Kurzfassung

Im Rahmen dieser Diplomarbeit wurde ein neuer Algorithmus zur Erkennung von LATEX-Bildern, das sind Linienzeichnungen, entwickelt und implementiert. Die Grundidee des hier vorgestellten Lösungsanatzes besteht in der Verwendung einer regelmäßigen 2*2/4 Pyramide, um die riesige Menge einzelner Kantenpunkte im Bild zu möglichst langen Geraden und Kreisbögen zusammenzufassen. Dadurch wird eine erste Reduktion der Datenmenge erreicht, und auch der erste Schritt hin zur Erkennung des Bildes gemacht. Die Approximation der Kanten mit Geraden und Kreisen erfolgt mittels statistischer Methoden.

Außer einer detaillierten Beschreibung dieses Algorithmus enthält die Diplomarbeit einen Überblick über den Stand der Technik in der Erkennung von Linienzeichnungen und einen Bericht über experimentelle Ergebnisse.

Abstract

Within the scope of this diploma thesis a new algorithm for the recognition of LATEX-pictures, e. g. line drawings, was developed and implemented. The basic idea of this approach is the use of a regular 2*2/4 pyramid for building up lines and arcs out of the huge mass of single edge points. In this manner the amount of data is reduced and the first step towards recognition of the image is done. The approximation of the edges with lines and arcs is accomplished by statistical methods.

In addition to a detailed description of this algorithm, this diploma thesis gives a survey over the state-of-the-art in recognition of line drawings and a report on some experimental results.

1. Problemstellung

1.1 Einleitung

In den letzten Jahren wurde dem vollautomatischen Erkennen technischer Zeichnungen großes Interesse entgegengebracht, da in der Industrie dringender Bedarf danach besteht. In den Archiven vieler Firmen lagern riesige Mengen technischer Zeichnungen, die größtenteils nur auf Papier verfügbar sind. Allein in den USA gibt es zwei Milliarden technische Zeichnungen, die von aktuellem Wert sind [Naga90]. Diese Zahl kann eigentlich gar nicht mehr überraschen, wenn man bedenkt, daß große Projekte aus 30.000 Zeichnungen bestehen, die dann 10-40 Jahre lang aktuell sind [Kari85].

Natürlich hätte es sehr viele Vorteile, wären diese Zeichnungen über CAD-Systeme verfügbar. So wäre die Archivierung einfacher, Zeichnungen wären schneller und von mehreren Personen gleichzeitig einsehbar und könnten auch für neue Konstruktionen wiederverwendet werden. Viele Firmen würden deshalb am liebsten ihr ganzes Archiv in eine CAD Datenbank eingeben. Der Arbeitsaufwand dafür ist aber viel zu groß. Sowohl die Nachkonstruktion mit einem CAD System als auch das Digitalisieren mit einem Tablett verursachen stundenlange Arbeit. Auch die Verwendung bestehender Konversions-programme hat sich wegen der anschließend notwendigen Nachbearbeitung und der Fehleranfälligkeit als nicht produktiv erwiesen [Krau89].

Ganz im Gegensatz zu dem Wunsch, alte Zeichnungen in CAD Systeme zu übertragen, steht die Praxis, daß auch heute noch Neukonstruktionen manchmal mit Papier und Bleistift durchgeführt werden [Zhan92]. Gerade die erfahrensten Konstrukteure zeichnen ihre Entwürfe lieber auf Papier. Meistens müssen solche Zeichnungen von gut ausgebildetem Personal auf einem CAD / CAM System nachkonstruiert werden.

Die Diplomarbeit ist folgendermaßen aufgebaut: Zunächst wird ein Überblick über den State-of-the-Art beim Erkennen technischer Zeichnungen gegeben. Danach folgt eine genaue Beschreibung der Problemstellung, die hier behandelt wird. Kapitel 2 beschreibt einen neuen Lösungsansatz für das Erkennen von Linienzeichnungen, der im Rahmen dieser Diplom-arbeit entwickelt und auf LATEX-Bilder angewandt wurde. In Kapitel 3 wird von einigen experimentellen Ergebnissen berichtet und schließlich die Verwendbarkeit des hier vorgestellten Algorithmus für die allgemeinere Aufgabenstellung des Erkennens technischer Zeichnungen behandelt.

1.2 State-of-the-Art

In diesem Kapitel soll ein kurzer Überblick über die Funktionsweise bestehender Konversionsprogramme gegeben werden. Die Literatur zu diesem Thema ist leider nicht sehr abwechslungsreich. Meistens wird ein Algorithmus präsentiert, der im wesentlichen aus folgenden Schritten besteht:

- Binarisierung: Falls die Zeichnung nicht ohnehin schon als Binärbild eingescannt wird, ist der erste Schritt die Binarisierung des Bildes. Der Schwellwert wird meistens dynamisch aus der Umgebung des Pixels bestimmt, manchmal aber auch global festgelegt.

- Text und Zeichnung trennen: Falls die Zeichnung auch Text enthalten kann, wird dieser von der Zeichnung getrennt. Um zwischen Text und Zeichnung unterscheiden zu können, wird um jeden zusammenhängenden Bereich von Vordergrundpixeln das kleinste umschreibende Rechteck gelegt. Anhand von Form, Größe und Anzahl der gesetzten Pixel in diesem Rechteck ist der Text recht gut von der Zeichnung trennbar. In [Kast90] ist ein solcher Algorithmus sehr gut beschrieben. Dort ist auch das Problem von Buchstaben, die die Zeichnung berühren, gelöst.

- Preprocessing: Bei der Binarisierung können kleine Störungen entstehen. Einzelne Pixel des Hintergrundes sind gesetzt, während einzelne Vordergrundpixel nicht gesetzt sind. Dieses Problem wird meistens durch Anwendung der morphologischen Operationen Open und Close mit einem 3*3 Formelement gelöst. Kompliziertere, aber sicherlich auch bessere Verfahren, mit denen Störungen ausgebessert werden können, sind zwar aus der Literatur bekannt (z.B.: [Davi78]), werden aber sehr selten eingesetzt, z.B.: [Krau89].

- Ausgefüllte Objekte: Technische Zeichnungen bestehen hauptsächlich aus Linien, enthalten aber recht häufig auch ausgefüllte Objekte, z.B.: Pfeilspitzen. Sie müssen von den Linien getrennt werden. Das geschieht wieder mit Hilfe morphologischer Operationen: Das Bild wird mit einem etwas größeren Formelement erodiert. Alle Objekte, die dann immer noch sichtbar sind, werden als ausgefüllte Objekte angesehen und speziell behandelt. Versuche, ohne eine derartige Trennung auszukommen, sind fehlgeschlagen [Kast90].

- Skelettierung: Die im Bild verbleibenden Linien werden nun skelettiert, d.h.: es werden so lange Pixel gelöscht, bis die Linie nur noch einen Pixel breit ist. Dabei kommen verschiedene, aus der Literatur bekannte, Methoden zur Anwendung, z.B.: [Harr82], [Zhan84].

- Vektorisierung: Es werden zunächst die Knotenpunkte, also jene Punkte, an denen mehrere Linien zusammenstoßen, gesucht. Die Linien zwischen Knotenpunkten bzw. Endpunkten werden mit kurzen Geradenstücken approximiert.

- Erkennen von Geraden und Kreisen: Es wird versucht, die Linien mit Geraden, Kreisen und eventuell auch Kegelschnitten zu beschreiben. Dazu müssen die Linien auch zwischen den Knotenpunkten noch weiter unterteilt werden, bis jeder Abschnitt mit einem Element beschrieben werden kann. Als Unterteilungspunkte werden meistens jene Punkte gewählt, an denen sich die Steigung oder die Krümmung der Linie ändert. Bei der Approximation des Linienverlaufs wird darauf geachtet, daß der maximale Abstand zwischen der Beschreibung und der Linie im Bild unter einer Toleranzgrenze bleibt. Die dabei verwendeten Methoden sind nicht immer genau beschrieben.
  Sehr gut erklärt ist dieser Schritt aber in [Naga90]. Dort werden die Linienstücke durch jenen Kegelschnitt beschrieben, der die Linie im Sinne des kleinsten quadratischen Fehlers am besten approximiert. Bei der anschließenden Bestimmung der maximalen Abweichung, wird allerdings die Distanz zwischen der Approximationskurve und jedem einzelnen Linienpunkt bestimmt, was doch recht rechenintensiv sein dürfte.

Der hier beschriebene Ansatz ist z.B.: in [Musa88], [Krau89], [Kast90], [Naga90], [Suzu90], [Zhan92], [Krau93], u.v.a. zu finden. Zu [Musa88] und [Suzu90] muß allerdings ergänzend bemerkt werden, daß in diesen Arbeiten das Erkennen und das komprimierte Abspeichern von Landkarten angestrebt wird. Deshalb entfällt dort die Kreiserkennung, und es ist auch nicht wichtig, Geraden in ihrer vollen Länge zu erkennen. Die übrigen Arbeiten verfolgen das Ziel, technische Zeichnungen in CAD Format zu beschreiben.

Auch wenn dieser Algorithmus so oft implementiert wurde, sind die Ergebnisse in bezug auf Genauigkeit und Fehleranfälligkeit nicht zufriedenstellend. Gleich der erste Schritt, die Binarisierung, ist von zweifelhaftem Wert. Bereits 1987 bezeichnet Hofer-Alfeis in [Hofe87] diesen Schritt als Informationsverlust. Auch in dem wesentlich neueren Artikel [Krau93] wird die Verwendung von Grauwertbildern für zukünftige Systeme empfohlen, um die Genauigkeit der Ergebnisse zu verbessern. Das ist aber nicht so leicht möglich. Der ganze Algorithmus (Morphologie, Skelettierung und Vektorisierung) basiert auf der Verwendung von Binärbildern. Sollten Grauwertbilder verarbeitet werden, müßte der Algorithmus grundlegend geändert werden.

Ein anderer Grund, der für die Verwendung von Binärbildern angegeben wird, ist die Größe der Bilder. Obwohl heute die Computer mit viel Speicherplatz ausgerüstet sind, ist das sicherlich ein wichtiges Argument. Es verliert aber an Überzeugungskraft, wenn dann doch Grauwertbilder eingescannt werden, um intelligentere Binarisierungsmethoden verwenden zu können.

Ziemlich bedeutungslos wäre dieses Argument, wenn das Bild gar nicht in den Hauptspeicher geladen werden müßte, sondern in einem Durchlauf Zeile für Zeile abgearbeitet werden könnte. Beim angeführten Lösungsansatz scheint das aber unmöglich.

Eine weitere Quelle für die Probleme dieses Ansatzes ist die Skelettierung. Sie liefert insbesondere in der Nähe von Schnittpunkten schlechte Ergebnisse und schleppt neue Störungen in die Zeichnung ein.

Ideen, die von dem „Normalalgorithmus“ abweichen, gibt es nur sehr wenige. Eine Alternative ist in [Lu88] beschrieben. Der dort vorgestellte Algorithmus arbeitet kantenorientiert, d.h.: im Bild wird der Verlauf der Kanten der Linien verfolgt und vektorisiert. Aus zwei gegenüberliegenden Kantenvektoren wird dann ein „Corevector“ ermittelt, der den Verlauf der Linie approximiert. Somit gelangt man wieder zu einer Bildbeschreibung in vektorisierter Form. Auch hier werden Binärbilder verwendet. Da im Bild aber Kanten gesucht werden, ist dieser Algorithmus leicht für Grauwertbilder adaptierbar.

Ein ganz anderer Ansatz ist in [Dori93] zu finden. Dort wird versucht, die Erkennung dadurch zu beschleunigen, daß nur wenige Punkte des Bildes betrachtet werden. Das Bild wird entlang einem Raster nach Vordergrundpixeln durchsucht. Wird ein gesetztes Pixel gefunden, so wird, ausgehend von diesem Pixel, die Linie verfolgt. Auch bei der Linienverfolgung wird darauf geachtet, daß möglichst wenige Punkte betrachtet werden.

Ein Nachteil des Algorithmus liegt darin, daß sehr kurze Linien übersehen werden können. Linien werden also nur dann sicher erkannt, wenn sie eine gewisse Mindestlänge erreichen. Außerdem besteht ein Spannungsverhältnis zwischen der Grundidee, möglichst wenige Punkte zu betrachten und damit auch wenige Linienpunkte zu finden, und der Tatsache, daß die Lage und Form einer Linie umso genauer bestimmt werden kann je mehr Punkte von ihr bekannt sind.

Aus der Betrachtung der Literatur über das Erkennen technischer Zeichnungen ergeben sich folgende Punkte, die bei der Entwicklung eines neuen Algorithmus im Auge behalten werden sollten:

- Grauwertbilder: Es soll versucht werden, durch die Verwendung von Grauwertbildern die Genauigkeit der Ergebnisse zu verbessern.

- Speicherbedarf: Ganz besonders bei der Verwendung von Grauwertbildern muß auch auf den Speicherverbrauch geachtet werden. In [Hofe87] wird nach dem Vorschlag, Grauwertbilder zu verwenden, vorgerechnet, daß eine Zeichnung in A0 Format bei einer Auflösung von 20 Pixel/mm 400MB groß ist. So ein Bild kann auch als Binärbild nicht mehr in den Hauptspeicher geladen werden. Auch für etwas kleinere Bilder wäre ein Algorithmus, der das Bild in einem Durchlauf zeilenweise abarbeitet, sehr vorteilhaft. Dann bräuchte das Bild niemals komplett in den Speicher geladen werden, sondern könnte zeilenweise durch den Speicher geschleust werden.

- Parallelisierbarkeit: Selbstverständlich muß bei jedem Algorithmus auf die Rechenzeit geachtet werden. In der Bildverarbeitung sind aber aufgrund der großen Datenmengen fast alle Algorithmen ziemlich rechenintensiv. Deshalb werden bei praktischen Anwendungen sehr gerne parallele Rechnerarchitekturen verwendet. Es ist deshalb schon bei der Entwicklung neuer Verfahren auf die Parallelisierbarkeit zu achten.

1.3 LATEX-Bilderkennung

Die Aufgabenstellung für diese Diplomarbeit heißt LATEX-Bilderkennung. LATEX ist ein Textsatzsystem, das die Möglichkeit bietet, den Text mit Skizzen zu illustrieren. Sie bestehen, ganz ähnlich wie technische Zeichnungen, aus Linienelementen, vor allem Geraden und Kreisen. Somit ist die Diplomarbeit in das Forschungsgebiet „Erkennung von technischen Zeichnungen“ einzuordnen.

Durch die Spezialisierung der Problemstellung auf LATEX-Bilder ist das Vokabular der verwendeten Zeichenelemente klar definiert. Sowohl das Eingabebild als auch die Ausgabe, die Beschreibung des Bildes, müssen sich an die LATEX Syntax halten. Die Problemstellung ist somit in folgendem Kreis (Abb.1.1) darstellbar.

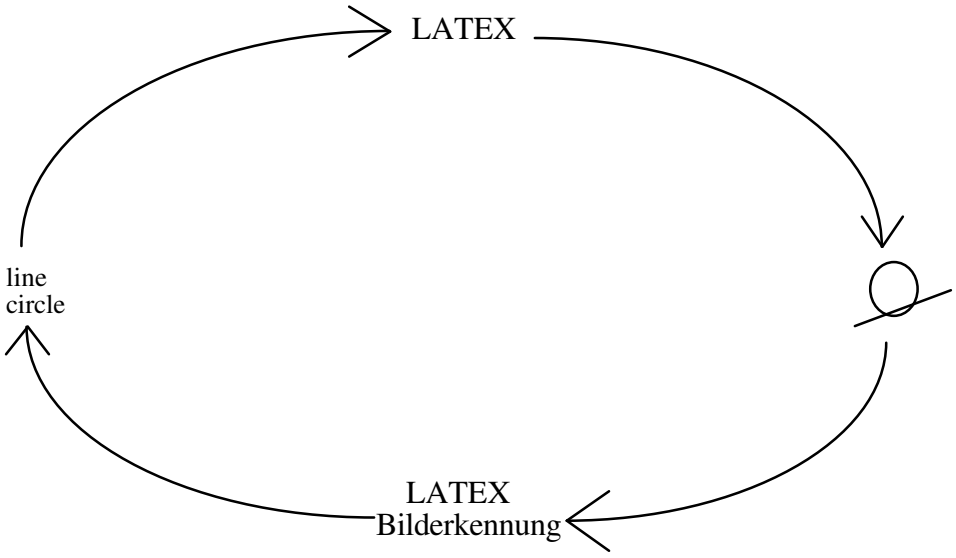

Abb.1.1 - Die Wechselwirkung zwischen LATEX und der LATEX-Bilderkennung

LATEX nimmt als Input eine Beschreibung der Skizze. Sie besteht aus Anweisungen, wo welches Primitiv zu zeichnen ist. Ausgabe ist die Zeichnung, die dann der Drucker ausgibt. Die LATEX-Bilderkennung leistet genau das Gegenteil. Hier ist die Zeichnung bzw. das digitale Bild, das der Scanner liefert, die Eingabe. Aus dem Bild soll wieder die LATEX-Beschreibung gewonnen werden. Da die Beschreibung ein gültiges LATEX Dokument sein soll, kann man das Ergebnis auch wieder mit LATEX ausdrucken.

Die LATEX-Bilderkennung schließt somit den Kreis zwischen LATEX-Bildbeschreibung und LATEX-Bild. Daraus ergeben sich gute Kontrollverfahren, mit denen die Korrektheit der Bilderkennung geprüft werden kann:

1. Bildvergleich: Aus einem LATEX-Bild wird mit der Bilderkennung ein LATEX-Dokument erstellt, das mit LATEX wieder ausgedruckt wird. Nun kann man die beiden Bilder, das Original und das Ergebnis nach dem Durchlaufen des Kreises, optisch

miteinander vergleichen. Auf diese Art kann man kontrollieren, ob alle Objekte eine Entsprechung im Ergebnis haben. Außerdem kann man die beiden Bilder übereinander legen und gegen das Licht halten. Dabei werden auch kleinere Abweichungen vom Original erkennbar. Daraus ergibt sich das Ziel, daß bei einer guten Beschreibung die maximale Distanz zwischen Beschreibung und Original klein bleiben soll.

Dieses Kontrollverfahren hat aber den Nachteil, daß auch eine Beschreibung des Bildes aus lauter einzelnen Punkten die Prüfung bestehen kann, wenn die Punkte richtig plaziert sind. Es ist aber keinesfalls das Ziel dieser Diplomarbeit, eine Beschreibung des Bildes aus einzelnen Punkten zu generieren, sondern es soll eine möglichst einfache und kurze Beschreibung gefunden werden, ähnlich der Beschreibung, die auch ein Mensch geben würde. Deshalb soll *eine* Gerade im Bild als *eine* Gerade in der Beschreibung enthalten sein.

2. Vergleich der Beschreibungen: Mit dem Durchlaufen des Kreises beginnend mit einem LATEX-Dokument, das man ausdruckt und anschließend der Bilderkennung präsentiert, ergibt sich ein anderes praktisches Kontrollverfahren, das die Schwäche des ersten Tests ausgleicht. Vergleicht man die beiden Beschreibungen, so sollten beide die gleichen Objekte enthalten, die außerdem noch in Position und Lage übereinstimmen sollen.

   Bei manchen Zeichnungen kann dieser Vergleich allerdings etwas schwierig sein. Wir werden diesen Test in Kapitel 3 auf einige Testzeichnungen anwenden und dabei auch scheinbar große Unterschiede zwischen Beschreibungen feststellen, die trotzdem die gleichen Bilder definieren. So ist z.B.: ein Startpunkt einer Geraden, der von einem ausgefüllten Kreis verdeckt ist, beliebig auf der Kreisfläche wählbar. (Er muß nur auf der Verlängerung der Geraden liegen.)

   Findet man echte Unterschiede in den Beschreibungen, so ist es recht schwierig zu erkennen, wie wichtig dieser Unterschied ist. Liegt z.B.: eine Gerade 1 mm neben ihrer wirklichen Position, so ist das ein großer Fehler. Ist aber andererseits der Radius eines Viertelkreises um einen 1 mm falsch und sein Mittelpunkt dementsprechend verschoben, so ist der Fehler wesentlich kleiner. Dieser Effekt wird in Kapitel 3 bei Testzeichnung 2 noch näher behandelt werden. Generell kann aber gesagt werden, daß das wesentliche Merkmal einer genauen Beschreibung eine kleine maximale Abweichung der Beschreibung von dem Original ist.

Um die Funktionalität der LATEX-Bilderkennung genauer zu definieren, soll im nächsten Kapitel das LATEX Vokabular beschrieben werden.

LATEX Vokabular für die Picture-Umgebung

Bei der Erstellung von LATEX Zeichnungen, können folgende Primitive verwendet werden. Eine genaue Beschreibung ist in [Lamp86] zu finden.

line (Richtungsvektor) {Länge} -	Gerade
vector (Richtungsvektor) {Länge} -	Pfeil
circle {Durchmesser} -	Kreis
*circle** {Durchmesser} -	Punkt, ausgefüllter Kreis
framebox (Größe) -	achsenparalleles Rechteck
dashbox {Strichlänge} (Größe) -	strichliertes Rechteck
oval (Größe) [Optionen]-	Rechteck mit abgerundeten Ecken

Die Bedeutung der Parameter ist selbsterklärend. Einzig die Optionen bei Ovalen müssen noch erläutert werden. Durch die Verwendung der Optionen (t)op, (b)ottom, (l)eft und (r)ight kann auch ein halbes Oval gezeichnet werden. Um ein Viertel-Oval zu spezifizieren, müssen zwei solche Optionen verwendet werden, z.B.: [tr] definiert das rechte obere Viertel eines Ovals.

Den Längen- und Größenangaben liegt eine, am Anfang eines LATEX-Bildes definierte, Längeneinheit zugrunde, üblicherweise 1mm.

All diese Primitive definieren nur Form und Größe der Objekte, nicht aber, wo sie gezeichnet werden sollen. Sie müssen deshalb noch mit einer *put* oder *multiput* Anweisung kombiniert werden:

put (Position) {Objekt}

multiput (Position) (d_x, d_y) {n} {Objekt}

put zeichnet das Objekt an der angegebenen Position. Die Position bezieht sich bei Geraden und Pfeilen auf den Startpunkt, bei Kreisen auf den Mittelpunkt, bei Rechtecken auf das linke obere Eck, bei Ovalen allerdings auf die Mitte.

Mit einer *multiput* Anweisung kann das Objekt gleich mehrfach gezeichnet werden. Beginnend bei der angegebenen Position wird das Objekt n mal gezeichnet. (d_x, d_y) gibt dabei den Abstand zwischen zwei aufeinanderfolgenden Objekten an.

In LATEX gibt es noch etliche Einschränkungen, die aus der obigen Liste nicht zu erkennen sind. So besteht der Richtungsvektor (x, y), der beim Zeichnen einer Geraden angegeben wird, aus ganzen Zahlen aus dem Intervall [-6, +6]. D.h.: der Anstieg einer Geraden ist auf

wenige diskrete Werte beschränkt. Noch stärker ist die Einschränkung bei Pfeilen: Hier müssen x und y aus dem Intervall [-4, +4] sein.

Auch für Kreise gibt es eine Einschränkung. Bei normalen Kreisen muß der Durchmesser kleiner als 14mm sein, bei ausgefüllten Kreisen sogar kleiner als 5.3mm.

Zuletzt sei noch bemerkt, daß Geraden immer eine Mindestlänge von 3.5mm haben müssen. Kürzere Geraden werden überhaupt nicht gezeichnet.

All diese Einschränkungen machen vor allem dem Benutzer von LATEX Schwierigkeiten. Aber auch bei der Bilderkennung verursachen sie Probleme. Die Beschreibung der Zeichnung, also das Ergebnis der Bilderkennung, ist ein LATEX Dokument. Es muß sich deshalb auch an die LATEX Syntax mit all ihren Einschränkungen halten und darf z.B. nur Geraden mit gültigem Anstieg enthalten.

2. Algorithmus

Der in diesem Kapitel beschriebene Algorithmus zur Erkennung von LATEX-Bildern wurde im Rahmen der Diplomarbeit auf einem PC implementiert. Er weicht ganz wesentlich von den aus der Literatur bekannten Lösungsansätzen ab. Der im folgenden erklärte Algorithmus besitzt als Herzstück eine regelmäßige 2*2/4 Pyramide [Krop91], [Rose84], die hier verwendet wird, um die einzelnen Kantenpunkte zu möglichst langen Geraden und Kreisbögen zusammenzusetzen. Außerdem arbeitet er kantenorientiert. Das bedeutet, daß aus dem Bild die Kanteninformation extrahiert wird. Dadurch ist es auch nicht notwendig, das Bild zu binarisieren. Vielmehr kann der Algorithmus als Eingabebild ein Grauwertbild verwenden und somit genauere Ergebnisse liefern.

Bevor die einzelnen Arbeitsschritte ausführlich beschrieben werden, wird noch ein Überblick über den Lösungsansatz gegeben, in dem die Funktionalität der einzelnen Schritte erläutert wird.

2.1 Überblick

Pyramide

Die Pyramide hat die Aufgabe, die riesige Menge einzelner Kantenpunkte, die ein Kantenoperator im Bild findet, zu Geraden und Kreisbögen zusammenzufassen. Jeder Pixel der Pyramide enthält als Information eine Liste aller Kanten in seinem rezeptiven Feld. Diese Pyramide darf nicht mit der Kurvenpyramide aus [Krop86] verwechselt werden, da bei der Reduktion in der hier verwendeten Pyramide die Information über die genaue Lage aller Kanten erhalten bleibt. Im Apex, der Spitze der Pyramide, ist somit die genaue Lage aller Kanten des Bildes verfügbar.

Die Reduktion - und auch der Sinn - der Pyramide besteht lediglich darin, die Anzahl der Kanten soweit wie möglich zu verringern, indem aneinanderstoßende Kanten, die auch durch eine Kante beschrieben werden können, vereinigt werden.

Abb.2.1 illustriert an einem Beispiel das Ergebnis der Pyramide, das ist die Information, die im Apex enthalten ist. Dem Grauwertbild sind die Kanteninformation und das Raster des Bildes (eine Linie entspricht 8 Pixel) überlagert. Die Rauten markieren die Endpunkte der einzelnen Kanten.

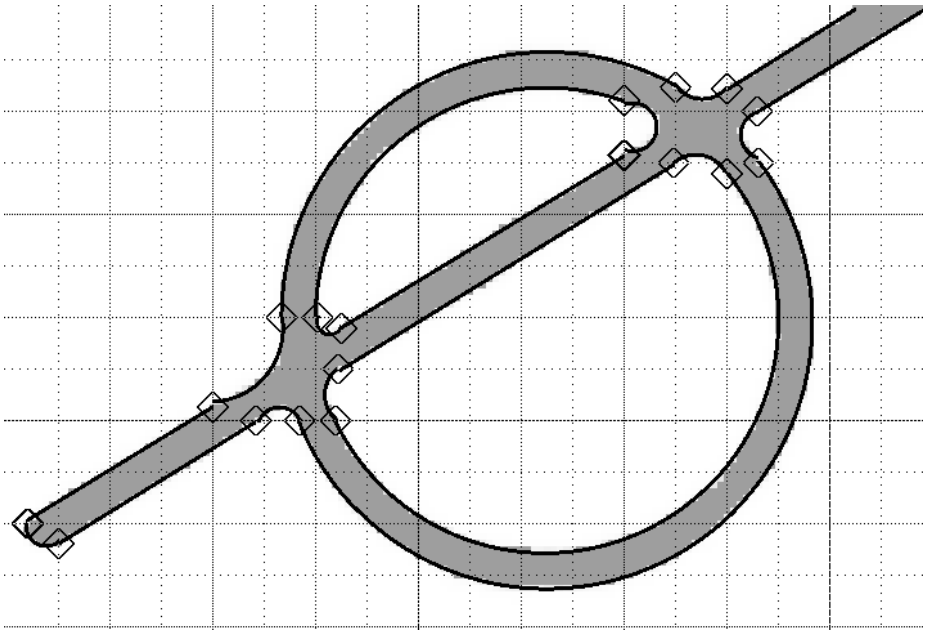

Abb.2.1 - Das Ergebnis der Pyramide. Dem Grauwertbild ist die im Apex enthaltene Kanteninformation überlagert. Die Rauten markieren das Ende einer Kante.

Artefakte

Abb.2.1 zeigt einerseits, daß sehr lange Geraden und Kreisbögen durch nur eine Kante beschrieben werden, andererseits werden aber auch zwei Probleme offenbar:

- Shiftvarianz: Die meisten Endpunkte der Kanten fallen genau auf die Rasterlinien. Die Ursache dafür liegt in der regelmäßigen Architektur der Pyramide und wird in Kapitel 2.3 bei einer genaueren Betrachtung der Pyramide verständlich werden.

- Artefakte: Es ist gewissermaßen eine Folge der Shiftvarianz, daß bei Schnittpunkten zweier Kanten meistens ein kurzes Kantenstück aus Punkten beider Kanten entsteht. Dieses kurze Kantenstück wird von nun an "Artefakt" genannt.

Der zweite Schritt des Algorithmus beschäftigt sich mit dem *Erkennen* und der *Elimination* dieser Artefakte. Die *Elimination* kann recht einfach dadurch erfolgen, daß das Artefakt aus der Liste der Kanten gelöscht wird und die beiden benachbarten Kanten bis zu ihrem Schnittpunkt verlängert werden. Die Schwierigkeit besteht aber im *Erkennen*, ob eine Kurve eliminiert werden kann, ohne daß eine zu große Abweichung von den Bilddaten verursacht wird.

Diese Problematik wird in Kapitel 2.4 näher behandelt. Außerdem wird dort gezeigt werden, daß die Elimination der Artefakte kein lästiger zusätzlicher Arbeitsschritt ist, der durch eine Unzulänglichkeit der Pyramide notwendig geworden ist, sondern positive Auswirkungen auf die Genauigkeit der Ergebnisse hat.

Das Testbild nach Elimination der Artefakte ist in Abb.2.2 abgebildet.

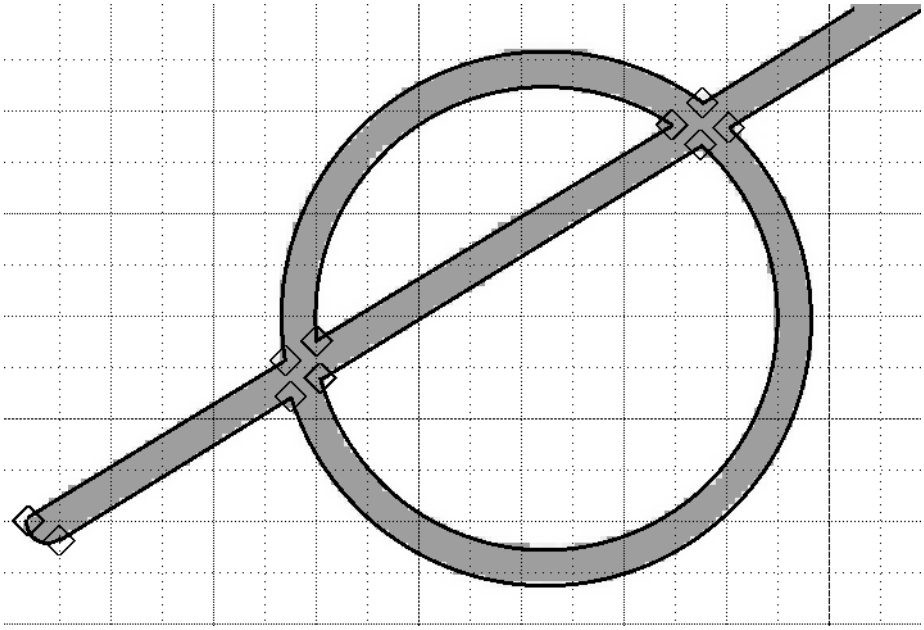

Abb.2.2 - Das Testbild nach Elimination der Artefakte.

Verdrehungswinkel

Es ist fast unmöglich, eine Zeichnung so einzuscannen, daß die x-Achse der Zeichnung mit jener des digitalen Bildes übereinstimmt. Es wird bei der Digitalisierung immer das Bild verdreht. Normalerweise ist es absolut unmöglich, diesen Verdrehungswinkel zu bestimmen, da das digitalisierte Bild diese Information einfach nicht enthält. Im speziellen Fall der LATEX-Bilderkennung ist das sehr wohl möglich.

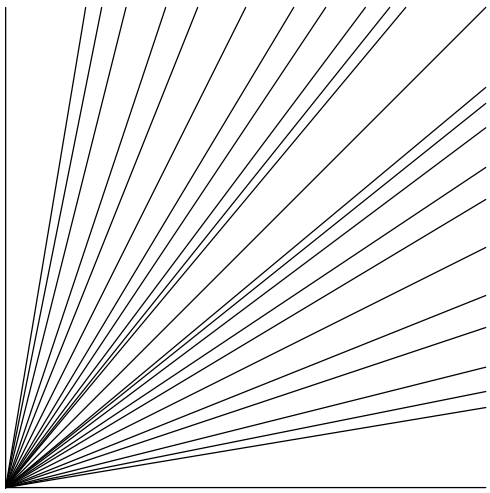

Abb.2.3 - Alle in LATEX erlaubten Geradenanstiege im ersten Quadranten

Die Information zur Bestimmung des Verdrehungswinkels wird aus den Geraden gewonnen. In LATEX ist der Anstieg einer Geraden auf einige diskrete Werte beschränkt (vgl. Kap.1.3). Abb.2.3 zeigt alle gültigen Anstiege im ersten Quadranten. Würde man einfach annehmen, das Bild sei überhaupt nicht verdreht, so könnte man die meisten Geraden im Ergebnisdokument, das sich ja an die LATEX-Syntax halten muß, gar nicht beschreiben.

Die Bestimmung des Verdrehungswinkels ist somit eine Suche nach jenem Winkel, um den das gesamte Bild gedreht werden muß, so daß möglichst viele Geraden einen gültigen Anstieg haben.

Da die gültigen Anstiege unregelmäßig verteilt sind, ist es möglich den Verdrehungswinkel bis auf ein Vielfaches von 90° zu bestimmen, wenn mehrere Geraden verschiedenen Anstiegs im Bild vorkommen.

Kantenfolgen

In dem Testbild aus Abb.2.1 und Abb.2.2 war *eine* Gerade und *ein* Kreis zu sehen. Im Gegensatz dazu bestand die Beschreibung, etwa der Geraden, aus je zwei Kanten *vor*, *in* und *hinter* dem Kreis. Um einen Schritt näher zum Endergebnis zu kommen, werden nun Folgen von Kanten gesucht, die aus irgendeinem Grund, wie z.B.: einer Überschneidung, unterbrochen sind.

Anders formuliert bedeutet das, daß in der Menge der Kanten nach Fortsetzungen gesucht wird. Abb.2.4 zeigt das Ergebnis dieses Schrittes. Die Kantenfolgen sind als durchgezogene Linien eingezeichnet. Das bedeutet aber nicht, daß die Kantenfolgen von nun an als eine Kante betrachtet werden. Die Information, wo wirklich eine Kante zu sehen ist, und wo eine Unterbrechung ist, bleibt erhalten.

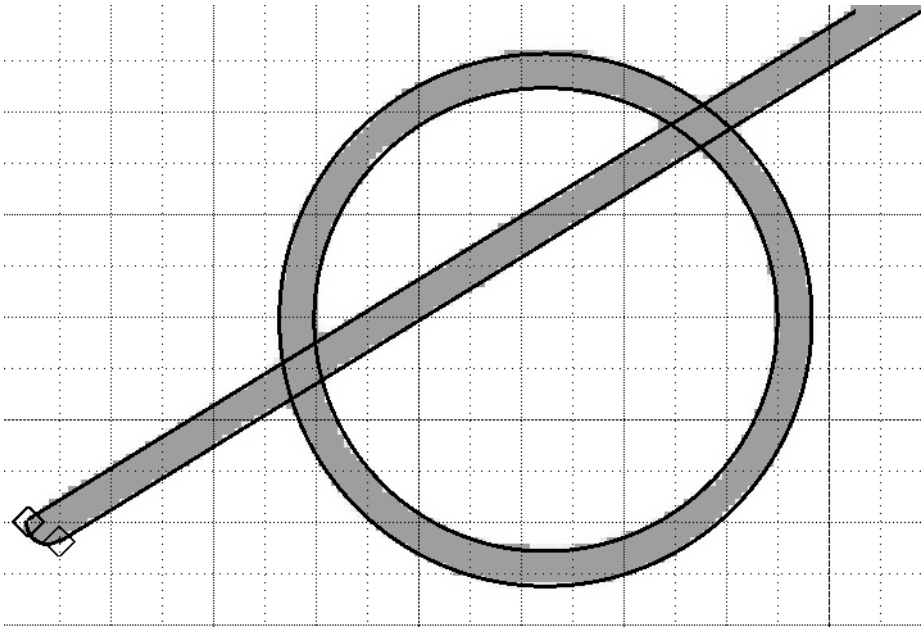

Abb.2.4 - Bei der Suche nach Kantenfolgen werden Kanten, die durch Überschneidungen mit anderen Linien zerrissen sind, zu einer Kantenfolge zusammengefaßt.

Linienanalyse

Die Linienanalyse ist der letzte Schritt, der noch nötig ist, um das Testbild mit einer Geraden und einem Kreis beschreiben zu können. Dazu wird aus je zwei gegenüberliegenden Kantenfolgen eine Linie gebildet.

Die Schwierigkeit liegt hier nicht darin, gegenüberliegende Folgen zu finden, sondern in der Behandlung der Unterbrechungen in den Kantenfolgen. Es muß jeweils erkannt werden, ob es sich um eine Unterbrechung der *Linie* handelt (z.B.: strichlierte Linie), oder ob nur eine Unterbrechung der *Kante* (z.B.: Überschneidung mit einer anderen Linie) vorliegt. Um diese Frage beantworten zu können, müssen auch die benachbarten Kanten betrachtet werden. Liegt eine Unterbrechung der Linie vor, so muß eine Kante existieren, die das Ende der Linie markiert. Ansonsten darf die Linie durchgezogen werden.

Weitere Punkte, die in diesen Arbeitsschritt fallen und mit der Linienanalyse in Kapitel 2.7 näher beschrieben werden, sind die Erkennung von Pfeilspitzen als ein besonderes Ende einer Geraden und das Vereinigen von Geraden und Kreisbögen zu Ovalen und Rechtecken.

Multiput und Ausgabe

Das Ergebnis der Linienanalyse besteht schon aus den LATEX-Primitiven (Gerade, Vektor, Kreis, Oval und Rechteck). Bei der Ausgabe müssen die Objekte nur noch in LATEX-Syntax konvertiert werden. Der einzige Rechenaufwand besteht darin, daß versucht wird, gleiche Objekte mit einer *multiput* Anweisung auszugeben.

2.2 Kantenrepräsentation in der Pyramide

Die gesamte LATEX-Bilderkennung arbeitet kantenorientiert. Deshalb wird noch vor einer detaillierten Beschreibung des Algorithmus die interne Kantenrepräsentation erklärt. Die wichtigste Anforderung an die Kantenrepräsentation kennen wir bereits aus dem Überblick: Die genaue Lage und Form der Kante soll möglichst schnell und einfach aus ihrer Repräsentation bestimmbar sein.

Um das zu ermöglichen, wird eine Akkumulator-Repräsentation, ähnlich wie in [Hart85], verwendet. Von jeder Kante wird neben dem Anfangs- und Endpunkt auch die Summe über alle Kantenpunkte (x, y) von 1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^2y^2 und y^4 gespeichert. Im folgenden wird gezeigt, wie aus dieser Repräsentation die Lage der Kante durch Approximation mit einer Geraden bzw. Kreis nach der Methode des kleinsten quadratischen Fehlers gewonnen werden kann. Dabei werden nur lineare Gleichungssysteme aufgelöst. Außerdem wird die numerische Stabilität überprüft.

Folgende Konventionen gelten im ganzen Kapitel:

(x_i, y_i)	Kantenpunkt
n	Anzahl der Kantenpunkte
$\overline{x} = \frac{1}{n}\sum_i x_i$	Mittelwert von x
$V_x = \frac{1}{n}\sum_i (x_i - \overline{x})^2 = \overline{x^2} - \overline{x}^2$	Varianz von x
$K_{x,y} = \frac{1}{n}\sum_i (x_i - \overline{x})(y_i - \overline{y}) = \overline{xy} - \overline{x} \cdot \overline{y}$	Kovarianz von x und y

Approximation mit einer Geraden

Zunächst wird gezeigt, wie die Menge der Kantenpunkte durch eine Gerade approximiert werden kann. Wichtig ist dabei die Voraussetzung, daß die Kantenpunkte nur durch die Akkumulator-Repräsentation gegeben sind. Die Lage der einzelnen Punkte darf bei der Approximation nicht verwendet werden.

Die Approximation der Kante erfolgt mit der Geraden:

$$g(\varphi,r): x \cdot \sin\varphi - y \cdot \cos\varphi + r = 0 \qquad (1.1)$$

Die Parameter werden so gewählt, daß der durchschnittliche quadratische Fehler

$$F_G(\varphi,r) = \frac{1}{n}\sum_i \left(x_i \cdot \sin\varphi - y_i \cdot \cos\varphi + r\right)^2 \tag{1.2}$$

minimal wird. Somit gilt:

$$0 = \frac{\partial F_G(\varphi,r)}{\partial r} = \frac{2}{n}\sum_i \left(x_i \cdot \sin\varphi - y_i \cdot \cos\varphi + r\right) \tag{1.3}$$

$$\Rightarrow \quad r = \bar{y} \cdot \cos\varphi - \bar{x} \cdot \sin\varphi \tag{1.4}$$

Eingesetzt in die Fehlerfunktion (1.2) ergibt das:

$$\begin{aligned} F_G(\varphi) &= \frac{1}{n}\sum_i \left(\left(x_i - \bar{x}\right) \cdot \sin\varphi - \left(y_i - \bar{y}\right) \cdot \cos\varphi\right)^2 = \\ &= \sin^2\varphi \cdot V_x + \cos^2\varphi \cdot V_y - 2 \cdot \sin\varphi \cdot \cos\varphi \cdot K_{x,y} \end{aligned} \tag{1.5}$$

Differenzieren nach φ und Nullsetzen liefert dann:

$$2 \cdot \varphi = \arctan\left(\frac{2 \cdot K_{x,y}}{V_x - V_y}\right) \tag{1.6}$$

Mit den Formeln (1.4) und (1.6) lassen sich also die beiden Parameter φ und r leicht aus dem Akkumulator bestimmen. Große Bedeutung für den Erkennungsprozeß hat die Frage, ob die Approximation gut genug ist, daß man die Punkte als eine Gerade erkennen darf. Dafür wird der durchschnittliche quadratische Fehler mit einem Schwellwert verglichen. Bleibt er unter dem Schwellwert und liegen außerdem die Endpunkte nahe an der Approximationsgeraden, so wird die Approximation als gut genug bezeichnet. Auch die Fehlerfunktion ist mittels (1.5) aus dem Akkumulator leicht bestimmbar. Auf die Bedeutung des Schwellwerts wird in den Kapiteln 3.1 und 3.3 noch näher eingegangen.

Approximation mit einem Kreis

Auch bei der Approximation mit einem Kreis

$$k(c_x, c_y, r): \ \left(c_x - x\right)^2 + \left(c_y - y\right)^2 = r^2 \tag{2.1}$$

wird zur Bestimmung der Parameter eine Fehlerfunktion minimiert:

$$F_K\left(c_x, c_y, r\right) = \frac{1}{n}\sum_i \left(\left(c_x - x_i\right)^2 + \left(c_y - y_i\right)^2 - r^2\right)^2 \tag{2.2}$$

Auch hier lassen sich die Parameter auf linearem Weg bestimmen. Das wird nach der Elimination von r offensichtlich:

$$0=\frac{\partial F_K\left(c_x, c_y, r\right)}{\partial r}=\frac{-4\cdot r}{n}\sum_i\left(\left(c_x-x_i\right)^2+\left(c_y-y_i\right)^2-r^2\right) \tag{2.3}$$

$$\Rightarrow\quad r^2=c_x^2+c_y^2-2\cdot\overline{x}\cdot c_x-2\cdot\overline{y}\cdot c_y+\overline{x^2}+\overline{y^2} \tag{2.4}$$

Eingesetzt in (2.2) ergibt das:

$$\begin{aligned}F_K\left(c_x, c_y\right)=\frac{1}{n}\sum_i\left(-2\cdot c_x\cdot\left(x_i-\overline{x}\right)-2\cdot c_y\cdot\left(y_i-\overline{y}\right)+x_i^2-\overline{x^2}+y_i^2-\overline{y^2}\right)^2=\\ =4\cdot c_x^2V_x+4\cdot c_y^2V_y+8\cdot c_xc_yK_{x,y}-4\cdot c_x\cdot\left(K_{x,x^2}+K_{x,y^2}\right)-\\ -4\cdot c_y\cdot\left(K_{y,x^2}+K_{y,y^2}\right)+V_{x^2}+V_{y^2}+2\cdot K_{x^2,y^2}\end{aligned} \tag{2.5}$$

Auch wenn dieser Term etwas länger ist und kompliziert aussieht, ist der gemeinsame Grad von c_x und c_y in diesem Polynom doch nicht größer als zwei. Nach dem Differenzieren entsteht somit ein lineares Gleichungssystem, aus dem c_x und c_y leicht zu bestimmen sind.

Bei der Bestimmung des durchschnittlichen quadratischen Fehlers, um wiederum ein Qualitätsmaß für die Approximation zu erhalten, muß man aber etwas vorsichtiger sein. Die Formel für die durchschnittliche quadratische Abweichung der Punkte vom Kreisbogen lautet nämlich:

$$F_K^*\left(c_x, c_y, r\right)=\frac{1}{n}\sum_i\left(\sqrt{\left(c_x-x_i\right)^2+\left(c_y-y_i\right)^2}-r\right)^2 \tag{2.6}$$

Da aus dieser Fehlerfunktion die Kreisparameter nicht linear bestimmbar sind und sie aus dem Akkumulator auch gar nicht berechenbar ist, wurde die ähnliche Fehlerfunktion (2.2) gewählt. Für die Umrechnung zwischen den beiden Funktionen gilt:

$$\begin{aligned}F_K\left(c_x, c_y, r\right)=\frac{1}{n}\sum_i\left(\left(c_x-x_i\right)^2+\left(c_y-y_i\right)^2-r^2\right)^2=\\ =\frac{1}{n}\sum_i\left(\sqrt{\left(c_x-x_i\right)^2+\left(c_y-y_i\right)^2}-r\right)^2\cdot\left(\sqrt{\left(c_x-x_i\right)^2+\left(c_y-y_i\right)^2}+r\right)^2\approx\\ \approx 4\cdot r^2\cdot F_K^*\left(c_x, c_y, r\right)\end{aligned} \tag{2.7}$$

Das Kriterium, ob die Kante als Kreis beschrieben werden kann, lautet deshalb:

$$\frac{F_K(c_x, c_y, r)}{4 \cdot r^2} \quad < \quad \text{Schwellwert} \tag{2.8}$$

Numerische Stabilität

Dieser Abschnitt behandelt die Frage, ob die oben beschriebene Approximation numerisch stabil ist. Bei näherer Betrachtung erscheint nämlich die Stabilität bei der Auswertung der Fehlerfunktionen (1.5) bzw. (2.5) fragwürdig. Beide Funktionen werden als Summe sehr großer Zahlen berechnet. So enthält (2.5) z.B.: den Summanden $V_{x^2} = \overline{x^4} - \overline{x^2}^2$. Bei einem recht großen Bild, etwa 3000*3000 Pixel, kann $\overline{x^4}$ die Größenordnung von 10^{14} erreichen. Der Funktionswert von F_K liegt aber nach Division durch $4r^2$ in der Größenordnung von eins (er wird ja nachher mit einem Schwellwert, z.B.: 0.3, verglichen)! Somit tritt bei der Auswertung von F_K ein Auslöschungseffekt auf. Das bedeutet, daß der *absolute* Fehler, der bei der Berechnung der einzelnen Summanden gemacht wird und dort noch *relativ* klein ist, im Ergebnis relativ *groß* werden kann, da nun das sehr kleine Ergebnis mit dem absoluten Rechenfehler der sehr großen Summanden behaftet ist.

Glücklicherweise muß die Bestimmung des Approximationsfehlers nicht sehr genau sein. Dennoch muß wenigstens die erste Stelle hinter dem Komma stimmen. Deshalb wird in diesem Abschnitt ein maximal tolerierbarer Rechenfehler von 0.05 angenommen.

Auch für diesen Abschnitt müssen ein paar Konventionen getroffen werden:

$\tilde{F}(x)$	Auswertung von F in der Gleitkommaarithmetik	
$a_1 \oplus a_2 = (a_1 + a_2) \cdot (1+\mu)$	Addition in der Gleitkommaarithmetik	(3.1)

Dabei ist μ der relative Rechenfehler, der bei der Addition gemacht wird. Er ist betragsmäßig immer kleiner als ε, die Maschinengenauigkeit. Bei Verwendung der doppelt genauen Gleitkommaarithmetik (das entspricht dem Typ *double* in C) gilt:

$$|\mu| \le \varepsilon = 2^{-53} \approx 1.1 \cdot 10^{-16} \tag{3.2}$$

Zunächst wird nun der Fehler abgeschätzt, der bei der Addition mehrerer Zahlen entsteht:

$$\begin{aligned}
\sum_i^{\oplus} a_i &= \left(\left(a_1 \oplus a_2 \right) \oplus a_3 \right) \oplus \ \ldots \ \oplus a_n = \\
&= \left(\left(\left(a_1 + a_2 \right) \cdot \left(1 + \mu_2 \right) + a_3 \right) \cdot \left(1 + \mu_3 \right) + \ \ldots \ + a_n \right) \cdot \left(1 + \mu_n \right) \approx \\
&\approx \left(a_1 + a_2 \right) \cdot \left(1 + \mu_2 + \mu_3 + \ \ldots \ + \mu_n \right) + \\
&\qquad + a_3 \cdot \left(1 + \mu_3 + \mu_4 + \ \ldots \ + \mu_n \right) + \ \ldots \ + a_n \cdot \left(1 + \mu_n \right)
\end{aligned} \tag{3.3}$$

Dabei wurden höhere Potenzen der φ_i weggelassen. Für den absoluten Rechenfehler in der Summe folgt daraus:

$$\left|\sum_i a_i - \sum_i^{\oplus} a_i\right| \le (a_1 + a_2)\cdot(n-1)\cdot\varepsilon + a_3\cdot(n-2)\cdot\varepsilon + \ \ldots\ + a_n\cdot\varepsilon \le \\ \le \max_i\{|a_i|\}\cdot\varepsilon\cdot\frac{n\cdot(n+1)}{2} \tag{3.4}$$

Damit ist es nicht mehr schwer, den Fehler bei der Berechnung der Varianz von x abzuschätzen. In den folgenden Formeln wird davon ausgegangen, daß ein N*N Bild analysiert wird. N ist dann eine obere Schranke für $|x_i|$ und $|y_i|$.

$$\left|V_x - \tilde{V}_x\right| \le \frac{1}{n}\cdot\left|\sum_i x_i^2 - \sum_i^{\oplus} x_i^2\right| + \frac{2}{n^2}\cdot\left|\sum_i x_i\right|\cdot\left|\sum_i x_i - \sum_i^{\oplus} x_i\right| \le \\ \le \varepsilon\cdot N^2\cdot\frac{n\cdot(n+1)}{2\cdot n} + n\cdot N\cdot\varepsilon\cdot N\cdot\frac{2\cdot n\cdot(n+1)}{2\cdot n^2} = \\ = \frac{3\cdot(n+1)}{2}\cdot\varepsilon\cdot N^2 \tag{3.5}$$

Dieselbe Abschätzung gilt auch für V_y und $K_{x,y}$. Bei höheren Potenzen ändert sich nur der Exponent von N. z.B.:

$$\left|V_{x^2} - \tilde{V}_{x^2}\right| \le \frac{1}{n}\cdot\left|\sum_i x_i^4 - \sum_i^{\oplus} x_i^4\right| + \frac{2}{n^2}\cdot\left|\sum_i x_i^2\right|\cdot\left|\sum_i x_i^2 - \sum_i^{\oplus} x_i^2\right| \le \\ \le \varepsilon\cdot N^4\cdot\frac{n\cdot(n+1)}{2\cdot n} + n\cdot N^2\cdot\varepsilon\cdot N^2\cdot\frac{2\cdot n\cdot(n+1)}{2\cdot n^2} = \\ = \frac{3\cdot(n+1)}{2}\cdot\varepsilon\cdot N^4 \tag{3.6}$$

Damit läßt sich der absolute Rechenfehler bei der Bestimmung von F_G folgendermaßen abschätzen:

$$\left|F_G(\varphi) - \tilde{F}_G(\varphi)\right| \le \left|V_x - \tilde{V}_x\right|\sin^2\varphi + \left|V_y - \tilde{V}_y\right|\cos^2\varphi + 2\cdot\left|K_{x,y} - \tilde{K}_{x,y}\right|\cdot\left|\sin\varphi\cdot\cos\varphi\right| \le \\ \le \left(\sin^2\varphi + \cos^2\varphi + 2\cdot\left|\sin\varphi\cdot\cos\varphi\right|\right)\cdot\frac{3\cdot(n+1)}{2}\cdot N^2\cdot\varepsilon \le \\ \le 3\cdot(n+1)\cdot N^2\cdot\varepsilon \tag{3.7}$$

Beachtet man nun noch, daß eine Gerade in einem N*N Bild aus höchstens 2*N Punkten bestehen kann, und setzt man für ε ein, so erhält man für einen maximal tolerierbaren Rechenfehler von 0.05:

$$6 \cdot N^3 \cdot 1.1 \cdot 10^{-16} < 0.05 \quad \Rightarrow \quad N < 42000 \tag{3.8}$$

Demnach bleibt die Auswertung der Fehlerfunktion bis zu einer Bildgröße von 42000*42000 Pixel stabil. Selbst für große Bilder von z.B.: 5000*5000 Pixel bleibt der Rechenfehler stets kleiner als 10^{-4} und stellt kein Problem bei der Geradenerkennung dar.

Noch gefährlicher erscheint die Situation bei Kreisen, da hier die Koordinaten bis zur vierten Potenz erhoben werden. Bei der Auswertung nach (2.5) ergibt sich folgende Abschätzung für den Rechenfehler, wobei noch berücksichtigt wird, daß die Anzahl der Kantenpunkte immer kleiner als 8*r sein muß:

$$\begin{aligned} RF_{abs} &= \frac{1}{4 \cdot r^2} \left| F_K\left(c_x, c_y\right) - \tilde{F}_K\left(c_x, c_y\right) \right| \leq \\ &\leq \frac{1}{4 \cdot r^2} \cdot \frac{3 \cdot (n+1)}{2} \cdot \varepsilon \cdot N^2 \cdot \left(4c_x^2 + 4c_y^2 + 8c_x c_y + 8Nc_x + 8Nc_y + 4N^2\right) \leq \\ &\leq \frac{108}{r} \cdot \varepsilon \cdot N^4 \end{aligned} \tag{3.9}$$

Ist z.B.: N = 4000 und r = 10, so lautet die Abschätzung für den Rechenfehler 0.3. Die Kreiserkennung bricht also numerisch zusammen.

Dieses Beispiel war allerdings besonders ungünstig gewählt, da N sehr groß und r sehr klein gewählt wurde. Mit einem kleinen Trick ist die Stabilität wieder zu gewährleisten. Man wählt für den Akkumulator jeder Kante einen eigenen Ursprung. z.B.: den ersten Kantenpunkt, den man von dieser Kante findet. Im Akkumulator werden dann die Bildkoordinaten relativ zu diesem Ursprung aufsummiert. Dadurch geht jeder Kreis in etwa durch den Ursprung, und N, jetzt wieder als Abschätzung für die $|x_i|$ und $|y_i|$ verwendet, bleibt deshalb kleiner als 2*r. Daraus folgt:

$$RF_{abs} \leq 1728 \cdot 1.1 \cdot 10^{-16} \cdot r^3 < 0.05 \quad \Rightarrow \quad r < 6400 \tag{3.10}$$

Auch diese Einschränkung stellt in der Praxis kein Problem dar.

Dieses Kapitel zeigt, daß die Akkumulator-Repräsentation die Information über die genaue Lage und Form einer Kante enthält. Die Bestapproximation einer Kante mit einer Geraden oder einem Kreis erfordert keine aufwendige Nullstellensuche, sondern ist auf linearem Weg durchführbar. Die Überprüfung der numerischen Stabilität ergab Einschränkungen für die Bildgröße und den Kreisradius, die problemlos erfüllbar sind. Beide Einschränkungen sind wohl eher theoretischer Natur und verursachen in der Praxis keine Schwierigkeiten.

Es mag vielleicht überraschen, daß hier ganz einfach die Methode des kleinsten quadratischen Fehlers verwendet wird, obwohl heute meistens die robuste Statistik empfohlen wird [Rous87]. Die Motivation, robuste Methoden zu entwickeln, war das Problem, daß einzelne Ausreißer die normale Approximation sehr stark stören können. Hier wird aber eine Punktmenge approximiert, die aus dicht hintereinander liegenden Punkten einer Kante besteht. Es kann deshalb keine einzelnen Ausreißer geben. Somit ist die Methode des kleinsten quadratischen Fehlers für diese Aufgabenstellung absolut ausreichend.

2.3 Pyramide

Die Aufgabe der Pyramide besteht darin, die große Anzahl einzelner Kantenpunkte zu möglichst wenigen langen Kanten zusammenzufassen. Jeder einzelne Pixel dieser Pyramide enthält eine Liste aller Kanten in seinem rezeptiven Feld. Die Repräsentation der Kanten wurde im vorigen Kapitel ausführlich beschrieben und analysiert. Hier soll nur noch einmal betont werden, daß in der Pyramide keine Information über Lage und Form der Kanten verlorengeht. Die Reduktion, die in dieser Pyramide geschehen soll, besteht einzig und allein in der Verringerung der Anzahl der Kanten. Auf der untersten Ebene ist jede Kante höchstens einen Pixel lang. Während der Reduktion werden dann immer wieder kurze Kantenstücke zu einem längeren vereinigt.

Die erste Ebene der Pyramide

Die Berechnung der ersten Ebene der Pyramide ist durch Abb.2.5 illustriert. Dort ist dem Grauwertbild das Raster der ersten Pyramidenebene überlagert. Dieses Raster hat dieselbe Auflösung wie das Bild, ist aber um ½Pixel in x- und y-Richtung verschoben. Fett eingezeichnet ist die Kanteninformation, die aus diesem Bild gewonnen wird.

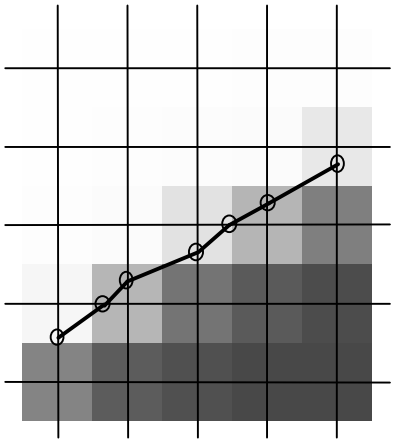

Abb.2.5 - Die erste Ebene der Pyramide

Zur Berechnung eines Pixels der untersten Pyramidenebene wird ein 2*2 Fenster des Grauwertbildes betrachtet. Verläuft eine Kante durch dieses Fenster, so werden die in der Abbildung mit o markierten Punkte, an denen die Kante das Pyramidenpixel verläßt, bestimmt. Für die Lokalisierung dieser Punkte mit Subpixelgenauigkeit wird der Grauwertverlauf zwischen den beiden Pixel linear interpoliert, und jener Punkt bestimmt, an dem der Grauwert einen vorgegebenen Schwellwert erreicht. Der Akkumulator des Kantenstücks auf der ersten Ebene wird dann mit diesen beiden Punkten initialisiert.

Reduktion in der Pyramide

Bei der Reduktion in einer 2*2/4 Pyramide bilden jeweils 2*2 Pixel einer Ebene einen Pixel der nächst höheren Ebene. In dieser Pyramide enthält jeder Pixel eine Liste aller Kantenstücke, die in dem entsprechenden Bildausschnitt liegen. Somit muß bei der Reduktion aus den Beschreibungen der vier kleinen Ausschnitte eine Gesamtbeschreibung gewonnen werden. (Abb.2.6) Die Schwierigkeit besteht darin, aneinanderstoßende Kantenstücke nach Möglichkeit zu einer Kante zu vereinigen, um die Anzahl der Kanten zu reduzieren und die Beschreibung zu vereinfachen.

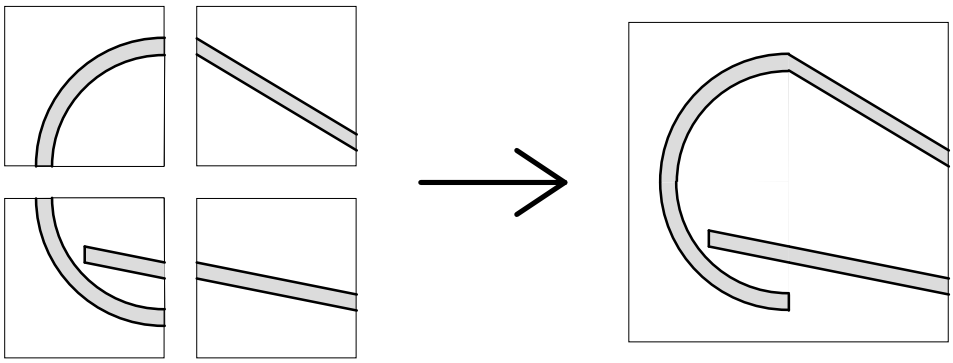

Abb.2.6 - Reduktion in der Pyramide

Abb.2.6 zeigt links das 2*2 Fenster der unteren Ebene, rechts den Pixel der darüberliegenden Ebene. (Daß hier immer zwei parallele Kanten eingezeichnet sind, soll nur daran erinnern, daß der Algorithmus kanten- und nicht linienorientiert arbeitet.) Zwei Kanten werden nur dann zu einer Kante zusammengefaßt, wenn sie aneinanderstoßen und passen, wie z.B. die beiden Kreisbögen in der linken Hälfte oder die Geraden in der unteren Hälfte. Hingegen können in der oberen Hälfte die Gerade und der Kreisbogen natürlich nicht durch eine Kante beschrieben werden.

Um die Entscheidung, ob zwei Kanten passen, treffen zu können, werden zunächst die beiden Kanten miteinander vereinigt. Das bedeutet, daß ihre beiden Akkumulatoren addiert werden. Dann wird die Gesamtkurve mit einer Geraden bzw. Kreis approximiert (siehe Kap. 2.2). Ist die Approximation gut genug, d.h.: bleibt die durchschnittliche quadratische Abweichung unter einem Schwellwert und befinden sich auch die Endpunkte nahe an der gemeinsamen Approximationskurve, so werden die beiden Kanten für immer vereinigt. Ansonsten passen die Kurven nicht, und die Vereinigung wird nicht durchgeführt.

Normalerweise ist in einer Bildpyramide der Rechenaufwand für die Reduktion konstant und weder von der Ebene noch vom Inhalt der Zellen abhängig. In der hier verwendeten Pyramide ist der Aufwand aber offensichtlich vom Inhalt der Pixel, insbesondere von der Länge der Kantenliste, abhängig.

Bei der Vereinigung von zwei Zellen muß zu jeder Kante der ersten Liste nach einer Fortsetzung in der zweiten Liste gesucht werden. Das ergibt einen Rechenaufwand der Ordnung $O(N^2)$, wobei N das Maximum aus den Längen der beiden Kanten ist. Geht man bei der Vereinigung von vier Zellen so vor, daß man zunächst die beiden oberen Zellen, dann die beiden unteren und zuletzt die Ergebnisse daraus vereinigt, so kann der Rechenaufwand immer noch mit $O(N^2)$ abgeschätzt werden. N ist wieder das Maximum der Längen aller beteiligten Kanten.

Die Reduktion kann aber noch beschleunigt werden, indem man als Vorbereitung alle Kanten aussortiert, die nicht an den Rand ihrer Zelle stoßen. Sie können keine Fortsetzung in einer anderen Liste haben, da sie niemals die Bedingung des Zusammenstoßens erfüllen können. Diese Kanten können deshalb direkt in die Gesamtliste übernommen werden. Der

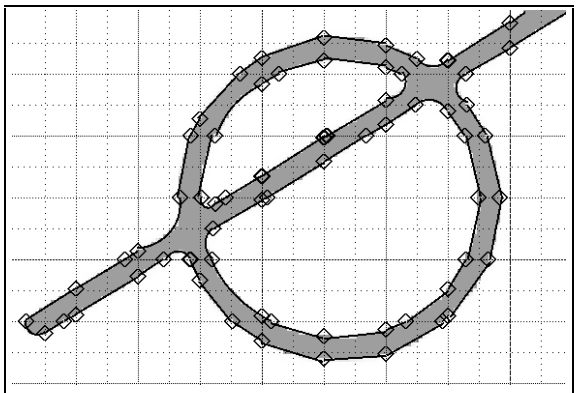

Abb.2.7 - Ebene 5 der Pyramide. Ein Pyramiden-Pixel ist hier 16*16 Pixel groß.

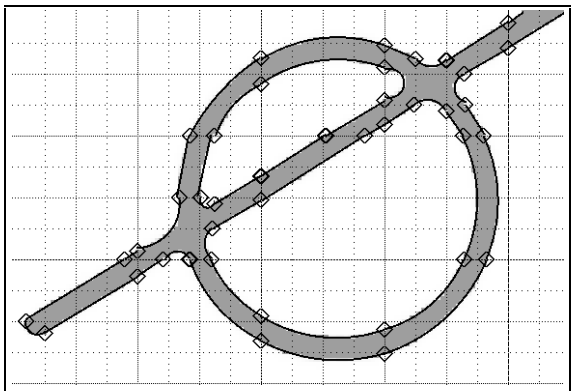

Abb.2.8 - Ebene 6 der Pyramide. Die Pyramiden-Pixel sind hier 32*32 Pixel groß.

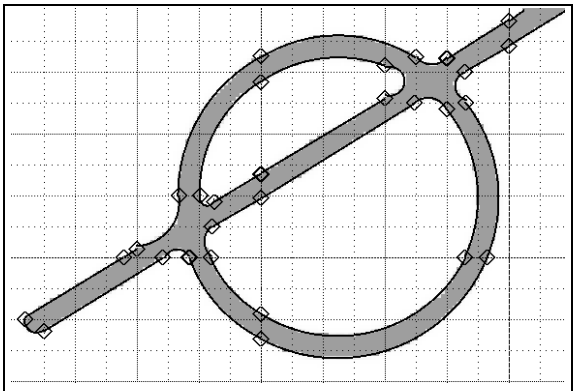

Abb.2.9 - Ebene 7 der Pyramide. Ein Pixel der Pyramide entspricht hier 64*64 Pixel im Bild.

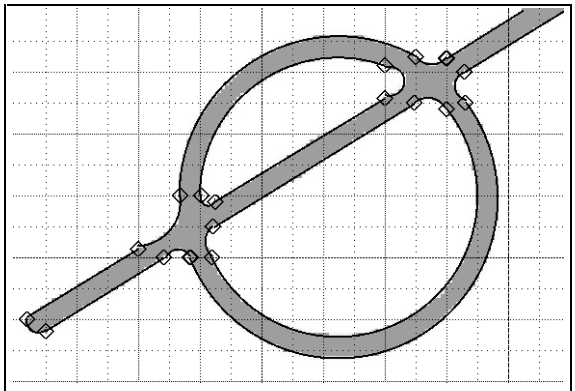

Abb.2.10 - Kanteninformation auf der neunten Ebene der Pyramide, dem Apex.

Aufwand für diese Vorbereitung beträgt O (N), dafür werden die Kantenlisten bei der Vereinigung kürzer.

Die Abbildungen 2.7-2.10 illustrieren an einem Beispiel die Funktionsweise der Pyramide. Abb.2.7 zeigt die Kanteninformation, die auf der fünften Ebene der Pyramide vorhanden ist. Das rezeptive Feld eines Pyramiden-Pixels ist hier 16*16 Pixel groß. Auf dieser Ebene kann deshalb keine Kante über eine 16-Pixel-Rasterlinie hinauslaufen. In der Abbildung erscheint die Kontur des Kreises etwas eckig. Der Grund dafür ist, daß alle Kanten immer zuerst mit einer Geraden approximiert werden. Nur wenn die Geradenapproximation zu schlecht ist, wird die Approximation mit einem Kreis probiert. Da auf der fünften Ebene die Geraden den Kreisbogen meistens noch gut genug annähern, sind hier die Kanten auf den Kreisbögen als Geraden eingezeichnet.

In Abb.2.8 und Abb.2.9 sind die sechste und siebente Ebene der Pyramide abgebildet. Abb.2.10 zeigt schließlich die höchste Ebene der Pyramide, den Apex. Er enthält die gesamte Kanteninformation.

Praktische Berechnung der Pyramide

In den vorigen Abschnitten wurde beschrieben, wie die Pyramide theoretisch aufgebaut ist. In der Praxis ist aber Vorsicht geboten. Würde man die Pyramide einfach Ebene für Ebene aufbauen, so wäre der Speicherverbrauch enorm. Für jedes einzelne Kantenpixel auf der ersten Ebene würde ja ein ganzer Akkumulator angelegt werden. Das darf auf gar keinen Fall geschehen. Es wurde aber schon mehrfach erwähnt, daß das Ergebnis der Pyramide der Apex ist. Der Rest der Pyramide ist nicht von Interesse. Es werden nun zwei Vorschläge gemacht, wie der Apex der Pyramide speicherschonend berechnet werden kann.

Version 1: Der folgende, in Pseudo-Code beschriebene, rekursive Algorithmus zur Berechnung eines Pyramiden-Pixels minimiert den Speicherverbrauch:

```
function Pixel (x, y, level): Kurvenliste
    if level = 0
    then Algorithmus für die erste Ebene
    else p1 = Pixel (x/2, y/2, level-1)
         p2 = Pixel (x/2 + 1, y/2, level-1)
         p3 = Pixel (x/2, y/2 + 1, level-1)
         p4 = Pixel (x/2 + 1, y/2 + 1, level-1)
         result = Reduziere (p1, p2, p3, p4)
```

Beim Aufruf von Reduziere kann der Speicherplatz für die Pixel p1 bis p4 schon wieder freigegeben werden, da Reduziere ihre Information in einem Pixel, d.h. in einer Kurvenliste, zusammenfaßt. Wird nun der Apex berechnet, so sind zu jedem Zeitpunkt auf jeder Ebene der Pyramide höchstens vier Pixel aktiv und höchstens vier Kurvenlisten angelegt. Der Speicherplatzverbrauch wird somit nicht von der Datenstruktur (der Pyramide) und ihrem Overhead, sondern allein vom Bildinhalt, d.h.: der Anzahl der Kanten, bestimmt.

Version 2: Nach der Betrachtung des State-of-the-Art wurde als besonders erstrebenswertes Ziel die zeilenweise Abarbeitung des Bildes genannt. Auch das kann erreicht werden:

1. Berechne Zeile 1 der Ebene 0 der Pyramide

2. Berechne Zeile 2 der Ebene 0 der Pyramide. Während dieser Berechnung werden je zwei Pixel mit den entsprechenden zwei Pixel aus Zeile 1 zu einem Element von Zeile 1 der Ebene 1 reduziert. Der Speicherplatz der Pixel auf Ebene 0 kann schon wieder freigegeben werden.

3. Berechne Zeile 3 der Ebene 0 und speichere sie ab.

4. Berechne Zeile 4 der Ebene 0. Mit Zeile 3 gemeinsam bildet sie Zeile 2 auf Ebene 1. Auf Ebene 1 werden jetzt die zwei ersten Zeilen zu Zeile 1 der Ebene 2 reduziert. Nach diesem Schritt ist nur noch Zeile 1 auf Ebene 2 im Speicher, der Rest ist bereits freigegeben.

5. u.s.w.

Dieser Algorithmus verbraucht mehr Speicherplatz als der vorige. Im schlimmsten Fall, wenn eine exakt horizontale Kante durch das Bild geht, wird bei der Berechnung einer Zeile für jeden Pixel eine eigene Kante angelegt. Da ein Akkumulator etwas mehr als 100 Byte groß ist, kann das schon ein wenig unangenehm sein. Abgesehen von Verfeinerungen, mit denen dieses Problem stark abgeschwächt werden kann, ist es ein sehr kurzfristiges Problem, da nach drei weiteren Zeilen 75% der Kanten schon wieder freigegeben werden. Dadurch verschlimmert sich das Problem kaum, wenn mehrere horizontale Kanten im Bild sind.

Außerdem arbeitet dieser Algorithmus das Bild Zeile für Zeile ab. Es besteht somit keine Notwendigkeit mehr, das gesamte Bild im Speicher zu behalten. Dieser Gewinn ist wesentlich größer als die kurzfristige Speicherverschwendung bei horizontalen Kanten. Dadurch ist auch die Verarbeitung von sehr großen Bildern nicht mehr im Bereich des Unmöglichen.

2.4 Elimination der Artefakte

Im Überblick wurde bereits das Problem der Shiftvarianz und der Artefakte angesprochen. Mit dem Wissen über die Pyramide sind nun diese Effekte auch erklärbar. Beide Effekte entstehen in den unteren Ebenen der Pyramide. Solange nämlich die Kanten aus sehr wenigen Punkten bestehen, werden bei der Reduktion alle aneinanderstoßenden Kanten auch vereinigt. Die Bedingung, daß Kanten auch zusammenpassen müssen, kann noch nicht greifen, da auch die Vereinigung noch so kurz ist, daß eine ausreichend gute Approximation durch einen Kreisbogen praktisch immer möglich ist.

Dieser Effekt verschwindet erst auf höheren Ebenen, wenn mehr Kantenpunkte bekannt sind und dadurch der Verlauf der Kante besser erkennbar ist. Die unmittelbare Folge davon ist, daß Endpunkte von Kanten immer in ihrer x oder y-Koordinate ein Vielfaches von acht haben. Somit ist das Ergebnis der Pyramide shiftvariant.

Eng damit verbunden ist auch die Ursache für Artefakte an Schnittpunkten von Linien. Wenn auf den untersten Ebenen der Pyramide zufällig ein paar Punkte beider Kanten zusammengefaßt werden - und das passiert fast immer -, dann enthält die Kantenliste des Bildes außer den beiden „echten“ Kanten auch noch dieses künstlich erzeugte Artefakt.

Die Elimination dieser Artefakte besteht aus zwei Schritten: zuerst müssen die Artefakte erkannt werden und anschließend gelöscht werden. Der einfachere Teil ist sicherlich das Löschen. Das Artefakt wird einfach aus der Liste der gefundenen Kanten entfernt, und die beiden Nachbarkanten werden bis zu ihrem Schnittpunkt verlängert. Dabei gehen alle Punkte, die im Akkumulator des Artefakts enthalten waren verloren. Sie können nachträglich nicht mehr den beiden Nachbarkanten hinzugefügt werden.

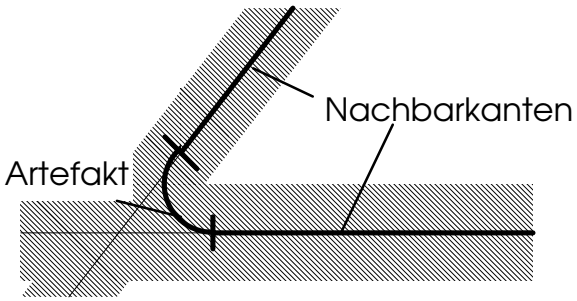

Abb.2.11 - eine Kante, die in einem Toleranzbereich, hier schraffiert eingezeichnet, um ihre beiden Nachbarkanten liegt, wird als Artefakt erkannt und eliminiert.

Etwas schwieriger ist die Erkennung der Artefakte. Dafür wurde folgende Bedingung implementiert:

> Artefakt-Bedingung: Eine Kante wird als Artefakt erkannt, wenn alle Punkte dieser Kante zu mindestens einer der beiden Nachbarkanten einen Abstand haben, der unter einer Toleranzgrenze liegt (Abb.2.11).

Für die Überprüfung, ob diese Bedingung auf eine Kante zutrifft, wird natürlich nicht der Abstand jedes einzelnen Punktes zu den beiden Nachbarkanten bestimmt. Das wäre ja auch gar nicht möglich, da von der Kante nur ihr Akkumulator bekannt ist. Die Lage der

einzelnen Punkte ist nicht mehr verfügbar. Deshalb wird bei der Überprüfung zu allen drei beteiligten Kanten (dem Artefakt und seinen beiden Nachbarn) die bestapproximierende Gerade bzw. Kreis bestimmt. Die Artefakt-Bedingung wird dann für die Approximationskurven ausgewertet.

Im Überblick wurde behauptet, daß das Entstehen und die Elimination der Artefakte letztendlich zu genaueren Ergebnissen führt. Die Begründung dafür soll nun anhand von Abb.2.12 gegeben werden.

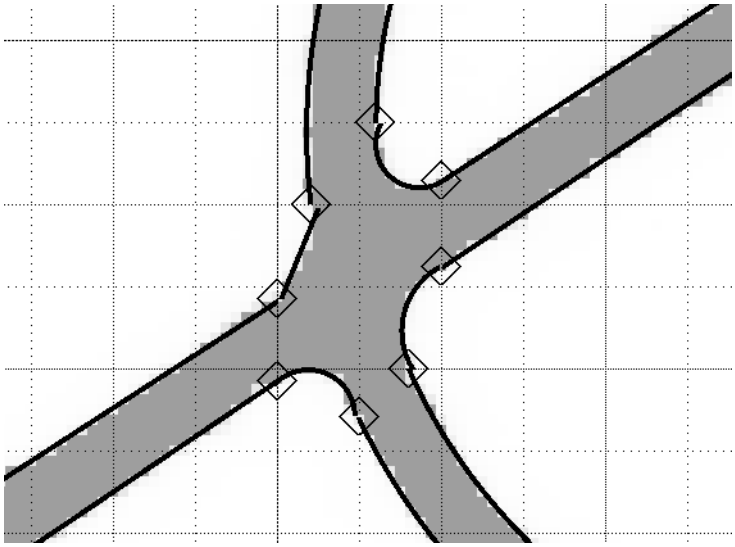

Abb.2.12 - Detailvergrößerung eines Schnittpunktes zweier Linien.

Abb.2.12 zeigt eine Detailvergrößerung eines Schnittpunktes im Testbild. Durch die starke Vergrößerung wird deutlich, daß die beiden Linien bereits vor dem Schnittpunkt verschmieren. Die Kante verläuft im Bild durch einen Bereich, der bei einem scharfen Bild eigentlich weiß sein sollte. Im Bereich des Schnittpunktes gibt es also etliche „falsche“ Kantenpunkte, die eigentlich keiner der beiden Kanten zuzuordnen sind. Egal welcher der beiden Kanten man diese Punkte auch zuordnet, sie haben einen relativ großen Abstand von der wahren Lage der Kante.

Dadurch ergeben sich zwei durchaus positive Eigenschaften der Artefakt-Elimination:

- Das Artefakt deckt diesen unscharfen Bereich ab und enthält die „falschen“ Kantenpunkte. Mit dem Artefakt werden auch all diese Punkte gelöscht. Sie sind dann in keinem Akkumulator mehr enthalten und können keinen (negativen) Einfluß mehr auf das Endergebnis haben.

- Der Schnittpunkt der beiden Kanten ist im Bild gar nicht enthalten. Dort, wo der Schnittpunkt in Wahrheit liegt, ist das Bild schwarz. Es ist deshalb sinnlos, den Schnittpunkt im Bild zu suchen. Durch die Artefakt-Elimination wird dieser Schnittpunkt

auf arithmetische Weise bestimmt, was sicherlich die einzige Möglichkeit ist, ein genaues Ergebnis zu erhalten.

2.5 Bestimmung des Verdrehungswinkels

Beim Einscannen eines Bildes ist es praktisch unmöglich, das Papier absolut gerade auf den Scanner zu legen. Dadurch ist das digitalisierte Bild immer leicht verdreht. Bei den meisten Anwendungen hat man einerseits keine Möglichkeit, das Ausmaß dieser Verdrehung zu bestimmen, da das digitalisierte Bild diese Information nicht enthält, andererseits verursacht diese Störung auch keine großen Probleme.

Ganz anders ist die Situation bei der LATEX-Bilderkennung. LATEX erlaubt nur wenige diskrete Steigungswinkel für Geraden (vgl. Abb.2.3). Da sich sowohl der Input (die Zeichnung) als auch der Output (die Beschreibung) an die LATEX Syntax halten, ergeben sich folgende zwei Überlegungen:

- Die Verdrehung ist nicht ignorierbar: Vernachlässigt man einfach die Verdrehung, so hat keine der langen Geraden einen gültigen Anstieg. Der Anstieg einer langen Geraden ist nämlich aus dem Akkumulator mit den statistischen Methoden sehr genau berechenbar. Er wird deshalb immer um den Verdrehungswinkel vom gültigen und korrekten Anstieg abweichen. Daraus ergibt sich die Notwendigkeit, den Verdrehungswinkel zu bestimmen.

- Die Verdrehung ist bestimmbar: Da nur wenige Anstiege erlaubt sind, enthält das digitalisierte Bild genügend Information, um den Verdrehungswinkel zu berechnen. Die Grundidee für die Bestimmung des Verdrehungswinkels besteht darin, jenen Winkel zu finden, bei dem möglichst viele Geraden, besonders die langen, einen gültigen Anstieg haben.

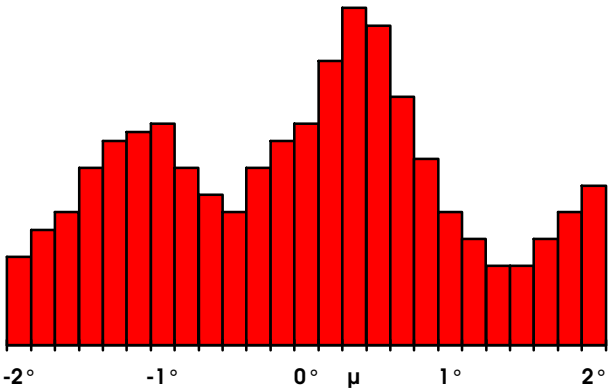

Abb.2.13 - Histogramm zur Bestimmung des Verdrehungswinkels μ

Um diesen Winkel zu finden, wird ein Histogramm erstellt (Abb.2.13). Zu einigen diskreten Winkeln wird ein Maß dafür berechnet, wie gut dieser Winkel als Verdrehungswinkel geeignet ist. Der Winkel mit der höchsten Maßzahl wird dann als Verdrehungswinkel angenommen.

Für die Bestimmung der Maßzahl werden in einem Durchlauf alle Geraden des Bildes abgearbeitet. Zu jeder Geraden werden alle Verdrehungswinkel bestimmt, für die diese Gerade einen gültigen Anstieg hat. Die Gerade leistet dann für all diese Winkel einen Beitrag im Histogramm, der von der Länge der Geraden abhängt. z.B.: Anzahl der Kantenpunkte / Anzahl der passenden Verdrehungswinkel. Lange Geraden nehmen deshalb größeren Einfluß auf das Ergebnis, da ihr Anstieg genauer bekannt ist.

Da die gültigen Anstiege unregelmäßig verteilt sind, ist mit dieser Methode der Verdrehungswinkel bis auf ein Vielfaches von 90° bestimmbar, sofern im Bild Geraden mit vielen verschiedenen Anstiegen vorkommen.

2.6. Kantenfolgen

Im Zwischenergebnis, das wir nach der Elimination der Artefakte erhalten, werden alle Kanten von dicht aneinander liegenden Punkten gebildet. Da bei Überschneidungen von Linien die Kanten unterbrochen sind, enthält das Zwischenergebnis je eine Kanten vor und nach dem Schnittpunkt. Ziel dieses Schrittes ist es, mehrere Kanten, die durch eine Kurve approximierbar, aber unterbrochen sind, zu einer Kantenfolge zu vereinigen.

Das ist mit den Methoden, die bereits in der Pyramide zur Anwendung kamen, gar nicht mehr schwierig durchzuführen. Der Datentyp einer Kantenfolge besteht aus einem Akkumulator und einer Kantenliste. Der Akkumulator wird als Summe aus den Akkumulatoren der einzelnen Kanten berechnet. Aus ihm kann im weiteren Verlauf die Lage der Kantenfolge noch genauer bestimmt werden, da nun noch mehr Kantenpunkte bekannt sind. Die Kantenliste ist eine doppelt verkettete Liste, in der jede Kante einen Zeiger auf ihren Vorgänger und Nachfolger in der Kantenfolge enthält. Dadurch bleibt die Information erhalten, wo wirklich eine Kante sichtbar ist, und wo die Kantenfolge unterbrochen ist.

Der Algorithmus für das Zusammenfassen zu Kantenfolgen ist am leichtesten in Pseudocode Schreibweise verständlich:

```
für alle Kanten K
    für alle bereits angelegten Kantenfolgen F
        A = gemeinsame Approximationskurve von K und F
        if der durchschnittliche quadratische Fehler bei Approximation mit A
        sowohl von K als auch  von F unter einem Schwellwert ist
        then füge K in F ein und gehe zur nächsten Kante
    lege neue Kantenfolge mit K als einziger Kante an
```

Das Ergebnis dieses Schrittes wurde bereits im Überblick anhand von Abb.2.4 veranschaulicht.

2.7. Linienanalyse

Auch nach der Suche nach Kantenfolgen werden die *Linien*-Objekte in der Zeichnung durch *Kanten* beschrieben. Ziel der Linienanalyse ist es, den Übergang von der kantenorientierten Beschreibung des Bildes zu einer linienorientierten Beschreibung zu bewältigen. Das einzige LATEX-Primitiv, das kein Linienobjekt ist, ist der ausgefüllte Kreis. Er ist damit auch das einzige Objekt, das im Zwischenergebnis aus nur einer Kantenfolge besteht. Alle anderen Objekte sind hier noch durch zwei Kantenfolgen repräsentiert.

Bei der Linienanalyse werden zunächst gegenüberliegende Kantenpaare gesucht. Geradenpaare zeichnen sich dadurch aus, daß beide Geraden denselben Anstieg haben, aber entgegengesetzt orientiert sind und nahe beieinander liegen. Für die Orientierung gilt die Konvention, daß Kanten immer in der Richtung durchlaufen werden, daß links die schwarzen und rechts die weißen Pixel liegen.

Kreispaare werden daran erkannt, daß beide Kreisfolgen denselben Mittelpunkt haben und der äußere Bogen, der gegen den Uhrzeigersinn durchlaufen wird, einen etwas größeren Radius hat als der innere Bogen.

Für jedes Kantenpaar werden nun alle Unterbrechungen genauer betrachtet. Für jede Unterbrechung muß geklärt werden, ob nur die Kante unterbrochen ist, die Linie aber durchgezogen werden darf, oder ob die Linie unterbrochen ist.

Abb.2.14a zeigt den typischen Fall für eine Überschneidung. Beide Nachbarkanten laufen nach außen weg, und deren Gegenüber sind die Nachbarkanten der Fortsetzung der Linie. Die Situation wird somit als Linienüberschneidung identifiziert, und die Linie wird durchgezogen.

In Abb.2.14b kreuzt die Nachbarkante N1 die Linie. Sie zeigt damit an, daß in diesem Fall die Linie, und nicht nur die Kante, unterbrochen ist. Im Fall c) laufen zwar beide Nachbarkanten nach außen weg, aber die gegenüberliegende Kante schneidet die Linie und zeigt damit eine Linienunterbrechung an. Die Linie muß in diesem Fall zwischen ihrem

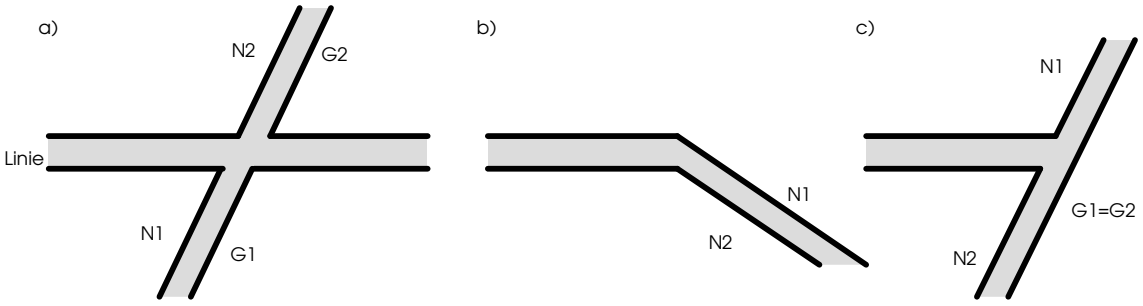

Abb.2.14 - Die drei häufigsten Arten von Kantenunterbrechungen. N1 und N2 sind die Nachbarkanten des gerade betrachteten Linienabschnitts, und G1 und G2 deren Gegenüber. Einzig im Fall (a) darf die Linie durchgezogen werden. In den beiden anderen Fällen wird ein Ende der Linie erkannt.

Schnittpunkt mit den Nachbarkanten und dem Schnittpunkt mit deren Gegenüber enden.

Mit diesen Kriterien ist die Art der Unterbrechung meistens erkennbar. In seltenen Fällen, wie in Abb.2.15 dargestellt, kann aber auch ein größerer Suchaufwand notwendig werden. Laufen beide Nachbarn nach außen weg und ist deren Gegenüber nicht bekannt, so muß nach einer Kante gesucht werden, die die Verlängerung der Linie noch vor ihrer Fortsetzung schneidet und ein Ende der Linie markiert. Da man keinen Anhaltspunkt hat, welche Kante dafür in Frage kommt, müssen alle Kanten betrachtet werden.

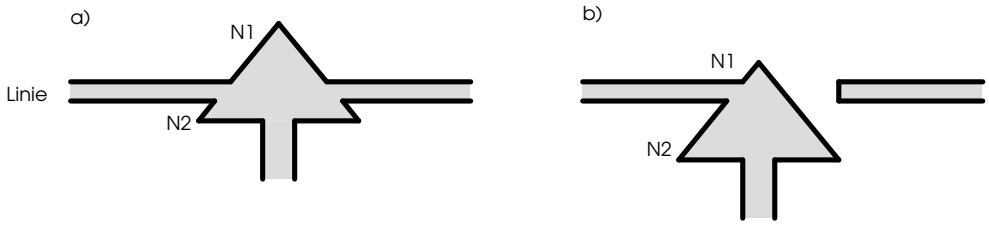

Abb.2.15 - Da bei Pfeilspitzen kein Gegenüber definiert ist, kann eine Suche über alle Kanten notwendig werden, um eine Kantenunterbrechung (a) von einer Linienunterbrechung (b) unterscheiden zu können.

Wird eine Unterbrechung bzw. ein Ende der Linie erkannt, so wird auch noch geprüft, ob sich am Ende der Linie eine Pfeilspitze befindet. Der dafür verwendete Algorithmus ist sehr ähnlich der Vektorerkennung aus [Lu88]. Auch dort wird versucht, aus der Beschreibung des Kantenverlaufs Vektoren zu erkennen. Die Problematik bei der Vektorerkennung besteht hauptsächlich darin, daß kaum eine Aussage über das Aussehen der Pfeilspitzen in der Kantenbeschreibung gemacht werden kann. In Abb.2.16 ist eine Seite der Pfeilspitze durch zwei Geraden approximiert, während die andere Seite mit zwei Kreisbögen und einem sehr kurzen Geradenstück beschrieben ist. Das soll veranschaulichen, wie stark hier die Beschreibung vom Zufall bzw. vom Pixelraster abhängt.

Vektorerkennung: Falls zwischen dem Ende der Kanten und dem Ende der Linie, das ist der Schnittpunkt mit einer kreuzenden Kante, genug Platz für eine Pfeilspitze ist, wird die Hypothese aufgestellt, daß eine Pfeilspitze vorhanden ist. Dann werden, ausgehend von den Kanten der Linie, die Nachbarkanten verfolgt. Solange sie nahe an der Linie liegen, werden sie als Kanten der Pfeilspitze angesehen. Existieren auf beiden Seiten mindestens zwei Kanten der Pfeilspitze, so wird ein Vektor erkannt.

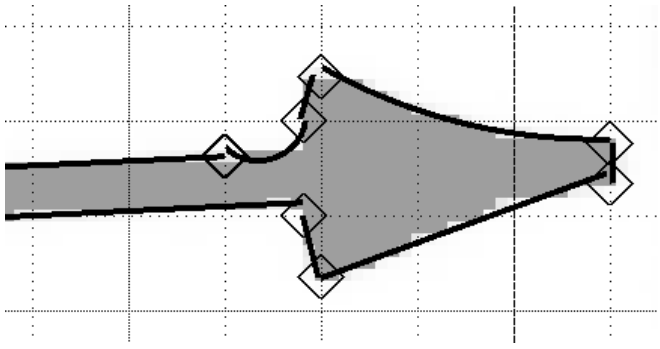

Abb.2.16 - Kantenbeschreibung bei Pfeilspitzen

Erkennung von Rechtecken und Ovalen:

Nach der Linienanalyse besteht die Beschreibung des Bildes aus Linien-Elementen, die, direkt in ein LATEX-Dokument geschrieben, eine richtige Beschreibung der Zeichnung ergäben. Es wird aber eine möglichst kurze und einfache Beschreibung angestrebt. Deshalb sollen aus Geraden und Kreisbögen noch Rechtecke und Ovale gebildet werden.

Der Erkennung von Rechtecken wurde folgende Definition zugrunde gelegt:

> Definition Rechteck: Ein Rechteck besteht aus vier Geraden, wobei jede Gerade im Endpunkt der vorigen Gerade beginnt, die Geraden abwechselnd horizontal und vertikal sind und die vier Geraden eine geschlossene Figur bilden.

Die Rechteckerkennung wird mit einer beliebigen horizontalen Gerade angestoßen. Ausgehend von dieser ersten Rechteckseite werden die drei übrigen Seiten gesucht. Jede Seite muß Bedingungen erfüllen, die sich aus der Definition und den bereits gefundenen Seiten ergeben. So muß die zweite Seite vertikal sein und einen Endpunkt mit der ersten Gerade gemeinsam haben. Da diese zwei Seiten die Lage des Rechtecks eindeutig definieren, können nun die Endpunkte der dritten und vierten Seite berechnet werden. Werden auch diese beiden Seiten in der Datenstruktur gefunden, so können die vier Geraden zu einem Rechteck vereinigt werden.

Ganz ähnlich verläuft das Zusammenbauen von Ovalen. Der Unterschied besteht darin, daß zwischen je zwei Geraden noch ein Kreisbogen liegen muß. Außerdem muß ein Oval nicht geschlossen sein. Ein Oval muß auch dann generiert werden, wenn nur eine Hälfte oder ein Viertel vorhanden ist. Die LATEX-Syntax erlaubt die entsprechenden Einschränkungen (vgl. Kap.1.3).

2.8. Multiput und Ausgabe

Bei der Ausgabe der Bildbeschreibung wird versucht, gleichartige Objekte, etwa Geraden gleichen Anstiegs und gleicher Länge oder Kreise mit gleichem Radius, mit einer einzigen *multiput* Anweisung auszugeben. Dazu wird jede Gruppe gleicher Objekte getrennt betrachtet. Zu jedem Paar aus einer Gruppe wird nach einem dritten, vierten, usw. Objekt gesucht, das mit diesem Paar in ein *multiput* zusammengefaßt werden kann. Werden mindestens drei passende Objekte gefunden, so werden sie gemeinsam zu einem multiput zusammengefaßt.

Zuletzt wird die intern gefundene Beschreibung des Bildes in LATEX-Syntax ausgegeben. Das Ergebnis sollte mit der Originalbeschreibung, mit der die Zeichnung erstellt wurde, weitgehend übereinstimmen. Trotzdem werden wir im nächsten Kapitel sehen, daß auf den ersten Blick die beiden Beschreibungen wenig Ähnlichkeit miteinander zu haben scheinen. Das liegt in erster Linie daran, daß der Ursprung der Zeichnung und die, der Beschreibung zugrunde liegende, Längeneinheit frei gewählt werden müssen. Dadurch sind alle Angaben über Größe und Lage der Objekte durch einen konstanten Faktor, nämlich den Quotient aus

den Längeneinheiten beider Beschreibungen, verändert. Die Angaben über die Lage der Objekte ist außerdem noch um einen Translationsvektor verschoben.

Nach dem Ausdrucken der erzeugten Beschreibung, oder auch nach einer rechnerischen Bereinigung dieser beiden scheinbaren Störungen, wird aber erkennbar, daß die wirklichen Abweichungen in der Größenordnung von Zehntelmillimetern liegen. Eine genauere Analyse der Ergebnisse wird im nächsten Kapitel gegeben.

3. Ergebnisse

Der in Kapitel 2 vorgestellte Algorithmus wurde auf dem PC implementiert. In diesem Kapitel wird von einigen experimentellen Ergebnissen berichtet. Alle Angaben über die Rechenzeit beziehen sich auf einen PC 486DX2 / 66 MHz.

3.1 Testzeichnung 1

In diesem Abschnitt wird die Qualität der Ergebnisse der LATEX-Bilderkennung anhand von Testzeichnung 1 (Abb.3.1) untersucht. Das Interesse gilt dabei hauptsächlich der Genauigkeit, der Rechenzeit und dem Speicherverbrauch.

Testzeichnung 1 wurde mit folgender LATEX-Beschreibung ausgedruckt.

```
\unitlength 0.7mm
\begin {picture} (150, 100)
\thicklines
\put ( 5, 15) {\line (4, 3) {90}}
\multiput (15, 20)(20, 18){4} {\circle {14}}
\put (85, 25) {\oval (60, 30)}
\put (20, 69) {\line (6, -5) {80}}
\put (20, 69) {\circle*{4}}
\end{picture}
```

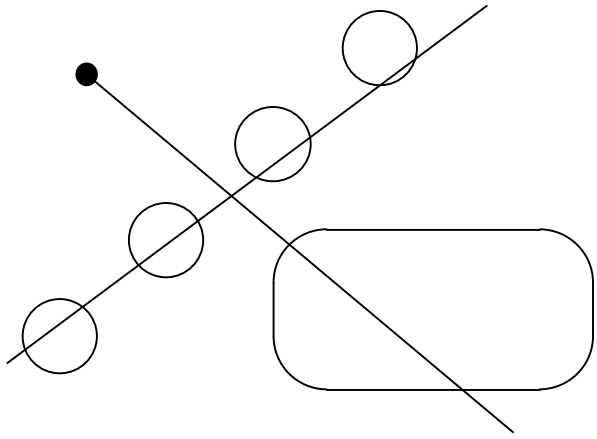

Abb.3.1 - Testzeichnung 1

Die Zeichnung wurde mit verschiedenen Auflösungen eingescannt. Dazu wurde ein Hewlett Packard ScanJet IIc und die Standardsoftware „Photo Styler“ verwendet. Die folgende Tabelle faßt einige wichtige Daten über die LATEX-Bilderkennung zusammen:

Auflösung	*200 dpi*	*300 dpi*	*400 dpi*	*500 dpi*	*600 dpi*	*700 dpi*
Bildgröße in Pixel	640*472 (302 kB)	959*708 (679 kB)	1279*944 (1207 kB)	1599*1180 (1887 kB)	1918*1416 (2716 kB)	2238*1652 (3697 kB)
Pyramide: Rechenzeit	27.08 s	49.95 s	79.34 s	113.58 s	155.66 s	206.98 s
Kanten im Apex	125	128	123	127	132	131
Maximum beim Aufbau	367	444	480	440	524	481
Artefakt-Elimination: Rechenzeit	0.11 s	0.05 s	0.05 s	0.11 s	0.11 s	0.11 s
Anzahl der Kanten	75	74	74	76	77	78
Verdrehungswinkel	0.05 s	0.05 s	0.05 s	0.11 s	0.05 s	0.11 s
Kantenfolgen: Rechenzeit	0.01 s	0.05 s	0.05 s	0.01 s	0.01 s	0.01 s
Anzahl der Folgen	32	33	34	38	38	37
Linienanalyse	0.77 s	0.66 s	0.66 s	0.77 s	0.93 s	0.82 s
Multiput	0.01 s	0.01 s	0.01 s	0.05 s	0.01 s	0.05 s
Gesamt	28.03 s	50.77 s	80.16 s	114.63 s	156.77 s	208.09 s

Tabelle 3.1 - Rechenaufwand und Speicherverbrauch bei der Erkennung von Testzeichnung 1 mit verschiedenen Auflösungen. Es wurde ein PC 486DX2 mit 66MHz verwendet.

Genauigkeit der Ergebnisse

Das sicherlich wichtigste Qualitätsmerkmal bei der Erkennung technischer Zeichnungen ist die Korrektheit und die Genauigkeit der Beschreibung. Sie geht aber aus der Tabelle nicht hervor. Es soll deshalb in diesem Abschnitt die Genauigkeit der Ergebnisse anhand des Testbildes mit der Auflösung von 300dpi untersucht werden. Die LATEX-Bilderkennung liefert folgende Beschreibung:

```
\documentstyle {article}
\begin {document}
\unitlength 1.0mm
\begin {picture}( 85,  64)
\thicklines
\put (15.37, 48.25) {\line (6, -5) {55.10}}
\put (67.06, 58.70) {\line (-4, -3) {63.35}}
\put (14.33, 49.20) {\circle*{2.80}}
\multiput (52.94, 52.73)(-14.05,-12.67){4} {\circle{9.86}}
\put (59.94, 18.32) {\oval (42.12, 21.09)}
\end {picture}
\end {document}
```

Zunächst erkennen wir, daß alle Elemente der Zeichnung gefunden wurden: Die Beschreibung enthält zwei Geraden (mit dem richtigen Anstieg), einen ausgefüllten Kreis, vier Kreise, die durch eine multiput-Anweisung beschrieben sind und ein Oval. Größen, Längenangaben und Positionen stimmen aber nicht mit denen des Originaldokuments überein. Die beiden Gründe dafür wurden bereits am Ende von Kapitel 2 vorweggenommen:

1. Die Längeneinheit, in Zeile 3 mit der Anweisung *unitlength* spezifiziert, kann aus dem Bild nicht bestimmt werden. Sie muß deshalb willkürlich festgelegt werden. Sie wurde auf 1 mm gesetzt, war aber in der Originalbeschreibung 0.7 mm. Deshalb sind alle Wertangaben um 30% kleiner als im Original.

2. Der Ursprung des Bildes kann nicht gefunden werden. Auch er muß willkürlich festgelegt werden. Die Folge davon ist, daß alle Positionsangaben zusätzlich noch um einen konstanten Translationsvektor gegenüber der Originalposition verschoben sind.

Um nun die Genauigkeit der Erkennung zu untersuchen, wird das Ergebnis rechnerisch um diese beiden „Fehler“ bereinigt, d.h.: das Ergebnis wird mit der Längeneinheit 0.7mm ausgegeben und der Ursprung um (0.5, 1.0) verschoben. Außerdem wird noch die Orientierung der zweiten Geraden und des multiputs geändert. Daraus ergibt sich folgende, äquivalente Beschreibung:

```
\documentstyle {article}
\begin {document}
\unitlength 0.7mm
\begin {picture}( 121,  91)
\thicklines
```

```
\put (21.46, 67.93) {\line (6, -5) {78.71}}
\put ( 4.80, 14.98) {\line (4, 3) {90.50}}
\put (19.97, 69.29) {\circle*{4.00}}
\multiput (14.92, 20.03)(20.07, 18.10){4} {\circle{14.09}}
\put (85.13, 25.17) {\oval (60.17, 30.13)}
\end {picture}
\end {document}
```

Nun ist die Beschreibung dem Original schon sehr ähnlich. Die Abweichungen bewegen sich in der Größenordnung von 0.2 Einheiten, also 0.1 - 0.2 mm. Nur der Startpunkt der ersten Gerade, und deshalb auch die Länge dieser Gerade, wurde deutlich ungenauer erkannt. Er liegt mehr als 1 mm neben seiner ursprünglichen Position. Auch das ist aber kein Fehler der Erkennung. Dieser Startpunkt ist nämlich von dem ausgefüllten Kreis verdeckt. Deshalb darf der Startpunkt beliebig im Kreis gewählt werden.

Auch aus den mit 200 - 700 dpi eingescannten Bildern berechnet die Bilderkennung im wesentlichen dasselbe Ergebnis. Bei noch niedrigerer Auflösung können aber die Linien zerreißen, wodurch es für die Bilderkennung unmöglich wird, sinnvolle Ergebnisse zu liefern.

Rechenzeit

Die Rechenzeiten für die einzelnen Schritte des Algorithmus sind in Tab.3.1 angegeben. Es fällt sofort auf, daß fast die gesamte Rechenzeit für den Aufbau der Pyramide benötigt wird. Der Zeitverbrauch der übrigen Schritte ist so gering, daß die Messungen schon sehr ungenau werden.

Ein großer Vorteil bei der Verwendung von Pyramiden ist die einfache Parallelisierbarkeit. Da dieser Algorithmus fast die gesamte Rechenzeit zum Aufbau der Pyramide benötigt, ist der vorgestellte Algorithmus leicht für Parallelrechner adaptierbar.

Speicherverbrauch

Hier muß nun zunächst erwähnt werden, daß beim Aufbau der Pyramide Version 2 (vgl. Kap. 2.3) verwendet wurde, die das Bild zeilenweise abarbeitet und dafür etwas Speicherplatz vergeudet.

Der Speicherverbrauch ist in Tab.3.1 aus den Angaben über die Anzahl der Kanten erkennbar. In einer Zeile der Tabelle („Maximum beim Aufbau“) ist angegeben, wieviele Kanten während dem Aufbau der Pyramide höchstens gleichzeitig im Speicher waren. Dieser Spitzenverbrauch entsteht meistens beim Erreichen einer horizontalen Geraden. Die größten Anforderungen an den Speicher stellt demnach das 600dpi-Bild. Bei diesem Bild müssen 524 Kanten angelegt werden. Da eine Kante 172 Byte verbraucht, ergibt das einen Speicherverbrauch von 90 kB für die Kanteninformation. Dazu kommen dann noch 8 kB für die Datenstruktur der Pyramide und 4 kB für zwei Zeilen des Bildes, die im Speicher sein müssen.

Der Algorithmus benötigt somit nur gut 100 kB! - Das Bild muß ja nicht im Speicher sein. Im Gegensatz dazu braucht jeder Algorithmus, der das ganze Bild im Speicher halten muß, allein 339 kB für das Bild (600 dpi) - bei der Verwendung von Binärbildern.

Toleranzparameter

In Kap. 2.2 und 2.3 wurde ein Schwellwert eingeführt, der als Vergleichswert für den durchschnittlichen quadratischen Approximationsfehler verwendet wird, um zu entscheiden, ob die Approximation gut genug ist. Davon hängt vor allem ab, ob zwei Kanten in der Pyramide vereinigt werden. In diesem Abschnitt wird überprüft, welche Auswirkungen ein zu hoch bzw. zu niedrig gewählter Schwellwert auf das Ergebnis und auf die Performance des Algorithmus hat.

Von der Theorie aus Kap. 2.3 her sind die unmittelbaren Auswirkungen eines zu niedrig gewählten Schwellwertes klar: In der Pyramide werden viele Kanten, die eigentlich zusammenpassen, aufgrund kleiner Störungen nicht vereinigt. In der Bildbeschreibung im Apex der Pyramide sind lange Kanten in mehrere kurze Stücke zerrissen.

Bei Verwendung eines zu hohen Schwellwertes werden die Artefakte recht lang und die „echten“ Kanten kürzer. An manchen kritischen Stellen des Bildes kann das dazu führen, daß nicht mehr zwischen Artefakten und „echten“ Kanten unterschieden werden kann.

Tabelle 3.2 faßt eine Testserie mit verschiedenen Toleranzgrenzen bei Testbild 1, 300 dpi

Toleranz	*Kanten im Apex*	*Maximum beim Aufbau*	*Kanten nach Elimination*	*Ergebnis*
0.10	296	611	208	richtig
0.15	156	480	116	richtig
0.20	128	462	94	richtig
0.25	123	455	92	richtig
0.30	121	451	92	richtig
0.35	114	442	85	richtig
0.40	112	440	80	richtig
0.45	112	436	81	richtig
0.50	113	437	78	richtig
0.60	112	433	75	richtig
0.70	112	431	73	Oval zerreißt

Tab. 3.2 - Analyse von Testbild 1 mit verschiedenen Toleranzgrenzen

zusammen. Bei den extrem kleinen Werten, 0.10 und 0.15, werden sehr viele Kanten gefunden, d.h.: die Geraden und Kreise sind in mehrere Teile zerstückelt. Daß das Ergebnis trotzdem richtig ist (es wurde jedes Objekt als *ein* Objekt, und nicht in mehreren Einzelteilen, beschrieben), ist der Stabilität der übrigen Rechenschritte zu verdanken. Es ist aber trotzdem nicht gut, wenn derart viele Kanten gefunden werden, da das sowohl Speicherplatz als auch Rechenzeit kostet. Außerdem können die nachfolgenden Schritte nicht immer diese Fehler ausgleichen (vgl. Testzeichnung 3).

Der Schwellwert 0.7 ist zu hoch. Bei diesem Wert ist die Situation beim Schnittpunkt der Geraden mit dem Kreisbogen des Ovals nicht mehr eindeutig erkennbar. Da bei dem Übergang eines Kreisbogens in seine Tangente ohnehin sehr große Artefakte entstehen können, ist diese Stelle besonders anfällig gegen einen zu hohen Schwellwert. Vom echten Kreisbogen bleibt ein zu kleiner Teil übrig, um die Lage des Kreisbogens genau zu bestimmen und das Artefakt zu eliminieren.

Zwischen den zu kleinen und den zu großen Werten bleibt ein recht großes Intervall [0.2, 0.6] mit vernünftigen Schwellwerten übrig. Ein weiterer Test zu diesem Schwellwert wird bei Testzeichnung 3 durchgeführt.

3.2 Testzeichnung 2

Die LATEX-Bilderkennung soll noch mit einem zweiten Testbild (Abb.3.2) getestet werden. Die LATEX-Beschreibung dieses Bildes lautet:

```
\unitlength 0.7mm
\begin {picture} (150,100)
\thicklines
\put ( 10, 50) {\circle {10}}
\put (140, 50) {\circle {10}}
\put ( 60,  5) {\framebox (30, 10){}}
\put ( 60, 85) {\framebox (30, 10){}}
\put ( 20, 44) {\oval (20, 68) [bl]}
\put ( 20, 10) {\vector (1, 0) {39}}
\put ( 91, 20) {\oval (98, 20) [br]}
\put (140, 20) {\vector (0, 1) {24}}
\put (130, 56) {\oval (20, 68) [tr]}
\put (130, 90) {\vector (-1, 0) {39}}
\put ( 59, 80) {\oval (98, 20) [tl]}
\put ( 10, 80) {\vector (0, -1) {24}}
\end {picture}
```

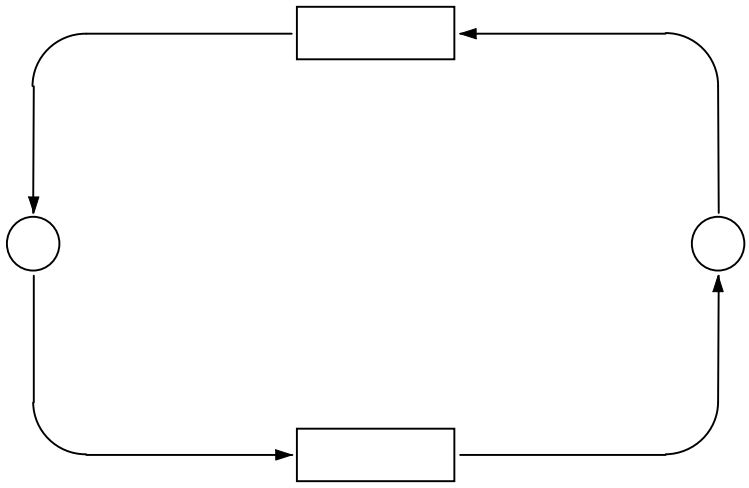

Abb.3.2 - Testzeichnung 2

Die Kenndaten der LATEX-Bilderkennung sind wieder in einer Tabelle (Tab.3.3) aufgelistet. Diesmal wurden aber die Angaben über die Rechenzeit der einzelnen Arbeitsschritte, ausgenommen den Aufbau der Pyramide, weggelassen.

Auflösung	*200 dpi*	*300 dpi*	*400 dpi*	*500 dpi*	*600 dpi*	*700 dpi*
Bildgröße in Pixel	812*552 (448 kB)	1218*828 (1009 kB)	1624*1104 (1793 kB)	2030*1380 (2801 kB)	2436*1656 (4034 kB)	2842*1932 (5491 kB)
Pyramide: Rechenzeit	34.89 s	65.17 s	106.15 s	157.75 s	219.45 s	293.25 s
Kanten im Apex	99	101	103	105	104	107
Maximum beim Aufbau	367	391	410	407	432	427
Anzahl der Kanten nach Elimination	69	70	72	73	71	74
Anzahl der Folgen	53	51	58	60	58	61
Gesamt-Rechenzeit	36.21 s	66.48 s	107.42 s	158.96 s	220.66 s	294.51 s

Tabelle 3.3 - Rechenaufwand und Speicherverbrauch bei der Erkennung von Testzeichnung 2 bei verschiedenen Auflösungen. Es wurde ein PC 486DX2 mit 66MHz verwendet.

Für eine Betrachtung des Ergebnisses der Bilderkennung wird diesmal das 700 dpi Bild herangezogen.

```
\documentstyle {article}
\begin {document}
\unitlength 1.0mm
\begin {picture}(108,  76)
\thicklines
\put (14.19, 10.76) {\vector (1, 0) {29.31}}
\put (9.27, 60.42) {\vector (0, -1) {17.34}}
\put (94.68, 66.88) {\vector (-1, 0) {28.28}}
\put (100.61, 17.09) {\vector (0, 1) {17.25}}
\put (44.35, 7.39) {\framebox (21.04, 7.13) {}}
\put (44.50, 63.53) {\framebox (21.04, 7.18) {}}
\put (100.62, 38.73) {\circle {7.05}}
\put (9.27, 38.74) {\circle {7.05}}
\put (94.30, 43.02) {\oval (12.69, 47.66) [tr]}
\put (66.07, 17.11) {\oval (68.97, 12.78) [br]}
\put (15.93, 34.55) {\oval (13.40, 47.68) [bl]}
\put (43.76, 60.42) {\oval (68.88, 12.86) [tl]}
\end {picture}
\end {document}
```

Verwendet man nun 0.7mm als Längeneinheit, und verschiebt man den Ursprung um (3.5, 5.6), so erhält man:

```
\documentstyle {article}
\begin {document}
\unitlength 0.7mm
\begin {picture}(154,  109)
\thicklines
\put (16.77, 9.77) {\vector (1, 0) {41.87}}
\put (9.74, 80.71) {\vector (0, -1) {24.77}}
\put (131.76, 89.94) {\vector (-1, 0) {40.40}}
\put (140.23, 18.81) {\vector (0, 1) {24.50}}
\put (59.86, 4.96) {\framebox (30.06, 10.19) {}}
\put (60.07, 85.16) {\framebox (30.06, 10.26) {}}
\put (140.24, 49.73) {\circle {10.07}}
\put (9.74, 49.74) {\circle {10.07}}
\put (131.21, 55.86) {\oval (18.13, 68.09) [tr]}
\put (90.89, 18.84) {\oval (98.53, 18.26) [br]}
\put (19.26, 43.76) {\oval (19.14, 68.11) [bl]}
\put (59.01, 80.71) {\oval (98.40, 18.37) [tl]}
\end {picture}
\end {document}
```

Druckt man dieses Dokument wieder aus und vergleicht es mit dem Original, so wird man kaum einen Unterschied erkennen können. Beim Vergleich der Beschreibungen fallen jedoch einige große Abweichungen auf: Erstens stimmen Anfangspunkt und Länge der Vektoren nicht, zweitens weicht die Größe der Ovale recht stark von der wirklichen Größe ab.

Beide Ungenauigkeiten haben eine gemeinsame Ursache: Der Zusammenstoßpunkt des Kreisbogens mit dem Vektor ist nicht so genau bestimmbar. Er wird aus dem Radius des Kreisbogens und der Bedingung, daß es in LATEX nur Viertelkreise gibt, bestimmt. Da aber nur ein Viertel-Kreisbogen sichtbar ist, ist die Bestimmung des Radius nicht so genau wie gewohnt. Damit ist die Größe des Ovals und die Lage des Startpunktes des Pfeils ebenso ungenau.

Trotzdem paßt die Beschreibung recht gut zur Zeichnung. Die maximale Distanz zwischen der Beschreibung und der Zeichnung bleibt trotz der scheinbar großen Abweichung gering.

3.3 Testzeichnung 3

Zuletzt soll die LATEX-Bilderkennung noch mit einem Bild getestet werden, das aus etwas mehr Elementen besteht. Testbild 3 (Abb.3.3) ist ein Graph, der mit folgendem LATEX-Dokument erstellt wurde:

```
\documentstyle {article}
\begin {document}
\unitlength=1.30mm
\begin{picture}(48.00,41.00)
\put(23.00,3.00){\circle*{2.00}}
\put(43.00,43.00){\circle*{2.00}}
\put(3.00,43.00){\circle*{2.00}}
\put(43.00,33.00){\circle{2.00}}
\put(23.00,23.00){\circle*{2.00}}
\put(3.00,33.00){\circle{2.00}}
\put(23.00,43.00){\circle{2.00}}
\put(23.00,13.00){\circle{2.00}}
\put(23.00,4.00){\line(0,1){8.00}}
\put(24.00,43.00){\line(1,0){19.00}}
\put(23.00,23.00){\line(0,-1){9.00}}
\put(4.00,43.00){\line(1,0){18.00}}
\put(23.89,42.55){\line(2,-1){18.21}}
\put(43.00,42.00){\line(0,-1){8.00}}
\put(3.00,42.00){\line(0,-1){8.00}}
\put(4.00,33.00){\line(1,0){38.00}}
\put(3.89,33.45){\line(2,1){18.21}}
\put(23.00,3.00){\line(-2,3){19.45}}
\put(23.00,3.00){\line(2,3){19.45}}
\put(23.71,13.71){\line(1,1){18.59}}
\put(23.89,23.45){\line(2,1){18.21}}
\put(23.00,23.00){\line(-2,1){19.11}}
\put(22.29,13.71){\line(-1,1){18.59}}
\end{picture}
\end{document}
```

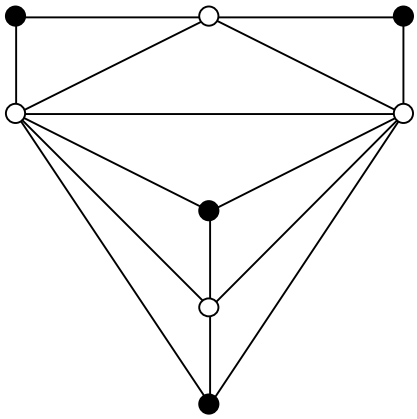

Abb.3.3 - Testzeichnung 3

Testzeichnung 3 wurde mit 400 dpi eingescannt. Das ergab ein 922*944 Pixel großes Bild. Während der Verarbeitung wurden 411 Kanten angelegt. Nach der Elimination der Artefakte bestand die Beschreibung nur noch aus 77 Kanten. Nach 67 Sekunden Rechenzeit wurde folgende Beschreibung ausgegeben:

```
\documentstyle {article}
\begin {document}
\unitlength 1.0mm
\begin {picture}( 63,  64)
\put (33.59, 57.70) {\line (1, 0) {23.60}}
\put (7.51, 44.63) {\line (1, 0) {49.67}}
\put (7.52, 57.70) {\line (1, 0) {23.62}}
\put (33.46, 57.11) {\line (2, -1) {23.86}}
\put (7.38, 44.01) {\line (2, -1) {28.76}}
\put (7.00, 43.79) {\line (1, -1) {24.46}}
\put (6.81, 43.64) {\line (2, -3) {24.53}}
\put (58.43, 56.44) {\line (0, -1) {10.60}}
\put (32.30, 17.21) {\line (0, -1) {9.94}}
\put (32.30, 37.52) {\line (0, -1) {17.85}}
\put (6.27, 56.34) {\line (0, -1) {10.56}}
\put (57.73, 43.59) {\line (-2, -3) {24.47}}
\put (57.45, 43.62) {\line (-1, -1) {24.30}}
\put (57.31, 44.07) {\line (-2, -1) {28.83}}
\put (31.27, 57.12) {\line (-2, -1) {23.94}}
\put (32.28, 5.49) {\circle*{2.49}}
\put (58.43, 57.69) {\circle*{2.49}}
\put (32.32, 31.55) {\circle*{2.50}}
\put (6.28, 57.63) {\circle*{2.49}}
\put (58.41, 44.61) {\circle {2.47}}
\put (32.30, 18.44) {\circle {2.46}}
\put (32.37, 57.66) {\circle {2.46}
```

```
\put (6.26, 44.55) {\circle {2.46}}
\end {picture}
\end {document}
```

Auch bei dieser Zeichnung wurden alle Elemente gefunden. Die Längenangaben bei den Geraden weichen aber teilweise recht stark vom Original ab. Das liegt, wie schon bei Testzeichnung 1 erwähnt, daran, daß der Start- oder Endpunkt dieser Geraden von einem ausgefüllten Kreis verdeckt ist. Die LATEX-Bilderkennung kann nur feststellen, daß die Gerade irgendwo im Kreis beginnt. Die genaue Position darf frei gewählt werden.

Toleranzparameter

Wie schon bei Testzeichnung 1 angekündigt, wurde auch hier eine Testserie mit dem Toleranzparameter für die Kurvenvereinigung durchgeführt. Tabelle 3.4 faßt die Ergebnisse zusammen.

Toleranz	*Kanten im Apex*	*Maximum beim Aufbau*	*Kanten nach Elimination*	*Ergebnisse*
0.1	324	458	225	1 zusätzlicher Punkt
0.2	148	407	103	1 zusätzlicher Punkt
0.3	112	395	81	richtig
0.4	98	391	74	richtig
0.5	96	391	72	richtig
0.6	97	391	72	richtig
0.7	96	389	67	richtig
0.8	92	388	68	richtig
0.9	89	386	67	richtig
1.0	89	382	73	richtig
1.1	85	382	72	1 Kreis nicht erkannt
1.2	84	382	71	1 Kreis nicht erkannt

Tab.3.4 - Testserie für den Toleranzparameter bei Testbild 3, 400 dpi

Im Unterschied zu dem ersten Test setzen sich die negativen Auswirkungen des zu kleinen Schwellwertes bis in das Ergebnis fort. Bei den Werten 0.1 und 0.2 wird ein sehr kleiner Kreis (der Durchmesser ist kaum größer als die Liniendicke) auf einer Geraden gefunden.

Auch hier bleibt wieder ein recht großes Intervall mit geeigneten Schwellwerten übrig. Es ist also für den Benutzer nicht schwierig einen vernünftigen Wert zu finden.

3.4 Allgemeine Verwendbarkeit des Ansatzes

Die Aufgabenstellung dieser Diplomarbeit erhielt ihre Motivation aus dem allgemeineren Forschungsgebiet „Erkennen technischer Zeichnungen“. Es stellt sich somit die Frage, wie gut der hier vorgeschlagene Algorithmus für das allgemeinere Problem geeignet ist.

Bei beiden Problemstellungen steht das Auffinden von Geraden und Kreisen im Vordergrund. Diese Aufgabe kann, wie in der Diplomarbeit gezeigt wurde, mit diesem Ansatz sehr gut bewältigt werden.

Es wurde aber nicht einfach nur ein Algorithmus zur Erkennung von Kreisen und Geraden entwickelt. Vielmehr ist der Algorithmus fähig, sich an Einschränkungen der vorgegebenen LATEX-Syntax zu halten. Es wurde somit exemplarisch gezeigt, wie Vorwissen über den Inhalt des Bildes in den Erkennungsprozeß eingearbeitet werden kann.

Bei der LATEX-Bilderkennung wurde außerdem gezeigt, wie mit diesem Ansatz auch Pfeilspitzen, Rechtecke und Ovale erkannt werden können. Analog dazu kann auch eine Erkennung von anderen Symbolen oder Figuren, z.B. Schaltsymbolen, programmiert werden. Wieder geht man von einer Definition der Objekte aus, z.B.:

Definition Lampe: Ein Lampe besteht aus einem Kreis und seinen beiden 45° geneigten Durchmessern. (Abb.3.4 a)

Definition Halbleiterdiode: Eine Halbleiterdiode ist dargestellt durch ein gleichschenkeliges Dreieck, das heißt drei Geraden die jeweils im Endpunkt der vorigen Gerade beginnen, wobei zwei Geraden die gleiche Länge haben, und durch eine weitere Gerade, die parallel zur Basis verläuft, die gleiche Länge hat und durch die Spitze des Dreiecks läuft. (Abb.3.4 b)

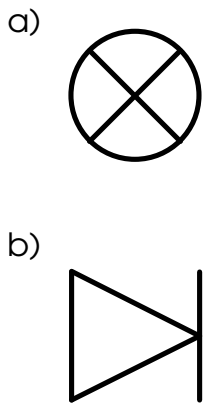

Abb.3.4 - Beispiele für Schaltsymbole

Um aus diesen Definitionen einen Erkennungsalgorithmus zu entwickeln, kann man dieselbe Strategie anwenden, die schon in Kapitel 2.7 bei der Rechteckerkennung zum Erfolg geführt hat. Es werden wiederum alle Teile des Objekts nacheinander gesucht. Jedes Teil muß Bedingungen über Lage und Form erfüllen, die sich aus der Definition des Gesamtobjekts und der Lage der bereits gefundenen Teile ergeben. Werden alle Teile gefunden, so können sie zu dem Gesamtobjekt zusammengefaßt werden.

Für die Lampenerkennung könnte der Erkennungsalgorithmus demnach folgendermaßen aussehen: Begonnen wird mit irgendeinem Kreis (der Radius kann evtl. aus der Anwendung vorgegeben sein). Die Lage der beiden Geraden ist durch den Kreis bereits festgelegt. Sie sind ja 45° geneigte Durchmesser des Kreises. Existieren beide Geraden, so stellen diese drei Objekte eine Lampe dar.

Auch ein Algorithmus zur Erkennung von Halbleiterdioden ist leicht anzugeben: Man beginnt mit der Basis des gleichschenkeligen Dreiecks. Bei der Auswahl dieser ersten Gerade kann auch wieder eine Bedingung über Größe oder Orientierung aus der Anwendung berücksichtigt werden. Die zweite Gerade, einer der beiden Schenkel, muß nun zwei Bedingungen erfüllen: Erstens muß sie an die Basis anschließen, d.h.: sie muß einen Endpunkt mit der Basis gemeinsam haben. Zweitens soll ein gleichschenkeliges Dreieck entstehen, d.h.: der andere Endpunkt dieser Geraden muß auf der Normalen durch den Mittelpunkt der Basis liegen.

Sind diese beiden Geraden gefunden, so ist damit die Position der Halbleiterdiode bereits festgelegt. Die Endpunkte der beiden noch fehlenden Geraden werden aus den gefundenen Teilen berechnet. Existieren auch diese beiden Geraden, so dürfen alle vier zu einer Halbleiterdiode zusammengefaßt werden.

Wir sehen daran, daß die Verwendbarkeit des Lösungsansatzes, der in dieser Diplomarbeit vorgestellt wurde, weit über den Spezialfall LATEX-Bilderkennung hinausgeht. Die Analyse des Bildes durch eine Pyramide und die anschließende Weiterverarbeitung der Kanten-information zu Geraden und Kreisen stellt ein sehr gutes Fundament für einen allgemeinen Algorithmus zur Erkennung technischer Zeichnungen dar.

3.5 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde ein Algorithmus zur Erkennung von LATEX-Bildern entwickelt. Da LATEX-Bilder Linienzeichnungen sind, fügt sich die Aufgabenstellung sehr gut in das Forschungsgebiet „Erkennen technischer Zeichnungen“ ein. Dennoch weicht der hier beschriebene Algorithmus stark von den aus der Literatur bekannten Lösungsansätzen ab.

Es wird eine 2*2/4 Pyramide verwendet, um die große Menge einzelner Kantenpunkte zu wenigen Geraden und Kreisbögen zusammenzufassen. Jeder Pixel in der Pyramide enthält eine Liste aller Kanten in seinem rezeptiven Feld. Aufgrund der Repräsentation der Kanten durch Akkumulatoren bleibt die Information über die genaue Lage und Form der Kanten bis hinauf in den Apex erhalten. Aus dem Akkumulator kann jederzeit die Lage der Kante nach der Methode des kleinsten quadratischen Fehlers approximiert werden. Der Apex enthält somit eine genaue Beschreibung aller Kanten im gesamten Bild.

Aus dieser Kantenliste werden dann Artefakte, die beim Aufbau der Pyramide entstehen, eliminiert. Anschließend werden Kanten, die durch Überschneidungen mit anderen Linien unterbrochen sind, zu Kantenfolgen zusammengefaßt. In weiterer Folge werden aus gegenüberliegenden Kantenfolgen Linien gebildet, die dann noch zu Rechtecken und Ovalen weiter zusammengefaßt werden können.

Dieser Algorithmus ist fähig, Grauwertbilder zu verarbeiten, und kommt somit einer häufig in der Literatur gestellten Forderung nach. Der Vorteil besteht darin, daß Grauwertbilder mehr Information enthalten als Binärbilder. Dadurch ist es möglich, die Bilder mit geringerer Auflösung zu digitalisieren, oder man nützt die Mehr-Information einfach zu genaueren Ergebnissen.

Trotz der Verarbeitung von Grauwertbildern verbraucht der Algorithmus extrem wenig Speicherplatz. Da das Bild zeilenweise abgearbeitet werden kann, ist es nicht notwendig, das gesamte Bild im Speicher zu haben. Es muß somit nur der wirkliche Bildinhalt gemerkt werden. Der Aufwand dafür ist deutlich geringer als der für das Abspeichern des Bildes als Binärbild. Dadurch wird es auch möglich, extrem große Bilder zu verarbeiten.

Zuletzt sei noch bemerkt, daß dieser Ansatz auch für die allgemeinere Problemstellung des Erkennens technischer Zeichnungen geeignet ist. Auch dort geht es hauptsächlich um das Auffinden der Geraden und Kreise im Bild. Da diese Aufgabe, wie in der Diplomarbeit gezeigt wurde, mit dem hier vorgestellten Lösungsansatz sehr gut bewältigt werden kann, bietet dieser Algorithmus eine sehr gute Basis für die Bewältigung allgemeinerer Problemstellungen.

Referenzen

[Davi78] Davis L.S., Rosenfeld A., „Noise Cleaning by Iterated Local Averaging“, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-8, No.9, S.705-710, 1978

[Dori93] Dori D., Liang Y., Dowell J., Chai I., „Sparse-Pixel Recognition of Primitives in Engineering Drawings“, Machine Vision and Applications, Vol.6, S.69-82, 1993

[Harr82] Harris J.F., Kittler J., Llewellyn B., Preston G., „A Modular System for Interpreting Binary Pixel Representations of Line-structured Data“, in Pattern Recognition: Theory and Applications, S.311-351, D.Reidel, Dordrecht, Niederlande, 1982

[Hart85] Hartley R.L., Rosenfeld A., „Hierarchical Curve Linking for Corner Detection“. In Levialdi S., Editor, „Integrated Technology for Parallel Image Processing“, S.101-119, Academic Press, London, 1985

[Hofe87] Hofer-Alfeis J., Frank K., „Automatisierte Umsetzung mechanischer Konstruktionszeichnungen in CAD-Modelle; Stand der Technik, Probleme, Lösungsansätze“, Mustererkennung 1987, Informatik Fachberichte 149, S.82-86, Springer Verlag, 1987

[Kari85] Karima M., Sadhal K.S., McNeil T.O., „From Paper Drawings to Computer Aided Design“, IEEE Compututer Graphics and Applications, Vol.5, No.2, S.27-39, 1985

[Kast90] Kasturi R., Bow S.T., El-Masri W., Shah J., Gattiker J., Mokate U., „A System for Interpretation of Line Drawings“, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.12, No.10, S.978-991, October 1990

[Krau89] Krause F.L., Jansen H., Großmann G., „Automatic Scanning and Interpretation of Engineering Drawings for CAD-Processes“, Annals of the CIRP, Vol.38/1, S.437-441, 1989

[Krau93] Krause F.L., Jansen H., Luth N, „Automatic Recognition of Scanned Technical Drawings“, Lecture Notes on Computer Science 719, Proc. of 5th Int. Conf. on Computer Analysis of Images and Patterns (CAIP), Budapest, S.609-613, 1993

[Krop86] Kropatsch W.G., „Curve Representations in Multiple Resolutions“, Proceedings Eighth Int. Conf. on Pattern Recognition, S.1283-1285, 1986

[Krop91] Kropatsch W.G., „Image Pyramids and Curves, An Overview“, Techn. Report PRIP-TR-2, Institut für Automation, TU Wien, 1991

[Lamp86] Lamport L., „LATEX - A Document Preparation System“, S.101-110, 1986

[Lu88] Lu W., Ohsawa Y., Sakauchi M., „A Database Capture System for Mechanical Drawings Using an Efficient Multi-dimensional Graphical Data Structure“, Proceedings of Int. Conf. on Pattern Recognition (ICPR), S.266.269, 1988

[Musa88] Musavi M.T., Shirvaikar M.V., Ramanathan E., Nekovei A.R., „A Vision Based Method to Automate Map Processing“, Pattern Recognition, Vol.21, No.4, S.319-326, 1988

[Naga90] Nagasamy V., Langrana N., „Engineering Drawing Processing and Vectorization System“, Computer Vision, Graphics and Image Processing, Vol.49, S.379-397, 1990

[Rose84] Rosenfeld A., Editor, „Multiresolution Image Processing and Analysis“ Springer Berlin, 1984

[Rous87] Rousseeuw P.J., Leroy A.M., „Robust Regression & Outlier Detection“, John Wiley & Sons, 1987

[Suzu90] Suzuki S., Yamada T., „MARIS: Map Recognition Input System“, Pattern Recognition, Vol.23, No.8, S.919-933, 1990

[Zhan84] Zhang T.Y., Suen C.Y., „A Fast Parallel Algorithm for Thinning Digital Patterns“, Communications of the ACM, Vol.27, No.3, S.236-239, 1984

[Zhan92] Zhang L.Y., Zhu J.Y., Zhu Q.Y., „A New System for Automatic Understanding Engeneering Drawings“, Annals of the CIRP, Vol.41/1, S.477-479, 1992