Georg-August-Universität Göttingen – Institut für Informatik

Seminar Computergrafik

Euklidische Abstandstransformation mit Anti-Aliasing

Nach einer Veröffentlichung von
Stefan Gustavson und Robin Strand (2010)

Georg Jahn
mail@gjahn.com

Leitung: Prof. Dr. Winfried Kurth
Dipl.-Inf. Reinhard Hemmerling
Datum: 28.02.2012

Inhaltsverzeichnis

Zusammenfassung

Der Referenzartikel[1] beschreibt eine Methode zur Verbesserung der Genauigkeit von euklidischen Abstandstransformationen durch Redefinition der Problemstellung: Als Ausgangspunkt dient nicht eine binär gerasterte Kontur, sondern eine flächengemittelte Abtastung. Aus dieser wird die Ursprungskontur geschätzt und so eine durchschnittliche Genauigkeit der Abstandstransformation im Bereich von etwa 0.02 Pixeln erzielt – dies entspricht etwa der Anwendung des herkömmlichen binären Verfahrens auf eine Rasterung der 16×16-fachen Auflösung.
Diese Arbeit erklärt die Grundlagen der Abstandstransformation, viel verwendete herkömmliche Algorithmen und die Erweiterung von vektorbasierten Verfahren zur Berücksichtigung der Kantenglättung. Schließlich werden die Ergebnisse des verbesserten Algorithmus und ihre Anwendungsmöglichkeiten diskutiert.

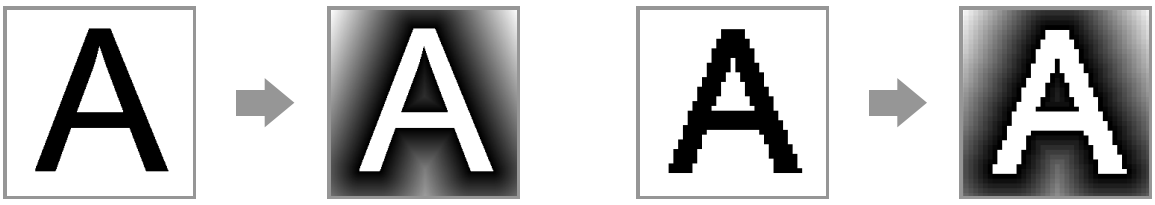
1 Einleitung

Das Ziel der Abstandstransformation ist es, jedem Bildpunkt den Abstand zum nächsten Element einer gegebenen Menge zuzuordnen. Obwohl diese Operation bereits 1966 von Rosenfeld und Pfaltz vorgeschlagen wurde[2] und vielfältige Anwendungen nicht nur im Bereich der Computergrafik hat, ist die Entwicklung verschiedener Algorithmen bis heute Forschungsthema. Da noch in den letzten Jahren vielerlei Verfahren vorgeschlagen wurden, ist die Frage der besten Implementation nach verschiedenen Kriterien noch nicht abschließend geklärt.[3]

In ihrer Veröffentlichung von 2010 verfolgen Gustavson und Strand[1] jedoch das Ziel einer qualitativen Verbesserung durch Erweiterung des herkömmlichen Konzepts der Abstandstransformation, indem Kantenglättung[1] einbezogen wird. Diese Idee kann in der Anwendung der Algorithmen deutliche Vorteile bringen, da sie Genauigkeiten im Subpixel-Bereich erlaubt. So kann u. a. in der Bildanalyse, der Grafikbearbeitung oder der hochdimensionalen Datenanalyse von der höheren Präzision profitiert werden. Doch zunächst einmal sollen die Grundlagen der Abstandstransformation erläutert werden.

1.1 Die Abstandstransformation

Die abstrakte Idee der Abstandstransformation besteht darin, jeder gegebenen Form, der sogenannten Hintergrundmenge $A \subset \mathbb{R}^2$, eine Abstandsfunktion zur Menge $f : \mathbb{R}^2 \backslash A \to \mathbb{R}$ zuzuordnen. So ergibt die Abstandstransformation für jeden Punkt der Vordergrundmenge $\mathbb{R}^2 \backslash A$ den Abstand zum nächsten Punkt der Hintergrundmenge. Abbildung 1 **(a)** demonstriert dies anhand einer Beispielform. Da Operationen in der Informatik jedoch stets nur auf endliche, diskretisierte Objekte wirken können, wird die Vordergrundmenge zunächst in einem ersten Schritt als binäres Bild gerastert, wie Abbildung 1 **(b)** zeigt. Als Resultat ergibt sich eine Abstandsmatrix.



(a) **(b)**

Abbildung 1: **(a)** Wirkung der Abstandstransformation auf die vorgegebene Form "A": Das Ergebnis der Transformation lässt sich wiederum als Bild interpretieren, hierbei stellen dunklere Grautöne niedrigere Abstände dar. Auf der Form selbst ist die Abstandsfunktion undefiniert (weiß eingefärbt). **(b)** Im diskretisierten Fall wirkt die Abstandstransformation auf eine binäre Rastergrafik und das Ergebnis ist eine wiederum als Bild interpretierte Abstandsmatrix.

[1] Im Folgenden wird zumeist der englische Begriff *Anti-Aliasing* verwendet, da er die Idee der Vermeidung von sogenannten *Alias*-Effekten besser beschreibt.

1.2 Verschiedene Metriken

Die Definition der Abstandstransformation lässt zunächst die Frage nach der zur Abstandsmessung verwendeten Metrik offen. (Oft wird sogar die Dimension der Räume nicht explizit festgelegt.) Für die meisten Anwendungen ist die euklidische Metrik am besten geeignet, da die zugrundeliegenden Probleme der euklidischen Geometrie oder zumindest einer gewissen Isotropie genügen. Die ersten "schnellen"[2] Algorithmen zur Berechnung der Abstandstransformation eigneten sich jedoch nur für andere Metriken; die euklidische Metrik schien komplizierter. Um dieser nahe zu kommen, treten in diesem Kontext oft zwei Metriken auf:

Manhattan-Metrik Sich an der Anzahl der Blocks, die man in der Planstadt zurücklegen müsste, orientierend, ist diese Metrik definiert durch

$$d_M(\vec{x}, \vec{y}) = \sum_i |x_i - y_i|.$$

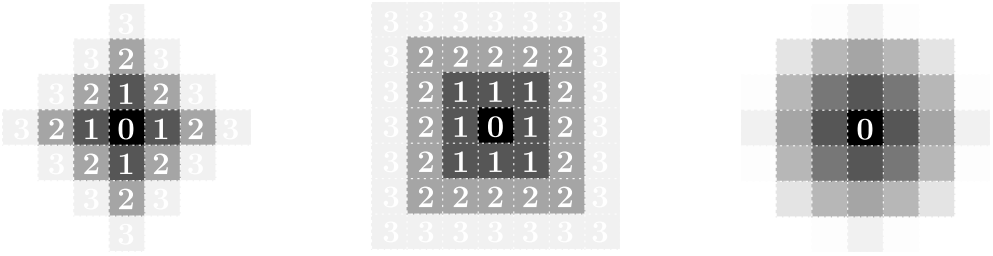
Chessboard-Metrik Dieses Abstandsmaß erhält seinen Namen dadurch, dass es auch der Anzahl der Züge des Königs beim Schach entspricht, die er minimal von $\vec{x}$ nach $\vec{y}$ braucht. Mathematisch spricht man auch von der ∞-Norm:

$$d_C(\vec{x}, \vec{y}) = \max_i |x_i - y_i|.$$

In Abbildung 2 werden die Metriken grafisch miteinander verglichen, man sieht schnell ein, dass die euklidische Metrik

$$d_E(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2}$$

von der Manhattan-Metrik unterschätzt und von der Chessboard-Metrik überschätzt wird. In manchen Problemen ist die Anwendung dieser Metriken auch explizit gewünscht.



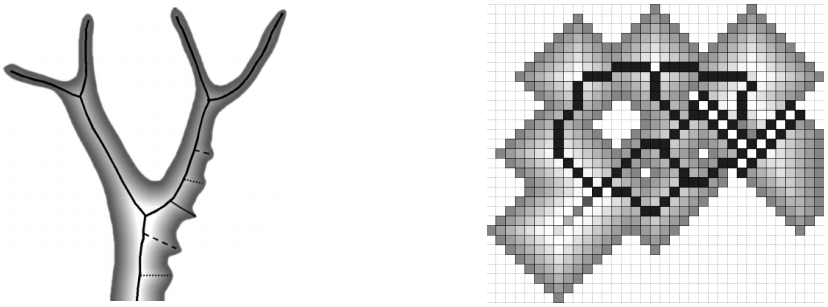
Manhattan-Metrik Chessboard-Metrik Euklidische Metrik

Abbildung 2: Schematische Darstellung der diskutierten Metriken: Eingetragen sind die Abstände vom schwarz markierten Mittelpunkt. Da die euklidische Metrik wenig ganzzahlige Ergebnisse liefert, wird der Abstand durch Graustufen visualisiert.

[2]Damit ist hier lineare Laufzeit in Abhängigkeit von der Pixelzahl gemeint, mehr dazu im nächsten Kapitel.

1.3 Anwendungsbeispiele

In der Computergrafik wird die Abstandstransformation häufig zur Ausdünnung (Erosion) oder Erweiterung (Dilatation) von Formen verwendet: Durch die Variation einer Schwelle für die Werte der Abstandstransformation können auf einfache Weise Objekte vergrößert oder auch verkleinert werden. Oft ist es jedoch das Ziel, das Betrachtete nicht nur zu erodieren, sondern in seine grundlegende Form zu skelettieren. Eine solche Skelettierung ist leicht mit dem Differential der Abstandsfunktion möglich, da diese auf dem Skelett ihr Maximum annimmt und das Differential somit verschwindet. Abbildung 3 **(a)** veranschaulicht dies anhand eines Beispiels aus der Biologie.



(a) **(b)**

Abbildung 3: Beispiele der Anwendung von Abstandstransformationen: **(a)** Skelettierung einer Zellform anhand ihres Differentials. **(b)** Skelettierung einer Karte zur Verbesserung der Effizienz von Wegfindungsalgorithmen.

Offensichtlich kann die Abstandstransformation auf diese Weise in der Bilderkennung und -analyse eingesetzt werden. Aber auch für ganz andere Probleme, zum Beispiel für die Wegfindung, ist es geeignet: Die Geschwindigkeit von Wegfindungsalgorithmen ist vor allem abhängig von der Größe des Zustandsraums, deshalb erweist es sich als sinnvoll, die verwendeten Karten zunächst zu einzelnen Pfaden auszudünnen, wie Abbildung 3 **(b)** zeigt. Wie im nächsten Kapitel diskutiert wird, kann eine solche Ausdünnung in linearer Laufzeit erfolgen, sodass sich der Aufwand zeitlich durchaus lohnt.

2 Algorithmen

Zur Berechnung der Abstandstransformation existiert eine Vielzahl an Algorithmen, an denen bis heute geforscht wird. Deshalb erhebt diese kurze Einführung in die Funktionsweise einiger Algorithmen keinerlei Anspruch auf Vollständigkeit, eine ausführlichere Beschreibung aktuell relevanter Algorithmen findet sich in einer Vergleichsstudie von Fabbri, Costa, Torelli und Bruno[3].

2.1 Sequentieller Algorithmus mit Wertpropagation

Rosenfeld und Pfalz veröffentlichten 1966 einen ersten Algorithmus zur Berechnung der Abstandstransformation in der Laufzeit $\mathcal{O}(n)$, wobei n die Pixelzahl bezeichne.[2] Da

keine geringere Laufzeitkomplexität möglich ist (jeder Pixel muss natürlich zumindest einmal betrachtet werden), nennen wir solche Algorithmen "schnell". Die Idee zu dem Verfahren kommt aus der Beobachtung, dass bei der Manhattan-Metrik der Wert eines Pixels der Abstandstransformierten nur von den vier umgebenden Pixeln abhängt. Daraus kann man sich überlegen, dass es somit möglich ist, die Abstandswerte in die vier Richtungen über das Bild hinweg zu propagieren und dabei stets das Minimum zu betrachten. Hierbei fällt wiederum auf, dass eine sinnvolle Propagation lediglich in zwei Richtungen gleichzeitig, d. h. in Richtung eines der vier Quadranten erfolgen kann; eine Propagation in die Rückrichtungen führt immer zu höheren Werten.

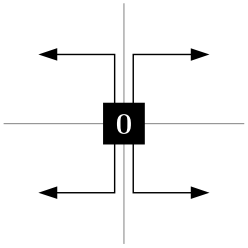


Abbildung 4: Scanrichtungen bei der sequentiellen Wertpropagation

Deshalb verfährt der sequentielle Algorithmus wie folgt: Zunächst wird die Abstandsmatrix überall mit dem höchstmöglichen Abstandswert initialisiert, nur an den Stellen der Hintergrundmenge wird der Wert auf 0 gesetzt. Daraufhin werden in einem ersten Bildscan zunächst alle Werte nach unten propagiert, d. h. der Wert eines Matrixeintrags ergibt sich aus dem Minimum des Eintrags selbst und des Matrixelements direkt darüber plus eins. Danach werden die nach unten propagierten Werte nach dem gleichen Verfahren nach rechts und nach links propagiert, sodass die ersten beiden Quadranten bereits erfüllt wurden. Im nächsten Scanlauf werden die Einträge nach oben und dann wieder nach rechts und links propagiert, sodass alle vier Quadranten betrachtet wurden das Minimum von allen vorliegt. Das Ergebnis ist die Abstandstransformation nach der Manhattan-Metrik.

Dieses Verfahren lässt sich leicht auf die Chessboard-Metrik anpassen, da sich diese Metrik stets aus den umliegenden acht Feldern ergibt. Somit muss auch in Richtung der Diagonalen propagiert werden, aber die Grundidee bleibt gleich. Offensichtlich kommt der sequentielle Algorithmus mit einer konstanten Anzahl von Bilddurchläufen aus und hat somit die gewünschte Laufzeitkomplexität $\mathcal{O}(n)$, auf die euklidische Metrik lässt er sich aber nicht direkt erweitern, da das Verfahren der Wertpropagation hier an seine Grenzen stößt.

2.2 Fast-Marching

Eine elegante Idee zur schnellen Lösung der euklidischen Abstandstransformation ist die Verwendung der Eikonalgleichung

$$(\nabla\phi)^2 = \frac{1}{c^2},$$

welche die Ankunftszeit eines Lichtstrahls beschreibt, der sich isotrop mit Geschwindigkeit c im Raum ausbreitet. Die Lösung der Differentialgleichung unter den entsprechenden An-

fangsbedingungen und mit $c = 1$ ergibt natürlich auch direkt die gewünschte euklidische Abstandsfunktion von der Startverteilung. Um die gewünschte Laufzeit zu erreichen, entwickelte Sethian 1996 einen sogenannten Fast-Marching-Algorithmus, der stets nur die Fronten der Wellenausbreitung betrachtet.[4] Dazu werden die Pixel zunächst alle als *weit weg* klassifiziert und nur Pixel um die Hintergrundmenge herum als *Grenze* eingestuft. Ist durch Lösung der Differentialgleichung der Wert eines Pixels bestimmt, so wird er als *tot* klassifiziert und die umliegenden Pixel, die *weit weg* sind, als *Grenze* eingeordnet. So läuft der Algorithmus, bis die Menge der *toten* Pixel alle Pixel enthält und die Differentialgleichung somit gelöst ist. Da jeder Pixel nur konstant oft betrachtet werden muss, erreicht der Algorithmus ebenfalls die gewünschte lineare Laufzeit.

2.3 Vektorpropagation

Einen Algorithmus, der lineare Laufzeit bei einer Vielzahl von verschiedenen Metriken erreicht, konnte Danielsson 1980 veröffentlichen:[5] Bei der sogenannten Vektorpropagation wird in der Abstandsmatrix zunächst nicht nur der Abstand zum nächsten Hintergrundelement gespeichert, sondern der gesamte Vektor zu diesem, Abbildung 5 **(a)** veranschaulicht dies. Damit lässt sich der zuvor vorgestellte sequentielle Algorithmus naiv auf andere Metriken übertragen: Anstatt der Zahlen werden die kompletten Vektoren propagiert und anstatt der Minimumsbildung derjenige Vektor bevorzugt, für den die Metrik den kürzesten Abstand ergibt. Unter der Annahme, dass der einem Pixel zugeordnete Vektor nur von seinen vier nächsten Nachbarn abhängt, kann auch die euklidische Abstandstransformation erzeugt werden (siehe Abbildung 5 **(b)**), man nennt diesen Algorithmus deshalb SED4.

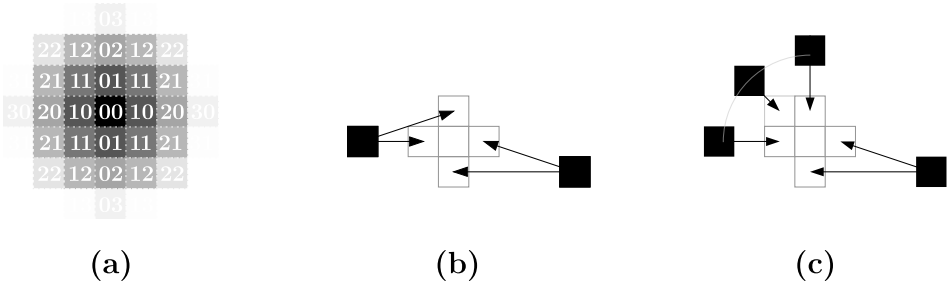


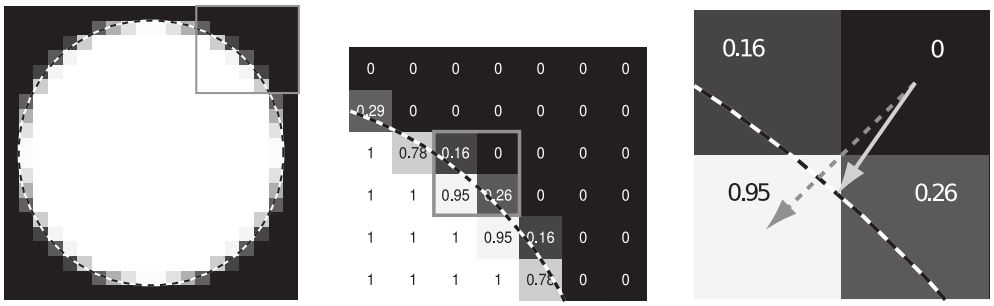
Abbildung 5: **(a)** Schema der Vektorpropagation, ein trennendes Komma zwischen den beiden Ziffern und ihre Vorzeichen wurden weggelassen. **(b)** Bei SED4 werden die vier umliegenden Pixel betrachtet, um das nächste Element der Hintergrundmenge zu finden. **(c)** Nicht immer reicht die Vierer-Nachbarschaft von SED4 aus, um wirklich das nächste Element zu bestimmen.

Dies ist jedoch nicht immer der Fall, wie in Abbildung 5 **(c)** zu sehen ist; in speziellen Fällen kann der falsche "nächste" Pixel ausgewählt werden. Maximal, so stellt Danielsson fest, tritt ein Abstandsfehler von 0.29 Pixeln auf. Offensichtlich würde für den dort dargestellten Fall die Betrachtung der Achter-Nachbarschaft genügen, der entsprechende Algorithmus wird als SED8 bezeichnet. Doch auch in diesem Fall können Fehler auftreten, maximal etwa 0.027 Pixel, was für die meisten Anwendungen aber sehr gut ausreicht.

Zusammenfassend ist die Vektorpropagation zwar leicht zu implementieren und zusätzlich steht nicht nur der Abstand zum nächsten Hintergrundelement, sondern sogar der Vektor zu diesem zur Verfügung, andererseits ist das Resultat in seltenen Fällen nicht exakt. In manchen Anwendungsgebieten ist das fatal, da schon eine kleine numerische Ungenauigkeit zur Instabilität von Algorithmen führen kann. Jedoch wird ein weiterer großer Vorteil die Methodik des nächsten Kapitels erst erlauben: Es sind quasi beliebige Metriken zulässig und damit auch solche, die Anti-Aliasing mit einbeziehen.

3 Anti-Aliasing

Bisher war der Ausgangspunkt der Abstandstransformation immer eine binäre Rasterung. Daten in der Praxis, so bemerken Gustavson und Strand, jedoch sind zumeist nicht binär; da während des Rasterungsvorgangs über die Pixelfläche gemittelt wird (Abbildung 6).[1] Die Idee des Referenzartikels besteht nun darin, diese zusätzliche Information für eine höhere Genauigkeit auszunutzen. Dazu wird angenommen, dass die vorliegende Grafik aus der Rasterung einer Form mit im Mittel geringer Kantenkrümmung gegenüber der Pixelgröße hervorgeht. Die Problemstellung wird nun so redefiniert, dass nicht mehr der Abstand zur Pixelmitte, sondern der Abstand zur rekonstruierten Kante berechnet werden soll, dies ist in Abbildung 6 **(c)** dargestellt.



(a) **(b)** **(c)**

Abbildung 6: **(a)**, **(b)** Üblicherweise findet bei der Rasterung ein Sampling, d. h. eine Flächenmittelung statt, das Ergebnis ist also zunächst keine binäre Rastergrafik. **(c)** Präzision der Problemstellung: Gesucht ist nun der Abstand zur rekonstruierten Kante anstatt zur Pixelmitte.

3.1 Motivation

Zunächst stellt sich die Frage, warum eine Genauigkeit im Subpixel-Bereich überhaupt erstrebenswert sein könnte. Als erstes ist dabei anzumerken, dass in den meisten Fällen eine Graustufeninformation sowieso vorliegt und nicht ungenutzt bleiben sollte. Hinzu kommt, dass sich Formen aus der realen Welt im Allgemeinen schlecht auf einem Gitter approximieren lassen und somit sogenannte *Alias*-Effekte auftreten, d. h. Bildstörungen durch den binären Rasterungsvorgang von höherfrequenten Daten. Diese Störungen schlagen sich im gesamten Frequenzbereich nieder und treten deutlicher zutage, wenn man die

Abstandstransformation betrachtet. Abbildung 7 zeigt die Auswirkungen des fehlenden Anti-Aliasings. Zunächst einmal hat die Abstandstransformation deutliche Fehler, deren hochfrequenter Anteil durch die Anwendung des Differentials noch deutlich verstärkt wird. Da das Differential der Abstandstransformation sehr häufig Verwendung findet, ist eine höhere Genauigkeit ebenso häufig von großem Vorteil.

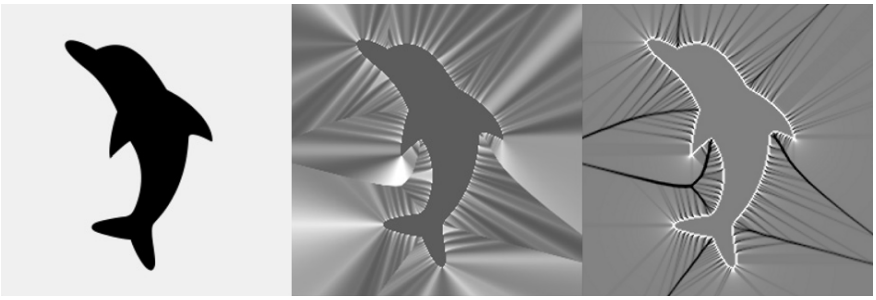


Abbildung 7: Links: Vorlage. Mitte: Fehler der euklidischen Abstandstransformation gegenüber dem Abstand zur perfekten Form, mittleres Grau steht für keinen Fehler. Rechts: Differential der euklidischen Abstandstransformation.

3.2 Mathematische Konstruktion

Dieser Abschnitt widmet sich der mathematischen Herleitung der Rekonstruktion der Kante aus dem Pixelfüllungsanteil $a \in [0, 1]$. Grundannahme dabei ist die Annahme einer geringen Kantenkrümmung im Vergleich zur Pixelgröße, im Speziellen sogar die Annahme einer lokal geradlinigen Kante. Sei zunächst die Richtung der Kante φ vorgegeben, $\mathbf{g} = (g_x, g_y)$ der Normalenvektor der Kante. Dann ergeben sich die verschiedenen Bereiche a_1 und a_2 in denen die Kantenposition d_f unterschiedlich mit a zusammenhängen. Mit den Bezeichnungen wie in Abbildung 8 ergibt sich folgende Gleichungen für die Unbekannten:

$$
\begin{aligned}
a_1 &= \tan\varphi \\
a_2 &= 1 - 2a_1 \\
d_1 &= \sin\varphi = g_y \\
d_2 &= \tfrac{1}{\sqrt{2}} \sin(\pi/4 - \varphi)
\end{aligned}
$$

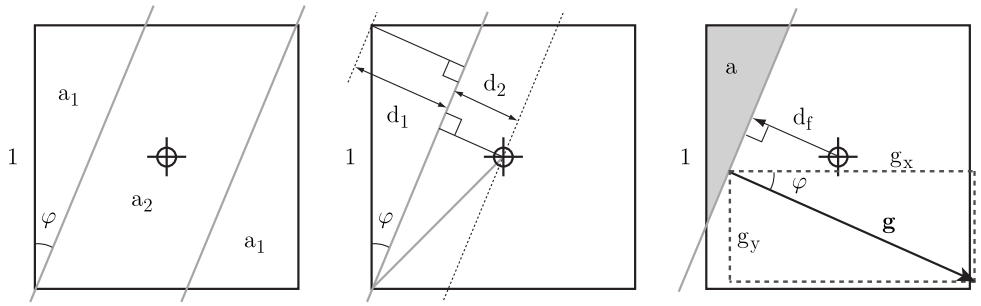


Abbildung 8: Konstruktionen zur Pixelfüllung a in Abhängigkeit von φ und d_f.

In den einzelnen Bereichen erhält man so schnell den Zusammenhang zwischen der Füllung a und d_f zu:

$$a = \begin{cases} 0, & \text{falls } d_f \leq -d_1 - d_2 \\ (d_1 + d_2 + d_f)^2 \cdot (a_1/d_1^2), & \text{falls } -d_1 - d_2 \leq d_f \leq -d_2 \\ a_1 + (a_2/2) \cdot (1 + d_f/d_2), & \text{falls } -d_2 \leq d_f \leq d_2 \\ 1 - (d_1 + d_2 - d_f)^2 \cdot (a_1/d_1^2), & \text{falls } d_2 \leq d_f \leq d_1 + d_2 \\ 1, & \text{falls } d_f \geq d_1 + d_2 \end{cases}$$

Gewünscht ist jedoch der Verlauf von d_f in Abhängigkeit von a und φ, man bildet also die Umkehrfunktion, und erhält sodann

$$d_f = \begin{cases} (g_x + g_y)/2 + \sqrt{2 g_x g_y a}, & \text{falls } 0 \leq a \leq a_1 \\ (1/2 - a) \cdot g_x, & \text{falls } a_1 \leq a_1 \leq 1 - a_1 \\ -(g_x + g_y)/2 + \sqrt{2 g_x g_y (1 - a)}, & \text{falls } 1 - a_1 \leq a \leq 1. \end{cases}$$

In Abbildung 9 ist d_f über a bei drei verschiedenen Werten von φ aufgetragen. Im Falle $\varphi = 0$ erhalten wir den offensichtlichen linearen Zusammenhang $d_f = 1/2 - a$, während bei anderen Werten von φ der lineare Bereich kleiner ist. Es ist auch gut zu erkennen, dass es nicht schlimm ist, wenn φ nicht besonders genau bekannt ist; im Allgemeinen ist d_f auch relativ gut durch den Zusammenhang bei $\varphi = 0$ approximierbar, falls φ gänzlich unbekannt sein sollte.

Als Verschiebung der Kantenposition zur Pixelmitte erhält man schließlich

$$\vec{d_f} = \begin{pmatrix} d_f \cos\varphi \\ d_f \sin\varphi \end{pmatrix} = d_f \cdot \begin{pmatrix} g_x \\ g_y \end{pmatrix}.$$

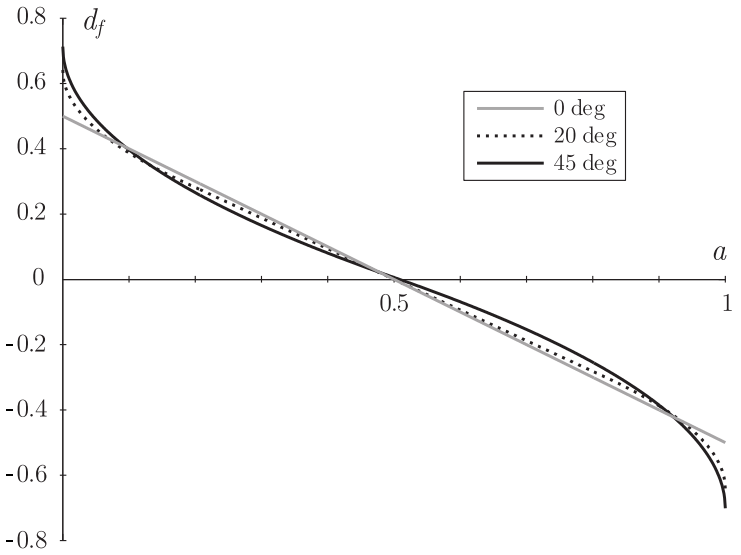


Abbildung 9: Verlauf von Kantenabstand d_f in Abhängig von Pixelfüllung a und Kantenwinkel φ.

3.3 Das Verfahren mit Anti-Aliasing

Das errechnete $\vec{d_f}$ wird nun als Korrektur innerhalb der euklidischen Metrik bei einem beliebigen Vektorpropagations-Algorithmus verwendet nach

$$\vec{d} = \vec{d_i} + \vec{d_f},$$

wobei $\vec{d_i}$ den ursprünglichen Vektor zur Pixelmitte bezeichne. Im konkreten Fall wurde der beschriebene SED8 Algorithmus verwendet. Gustavson und Strand beschreiben nun drei Möglichkeiten mit unterschiedlichem Rechenaufwand, in denen der Algorithmus implementiert werden kann. Dazu werden bestimmte Näherungen gemacht:

Kantenwinkel $\varphi = 0$: In diesem Fall wird der Winkel φ nicht bestimmt und der gefundene einfache lineare Zusammenhang für d_f benutzt.

Kantenwinkel φ senkrecht zu $\vec{d_i}$: Hier wird angenommen, dass die Kante stets senkrecht auf der Verbindungslinie zwischen betrachtetem Punkt und Kante steht. Mathematisch zutreffend ist jedoch, dass die Kante senkrecht auf $\vec{d}$ steht, sodass die Annahme nur mit größeren $\vec{d_f}$ asymptotisch korrekt ist. Dass die Kante überhaupt senkrecht auf der Verbindungslinie steht, folgt durch eine leichte Überlegung: Stände sie nicht senkrecht, so gäbe es einen Punkt auf der Kante direkt neben dem ursprünglichen, der dem betrachteten Punkt näher wäre.

Kantenwinkel φ voll berechnet: Da im Falle kleiner $\vec{d_f}$ die vorige Näherung ungültig wird, wird in diesem Fall zusätzlich durch zwei 3×3-Gradienten-Kernel die Kantenausrichtung abgeschätzt. Dadurch erhält man eine gute Schätzung für φ in der Nähe der Kante, mit steigendem $\vec{d_f}$ wird jedoch die Näherung über die Orthogonalität schnell deutlich genauer, sodass die Gradienten-Kernel nur bei direkt angrenzenden Pixeln verwendet werden.

Der Rechenaufwand wird zusätzlich in den ersten beiden Fällen dadurch verringert, dass $\vec{d_i}$ und $\vec{d_f}$ als parallel angenommen werden und sich die obige Vektorgleichung zur einer skalaren vereinfacht:

$$d = d_i + d_f$$

3.4 Ergebnisse

Zur Auswertung wurde der Algorithmus auf eine große Menge von Testbildern angewendet, um das Verhalten unter verschiedenen Bedingungen zu messen. Anhand einer repräsentativen Form mit großen und kleinen Krümmungen – der bereits gezeigte Delfin – sollen nun die Ergebnisse diskutiert werden.

Insgesamt erreicht das endgültige Ergebnis mit vollständiger Berechnung von φ einen durchschnittlichen absoluten Fehler von 0.02 Pixel, 99% der Pixel erreichen eine Genauigkeit innerhalb von ± 0.2 Pixel. Nur sehr wenige Pixel zeigen Fehler von bis zur 0.5 Pixel; diese treten dann auf, wenn die Grundannahme der kleinen Krümmung der Kontur nicht erfüllt ist. Im Beispielbild ist das bei der Spitze der linken Flosse der Fall, diese stellt auch die Hauptfehlerquelle in diesem Bild dar.

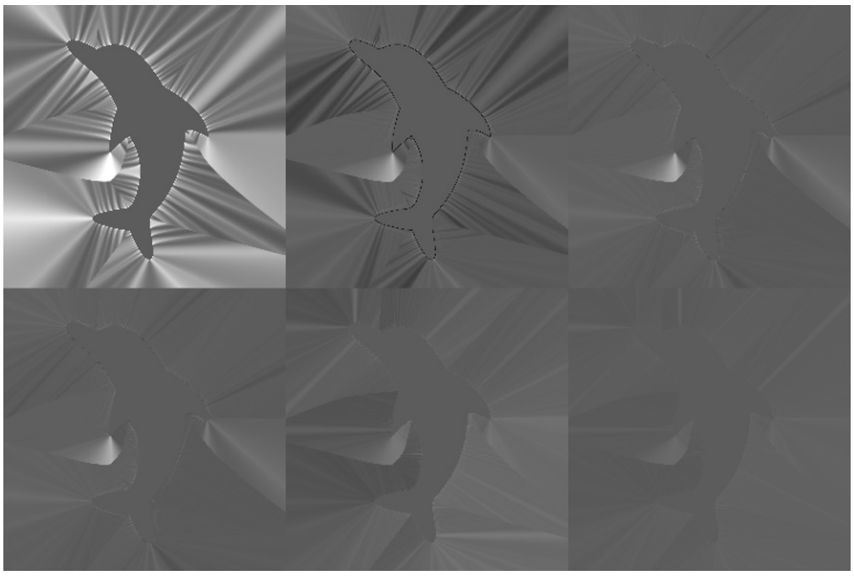


Abbildung 10: Vergleich der Fehlerbilder der Abstandstransformation mit verschiedenen Verfahren: In der oberen Zeile das herkömmliche Verfahren ohne Anti-Aliasing mit normaler Auflösung, 8×8-facher Auflösung und 16×16-facher Auflösung. In der unteren Zeile das Verfahren mit Anti-Aliasing stets in normaler Auflösung bei $\varphi = 0$, φ senkrecht und φ voll berechnet.

Abbildung 10 vergleicht die Ergebnisse des Verfahrens ohne Anti-Aliasing mit den drei verschiedenen Möglichkeiten der Implementation, die Kantenglättung berücksichtigen. Bereits das einfachste Verfahren mit der Annahme $\varphi = 0$ führt zu stark verringerten Fehlern, aber mit steigendem Rechenaufwand erhöht sich die Genauigkeit noch. Der volle Algorithmus zeigt in diesem Beispiel Fehler, die in der Größenordnung des herkömmlichen Verfahrens bei 16-facher Auflösung in beide Richtungen liegen.

Weiterhin ist durch die höhere Genauigkeit auch das Differential deutlich glatter und erlaubt eine einfachere Extraktion der Merkmale (siehe Abbildung 11). Dazu sei angemerkt, dass bereits bei der binären Abstandstransformation der Betrag des Gradienten nah bei 1 liegt, aber die Richtung willkürlich schwankt. Dieses Verhalten zeigt die verbesserte Methode nicht.

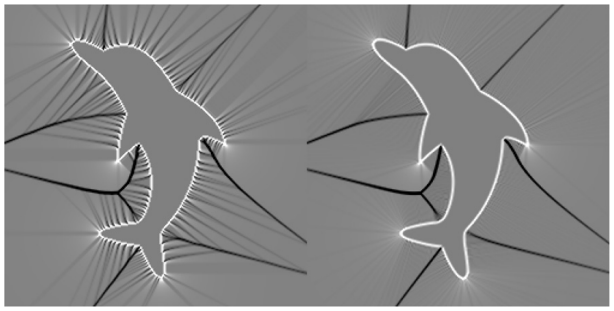


Abbildung 11: Vergleich des Differentials der herkömmlichen Methode (links) und des Verfahrens mit Anti-Aliasing (rechts): Die hochfrequenten Störungen sind viel schwächer.

Herkömmlich	$\varphi = 0$	φ senkrecht	φ voll berechnet
0.33 s	0.83 s	3.2 s	3.9 s
1.0×	2.5×	9.7×	11.8×

Tabelle 1: Vergleich der Laufzeiten der verschiedenen Implementationen

Schließlich bleibt noch die Laufzeit der Verfahren zu betrachten, Tabelle 1 zeigt die gemessenen Zeiten bei einer noch nicht geschwindigkeitsoptimierten Implementation. Die zusätzliche Genauigkeit kommt zu einem angemessenen Preis, wobei die relative Verlangsamung noch deutlich verbessert werden könnte: Da die Funktion $d_f(a, \varphi)$ beschränkt ist, kann durch sogenannte *early-out*-Tests ihre Berechnung oft vermieden werden. Weiterhin sind ihre Argumente ebenfalls beschränkt, sodass sie sich gut vorberechnen und tabellieren lässt.

3.5 Ausblick

Das beschriebene Verfahren erreicht sehr gute Genauigkeiten, die sich durchaus für Problemstellungen im Subpixel-Bereich, wie die zu Beginn genannten Dilatationen und Erosionen in der Grafikbearbeitung, eignen. Weiterhin beruhen die meisten Anwendungen der Abstandstransformation auf dem Differential des Abstandsfeldes, das ebenfalls signifikant verbessert werden konnte.

Zwar könnte die höhere Genauigkeit auch mit den herkömmlichen Algorithmen und höheren Auflösungen erreicht werden, aber dafür würde etwa die $16 \cdot 16 = 256$-fache Laufzeit und Speicherbelegung verwendet. Dies kann das vorgestellte Verfahren leicht unterbieten und zumeist liegen in der Praxis höhere Auflösungen nicht vor.

Gustavson und Strand merken zudem an, dass sie das Verfahren eigentlich entwickelt haben, um aus abgetasteten Rastergrafiken scharfe, hochauflösende Kanten zu rekonstruieren. Dass sich das Verfahren dafür hervorragend eignet, kann in einer Demonstration ausprobiert werden, die auf der Seite `http://www.itn.liu.se/~stegu/edtaa/` zum Herunterladen bereit steht. Dort findet sich ebenso der Quellcode der verwendeten Algorithmen.

A Quellen

Abbildung 3 (a): Anwendungsbeispiel der Skelettierung von `http://radiographics.rsna.org/content/24/1/287/F5.expansion.html` (01.02.2012).

Abbildung 3 (b): Beispiel der Form-Ausdünnung aus der Galerie von `http://liris.cnrs.fr/dgtal/gallery/` (01.02.2012).

Abbildung 6-11: Grafiken mit geringfügigen Modifikationen übernommen aus [1].

B Referenzen

Literatur

[1] S. Gustavson, R. Strand, *Anti-Aliased Euclidean Distance Transform*, Pattern Recognition Letters **32**, 252–257 (2011)

[2] A. Rosenfeld, J. L. Pfaltz, *Sequential Operations in Digital Picture Processing*, J. ACM **13**, 471–494 (1966), URL `http://doi.acm.org/10.1145/321356.321357`

[3] R. Fabbri, L. Da F. Costa, J. Torelli, O. Bruno, *2D Euclidean distance transform algorithms: A comparative survey*, COMPUTING SURVEYS **40(1)** (2008)

[4] J. Sethian, *A fast marching level set method for monotonically advancing fronts*, Proceedings of the National Academy of Sciences **93(4)**, 1591–1595 (1996)

[5] P. Danielsson, *Euclidean distance mapping*, Computer Graphics and image processing **14(3)**, 227–248 (1980)