Fachbereich Informatik,
Mathematik und Naturwissenschaften

Bachelorarbeit

Vergleich und Anwendungen von Skelettierungsalgorithmen in der digitalen Bildverarbeitung

Christoph Bullmann
Leipzig, Oktober 2008

Betreuender Professor: Prof.Dr.rer.nat.habil. Karl-Udo Jahn

Diese Arbeit wurde mit LaTeX gesetzt.

Inhaltsverzeichnis

1 Einleitung

Die digitale Bildverarbeitung als Teilgebiet der Informatik nimmt in der modernen Informationsgesellschaft zunehmend einen höheren Stellenwert ein. Speziell die Bildanalyse und die Bildinterpretation sind wesentliche Bestandteile vieler Bildverarbeitungs-Anwendungen. Als bekannte Beispiele sind hier die Fingerabdrucks- und Gesichtserkennung zu nennen. Auch in der Industrie ist die computergestützte Auswertung von Produktionsabläufen nicht mehr weg zu denken. In vielen Fällen kommen Rechner in der vollautomatisierten Qualitätskontrolle zum Einsatz. Hier werden beispielsweise Kameras an Fließbändern montiert, um fehlerhafte Produkte automatisch zu erfassen und gegebenenfalls auszusortieren. Dabei ist diese Art der Qualitätskontrolle oftmals nicht nur exakter, schneller und preisgünstiger, auch können solche Systeme Informationen mit einbeziehen, die dem Menschen naturgemäß nicht zur Verfügung stehen, beispielweise Infrarotaufnahmen, Aufnahmen mikroskopisch kleiner Strukturen oder Ultraschallbilder. Dadurch werden in einigen Fällen Qualitätskontrollen in dem erforderlichen Umfang überhaupt erst möglich gemacht. Auch in der Medizintechnik spielt die digitale Bildverarbeitung vor allem bei der Aufbereitung von Informationen eine unverzichtbare Rolle. Dabei besteht ein wesentlicher Teil der Aufgaben aus der Strukturierung, Kategorisierung und dem Verfügbarmachen von relevanten Informationen aus der bildgebenden Diagnostik. Zu nennen sind hier Ultraschallmessung, MRT- oder CT-Scans.

All diese Anwendungen haben die Gemeinsamkeit, mit sehr großen Datenmengen zu arbeiten. Bei der heutigen rasant fortschreitenden Informationstechnik werden zudem ständig neue Verfahren und Technologien entwickelt, welche immer größere, hochauflösendere - kurz: mehr Informationen umfassende Datenmengen liefern. Umso mehr wächst die Notwendigkeit von Abstraktion und Vereinfachung dieser Daten, sei es, um die Informationen dem Menschen in einem "erträglichen" Maße zugänglich zu machen oder um durch grundlegende Erkenntnisse erste Kategorisierungen durchführen zu können, die weitere Untersuchungen ermöglichen. Mit diesem Trend, immer umfassendere Informationen zu verarbeiten, wächst ebenfalls der Anspruch an die zugrunde liegenden Algorithmen. Speziell für zeitkritische Anwendungen, wie in Echtzeitsystemen, müssen oftmals die Daten in kürzester Zeit auf das Wesentliche reduziert werden.

Dem Entwickler steht dazu eine Reihe von Werkzeugen zur Verfügung. Dazu gehören unter anderem grundlegende Methoden, wie die Kantendetektion, Linearisierung, Vektorisierung oder Binarisierung von Bildern. Ein weiteres nützliches Tool ist die *Skelettberechnung*. Dabei werden

für Objekte in Binärbildern skelettartige, mittig-liegende Strukturen ermittelt, welche, je nach verwendetem Algorithmus, unterschiedliche Eigenschaften der Objekte widerspiegeln, insbesondere topologische und geometrische Eigenschaften.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist der Vergleich und die Anwendung verschiedener Skelettierungsalgorithmen. Dabei sollen nicht nur deren Resultate hinsichtlich ihrer Eigenschaften untersucht werden, auch sollen generelle Stärken und Schwächen der Algorithmen selbst analysiert werden. Um angemessen vergleichen zu können, müssen die untersuchten Algorithmen selbstverständlich implementiert werden. Dazu soll ein leicht zu bedienendes Softwaremodul erstellt werden, welches es dem Anwender ermöglicht, Skelettierungsalgorithmen auszuprobieren und gegebenenfalls die hier präsentierten Resultate nachzuvollziehen. Ein weiteres Ziel dieser Arbeit ist die Erstellung von Pseudocodes. Sie sollen helfen, die Algorithmen und Definitionen besser zu verstehen und den Leser in die Lage versetzen, die Verfahren schnell für eigene Zwecke zu implementieren. Zudem sollen, soweit möglich, Hinweise auf Verbesserungsmöglichkeiten gegeben werden. Letztlich soll diese Arbeit dem Leser ein fundiertes, umfangreiches Bild über Skelettierungsverfahren vermitteln, welches ihm die Auswahl eines geeigneten Algorithmus für seine Anwendungen erleichtert.

1.2 Anforderungen

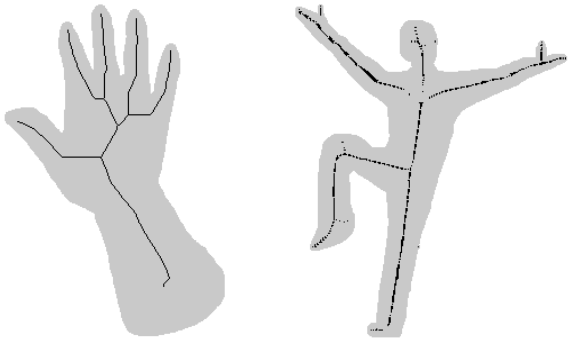
Eine Hauptanforderung dieser Arbeit besteht in der umfangreichen Literaturrecherche. Standardwerke, wie Gonzalez/Woods [20], Burger/Burge [4], Russ [48] oder Tönnies [54], behandeln nur einige wenige Standard-Skelettierungsalgorithmen und so ist eine ausgedehnte Recherche von Fachartikeln notwendig.

Eine zweite Anforderung stellt die Erstellung von geeignetem Bildmaterial dar. Voraussetzung dafür ist eine funktionierende Software, mit welcher sich die verschiedenen Skelette berechnen lassen. Diese wurde im Rahmen dieser Arbeit in Form eines ImageJ-Plugins erstellt. Eine zweite Voraussetzung ist eine Sammlung von Binärbildern, welche viele verschiedenartige Objekte umfasst, um möglichst vielseitige und aussagekräftige Resultate zu erhalten. Auch eine solche Sammlung wurde für diese Arbeit angelegt.

1.3 Bebilderung der Arbeit

Fast alle in dieser Arbeit angegebenen Resultate von Skelettierungsverfahren können mit Hilfe des Plugins und der Bildersammlung nachvollzogen werden. Beides wurde dieser Arbeit auf einem

Datenträger beigelegt oder kann aus dem Internet herunter geladen werden (siehe Abschnitt 6.2). Die Bildunterschriften enthalten dazu einen Hinweis auf die verwendeten Algorithmen und Bilder. So kann Bild (a) aus Abbildung 1.1 produziert werden, indem der **Hilditch-Skelett**-Algorithmus des Plugins auf das Bild **12_hand.png** angewendet wird. Ebenso erhält man Bild (b) durch die Anwendung des Algorithmus für das **morphologische Skelett** des Plugins auf Bild **23_mensch.png**. In Abschnitt 6.2 sind zusätzlich Installationshinweise für die Benutzung des Plugins zu finden.



(a) (b)

Abbildung 1.1: Nicht alle Skelettierungsverfahren produzieren ein zusammenhängendes Skelett. In (a) wurde das Objekt mit dem Hilditch-Algorithmus ausgedünnt [BILD 12]. Die Zusammengehörigkeit wurde erhalten. (b) zeigt ein nicht-zusammenhängendes, morphologisches Skelett [BILD 23].

Einige Bilder dieser Arbeit wurden lediglich zu Erklärungszwecken erstellt, etwa um die Arbeitsweise von Algorithmen oder deren Zwischenergebnisse zu demonstrieren. Diese sind mit Hilfe des Plugins nicht reproduzierbar. Deren Bildunterschriften enthalten deshalb keinen Hinweis auf eine Bilddatei auf dem Datenträger. Einige wenige Abbildungen wurden auch aus anderen Publikationen (leicht bearbeitet) übernommen. Für diese Bilder ist in der Bildunterschrift ein Verweis auf die Quelle zu finden.

2 Skelette in der digitalen Bildverarbeitung

2.1 Begriff des Skelettes

Aus biologisch-medizinischer Sicht ist das Skelett, zum Beispiel eines Menschen, eindeutig definiert. Es ist ein Körperbestandteil, der die Stützstruktur des Organismus bildet. Es ist sozusagen das Grundgerüst - das Wesentliche, auf das es, zum Beispiel auch in der Höhlenmalerei, reduziert wird (Abbildung 2.1 (a)).

In der Technik findet der Begriff des Skelettes ebenso Verwendung. So wurde der Pariser Eiffelturm in der so genannten *Skelettbauweise* errichtet. Auch hier wird das Skelett mit einer primär tragenden/stützenden Funktion assoziiert (Abbildung 2.1 (b)).

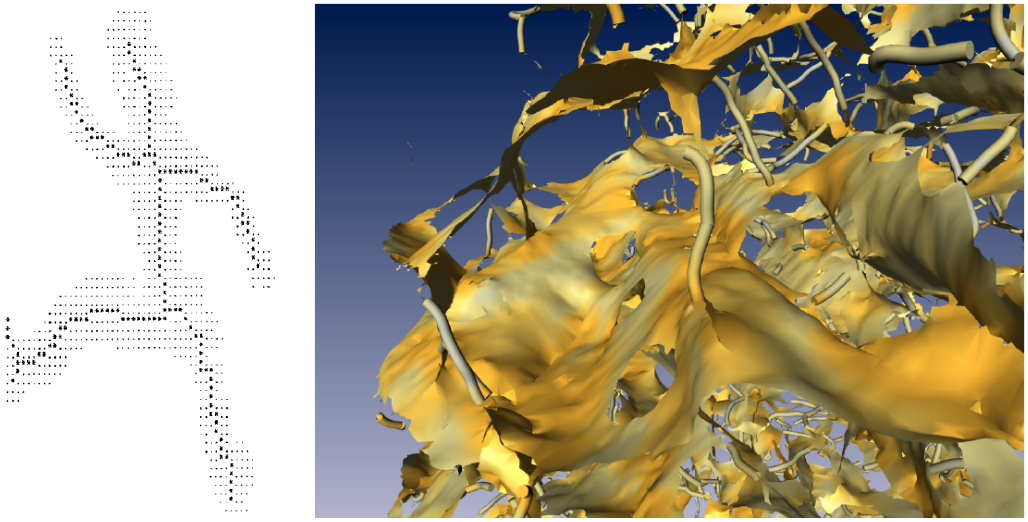
(a) (b)

Abbildung 2.1: Skelette verkörpern nicht nur in der digitalen Bildverarbeitung etwas Wesentliches der Objekte. (a) zeigt eine Malerei von Kurt Grosjean mit menschlichen Skeletten. Der Pariser Eiffelturm wurde in der Skelettbauweise errichtet (b).

Das Skelett ist also ein Teil des Ganzen und stellt etwas Wesentliches dar - das Grundgerüst. Es hat außerdem die Eigenschaft, dass man mit Hilfe des Skelettes Rückschlüsse auf das eigentliche Objekt ziehen kann. So sind beispielsweise nahezu alle Eigenschaften, wie Aussehen, Fortbewe-

gung oder Ernährungsgewohnheiten, der Dinosaurier auf Knochenfunde - also auf das Skelett - zurückzuführen.

Auch in der digitalen Bildverarbeitung versteht man unter einem Skelett prinzipiell ein Teil des Ganzen, etwas Wesentliches - ein Grundgerüst, welches Rückschlüsse auf bestimmte Eigenschaften des Objektes zulässt.



(a) (b)

Abbildung 2.2: Skelette in der digitalen Bildverarbeitung: (a) zeigt ein bekanntes Bild (2D-Skelett) aus der Publikation von Zhang/Suen [58]. In (b) ist ein 3D-Skelett eines CT-Scans zu sehen [43].

Jedoch ist der Begriff des Skelettes in der digitalen Bildverarbeitung längst nicht eindeutig definiert. Man findet in der Literatur zu diesem Thema eine Vielzahl von meist schwammig definierten Begriffen, die oftmals synonym zueinander verwendet werden. Auch haben sich Bedeutungen von Begriffen im Laufe der Zeit verändert.

Die wahrscheinlich erstmalige Verwendung des Begriffes *Skelett* im Sinne der heutigen digitalen Bildverarbeitung war durch Johann Benedict Listing in seiner 1862 veröffentlichten Abhandlung "Der Census räumlicher Complexe oder Verallgemeinerung des Euler'schen Satzes von den Polyedern" in dem Listing das *lineare Skelett* wie folgt beschreibt:

> Ertheilen wir nun der Grenze L an allen ihren Theilen eine stetige Veränderung in der Weise, dass dieselbe im Allgemeinen ohne Verletzung der ihr anfänglich zukommenden Zusammenhänge durch unendlich kleine, gleiche oder ungleiche Schritte immer tiefer ins Innere von K rücke, so lange bis durch diese Art der Verschmälerung, Verdünnung oder Zusammenschnürung K auf einen Complex bloss von Linien und Punkten reduziert ist, so geht aus K im Allgemeinen ein Complex hervor, welcher

> nur aus Constituten der beiden ersten Curien besteht, und welcher gleichsam als das **lineare Skelett** des gegebenen Constituten betrachtet weden kann. [33]

Diese Definition von Listing beschreibt ein Verfahren, welches relativ konkret dem Vorgehen des so genannten *Ausdünnens (Thinning)* entspricht, wobei an einem Objekt sukzessive von außen her der Rand "abgetragen" wird, sodass am Ende nur noch ein dünnes Gerüst stehen bleibt. Das verbleibende Gerüst wird dann als das *Skelett* des Objektes bezeichnet.

Tatsächlich wird in der Literatur der Begriff des Ausdünnens oftmals synonym mit dem Begriff des *Skelettierens* verwendet. Dies ist nicht verwunderlich, da sich ein deutlicher Großteil der Forschungen zu Skeletten in den letzten 40 Jahren mit eben den Ausdünnungsverfahren beschäftigt hat. Auch behandeln Standardwerke, wie von Gonzalez/Woods [20], Burger/Burge [4] oder Russ [48], fast nur Ausdünnungs-Algorithmen.

Jedoch existieren heutzutage weitaus mehr Verfahren, deren Resultate als Skelette bezeichnet werden und so kann eine Definition des Skelett-Begriffes nicht auf ein Verfahren beschränkt werden. Gisela Klette formulierte in ihrer Doktorarbeit eine Definition für eine *skelettale Kurve (skeletal curve)*, wobei sie sich teilweise auch auf Listings Definition stützt:

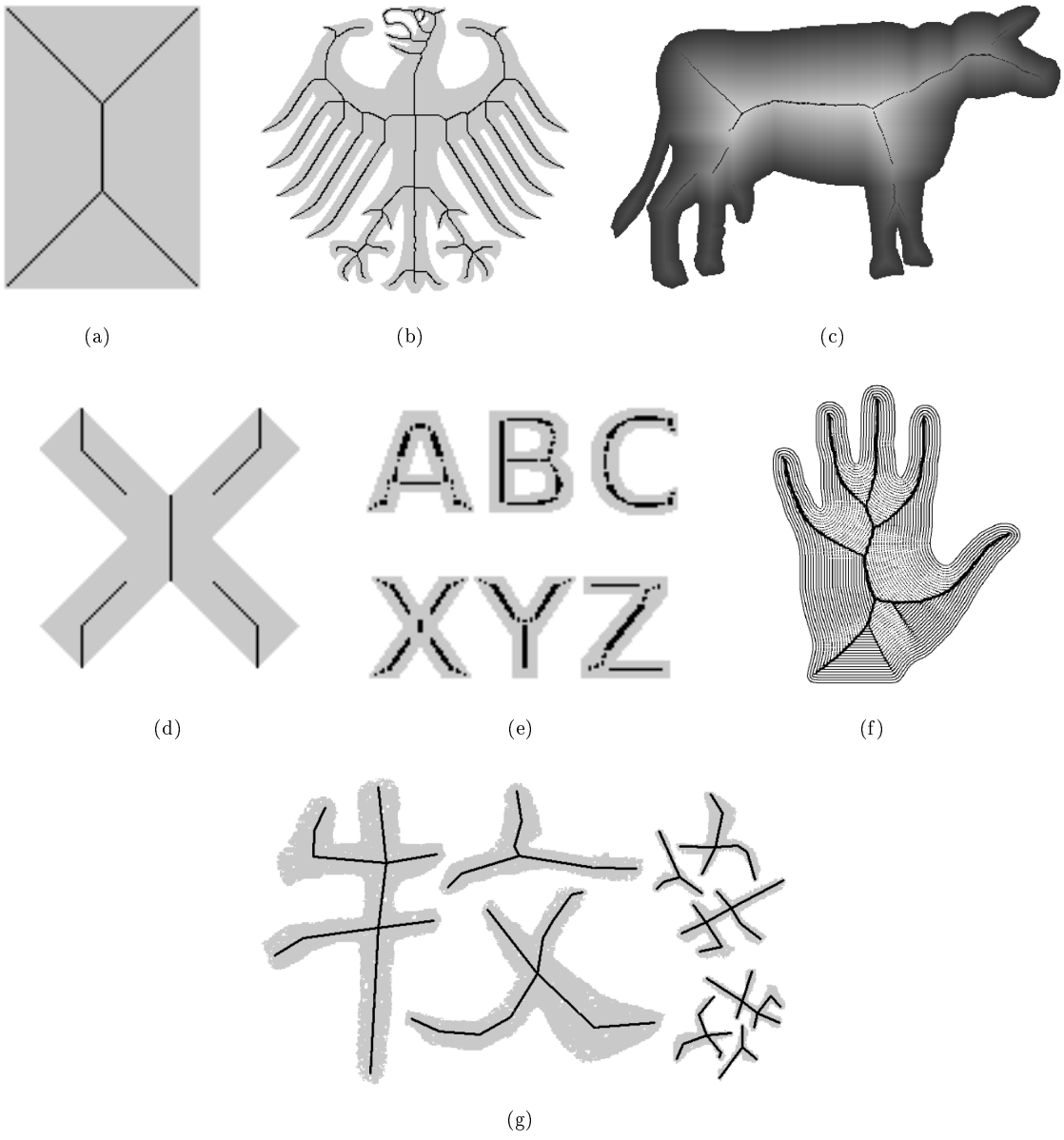
> A skeletal curve in the continuous space is either a (not uniquely defined) linear skeleton or a (uniquely defined) medial axis, aiming at topologic or geometric studies. [27]

Eine noch allgemeinere Zusammenfassung lieferten Lam/Lee/Suen:

> The term "skeleton" has been used in general to denote a representation of a pattern by a collection of thin (or nearly thin) arcs and curves. [29]

Diese beiden, noch immer nicht alle Skelettarten erfassenden, Definitionen zeigen, wie schwierig es ist, einfache Gemeinsamkeiten für alle Arten von Skeletten zu finden (siehe Abbildung 2.3).

Während Gisela Klette in ihrer Definition zusätzlich zum linearen Skelett die *Mediale Achse* einbezieht (eine der wenigen mathematisch exakt definierten Begriffe; siehe Abschnitt 3.1), versuchen Lam/Lee/Suen das Skelett nicht an Verfahren festzumachen, sondern benennen ein wichtiges Ziel - die Wiedergabe/Repräsentation der Objekte/Muster mit Hilfe von dünnen Bögen und Kurven, und Klette ergänzt: zur Untersuchung von topologischen und geometrischen Eigenschaften (siehe Abschnitt 2.3).



(a) (b) (c)

(d) (e) (f)

(g)

Abbildung 2.3: In der Bildverarbeitung gibt es eine Vielzahl verschiedener Definitionen eines Skelettes bzw. Verfahren sie zu berechnen. (a) zeigt die *Mediale Achse* eines Rechtecks [BILD 27], (b) ein Skelett nach dem *Ausdünnen (Thinning)* [BILD 5], (c) ein *Distanz-Skelett* [BILD 20], (d) das Ergebnis eines *Kritische-Punkte-Algorithmus* [BILD 17], (e) die *morphologische Skelettberechnung* [BILD 2], (f) ein *Hamilton-Jacobi* Skelett [50] und (g) ein Skelett, erstellt mittels *Triangulations*-Technik [62].

Der Begriff des Skelettes kann nicht einfach in eine mathematisch zufriedenstellende Definition gebracht werden, da jedes Verfahren, welches als Resultat ein "Skelett" liefert, den Begriff über die jeweiligen Eigenschaften neu definiert. Während zum Beispiel beim Ausdünnen das Hauptaugenmerk auf dem Erhalt der topologischen Struktur liegt, versucht man bei der Skelettberechnung mittels *Triangulation* geometrische Merkmale zu erfassen.

2.2 Kategorisierung von Skelettierungsverfahren

So erscheint es sinnvoll, Skelettierungsalgorithmen nach ihren Vorgehensweisen zu gruppieren. Eine mögliche Einteilung ist eine Unterscheidung in iterative und nicht-iterative Algorithmen, eine andere in pixelbasierte und nicht-pixelbasierte Verfahren. In dieser Arbeit findet eine Einteilung in folgende Kategorien statt:

- **Algorithmen, basierend auf *Distanz-Transformationen (distance transform)***

  Diese Algorithmen nutzen Distanzberechnungen, um Skelettpunkte zu finden (Objektpixel mit den höchsten Abständen zu Hintergrundpixeln).

  Beispiele dieser Kategorie:

  Mediale Achse

  Distanz-Skelett

- **Algorithmen, basierend auf *kritischen Punkten (critical points)***

  Diese Algorithmen versuchen kritische/wichtige Punkte in einem Objekt zu finden, um diese dann über einen Pfad zu verbinden.

  Beispiele dieser Kategorie:

  einfacher Kritische-Punkte-Ansatz

  Triangulations-Technik

- **Ausdünnungs-Algorithmen (Thinning)**

  Diese Algorithmen löschen iterativ simple Konturpixel, um die Objekte so auf Ein-Pixel-Breite auszudünnen.

  Beispiele dieser Kategorie:

  morphologisches Ausdünnen

  sequentielles Ausdünnen

  paralleles Ausdünnen

In diese drei genannten Kategorien lassen sich die meisten in der Literatur bekannten Verfahren einordnen, jedoch gibt es noch vereinzelte, sehr spezielle Verfahren, wie zum Beispiel das *Hamilton-Jacobi*-Skelett [50], welche nur schwer einer Kategorie zuzuordnen sind. Diese sind jedoch nicht Gegenstand dieser Arbeit.

2.3 Anwendungen von Skeletten

Skelettierungen besitzen in der digitalen Bildverarbeitung ein breites Feld von Einsatzmöglichkeiten. Wie bereits im vorhergehenden Abschnitt erwähnt, liegt das Augenmerk hauptsächlich auf topologischen und geometrischen Eigenschaften von Skeletten.

Topologische Eigenschaften sind Merkmale, die sich nicht explizit auf die konkrete Form einer Region beziehen, sondern auf ihre strukturellen Eigenschaften, die auch bei starken Verformungen erhalten bleiben. Als Region wird ein Ausschnitt eines Bildes bzw. ein Teilbild bezeichnet, auf das sich die Untersuchungen beziehen (*ROI - region of interest*). Ein bekanntes topologisches Merkmal ist die so genannte *Euler-Zahl* N_E einer Region $\mathcal{R}$. Sie ist die Differenz aus der Anzahl N_R der zusammenhängenden Regionen und der Anzahl N_L ihrer Löcher, d.h.

$$N_E(\mathcal{R}) := N_R(\mathcal{R}) - N_L(\mathcal{R}) \tag{2.1}$$

Beispielsweise ist der Buchstabe "**B**" eine zusammenhängende Region, also ist $N_R = 1$, und hat 2 Löcher, d.h. $N_L = 2$. Damit ist seine Euler-Zahl $N_E = 1 - 2 = -1$. Der Buchstabe "**Ö**" hat hingegen eine Euler-Zahl von $N_E = 3 - 1 = 2$ [4].

Vor allem Skelette einer Region die mittels Ausdünnung erstellt wurden, haben dieselbe topologische Struktur wie die ursprüngliche Region. Das bedeutet, dass eine topologische Analyse, beispielsweise zur Zeichenerkennung (OCR), auch auf dem Skelett der Region durchgeführt werden kann, welches in der Regel deutlich weniger Pixel umfasst und die Analysen damit weniger rechenintensiv sind.

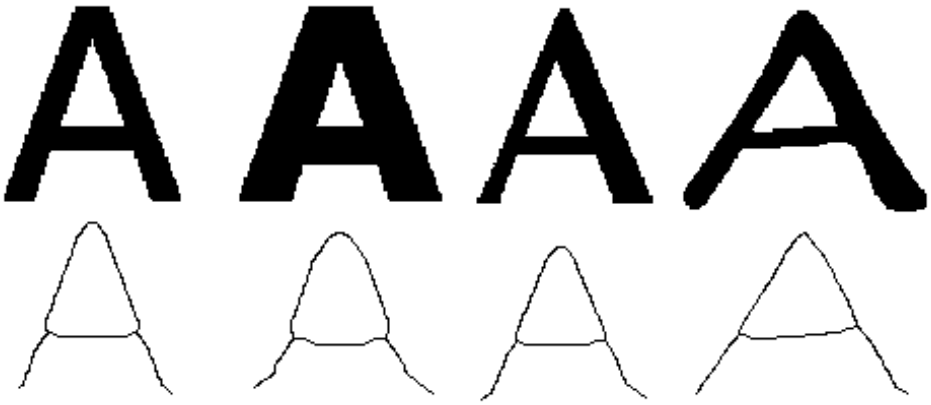
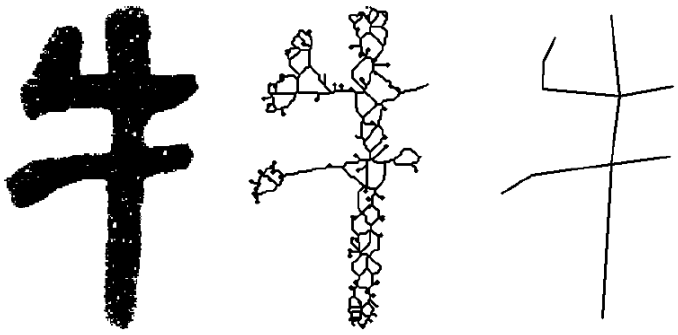


Abbildung 2.4: Skelettierungsalgorithmen bieten eine Datenreduktion und Abstraktion von Objekten. Hier wird der Buchstabe **A** in verschiedenen Strichstärken auf Ein-Pixel-Breite standardisiert, um eine optische Zeichenerkennung zu erleichtern [BILD 1].

Zur Erfassung von bestimmten Eigenschaften von Binärbildern vereinfachen Skelette im Allgemeinen die Analysen, da durch die Reduzierung der Daten unwesentliche Eigenschaften ausgeblendet werden. Zum Beispiel ist bei der Zeichenerkennung die Strichstärke egal - es muss das Zeichen unabhängig davon erkannt werden. Also reduziert eine Skelettierung nicht nur die Daten auf das Wesentliche, sie bietet außerdem eine Standardisierung der Zeichen auf Ein-Pixel-Breite (siehe Abbildung 2.4). Skelettierungsalgorithmen, die zudem noch unanfällig gegen Rauschen sind und damit streng genommen nicht die topologische Struktur erhalten (z.B. Triangulations-Skelettierungen), machen eine Erkennung teilweise überhaupt erst möglich (siehe Abbildung 2.5).

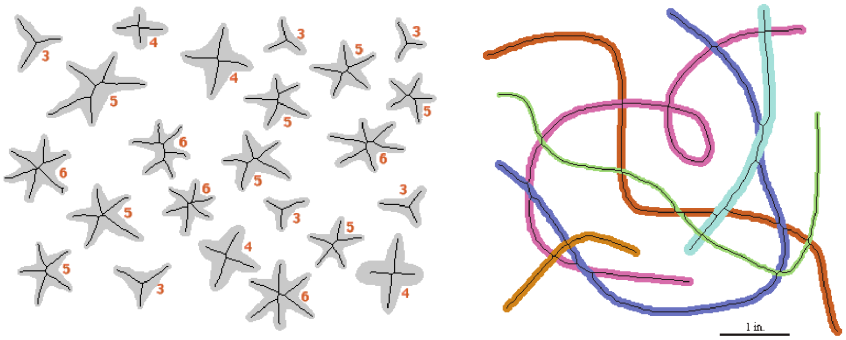


(a) (b) (c)

Abbildung 2.5: Rauschunempfindliche Algorithmen, die streng genommen nicht die topologische Struktur der Objekte erhalten, vereinfachen die optische Zeichenerkennung. (a) zeigt ein verrauschtes chinesisches Zeichen. In (b) wurde ein rauschempfindlicher Ausdünnungs-Algorithmus verwendet. In (c) ein rauschunempflindlicher Triangulations-Algorithmus [62].

Ein weiteres denkbares Einsatzgebiet von Skelettierungen ist in Abbildung 2.6 (a) zu sehen. Dem Menschen fällt es leicht die Anzahl der Zacken an einem Stern zu bestimmen. Indem die Endpunkte (Pixel mit nur einem Nachbarn) der Skelette der Objekte gezählt werden, kann dies auch von einer Maschine erfasst werden. Mit Hilfe der Skelette können in Abbildung 2.6 (b) nicht nur die Anzahl der Kabel (= (Anzahl der Endpunkte)/2), sondern auch deren Knotenpunkte erkannt werden. Diese befinden sich immer dort, wo Pixel des Skelettes mehr als zwei Nachbarn besitzen. Ungefähre Berechnungen der Längen der Kabel sind mittels Konturverfolgungsalgorithmen ebenfalls möglich.

Auch eine Skelettierung des Hintergrundes eines Bildes kann sinnvoll sein. Abbildung 2.7 zeigt ein beispielhaftes Mikroskopbild. Es wurde binarisiert und anschließend der Hintergrund skelettiert. Das Ergebnis teilt das Bild in Regionen, welche als Wirkungsbereiche der einzelnen Objekte interpretiert werden können.



(a) (b)

Abbildung 2.6: Bei ein-Pixel-breiten Skeletten können Objektinformationen durch das Zählen der verschiedenen Arten von Pixeln ermittelt werden. In (a) kann die Anzahl von Zacken der Sterne durch das Zählen der Endpunkte (Pixel mit nur einem Nachbarn) bestimmt werden. In (b) ist sogar die Anzahl an Kabeln (= (Anzahl der Endpunkte)/2) und deren Schnittpunkte (Pixel mit mehr als 2 Nachbarn) berechenbar [48].

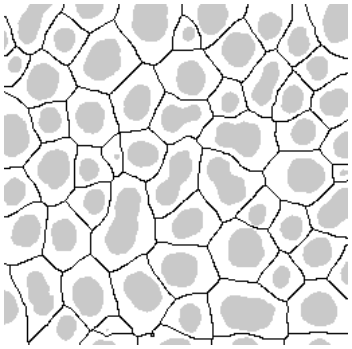
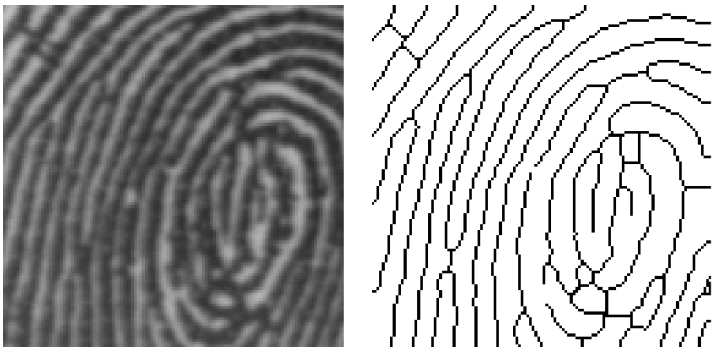


Abbildung 2.7: Es müssen nicht immer nur die Objekte skelettiert werden. Eine Skelettierung des Hintergrundes kann eine Art Wirkungsbereich der Objekte angeben [BILD 25].

Der Fakt, dass Skelette wesentliche Eigenschaften der ursprünglichen Objekte widerspiegeln und dabei eine hohe Reduktion der Datenmenge stattfindet, wird auch zur Datenkompression genutzt. So kann beispielsweise ein Fingerabdruck, der mittels Schwellenwertberechnung binarisiert und anschließend skelettiert wurde, wesentlich platzsparender als das Original abgespeichert werden (siehe Abbildung 2.8).

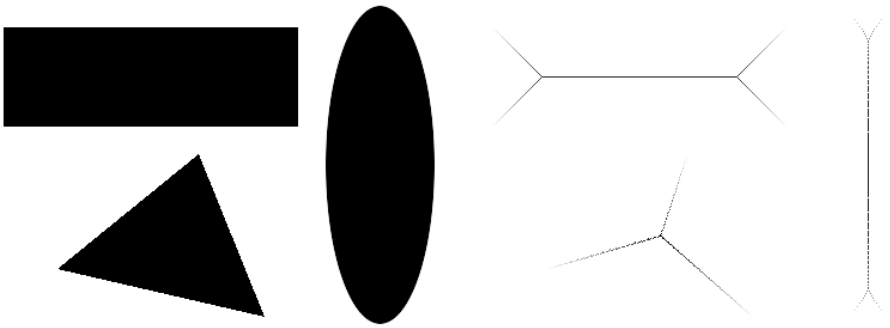
Eine besondere Form der Datenkompression von Binärbildern ist das Distanz-Skelett (siehe Abschnitt 3.2). Es ermöglicht eine verlustfreie Speicherung und vollständige Wiederherstellung des Originalbildes (Abbildung 2.9). Das Distanz-Skelett ist aber im Gegensatz zu den meisten anderen Skelettierungen kein Binärbild, da jedes Pixel des Skelettes noch mit einem Gewicht versehen ist.

Wie anhand der vorangegangenen Beispiele gezeigt, spielen Skelette in der digitalen Bildverarbeitung eine wichtige Rolle zur Bildanalyse. Es ist wichtig, je nach Einsatzgebiet, einen passenden



(a) (b)

Abbildung 2.8: Skelettierte Bilder sind in der Regel platzsparender speicherbar, wobei die wichtigen Informationen des Bildes oftmals erhalten werden können. In (a) ist ein Fingerabdruck zu sehen. Das skelettierte Bild (b) verbessert nicht nur die Erkennung und Klassifizierung, es benötigt als Binärbild auch weniger Speicherplatz [48].



(a) (b)

Abbildung 2.9: Das Distanz-Skelett ermöglicht eine Wiederherstellung des Originalbildes. Es ist kein Binärbild, sondern ein 2-dimensionales Array mit den gleichen Abmaßen wie das Original. Es kann als Grauwertbild dargestellt werden. (b) ist die Grauwertbilddarstellung des Distanz-Skelettes des Originalbildes (a) [BILD 11].

Skelettierungsalgorithmus zu wählen. Dabei spielen nicht nur die letztlichen Eigenschaften des Skelettes eine Rolle, auch muss, beispielsweise in der Echtzeitverarbeitung, auf zeiteffiziente Algorithmen geachtet werden.

In den nächsten Kapiteln werden nun typische Skelettierungsverfahren vorgestellt. Dabei wird entsprechend der Kategorisierung aus Abschnitt 2.2 vorgegangen und nach jeweils allgemeinen Informationen zu den Verfahren werden konkrete Beispiele behandelt.

2.4 Mathematische Schreibweise / Definitionen

Die in dieser Arbeit verwendete mathematische Schreibweise orientiert sich an [27] und [29]. Für den hier ausschließlich behandelten 2-dimensionalen Fall besteht ein digitales Bild aus einer Funktion I definiert auf einer diskreten Menge $\mathbb{C}$. $\mathbb{C}$ enthält als Elemente Pixel $p = (p_x, p_y)$. Der Wertebereich $W(I) = \{0, ..., G_{max}\} \subset \mathbb{N}$ ist für Binärbilder mit $G_{max} = 1$ festgelegt. Dabei werden Pixel mit $I(p) = 1$ als schwarze Pixel bzw. *Objektpixel* oder *Vordergrundpixel* bezeichnet, Pixel mit $I(p) = 0$ als weiße bzw. *Hintergrundpixel*. Die Menge $\langle I \rangle_{\mathbb{C}}$ soll alle Vordergrundpixel umfassen:

$$\langle I \rangle_{\mathbb{C}} := \{p \in \mathbb{C} \,|\, I(p) = 1\} \tag{2.2}$$

Da die Menge $\mathbb{C}$ im Folgenden keine entscheidende Rolle mehr spielt, wird zugunsten der Lesbarkeit auf den Index $\mathbb{C}$ verzichtet: $\langle I \rangle \mathrel{\widehat{=}} \langle I \rangle_{\mathbb{C}}$. An dieser Stelle sei betont, dass diese Schreibweise für jedes Bild gilt. Insbesondere soll der Begriff Skelett hier nicht nur die skelettalen Punkte (also Vordergrundpixel) umfassen, vielmehr ist unter dem Skelett S das Bild zu verstehen, welches Vorder- und Hintergrundpixel umfasst. $\langle S \rangle$ dagegen ist die Menge aller skelettalen Punkte von S.

Für Skelettierungsalgorithmen spielen *Adjazenz-* bzw. *Nachbarschafts*-Begriffe eine große Rolle. Zwei Pixel p und q werden als *4-adjazent* bezeichnet, wenn $p \neq q$ ist und p eine Kante mit q teilt, d.h.

$$\begin{aligned} &p, q \quad \text{sind } \textit{4-adjazent} \quad :\longleftrightarrow \\ &p_x = q_x \wedge |p_y - q_y| = 1 \vee |p_x - q_x| = 1 \wedge p_y = q_y \end{aligned} \tag{2.3}$$

Teilen p und q mindestens eine Ecke, werden sie als *8-adjazent* bezeichnet:

$$\begin{aligned} &p, q \quad \text{sind } \textit{8-adjazent} \quad :\longleftrightarrow \\ &p \neq q \wedge |p_x - q_x| \leq 1 \wedge |p_y - q_y| \leq 1 \end{aligned} \tag{2.4}$$

Die *Nachbarschaft* $N_\alpha(p)$ eines Pixels p umfasst alle α-adjazenten ($\alpha \in \{4,8\}$) Pixel von p. Nummeriert man die Nachbarn von p entsprechend Abbildung 2.10 durch, ergeben sich folgende Nachbarschaftsbegriffe:

$$N_4(p) = \{p_1, p_3, p_5, p_7\} \tag{2.5}$$

$$N_8(p) = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\} \tag{2.6}$$

Mit $b_\alpha(p, I)$ soll die Anzahl der Objektpixel innerhalb der α-Nachbarschaft von p, bezogen auf Bild I, bezeichnet werden.

$$b_\alpha(p, I) := \sum_{q \in N_\alpha(p)} I(q) \qquad \alpha \in \{4,8\} \tag{2.7}$$

p_4	p_3	p_2
p_5	p	p_1
p_6	p_7	p_8

Abbildung 2.10: Es werden zwei verschiedene Nachbarschaftsbegriffe definiert: $N_4(p)$ umfasst alle 4-adjazenten Pixel von p, also $\{p_1, p_3, p_5, p_7\}$, $N_8(p)$ hingegen alle 8-adjazenten Pixel von p, also $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.

Ein Objektpixel p ist für einen definierten Nachbarschaftsbegriff $N_\alpha(p)$ ein α-Konturpixel, wenn mindestens ein Pixel in der Nachbarschaft von p ein Hintergrundpixel ist:

$$\begin{aligned} &p \quad \text{ist } \alpha\text{-Konturpixel des Bildes } I \quad :\longleftrightarrow \\ &p \in \langle I \rangle \wedge \exists q (q \in N_\alpha(p) \wedge q \notin \langle I \rangle) \qquad \alpha \in \{4,8\} \end{aligned} \tag{2.8}$$

Für die Menge aller α-Konturpixel von I wird $C_\alpha(I)$ als Bezeichnung eingeführt:

$$C_\alpha(I) := \{p \in \langle I \rangle \mid \exists q (q \in N_\alpha(p) \wedge q \notin \langle I \rangle)\} \qquad \alpha \in \{4,8\} \tag{2.9}$$

Angaben zur Laufzeit von Algorithmen erfolgen, wenn nicht anders angegeben, immer für ein $n \times n$ Bild. Eine Laufzeit von $\mathcal{O}(n^2)$ entspricht also einem linearen Wachstum in Bezug auf die Gesamtzahl der Pixel.

Weitere Begriffe, wie *Zusammengehörigkeit (connectedness)*, *Kreuzungszahl (crossing number)* oder verschiedene *Metriken*, werden später einzeln in den Abschnitten definiert, in denen sie benötigt werden.

3 Distanz-Transformations-Algorithmen

Bei Distanz-Transformations-Algorithmen stehen, wie der Name schon vermuten lässt, Abstandsberechnungen im Vordergrund. Das Skelett ist dabei eine Teilmenge von $\langle I \rangle$ und es werden im Allgemeinen diejenigen Pixel aus $\langle I \rangle$ gewählt, die den größten Abstand zum Hintergrund haben.

Um Abstandsberechnungen vornehmen zu können, sind Metriken notwendig. In dieser Arbeit werden drei verschiedene Metriken verwendet. Für zwei Punkte p und q sind sie wie folgt definiert:

- **Betragssummenmetrik** d_1 Diese Metrik wird auch *Manhattan-* oder *Taxi-Metrik* genannt, da sie der Fahrtstrecke auf einem Stadtplan im Schachbrettmuster entspricht.

$$d_1(p,q) := |p_x - q_x| + |p_y - q_y| \tag{3.1}$$

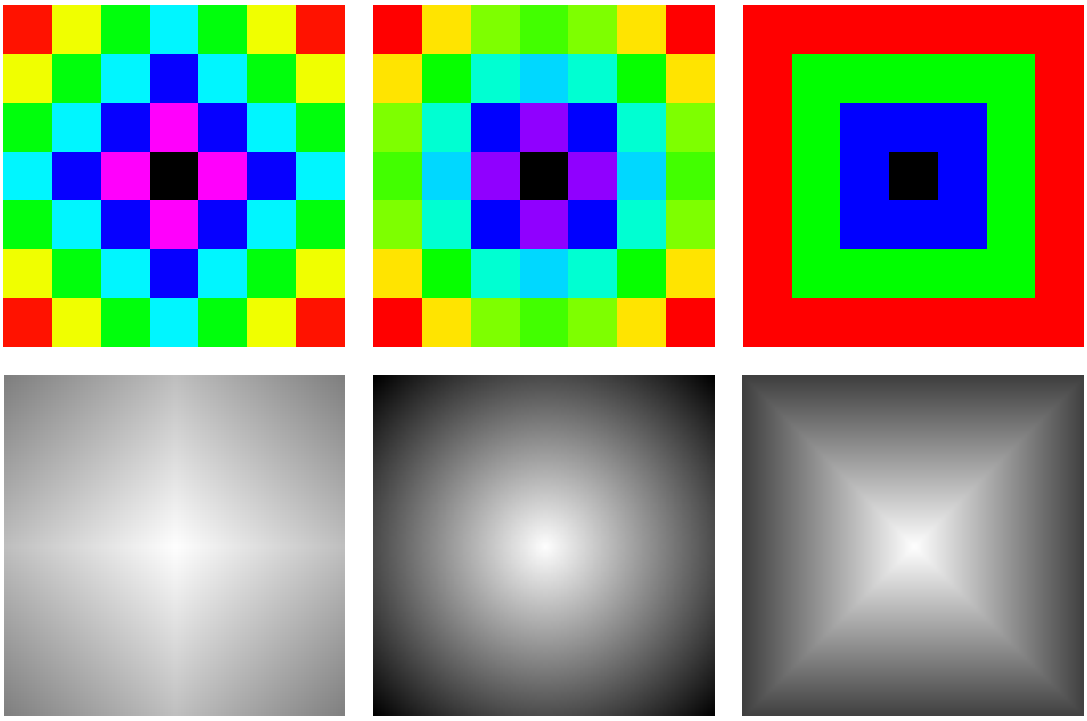
- **Euklidische Metrik** d_2

$$d_2(p,q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{3.2}$$

- **Maximummetrik** d_∞

$$d_\infty(p,q) := max\{|p_x - q_x|, |p_y - q_y|\} \tag{3.3}$$

In Abbildung 3.1 sind die verschiedenen Metriken illustriert. Dabei haben Pixel mit dem gleichen Abstand zum Mittelpunkt dieselbe Farbe.



(a) (b) (c)

Abbildung 3.1: Für Distanz-Transformations-Algorithmen hat die Wahl der Metrik einen großen Einfluss auf das Resultat. Die Bilder aus Spalte (a) veranschaulichen die Betragssummenmetrik d_1, Spalte (b) die euklidische Metrik d_2 und Spalte (c) die Maximummetrik d_∞. Pixel mit den gleichen Abständen zum Mittelpunkt des Bildes haben dabei dieselbe Farbe.

3.1 Mediale Achse

Die Mediale Achse ist eine der ältesten Ideen einer Skelettdarstellung. Sie geht zurück auf den Biologen Harry Blum, der in den 1960er Jahren eine Beschreibung für ebene abgeschlossene Gebiete suchte, um diese auf "inhärente Eigenschaften" zu untersuchen [57].

Die Mediale Achse M eines Bildes I ist abhängig von der Metrik d wie folgt definiert:

$$\begin{aligned} M_I := \{p \in \langle I \rangle \mid \exists r \exists s (r, s \in C_\alpha(I) \wedge r \neq s \wedge d(p,r) = d(p,s) \\ \wedge \forall t (t \in C_\alpha(I) \rightarrow d(p,t) \geq d(p,r)))\} \end{aligned} \tag{3.4}$$

Sie umfasst also diejenigen Objektpixel, deren minimaler Abstand zu einem Konturpixel zu einem weiteren Konturpixel besteht. Pixel die diese Eigenschaft besitzen, werden *symmetrisch* genannt. Man beachte, dass Konturpixel selbst nicht symmetrisch sein können. Eine Interpretation dieser Punkte ist, dass es die Mittelpunkte der maximalen Kreisscheiben sind. Dabei heißt eine Kreis-

scheibe k_1 maximal, wenn sie eine Teilmenge von $\langle I \rangle$ ist und es keine weitere Kreisscheibe k_2 mit demselben Mittelpunkt gibt, die ebenfalls Teilmenge von $\langle I \rangle$ ist, sodass k_1 eine echte Teilmenge von k_2 ist. In Abbildung 3.2 sind solche maximalen Kreisscheiben zu sehen. Zusätzlich ist eine alternative Definition einer maximalen Kreischreibe angegeben.

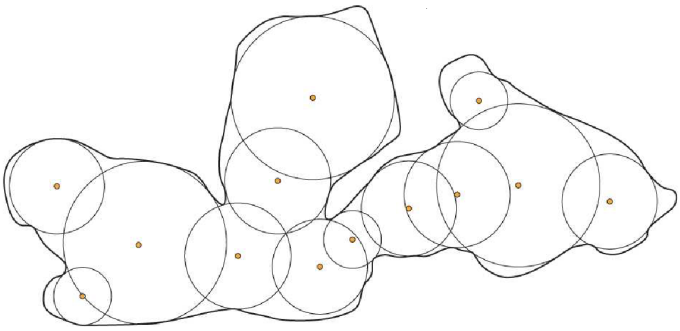
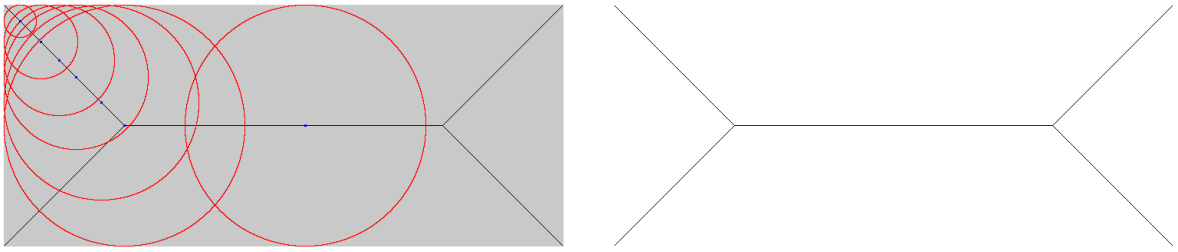


Abbildung 3.2: Symmetrische Punkte p repräsentieren Mittelpunkte von maximalen Kreisscheiben. Dabei ist eine Kreisscheibe maximal, wenn sie Teilmenge des Objektes ist, ihr Radius der kleinste Abstand von p zu einem Konturpixel ist und der Kreis die Kontur an mindestens zwei verschiedenen Punkten berührt. Im Bild sind ausgewählte symmetrische Punkte mit ihren Kreisscheiben eingezeichnet [27].

In Abbildung 3.3 wird die Mediale Achse eines Rechtecks gezeigt. Jeder Punkt des Skelettes repräsentiert den Mittelpunkt einer maximalen Kreisscheibe. In 3.3 (b) ist deutlich der aufspannende Charakter der Medialen Achse zu erkennen. Wird zu jedem Punkt aus M zusätzlich der Radius abgespeichert, kann (in der Theorie; siehe unten) das ursprüngliche Objekt wieder hergestellt werden.

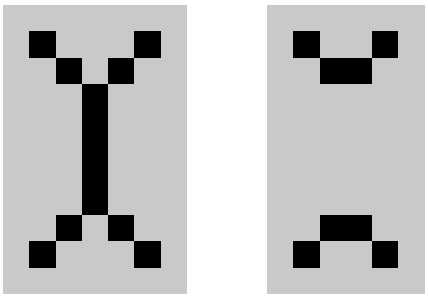


(a) (b)

Abbildung 3.3: Jeder Punkt der Medialen Achse ist ein Mittelpunkt einer maximalen Kreisscheibe. In (a) wurde die Mediale Achse einem Rechteck eingezeichnet. Zur Veranschaulichung sind für einzelne symmetrische Punkte die zugehörigen maximalen Kreisschreiben dargestellt. (b) zeigt die freigestellte Mediale Achse.

Die Mediale Achse wird in der Literatur oft als das ursprüngliche, das "perfekte" Skelett bezeichnet, da alle skelettalen Punkte "perfekt" mittig liegen. Doch gerade diese Voraussetzung, dass symmetrische Punkte exakt in der Mitte liegen müssen, bringt in der digitalen Bildverarbeitung, in der mit diskreten Bildern gearbeitet wird, Probleme mit sich.

Durch die Rasterung bei der Digitalisierung kann es sein, dass sich die exakte Mitte einer Kreisscheibe genau zwischen zwei Pixeln befindet. In Abbildung 3.4 wird dieses Problem verdeutlicht. Die Breite des linken Rechtecks hat nach der Digitalisierung eine ungerade Anzahl von Pixeln. Damit haben die mittleren Pixel zum linken und rechten Rand des Rechtecks exakt den gleichen Abstand und werden als symmetrische Pixel erkannt. Dem rechten Rechteck wurde bei der Digitalisierung eine gerade Anzahl von Pixeln in der Breite zugeordnet. Dies hat zur Folge, dass die exakte Mitte genau zwischen den zwei mittleren Pixelstreifen liegt und damit keine der Pixel symmetrisch sind.



(a) (b)

Abbildung 3.4: Für Rasterbilder liefert die Mediale Achse stark unterschiedliche Ergebnisse, da schon bei kleinen Veränderungen einzelne Pixel nicht mehr symmetrisch sein können. In (a) ist die Mediale Achse eines Rechtecks mit ungerader Pixelanzahl in der Breite zu sehen [BILD 29]. Die mittleren Pixel haben zu beiden Seiten den gleichen Abstand und sind symmetrisch. In (b) hat die Breite des Rechtecks eine gerade Anzahl von Pixeln und es gibt keine exakt mittigen Pixel [BILD 28].

Dieses Problem der Rasterung ist ein entscheidender Punkt, warum die Mediale Achse kaum Anwendung in der pixelbasierten digitalen Bildbearbeitung findet. Auch eignet sich die Mediale Achse, trotz theoretischer Wiederherstellbarkeit des ursprünglichen Objektes aus dem Skelett, aus diesem Grund nicht für die Datenkompression.

Pseudocode 1 beschreibt eine einfache, exakte Berechnung der Medialen Achse. Auffallend dabei ist eine im Vergleich zu anderen Skelettierungsalgorithmen hohe Laufzeitkomplexität. Diese ist ein weiterer wichtiger Punkt, warum die Mediale Achse kaum Verwendung findet. Für diesen einfachen Ansatz einer Berechnung der Medialen Achse wird prinzipiell eine Laufzeit von $\mathcal{O}(n^4)$ benötigt, wobei die zugehörige Konstante durch Betrachtung des Worst-Case-Szenarios sogar auf $\frac{1}{4}$ beschränkt werden kann. Die erste Schleife (Zeile 3) durchläuft die Menge aller Objektpixel, die keine Konturpixel sind, und die zweite Schleife (Zeile 6) durchläuft alle Konturpixel. Da diese beiden Mengen disjunkt sind und die Summe der Elemente beider Mengen maximal n^2 sein kann, ist das größtmögliche Produkt $\frac{1}{2}n^2 * \frac{1}{2}n^2 = \frac{1}{4}n^4$. Dennoch würde beispielsweise bei einem Bild mit 800*600 Pixeln das Schleifeninnere $\frac{1}{4} * (800 * 600)^2 = \frac{1}{4} * 480000^2 = 5,76 * 10^{10}$ mal durchlaufen. Bei angenommenen zehn Operationen pro Schleifendurchlauf würde diese Berechnung bei einem

Pseudocode 1 : Berechnung der Medialen Achse

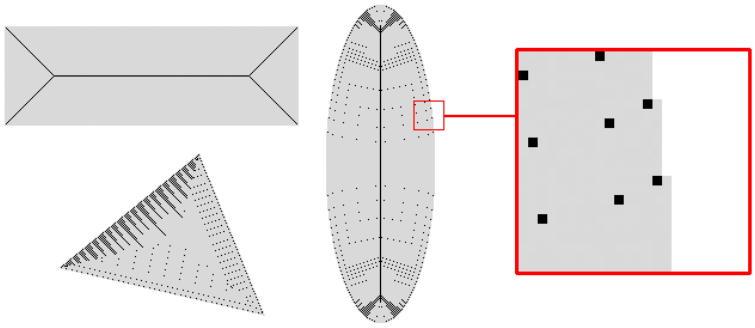
```
Eingabe : Bild ⟨I⟩                                   /* Menge aller Objektpixel */
Ausgabe : Mediale Achse M von I
1  In ← ⟨I⟩\C_α                                     /* Menge aller ''inneren'' Objektpixel */
2  M ← ∅
3  für alle Pixel p aus In tue
4      min ← ∞                                                  /* min. Abstand */
5      z ← falsch                                  /* Flag falls min. Abstand zweifach */
6      für alle Pixel q aus C_α tue
7          wenn d(p,q) < min dann
8              min ← d(p,q)
9              z ← falsch
10         sonst
11             wenn d(p,q) = min dann
12                 z ← wahr
13     wenn z = wahr dann
14         M ← M ∪ {p}
```

3GHz Prozessor noch immer $(5,76 * 10^{10} * 10)/(3 * 10^9) = 192$ Sekunden dauern. Selbst für ein 128x128-Pixel-Bild werden nach der gleichen Rechnung noch $0,22$ Sekunden benötigt. Dies ist gerade für zeitkritische Anwendungen deutlich zu viel[1].

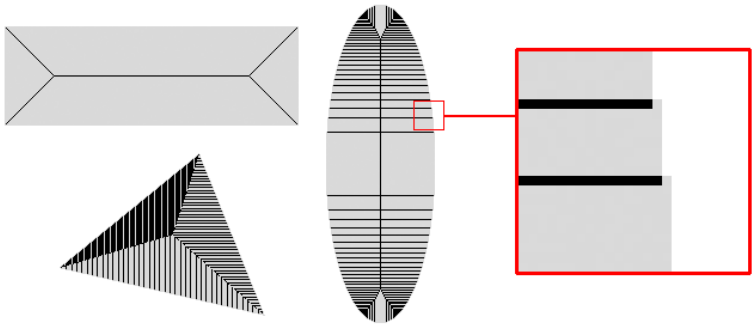
In Abbildung 3.5 wurden die Medialen Achsen für das gleiche Bild mit den drei unterschiedlichen Metriken berechnet. Es ist zu erkennen, dass bei der euklidischen Metrik aus den genannten Gründen nur vereinzelte Punkte als symmetrisch erkannt werden. Dadurch geht der aufspannende Charakter der Medialen Achse etwas verloren. Es ist offensichtlich, dass die Mediale Achse nicht zusammenhängend ist und damit nicht die topologische Struktur des Originalbildes wiedergibt. Unter Verwendung der Betragssummenmetrik werden, wie in der Vergrößerung zu sehen, alle Pixel die von einer "Treppenstufe" nach innen führen, als symmetrisch markiert. Dadurch werden die Objekte oftmals mit Linien durchzogen. Die Anwendung der Maximummetrik erscheint nicht sinnvoll, da die meisten Objektpixel den gleichen Abstand zu gleich zwei oder drei nebeneinander liegenden Konturpixeln haben. Damit werden fast alle Pixel als symmetrisch erkannt und es findet kaum eine Reduzierung statt. Für das Rechteck werden sogar alle Objektpixel, die nicht Konturpixel sind, markiert. Eine weitere auftretende Problematik wird in Abbildung 3.6 veranschaulicht.

Die Mediale Achse scheint also für die pixelbasierte digitale Bildverarbeitung wenig geeignet. Deshalb wurden in der Vergangenheit immer wieder Vorschläge für alternative Berechnungen gemacht, die die Mediale Achse annähern. So gibt es den *Steppenbrand-Algorithmus* oder auch

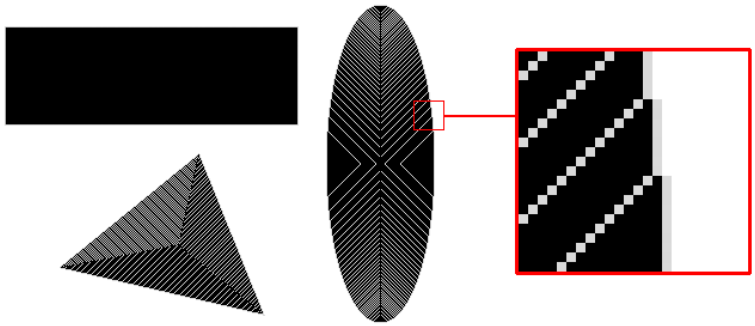
[1] Beispiel: Bei einem Video mit 25 Bildern pro Sekunde bleiben pro Bild $\frac{1}{25} = 0,04$ Sekunden Zeit.



(a)



(b)



(c)

Abbildung 3.5: Die Auswahl der den Berechnungen zugrundeliegenden Metrik hat auf die Mediale Achse starken Einfluss. In den Beispielen (a)-(c) wurde die Mediale Achse für dasselbe Bild mit unterschiedlichen Metriken erstellt. In (a) wurde die euklidische Metrik d_2 verwendet, in (b) die Betragssummenmetrik d_1 und in (c) die Maximummetrik d_∞ [BILD 11].

Wellenfront-Algorithmus genannt [27]. Andere Ansätze, die Mediale Achse nachzubilden, beschäftigen sich mit *Voronoi-Diagrammen* [57]. Montanari beschrieb 1968 ein weiteres Verfahren unter Verwendung einer eingeführten *Quasi-Euclidean-Distance* [35]. Diese Algorithmen sollen aber nicht Gegenstand dieser Arbeit sein. Eine sehr gute Alternative zur Medialen Achse bietet das Distanz-Skelett, welches im nächsten Kapitel behandelt wird.

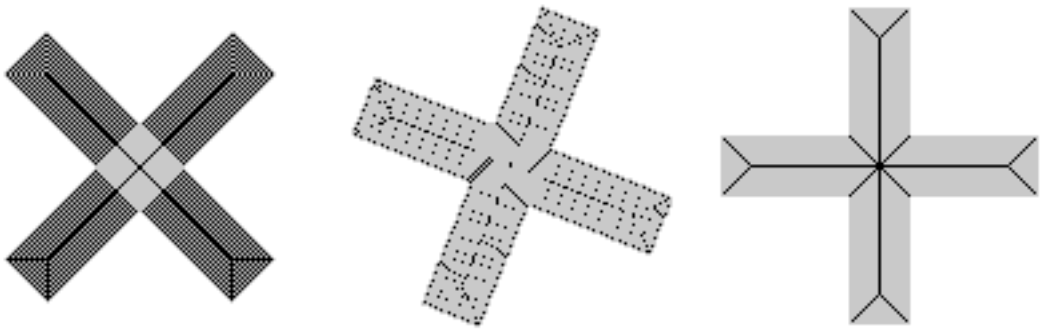


Abbildung 3.6: Die Mediale Achse liefert bei Drehung eines Objektes stark unterschiedliche Ergebnisse. Zur Verdeutlichung wurde in (a)-(c) ein Kreuz um jeweils 22,5° nach links gedreht [BILD 17,18,19]. Die dazugehörigen Medialen Achsen (Verwendung der d_2-Metrik) weisen untereinander kaum Ähnlichkeiten auf.

3.2 Distanz-Skelett

Das Distanz-Skelett beruht, wie auch die Mediale Achse, auf Abstandsberechnungen. Allerdings wird, anstelle des Abstandes zu den Konturpixeln, der Abstand zu den Hintergrundpixeln berechnet. Dazu wird eine *Distanz-Karte* (*Distance-Map*) angelegt. Diese lässt sich als Integer-Array mit den gleichen Dimensionen wie das Bild darstellen. Eine sehr einfach zu implementierende Distanz-Karte wurde 1966 von Rosenfeld/Pfalz vorgestellt [46]. Der Algorithmus dazu basiert auf der d_1-Metrik und die Distanz-Karte kann in $\mathcal{O}(n^2)$ mit nur zwei Rasterscans berechnet werden. Deshalb werden solche Algorithmen auch *two-pass-algorithms* genannt. Der Algorithmus von Rosenfeld/Pfalz macht sich eine alternative Definition der d_1-Metrik zunutze:

> Der Abstand $d_1(p,q)$ zwischen zwei Punkten $p \neq q$, ist die kleinste natürliche Zahl k, sodass eine Folge $p = p_0, p_1, p_2, ..., p_k = q$ existiert, wobei p_i 4-adjazent zu p_{i-1} ist $(1 \leq i \leq k)$. Wenn $p = q$, so ist der Abstand 0 [26].

Wenn nun eine Distanz-Karte D erstellt werden soll, auf der jedes Pixel p mit einem Wert belegt wird, der dem geringsten Abstand zu einem Hintergrundpixel nach der d_1-Metrik entspricht, kann aufgrund der genannten Definition einfach das Minimum aller Werte aus der 4er-Nachbarschaft $N_4(p)$ von p genommen und dieses um eins erhöht p zugeordnet werden. Hintergrundpixel selbst

erhalten den Wert 0. Dieses Verfahren kann in zwei Rasterscans durchgeführt werden, wobei im ersten Durchlauf in Standardrasterscan-Reihenfolge das Minimum von Nord- und Westnachbar (p_3, p_5) betrachtet wird und im zweiten Durchlauf, in umgekehrter Standardrasterscan-Reihenfolge, das Minimum von Süd- und Ostnachbar (p_7, p_1) einbezogen wird. Pseudocode 2 zeigt den entsprechenden Ablauf. Für Randpixel existieren nicht alle 4er Nachbarn auf der Distanz-Karte (beispielsweise haben die nördlichen Nachbarn der Pixel der ersten Bildzeile in der Distanz-Karte keine Werte). Fehlende Werte werden in diesem Fall ignoriert oder können alternativ als unendlich hoch betrachtet werden.

Pseudocode 2 : Berechnung der d_1-Distanz-Karte: ***distanzKarte(I)***

Eingabe : Bild I
Ausgabe : Distanz-Karte D

```
1  für alle Pixel p aus I tue                          /* Standardrasterscan-Reihenfolge */
2      wenn I(p) = 0 dann
3          D(p) ← 0
4      sonst
5          min ← ∞
6          wenn D(p3) existiert dann
7              min ← D(p3) + 1
8          wenn D(p5) existiert ∧ D(p5) + 1 < min dann
9              min ← D(p5) + 1
10         D(p) ← min
11 für alle Pixel p aus I tue          /* umgekehrte Standardrasterscan-Reihenfolge */
12     min ← D(p)
13     wenn D(p7) existiert ∧ D(p7) + 1 < min dann
14         min ← D(p7) + 1
15     wenn D(p1) existiert ∧ D(p1) + 1 < min dann
16         min ← D(p1) + 1
17     D(p) ← min
```

Abbildung 3.7 zeigt die d_1-Distanz-Karte einer Ellipse. In 3.7 (d) wurde zur Visualisierung die Distanz-Karte als Grauwertbild dargestellt, dabei wurde dem Wert 0 schwarz und dem höchsten Abstand weiß zugeordnet, alle Zwischenwerte linear verteilt.

Eine solche Distanz-Karte ist nun Ausgangspunkt für die Berechnung des Distanz-Skelettes S. Dafür werden einzelne Werte aus D in S übernommen. Einfach gesagt, wird ein Wert $D(p)$ übernommen, wenn in dessen 4er-Nachbarschaft in D kein größerer Wert vorkommt, es also gilt: $max(D(p_1), D(p_3), D(p_5), D(p_7)) \leq D(p)$. Für alle Pixel p, die nicht aus D übernommen werden, wird der Wert in S auf 0 gesetzt. In Pseudocode 3 wird gezeigt, wie auf diese Art und Weise das Distanz-Skelett bestimmt werden kann.

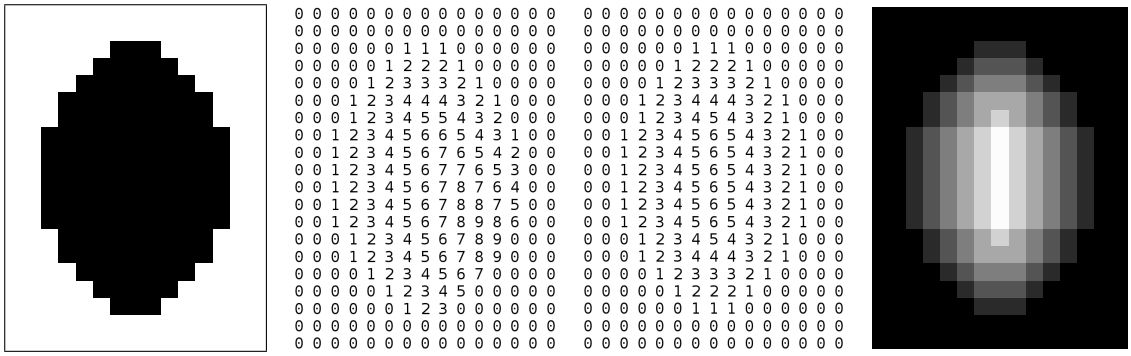


Abbildung 3.7: Berechnung der d_1-Distanz-Karte von (a) [Bild 8]: In (b) ist das Zwischenergebnis nach dem ersten Rasterscan (Pseudocode 2, Zeile 1) zu sehen, (c) zeigt die fertige d_1-Distanz-Karte nach dem zweiten Rasterscan (Pseudocode 2, Zeile 11). Zur Visualisierung wird in (d) die Distanz-Karte als Grauwertbild gezeigt (kleinster Wert - schwarz, größter Wert - weiß).

In Abbildung 3.8 wurde diese Selektion für die Ellipse durchgeführt - übrig bleibt das Distanz-Skelett, welches eine deutliche Datenreduktion erkennen lässt.

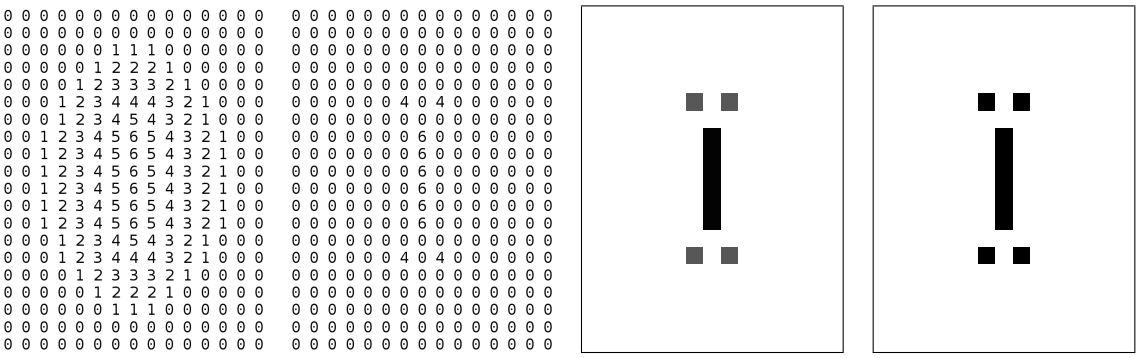


Abbildung 3.8: Berechnung des Distanz-Skelettes: Ausgangspunkt ist die Distanz-Karte (a). Nach Anwendung des Verfahrens aus Pseudocodes 3 erhält man das Distanz-Skelett (b). (c) ist eine Visualisierung von (b) als Grauwertbild (kleinster Wert - weiß, größter Wert - schwarz). In (d) wurde das Distanz-Skelett binarisiert. [BILD 8]

Im Gegensatz zur Medialen Achse bleiben beim Distanz-Skelett mittlere Linien auch dann erhalten, wenn die Breite des Objektes eine gerade Anzahl von Pixeln aufweist (siehe Abbildung 3.9). Diese sind dann jedoch zwei Pixel breit und damit nicht mehr *ideal thin*, wie es zum Beispiel von den Ausdünnungs-Algorithmen gefordert wird. Auch das Distanz-Skelett ist nicht zusammenhängend und erhält damit nicht die topologische Struktur, aber es hat eine andere interessante Eigenschaft: es ermöglicht das Wiederherstellen des ursprünglichen Objektes.

Um dies zu erreichen, muss das Skelett S wieder sukzessive ausgeweitet werden. Dabei kommt ein ähnliches Verfahren wie bei der Berechnung der Distanz-Karte zum Einsatz. Wiederum werden

Pseudocode 3 : Berechnung des Distanz-Skelettes

Eingabe : Bild I
Ausgabe : Distanz-Skelett S

```
1  D ←distanzKarte(I)                                   /* siehe Pseudocode 2 */
2  für alle Pixel p aus D tue
3      S(p) ← 0
4      wenn D(p_1) existiert ∧ D(p_1) > D(p) dann
5          beginne Nächste Iteration
6      wenn D(p_3) existiert ∧ D(p_3) > D(p) dann
7          beginne Nächste Iteration
8      wenn D(p_5) existiert ∧ D(p_5) > D(p) dann
9          beginne Nächste Iteration
10     wenn D(p_7) existiert ∧ D(p_7) > D(p) dann
11         beginne Nächste Iteration
12     S(p) ← D(p)
```

zwei Rasterscans durchgeführt, wobei diesmal das Maximum aus der 4er-Nachbarschaft in S um eins reduziert p zugeordnet wird. Als Ergebnis dieses Verfahrens erhält man die d_1-Distanz-Karte D. In Pseudocode 4 wird aus dem Distanz-Skelett S die ursprüngliche Distanz-Karte D berechnet.

Binarisiert man die Distanz-Karte D in der Weise, dass alle Werte ungleich 0 ein schwarzes Pixel zugewiesen bekommen und alle Werte gleich 0 ein weißes, erhält man wieder das ursprüngliche Bild I.

Diese Tatsache macht das Distanz-Skelett auch für die Datenkompression interessant. Wenn es ausreicht, lediglich die Positionen der skelettalen Punkte zusammen mit dem jeweiligen Wert abzuspeichern, so ist dies wesentlich effizienter als beispielsweise eine Abspeicherung im *Bitmap*-Format. Jedoch existieren bereits andere, weit verbreitete Kompressionstechniken, wie zum Beispiel die *Lauflängenkodierung*, die insbesondere bei Binärbildern oft Verwendung findet. Um die Effizienz der Kompression durch Speicherung des Distanz-Skelettes einordnen zu können, wurden im Rahmen dieser Arbeit die Lauflängenkodierung und die "Distanz-Skelett-Kodierung" für eine Reihe von Testbildern gegenübergestellt (siehe Anhang A). Dabei wurde festgestellt, dass die Lauflängenkodierung der Kodierung mittels Distanz-Skelett überlegen ist und in fast allen Fällen weniger Speicherplatz erfordert. Damit erscheint eine Abspeicherung eines Binärbildes als Distanz-Skelett aus Gründen der Speicherplatzeinsparung nicht sinnvoll.

In den Abbildungen 3.10 und 3.11 werden noch einmal ein paar Beispiele von binarisierten Distanz-Skeletten gezeigt.

Pseudocode 4 : Berechnung der d_1-Distanz-Karte aus dem Skelett S

Eingabe : Distanz-Skelett S
Ausgabe : Distanz-Karte D

```
1  für alle Pixel p aus S tue                    /* Standardrasterscan-Reihenfolge */
2      max ← S(p)
3      wenn D(p3) existiert ∧ D(p3) − 1 > max dann
4          max ← D(p3) − 1
5      wenn D(p5) existiert ∧ D(p5) − 1 > max dann
6          max ← D(p5) − 1
7      D(p) ← max
8  für alle Pixel p aus S tue          /* umgekehrte Standardrasterscan-Reihenfolge */
9      max ← D(p)
10     wenn D(p7) existiert ∧ D(p7) − 1 > max dann
11         max ← D(p7) − 1
12     wenn D(p1) existiert ∧ D(p1) − 1 > max dann
13         max ← D(p1) − 1
14     D(p) ← max
```

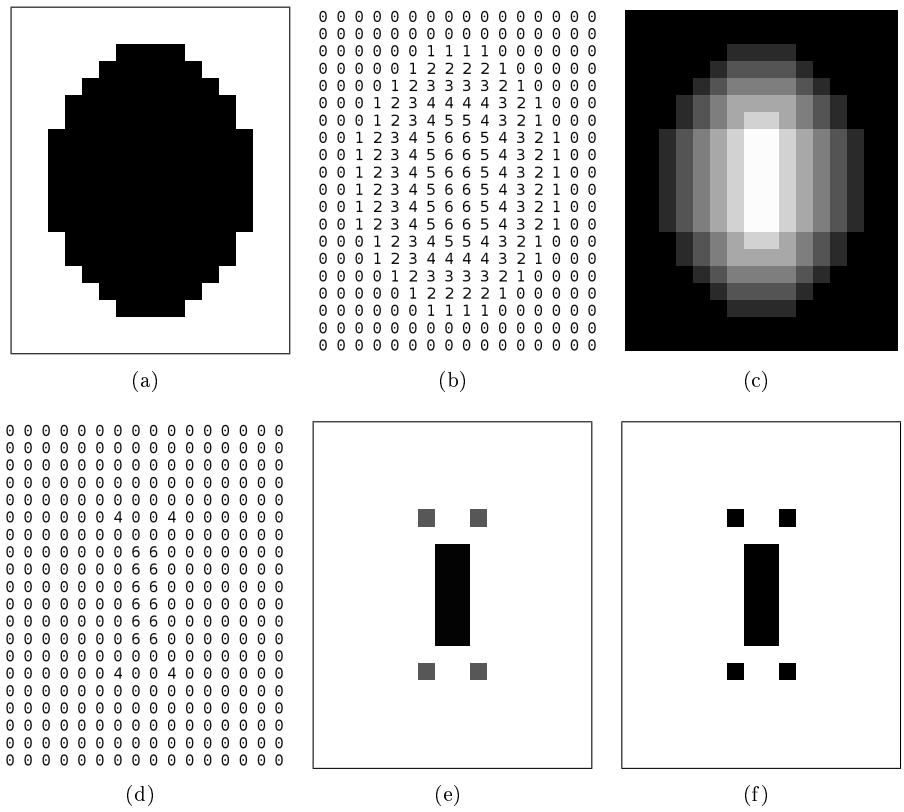


Abbildung 3.9: Distanz-Skelett-Berechnung einer Ellipse mit gerader Anzahl von Pixeln in der Breite (a) [BILD 7]. Im Gegensatz zur Medialen Achse bleiben beim Distanz-Skelett (d) mittlere Linien erhalten. (b) ist die Distanz-Karte von (a), (c) die Darstellung als Grauwertbild. In (e) ist das Distanz-Skelett als Grauwertbild abgebildet. In (f) wurde es binarisiert.

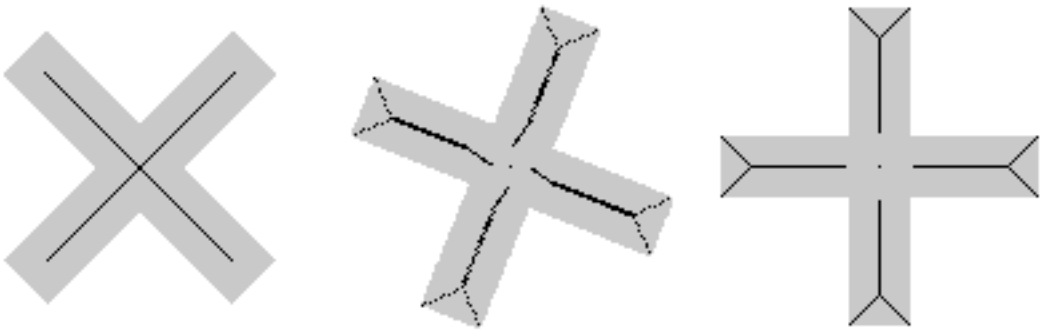


Abbildung 3.10: Im Gegensatz zur Medialen Achse weisen die Distanz-Skelette bei Drehungen deutlich höhere Ähnlichkeiten untereinander auf. In (a)-(c) wurde ein Kreuz um jeweils 22,5° nach links gedreht [BILD 17,18,19]. Bild (a) liefert ein nach menschlicher Bewertung optimales Ergebnis. Auch in Bild (b) und (c) sind Ähnlichkeiten der Skelette zu erkennen.

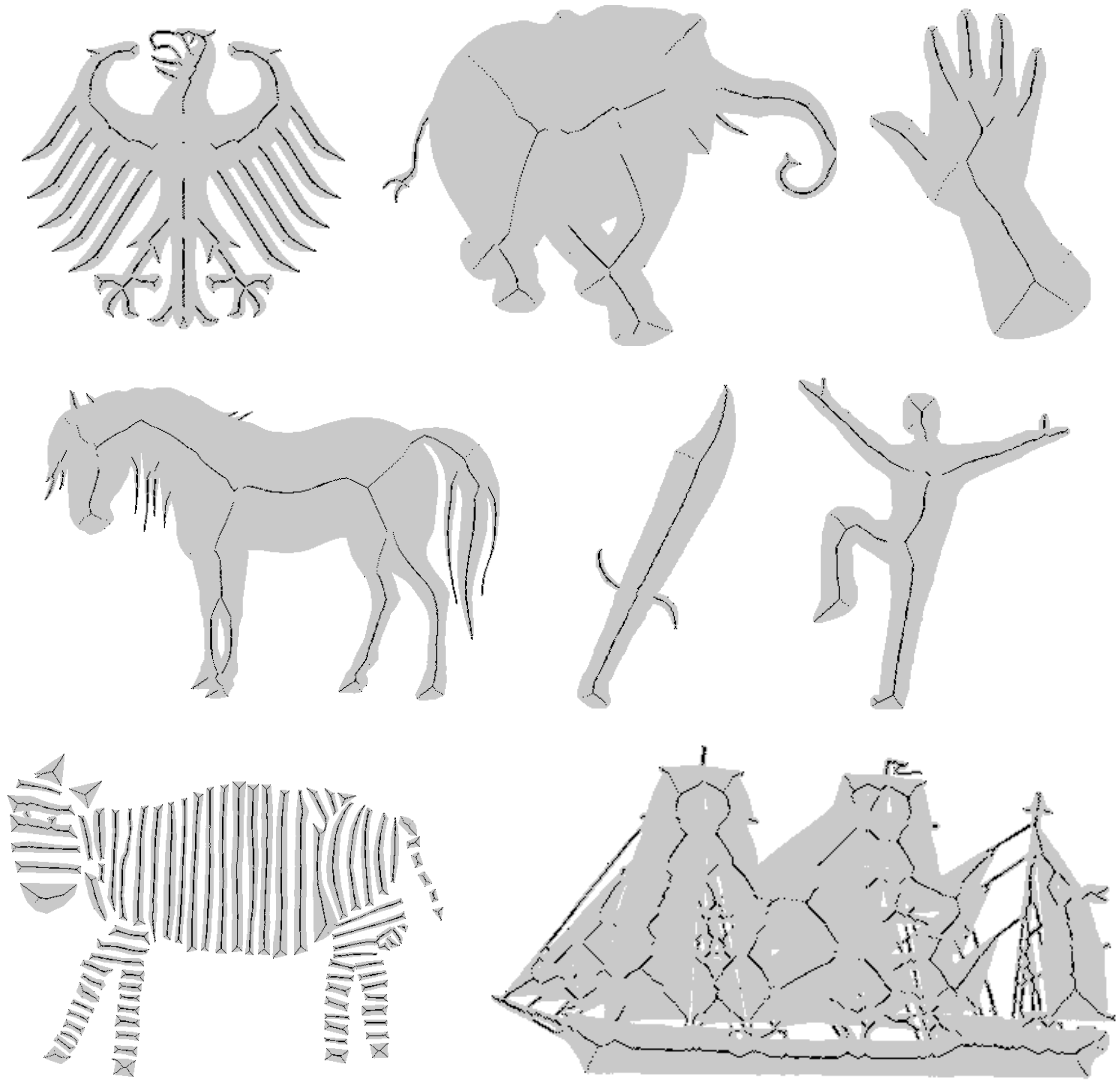
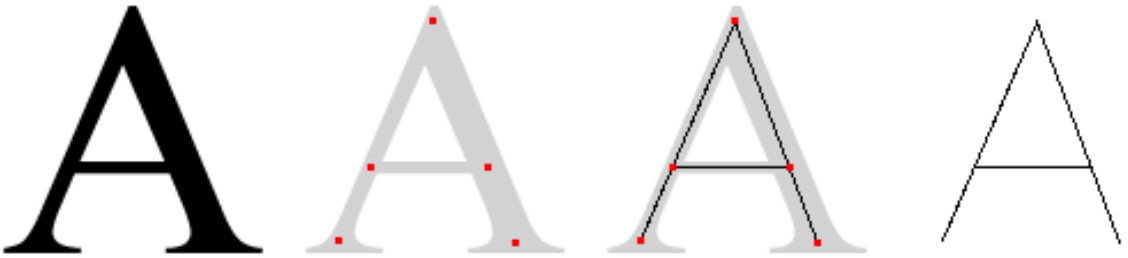


Abbildung 3.11: Weitere im Rahmen dieser Arbeit erstellte Beispiele von binarisierten Distanz-Skeletten: von links nach rechts, von oben nach unten: [BILD 5,6,12,26,24,23,35,30].

4 Kritische-Punkte-Algorithmen

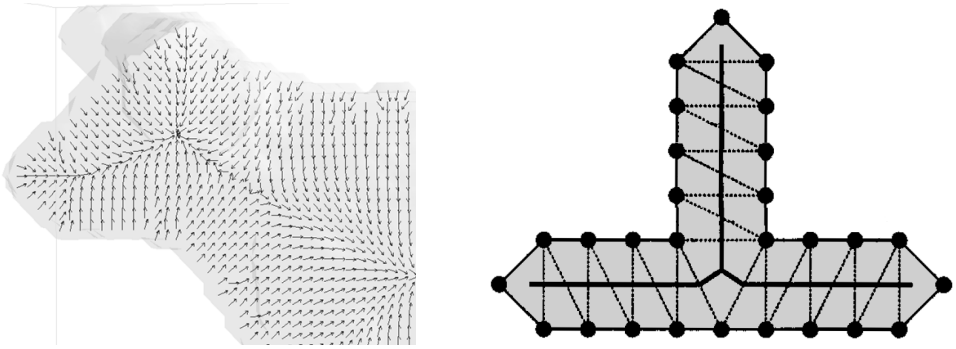
Kritische-Punkte-Algorithmen errechnen ein Skelett, indem sie versuchen kritische/wichtige skelettale Punkte (critical points) in einem Objekt auszumachen, um diese dann über einen Pfad zu verbinden. Abbildung 4.1 zeigt, wie dieses Vorgehen im Idealfall aussieht.



(a) (b) (c) (d)

Abbildung 4.1: Bei Kritische-Punkte-Algorithmen werden wichtige, skelettale Punkte ermittelt, um diese dann geeignet zu verbinden. In (b) wurden beispielhaft kritische Punkte für den Buchstaben **A** (a) markiert. Diese können dann durch Strecken verbunden werden (c). In (d) ist das freigestellte Skelett zu sehen.

In Abbildung 4.1 (c) wurden die kritischen Punkte vom Autor manuell eingezeichnet und liefern deshalb für das Skelett ein gutes Ergebnis. Die größte Schwierigkeit bei der automatisierten Umsetzung besteht darin, die geeigneten kritischen Punkte zu finden und so gibt es für die Ermittlung dieser Punkte viele verschiedene Ansätze: In [10] und [14] werden beispielsweise kritische Punkte mit Hilfe eines *Potentialfeldes* ermittelt (siehe Abbildung 4.2 (a)). Eine andere Idee ist es, die Objekte in Polygone zu zerlegen und dann Mittelpunkte der Polygone bzw. ihrer Stecken zu wählen (z.B. mittels Triangulation; Abschnitt 4.2). Kritische Punkte können für 3-dimensionale Skelette auch mit Hilfe der Medialen Achse und kürzesten geodätischen Abständen bestimmt werden (siehe [15] und [44]).

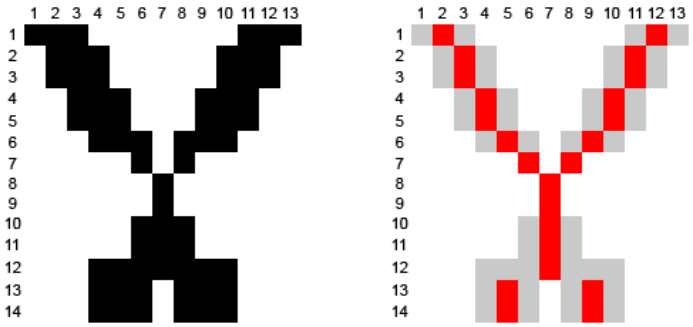


(a) (b)

Abbildung 4.2: Das Ermitteln geeigneter kritischer Punkte ist die größte Schwierigkeit bei Kritische-Punkte-Algorithmen. In [13] wird versucht Punkte mit Hilfe eines Potentialfeldes zu ermitteln (a). Bei der Triangulations-Technik (b) wird das Objekt in Dreiecke unterteilt und Mittelpunkte der Dreiecksseiten oder Schwerpunkte der Dreiecke werden zu kritischen Punkten [63].

4.1 Einfacher Kritische-Punkte-Ansatz

Die möglicherweise einfachste Art und Weise kritische Punkte zu bestimmen ist es, das Bild in Standardrasterscan-Reihenfolge "abzulaufen" und für jede Sequenz von Objektpixeln ein Mittelpunkt zu wählen (eine Sequenz von Objektpixeln wird im Folgenden auch als Block von Objektpixeln bezeichnet). Abbildung 4.3 soll dies an einem einfachen Beispiel verdeutlichen.



(a) (b)

Abbildung 4.3: Beim einfachen Kritische-Punkte-Ansatz wird für jede Sequenz von Objektpixeln bei Ablauf in Standardrasterscan-Reihenfolge ein Mittelpunkt bestimmt. In der ersten Zeile wird so als kritischer Punkt für die Sequenz (1,1)-(3,1) der Mittelpunkt (2,1) ermittelt. Analog in der nächsten Sequenz (11,1)-(13,1) der Mittelpunkt (12,1). Bild (a) zeigt das Original. In (b) sind alle kritischen Punkte rot eingezeichnet.

In Pseudocode 5 wird gezeigt, wie diese kritischen Punkte ermittelt werden können. Dabei werden Randpixel wie Hintergrundpixel behandelt. In Abbildung 4.3 ist bereits erkennbar, dass im Gegensatz zur "Theorie" (siehe Abbildung 4.1) sehr viel mehr kritische Punkte markiert wurden.

Da meist der Großteil dieser Punkte bereits zusammenhängend ist, wird auf ein anschließendes Verbinden der kritischen Punkte verzichtet. Abbildung 4.4 zeigt weitere Beispielbilder mit eingezeichneten kritischen Punkten.

Pseudocode 5 : Skelettberechnung mit dem einfachen Kritische-Punkte-Ansatz

Eingabe : Bild I
Ausgabe : Skelett $\langle S \rangle$ `/* Menge aller kritischen Punkte */`

```
1  xmax ← Breite von I
2  ymax ← Höhe von I
3  ⟨S⟩ ← ∅
4  flag ← falsch                          /* Flag wenn x in einer Sequenz liegt */
5  für y ← 0 bis ymax − 1 tue
6      für x ← 0 bis xmax tue
7          wenn flag = falsch ∧ x ≠ xmax ∧ I(x, y) = 1 dann
8              flag ← wahr
9              anfang ← x
10         sonst wenn flag = wahr ∧ (x = xmax ∨ I(x, y) = 0) dann
11             flag ← falsch
12             ende ← x
13             mitte ← anfang + ⌊(ende − anfang)/2⌋
14             ⟨S⟩ ← ⟨S⟩ ∪ {(mitte, y)}
```

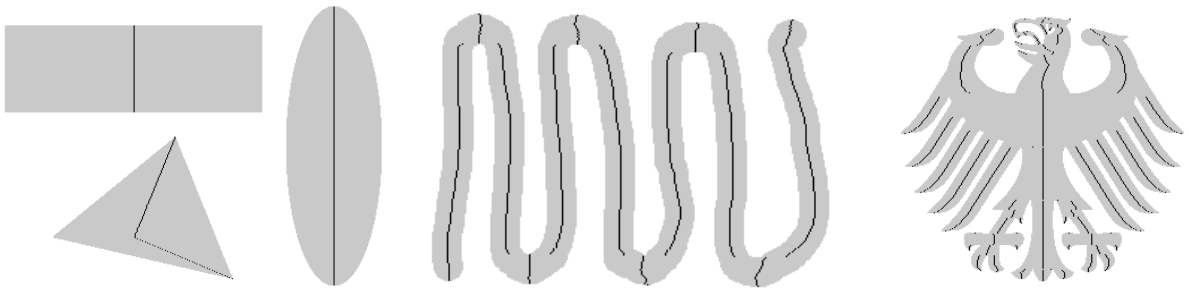


Abbildung 4.4: Beim einfachen Kritische-Punkte-Ansatz wird auf das Verbinden der kritischen Punkte verzichtet, da der Großteil der Punkte bereits verbunden ist. (a)-(c) zeigt weitere Skelette nach dem einfachen Kritische-Punkte-Ansatz: von links nach rechts: [BILD 11,31,5].

Kritische Punkte werden also ermittelt, indem zeilenweise Blocks von Objektpixeln betrachtet werden. Dabei ist es entscheidend, nach welchem System Zeilen aus einem Bild entnommen werden. Bei den obigen Beispielen wurden die Zeilen in Standardrasterscan-Reihenfolge betrachtet, aber in der gleichen Weise können die Spalten eines Bildes verwendet werden. Auch eine Entnahme von Zeilen im Winkel von 45° ist denkbar. Die Resultate unterscheiden sich dabei in der Regel stark, da sich andere Blocks von Objektpixeln ergeben, und diese natürlich andere Mittel-

punkte haben. In Abbildung 4.5 wurden verschiedene "Scanrichtungen" simuliert, indem das Bild gedreht wurde. Es ist deutlich zu erkennen, dass sich die Skelette stark unterscheiden und dass zudem dieser einfache Kritische-Punkte-Ansatz nicht isotrop, d.h. nicht invariant bei Drehungen ist. Ebenso wird offensichtlich nicht die topologische Struktur des Originalbildes gewahrt.

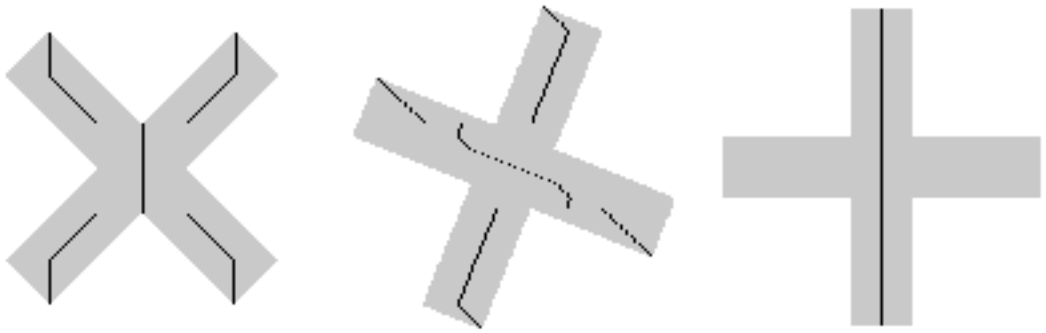
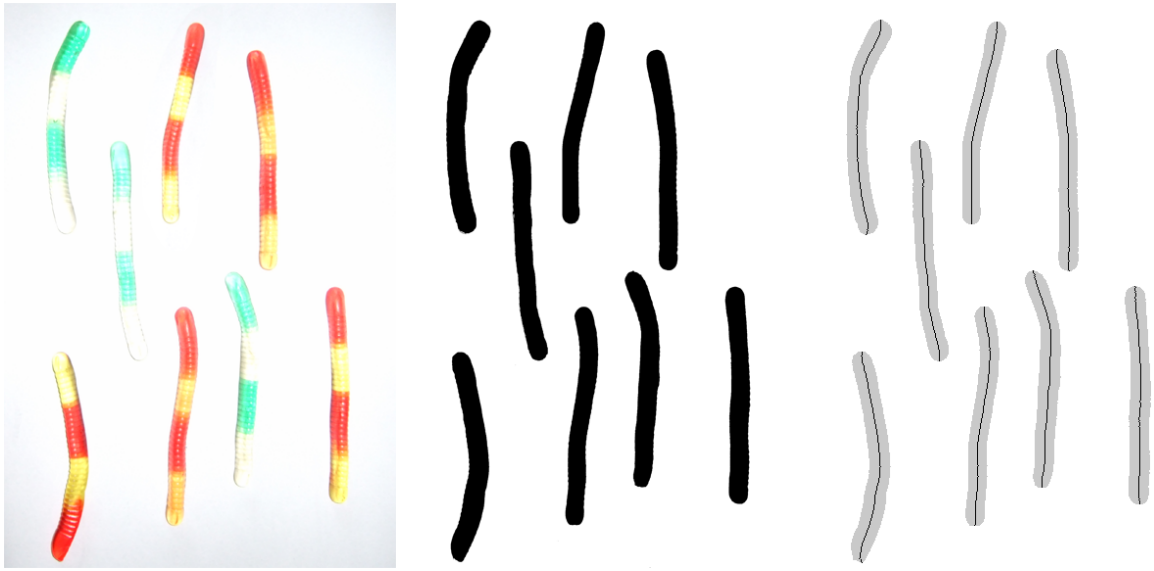


Abbildung 4.5: Durch eine festgelegte Rasterscan-Reihenfolge beim einfachen Kritische-Punkte-Ansatz ergeben sich bei Drehung eines Objektes andere Sequenzen von Objektpixeln. Dadurch ergeben sich folglich andere kritische Punkte, was zu stark unterschiedlichen Ergebnissen führt. In (a)-(c) wurde ein Kreuz um jeweils 22,5° nach links gedreht und die Skelette nach dem einfachen Kritische-Punkte-Ansatz berechnet [BILD 17,18,19].

Der Einsatz dieses einfachen Kritische-Punkte-Ansatzes ist also abhängig von der Scanrichtung. Um die Resultate adäquat nutzen zu können, muss die geeignete Scanrichtung vorher bekannt bzw. festgelegt sein. So eignet sich dieser Ansatz besonders gut, wenn die Ausrichtung der Objekte fest ist. Dies ist oftmals bei einem Fließband der Fall. Abbildung 4.6 zeigt ein solch denkbares Szenario. Die Objekte auf dem Fließband sind vertikal ausgerichtet, also eignet sich die Standardrasterscan-Reihenfolge, um die Objekte in voller Länge zu erfassen. Das Resultat ähnelt dem des oftmals verwendeten Ausdünnens (siehe Abschnitt 5), jedoch hat es zwei entscheidende Vorteile. Die Objekte werden zum einen in voller Länge repräsentiert und zum anderen liegt die benötigte Zeit zur Berechnung der kritischen Punkte deutlich unter der Rechenzeit für Ausdünnungs-Algorithmen (siehe auch Abschnitt 5).

Für das Ermitteln der kritischen Punkte wird nur ein Rasterscan benötigt, da das Ende eines Blockes an einem Übergang von einem Objektpixel zu einem Hintergrundpixel erkannt und anschließend gleich der Mittelpunkt bestimmt werden kann. Mit einer Laufzeit von $\mathcal{O}(n^2)$ ist dieser Ansatz also äußerst schnell und für spezielle, insbesondere zeitkritische Anwendungen mit bekannter Scanrichtung empfehlenswert.



(a) (b) (c)

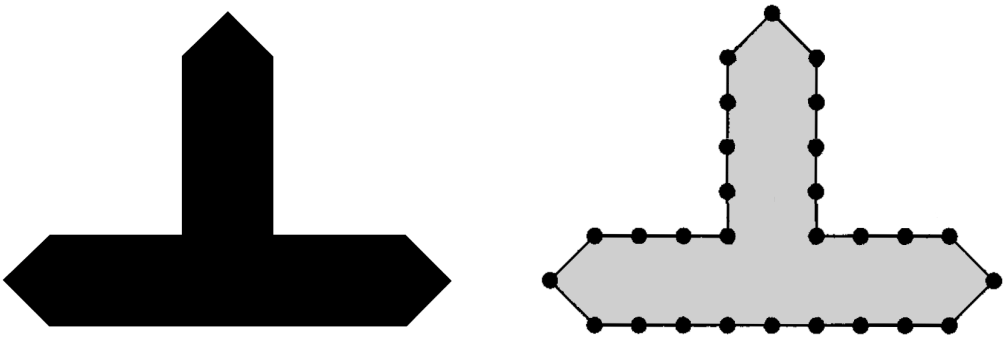
Abbildung 4.6: Für Anwendungen bei denen eine Scanrichtung vorgegeben ist, eignet sich der einfache Kritische-Punkte-Ansatz besonders. Beispielsweise sind auf einem Fließband oftmals die Objekte ausgerichtet. In (a) wird ein solches Szenario gezeigt. Die Objekte wurden vertikal ausgerichtet aufgenommen. Durch eine Binarisierung (b) können die Objekte erfasst werden. In (c) wurden sie skelettiert. So kann beispielsweise die Länge der Objekte sehr einfach bestimmt werden.

4.2 Triangulations-Algorithmus

Ein weitaus flexiblerer, aber auch algorithmisch anspruchsvollerer Ansatz, ist die Ermittlung von kritischen Punkten mit Hilfe von Triangulation. In diesem Abschnitt wird auf die Publikationen von Zou, Yan und Chang eingegangen [8][62][63], die dieses Prinzip vorgestellt haben. Die Skelettierung mit Hilfe von Triangulation wurde im Rahmen dieser Arbeit nicht implementiert. Dies liegt nicht zuletzt daran, dass aus den Publikationen nicht immer hervorgeht, mit welchen Verfahren Zwischenergebnisse erreicht werden können. Aus diesem Grund muss in diesem Abschnitt auf eigene Bilder verzichtet werden. Stattdessen werden überarbeitete Bilder aus den Publikationen verwendet. Es sei betont, dass dieser Abschnitt lediglich einen informellen Charakter haben soll, da eher eine Idee als eine konkrete Umsetzung vorgestellt wird.

Beim Triangulations-Verfahren erfolgt das Errechnen der kritischen Punkte nicht pixelbasiert, sondern beruht auf einem Graphen. Ausgangspunkt ist ein ungerichteter Graph, der die Kontur des Objektes widerspiegelt. Dabei sind die Knoten einzelne Konturpixel und dazwischenliegende Konturabschnitte sollen die Kanten bilden. Abbildung 4.7 zeigt, wie ein solcher Graph aussehen kann.

Solch einen Graphen kann man beispielsweise durch einen Linearisierungsprozess erhalten. Dabei



(a) (b)

Abbildung 4.7: Der erste Schritt bei der Skelettierung mittels Triangulations-Technik ist das Überführen der Kontur des Objektes in einen ungerichteten Graphen. In (b) wird ein solcher Graph für Bild (a) gezeigt. (überarbeitete Bilder aus [63])

wird die Kontur in Streckensegmente unterteilt, deren Endpunkte zu den Knoten des Graphen werden. Die Kontur $C_\alpha(I)$, bestehend aus t Pixeln $\{c_1, c_2, ..., c_t\}$, wird linearisiert, indem eine neue Kontur C' mit t' Knoten $\{v_1, v_2, ..., v_{t'}\}$ gebildet wird. Jeder Knoten v_i $(i = 1, 2, ..., t')$ ist ein Pixel von $C_\alpha(I)$. Die Zuordnung von Pixeln zu Knoten kann beispielsweise durch ein vorher festgelegtes λ und ein sich daraus ergebendes t' wie folgt berechnet werden:

$$v_i := c_{(i-1)\lambda+1} \qquad \text{für } i = 1, 2, ..., t' \tag{4.1}$$

$$t' := \lfloor \frac{t}{\lambda} \rfloor + 1 \tag{4.2}$$

λ entspricht der Anzahl von Pixeln zwischen zwei Knoten v_i und v_{i+1} $(1 \leq i < t')$ und ist ganzzahlig aus dem Intervall $[1, \lfloor \frac{t}{3} \rfloor]$ zu wählen, da für den nächsten Schritt mindestens drei Knoten benötigt werden.

Anschließend wird der erhaltene Graph trianguliert. Das heißt, dem Graphen werden so Kanten hinzugefügt, dass das Objekt in Dreiecke unterteilt wird. Ziel ist eine Delaunay-Triangulation, benannt nach dem russischen Mathematiker Boris Nikolajewitsch Delone, der sich mit Triangulations-Verfahren auseinandergesetzt hat. Die Delaunay-Triangulation hat die Eigenschaft, dass innerhalb des Umkreises eines Dreiecks keine anderen Knoten liegen. Dies hat zur Folge, dass der kleinste Innenwinkel über alle Dreiecke maximiert wird. Zur Berechnung der Delaunay-Triangulation existieren mehrere Methoden: sie kann inkrementell aufgebaut, nach dem "devide and conquer" Prinzip, mittels eines Sweep-Algorithmus oder mit Hilfe von Voronoi-Diagrammen ermittelt werden [31]. Es sei angemerkt, dass normalerweise eine Punktmenge in eine Delaunay-Triangulation überführt wird. Durch die Verwendung eines Graphen als Ausgangspunkt werden bereits Kanten vorgegeben. Eventuell existiert mit diesen Einschränkungen dann keine Delaunay-

Triangulation für die Knotenmenge. Zou/Yan gehen auf diese Problematik in den vorliegenden Publikationen nicht ein.

Abbildung 4.8 zeigt laut [63] eine Delaunay-Triangulation des obigen Beispiels. Es wurden bereits alle Dreiecke verworfen, die nicht innerhalb des Objektes liegen, da diese für die Skelettberechnung keine Rolle spielen.

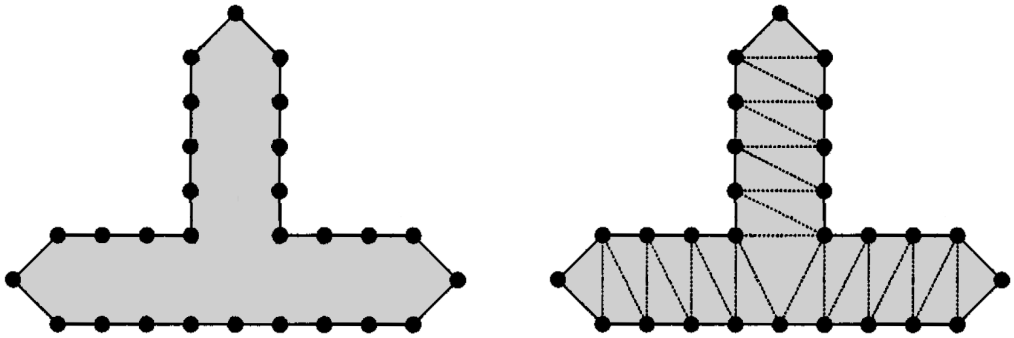


Abbildung 4.8: Schritt zwei der Skelettierung ist die Ermittlung einer Delaunay-Triangulation des Kontur-Graphen. Abbildung (b) zeigt nach [63] eine Delaunay-Triangulation des Graphen aus (a). Alle Dreiecke die nicht innerhalb des Objektes liegen wurden bereits verworfen. (überarbeitete Bilder aus [63])

Die so entstandenen Dreiecke werden unterschieden in:

- **End-Dreiecke:** Dreiecke mit zwei Außenseiten
- **Normale Dreiecke:** Dreiecke mit einer Außenseite
- **Innere Dreiecke:** Dreiecke ohne Außenseite
- **Externe Dreiecke:** Dreiecke mit drei Außenseiten

Anhand der so unterschiedenen Dreiecke können nun kritische Punkte bestimmt werden. Dabei bekommen End-, innere und externe Dreiecke einen kritischen Punkt im Schwerpunkt des Dreiecks zugewiesen. Bei normalen Dreiecken hingegen werden kritische Punkte an den Mittelpunkten aller internen Kanten markiert. Dabei ist eine Kante intern, wenn sie kein Kontursegment ist (siehe Abbildung 4.9 (b)). Anschließend werden kritische Punkte von benachbarten Dreiecken mit Strecken verbunden.

Ein möglicherweise einfacheres Vorgehen, um dieses Skelett zu erhalten, ist es, jedem Dreieck ein eigenes "lokales" Skelett einzuzeichnen. Die Summe der lokalen Skelette aller Dreiecke ergibt dann das gesamte Skelett. Um diese lokalen Skelette zu erhalten, wird bei End-, inneren und externen Dreiecken der Schwerpunkt mit den Mittelpunkten aller internen Kanten verbunden. Bei

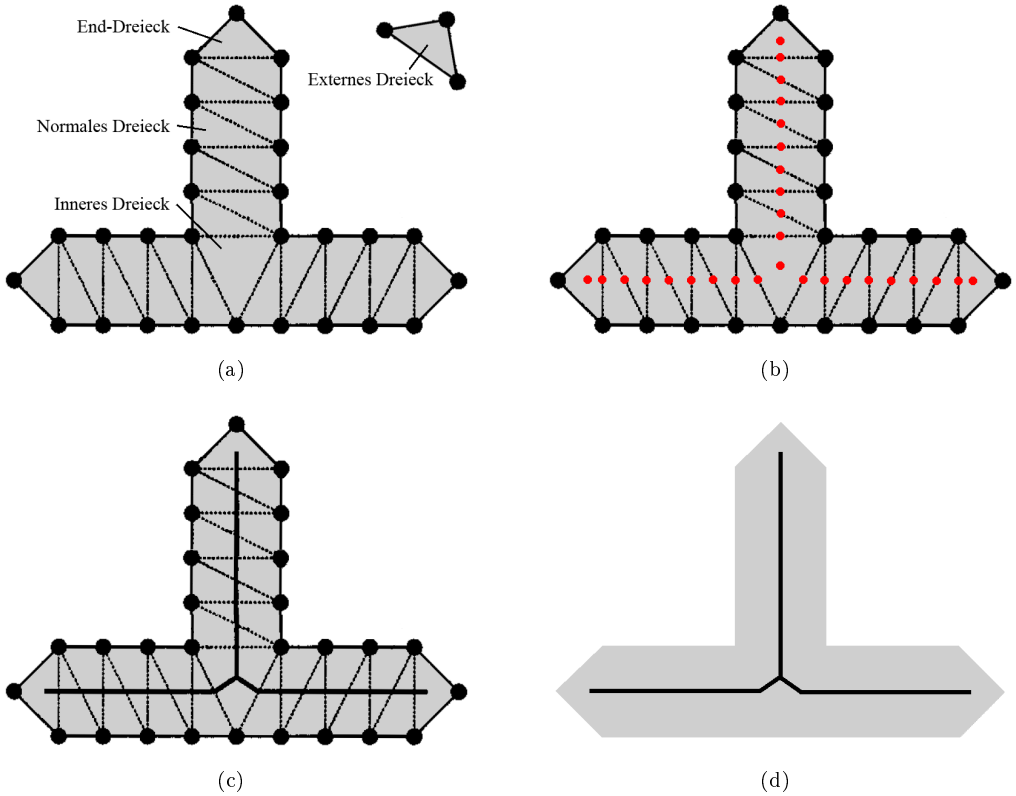


Abbildung 4.9: Die aus der Delaunay-Triangulation entstanden Dreiecke können nun kategorisiert werden (a). Je nach Kategorie eines Dreiecks werden für diese kritische Punkte im Schwerpunkt oder den Mittelpunkten von inneren Kanten bestimmt (b), um diese dann im nächsten Schritt zu verbinden (c). In (d) ist das fertige Skelett zu sehen. (überarbeitete Bilder aus [63])

normalen Dreiecken werden die beiden Mittelpunkte der internen Kanten verbunden. Abbildung 4.9 (c) zeigt das Resultat dieses Vorgehens.

Zou und Yan schlagen in ihrer Publikation [63] noch weitere Verbesserungen vor, um unnötige Verzweigungen/Äste zu verhindern. Dabei findet eine Einteilung in stabile und instabile Regionen statt, welche allerdings auf einer fraglichen Definition beruht:

> A singular region is *stable* if it conforms to human perceptions. Otherwise, it is *unstable*. [63]

Diese Definition wirft zwei Fragen auf. Zum einen ist der Begriff einer Region unklar, da er von Zou/Yan nicht definiert wurde. So geht der Autor im Folgenden davon aus, dass eine Region ein Zusammenschluss von mehreren Dreiecken zu einem Polygon ist. Offen bleibt dann trotzdem, welche der Dreiecke eine Region bilden sollen. Zum anderen stellt sich die Frage, in welcher Art und Weise instabile Regionen detektiert werden. Möglich wäre die Erstellung einer Oberfläche,

bei der der Anwender instabile Regionen markieren kann. Diese Vorgehensweise stellt aber die, laut Zou/Yan, Hauptanwendung zur optischen Zeichenerkennung in Frage.

Singulare Regionen besitzen nach Zou/Yan keine, eine oder mehr als zwei interne Kanten. Instabile singulare Regionen benötigen eine Stabilisierung.

Eine End-Region (Region mit einer inneren Kante) gilt als instabil und wird gelöscht, wenn der Vorsprung den eine solche Region bildet (siehe Abbildung 4.10 (c)), nicht als eigenes "Band" erkennbar ist. In Abbildung 4.10 (a) ist r_1 der kleinste mögliche Vorsprung, der als eigenes Band erkannt wird. Kleinere Regionen R, die kein Punkt außerhalb des eingezeichneten Kreises U besitzen (Abbildung (c)), werden nicht erkannt.

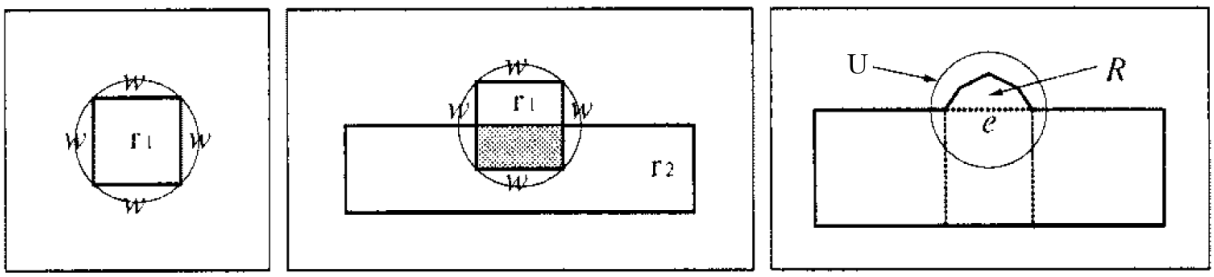


Abbildung 4.10: Eine End-Region ist nach Zou/Yan instabil, wenn sie nicht als eigenes "Band" erkennbar ist. (a),(b) zeigt die kleinstmögliche End-Region r_1 mit dem Radius w des Umkreises U, die noch als eigenes "Band" erkennbar wäre. Kleinere Regionen R, von denen kein Pixel außerhalb des Umkreises U von r_1 liegt, werden als instabil eingestuft und gelöscht (c). w ergibt sich dabei aus der Länge der inneren Kante der angrenzenden Region. (überarbeitete Bilder aus [63])

Eine für geometrische Eigenschaften des Skelettes wichtige Korrektur ist die Stabilisierung von inneren Regionen (Regionen mit mehr als zwei inneren Kanten). Innere Regionen sind nach Zou/Yan oft instabil, wenn mehrere innere Dreiecke in Schnittpunkten von Bändern vorkommen (siehe Abbildung 4.11). Um keine verzweigten und unnatürlich aussehenden Skelette zu erhalten, soll für solch eine Regionen ein einziger kritischer Punkt gefunden werden.

In einer inneren Region R, in der m Regionen r_i "aufeinander treffen", werden m Geraden l_i betrachtet, die die lokalen Skelette s_i annähern sollen ($1 \leq i \leq m$). Auch hier bleiben Zou/Yan dem Leser schuldig, wie diese Annäherungen umgesetzt werden sollen. Als kritischer Punkt P wird dann derjenige Punkt gewählt, dessen Summe der Abstände zu den Geraden l_i minimal ist. Auf diese Art und Weise wird für eine solche innere Region ein einziger kritischer Punkt festgelegt. Diese Stabilisierung bewirkt, dass klare Schnittpunkte entstehen und geometrische Eigenschaften, wie Winkel, Richtungen oder Längen von Zweigen, die zum Beispiel bei der Zeichenerkennung wichtig sind, besser wiedergegeben werden. Auch bleibt die Anzahl der Verzweigungen durch die Stabilisierung im Vergleich zum Original dieselbe. Dies wird an den Beispielen von Abbildung 4.12 deutlich, wo die Skelettierung mittels Triangulation dem Ausdünnen (siehe Abschnitt 5) ge-

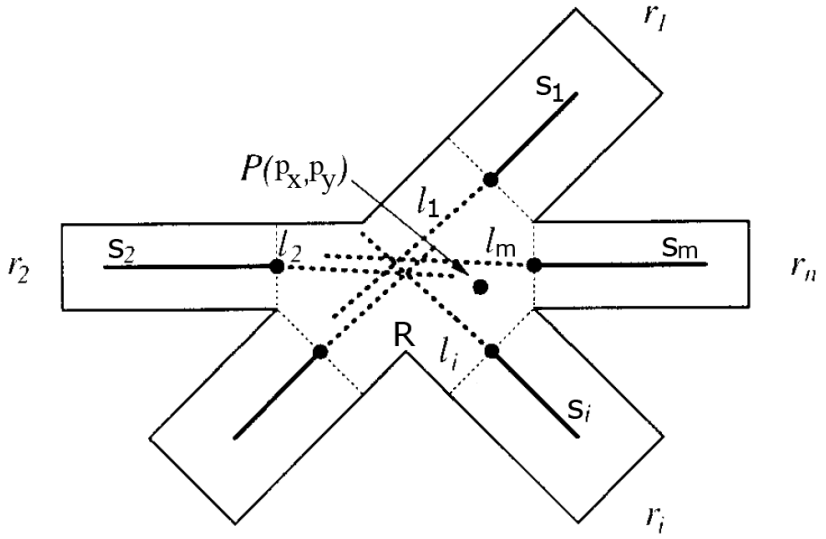


Abbildung 4.11: Eine instabile innere Region R benötigt nach Zou/Yan eine Stabilisierung. Dazu werden die Geraden l_i betrachtet, welche die lokalen Skelette s_i der angrenzenden Regionen r_i annähern sollen ($1 \leq i \leq m$). Als einziger kritischer Punkt einer instabilen Region wird dann derjenige Punkt P gewählt, dessen Summe der Abstände zu den Geraden am geringsten ist. (überarbeitetes Bild aus [63])

genübergestellt wird. Bei den oft verwendeten Ausdünnungs-Algorithmen wird beispielsweise der Buchstabe V bei dicker Schriftbreite zum Y. Dies lässt sich mit der Triangulations-Skelettierung und anschließender Stabilisierung vermeiden. Es wird deutlich, dass allgemein die Schnittpunkte gegenüber dem Ausdünnen besser erkannt werden.

Noch einmal sei erwähnt, dass die Triangulations-Skelettierung im Rahmen dieser Arbeit nicht implementiert wurde, da einige wichtige Fragen zur Umsetzung offen blieben. So musste auf Resultate von Zou/Yan/Chang zurückgegriffen werden, die nicht überprüft werden konnten. Auch ist eine komplette Laufzeitanalyse nicht möglich, da nicht alle Teilverfahren bekannt sind. Eine Linearisierung einer Kontur mit t Pixeln ist in $\mathcal{O}(t)$ berechenbar. Die Delaunay-Triangulation kann für t' Knoten in $\mathcal{O}(t' \log t')$ durchgeführt werden [31]. Offen bleibt der Aufwand für die Stabilisierung. Jedoch wird in [62] der Aufwand für die Triangulation als ausschlaggebend bezeichnet, was vermuten lässt, dass der Gesamtaufwand mit $\mathcal{O}(t' \log t')$ angegeben werden kann. Das heißt, dass die Anzahl der Knoten t' und damit die Länge λ der Kontursegmente bei der Linearisierung wichtige Faktoren für die Geschwindigkeit dieses Verfahrens sind. Bestimmt man diese Länge der Kontursegmente dynamisch, indem man nach Zou/Yan die durchschnittliche Breite w der Objekte in Pixeln heranzieht (w bleibt durch Zou/Yan ebenfalls undefiniert), so kann die Anzahl der Knoten bei unterschiedlichen Auflösungen relativ konstant gehalten werden.

$$\lambda = \lfloor \xi \cdot w + 1 \rfloor \qquad \xi \in [0,2; 0,7] \tag{4.3}$$

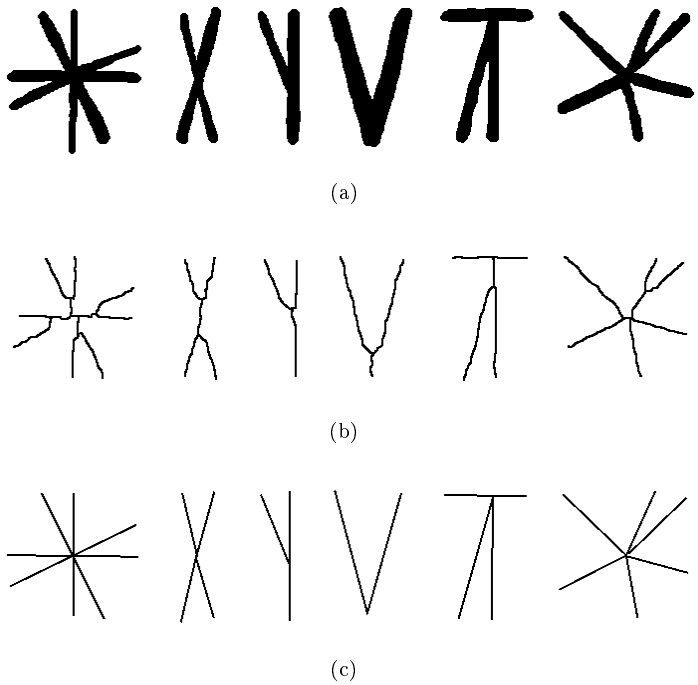


Abbildung 4.12: Durch die Stabilisierung von instabilen Regionen werden klare und eindeutige Schnittpunkte gebildet. Dies wird im Vergleich zu Ausdünnungs-Algorithmen deutlich. In (b) wurde das Bild (a) mit dem Zhang/Suen-Algorithmus ausgedünnt (siehe Abschnitt 5.3). In (c) wurde die Triangulations-Technik angewandt. Es ist erkennbar, dass in (c) deutliche Schnittpunkte entstehen und nach menschlichem Ermessen bessere Resultate erzielt werden. (überarbeitete Bilder aus [63])

ξ ist dabei als noch festzulegender konstanter Faktor aus dem Intervall $[0,2;0,7]$ zu wählen. Dies bedeutet, dass die Geschwindigkeit der Skelettierung mit Hilfe der Triangulation unabhängig von der Größe und Auflösung des Bildes sein kann. Dies wiederum ist ein entscheidender Vorteil gegenüber pixelbasierten Skelettierungsalgorithmen, deren Laufzeit mit steigender Pixelanzahl in der Regel stark zunimmt (siehe auch Anhang B).

Um *Impulsrauschen (Salt-and-Pepper-Rauschen)* entgegen zu wirken, werden Konturen, die weniger als eine bestimmte Anzahl an Pixeln umfassen, im Linearisierungsprozess ignoriert. Dieser Schwellenwert kann ebenfalls von der Länge der Kontursegmente abhängig gemacht werden, um sich an die Auflösung des Bildes anzupassen.

In Pseudocode 6 wurde das Vorgehen beim Skelettieren mittels Triangulation zusammengefasst. Leider muss auch hier auf einige konkrete Definitionen, wie beispielsweise die der durchschnittlichen Breite der Objekte oder die des Begriffes der instabilen Regionen, verzichtet werden, sodass der Pseudocode eher als genereller Ablauf als eine konkrete Implementationsanweisung zu verstehen ist.

Mit den genannten Vorteilen stellt die Skelettierung mit Hilfe der Triangulation eine robuste und schnelle Alternative zu den oft genutzten Ausdünnungs-Algorithmen dar. Die Anwendung

Pseudocode 6 : Skelettierung mittels Triangulation

Eingabe : Bild I
Ausgabe : Skelett S /* Menge von Strecken */

1 $\xi \leftarrow$ Konstante aus $[0,2;0,7]$
2 $w \leftarrow$ durchschnittliche Breite der Objekte aus I
3 $\lambda \leftarrow \lfloor \xi \cdot w + 1 \rfloor$ /* Länge der Kontursegmente */
4 $C' \leftarrow$ Graph aus Linearisierung von $C_\alpha(I)$ mit λ
5 $G \leftarrow$ Delaunay-Triangulation von C' unter Berücksichtigung vorhandener Kanten /* Menge von $\triangle$ */
6 $G \leftarrow$ Menge aller $\triangle$ aus G die innerhalb der Objekte aus I liegen
7 $R \leftarrow$ Menge aller Regionen (durch Zusammenfassen der $\triangle$ aus G zu Polygonen)
8 $IR \leftarrow$ Menge aller instabilen Regionen aus R
9 $G \leftarrow G \backslash$(Menge aller $\triangle$ der Polygone aus IR)
10 $IR \leftarrow IR \backslash$(Menge aller End-Regionen aus IR)
11 $S_G \leftarrow$ Menge aller lokalen Skelette aller $\triangle$ aus G /* Menge von Strecken */
12 $S_{IR} \leftarrow$ Menge aller lokalen Skelette aller Polygone aus IR durch Stabilisierung /* Menge von Strecken */
13 $S \leftarrow S_G \cup S_{IR}$

wird von Zou/Yan aber auf "bandartige" Objekte beschränkt. Der Autor geht davon aus, dass damit Objekte gemeint sind, welche sich aus mehreren Strichen/Linien zusammensetzen, wie etwa Buchstaben oder Zahlen. Damit ist dieses Skelettierungsverfahren besonders für die optische Zeichenerkennung geeignet. Allerdings bleiben, wie bereits mehrfach erwähnt, Fragen zur Implementierung offen.

Abbildung 4.13 zeigt weitere Beispiele aus [62] und [63].

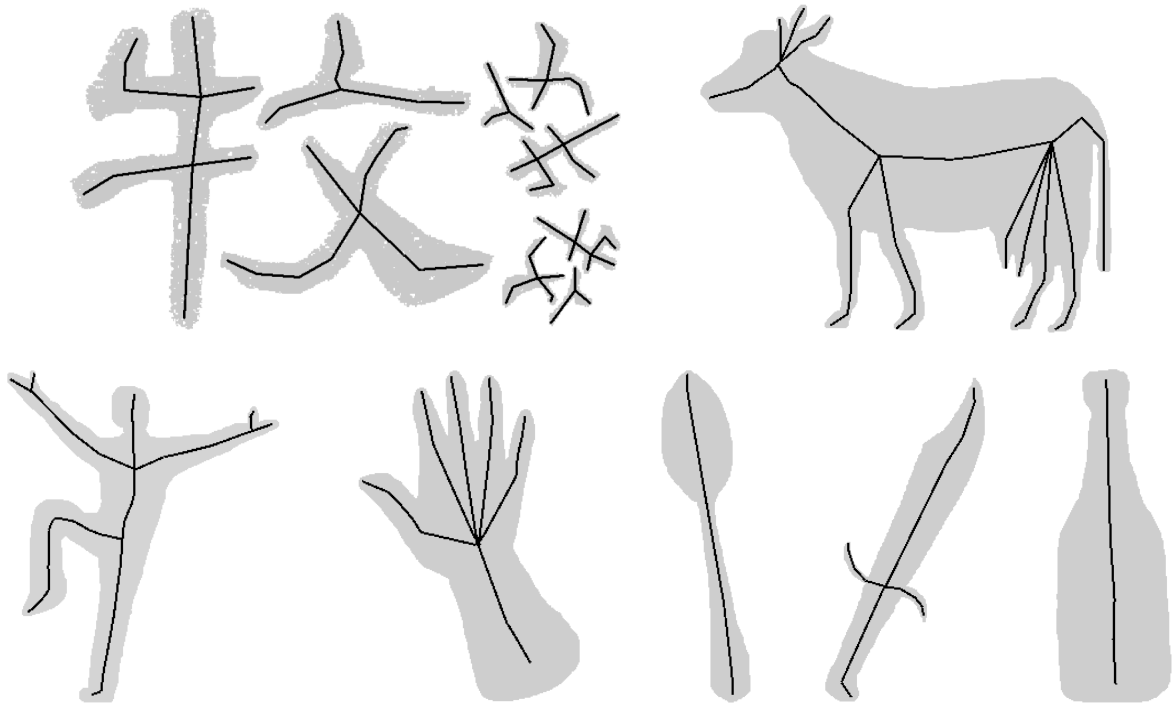


Abbildung 4.13: Diese Beispiele weiterer Triangulations-Skelette wurden aus [63] übernommen. Die Skelette wirken gegenüber Skeletten anderer Algorithmen, wie beispielsweise Ausdünnungs-Skeletten, strukturierter und deutlicher.

5 Ausdünnungs-Algorithmen (Thinning)

Das sukzessive Ausdünnen von Objekten und das damit verbundene Freistellen eines Skelettes ist sicher das bekannteste Vorgehen, um ein Skelett zu berechnen. Insbesondere Arcelli [3], Rosenfeld [39][45][46][52][53], Suen [30][41][58], Wang [30] [34][56][59][60][61] und Zhang [56][58][59][60][61] haben sich in den 60er bis 80er Jahren intensiv mit Ausdünnungs-Algorithmen beschäftigt. Die Anzahl neuer Publikationen zu diesem Thema hat in den letzten 20 Jahren stark abgenommen. Dies liegt vermutlich auch daran, dass es mit immer schnelleren Rechnern keine Notwendigkeit mehr gibt, Publikationen zu veröffentlichen, in denen nur sehr kleine Änderungen vorgeschlagen werden, um minimal Rechenzeit einzusparen. So folgten beispielsweise auf den heutigen "Standard-Ausdünnungs-Algorithmus"[1] der 1984 in "A Fast Parallel Algorithm for Thinning Digital Patterns" [58] von Zhang/Suen vorgestellt wurde Publikationen, wie "A Comment on 'A Fast Parallel Algorithm for Thinning Digital Patterns"' [34], "An Improved Parallel Thinning Algorithm" [23] oder "A Modified Parallel Thinning Algorithm" [59], in denen nur sehr kleine Modifikationen vorgeschlagen wurden.

Das Ausdünnen von Objekten ist ein iterativer Prozess, bei dem in jedem Durchgang die Konturpixel der Objekte daraufhin untersucht werden, ob Pixel gelöscht (d.h. auf die Hintergrundfarbe gesetzt) werden können, ohne die *Zusammengehörigkeit (connectedness)* der Objekte zu verändern.

Eine Region $\mathcal{R} \subseteq I$ wird als *4-zusammengehörig* bezeichnet, wenn für alle Paare p, q von Vordergrundpixeln aus $\mathcal{R}$ eine Folge $p = p_0, p_1, p_2, ..., p_n = q$ existiert, wobei $p_i \in \mathcal{R} \wedge p_i \in \langle I \rangle$ und p_i 4-adjazent zu p_{i+1} ist $(0 \leq i < n)$.

Analog dazu wird eine Region $\mathcal{R} \subseteq I$ als *8-zusammengehörig* bezeichnet, wenn für alle Paare p, q von Vordergrundpixeln aus $\mathcal{R}$ eine Folge $p = p_0, p_1, p_2, ..., p_n = q$ existiert, wobei $p_i \in \mathcal{R} \wedge p_i \in \langle I \rangle$ und p_i 8-adjazent zu p_{i+1} ist $(0 \leq i < n)$.

Da beim Ausdünnen die Zusammengehörigkeit von Objekten nicht verändert wird, sind die entstehenden Skelette topologisch äquivalent zu den Objekten, weisen also dieselben topologischen Merkmale wie die Objekte selbst auf (siehe auch Abschnitt 2.3). Eben dieser Erhalt der topologischen Struktur steht bei den Ausdünnungs-Algorithmen im Vordergrund und so unterscheiden sich die meisten Algorithmen nur darin, in welcher Art und Weise die Konturpixel der Objekte

[1] Dieser wurde in viele Standardwerke übernommen, u.a.: Gonzales/Woods [20] und Hussain [24].

untersucht werden und ob eine Löschung die Zusammengehörigkeit der Objekte ändert oder nicht. Pixel die demnach gelöscht werden können, werden *simple Pixel* genannt. Die Entscheidung, ob ein Konturpixel ein simples Pixel ist, kann in der Regel nur mit Hilfe der Nachbarschaftspixel entschieden werden. Pseudocode 7 zeigt stark vereinfacht das Vorgehen beim Ausdünnen. Der Hauptunterschied der verschiedenen Algorithmen liegt in der Umsetzung der Funktion *istSimpel(p)* (Zeile 6). Diese Funktion beinhaltet im Allgemeinen auch die Einschränkung nur Konturpixel zu betrachten.

Pseudocode 7 : Ausdünnungs-Algorithmus

```
   Eingabe : Bild ⟨I⟩                                  /* Menge aller Objektpixel */
   Ausgabe : Skelett ⟨S⟩                               /* Menge aller Skelettpixel */
1  ⟨S⟩ ← Kopie von ⟨I⟩
2  a ← |⟨S⟩| + 1                                      /* Anzahl der Elemente von ⟨S⟩ + 1 */
3  solange |⟨S⟩| < a tue
4  |  a ← |⟨S⟩|
5  |  für alle Pixel p aus ⟨S⟩ tue
6  |  |  wenn istSimpel(p) dann
7  |  |  |  ⟨S⟩ ← ⟨S⟩\{p}
```

Das Untersuchen der Kontur und Löschen simpler Pixel wird so lange durchgeführt, bis sich keine weiteren Veränderungen mehr ergeben (Zeile 3). Das resultierende Skelett hat dann im Idealfall eine Breite von einem Pixel (*ideal thin*). Jedoch haben insbesondere *sequentielle* Ausdünnungs-Algorithmen Schwierigkeiten, alle simplen Konturpixel zu finden. Das führt dazu, dass Skelette oftmals mehrere Pixel breit sind und deshalb, falls eine Ein-Pixel-Breite gefordert wird, eine Nachbehandlung nötig ist. In Abbildung 5.1 wird das Ausdünnen eines Objektes auf Ein-Pixel-Breite veranschaulicht. Eine Iteration entspricht dabei einem Durchlauf der Schleife aus Zeile 3.

Die Vorgehen, um simple Konturpixel zu erkennen und zu markieren, können in zwei Klassen eingeteilt werden:

- **sequentielles Ausdünnen:** Die Entscheidung, ob ein Pixel in der n-ten Iteration gelöscht werden kann, ist nicht nur von dem Zustand des Bildes nach der letzten, also der (n-1)-ten Iteration abhängig, sondern auch von allen bereits in der n-ten Iteration getätigten Veränderungen. Die Reihenfolge der Abarbeitung innerhalb einer Iteration ist also entscheidend und wird entweder in Rasterscan-Reihenfolge oder mit Hilfe von Konturverfolgungs-Algorithmen festgelegt.
- **paralleles Ausdünnen:** Hier wird als Grundlage der Entscheidung, ob ein Pixel gelöscht werden kann, nur das Bild nach der (n-1)-ten Iteration herangezogen. Dies bedeutet, dass

alle Pixel unabhängig voneinander, also parallel bearbeitet werden können. Allerdings wird bei parallelen Ausdünnungs-Algorithmen oft mit Unteriterationen gearbeitet.

Bevor diese beiden Arten des Ausdünnens näher betrachtet werden, soll zunächst auf ein dem Ausdünnen sehr ähnliches Verfahren, die morphologische Skelettberechnung, eingegangen werden.

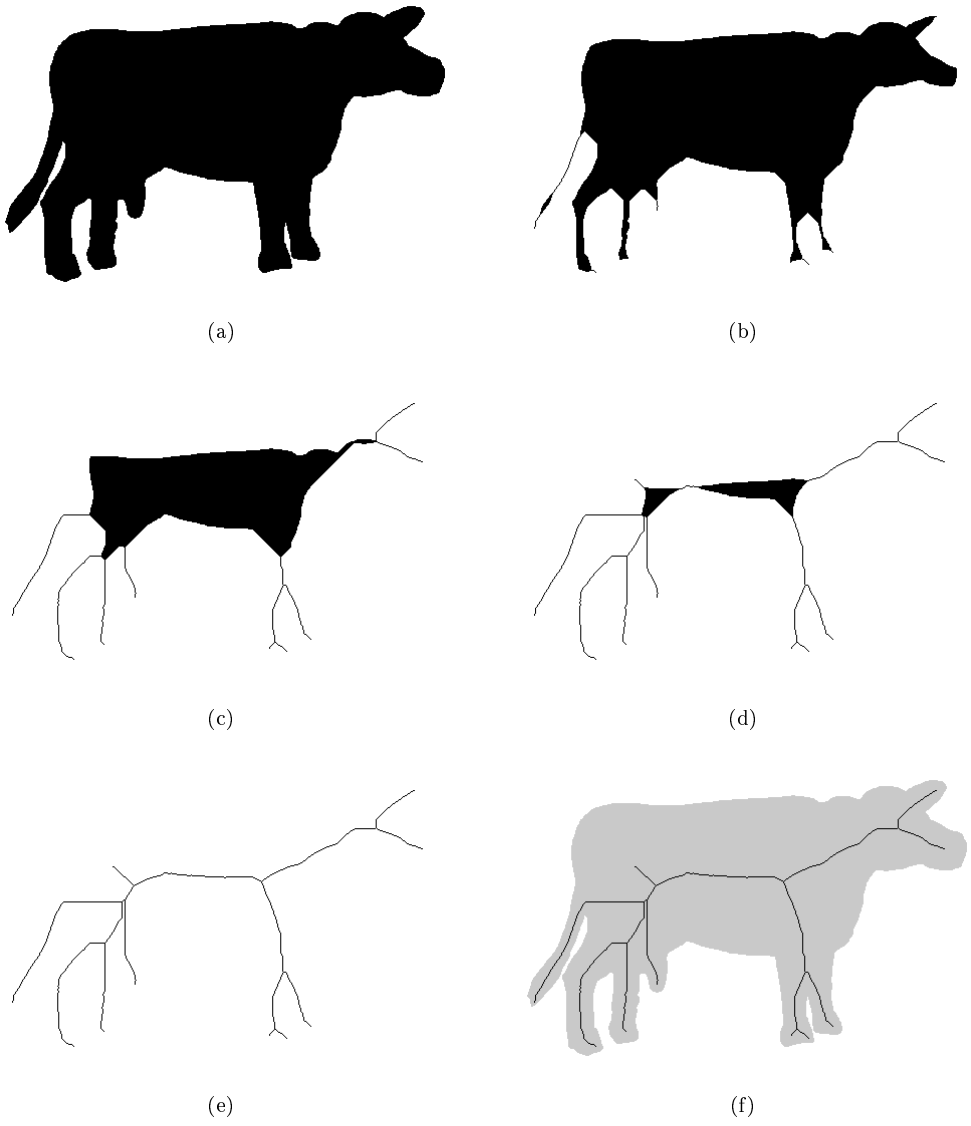


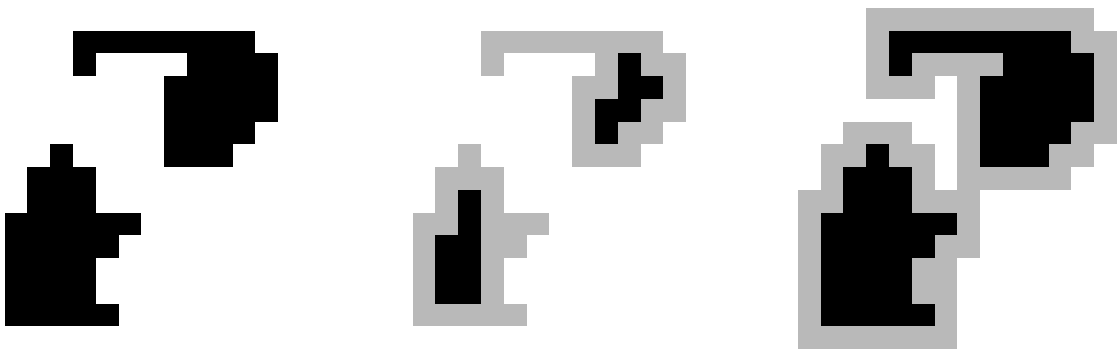
Abbildung 5.1: Beim Ausdünnen wird in mehreren Iterationen die Kontur der Objekte auf simple Pixel untersucht, um diese zu löschen. In (b)-(e) wurde das Original (a) [BILD 20] auf Ein-Pixel-Breite ausgedünnt. (b) zeigt das Zwischenergebnis nach 10 Iterationen, (c) nach 40 Iterationen, (d) nach 70 Iterationen und (e) das fertige Skelett. In (f) wurde das Skelett dem Original eingezeichnet.

5.1 Morphologisches Ausdünnen

Die morphologische Skelettberechnung ist kein klassischer Ausdünnungs-Algorithmus, arbeitet aber ebenfalls iterativ mit den Konturpixeln und soll deshalb hier behandelt werden. Das morphologische Skelett wird mit Hilfe der klassischen morphologischen Grundoperationen *Erosion* und *Dilatation* berechnet. Beide Operationen können als nicht-lineares Filter wie folgt definiert werden:

- **Erosion:** Das Pixel unter dem Bezugspixel der Maske bekommt als neuen Wert das Minimum aller unter der Maske liegenden Pixel. Schreibweise: $A \ominus B$ (Anwendung der Erosion auf das Bild A mit der Maske B).
- **Dilatation:** Das Pixel unter dem Bezugspixel der Maske bekommt als neuen Wert das Maximum aller unter der Maske liegenden Pixel. Schreibweise: $A \oplus B$ (Anwendung der Dilatation auf das Bild A mit der Maske B).

Einfach gesagt, werden bei der Erosion alle Vordergrundpixel gelöscht, die innerhalb der Maske ein Hintergrundpixel haben. Wird beispielsweise als Maske die 8er-Nachbarschaft gewählt, werden alle Pixel von $C_8(I)$ gelöscht. Die Objekte "schrumpfen" also. Bei der Dilatation werden alle Hintergrundpixel, die innerhalb der Maske ein Vordergrundpixel haben, zu Vordergrundpixeln. Für die 8er-Nachbarschaft als Maske bedeutet das, dass alle zur Kontur $C_8(I)$ 8-adjazenten Hintergrundpixel zu Vordergrundpixeln werden. Die Objekte "wachsen" also. Abbildung 5.2 zeigt eine Erosion und Dilatation auf ein Beispielbild mit der Maske der 8er-Nachbarschaft. Es ist zu erkennen, dass die Anwendung von Erosion und Dilatation die Zusammengehörigkeit der Objekte ändern kann.



(a) (b) (c)

Abbildung 5.2: Die morphologische Skelettberechnung basiert auf den morphologischen Operationen Erosion und Dilatation. In (b) wird eine Erosion auf (a) ausgeführt. Dabei werden die grauen Pixel gelöscht. In (c) wird eine Dilatation auf Bild (a) ausgeführt. Dabei wird das Objekt um die grauen Pixel erweitert. Zur Anwendung kam jeweils die Maske der 8er-Nachbarschaft.

Die Operationen Erosion und Dilatation sind nicht invers zueinander, obwohl sie teilweise eine gegensätzliche Wirkung haben. Im Prinzip kann eine Dilatation eine Erosion rückgängig machen, jedoch nur, wenn nach der Erosion noch genug Objektpixel vorhanden sind aus denen das Objekt wieder "wachsen" kann. Es kann nämlich passieren, dass Objekte vollständig gelöscht werden und dann kann auch mit einer Dilatation kein Wachsen mehr bewirkt werden. Genau diese Tatsache wird bei der morphologischen Skelettberechnung ausgenutzt. Vereinfacht gesagt, wird dabei auf das Bild solange die Erosion angewandt, bis die Veränderungen nicht mehr durch eine Dilatation rückgängig gemacht werden können. Die Pixel, die nicht mehr hergestellt werden können, werden als elementar angesehen und zur Menge der Skelettpunkte gezählt. Nach jeder Erosion wird also eine Dilatation durchgeführt, um das entstandene Bild mit dem Bild vor der Erosion zu vergleichen und Veränderungen festzustellen.

In Mengenschreibweise kann das Skelett S wie folgt beschrieben werden [20]:

$$S := \bigcup_{k=0}^{K} ((I \ominus kB) - ((I \ominus kB) \circ B)) \tag{5.1}$$

$I \ominus kB$ steht für die k-fache Hintereinanderausführung der Erosion mit der Maske B auf das Bild I.

$$I \ominus kB := \underbrace{((...(I \ominus B) \ominus B)...) \ominus B}_{k \text{ mal}} \tag{5.2}$$

$A \circ B$ ist eine Erosion und anschließende Dilatation auf das Bild A, jeweils mit der Maske B. Diese Hintereinanderausführung von Erosion und Dilatation wird auch *Opening* genannt.

$$A \circ B := (A \ominus B) \oplus B \tag{5.3}$$

Gleichung 5.1 kann also als iteratives Vorgehen implementiert werden. Der Vorgang bricht erst dann ab, wenn alle Pixel durch das Abtragen der Erosion verschwunden sind. Es gilt also:

$$K = max\{k \,|\, (I \ominus kB) \neq \varnothing\} \tag{5.4}$$

In Pseudocode 8 ist dieses Vorgehen noch einmal in einem Algorithmus dargestellt.

Pseudocode 8 : morphologisches Ausdünnen

Eingabe : Bild I
Ausgabe : Skelett $\langle S \rangle$ `/* Menge aller Skelettpixel */`

```
1 U ← Kopie von I
2 ⟨S⟩ ← ∅
3 solange ⟨U⟩ ≠ ∅ tue
4 |   T ← dilatation(erosion(U))          /* Opening-Operation */
5 |   ⟨S⟩ ← ⟨S⟩ ∪ (⟨U⟩\⟨T⟩)
6 |   U ← erosion(U)
```

In Abbildung 5.3 ist die Ermittlung skelettaler Punkte nach k Erosionen veranschaulicht. Bild (a) zeigt das Bild nach k Iterationen, die Bilder (b) bis (e) ein Opening an Bild (a). Nun werden Bild (a) und (e) verglichen. Alle Pixel, die durch das Opening "verloren" gegangen sind, gehören zu den skelettalen Punkten (Bild (g)).

Da Erosion und Dilatation keine "Rücksicht" auf die Zusammengehörigkeit von Objekten nehmen, spiegelt das morphologische Skelett, anders als die sonst üblichen Ausdünnungs-Algorithmen (siehe Abschnitt 5.2 und 5.3), nicht die topologische Struktur der Objekte wider. Auch sind die Skelette in der Regel nicht *ideal thin*, das heißt, es kann keine Ein-Pixel-Breite garantiert werden. Im Gegensatz zu den anderen Ausdünnungs-Algorithmen ist das morphologische Skelett für eine Maske B aber eindeutig definiert (siehe Gleichung 5.1). Sind zudem die einzelnen Teilmengen S_k des Skelettes S bekannt, kann das ursprüngliche Objekt rekonstruiert werden.

$$S_k := (I \ominus kB) - ((I \ominus kB) \circ B) \tag{5.5}$$

Dazu werden die Teilmengen wieder sukzessive durch Dilatationen aufgebaut. Die Menge aller Teilmengen ergibt dann das Bild I:

$$I = \bigcup_{k=0}^{K} (S_k \oplus kB)) \tag{5.6}$$

$$S_k \oplus kB := \underbrace{((...(S_k \oplus B) \oplus B)...) \oplus B}_{k \text{ mal}} \tag{5.7}$$

Das morphologische Skelett kann in $\mathcal{O}(n^3)$ berechnet werden. Allerdings kann die tatsächliche Rechenzeit mit unterschiedlich großen Objekten und damit einer unterschiedlichen Anzahl K an Iterationen stark schwanken. Mit K Iterationen müssen K Erosionen und K Dilatationen

ausgeführt werden. In Pseudocode 8 wird lediglich zugunsten der Lesbarkeit in Zeile 4 und 6 dieselbe Erosion durchgeführt. Eine genauere Abschätzung ist $\mathcal{O}(2K * n^2)$.

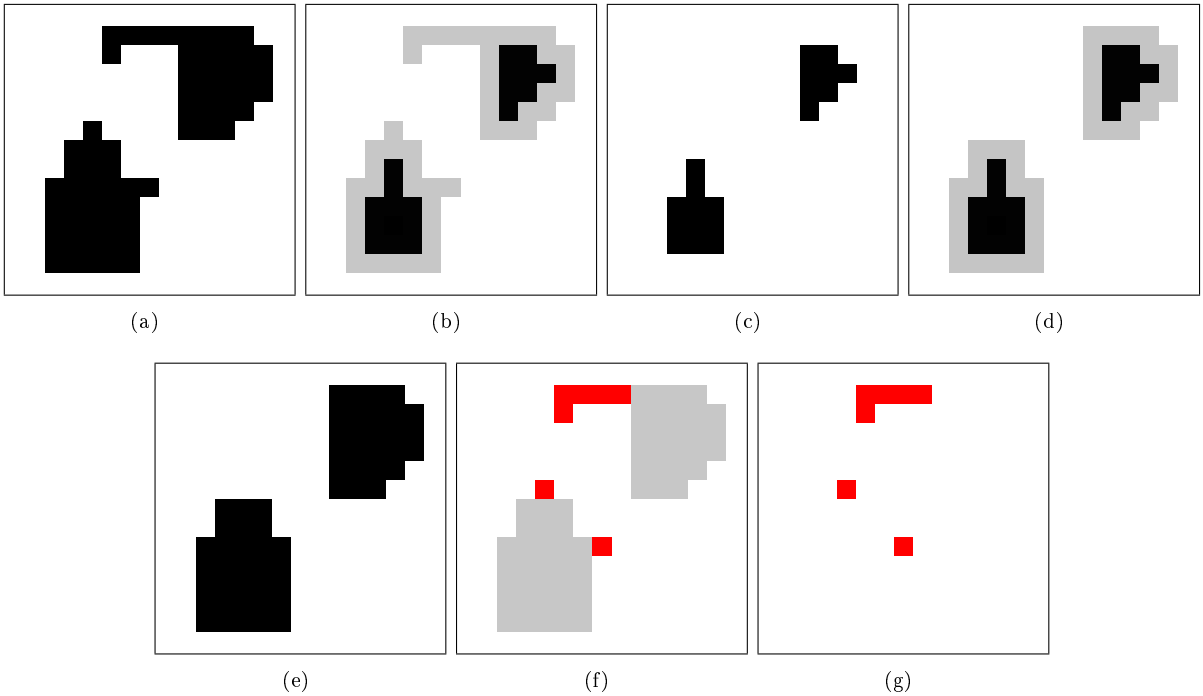


Abbildung 5.3: Beim morphologischen Ausdünnen sind Punkte skelettal, wenn sie bei einer Opening-Operation "verschwinden". Dies soll an einem Beispiel veranschaulicht werden: (a) ist das Ausgangsbild nach k Erosionen. In Bild (b) wird auf (a) eine weitere Erosion durchgeführt. (c) zeigt das Ergebnis. Danach wird in (d) eine Dilatation auf (c) durchgeführt. Das Resultat zeigt (e). (e) ist nun das Ergebnis von *dilatation(erosion((a)))*, also einer Opening-Operation angewendet auf (a). Nun werden in (f) Bild (a) und (e) verglichen. Alle Punkte, die aus (a) verschwunden sind, gehören zu den skelettalen Punkten (g).

Abschließend werden in den Abbildungen 5.4 und 5.5 weitere Bilder gezeigt, für die eine morphologische Skelettberechnung durchgeführt wurde.

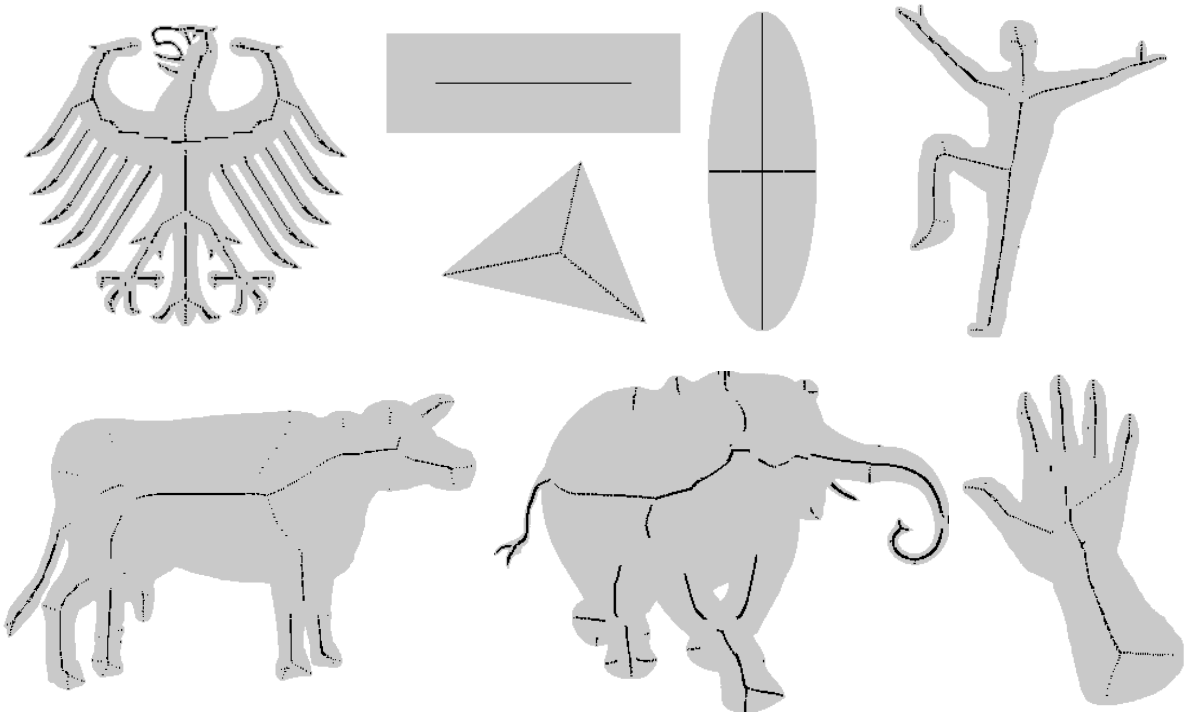


Abbildung 5.4: Weitere im Rahmen dieser Arbeit erstellte Beispiele morphologischer Skelette: von links nach rechts, von oben nach unten: [BILD 5,11,23,20,6,12].

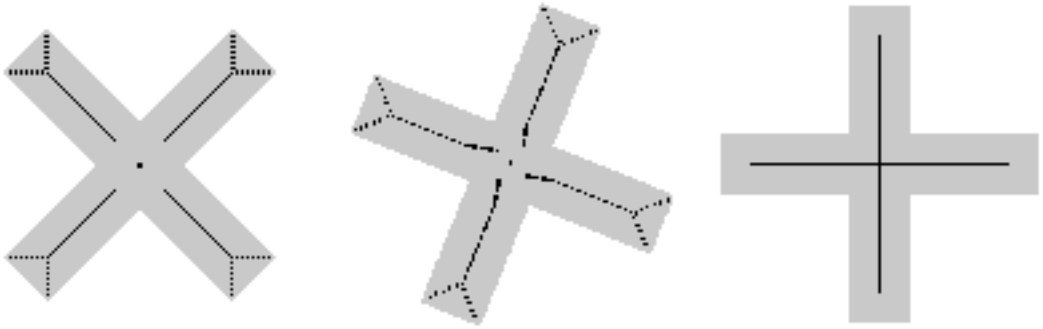


Abbildung 5.5: Auch die morphologische Skelettberechnung ist nicht isotrop. In (a)-(c) wurde ein Kreuz um jeweils 22,5° nach links gedreht [BILD 17,18,19]. Auffällig sind Ähnlichkeiten zum Distanz-Skelett (siehe Abbildung 3.10). Hier liefert Bild (c) ein nach menschlicher Bewertung optimales Ergebnis. Die Skelette aus Bild (a) und (b) sind sich ebenfalls ähnlich.

5.2 Sequentielles Ausdünnen

Beim sequentiellen Ausdünnen werden die Pixel in einer vorgegebenen Reihenfolge daraufhin untersucht, ob sie simple Pixel und damit löschbar sind. Dafür bietet sich zum einen die Rasterscan-

Reihenfolge an und zum anderen eine Reihenfolge, festgelegt durch Konturverfolgungs-Algorithmen. Letzteres hat den Vorteil, dass nur Konturpixel betrachtet werden müssen und damit die Abarbeitung einer Iteration schneller ist. Die Pixel werden, wenn sie als simple Pixel erkannt wurden, nicht sofort gelöscht, sondern nur markiert. Erst am Ende einer Iteration werden alle markierten Pixel gelöscht. Dies stellt sicher, dass nur Konturpixel gelöscht werden und verhindert, dass ganze Äste in einer Iteration verschwinden bzw. Löcher entstehen. Den prinzipiellen Ablauf beim sequentiellen Ausdünnen gibt Pseudocode 9 an.

Pseudocode 9 : sequentieller Ausdünnungs-Algorithmus

Eingabe : Bild I
Ausgabe : Skelett S

```
1  S ← Kopie von I
2  M ← ∅                                          /* Menge markierter Pixel */
3  v ← wahr                                       /* Flag zum Abbrechen des Algorithmus */
4  solange v = wahr tue
5      v ← falsch
6      für alle Pixel p aus S tue                 /* feste Reihenfolge der Abarbeitung */
7          wenn istSimpel(p,S,M) dann
8              M ← M ∪ {p}
9      für alle Pixel p aus M tue
10         S(p) ← 0                               /* setze p in S auf Hintergrundfarbe */
11     wenn M ≠ ∅ dann
12         v ← wahr
13     M ← ∅
```

Ob ein Pixel simpel ist, kann nur mit Hilfe der 8er-Nachbarschaft bestimmt werden. Dazu muss ein Pixel p folgende Mindestvoraussetzungen haben:

- p muss ein Objektpixel sein, d.h. $p \in \langle I \rangle$
- p darf kein Endpunkt bzw. isoliertes Pixel sein, d.h. $b_8(p, I) > 1$
- p muss mindestens ein 4-adjazentes Hintergrundpixel haben, d.h. $p \in C_4(I)$

Da das Skelett am Ende idealerweise ein Pixel breit sein soll, wird in der Literatur vornehmlich mit der 8er-Zusammengehörigkeit für die Objekte und demzufolge mit einer 4er-Zusammengehörigkeit für den Hintergrund gearbeitet. Daraus abgeleitet, brauchen nur Konturpixel betrachtet werden, die 4-adjazent zum Hintergrund sind, um die Zusammengehörigkeit der Objekte zu wahren. Die obigen drei Bedingungen sind Grundvoraussetzungen für alle Ausdünnungs-Algorithmen. Die meisten Algorithmen unterscheiden sich nur in der Prüfung der Pixel, ob eine Löschung die Zusammengehörigkeit ändert. Als Hilfsmittel dafür eignet sich die *Kreuzungszahl*. Diese ergibt

sich beim "Ablaufen" der 8er-Nachbarn eines Pixels p im Bild I in der Reihenfolge $p_1, p_2, ..., p_8$ (siehe Abbildung 2.10) und Zählen der Übergänge von Objektpixeln zu Hintergrundpixeln und umgekehrt. Dabei haben sich zwei verschiedene Definitionen bewährt:

$$X_R(p, I) := \sum_{i=1}^{8} |I(p_{i+1}) - I(p_i)| \tag{5.8}$$

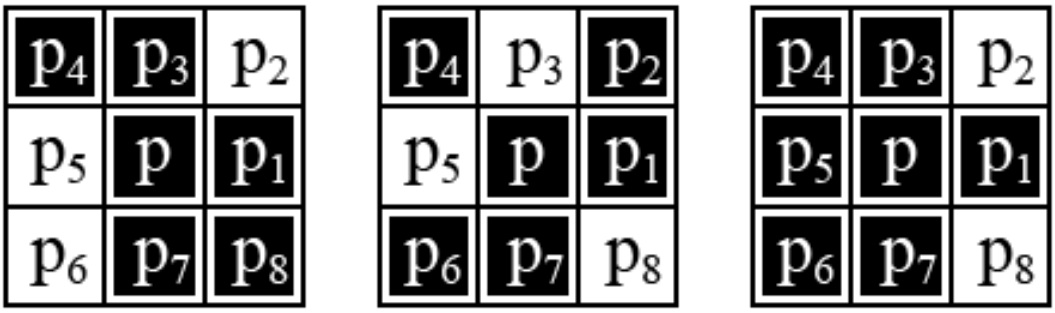
$$X_H(p, I) := \sum_{i=1}^{4} h_i(I) \tag{5.9}$$

$$h_i(I) := \begin{cases} 1 & \text{wenn } I(p_{2i-1}) = 0 \wedge (I(p_{2i}) = 1 \vee I(p_{2i+1}) = 1) \qquad (1 \leq i \leq 4) \\ 0 & \text{sonst} \end{cases} \tag{5.10}$$

wobei gilt: $p_9 = p_1$. Die *Rutovitz-Kreuzungszahl* $X_R(p, I)$ zählt alle Übergänge von Objektpixeln zu Hintergrundpixeln und umgekehrt und entspricht der doppelten Anzahl von 4-adjazenten Objekten innerhalb der 8er-Nachbarschaft. Alternativ kann auch nur eine Art von Übergang gezählt werden, etwa von Objekt- zu Hintergrundpixel. Diese Zahl entspricht dann der Hälfte von $X_R(p, I)$. Die Rutovitz-Kreuzungszahl eignet sich besonders für 4-zusammengehörige Objekte, da die 4er-Zusammengehörigkeit gewahrt bleibt, wenn $X_R(p, I) = 2$. Da die Objekte jedoch meist durch die 8er-Zusammengehörigkeit definiert werden, ist bei Nutzung dieser Kreuzungszahl eine Nachbehandlung der Skelette notwendig, um diese auf Ein-Pixel-Breite zu reduzieren. Um das zu umgehen, wurde von Hilditch 1969 die *Hilditch-Kreuzungszahl* $X_H(p, I)$ eingeführt. Sie entspricht der Anzahl von 8-adjazenten Objekten innerhalb der 8er-Nachbarschaft von p, außer p hat kein 4-adjazentes Hintergrundpixel. Hat ein Pixel p die Hilditch-Kreuzungszahl $X_H(p, I) = 1$, kann es gelöscht werden, ohne die 8er-Zusammengehörigkeit des Objektes zu verändern. Mit $X_H(p, I) = 1$ ist außerdem sichergestellt, dass p ein Konturpixel mit einem 4-adjazenten Hintergrundpixel ist, was allein durch $X_R(p, I) = 2$ nicht der Fall ist. [29]

Ein sequentieller Ausdünnungs-Algorithmus, der in der Literatur oft für Vergleiche herangezogen wird, ist der Algorithmus von Hilditch [29]. Dieser arbeitet das Bild in Standardrasterscan-Reihenfolge ab und hat zu den drei obigen grundlegenden Voraussetzungen weitere vier Bedingungen für ein simples Pixel p:

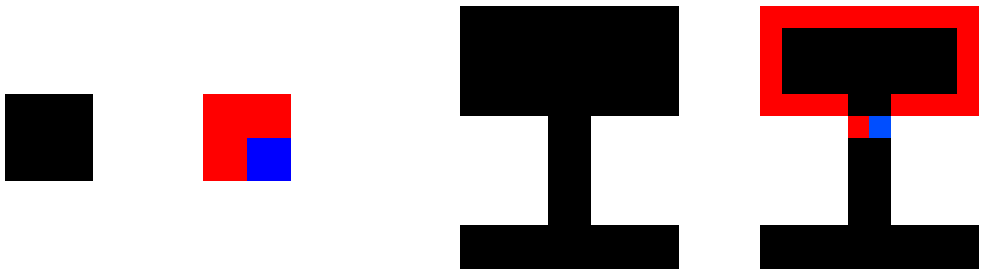
- mindestens ein Objektpixel aus $N_8(p)$ darf noch nicht markiert sein
- $X_H(p, I) = 1$ (am Anfang der Iteration)
- Wenn p_3 markiert ist, muss $X_H(p, I) = 1$ bleiben auch wenn p_3 gelöscht wird
- Wenn p_5 markiert ist, muss $X_H(p, I) = 1$ bleiben auch wenn p_5 gelöscht wird



(a) (b) (c)

Abbildung 5.6: Die Rutovitz-Kreuzungszahl $X_R(p, I)$ entspricht der doppelten Anzahl von 4-adjazenten Objekten innerhalb der 8er-Nachbarschaft von p. Die Hilditch-Kreuzungszahl $X_H(p, I)$ hingegen der Anzahl von 8-adjazenten Objekten innerhalb der 8er-Nachbarschaft von p (außer wenn alle 4er Nachbarn von p Objektpixel sind). Beispiele: (a) $X_R(p, I) = 4$; $X_H(p, I) = 1$. (b) $X_R(p, I) = 6$; $X_H(p, I) = 2$. (c) $X_R(p, I) = 4$; $X_H(p, I) = 0$.

Die erste Bedingung verhindert, dass kleine Objekte vollständig verschwinden. Bedingungen drei und vier sorgen dafür, dass zwei-Pixel-breite Linien nicht innerhalb einer Iteration unterbrochen werden. Abbildung 5.7 veranschaulicht die Notwendigkeit dieser Bedingungen.



(a) (b) (c) (d)

Abbildung 5.7: Bewahrung der Zusammengehörigkeit von Objekten beim Hilditch-Algorithmus: Ein quadratisches Objekt (a), bestehend aus vier Pixeln, wird dank der ersten Bedingung nicht komplett gelöscht. In (b) sind die roten Pixel bereits zum Löschen markiert worden. Das blaue Pixel wird wegen der ersten Bedingung nicht markiert. (c) zeigt ein Objekt mit einer zwei-Pixel-breiten Linie, diese wird aufgrund von Bedingung vier nicht unterbrochen. In (d) sind die roten Pixel bereits zum Löschen markiert worden. Das Löschen des blauen Pixels wird aufgrund der vierten Bedingung verhindert.

In Pseudocode 10 wurden alle sieben Bedingungen für simple Pixel beim Hilditch-Algorithmus in die Funktion *istSimpel(p,I,M)* aus Pseudocode 9 umgesetzt. Die dritte grundlegende Voraussetzung $p \in C_4(I)$ wird durch die zweite Hilditch-Bedingung $X_H(p, I) = 1$ mit abgedeckt und muss somit nicht überprüft werden. Sie wurde im Pseudocode lediglich zugunsten der Übersichtlichkeit mit einbezogen.

Die Abbildungen 5.8 und 5.9 zeigen abschließend Beispiele von Hilditch-Skeletten.

Pseudocode 10 : Hilditch-Algorithmus: *istsimpel(p,S,M)*

Eingabe : Pixel p
Eingabe : aktuelles Skelett S
Eingabe : Menge M der markierten Pixel
Ausgabe : Wahrheitswert ob p simpel ist oder nicht

```
1  wenn p ∉ ⟨S⟩ dann                                          /* Grund-Bedingung 1 */
2  |  return falsch
3  wenn b_8(p,S) ≤ 1 dann                                     /* Grund-Bedingung 2 */
4  |  return falsch
5  wenn  p ∉ C_4(S) dann                          /* Grund-Bedingung 3 (überflüssig) */
6  |  return falsch
7  wenn ∀q((q ∈ ⟨S⟩ ∧ q ∈ N_8(p)) → q ∈ M) dann          /* Hilditch-Bedingung 1 */
8  |  return falsch
9  wenn X_H(p,S) ≠ 1 dann          /* Hilditch-Bedingung 2 (beinhaltet Zeile 5) */
10 |  return falsch
11 wenn p_3 ∈ M ∧ (X_R(p,S) ≠ 1 für S(p_3) = 0) dann       /* Hilditch-Bedingung 3 */
12 |  return falsch
13 wenn p_5 ∈ M ∧ (X_R(p,S) ≠ 1 für S(p_5) = 0) dann       /* Hilditch-Bedingung 4 */
14 |  return falsch
15 return wahr
```

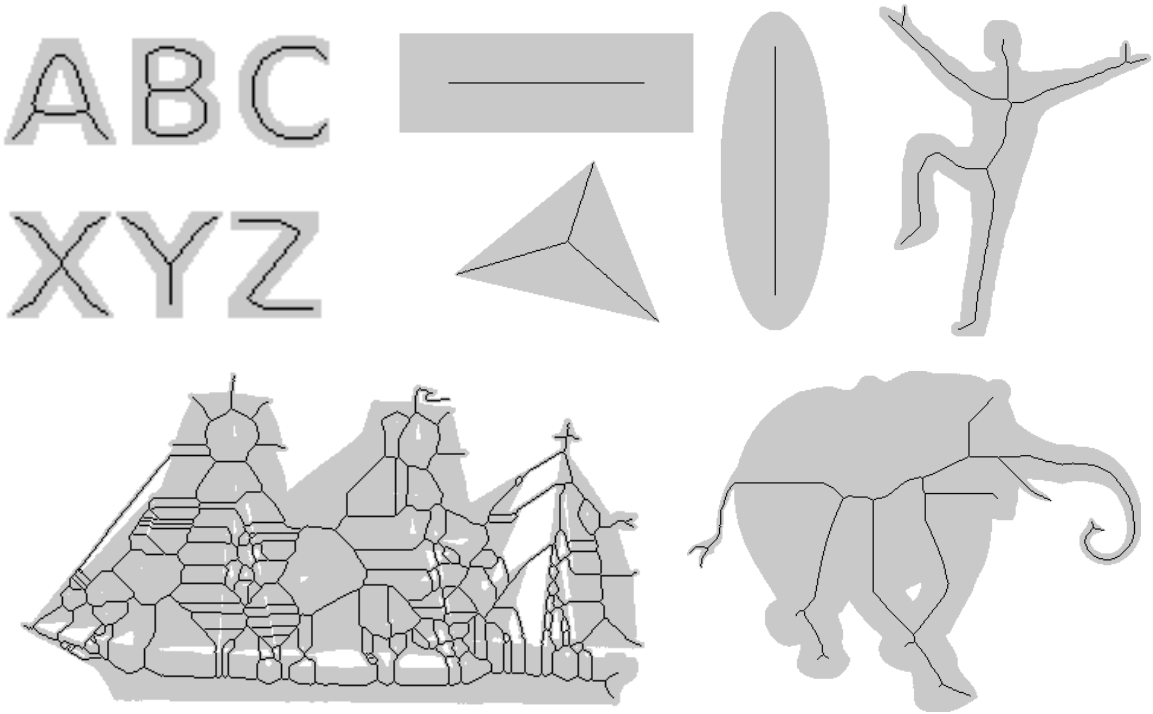


Abbildung 5.8: Im Rahmen dieser Arbeit erstellte Beispiele von Hilditch-Skeletten: von links nach rechts, von oben nach unten: [BILD 2,11,23,30,6].

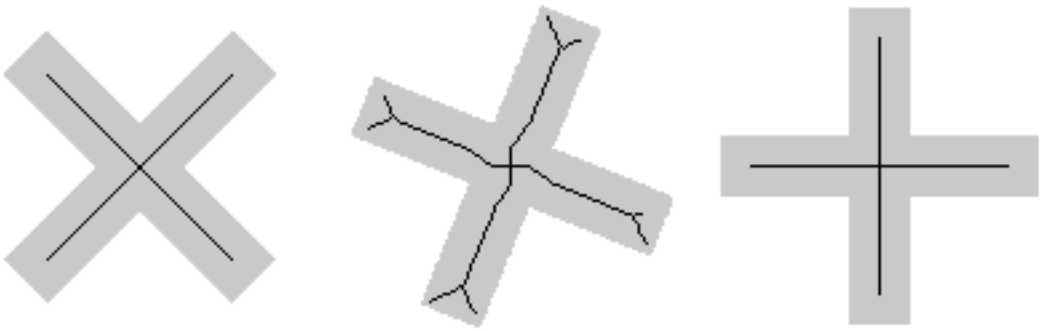


Abbildung 5.9: Ausdünnungs-Algorithmen liefern nach Meinung des Autors die besten Resultate für gedrehte Objekte aller im Rahmen dieser Arbeit getesteten Algorithmen (siehe Abschnitt 6.2). Beispielhaft dafür wurde in (a)-(c) ein Kreuz um jeweils 22,5° nach links gedreht [BILD 17,18,19]. Die berechneten Hilditch-Skelette sind sich, im Vergleich zu den bisherigen Verfahren, sehr ähnlich. Bild (a) und (c) liefern nach menschlicher Bewertung optimale Ergebnisse.

5.3 Paralleles Ausdünnen

Paralleles Ausdünnen bedeutet, dass innerhalb einer Iteration simple Pixel nur anhand des Zustandes des Bildes vor der Iteration erkannt werden können. Das heißt, andere innerhalb der Iteration bereits abgearbeitete Pixel, spielen bei der Bewertung eines Pixels keine Rolle. Es ist also keine festgelegte Reihenfolge nötig und alle Pixel können parallel (d.h. gleichzeitig) bearbeitet werden. Diese Algorithmen sind damit auch für Mehrprozessor-Systeme geeignet. Bei sequentiellen Algorithmen reicht eine 3x3 Maske, um sicher simple Pixel zu erkennen und so die Zusammengehörigkeit zu bewahren. Für parallele Ausdünnungs-Algorithmen gilt dies nicht. Allein mit einer 3x3 Maske gelingt es beispielsweise nicht eine zwei-Pixel-breite Linie zu erhalten. Eine Lösung dieses Problems ist es, die Maske auszuweiten. So wird in [23] mit einer 4x4 Maske gearbeitet, um die Zusammengehörigkeit der Objekte zu sichern. Üblicherweise jedoch wird bei parallelen Ausdünnungs-Algorithmen eine Iteration in mehrere Unteriterationen geteilt, wobei nach jeder Unteriteration die markierten Pixel gelöscht werden. In Pseudocode 11 wurde dieses Vorgehen in einem Algorithmus umgesetzt.

Beim Vergleich der Funktion *istSimpel_ U(i,p,S)* der parallelen Ausdünnungs-Algorithmen (Pseudocode 11, Zeile 8) und der Funktion *istSimpel(p,S,M)* der sequentiellen Ausdünnungs-Algorithmen (Pseudocode 9, Zeile 7) wird deutlich, dass bei den parallelen Algorithmen keine Informationen über bereits markierte Pixel (Menge M) benötigt werden. Dafür ist die Funktion der parallelen Ausdünnungs-Algorithmen abhängig von dem Index der aktuellen Unteriteration i. Das bedeutet, dass in jeder Unteriteration andere Arten von Pixeln gelöscht werden können.

Pseudocode 11 : Paralleler Ausdünnungs-Algorithmus

Eingabe : Bild I
Ausgabe : Skelett S

```
1  S ← Kopie von I
2  M ← ∅                                              /* Menge markierter Pixel */
3  v ← wahr                                  /* Flag zum Abbrechen des Algorithmus */
4  solange v = wahr tue
5      für jede Unteriteration i tue
6          v ← falsch
7          für alle Pixel p aus S tue        /* beliebige Reihenfolge der Abarbeitung */
8              wenn istSimpel_ U(i,p,S) dann
9                  M ← M ∪ {p}
10         für alle Pixel p aus M tue
11             S(p) ← 0                       /* setze p in S auf Hintergrundfarbe */
12         wenn M ≠ ∅ dann
13             v ← wahr
14         M ← ∅
```

In [52] teilen Stefanelli/Rosenfeld eine Iteration in vier Unteriterationen, wobei vier verschiedene Typen von Konturpixeln behandelt werden: Nord-, West-, Süd- und Ostpixel (Pixel mit 4-adjazenten Hintergrundpixeln in der jeweiligen Himmelsrichtung; siehe Abbildung 5.10).

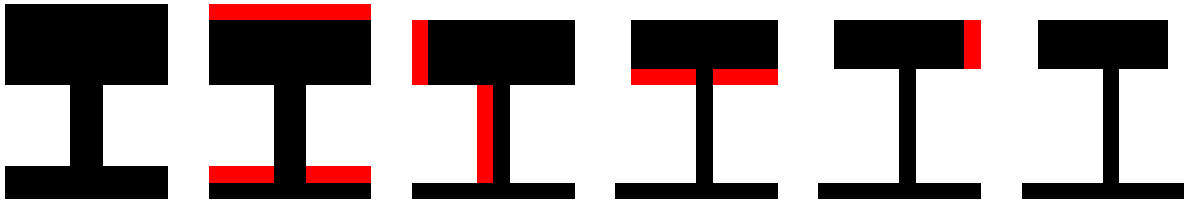


Abbildung 5.10: Um bei parallelen Ausdünnungs-Algorithmen die Zusammengehörigkeit zu bewahren, wird eine Iteration in mehrere Unteriterationen geteilt. In [52] wird mit vier Unteriterationen gearbeitet, wobei Pixel nach ihrer "Himmelsrichtung" unterschieden werden. (a) zeigt das Originalbild. In der ersten Iteration werden die simplen Nordpixel gelöscht (b) (rot markiert), in der zweiten Iteration die Westpixel (c), dann die Südpixel (d) und in der letzten Iteration die Ostpixel (e). (f) zeigt das Ergebnis nach einer durchlaufenen Iteration.

Zhang/Suen haben in [58] gezeigt, dass bereits zwei Unteriterationen ausreichen. Dabei werden in einer Unteriteration gemeinsam Süd- und Ostpixel und in der anderen Unteriteration Nord- und Westpixel betrachtet (Abbildung 5.11). Dieser Algorithmus hat zudem wenige und einfach zu implementierende Bedingungen, weshalb er auch in viele Standardwerke übernommen wurde [20][24].

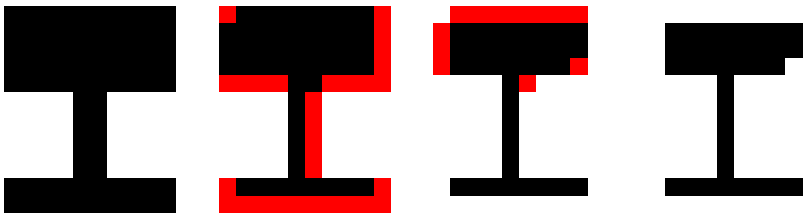
Erste Iteration

- $2 \leq b_8(p, I) \leq 6$
- $X_R(p, I) = 2$
- $I(p_1) \cdot I(p_3) \cdot I(p_7) = 0$
- $I(p_1) \cdot I(p_5) \cdot I(p_7) = 0$

Zweite Iteration

- $2 \leq b_8(p, I) \leq 6$
- $X_R(p, I) = 2$
- $I(p_1) \cdot I(p_3) \cdot I(p_5) = 0$
- $I(p_3) \cdot I(p_5) \cdot I(p_7) = 0$

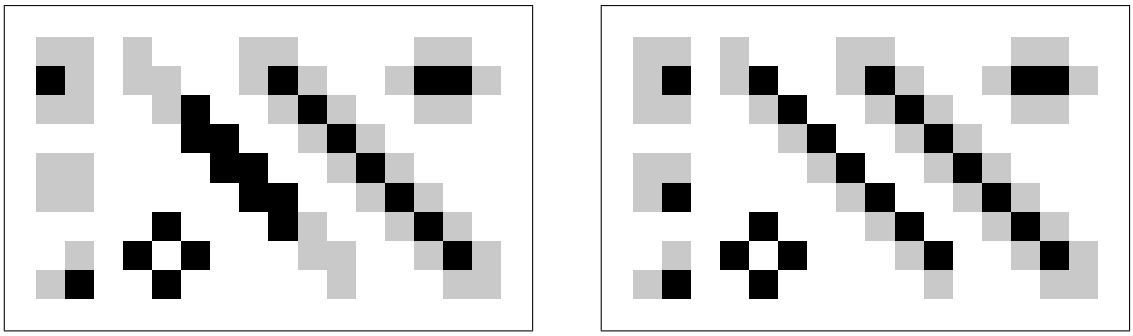
Die Kriterien für die zweite Unteriteration unterscheiden sich von der ersten nur in den letzten zwei Bedingungen, bei denen auf die "Richtung" der Pixel geprüft wird. In Pseudocode 12 ist die Funktion *istsimpel_ U(i,p,S)* für den Zhang/Suen-Algorithmus angegeben. Auch hier wird eine Grundbedingung für simple Pixel bereits durch eine Zhang/Suen-Bedingung abgedeckt und kann bei der Implementierung weggelassen werden. Aufgrund der relativ einfach gehaltenen Bedingungen werden innerhalb einer Unteriteration "ungewollt" die jeweils gegenüberliegenden Eckpunkte mitgelöscht. In der ersten Unteriteration betrifft das beispielsweise die Nord-West-Eckpunkte. Dies hat in der Regel keine negativen Auswirkungen, da diese Pixel in der jeweils nächsten Unteriteration sowieso gelöscht werden würden. Für den speziellen Fall eines 2x2 Pixel-Blockes



(a) (b) (c) (d)

Abbildung 5.11: Zhang/Suen [58] unterteilen eine Iteration in nur zwei Unteriterationen. Im Beispiel wird dies verdeutlicht. (a) zeigt das Original. In (b) werden zusammen simple Süd- und Ostpixel gelöscht (rot markiert). In (c) anschließend simple Nord- und Westpixel. (d) ist das Ergebnis nach einer Iteration.

führt dieses Verhalten allerdings dazu, dass dieser Block komplett verschwindet. Ebenso werden zwei-Pixel-starke diagonale Linien stark gekürzt [34]. Beide problematischen Aspekte werden in Abbildung 5.12 im Vergleich zum Hilditch-Algorithmus deutlich.



(a) (b)

Abbildung 5.12: Der Zhang/Suen-Algorithmus arbeitet für den speziellen Fall eines 2x2 Pixel-Blockes nicht korrekt, da dieser komplett verschwindet und so die topologische Struktur nicht erhalten bleibt. Dies wird im Vergleich zum Hilditch-Algorithmus deutlich. In (a) wurde der Zhang/Suen-Algorithmus angewendet. In (b) der Hilditch-Algorithmus. Eine weitere Schwäche des Zhang/Suen-Algorithmus, das starke Verkürzen von zwei-Pixel-breiten diagonalen Linien, ist ebenfalls gut erkennbar [BILD 15].

In Abbildung 5.12 wird ebenso deutlich, dass der Zhang/Suen Algorithmus keine Ein-Pixel-Breite der Skelette garantiert. Dies liegt auch daran, dass die Rutovitz-Kreuzungszahl $X_R(p, I)$ zum Einsatz kommt (siehe Abschnitt 5.2). Ein einfacher Austausch mit der Hilditch-Kreuzungszahl $X_H(p, I)$ ist allerdings nicht möglich, da dann die Zusammengehörigkeit nicht mehr garantiert werden könnte.

Pseudocode 12 : Zhang/Suen-Algorithmus - 1te Unteriteration: *istsimpel_ U(i,p,S)*

Eingabe : Index i der aktuellen Unteriteration
Eingabe : Pixel p
Eingabe : aktuelles Skelett S
Ausgabe : Wahrheitswert ob p simpel ist oder nicht

```
1  wenn p ∉ ⟨S⟩ dann                                   /* Grund-Bedingung 1 */
2      return falsch
3  wenn  b_8(p,S) ≤ 1 dann                  /* Grund-Bedingung 2 (überflüssig) */
4      return falsch
5  wenn p ∉ C_4(S) dann                                /* Grund-Bedingung 3 */
6      return falsch
7  wenn ¬(2 ≤ b_8(p,S) ≤ 6) dann      /* Z/S-Bedingung 1 (beinhaltet Zeile 3) */
8      return falsch
9  wenn X_R(p,S) ≠ 2 dann                              /* Z/S-Bedingung 2 */
10     return falsch
11 wenn i = 1 dann                                   /* erste Unteriteration */
12     wenn S(p_1)·S(p_3)·S(p_7) ≠ 0 dann                /* Z/S-Bedingung 3 */
13         return falsch
14     wenn S(p_1)·S(p_5)·S(p_7) ≠ 0 dann                /* Z/S-Bedingung 4 */
15         return falsch
16 sonst                                             /* zweite Unteriteration */
17     wenn S(p_1)·S(p_3)·S(p_5) ≠ 0 dann                /* Z/S-Bedingung 3 */
18         return falsch
19     wenn S(p_3)·S(p_5)·S(p_7) ≠ 0 dann                /* Z/S-Bedingung 4 */
20         return falsch
21 return wahr
```

Das starke Verkürzen von zwei-Pixel-starken diagonalen Linien ist topologisch gesehen kein Problem, wohl aber das Verschwinden von 2x2 Pixel-Blöcken. Modifiziert man den Algorithmus von Zhang/Suen mit zwei zusätzlichen Bedingungen, kann dies verhindert werden [16]:

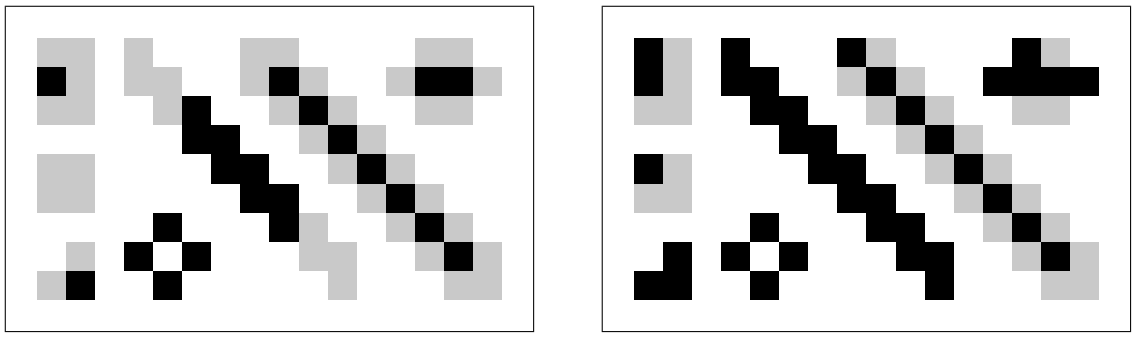
Erste Iteration

- $b_4(p, I) > 1 \rightarrow (I(p_1) = 0 \vee I(p_7) = 0)$

Zweite Iteration

- $b_4(p, I) > 1 \rightarrow (I(p_3) = 0 \vee I(p_5) = 0)$

Abbildung 5.13 zeigt die Auswirkungen dieser Modifizierung. Es ist zu erkennen, dass durch die zusätzlichen Einschränkungen die Skelette im Allgemeinen mehr Pixel enthalten.



(a) (b)

Abbildung 5.13: Eine Modifikation von Eckardt [16] des Zhang/Suen-Algorithmus verhindert das Verschwinden von 2x2 Pixel-Blöcken. In (a) wurde der Zhang/Suen-Algorithmus angewendet. In (b) die modifizierte Version. Der 2x2 Pixel-Block bleibt erhalten, auch werden zwei-Pixel-breite diagonale Linien nicht verkürzt [BILD 15].

Ein prinzipielles Problem von Ausdünnungs-Algorithmen ist die Entstehung von zu vielen Verzweigungen/Ästen. Oftmals bilden sich "unnötige" Äste, die eher unnatürlich und störend wirken. Schuld daran ist eine "unsaubere" Kontur. Einzelne Pixel der Kontur erfüllen dann die Endpunktbedingung und bleiben so erhalten. Abhilfe gegen solche unsauberen Konturen bietet eine Opening-Operation. Abbildung 5.14 soll dies verdeutlichen.

Ausdünnungs-Algorithmen haben in der Regel eine Laufzeit von $\mathcal{O}(n^3)$, wie etwa der sequentielle Hilditch-Algorithmus oder der parallele Zhang/Suen-Algorithmus. Einer der derzeit schnellsten Algorithmen ist der von Kwok [28]. Dieser arbeitet mit einmal erstellten Ketten-Codes der Konturen (Freeman-Code) und erstellt dann iterativ neue Konturen für die jeweils nächste Iteration. Dieser Algorithmus benötigt lediglich $\mathcal{O}(n^2)$ [37]. Um Rechenzeit einzusparen, wurden in der Vergangenheit viele Vorschläge für Verbesserungen gemacht. So wird in [9] bei der Prüfung der simplen Pixel nur in einer Lookup-Table mit allen 256 Möglichkeiten der Belegung der 8er-Nachbarn nachgeschlagen. Für sequentielle Algorithmen wurde versucht, die Definition simpler Pixel auf ein $k \times k$ Fenster auszuweiten, um gleich mehrere Pixel mit einmal abtragen zu können [29]. Dies hat aber auch zur Folge, dass die Kontur unterschiedlich stark abgetragen wird und das Skelett am Ende möglicherweise nicht mittig liegt.

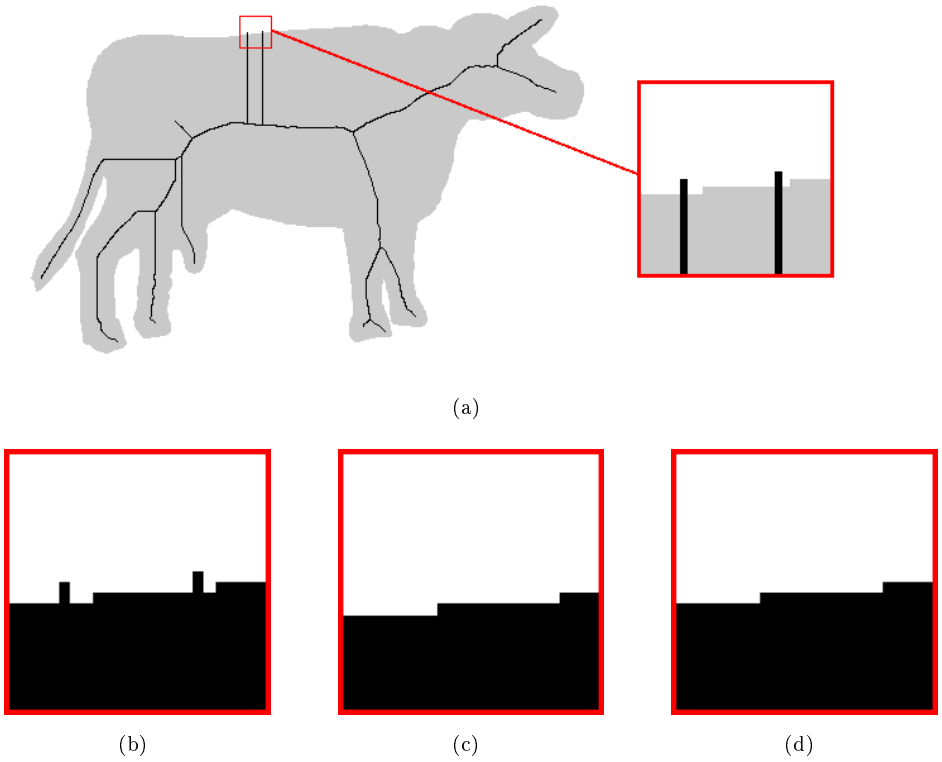


Abbildung 5.14: Bei unsauberen Konturen können bei Ausdünnungs-Algorithmen durch die Endpunktbedingung leicht "überflüssige" Verzweigungen/Äste entstehen. Siehe Bild (a). Durch eine vorherige Opening-Operation (c)-(d) auf die unsaubere Kontur (b) können solche störenden Pixel elemeniert werden. (c) zeigt das Resultat nach einer Erosion auf (b), In (d) wurde eine Dilatation auf (c) angewendet.

Der Autor schlägt vor, für parallele Algorithmen eine Liste mit Konturpunkten anzulegen, um diese dann in jeder Iteration zu aktualisieren (neue Konturpunkte können nur in der 4er-Nachbarschaft eines alten Konturpunktes liegen). So muss lediglich diese Liste auf simple Pixel überprüft werden. Dieses Vorgehen wurde auch im Rahmen dieser Arbeit implementiert und führte für den Zhang/Suen-Algorithmus zu einer deutlich geringen Rechenzeit. Pseudocode 13 zeigt einen parallelen Ausdünnungs-Algorithmus mit dieser Modifikation. Ein weiterer Nebeneffekt dieser Modifikation ist, dass auf die Prüfung der ersten Grund-Bedingung $p \in \langle I \rangle$ und ebenso auf die dritte Grund-Bedingung $p \in C_4(I)$ innerhalb der Funktion *istSimpel_ U(i,p,S)* verzichtet werden kann. Damit müssen im Fall des Zhang/Suen-Algorithmus nur noch die vier speziellen Zhang/Suen-Bedingungen überprüft werden.

Wie gezeigt, können Ausdünnungs-Algorithmen sehr verschieden sein - nicht nur in der Implementierung, sondern auch in der Laufzeit. Ebenso können sich die Resultat in ihren Eigenschaften unterscheiden. Für die Auswahl eines geeigneten Algorithmus ist es ratsam, auch über Alternativen zum bekannten Zhang/Suen-Algorithmus nachzudenken, insbesondere wenn 8er-Zusammengehörigkeit gefordert wird. Ebenso können Vor- und Nachbehandlungen, wie Opening-

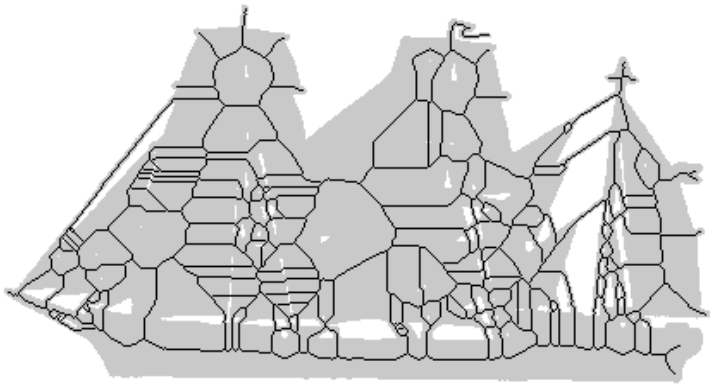
Operationen, Glättungsfilter oder das Löschen zu kurzer Äste (*pruning*), die Qualität der Skelette stark verbessern.

Auf weitere Beispiele von Zhang/Suen-Skeletten wird an dieser Stelle verzichtet, da diese den Hilditch-Skeletten (Abbildung 5.8) sehr ähnlich sehen. Stattdessen soll in Abbildung 5.15 an einem Beispiel das Hilditch- und Zhang/Suen-Skelett gegenübergestellt werden. Beide Skelette wirken auf den ersten Blick fast identisch, eine Überlagerung beider Bilder zeigt dennoch einige Unterschiede.

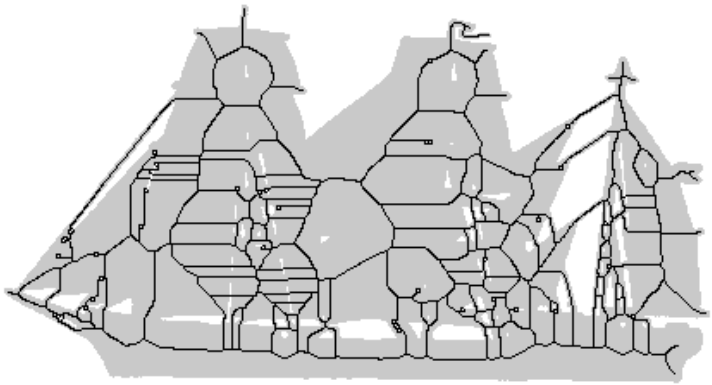
Pseudocode 13 : Modifizierter paralleler Ausdünnungs-Algorithmus

Eingabe : Bild I
Ausgabe : Skelett S

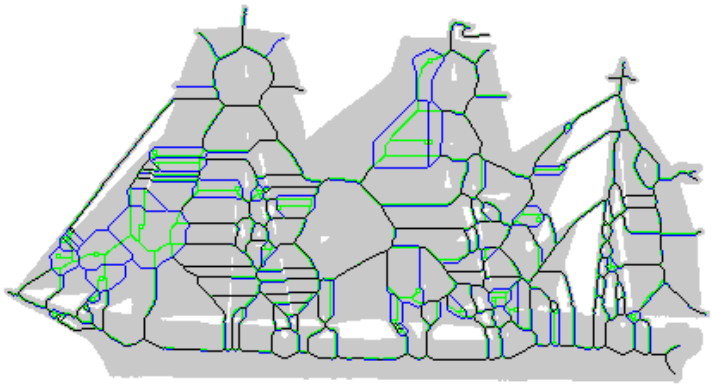
```
1  S ← Kopie von I
2  C ← Menge der Konturpixel C_4(S)                    /* abzuarbeitende Pixel */
3  CN ← ∅                          /* abzuarbeitende Pixel der nächsten Iteration */
4  M ← ∅                                               /* Menge markierter Pixel */
5  v ← wahr                                  /* Flag zum Abbrechen des Algorithmus */
6  solange v = wahr tue
7      für jede Unteriteration i tue
8          v ← falsch
9          für alle Pixel p aus C tue        /* beliebige Reihenfolge der Abarbeitung */
10             wenn istSimpel_U(i,p,S) dann
11                 M ← M ∪ {p}
12         für alle Pixel p aus M tue
13             S(p) ← 0                        /* setze p in S auf Hintergrundfarbe */
14             C ← C\{p}
15             für alle Pixel q aus N_4(p) tue
16                 wenn q ∈ ⟨S⟩ ∧ q ∉ M dann
17                     CN ← CN ∪ {q}
18         wenn M ≠ ∅ dann
19             v ← wahr
20         M ← ∅
21     C ← C ∪ CN
22     CN ← ∅
```



(a)



(b)



(c)

Abbildung 5.15: Ausdünnungs-Algorithmen liefern meist sehr ähnliche Resultate. So wirken das Hilditch-Skelett (a) und das Zhang/Suen-Skelett (b) fast identisch. Bei Überlagerung beider Skelette zeigen sich jedoch einige Unterschiede. In (c) wurden übereinstimmende Pixel schwarz markiert. Blaue Pixel gehören zum Hilditch-Skelett, grüne Pixel zum Zhang/Suen-Skelett [BILD 30].

6 Schlussbemerkungen

6.1 Vergleichbarkeit von Skelettierungsalgorithmen

Wie eingangs erwähnt, ist der Begriff des Skelettes nicht eindeutig definiert. Alle hier vorgestellten Definitionen und Algorithmen liefern als Resultate Skelette und die vielen unterschiedlichen Ergebnisse lassen erahnen, warum es so schwer ist, eine einheitliche Definition für den Begriff des Skelettes zu finden. Der ideale Skelettierungsalgorithmus sollte die topologischen und geometrischen Eigenschaften des Originalbildes widerspiegeln, isotrop sein, eine Wiederherstellung des Originalbildes zulassen und dabei sehr schnell arbeiten [29]. Leider gibt es diesen Algorithmus noch nicht, jedoch eignen sich diese fünf Forderungen, um Skelettierungsalgorithmen zu charakterisieren. Eine Beurteilung, beispielsweise der Isotropie (Invarianz bei Drehungen), ist dabei oftmals nur nach subjektiver Bewertung mit "gut", "weniger gut", "schlecht" usw. möglich (siehe Abbildungen 3.6, 3.10, 4.5, 5.5 und 5.9). Es fehlt ein fester Maßstab mit dem sich Skelettierungsalgorithmen objektiv bewerten lassen. Auch die hier verwendete *Landau-Notation* für die Angabe der Laufzeit ist in der Praxis nur bedingt nützlich. Für zeitkritische Anwendungen bedeutet die Bevorzugung des einfachen Kritische-Punkte-Ansatzes (ein Rasterscan) gegenüber der Errechnung des Distanz-Skelettes (drei Rasterscans) durchaus eine Zeitersparnis, auch wenn beide Algorithmen eine Laufzeit von $\mathcal{O}(n^2)$ haben. Ähnlich unterscheiden sich viele Ausdünnungs-Algorithmen beträchtlich in ihrer tatsächlich benötigten Rechenzeit. In der Literatur werden die Algorithmen gern anhand konkreter Rechenzeiten eines Testrechners für drei bis vier Objekte verglichen, so geschehen in [51][56][58][59] und [63]. Für diese geringe Anzahl an verschiedenen Mustern sollten solche Vergleiche allerdings kritisch betrachtet werden, zumal in all diesen Publikationen der neu vorgestellte Algorithmus immer am schnellsten ist und sicher auch nicht nachteilig dargestellt werden sollte. Für den Vergleich von Ausdünnungs-Algorithmen eignet sich die Gegenüberstellung der tatsächlich erfolgten Iterationen bzw. Pixelzugriffe, wie in [9][61] und [60] geschehen. Die ist natürlich nur für pixelbasierte Algorithmen möglich, damit ist diese Art des Vergleiches für verfahrensübergreifende Untersuchungen ungeeignet.

Um überhaupt Vergleiche bezüglich der benötigten Rechenzeiten anstellen zu können, sollen abschließend auch in dieser Arbeit die implementierten Algorithmen anhand konkreter Beispiele gegenübergestellt werden. In Anhang B wurden für drei verschiedenartige Bilder in jeweils acht verschiedenen Auflösungen die Rechenzeiten der Algorithmen ermittelt und in Diagrammen

zusammengefasst. Die Ergebnisse lassen nach Meinung des Autors etwas konkretere Aussagen zu, als es allein die Laufzeitkomplexitäten der Algorithmen ermöglichen, allerdings können durch solche Tests nicht alle Eventualitäten, insbesondere Worst-Case-Szenarien, durchgespielt werden.

Andere Bewertungen hinsichtlich topologischer Eigenschaften oder der Wiederherstellbarkeit sind dagegen eindeutig und mathematisch nachvollziehbar. Insgesamt kann man festhalten, dass sich die oben genannten fünf Forderungen als Bewertungsgrundlage für Skelettierungsalgorithmen eignen, ein fester und objektiver Maßstab aber fehlt. Der nach einem passenden Algorithmus suchende Anwender, wird seine Suche anhand der obigen Kriterien sicher eingrenzen können, jedoch wird ihm ein Ausprobieren, um den richtigen Algorithmus zu finden, wohl nicht erspart bleiben.

6.2 ImageJ-Plugin

Im Rahmen dieser Arbeit wurden einige Skelettierungsalgorithmen implementiert. Als Grundlage kam dazu das quelloffene Bildverarbeitungsprogramm *ImageJ*[1] zum Einsatz. Dafür wurde ein Plugin geschrieben, mit welchem sich folgende Skelette errechnen lassen:

- Mediale Achse mit drei verschiedenen Metriken (siehe Abschnitt 3.1)
- Distanz Skelett (siehe Abschnitt 3.2)
- Skelett nach dem einfachen Kritische-Punkte-Ansatz (siehe Abschnitt 4.1)
- morphologisches Skelett (siehe Abschnitt 5.1)
- Hilditch-Skelett (siehe Abschnitt 5.2)
- Zhang/Suen-Skelett (siehe Abschnitt 5.3)
- modifiziertes Zhang/Suen-Skelett (siehe Abschnitt 5.3)

Der Quelltext und das fertige Plugin sind der Arbeit auf einem Datenträger beigelegt oder können unter `http://www.christoph-bullmann.de/bachelor` im Internet herunter geladen werden. Zur Installation muss die Plugin-Datei *Plugin_Skelettberechnung.jar* in das ImageJ-Verzeichnis *ImageJ/plugins* kopiert werden. Das Plugin kann dann im ImageJ-Menü unter *Plugins* → *Skelettberechnung* geöffnet werden. Die Installationsdateien für das Programm ImageJ befinden sich ebenfalls auf dem Datenträger.

[1] `http://rsb.info.nih.gov/ij/`

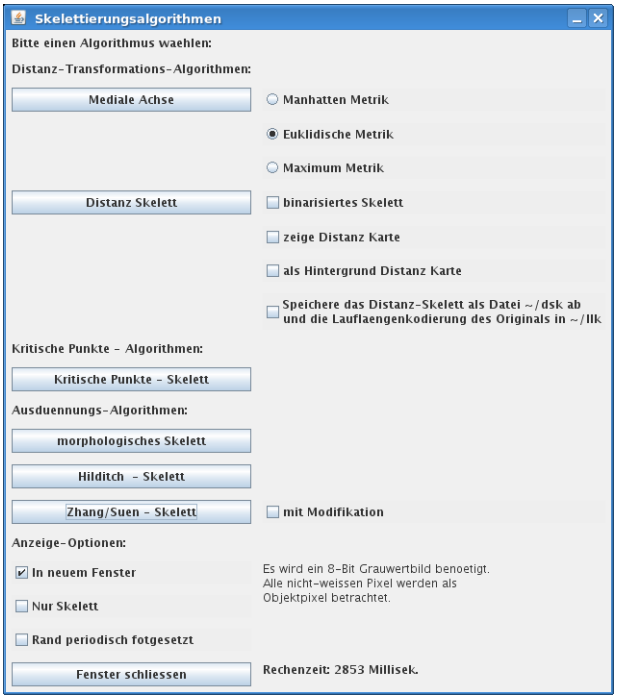


Abbildung 6.1: Um die in dieser Arbeit vorgestellten Skelettierungsalgorithmen hinsichtlich ihrer Resultate untersuchen zu können, wurde ein Plugin für das Bildverarbeitungsprogramm *ImageJ* erstellt.

6.3 Ausblick

Das Berechnen von Skeletten in der digitalen Bildverarbeitung ist nach wie vor ein sehr aktives Forschungsfeld. Dabei stehen weniger die "klassischen" Skelettberechnungen durch Ausdünnungs-Algorithmen im Vordergrund, vielmehr beschäftigen sich neuere Publikationen mit Kritische-Punkte-Algorithmen (beispielsweise [10][12][43][44]). Dies liegt sicher auch daran, dass Kritische-Punkte-Algorithmen aus der menschlichen Sichtweise bessere Resultate liefern und globale geometrische Eigenschaften besser widerspiegeln. Außerdem sind sie durch deutlichere, linienartige Strukturen für die automatisierte Erkennung und Klassifizierung geeigneter als "klassische" Ausdünnungs-Algorithmen. Zudem werden mit neuen Skelettierungsalgorithmen auch immer mehr Anwendungsgebiete erschlossen. Der Einsatz in der Bewegungsanalyse von Videoaufnahmen ist ein Beispiel. In [19] wird dazu ein vielversprechender Ansatz, basierend auf kritischen Punkten, präsentiert. Dabei werden kritische Punkte bestimmt, indem für jedes Konturpixel der Abstand zum Schwerpunkt des Objektes in ein Diagramm eingezeichnet wird. Die Reihenfolge für die Konturpixel wird durch Konturverfolgung festgelegt. Nach einer Glättung der sich ergebenen Funktion, werden die Pixel der lokalen Maxima als kritische Punkte ausgewählt (siehe Abbildung 6.2). Solche innovativen Ideen sind Anstoß für weitergehende Forschungen und einer vielseitigen Anwendung von Skelettierungsalgorithmen.

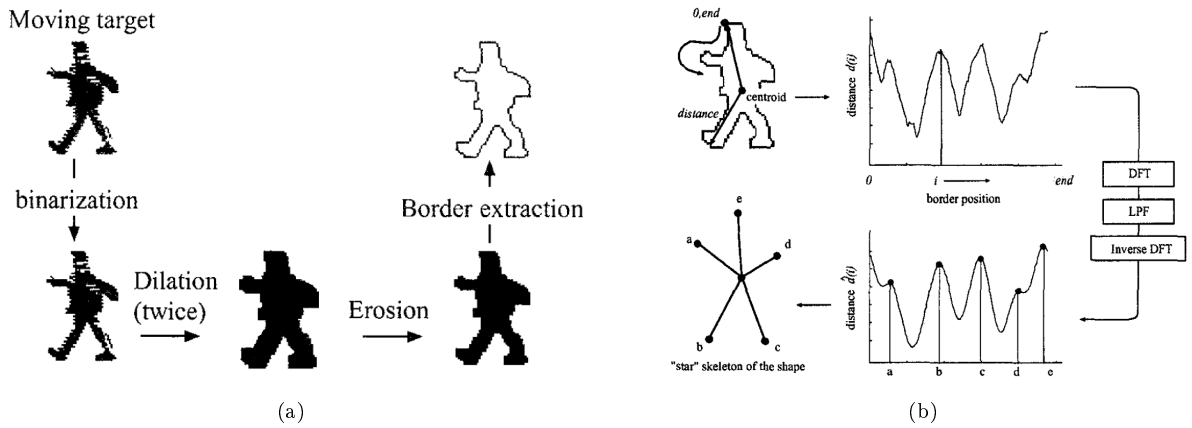


Abbildung 6.2: Skelettierungsalgorithmen werden in immer mehr Einsatzgebieten genutzt. Hier wird ein Stern-Skelett berechnet. Nach Extraktion des Objektes und einer Vorbehandlung wird die Kontur freigestellt (a). Danach werden die Abstände der Konturpunkte zum Schwerpunkt in ein Diagramm eingetragen. Nach Anwendung eines Glättungsfilter auf die enstandene Funktion werden die lokalen Maxima als kritische Punkte gewählt. Um das Stern-Skelett zu erhalten, werden die kritischen Punkte mit dem Schwerpunkt verbunden (b). Solch ein Skelett eignet sich beispielsweise zur Bewegungsanalyse in Videos [19].

Mit einer stetig wachsenden Leistungsfähigkeit der Computertechnik sind auch immer anspruchsvollere Ansätze realisierbar. Zudem ist ein deutlicher Trend zu Berechnungen von 3-dimensionalen Skeletten erkennbar. Diese werden verstärkt zur Visualisierung von komplexen 3-dimensionalen Objekten verwendet, um dem Anwender eine vereinfachte Ansicht zu bieten. Als Beispiel seien hochauflösende medizinische Aufnahmen (CT-Scan) genannt [43]. Gerade in der Medizin werden Skelettierungsalgorithmen aber auch zur automatisierten Analyse verwendet, beispielsweise zur Skelettierung von Lungenaufnahmen und dem Zählen von Lungenbläschen. Auch in Zukunft wird mit immer größeren zu verarbeitenden Datenmengen die Notwendigkeit von Abstraktion und Datenreduktion steigen. Damit ist auch in den nächsten Jahren mit vielen Publikationen und Forschungen zu Skelettierungsalgorithmen zu rechnen.

6.4 Fazit

Diese Arbeit hatte das Ziel, sich intensiv mit Skelettierungsalgorithmen auseinanderzusetzen und dabei die verschiedenen Arten der Skelettierungen zu untersuchen und zu charakterisieren. Stärken und Schwächen der Algorithmen, aber auch die Eigenschaften ihrer Resultate, sollten benannt werden, um Einsatzgebiete auszumachen und möglicherweise Verbesserungsvorschläge angeben zu können.

In der entstandenen Arbeit wurde zuerst eine Kategorisierung von Skelettierungsalgorithmen vorgenommen, um sie nach ihren generellen Vorgehensweisen zu unterscheiden. Danach konnten

einzelne konkrete Vertreter der verschiedenen Kategorien näher betrachtet werden. Dazu wurden Pseudocodes erstellt, die es erleichtern sollen, die präsentierten Ergebnisse nachzuvollziehen. Zudem wurden zur Erklärung der Vorgehensweisen der Algorithmen Illustrationen und Bilder von Zwischenergebnissen verwendet. So konnte in dieser Arbeit ein umfangreiches Wissen über Skelettierungsalgorithmen zusammen getragen werden.

Ein anderes Anliegen war die konkrete Anwendung der untersuchten Algorithmen. Dazu wurde eine Software entwickelt, die es zum einen überhaupt erst ermöglicht hat, konkrete Skelette in dieser Arbeit zu präsentieren und zum anderen dem Leser ein Werkzeug zur Verfügung stellt, mit dem er die in dieser Arbeit genannten Ergebnisse nachvollziehen und die verschiedenen Algorithmen ausprobieren bzw. für eigene Untersuchungen nutzen kann.

Bei der Bearbeitung stellte sich heraus, dass Skelettierungsalgorithmen ein sehr großes Potential haben. Es gibt eine Vielzahl von interessanten Publikationen, oftmals im Zusammenhang mit konkreten Problemstellungen, die Lust auf weitere Untersuchungen machen. Ein persönliches Ziel ist beispielsweise eine nähere Beschäftigung mit dem Stern-Skelett zur Analyse von Bewegungen in Videobildern (Abschnitt 6.3).

Ich war überrascht von der hohen Anzahl an Fachartikeln zu diesem Thema, da eine anfängliche Recherche in der Standardliteratur nur einige wenige Informationen zu diesem Thema hervorbrachte. Diese hohe Anzahl an Publikationen stellte aber auch eine Herausforderung dar, da sich die Aufgabe stellte, die vielen unterschiedlichen Ansätze zu kategorisieren.

Abschließend stelle ich fest, dass ich die meisten der mir selbst gesteckten Ziele erreicht habe. Gern hätte ich noch den einen oder anderen Algorithmus in diese Arbeit mit einbezogen oder intensiver untersucht, wie zum Beispiel den derzeit schnellsten Ausdünnungs-Algorithmus (Kwok [28]), nur hätte dies nicht nur den Umfang der Arbeit deutlich erhöht, sondern auch den zeitlichen Rahmen gesprengt. So bleibt Platz für weitere Untersuchungen und künftige Aufgaben.

6.5 Danksagung

Ich möchte mich zunächst bei meinem Betreuer Prof. K.-U. Jahn bedanken, der mich nicht nur auf das Thema dieser Arbeit brachte, sondern mir stets mit hilfreichen Hinweisen und Ideen sehr geholfen hat.

Genauso bedanke ich mich bei den vielen Helfern, die mit guten Ratschlägen und Korrekturen ebenso zu helfen wussten, wie mit technischer Unterstützung bezüglich dem nicht immer einfach bedienbaren LaTeX.

A Lauflängenkodierung vs. "Distanz-Skelett-Kodierung"

Bei der Lauflängenkodierung macht man sich häufige aufeinander folgende Wiederholungen von gleichen Sequenzen zu Nutze und speichert die jeweilige Sequenz lediglich einmal zusammen mit der Anzahl der Wiederholungen. Für Binärbilder, deren Pixel bekanntlich zwei verschiedene Werte haben können, kann sogar auf das Abspeichern der Sequenz (schwarzes oder weißes Pixel) verzichtet werden. Es reicht die Anzahl der jeweiligen Pixel abzuspeichern, da auf eine Gruppe weißer Pixel nur schwarze Pixel folgen können und umgekehrt. Für eine Kodierung eines Binärbildes müssen zusätzlich noch die Abmaße des Bildes gespeichert werden. Außerdem wird sich darauf geeinigt, dass mit schwarzen Pixeln begonnen wird. Die Kodierung des Beispiels aus Abbildung A.1 (a) sieht dann folgendermaßen aus:

$$\underbrace{5}_{\text{Breite}}, \underbrace{6}_{\text{Höhe}}, \underbrace{0}_{\text{schwarz}}, \underbrace{3}_{\text{weiß}}, \underbrace{4}_{\text{schwarz}}, \underbrace{1,2,4,5,9,2}_{\ldots} \tag{A.1}$$

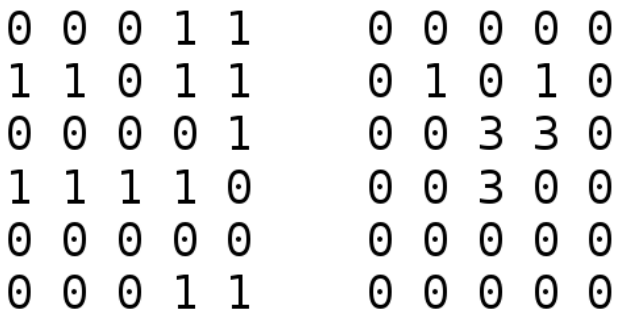


Abbildung A.1: Da es in Binärbildern (Beispiel (a)) nur zwei Werte von Pixeln gibt, bietet sich zur Abspeicherung solcher Bilder die Lauflängenkodierung an (siehe A.1). Um das Distanz-Skelett eines Bildes (Beispiel (b)) abzuspeichern, können alle Werte ungleich 0 einzeln erfasst werden (siehe A.2 und A.3). Anmerkung: (b) ist nicht das Distanz-Skelett von (a).

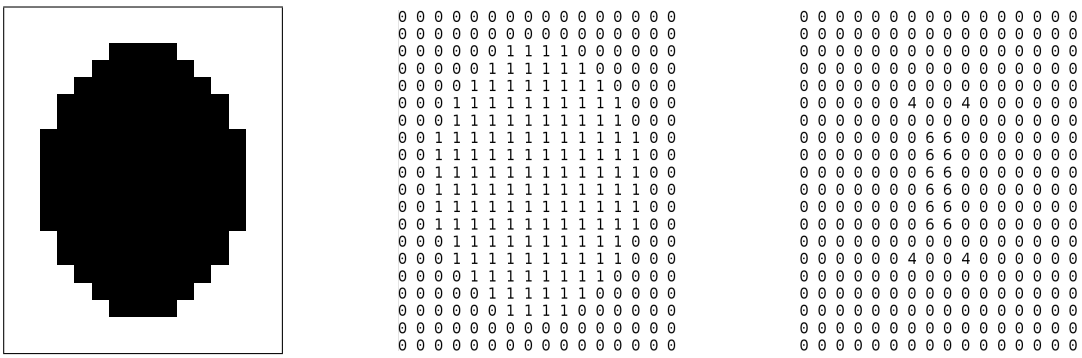
Es soll nun untersucht werden, ob die Kodierung mit Hilfe des Distanz-Skelettes eine echte Alternative bezüglich Speicherplatzeinsparungen darstellt. Für die "Distanz-Skelett-Kodierung"

müssen die Positionen der skelettalen Punkte zusammen mit dem jeweiligen Wert abgespeichert werden. Dies kann für das Distanz-Skelett auf zwei verschiedene Arten erfolgen. Die Position kann als Kombination von x- und y-Koordinate gespeichert werden, oder als Index des Pixelarrays. Für das Beispiel aus Abbildung A.1 (b) ergeben sich so folgende Kodierungen:

$$\underbrace{5}_{\text{Breite}},\underbrace{6}_{\text{Höhe}},\underbrace{1}_{\text{x}},\underbrace{1}_{\text{y}},\underbrace{1}_{\text{Wert}},\underbrace{3}_{\text{x}},\underbrace{1}_{\text{y}},\underbrace{1}_{\text{Wert}},\underbrace{2,2,3,3,2,3,2,3,3}_{\dots} \tag{A.2}$$

$$\underbrace{5}_{\text{Breite}},\underbrace{6}_{\text{Höhe}},\underbrace{6}_{\text{Index}},\underbrace{1}_{\text{Wert}},\underbrace{8}_{\text{Index}},\underbrace{1}_{\text{Wert}},\underbrace{12,3,13,3,17,3}_{\dots} \tag{A.3}$$

In dieser Arbeit wird die letztere Methode verwendet, da alle Zahlen als 32Bit Integer-Wert abgespeichert werden und so als Positionsangabe nur eine Zahl von Nöten ist. Da alle Werte mit 4 Byte abgespeichert werden, kann auch auf ein Seperator verzichtet werden[1].



(a) (b) (c)

Abbildung A.2: Für Bild (a) muss bei der Lauflängenkodierung das Array aus (b) abgespeichert werden, bei der "Distanz-Skelett-Kodierung" das Array aus (c) [BILD 7].

[1] Für beide Kompressionstechniken beträgt die maximale Wortlänge $m * n$ (Breite $*$ Höhe), allerdings werden sich die Wortlängen real stark unterscheiden. Die generelle Kodierung mit 4 Byte wurde nur zur vereinfachten Implementierung gewählt.

Die Lauflängenkodierung der Ellipse aus Abbildung A.2 sieht dann folgendermaßen aus (Zeilennummer / hexadezimale Darstellung / dezimale Darstellung):

```
0 | 10 00 00 00  14 00 00 00  00 00 00 00  26 00 00 00 | 016 020 000 038
1 | 04 00 00 00  0b 00 00 00  06 00 00 00  09 00 00 00 | 004 011 006 009
2 | 08 00 00 00  07 00 00 00  0a 00 00 00  06 00 00 00 | 008 007 010 006
3 | 0a 00 00 00  05 00 00 00  0c 00 00 00  04 00 00 00 | 010 005 012 004
4 | 0c 00 00 00  04 00 00 00  0c 00 00 00  04 00 00 00 | 012 004 012 004
5 | 0c 00 00 00  04 00 00 00  0c 00 00 00  04 00 00 00 | 012 004 012 004
6 | 0c 00 00 00  05 00 00 00  0a 00 00 00  06 00 00 00 | 012 005 010 006
7 | 0a 00 00 00  07 00 00 00  08 00 00 00  09 00 00 00 | 010 007 008 009
8 | 06 00 00 00  0b 00 00 00  04 00 00 00  26 00 00 00 | 006 011 004 038
```

Dem gegenüber gestellt ist die Kodierung des Distanz-Skelettes nach Methode A.3:

```
0 | 10 00 00 00  14 00 00 00  56 00 00 00  04 00 00 00 | 010 020 086 004
1 | 59 00 00 00  04 00 00 00  77 00 00 00  06 00 00 00 | 089 004 119 006
2 | 78 00 00 00  06 00 00 00  87 00 00 00  06 00 00 00 | 120 006 135 006
3 | 88 00 00 00  06 00 00 00  97 00 00 00  06 00 00 00 | 136 006 151 006
4 | 98 00 00 00  06 00 00 00  a7 00 00 00  06 00 00 00 | 152 006 167 006
5 | a8 00 00 00  06 00 00 00  b7 00 00 00  06 00 00 00 | 168 006 183 006
6 | b8 00 00 00  06 00 00 00  c7 00 00 00  06 00 00 00 | 184 006 199 006
7 | c8 00 00 00  06 00 00 00  e6 00 00 00  04 00 00 00 | 200 006 230 004
8 | e9 00 00 00  04 00 00 00                           | 233 004
```

In diesem Fall beansprucht die Distanz-Skelett-Kodierung mit 136 Byte weniger Speicherplatz als die Lauflängenkodierung mit 144 Byte. In A.3 und A.4 sind weitere Testbilder mit jeweils benötigtem Speicherplatz für beide Methoden angegeben.

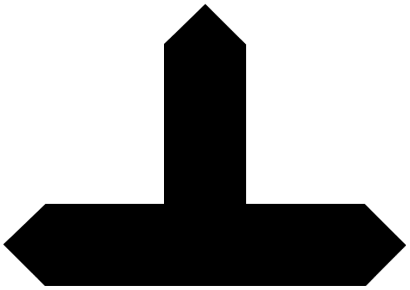
Erkennbar ist, dass für fast alle Testbilder die Lauflängenkodierung der Distanz-Skelett-Kodierung bezüglich des benötigten Speicherplatzes überlegen ist. Es fällt auf, dass Bilder mit vielen vertikalen Unterbrechungen, welche für die Lauflängenkodierung unvorteilhaft sind (etwa A.3 (f) und (g)), ebenfalls ungünstig für die Distanz-Skelett-Kodierung sind. Dies liegt daran, dass der Vorteil der Distanz-Skelett-Kodierung nur bei sehr großen, sich nach außen gleichmäßig ausbreitenden Flächen zum Tragen kommt. So verwundert es nicht, dass lediglich für sehr künstliche und geometrische Bilder, wie A.2, A.4 (d) und (j), das Distanz-Skelett die bessere Alternative darzustellen scheint. Gerade für solche Bilder gibt es mit Bildformaten die vektorielle Beschreibungen nutzen aber noch zusätzlich sehr effiziente Alternativen.

(a) LLK: 5408 Byte / DSK: 8824 Byte

(b) LLK: 2600 Byte / DSK: 5584 Byte

(c) LLK: 9816 Byte / DSK: 14208 Byte

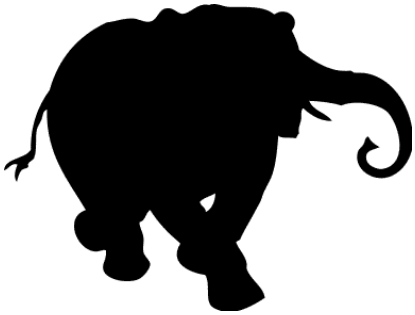
(d) LLK: 10008 Byte / DSK: 14208 Byte



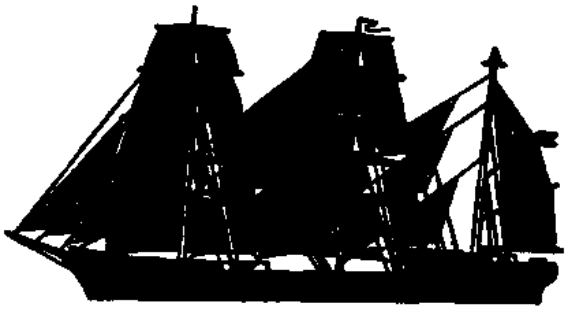
(e) LLK: 4168 Byte / DSK: 7288 Byte

(f) LLK: 58704 Byte / DSK: 89704 Byte

(g) LLK: 3472 Byte / DSK: 7064 Byte

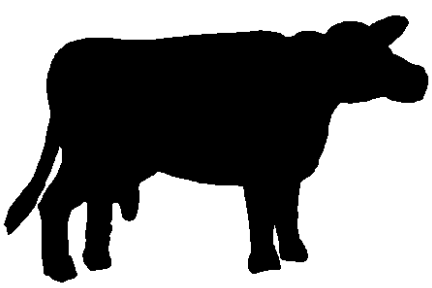


(h) LLK: 4600 Byte / DSK: 7576 Byte

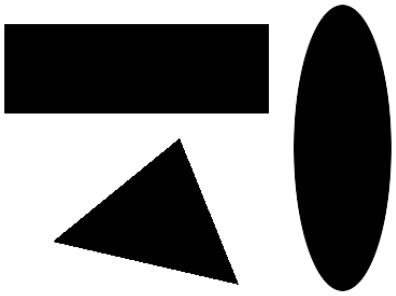


(i) LLK: 9944 Byte / DSK: 24592 Byte

Abbildung A.3: Vergleich des Speicherplatzbedarfs von Lauflängenkodierung (LLK) und Distanz-Skelett-Kodierung (DSK) I. Von links nach rechts, von oben nach unten: [BILD 3,2,9,13,34,33,32,6,30].



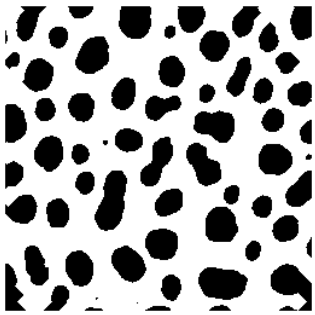
(a) LLK: 5000 Byte / DSK: 9112 Byte



(b) LLK: 3896 Byte / DSK: 6024 Byte

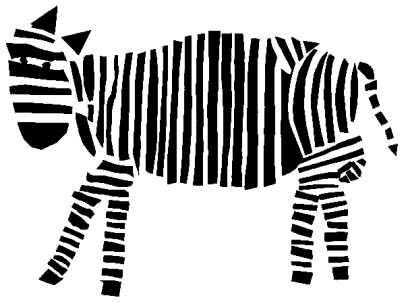
(c) LLK: 11400 Byte / DSK: 20080 Byte

(d) LLK: 1376 Byte / DSK: 1144 Byte



(e) LLK: 1376 Byte / DSK: 1144 Byte

(f) LLK: 11392 Byte / DSK: 13520 Byte

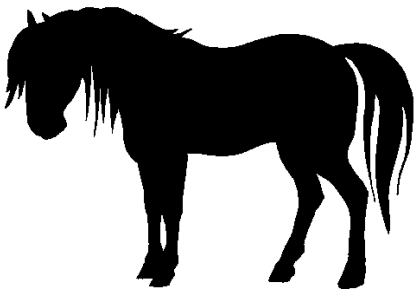


(g) LLK: 40704 Byte / DSK: 72272 Byte

(h) LLK: 3664 Byte / DSK: 5056 Byte

(i) LLK: 6312 Byte / DSK: 9744 Byte

(j) LLK: 976 Byte / DSK: 808 Byte



(k) LLK: 11664 Byte / DSK: 18400 Byte

Abbildung A.4: Vergleich des Speicherplatzbedarfs von Lauflängenkodierung (LLK) und Distanz-Skelett-Kodierung (DSK) II. Von links nach rechts, von oben nach unten: [BILD 20,11,3,17,25,31,35,12,14,16,26].

B Vergleich von Rechenzeiten anhand konkreter Beispiele

In dieser Arbeit wurden für die verschiedenen Skelettierungsalgorithmen unter anderem die Laufzeitkomplexitäten in *Landau-Notation* angegeben. Diese lassen, wie bereits erwähnt, für die Praxis nur bedingt nützliche Aussagen zu, vor allem wenn mehrere Algorithmen die gleiche Komplexität besitzen. Beispielsweise haben die meisten Ausdünnungs-Algorithmen eine Laufzeit von $\mathcal{O}(n^3)$. Um eventuell bessere Aussagen machen zu können, sollen im Folgenden die implementierten Algorithmen anhand drei verschiedener Bilder verglichen werden. Dafür wurde jedes Bild in unterschiedlichen Auflösungen, also für unterschiedliche Bildbreiten n, skelettiert ($n \times n$ Bilder). Dies geschah für die Bildbreiten 100, 200, 400, 600, 800, 1 000, 1 500 und 2 000 Pixel.

Für den Vergleich wurden drei möglichst verschiedenartige Bilder gewählt.

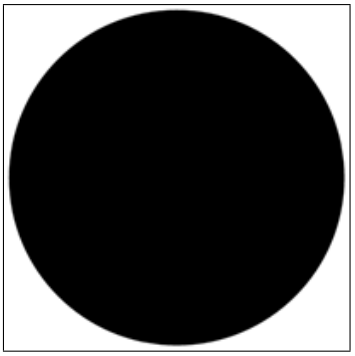


Bild 1 (links) besitzt sehr viele Objektpixel bei einer relativ geringen Anzahl von Konturpixeln. Das Bild setzt sich bezogen auf die Gesamtzahl der Pixel n^2 wie folgt zusammen:

- $75,88\%$ Objektpixel (exklusive der Konturpixel)
- $0,14\%$ Konturpixel (8er-Zusammengehörigkeit)
- $23,98\%$ Hintergrundpixel

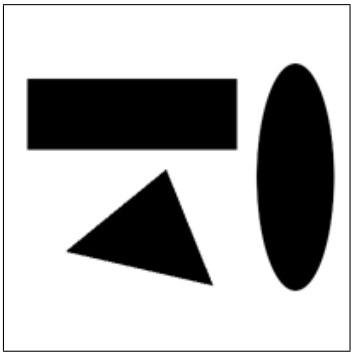


Bild 2 (links) soll ein "Durchschnittsbild" sein, mit einigen wenigen Objekten und viel Hintergrund. Das Bild setzt sich bezogen auf die Gesamtzahl der Pixel n^2 wie folgt zusammen:

- $31,95\%$ Objektpixel (exklusive der Konturpixel)
- $0,23\%$ Konturpixel (8er-Zusammengehörigkeit)
- $67,82\%$ Hintergrundpixel

Bild 3 (links) besitzt eine relativ hohe Anzahl an Konturpixeln, ebenfalls bei viel Hintergrund. Das Bild setzt sich bezogen auf die Gesamtzahl der Pixel n^2 wie folgt zusammen:

- $22,52\%$ Objektpixel (exklusive der Konturpixel)
- $1,27\%$ Konturpixel (8er-Zusammengehörigkeit)
- $76,21\%$ Hintergrundpixel

Die Rechenzeiten aus den nachfolgenden Tabellen wurden mit Hilfe des Plugins ermittelt. Zum Einsatz kam ein Thinkpad X61s, Core 2 Duo L7700, 2x1,8 GHz, 2048MB RAM. Die Mehrprozessor-Fähigkeiten des Systems wurden nicht genutzt, die Berechnungen fanden also nur auf einem Prozessor statt. In den Abbildungen B.1, B.2 und B.3 wurden die ermittelten Werte in Diagramme übertragen. Im jeweils ersten Diagramm wurde eine lineare Skala für die Rechenzeiten benutzt, im zweiten eine logarithmische Skalierung.

Die Ergebnisse bestätigen einige gemachte Aussagen dieser Arbeit. Wie erwartet ist der Algorithmus zur Berechnung des einfachen Kritische-Punkte-Skelettes am schnellsten. Mit $\mathcal{O}(n^2)$ und nur einem benötigten Rasterscan ist dieses Skelett schneller zu berechnen als das Distanz-Skelett, ebenfalls mit $\mathcal{O}(n^2)$, aber drei benötigten Rasterscans. Die drei Algorithmen für die Ausdünnungs-Skelette (morphologisches Skelett, Hilditch-Skelett und Zhang/Suen-Skelett) folgen mit $\mathcal{O}(n^3)$. Dabei fällt der relativ große Abstand zum Zhang/Suen-Skelett auf, dessen Berechnung wesentlich weniger Zeit benötigt als beispielsweise die Berechnung des Hilditch-Skelettes. Der Autor führt dies auf die in Abschnitt 5.3 beschriebene Modifikation für parallele Ausdünnungs-Algorithmen zurück. Die Konturpunkte für die jeweils nächste Iteration werden in sehr effizienten einfach-verketteten Listen gespeichert. Dadurch können die Mengenoperationen aus Pseudocode 13 in kurzer Rechenzeit umgesetzt werden. Die Berechnung der Medialen Achse dauert für den implementierten Algorithmus mit $\mathcal{O}(n^4)$ am längsten.

Weiterhin fällt auf, dass die Berechnung der Medialen Achse für das Bild 3 (Zebra) ungefähr drei mal so lange dauert wie für Bild 1 (Kreis), obwohl Bild 1 wesentlich mehr Objektpixel enthält. Der Grund hierfür ist, dass sich bei beiden das Produkt aus der Anzahl von Konturpixeln und der Anzahl innerer Objektpixel stark unterscheidet. Für Bild 1 finden bezogen auf Pseudocode 1 $0,7588n^2 * 0,0014n^2 = 0,00106n^4$ Schleifendurchläufe statt. Für Bild 3 sind es hingegen $0,2252n^2 * 0,0127n^2 = 0,00286n^4$ Schleifendurchläufe (siehe Prozentzahlen von oben). Das Verhältnis zwischen beiden Zahlen beträgt, wie auch das Verhältnis der Rechenzeiten, ungefähr 3:1.

Es sei betont, dass auch ein solcher Vergleich anhand konkreter Beispiele nur eine begrenzte Aussagekraft hat. Er liefert jedoch erste gute Hinweise, die es in der Praxis erleichtern, die Algorithmen bezüglich der Rechenzeiten zu bewerten.

Bild 1 (Kreis):

Skelett \ Bildbreite n	2 000	1 500	1 000	800	600	400	200	100
Mediale Achse	574 319	252 951	72 481	37 276	15 857	4 736	634	141
Distanz-Skelett	257	191	96	77	45	24	9	6
krit.-Punkte-Skelett	173	87	41	28	18	11	6	5
morph. Skelett	211 106	86 340	25 304	12 838	5 485	1 716	250	70
Hilditch-Skelett	189 872	84 142	25 134	12 995	5 542	1 722	279	72
Zhang/Suen-Skelett	2 056	1 190	534	350	217	130	50	16

Bild 2 (Geometr. Objekte):

Skelett \ Bildbreite n	2 000	1 500	1 000	800	600	400	200	100
Mediale Achse	396 331	161 938	47 406	23 920	9 985	2 718	401	94
Distanz-Skelett	269	151	83	68	41	21	9	6
krit.-Punkte-Skelett	96	73	41	28	18	10	6	5
morph. Skelett	67 759	27 975	8 111	4 235	1 766	582	105	25
Hilditch-Skelett	34 543	15 130	4 255	2 374	1 013	368	86	19
Zhang/Suen-Skelett	2 186	802	297	220	142	98	29	11

Bild 3 (Zebra):

Skelett \ Bildbreite n	2 000	1 500	1 000	800	600	400	200	100
Mediale Achse	1 611 626	660 934	185 456	91 596	34 626	11 326	1 505	154
Distanz-Skelett	243	176	100	75	42	22	10	6
krit.-Punkte-Skelett	132	84	42	29	18	10	7	5
morph. Skelett	17 774	7 161	2 326	1 153	477	195	53	29
Hilditch-Skelett	6 571	2 963	887	490	245	113	46	23
Zhang/Suen-Skelett	2 204	1 095	442	276	171	97	57	28

Die Bilder sind in den entsprechenden Auflösungen auf dem Datenträger beigelegt. Mit Hilfe des Plugins können die Rechenzeiten für die implementierten Algorithmen ermittelt werden. Es ist zu beachten, dass diese sich wahrscheinlich von den hier angegebenen Ergebissen unterscheiden.

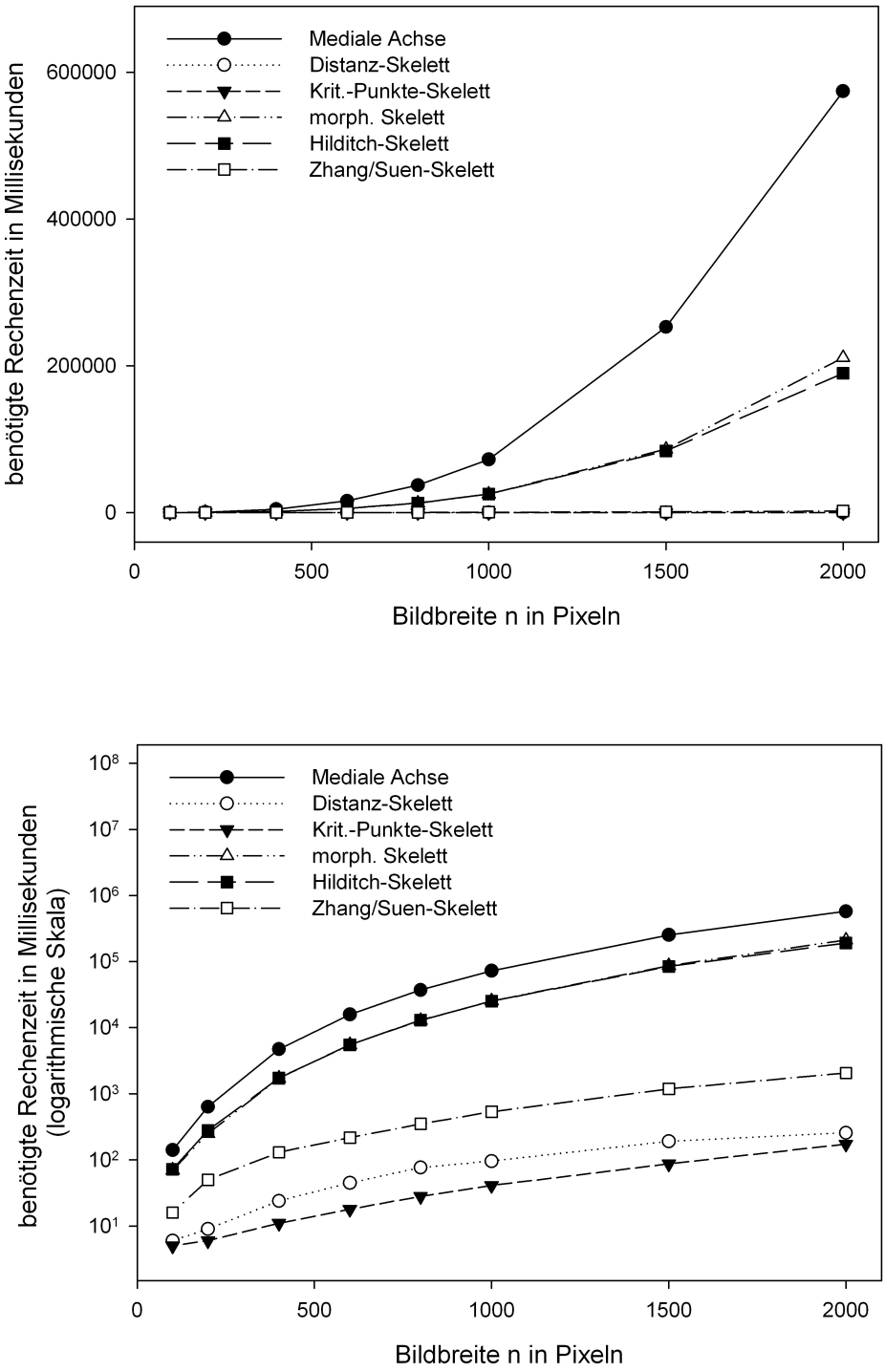


Abbildung B.1: Vergleich der Rechenzeiten der implementierten Algorithmen für Bild 1: Obwohl der Hilditch-Algorithmus und der Zhang/Suen-Algorithmus die gleiche Laufzeitkomplexität besitzen, unterscheiden sich die konkreten Rechenzeiten für dieses Beispiel deutlich voneinander. Der Zhang/Suen-Algorithmus ist dank effizienter Implementierung ungefähr 100-mal schneller als der Hilditch-Algorithmus.

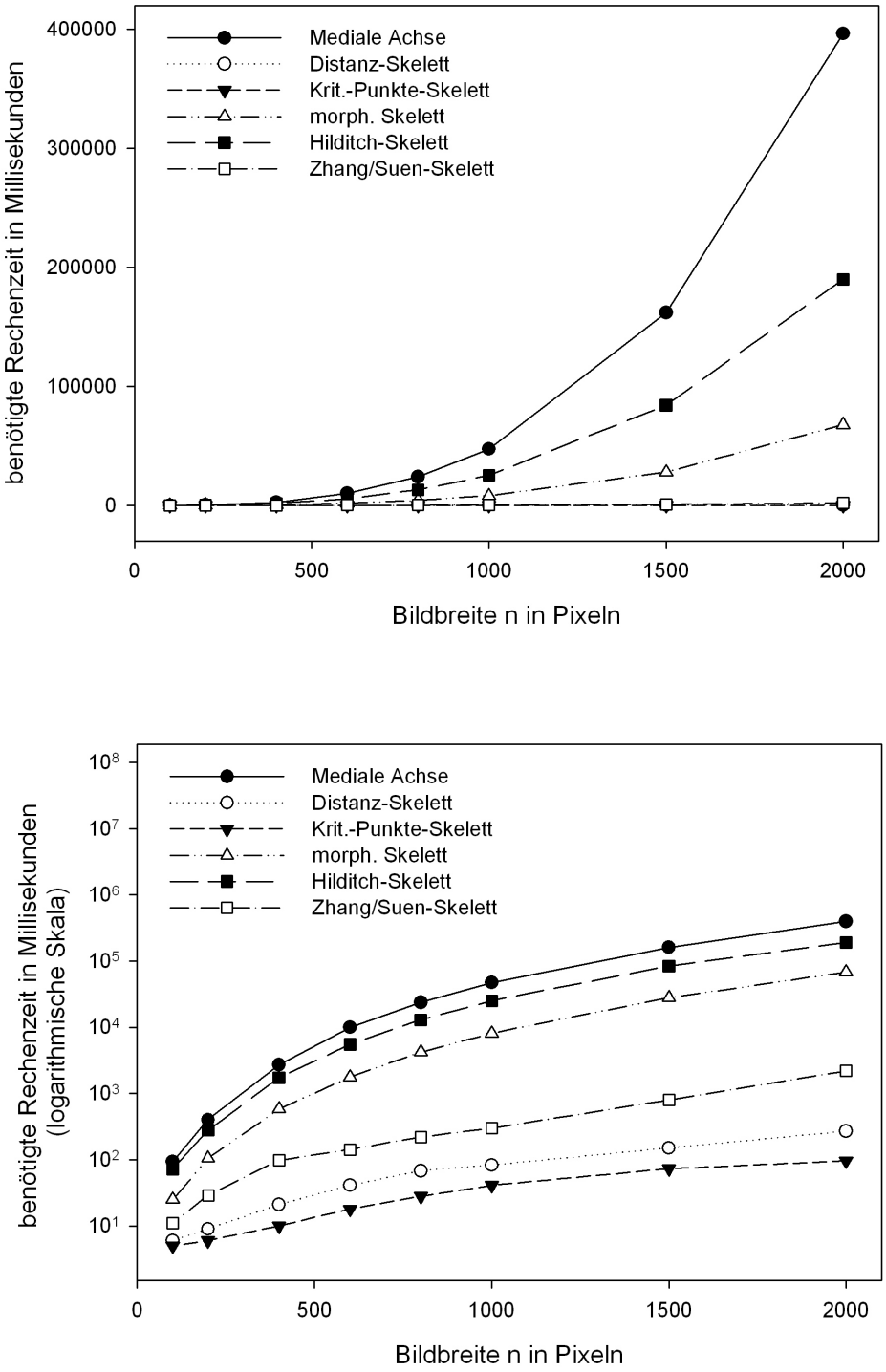


Abbildung B.2: Vergleich der Rechenzeiten der implementierten Algorithmen für Bild 2: Auch bei diesem Beispiel ist ein deutlicher Unterschied bezüglich der Rechenzeit zwischen dem Hilditch-Algorithmus und dem Zhang/Suen-Algorithmus erkennbar. Die Berechnung der Medialen Achse benötigt ungefähr doppelt so viel Zeit wie der Hilditch-Algorithmus und ist damit deutlich schneller als für Bild 3 (nächstes Beispiel).

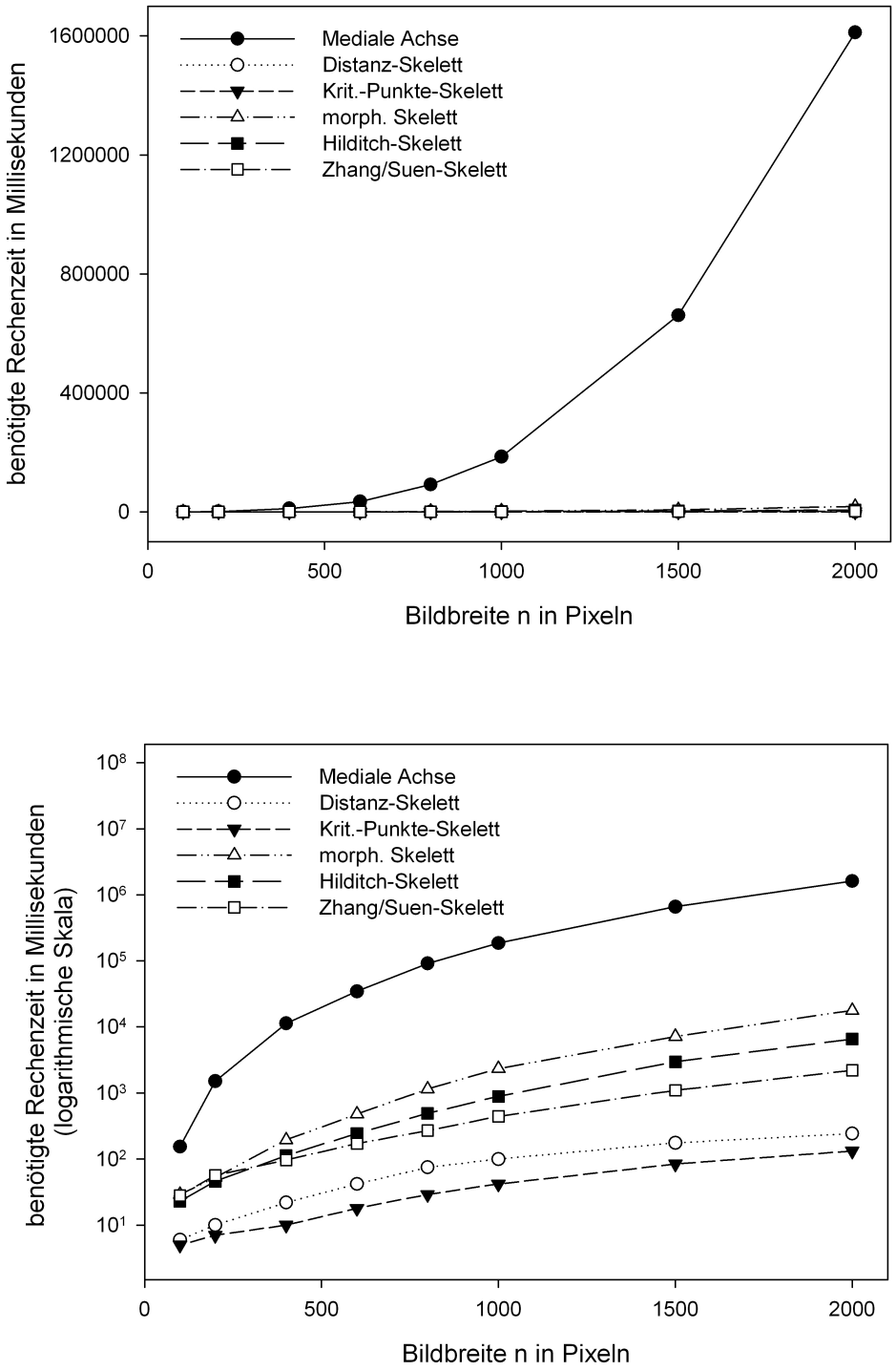


Abbildung B.3: Vergleich der Rechenzeiten der implementierten Algorithmen für Bild 3: Aufgrund der hohen Anzahl von Konturpixeln ist die Berechnung der Medialen Achse im Vergleich zu den vorherigen Beispielen sehr viel langsamer. Während sich bei Bild 1 und 2 ein Faktor von 2 gegenüber dem Hilditch-Algorithmus ergibt, ist die Berechnung der Medialen Achse hier ungefähr 100 mal langsamer als der Hilditch-Algorithmus.

C Implementierung des Plugins

Das für diese Arbeit entwickelte Plugin baut auf dem quelloffene Bildverarbeitungsprogramm *ImageJ*[1] auf. Dieses wurde komplett in Java geschrieben. ImageJ bietet verschiedene Möglichkeiten, um eigenen Code einzubinden: *Macros*, *Plugins*, *Plugin-Filter*, *Plugin-Frames*. Für diese Arbeit wurde ein Plugin-Frame verwendet, da so bei Ausführung des Plugins ein eigenes Fenster geöffnet wird, welches sich vom Programmierer frei gestalten lässt. Es wurde eine Oberfläche erstellt, die einzelne Buttons enthält, mit denen der Anwender in der Lage ist, die verschiedenen Algorithmen zu starten. Daneben gibt es einige Optionsfelder zur Modifizierung der Algorithmen.

Um ein Plugin-Frame zu erstellen, muss lediglich die Klasse *ij.plugin.frame.PlugInFrame* aus ImageJ vererbt werden. Eine Implementierung eines Interfaces ist nicht nötig. Die neu erstellte Klasse wird dann von ImageJ über den leeren Standard-Konstruktur aufgerufen.

In der nachstehenden Tabelle sind alle Klassen des Plugins aufgelistet.

`Skeleton_PluginFrame`	Plugin-Frame welches die Oberfläche enthält und von ImageJ gestartet wird
`data/Pixel`	Klasse zur Repräsentation eines Pixels
`data/PixelInfo`	Klasse zur Ber. von $b_\alpha(p,I)$,$N_\alpha(p)$,$X_R(p,I)$,$X_H(p,I)$ usw.
`data/StoreInHex`	Klasse zur Abspeicherung errechneter Skelette
`data/SkeletonException`	Eigene Exception zur Fehlerbehandlung
`data/Skeleton_CriticalPoint`	Klasse zur Ber. des einfachen Kritische-Punkte-Ansatzes
`data/Skeleton_Distance`	Klasse zur Ber. des Distanz-Skelettes
`data/Skeleton_Hilditch`	Klasse zur Ber. des Hilditch-Skelettes
`data/Skeleton_MedialAxis`	Klasse zur Ber. der Medialen Achse
`data/Skeleton_Morphologic`	Klasse zur Ber. des moprhologischen Skelettes
`data/Skeleton_ZhangSuen`	Klasse zur Ber. des Zhang/Suen-Skelettes

Die Klassen zur Berechnung der verschiedenen Skelette (`data/Skeleton_XXX`) wurden von der Art und Weise des Zugriffes standardisiert. Um ein Skelett zu berechnen, müssen folgende zwei Methoden aufgerufen werden:

- `public Skeleton_XXX(ImageProcessor imageProcessor, boolean periodically)`

[1] `http://rsb.info.nih.gov/ij/`

- `public ImageProcessor compute()`

In diesen Klassen kann dann auf die Klasse `PixelInfo`, welche unter anderem Nachbarschaftsberechnungen enthält und die Pixelwerte entsprechend der hier verwendeten Notation umwandelt, d.h. 0 - Hintergrundpixel und 1 - Objektpixel, zugegriffen werden:

- `protected PixelInfo(byte[] pixels, int width, int height, boolean repeatPeriodically, int borderValue, boolean binaryOutput)`

Auf eine Abstraktion der Klassen zur Berechnung der Skelette auf ein Interface wurde bewusst verzichtet, da es für einzelne Klassen noch Modifizierungsmöglichkeiten gibt, welche mit seperaten Methoden angegeben werden müssen.

Um den Vergleich zwischen der Distanz-Skelett-Kodierung und der Lauflängenkodierung von Binärbildern (siehe Anhang A) nachvollziehen zu können, wurde eine Option eingefügt, welche die entsprechend kodierten Dateien auf der Festplatte des Users ablegt. Unter UNIX sind diese Dateien im home-Verzeichnis zu finden. Unter Windows werden sie unter `C:/Dokumente und Einstellungen/User` gespeichert. Für die in Anhang B vorgenommenen Vergleiche der Rechenzeiten wird nach jeder Skelettberechnung die benötigte Zeit im Plugin-Fenster angezeigt.

Weitere Informationen zum Download und zur Installation des Plugins sind im Abschnitt 6.2 zu finden.

D Inhalt der CD

Der Arbeit ist eine CD beigelegt, welche alle zur Ausführung der entwickelten Software nötigen Dateien enthält, sowie die für diese Arbeit angelegte Bildersammlung.

Das Hauptverzeichnis enthält die Dateien:

`Bachlorarbeit_Christoph_Bullmann.pdf`
`info.txt`
`ImageJ_140.zip`
`ImageJ_140_jdk6_setup.exe`
`ImageJ_140_x86.tar.gz`
`Plugin_Skelettberechnung.jar`

Der Ordner `quellcode` enthält alle Quelldateien, die zum ImageJ-Plugin gehören:

`quellcode/plugins.config`
`quellcode/src/Skeleton_PluginFrame.java`
`quellcode/src/data/Pixel.java`
`quellcode/src/data/PixelInfo.java`
`quellcode/src/data/Skeleton_CriticalPoint.java`
`quellcode/src/data/Skeleton_Distance.java`
`quellcode/src/data/Skeleton_Hilditch.java`
`quellcode/src/data/Skeleton_MedialAxis.java`
`quellcode/src/data/Skeleton_Morphologic.java`
`quellcode/src/data/Skeleton_ZhangSuen.java`
`quellcode/src/data/SkeletonException.java`
`quellcode/src/data/StoreInHex.java`

Die Bildersammlung befindet sich im Ordner `bilder` und umfasst folgende Dateien:

`bilder/01_aaaa.png`
`bilder/02_abc_xyz.png`
`bilder/03_adler.png`
`bilder/04_bundesadler.png`
`bilder/05_bundesadler_gefuellt.png`

bilder/06_elefant.png
bilder/07_ellipse_gerade.png
bilder/08_ellipse_ungerade.png
bilder/09_familie.png
bilder/10_flasche.png
bilder/11_geometr_objekte.png
bilder/12_hand.png
bilder/13_htwk.png
bilder/14_kinder.png
bilder/15_kleine_objekte.png
bilder/16_kreis.png
bilder/17_kreuz.png
bilder/18_kreuz_22.png
bilder/19_kreuz_45.png
bilder/20_kuh.png
bilder/21_kuh2.png
bilder/22_loeffel.png
bilder/23_mensch.png
bilder/24_messer.png
bilder/25_mikroskopbild.png
bilder/26_pferd.png
bilder/27_rechteck.png
bilder/28_rechteck_gerade.png
bilder/29_rechteck_ungerade.png
bilder/30_schiff.png
bilder/31_schlange.png
bilder/32_strauss.png
bilder/33_tanzende_leute.png
bilder/34_triang_objekt.png
bilder/35_zebra.png

Die Bilder für den in Anhang B durchgeführten Vergleich der Rechenzeiten befinden sich im Ordner `bilder_tests`:

bilder_tests/kreis_2000.png
bilder_tests/kreis_1500.png
bilder_tests/kreis_1000.png
bilder_tests/kreis_800.png
bilder_tests/kreis_600.png
bilder_tests/kreis_400.png

`bilder_tests/kreis_200.png`
`bilder_tests/kreis_100.png`
`bilder_tests/geom_2000.png`
`bilder_tests/geom_1500.png`
`bilder_tests/geom_1000.png`
`bilder_tests/geom_800.png`
`bilder_tests/geom_600.png`
`bilder_tests/geom_400.png`
`bilder_tests/geom_200.png`
`bilder_tests/geom_100.png`
`bilder_tests/zebra_2000.png`
`bilder_tests/zebra_1500.png`
`bilder_tests/zebra_1000.png`
`bilder_tests/zebra_800.png`
`bilder_tests/zebra_600.png`
`bilder_tests/zebra_400.png`
`bilder_tests/zebra_200.png`
`bilder_tests/zebra_100.png`

Der Ordner `literatur` enthält alle Quellen dieser Arbeit, die dem Autor in digitaler Form zur Verfügung standen. Auf eine Auflistung wird an dieser Stelle aufgrund des großen Umfangs vezichtet.

Glossar

Adjazenz Adjazenz beschreibt eine Beziehung zwischen zwei Pixeln. In dieser Arbeit wurden die 4-Adjazenz und die 8-Adjazenz definiert (siehe Seite 13).

Ausdünnen Das Ausdünnen ist ein prinzipielles Verfahren einer Skelettberechnung (siehe Abschnitt 5).

Bitmap Das Bitmap-Format ist ein verlustfreies Format zur Abspeicherung eines Rasterbildes. Typischerweise findet keine Datenkompression statt, was dazu führt, dass es im Vergleich zu anderen moderneren Formaten wie PNG sehr viel Speicherplatz benötigt.

Delaunay-Triangulation Die Delaunay-Triangulation ist eine besondere Form der Triangulation (siehe Seite 34).

Dilatation Die Dilatation ist eine morphologische Grundoperation, welche als nicht-lineares Filter definiert werden kann (siehe Seite 45).

Distanz-Karte Eine Distanz-Karte für ein Binärbild gibt für jedes Pixel des Bildes an, wie groß der Abstand des Pixels zu dem nächstgelegenen Hintergrundpixel ist. Für die Berechnung einer Distanz-Karte ist die Festlegung einer Metrik von Nöten (siehe Seite 22).

Distanz-Skelett Das Distanz-Skelett eines Bildes ist ein Skelett, welches auf Distanz-Transformationen beruht und mit Hilfe der Distanz-Karte berechnet wird (siehe Abschnitt 3.2).

Erosion Die Erosion ist eine morphologische Grundoperation, welche als nicht-lineares Filter definiert werden kann (siehe Seite 45).

Euler-Zahl Die Euler-Zahl $N_E(\mathcal{R})$ ist ein topologisches Merkmal einer Region $\mathcal{R}$ (siehe Seite 9).

Hilditch-Kreuzungszahl Die Hilditch-Kreuzungszahl $X_R(p, I)$ kommt bei den Ausdünnungs-Algorithmen zum Einsatz (siehe Seite 51).

ideal thin Eine Anforderung an Skelette ist oftmals, dass es *ideal thin* ist. Das bedeutet, dass das Skelett "ein-Pixel-breit" ist. Dies wird beispielsweise durch die Anwendung von Ausdünnungs-Algorithmen erreicht, wenn bei 8er-Zusammengehörigkeit alle simplen Pixel gelöscht werden.

ImageJ ImageJ ist ein quelloffenes Bildverarbeitungsprogramm, welches bei der Implementierung der Algorithmen aus dieser Arbeit verwendet wurde.

Impulsrauschen Durch Impulsrauschen (auch Salt-and-Pepper-Rauschen) enstehen im Bild einzelne fehlerhafte weiße und schwarze Pixel. Durch diese Fehler können besonders bei Ausdünnungs-Algorithmen viele unnötige Äste entstehen, da sehr streng die topologische Struktur eingehalten wird.

Isotropie Isotrophie ist die Unabhängigkeit einer Eigenschaft von einer Richtung.

Kreuzungszahl Die Kreuzungszahl wurde unter anderem von Rutovitz und Hilditch definiert. Sie wird insbesondere für die Ausdünnungs-Algorithmen benötigt (siehe Seite 51).

kritische Punkte Kritische Punkte werden bei speziellen Skelettierungsalgorithmen ermittelt (siehe Abschnitt 4).

Kritische-Punkte-Algorithmus Kritische-Punkte-Algorithmen sind prinzipielle Verfahren einer Skelettberechnung (siehe Abschnitt 4).

Landau-Notation Die Landau-Notation findet in dieser Arbeit Verwendung, um für die vorgestellten Algorithmen eine Laufzeitkomplexität anzugeben. Mit Hilfe der Landau-Notation kann das asymptotische Verhalten der Algorithmen angegeben werden.

Lauflängenkodierung Die Lauflängenkodierung ist eine Kompressionstechnik. Sie ist besonders effizient bei häufigen Wiederholungen gleicher Sequenzen (siehe Seite I).

Mediale Achse Die Mediale Achse ist eine Definition eines Skelettes (siehe Abschnitt 3.1).

Metrik Eine Metrik definiert einen Abstand zwischen zwei Pixeln. In dieser Arbeit werden drei verschiedene Metriken verwendet: Betragssummenmetrik, euklidische Metrik und Maximummetrik (siehe Seite 16).

morphologische Skelettberechnung Die morpholosche Skelettberechnung ist eine besondere Art der Ausdünnung, da sie nur mit den morphologischen Grundoperationen Erosion und Dilatation arbeitet (siehe Abschnitt 5.1).

Nachbarschaft Ein Nachbarschaftsbegriff definiert die Menge der Pixel, die zu einem bestimmten Pixel benachbart sind. In dieser Arbeit werden zwei verschiedene Nachbarschaftsbegriffe verwendet: 4er-Nachbarschaft und 8er-Nachbarschaft (siehe Seite 14).

Opening Eine Opening-Operation ist die Hintereinanderausführung von *Erosion* und *Dilatation* mit derselben Maske (siehe Seite 46).

parallele Ausdünnungs-Algorithmen Parallele Ausdünnungs-Algorithmen sind eine Unterkategorie der Ausdünnungs-Algorithmen. Dabei können alle Pixel innerhalb einer Iteration unabhängig voneinander bearbeitet werden (siehe Abschnitt 5.3).

pruning "pruning" (engl.) bezeichnet das Nachbehandeln von Skeletten - insbesondere das Kürzen oder Abschneiden von unwichtigen/störenden Ästen.

Rutovitz-Kreuzungszahl Die Rutovitz-Kreuzungszahl $X_R(p, I)$ kommt bei den Ausdünnungs-Algorithmen zum Einsatz (siehe Seite 51).

sequentielle Ausdünnungs-Algorithmen Sequentielle Ausdünnungs-Algorithmen sind eine Unterkategorie der Ausdünnungs-Algorithmen. Dabei kann ein Pixel innerhalb einer Iteration nicht unabhängig von den anderen Pixeln bearbeitet werden. Dies hat zur Folge, dass eine feste Reihenfolge bei der Abarbeitung der Pixel nötig ist (siehe Abschnitt 5.2).

simples Pixel Bei Ausdünnungs-Algorithmen wird ein Pixel simpel genannt, wenn es gelöscht werden kann, ohne dass sich die Zusammengehörigkeit ändert (siehe Seite 43).

Skelettieren Mit dem Skelettieren ist der Vorgang der Skelettberechnung gemeint.

Standardrasterscan-Reihenfolge Unter der Standardrasterscan-Reihenfolge ist die Abarbeitung der Pixel eines Bildes von links nach rechts, von oben nach unten gemeint.

symmetrische Pixel Bei der Definition der Medialen Achse ist ein Pixel symmetrisch, wenn der kürzeste Abstand zu einem Konturpixel zu einem weiteren Konturpixel existiert. Ist ein Pixel symmetrisch, ist es Teil der Medialen Achse (siehe Seite 17).

topologische Eigenschaften Topologische Eigenschaften sind Merkmale, die sich nicht explizit auf die konkrete Form einer Region beziehen, sondern auf ihre strukturellen Eigenschaften, die auch bei starken Verformungen erhalten bleiben.

Triangulation Bei der Triangulation wird aus einer Punktmenge ein Dreiecksnetz erstellt (siehe Abschnitt 4.2).

Voronoi-Diagramm Bei einem Voronoi-Diagramm wird der Raum derart in Polygone zerlegt, dass jeder Punkt einer vorgegebenen Punktmenge Zentrum eines Polygons wird. Für alle Punkte innerhalb eines Polygons ist das Zentrum der nächste Punkt der vorgegebenen Punktmenge. Mit Hilfe eines solchen Diagrammes, kann die Mediale Achse angenähert werden.

Zusammengehörigkeit Die Zusammengehörigkeit von Objekten ist für topologische Eigenschaften wichtig. In dieser Arbeit wurden zwei verschiedene Zusammengehörigkeitsbegriffe definiert: 4-zusammengehörig und 8-zusammengehörig (siehe Seite 42).

Variablenverzeichnis

$b_\alpha(p, I)$	Anzahl der α-adjazenten Objektpixel von p im Bild I	14
$\mathbb{C}$	Menge von Pixeln als Trägermenge eines Bildes	13
$C_\alpha(I)$	Menge aller α-Konturpixel des Bildes I	14
$d_1(p, q)$	Betragssummenmetrik (auch Manhattan- oder Schachbrett-Metrik)	16
$d_2(p, q)$	euklidische Metrik	16
$d_\infty(p, q)$	Maximummetrik	16
I	Bild, bestehend aus Vorder- und Hintergrundpixeln	13
$\langle I \rangle$	Menge aller Objektpunkte des Bildes I	13
n	Breite eines Bildes in Anzahl von Pixeln	15
$N_\alpha(p)$	Menge aller α-adjazenten Pixel von p	14
$N_E(\mathcal{R})$	Euler-Zahl einer Region $\mathcal{R}$	9
p	Pixel bestehend aus den Koordinaten p_x und p_y	13
p_x	x-Koordinate eines Pixels p	13
p_y	y-Koordinate eines Pixels p	13
S	Skelett als Bild, bestehend aus Vorder- und Hintergrundpixeln	13
$\langle S \rangle$	Menge aller skelettalen Pixel des Skelettes S	13
$X_H(p, I)$	Hilditch-Kreuzungszahl eines Pixels p bezogen auf das Bild I	51
$X_R(p, I)$	Rutovitz-Kreuzungszahl eines Pixels p bezogen auf das Bild I	51
$\ominus$	Erosion-Operationszeichen	45
$\oplus$	Dilatation-Operationszeichen	45

Abbildungsverzeichnis

Literaturverzeichnis

[1] M. Altuwaijri and M. Bayoumi. A new thinning algorithm for arabic characters using self-organizing neural network. In M. Bayoumi, editor, *Proc. IEEE International Symposium on Circuits and Systems ISCAS '95*, volume 3, pages 1824–1827 vol.3, 1995.

[2] M. Altuwaijri and M. Bayoumi. A thinning algorithm for arabic characters using art2 neural network. *IEEE Trans. Circuits Syst. II*, 45(2):260–264, 1998.

[3] C. Arcelli, L. Cordella, and S. Levialdi. Parallel thinning of binary pictures. *Electronics Letters*, 11(7):148–149, 1975.

[4] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung*. Springer Verlag Berlin, 2 edition, 2005/2006.

[5] Kenneth R. Castleman. *Digital Image Processing*. Prentice Hall, 1996.

[6] Fu Chang, Yung-Ping Cheng, T. Pavlidis, and Tsuey-Yuh Shuai. A line sweep thinning algorithm. In Yung-Ping Cheng, editor, *Proc. Third International Conference on Document Analysis and Recognition*, volume 1, pages 227–230 vol.1, 1995.

[7] Fu Chang, Ya-Ching Lu, and T. Pavlidis. Feature analysis using line sweep thinning algorithm. *IEEE Trans. Pattern Anal. Machine Intell.*, 21(2):145–158, 1999.

[8] Hung-Hsin Chang and Hong Yan. Analysis of stroke structures of handwritten chinese characters. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 29(1):47–61, 1999.

[9] Yung-Sheng Chen and Wen-Hsing Hsu. A modified fast parallel algorithm for thinning digital patterns. *Pattern Recogn. Lett.*, 7(2):99–106, 1988.

[10] Jen-Hui Chuang, Chi-Hao Tsai, and Min-Chi Ko. Skeletonisation of three-dimensional object using generalized potential field. *IEEE Trans. Pattern Anal. Machine Intell.*, 22(11):1241–1251, Nov. 2000.

[11] N.D. Cornea, D. Silver, and P. Min. Curve-skeleton applications. In *Proc. IEEE Visualization VIS 05*, pages 95–102, 2005.

[12] N.D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Trans. Visual. Comput. Graphics*, 13(3):530–548, 2007.

[13] Nicu Cornea. Computing hierarchical curve-skeletons of 3d objects. `http://www.caip.rutgers.edu/~cornea/Skeletonization/`, November 2005. Stand vom 10.08.2008.

[14] Nicu Cornea, Deborah Silver, Xiaosong Yuan, and Raman Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21:945–955, 2005.

[15] Tamal K. Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 143–152, 2006.

[16] Ulrich Eckardt. A note on rutovtz' method for parallel thinning. *Pattern Recogn. Lett.*, 8(1):35–38, 1988.

[17] E.L. Flores. A fast thinning algorithm. In *Proc. SBT/IEEE International Telecommunications Symposium ITS '98*, volume 2, pages 594–599, 9–13 Aug. 1998.

[18] Mark Foskey, Ming C. Lin, and Dinesh Manocha. Efficient computation of a simplified medial axis. *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 96–107, 2003.

[19] H. Fujiyoshi and A.J. Lipton. Real-time human motion analysis by image skeletonization. In *Proc. Fourth IEEE Workshop on Applications of Computer Vision WACV '98*, pages 15–21, 1998.

[20] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley, 2007.

[21] R. W. Hall. Fast parallel thinning algorithms: parallel speed and connectivity preservation. *Commun. ACM*, 32(1):124–131, 1989.

[22] L. Hayat, A. Naqvi, and M.B. Sandler. Parallel implementation of a fast thinking algorithm using image compression. *IEE Proceedings I Communications, Speech and Vision*, 138(6):615–620, 1991.

[23] Christopher M. Holt, Alan Stewart, Maurice Clint, and Ronald H. Perrott. An improved parallel thinning algorithm. *Commun. ACM*, 30(2):156–160, 1987.

[24] Zahid Hussain. *Digital Image Processing*. Ellis Horwood, 1991.

[25] Bernd Jaehne. *Digitale Bildverarbeitung*. Springer Verlag Berlin, 6 edition, 2005.

[26] Gisela Klette. Skeletons in digital image processing. University of Auckland, July 2002.

[27] Gisela Klette. *Topologic, Geometric, or Graph-Theoretic Properties of Skeletal Curves*. PhD thesis, Rijksuniversiteit Groningen, 2007.

[28] Paul Kwok. A thinning algorithm by contour generation. *Commun. ACM*, 31(11):1314–1324, 1988.

[29] L. Lam, S.-W. Lee, and C.Y. Suen. Thinning methodologies-a comprehensive survey. *IEEE Trans. Pattern Anal. Machine Intell.*, 14(9):869–885, 1992.

[30] C.L. Lee and P.S.P. Wang. A new thinning algorithm. In P.S.P. Wang, editor, *Proc. 12th IAPR International Conference on Pattern Recognition Vol. 1 - Conference A: Computer Vision & Image Processing*, volume 1, pages 546–548 vol.1, 1994.

[31] Tomas Lehner. Digitale Geländemodellierung mittels Delaunay-Triangulierung und Abbauplanung in AutoCAD. Master's thesis, Johannes Kepler Universität Linz, 2002.

[32] Wing-Nin Leung, C.M. Ng, and P.C. Yu. Contour following parallel thinning for simple binary images. In C.M. Ng, editor, *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 1650–1655 vol.3, 2000.

[33] Johann Benedict Listing. Der Census räumlicher Complexe oder Verallgemeinerung des Euler'schen Satzes von den Polyedern. *Abhandlungen der Mathematischen Classe der Königlichen Gesellschaft der Wissenschaften zu Göttingen*, pages 97–182, 1862.

[34] H. E. Lü and P. S. P. Wang. A comment on 'a fast parallel algorithm for thinning digital patterns'. *Commun. ACM*, 29(3):239–242, 1986.

[35] U. Montanari. A method for obtaining skeletons using a quasi-euclidean distance. *J. ACM*, 15(4):600–624, 1968.

[36] P. Morrison and Ju Jia Zou. An effective skeletonization method based on adaptive selection of contour points. In *Proc. Third International Conference on Information Technology and Applications ICITA 2005*, volume 1, pages 644–649 vol.1, 2005.

[37] Christian Neusius and Jan Olszewski. A noniterative thinning algorithm. *ACM Trans. Math. Softw.*, 20(1):5–20, 1994.

[38] Alfred Nischwitz and Peter Haberräcker. *Masterkurs Computergrafik und Bildverarbeitung*. Vieweg, 2004.

[39] John L. Pfaltz and Azriel Rosenfeld. Computer representation of planar regions by their skeletons. *Commun. ACM*, 10(2):119–122, 1967.

[40] Ioannis Pitas. *Digital Image Processing Algorithms*. Prentice Hall, 1993.

[41] R. Plamondon, M. Bourdeau, C. Chouinard, and C.Y. Suen. Validation of preprocessing algorithms: A methodology and its application to the design of a thinning algorithm for handwritten characters. In M. Bourdeau, editor, *Proc. Second International Conference on Document Analysis and Recognition*, pages 262–269, 1993.

[42] William K. Pratt. *Digital Image Processing.* John Wiley & Sons, 1991.

[43] Steffen Prohaska and Hans Christian Hege. Fast visualization of plane-like structures in voxel data. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 29–36, Washington, DC, USA, 2002. IEEE Computer Society.

[44] D. Reniers and A. Telea. Skeleton-based hierarchical shape segmentation. In *Proc. IEEE International Conference on Shape Modeling and Applications SMI '07*, pages 179–188, 2007.

[45] Azriel Rosenfeld. Connectivity in digital pictures. *J. ACM*, 17(1):146–160, 1970.

[46] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.

[47] Martin Rumpf and Alexandru Telea. A continuous skeletonization method based on level sets. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 151–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[48] John C. Russ. *The Image Processing Handbook , Fourth Edition.* CRC Press LLC, 2002.

[49] F.Y. Shih and Wai-Tak Wong. A new safe-point thinning algorithm based on the mid-crack code tracing. *IEEE Transactions on Systems, Man and Cybernetics*, 25(2):370–378, 1995.

[50] Kaleem Siddiqi, Sylvain Bouix, Allen Tannenbaum, and Steven W. Zucker. The hamilton-jacobi skeleton. *ICCV '99: Proceedings of the International Conference on Computer Vision*, 2:828, 1999.

[51] J.H. Sossa. An improved parallel algorithm for thinning digital patterns. *Pattern Recognition Letters*, 10:77–80, 1989.

[52] R. Stefanelli and A. Rosenfeld. Some parallel thinning algorithms for digital pictures. *J. ACM*, 18(2):255–264, 1971.

[53] King sun Fu and Azriel Rosenfeld. Pattern recognition and image processing. *IEEE Transactions on computers*, c-25(12):1336–1346, December 1976.

[54] Klaus Tönnies. *Grundlagen der Bildverarbeitung.* Pearson Studium, 2005.

[55] Ferdinand van der Heijden. *Image Based Measurement Systems.* John Wiley & Sons, 1994.

[56] P.S.P. Wang and Y.Y. Zhang. A fast and flexible thinning algorithm. *IEEE Trans. Comput.*, 38(5):741–745, 1989.

[57] Joachim Wipper. Mediale Achsen und Voronoj-Diagramme in der euklidischen Ebene. Master's thesis, Universität Stuttgart, 1997.

[58] T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, 1984.

[59] Y.Y. Zhang and P.S.P. Wang. A modified parallel thinning algorithm. In P.S.P. Wang, editor, *Proc. th International Conference on Pattern Recognition*, pages 1023–1025 vol.2, 1988.

[60] Y.Y. Zhang and P.S.P. Wang. Analysis of thinning algorithms. In P.S.P. Wang, editor, *Proc. th IAPR International Conference on Pattern Recognition Vol.III. Conference C: Image, Speech and Signal Analysis*, pages 763–766, 1992.

[61] Y.Y. Zhang and P.S.P. Wang. A parallel thinning algorithm with two-subiteration that generates one-pixel-wide skeletons. In P.S.P. Wang, editor, *Proc. 13th International Conference on Pattern Recognition*, volume 4, pages 457–461 vol.4, 1996.

[62] Ju Jia Zou. A fast skeletonization method. University of Western Sydney, December 2003.

[63] Ju Jia Zou and Hong Yan. Skeletonization of ribbon-like shapes based on regularity and singularity analyses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 31(3):401–407, 2001.

Erklärung

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

Leipzig, den 13. Oktober 2008

Christoph Bullmann