüber vergleichbaren Algorithmen, die stets den kleinsten Augmentierungspfad suchen, ist, dass der Suchbaum nie von Grund auf neu konstruiert werden muss. Gleichzeitig vermeidet der Algorithmus komplexe Datenstrukturen für die Verwaltung des Suchbaumes. Dies alles spart Laufzeit, führt allerdings dazu, dass nicht garantiert werden kann, dass immer der kürzeste Pfad gefunden wird. Somit sind im Worst-case mehr als nur $|V| \cdot |M|$ Pfad-Augmentierungen nötig. Die Worst-case Laufzeit ist daher abhängig von den Kosten des minimalen Cuts $O(mn^2 \cdot |C|)$.
Dennoch zeigen die Experimente in [6], dass der Algorithmus auf für die Bildverarbeitung typischen Graphen anderen effizienten Pfad-Augmentierungs- und Push-Relabel-Verfahren deutlich überlegen ist. Insbesondere in Graphenkonstruktionen zur Bildsegmentierung, bei denen die Pfadlänge von der Quelle zur Senke oft relativ klein ist, profitiert der Algorithmus von der Aufrechterhaltung der Suchbäume.

### Heuristiken

Der Graph Cut-Algorithmus ist stark heuristisch konzipiert. Die Laufzeit insgesamt ist zwar nicht beweisbar effizient, hingegen wurde der Algorithmus in praktischer Hinsicht stark optimiert. Einige zusätzliche Heuristiken, die in den Algorithmus eingehen sind:

- Die Baumerweiterung wird nur an Knoten versucht, die nicht durch Knoten desselben Baums verbaut sind (aktive Knoten). So wird keine Zeit an ergebnislose Erweiterungsversuche verschwendet.
- Die Abarbeitungsreihenfolge aktiver Knoten in der Erweiterungsphase wird über eine Form des *First-In-First-Out*-Prinzips gesteuert. Dadurch ähnelt die Baumerweiterung einer Breitensuche, und es werden eher kürzeste Pfade zur Senke gefunden.
- Knoten, die mit der Quelle verbunden sind, werden markiert. So kann in der Wiederherstellungsphase schnell ein neuer Vaterknoten des abgetrennten Unterbaums gefunden werden.
- Jeder Knoten wird mit der Entfernung zur Quelle versehen. So können im Erweiterungsschritt solche Knoten gewählt werden, die zu kürzeren Wegen führen.

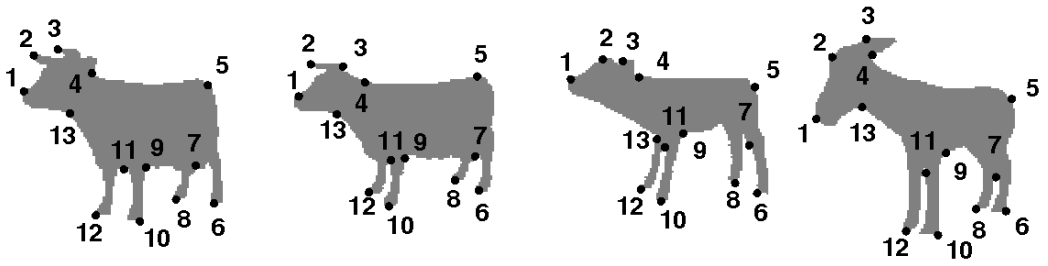



Abbildung 7.1: Vergleich eines Rindes (links oben) mit einem anderen Rind, zwei Kälbern und einem Esel über Graph Cuts.

### 7.2.3 Ergebnisse

In diesem Unterabschnitt soll zum einen die Effizienz des Algorithmus von Boykov und Kolmogorov auf dem Shape Matching Graphen aus Kapitel 6 untersucht werden. Zum anderen werden Ergebnisse von Formenvergleichen gezeigt.
Abbildung 7.1 zeigt mehrere solcher Vergleichsergebnisse. Die Korrespondenzen sind alle sinnvoll und intuitiv, obgleich sich die Formen gerade im Kopfbereich in der Anzahl der Ausbuchtungen unterscheiden. Per Konstruktion resultiert der Vergleich derselben Formen mittels des Verfahrens aus Kapitel 5 über dynamische Programmierung in identische Korrespondenzen und Formdistanzen. Die Distanzen in Abbildung 7.1 betragen $0.0208$, $0.1130$ und $0.0992$.
Abbildungen 7.2 und 7.3 zeigen die Ergebnisse der Laufzeittests. Es wurden zwei verschiedene Formenpaare verglichen. Die Zahl der am Vergleich beteiligten Punkte betrug $50$ bis $500$. Die Laufzeitkurven zeigen eine interessante Eigenschaft des Max-Flow Algorithmus im Zusammenhang mit Shape Matching: Das **erste Formenpaar** besteht aus zwei sehr unterschiedlichen Formen, die Fledermaus ist kompakt und ohne starke Ausbuchtungen, der Käfer besitzt viele langgezogene Formteile mit stellenweise sehr hohen Krümmungen. Die Spannweite der benötigten Zeiten reicht von $0.04$ Sekunden bei $50$ Punkten bis fast $1.5$ Minuten bei $500$ Abtastpunkten. Eine lineare Regression durch den logarithmisch abgetragenen Graphen ergab eine Liniensteigung von $3.15$. Der Algorithmus hat also eine Laufzeit, die etwas schlechter ist als kubisch. Ebenfalls bemerkenswert ist die zunehmende Streuung der gemessenen Laufzeiten mit wachsender Punktzahl, die nicht auf Messungenauigkeiten zurückzuführen sind. Das **zweite Formenpaar** besteht aus zwei identischen, zueinander gedrehten Formen. Die für den Vergleich benötigten Zeiten steigen von $0.01$ bis lediglich zu $7.8$ Sekunden. Es wurde ein Laufzeitexponent von $2.76$ gemessen; die Laufzeit ist damit etwas besser als kubisch.
Vergleichbare Experimente bestärkten die Folgerung, dass der Vergleich zueinander ähnlicher Formen eine kleine Laufzeit mit sich bringt. Im Mittel ist die benötigte Laufzeit jedoch eher schlechter als kubisch.

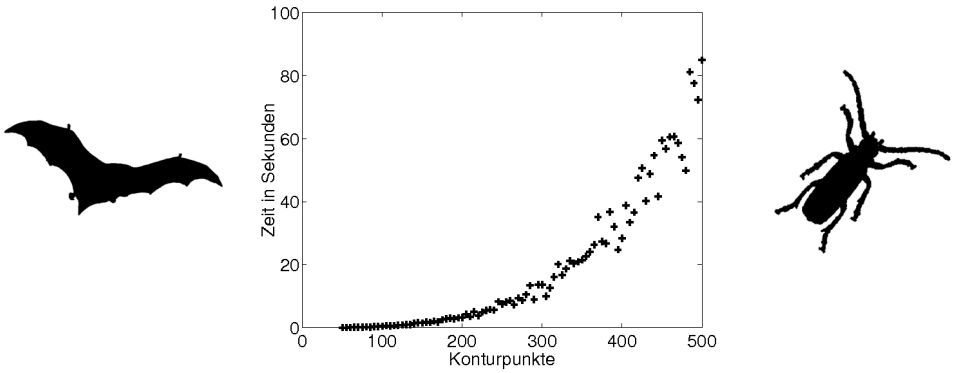



Abbildung 7.2: Laufzeiten der Vergleiche von Fledermaus und Käfer über den Algorithmus von Boykov und Kolmogorov bei $50$ bis $500$ Punkten entlang der Kontur.

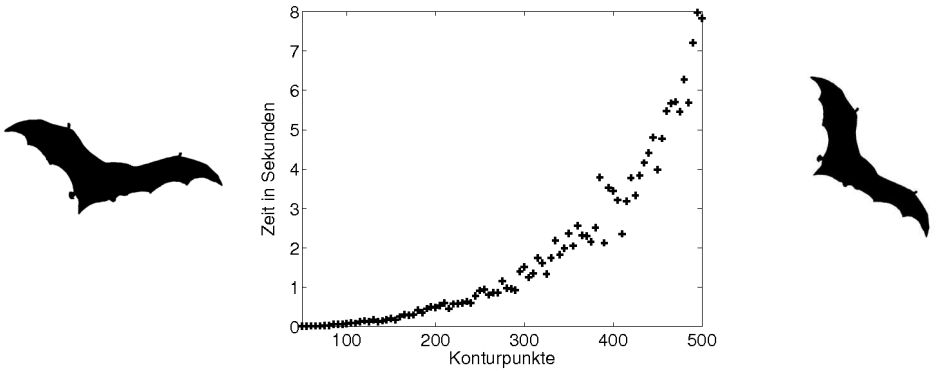



Abbildung 7.3: Laufzeiten der Vergleiche von Fledermaus und rotierter Fledermaus über den Algorithmus von Boykov und Kolmogorov bei $50$ bis $500$ Punkten entlang der Kontur. Man beachte den Kontrast zum Vergleich unähnlicher Formen.

Die beobachteten Laufzeiten decken sich mit der Analyse des Max-Flow Algorithmus. Dessen heuristische Veranlagung lässt keine verlässliche Schätzung der Laufzeit zu. Hinzu kommt, dass es im beschriebenen Graphen kaum kurze augmentierende Pfade gibt. Im Gegenteil: sämtliche Pfade haben eine Mindestlänge von $n$ Kanten. Die simple Datenstruktur zur Verwaltung des Suchbaumes zwingt jedem Augmentierungsschritt folglich mindestens $n$ Iterationen auf.

## 7.3 Shape Matching mittels planarer Graph Cut-Algorithmen

Schon bei der Konstruktion des Shape Matching Graphen in Kapitel 6 wurde betont, dass dieser planar ist. Nachdem die Laufzeiten des Algorithmus von Boykov und Kolmogorov im Shape Matching Kontext keine Verbesserung gegenüber dem

Verfahren über dynamische Programmierung bot, soll in diesem Abschnitt die Effizienz der Graph Cut Berechnung über planare Max-Flow Algorithmen untersucht werden.
Eine wesentliche Eigenschaft planarer Graphen ist die Möglichkeit, sämtliche Pfade zwischen zwei Knoten mit einer Ordnung versehen zu können. Dies resultiert daraus, dass für jeden Knoten eine solche Ordnung für die inzidenten Kanten bestimmt werden kann. Die Grundidee effizienter Maxflow-Algorithmen für planare Graphen ist, die Pfade zwischen Quelle und Senke systematisch aufzuzählen und zu augmentieren, so dass garantiert wird, dass jede Kante nur $O(1)$ mal angefasst werden muss.

Der erste bekannte $O(\log(n))$ Algorithmus speziell für planare Graphen wurde von Weihe 1994 veröffentlicht [29]. Ein Eingabegraph muss für Weihes Algorithmus bestimmte Voraussetzungen erfüllen:

1. Der Graph enthält keine Kreise im Uhrzeigersinn.

2. Jede Kante $(v, w)$ gehört zu einem einfachen $(v, t)$-Pfad und einem einfachen $(s, w)$-Pfad.

3. Jeder Knoten - bis auf $s$ und $t$ - besitzt genau 3 inzidente Kanten, hat also Grad 3. $s$ hat Grad 1.

*Jeder* planare Graph kann allerdings derart umgeformt werden, dass er die Bedingungen erfüllt. Weihes Algorithmus verwaltet - wie für Pfadaugmentierungsalgorithmen üblich - einen Wald von Suchbäumen und nutzt dafür die Datenstruktur von Sleator und Tarjan (siehe Abschnitt 7.1). Jede Kante wird höchstens zweimal angefasst. Zusammen mit der Zeitkomplexität der Datenstruktur ergibt sich die Gesamtlaufzeit.

Die Praxistauglichkeit des Algorithmus wird durch die hohen Voraussetzungen an den Eingabegraphen stark eingeschränkt. Auch wenn Bedingung 2 in linearer Zeit, und Bedingung 1 in Zeit $O(m \log(n))$ [1] umgesetzt werden kann, ist Bedingung 3 - wie Borradaile und Klein in [4] bemerken - problematischer, und eine untere Laufzeitschranke für die Graphentransformation ist $\Omega(n^2 \log(n))$.

2006 publizierten Borradaile und Klein in [4] einen Pfad-Augmentierungs Algorithmus, der bei gleicher Laufzeit weniger Voraussetzungen an den Eingabegraphen stellt als der Algorithmus von Weihe. Er wird zunächst beschrieben, um schließlich dessen Laufzeit auf dem Shape Matching Graphen zu testen.

[1] Siehe Abschnitt 7.3.5.

### 7.3.1 Voraussetzungen

Zunächst sollen einige begriffliche Voraussetzungen geschaffen werden, die für das Verständnis des Algorithmus von Bedeutung sind.

#### Darts

Jede Kante $(v, w)$ wird im Folgenden in zwei *Darts* unterteilt, einen *Vorwärtsdart* $(v, w)_d$ und einen *Rückwärtsdart* $(w, v)_d$. Für einen Dart $d$ wird der Dart in Gegenrichtung mit $rev(d)$ bezeichnet. Beide Darts besitzen eine separate Kapazität, falls nicht anders angegeben ist die Rückwärtskapazität jedoch $0$. Wird der Fluss der Kante in Vorwärtsrichtung erhöht, so wird der Fluss des Rückwärtsdarts gleichzeitig verringert. Analoges gilt für die Augmentierung der Rückwärtsrichtung. Fluss kann immer dann erhöht werden, falls $f(d) < c(d)$.
Der Vorteil der Darts-Sichtweise ist:

- Kanten $(v, w)$ und $(w, v)$ der Kantenmenge $E$ können zu *einer* Kante verschmolzen werden deren *Hinkapazität* $c(v, w)$ und deren *Rückkapazität* $c(w, v)$ beträgt. Dadurch wird der Graph einfacher.

- *Augmentierungen* können *einheitlich* behandelt werden. Augmentierungspfade bestehen im Folgenden immer aus Ketten von *Darts*, die von $s$ nach $t$ zeigen, statt aus Vorwärts- und Rückwärtskanten. Beim Augmentierungsschritt wird der Fluss für alle Darts um den gleichen Wert erhöht. Die korrespondierenden Rückwärtskanten werden um den gleichen Wert dekrementiert.

Die zweite Eigenschaft ist die Voraussetzung für die Verwendung der $O(\log(n))$-Datenstruktur. Für einen Dart $d$ in $G$ ist der duale Dart $d^*$ im dualen Graphen $G^*$ definiert als $d^* := (f_l(d), f_r(d))$. Dies entspricht einer Drehung des Darts um 90° im Uhrzeigersinn (vgl. Abschnitt 6.1.1).

#### Links- / Rechtsrelation

Ein Pfad $P$ von $v$ nach $w$ in einem planaren Graphen ist genau dann *links* von einem anderen Pfad $Q$ von $v$ nach $w$, falls die *Verknüpfung* der Pfade $P$ und $rev(Q)$ einen Kreis im Uhrzeigersinn darstellt. $rev(P)$ bezeichnet dabei die Umkehrung des Pfades $P$. Analoges gilt für die *Rechts*relation. Ein leicht beweisbares Theorem auf planaren Graphen ist folgendes: Falls zwei unterschiedliche Pfade $P$ und $Q$ von $v$ nach $w$ sich nicht schneiden, so folgt, dass $P$ entweder links von $Q$ oder rechts von $Q$ bzw. umgekehrt sein muss. Dadurch entsteht eine *Ordnung* von Pfaden auf einem planaren Graphen.

#### Unendliche Fläche $\mathbf{f_\infty}$

In Abschnitt 6.1.1 wurde die unbegrenzte Fläche $f_\infty$ von planaren Graphen eingeführt. Es wird im Folgenden davon ausgegangen, dass $f_\infty$ zu $t$ inzident *ist* oder - falls der Graph auf der Kugel eingebettet ist - *gewählt* wird.

#### Bedingung an den Eingabegraphen

Die einzige Bedingung an den Eingabegraphen ist nun:

*Der Graph darf keine residuale, d.h. nicht-saturierte Kreise im Uhrzeigersinn enthalten.*

Es reicht nicht, dass er keine Uhrzeigerkreise enthalten darf, da es für jede Kante eine positive Rückwärtskapazität geben kann.

### 7.3.2 Suchbäume

Zunächst werden für jeden Knoten die inzidenten Kanten, ähnlich wie in Weihes Algorithmus, mit einer *Ordnung im Gegenuhrzeigersinn* versehen. Es werden *zwei* Bäume während des Algorithmus aufrechterhalten.

#### Suchbaum T

**Der Algorithmus verwaltet einen Suchbaum T von *Darts* im Graphen G. Der Baum hat die Funktion, augmentierende Pfade in $G$ zu finden und wird so konstruiert und verwaltet, dass der Pfad von s nach t im Baum stets der am weitesten links stehende residuale Pfad im Graphen ist.** Folgende Eigenschaft des Baumes wird während des Algorithmus als *Invariante* aufrechterhalten: [2]

- $T$ ist ein *überspannender Baum* von $G$, dessen Kanten alle zur Wurzel $t$ zeigen.

Der Baum wird zu Beginn des Algorithmus konstruiert, indem eine **Tiefensuche** von $t$ aus auf dem Graphen erfolgt. Genauer wird eine *Right-first Search* [3] durchgeführt, die sicherstellt, dass der Pfad von $s$ nach $t$ der am weitesten links stehende augmentierende Pfad in $G$ ist. **Jeder Dart in T ist unsaturiert**. Zur Verwaltung von $T$ wird die Datenstruktur von Sleator und Tarjan aus Abschnitt 7.1 eingesetzt.

[2] Die Invariante gilt streng genommen erst, wenn von der - weiter unten beschriebenen - Tiefensuche unereichbare Knoten aus $G$ gelöscht werden. Diese Knoten und deren inzidenten Kanten leisten keinen Beitrag zum maximalen Fluss.

[3] Dabei wird stets der Abstieg über die in der Uhrzeigerordnung nächste Kante durchgeführt, ausgehend von derjenigen Kante, über die der Knoten zuletzt betreten wurde.

**Dualer Baum T$^*$**

**Es wird ein zweiter Baum $T^*$ aus dem *dualen* Graphen $G^*$ konstruiert. Er besteht aus sämtlichen Kanten, die *nicht* in $T$ sind. $T^*$ verwaltet die Kanten, die in einer Richtung saturiert sind. Er wird dazu verwendet, saturierte Kreise im dualen - und damit saturierte Cuts im primalen Graphen - zu erkennen**. Es gelten zur Laufzeit des Algorithmus folgende Invarianten:

1. $T^*$ ist ein überspannender Baum von $G^*$, dessen Darts alle zur Baumwurzel $f_\infty$ gerichtet sind.
2. Alle Darts in $T^*$ sind *nicht-saturiert*. Die dazu korrespondierenden Darts in Gegenrichtung sind alle *saturiert*.

Beide Eigenschaften können lediglich dann erfüllt sein, wenn es keine residualen Uhrzeigerkreise in $G$ gibt. Ansonsten kann es passieren, dass entweder die Kanten $\notin T$ die falsche Richtung aufweisen (und damit die Darts in $T^*$) oder die Rückwärtsdarts von Darts in $T^*$ residual sind.

### 7.3.3 Algorithmus

Die Grundidee des Algorithmus von Borradaile und Klein ist: *Augmentiere in jedem Schritt den von allen unsaturierten Pfaden am weitesten links stehenden.*

Nach der Definition von $T$ und $T^*$ **sind die Kantenmengen der beiden Bäume disjunkt** und die **Vereinigung ergibt die Kantenmenge des Graphen** $G$. Der Algorithmus tauscht in jedem Schritt Kanten (genauer Darts) zwischen $T$ und $T^*$ aus und erhält dabei die Invarianten aufrecht. Kann eine der Invarianten für $T^*$ nicht aufrecht erhalten werden, ist die Abbruchbedingung erfüllt und der maximale Fluss berechnet.
Die genauen Schritte sind im Algorithmus 7.3.1 aufgelistet. Dabei gelten folgende Bezeichnungen:

- $d := (v, w)_d$ ist nach der Pfadaugmentierung der saturierte Dart am nächsten zur Wurzel, $d^* := (f_l(d), f_r(d))_d$ seine duale Version.
- $e^* := (f_l(e), f_r(e))_d$ ist die Kante von der rechten Fläche von d zur Vaterfläche in $T^*$. Es gilt $f_r(d) = f_l(e)$. $e := (x, y)_d$ ist die primale Version in $G$.

Der Algorithmus ist in Abbildung 7.4 veranschaulicht. Die Schritte 10-13 bilden den Kern. Sie umgehen den alten Pfad über die saturierte Kante $d$. Dabei wird sichergestellt, dass der neue unsaturierte Pfad der am weitesten links stehende im Graphen ist. Es ist leicht zu sehen, dass die oben aufgestellten Invarianten bezüglich der Bäume $T$ und $T^*$ nach jedem *Umleitungsschritt* erfüllt bleiben.

**Algorithm 7.3.1** Algorithmus von Borradaile und Klein

1: Konstruiere $T$ und lösche die Knoten aus $G$, die nicht $\in T$ sind.
2: Konstruiere $T^*$ in $G^*$ aus den Vorwärtsdarts der Kanten, die $\notin T$
3: **repeat**
4: **augmentiere** den Pfad von $s$ nach $t$
5: $d := (v, w)_d$ ist der saturierte Dart am nächsten zur Wurzel $t$
6: **lösche** $d$ aus $T$
7: **if** $f_l(d)$ ist ein Nachfahre von $f_r(d)$ in $T^*$ **then**
8: **gib zurück** Fluss **f**
9: **else**
10: $e^*$ sei in $T^*$ der Verbindungsdart zum Vater von $f_r(d)$
11: **lösche** $e^*$ aus $T^*$ und **füge** $rev(d^*) = (f_r(d), f_l(d))_d$ in $T^*$ ein
12: **füge** $e$ in $T$ ein
13: **ersetze** in $T$ die Darts auf dem Pfad von Knoten $x$ zum Knoten $v$ durch ihre Gegendarts.
14: **end if**
15: **until** 0

Wenn die Abbruchbedingung eintritt, wurde ein saturierter Kreis im dualen Baum $T^*$ und damit im Graphen $G^*$ entdeckt. Ein saturierter Kreis im dualen Graphen ist aber gleich einem saturierten *Cut* im primalen Graphen (vgl. Abschnitt 6.3). In [4] wird gezeigt, dass Quelle $s$ innerhalb und Senke $t$ außerhalb des Cuts liegen müssen.

### 7.3.4 Laufzeit

Im Folgenden wird eine kurze Skizze der Laufzeitanalyse gegeben. Grundlage der Analyse ist folgendes Theorem, welches in [4] bewiesen wird:

**Theorem.** *Ist zunächst Dart $d$ Teil eines Augmentierungsschrittes und zu einem späteren Zeitpunkt der Rückwärtsdart $rev(d)$, so kann der Dart $d$ bis zum Ende des Algorithmus nicht mehr Teil eines Augmentierungsschrittes sein.*

Borradaile und Klein zeigen, dass als Folge des Theorems ein Vorwärtsdart $d$ *höchstens einmal* aus dem Suchbaum $T$ entfernt werden kann. Umgekehrt kann ein Rückwärtsdart $rev(d)$ lediglich *zweimal* entfernt werden. Insgesamt ergibt sich, dass während des Algorithmus jede Kante höchstens dreimal angefasst wird, demnach höchstens $3m$ Augmentierungsschritte ausgeführt werden. Die Verwaltung des Baumes $T$ benötigt in jedem Schritt $O(\log(n))$ Zeit. Daher beträgt die Gesamtlaufzeit $O(n \log(n))$.

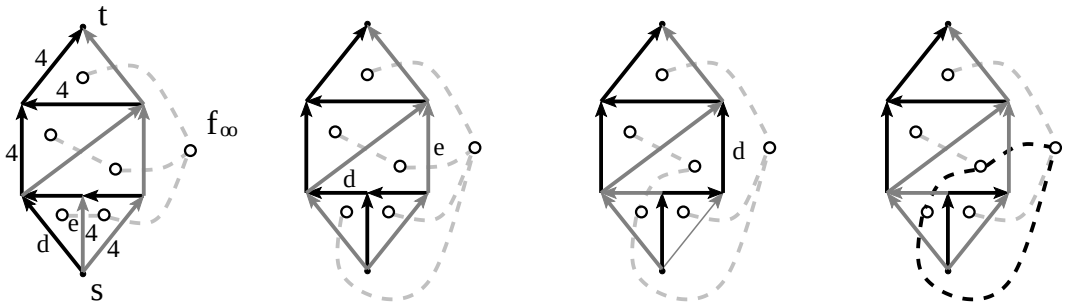



Abbildung 7.4: Drei Iterationen des Algorithmus von Borradaile und Klein auf einem Beispielgraph. Die Kanten in *Baum* $T$ sind dunkel dargestellt. Der *duale Baum* $T^*$ ist grau gestrichelt. Sämtliche Kantengewichte sind 1 bis auf die gekennzeichneten. Die Abfolge: *Links* ist die Initialisierung des Algorithmus abgebildet. $d$ und $e^*$ sind *vor* jedem Augmentierungsschritt gekennzeichnet. Jedes Bild zeigt den Status nach dem Umleitungsschritt. Sonderfälle sind in Bild 3 und Bild 4: In Bild 3 wird die Kante zwischen $e$ und $d$ in $T$ umgedreht. In Bild 4 ist die Abbruchbedingung erfüllt, da $f_l(d)$ ein Nachfolger von $f_r(d)$ in $T^*$ ist. Der Kreis in $T^*$ beschreibt dann den minimalen Cut.

### 7.3.5 Graphenvorverarbeitung

Falls der Eingabegraph $G$ keine residualen Kreise im Uhrzeigersinn enthält, kann der Algorithmus ohne weiteres ausgeführt werden. Andernfalls muss zunächst ein *Nullfluss* durch den Graphen berechnet werden. Ein Nullfluss ist ein Fluss, bei dem das Erhaltungsgesetz auch für $s$ und $t$ gilt (Der Wert des Flusses ist damit $0$). In [4] wird dazu eine Erweiterung eines Verfahrens von Khuller, Naor und Klein [18] verwendet. Im Wesentlichen läuft die Vorverarbeitung auf die Berechnung kürzester Wege von einem Startknoten zu allen anderen Knoten des Graphen hinaus.

Betrachtet wird der duale Graph von $G$ mit der Menge aller *Darts* als Kantenmenge $E$. Vom Knoten $f_\infty$ aus wird der kürzeste Weg zu jeder anderen Fläche berechnet und die Distanz in jeder Fläche gespeichert. Von jedem Dartgewicht in $G$ wird dann die Differenz der angrenzenden Flächendistanzen abgezogen. Der Ergebnisgraph ist frei von residualen Uhrzeigerkreisen. Die Berechnung der kürzesten Wege kann beispielsweise mit Dijkstras Algorithmus berechnet werden und benötigt dann $O(|V|\log(|V|))$ Zeit.

Der Shape Matching Graph aus Kapitel 6 ist nicht frei von residualen Kreisen im Uhrzeigersinn. Daher kann der Vorverarbeitungsschritt nicht ausgespart werden. Allerdings zeigten Experimente, dass sich die Berechnung der kürzesten Wege schneller mittels eines Dynamic Time Warping Algorithmus mit Ursprung in $f_\infty$ durchführen lässt. Da der duale Graph jedoch Kreise enthält, muss der Algorithmus

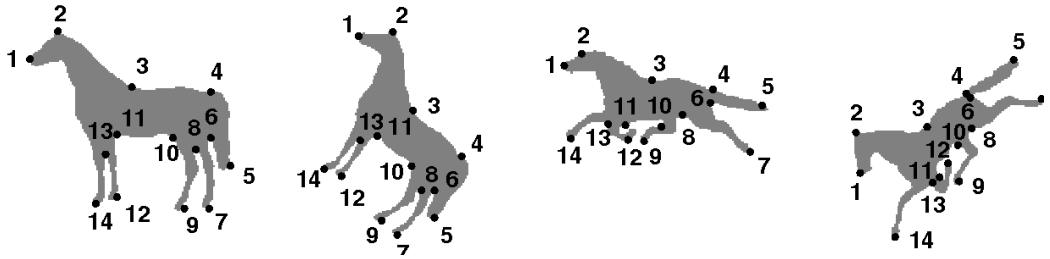


Abbildung 7.5: Vergleich eines Pferdes (links) mit drei anderen Pferden über Graph Cuts.

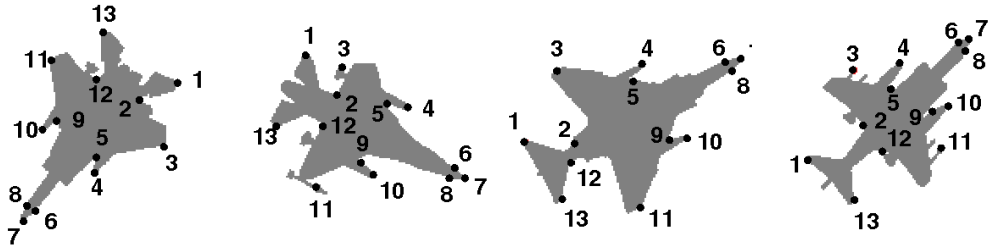


Abbildung 7.6: Vergleich eines Fliegers (links) mit anderen Fliegern verschiedenen Typs über Graph Cuts.

meist zweifach angewandt werden; theoretisch können $O(n)$ Iterationen nötig sein. Die obere Laufzeitschranke ist daher quadratisch in der Anzahl der Knoten im dualen Graphen. Mehr als drei Iterationen waren in der Praxis jedoch nie notwendig.

### 7.3.6 Ergebnisse

Abbildungen 7.5 und 7.6 zeigen weitere Formenvergleiche; diesmal geschieht die Minimierung über dem Algorithmus von Borradaile und Klein. Die Punktkorrespondenzen unterscheiden sich nicht von den Resultaten über dynamische Programmierung oder denen der Minimierung über den Algorithmus von Boykov und Kolmogorov. Die Distanzen der Pferde betragen $0.0579$, $0.4245$ und $0.9778$, die der Flieger $0.2567$, $0.6989$ und $0.7973$.
Die Laufzeiten sind in Abbildungen 7.7 und 7.8 zu sehen. Um einen Vergleich zu haben wurden dieselben zwei Formenpaare aus den Ergebnissen zu Abschnitt 7.2 verwendet. Die Abtastdichten wurden wiederum zwischen $50$ und $500$ Punkten gewählt. Der Vergleich der sehr **unterschiedlich beschaffenen Formen** - Fledermaus und Käfer - beginnt bei einer Laufzeit von $0.01$ Sekunden bei $50$ Punkten und endet mit $2.11$ Sekunden bei $500$ Konturpunkten. Der Laufzeitexponent wurde durch lineare Regression mit $2.1$ bestimmt.
Der Vergleich **sehr ähnlicher Formen** bei $500$ Punkten ist gegenüber dem unähnlichen Fall mit $1.01$ Sekunden nur doppelt so schnell (man vergleiche dies zum Faktor

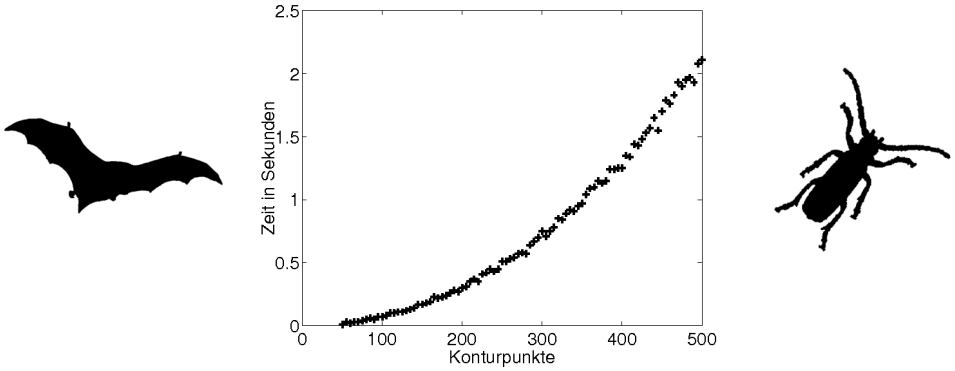



Abbildung 7.7: Laufzeiten der Vergleiche von Fledermaus und Käfer über den Algorithmus von Borradaile und Klein bei $50$ bis $500$ Punkten entlang der Kontur.

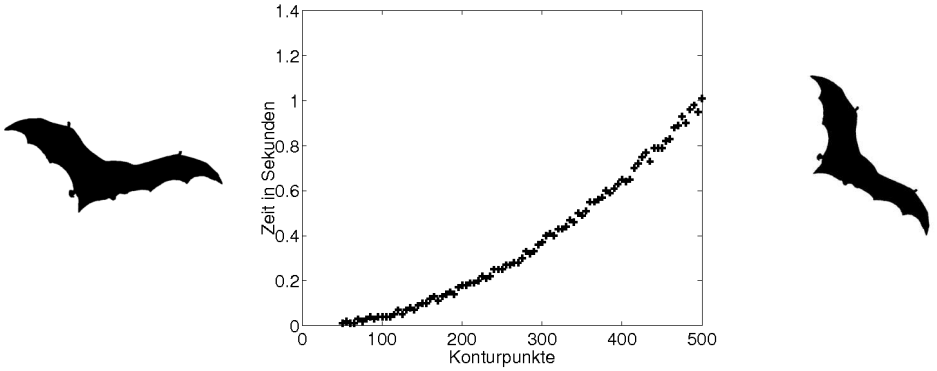



Abbildung 7.8: Laufzeiten der Vergleiche zweier Fledermäuse über den Algorithmus von Borradaile und Klein bei $50$ bis $500$ Punkten entlang der Kontur.

$10$ von Abschnitt 7.2). Noch entscheidender ist allerdings, dass der Laufzeitexponent mit $1.99$ fast gleich ist. Lediglich die Laufzeitkonstante ändert sich. Ebenfalls fällt im Vergleich auf, dass die Streuung der Laufzeiten mit zunehmender Punktzahl wesentlich leichter ausfällt als im Algorithmus von Boykov und Kolmogorov.

Die Ergebnisse machen in Hinblick auf die Algorithmusanalyse durchaus Sinn. Die systematische Abarbeitung der Augmentierungspfade stabilisiert die Laufzeit des Shape Matchings im Vergleich zur Minmierung über Boykov und Kolmogorovs Algorithmus. Die Sensitivität der Laufzeit in Bezug auf die Formenähnlichkeit wird dadurch stark gemindert, ist jedoch im Gegensatz zum Shape Matching über dynamische Programmierung messbar.

Abbildung 7.9 fasst die Laufzeitergebnisse noch einmal zusammen und vergleicht sie. Deutlich zu sehen ist die starke Unterlegenheit des Algorithmus von Boykov und Kolmogorov bei unähnlichen Formen (links). Insbesondere im Graphen rechts

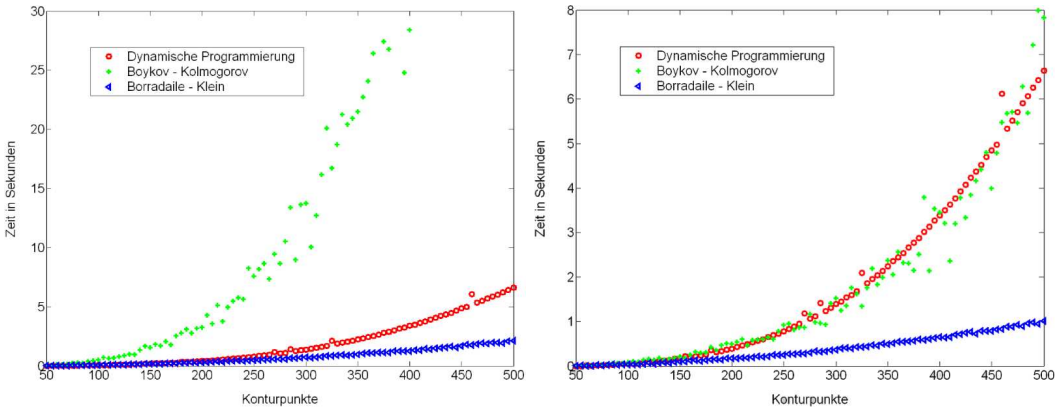



Abbildung 7.9: *links*: Gesamtvergleich der Laufzeiten von Shape Matching über dynamische Programmierung und den beiden Max-Flow Algorithmen bei unähnlichen Formen. *rechts*: Dasselbe für den Vergleich ähnlicher Formen.

ist jedoch gleichzeitig die Verbesserung der Laufzeitkomplexität vom Borradaile und Klein Algorithmus gegenüber dynamischer Programmierung zu erkennen. Die Unvorhersehbarkeit im Unterschied von Laufzeiten selbst benachbarter Abtastraten ist bei Boykov und Kolmogorov ebenfalls am höchsten.
Beide Graph Cut Experimente zusammen - das über den Algorithmus von Boykov und Kolmogorov sowie das über den Algorithmus von Borradaile und Klein - zeigen einen weiteren Vorteil gegenüber vergleichbaren Verfahren mittels dynamischer Programmierung auf: die Laufzeit des Formenvergleichs hängt ganz entscheidend vom unterliegenden Verfahren zur Berechnung des maximalen Flusses ab. Dies macht Shape Matching über Graph Cuts zukunftsträchtig, indem auf effizientere Verfahren zur Berechnung des maximalen Flusses gehofft werden kann, die optimal auf den konstruierten Graphen zugeschnitten sind.

# Kapitel 8

# Zusammenfassung

In dieser Arbeit wurde das Shape Matching Problem über ein Graph Cut Verfahren gelöst. Es wurde ein Vergleichsmodell über dynamische Programmierung hergeleitet, das korrespondierende Punkte auf zwei Formen berechnet, indem kürzeste Wege auf einem Graphen gesucht werden. Dieser Graph wurde anschließend so transformiert, dass ein planares Flussnetzwerk entstand. Es wurde gezeigt, dass die Kanten des minimalen Cuts des Graphen die Punktkorrespondenzen kodieren und dass diese identisch zu denen sind, die durch die Kürzeste-Wege-Formulierung gefunden werden. Darüber hinaus gibt der Wert des maximalen Flusses den Abstand der Formen zueinander an.

Den zweiten Schwerpunkt der Arbeit bildete das Effizienzverhalten verschiedener Max-Flow / Min-Cut Verfahren auf dem konstruierten Graphen. Dabei wurde der Algorithmus von Boykov und Kolmogorov getestet mit dem Ergebnis, dass die Laufzeit zwar einerseits sehr stark von der Ähnlichkeit der verglichenen Formen abhing, im Mittel jedoch der Graph Cut Ansatz langsamer war als die Korrespondenzberechnung über dynamische Programmierung. Das Ergebnis erstaunt insofern nicht, als dass in Abschnitt 6.1 bereits auf die Verschiedenheit der Shape Matching Problematik von typischen Graph Cut Problemen in der Bildverarbeitung - wie sie Boykov et. al. in [6] oder [5] untersuchen - hingewiesen wurde.

Daher wurden Max-Flow Verfahren speziell für planare Graphen betrachtet, um von deren geringer Laufzeitkomplexität zu profitieren. Der Algorithmus von Borradaile und Klein für planare Graphen brachte trotz zusätzlichem Vorverarbeitungsschritt eine wesentliche Laufzeitverbesserung und zeigte deutlich die Überlegenheit gegenüber dem Verfahren mittels dynamischer Programmierung auf. Es sei allerdings darauf hingewiesen, dass F. Schmidt und D. Farin 2007 [27] ein noch effizienteres Shape Matching Verfahren veröffentlichten. Der relativ hohe Verwaltungsaufwand und Speicherplatz des Graph Cut Ansatzes kann sicher nicht mit hochoptimierten Algorithmen mithalten, die lediglich kürzeste Wege auf einem regulären

Gitter suchen. Allerdings wird das Problem der Initialkorrespondenz in [27] wieder nur durch eine effiziente Suchstrategie gelöst.

Der Vorteil der Graph Cut Formulierung liegt nämlich vor allem in zwei Punkten: Zum einen wird keine wertvolle Zeit für die Suche nach der Initalkorrespondenz vergeudet. Zum anderen ist die Laufzeitkomplexität des Shape Matching Verfahrens nicht mehr so stark vom Verfahren selbst, sondern vielmehr vom verwendeten Max-Flow / Min-Cut-Algorithmus abhängig. Außerdem kann für bestimmte Anwendungsbereiche die Tatsache genutzt werden, dass der Vergleich ähnlicher Formen weniger Zeit benötigt, als der von sehr unterschiedlichen Formen. Dies ist zum Beispiel beim Tracking nützlich, bei dem sich die verfolgten Konturen in benachbarten Bildern von Bildsequenzen nur minimal verändern.

Verbesserungspotential des Shape Matching Ansatzes über Graph Cuts steckt vor allem in der Berechnung des maximalen Flusses bzw. minimalen Cuts. Eine weitergehende Analyse der Eigenschaften des Graphen lassen auf Algorithmen oder Datenstrukturen hoffen, die noch besser auf den Ansatz zugeschnitten sind. Auch der Vorverarbeitungsschritt, der für die Korrektheit des Algorithmus von Borradaile und Klein notwendig ist, treibt die Gesamtlaufzeit des Shape Matching Verfahrens nicht unerheblich in die Höhe.

# Literaturverzeichnis


[1] Basri, R., Costa, L., Geiger, D., Jacobs, D.: Determining the Similarity of Deformable Shapes. Vision Research 38, p.2365-2385, 1998

[2] Bellman, R., Dreyfus, S.E.: Applied Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1962

[3] Belongie, S., Malik, J., Puzicha, J.: Shape Matching and Object Recognition Using Shape Context. IEEE Trans. on PAMI, 24(24), p.509-522, 2002

[4] Borradaile, G., Klein, P.N.: An $O(n \log(n))$ Algorithm for Maximum $(s,t)$-Flow in a Directed Planar Graph. Proceedings, 17th ACM.SIAM Symposium on Discrete Algorithms, p.524-533, 2006

[5] Boykov, Y.; Veksler, O.; Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. on PAMI, 23(11), p.1222-1239, 2001

[6] Boykov, Y., Kolmogorov, V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE Trans. on PAMI. 26(0), p.1124-1137, 2004

[7] Cohen, I., Ayache, N., Sulger, P.: Tracking Points on Deformable Objects Using Curvature Information. Proc. of the Europ. Conf. on Comp. Vision, p.458-466, 1992

[8] Dinic, E.A.: Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. Soviet Math. Dokl., 11, p.1277-1280, 1970

[9] Dryden, I.L., Mardia, K.V.: Statistical Shape Analysis. Wiley, Chichester, 1998

[10] Edmonds, J., Karp, R.M.: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. Journal of the ACM 19(2), p.248-264, 1972

[11] Ford Jr., L.R., Fulkerson, D.R.: Maximal Flow Through a Network. Canadian Journal of Mathematics 8, p.399-404, 1956

[12] Gdalyahu, Y., Weinshall, D.: Flexible Syntactic Matching of Curves and its Application to Automatic Hierachical Classification of Silhouettes. IEEE Trans. on PAMI 21(12), p.1312-1328, 1999

[13] Geiger, D., Gupta, A., Costa, L.A., Vlontzos, J.: Dynamic Programming for Detecting, Tracking and Matching Deformable Contours. IEEE Trans. on PAMI 13(3), p.294-302, 1995

[14] Gorman, J.W., Mitchell, O.R., Kuhl, F.P.: Partial Shape Recognition Using Dynamic Programming. IEEE Trans. on PAMI 10(2), p.257-266, 1988

[15] Greig, D.M., Porteous, B.T., Seheult, A.H.: Exact Maximum A Posteriori Estimation for Binary Images. J. Roy. Statist. Soc. Ser. B. 51(2), p.271-279, 1989

[16] Huttenlocher, D.P., Noh, J.J., Rucklidge, W.J.: Proceedings. Fourth Int. Conf. on Computer Vision, p.93-101, 1993

[17] Ling, H., Jacobs, D.W.: Shape Classification Using the Inner-Distance. IEEE Trans. on PAMI 29(2), p.286-299, 2007

[18] Khuller, S., Naor, J., Klein, P.: The Lattice Structure of Flow in Planar Graphs. SIAM Journal on Discrete Mathematics, 6(3), p.477-490, 1993

[19] Kolmogorov, V., Zabih, R.: What Energy Functions Can Be Minimized via Graph Cuts?, IEEE Trans. on PAMI, 26(2), p.147-159, 2004

[20] Manay, S., Cremers, D., Hong, B.-W., Yezzi, A., Soatto, S.: Integral Invariants for Shape Matching. IEEE Trans. on PAMI 28(10), p.1602-1618, 2006

[21] McConnell, R., Kwok, R., Curlander, J.C., Kober, W., Pang, S.S.: $\psi - s$ Correlation and Dynamic Time Warping: Two Methods for Tracking Ice Floes in SAR Images. IEEE Trans.on Geosc.and Rem.Sens. 29, p.1004-1012, 1991

[22] Sankoff, D., Kuskal, J.B.(Hrsg.): Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley, Massachusetts, 1983

[23] Sebastian, D., Klein, P., Kimia, B.: On Aligning Curves. IEEE Trans. on PAMI 25(1), p.116-125, 2003

[24] Sharvit, D., Chan, J., Tek, H., Kimia, B.: Symmetry-based Indexing of Image Databases. Proceedings of the IEEE Workshop on Content-Based Access of Image and Video Libraries, p.56, 1998

[25] Sleator, D.D., Tarjan, R.E.: A Data Structure for Dynamic Trees. JCSS, p.362-391, 1981

[26] Schmidt, F.R., Töppe, E., Cremers, D., Boykov, Y.: Efficient Shape Matching via Graph Cuts. Int. Conf. on Energy Minimization Methods for Computer Vision and Pattern Recognition, LNCS, Ezhou, China, Springer, 2007

[27] Schmidt, F.R., Farin, D., Cremers, D.: Fast Matching of Planar Shapes in Sub-cubic Runtime. IEEE International Conference on Computer Vision (ICCV), 2007

[28] Umeyama, S.: Parameterized Point Pattern Matching and its Application to Recognition of Object Families. IEEE Trans. on PAMI 15(2), p.136-144, 1993

[29] Weihe, K.: Maximum $(s, t)$-Flows in Planar Networks in $O(|V| \log |V|)$-Time. J. Comput. Syst. Sci. 55(3), p.454-476, 1997

[30] Whitney, H.: Congruent Graphs and the Connectivity of Graphs, Amer.J.Math. 54, p.150-168, 1932

[31] Zhu, S.C., Yuille, A.L.: FORMS: A Flexible Object Recognition and Modelling System. Proceedings., Fifth International Conference on Computer Vision 1995, p.465-472, 1995

# Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.