# 14. Bildsegmentierung

Mit der *Bildsegmentierung* beginnt der Übergang von der rein numerischen Darstellung des Bildes in einer Grauwertmatrix zu einer symbolischen Darstellung. Im Unterschied zur Mustererkennung (siehe Kapitel 16) werden bei der Bildsegmentierung nur einfache Objekte wie Linien, Flächen und Punkte ermittelt, aber nicht Linien oder Flächen mit symbolischen Namen wie "Umriss eines Hauses", "Silhouette eines Baumes" usw. versehen. Für die Segmentierung wird in der Regel kein explizit repräsentiertes Wissen oder nur wenig Information verwendet. Diese Informationen können Eigenschaften der zu segmentierenden Objekte wie der Grauwert, die Form, die Größe oder die Lage im Bild sein.

Das Ziel der Bildsegmentierung, nämlich die Unterteilung des Bildes in Teilbereiche mit gleichen Eigenschaften, wird meist mit Verfahren aus der Kanten- und Flächendetektion erreicht. Dadurch werden die für die Anwendung "interessanten Objekte" extrahiert, um sie leichter weiterverarbeiten zu können. Als Beispiele seien hier die Trennung von Text und Graphik zur Dokumentenanalyse und die Separierung von Organen, Zellen, Chromosomen usw. bei der biomedizinischen Bildverarbeitung genannt. Damit erfolgt eine erste Bedeutungszuweisung zu einzelnen Gebieten im Bild. Die Bildsegmentierung gehört daher nicht mehr zur klassischen Bildvorverarbeitung sondern zählt schon zur Mustererkennung und Bildinterpretation.

Die Unterteilung des Bildes in homogene Abschnitte geschieht nach verschiedenen Kriterien. Normalerweise werden die einzelnen Gebiete über Kantendetektoren oder Flächen-Wachstums-Algorithmen selektiert. Aber auch andere Kriterien wie "Lage im Bild" oder "Veränderung zum vorhergehenden Bild" sind denkbar. Falls es sich bei den Vorlagen um Farbbilder handelt, kann die zusätzliche Farb-Information ebenfalls zur Segmentierung verwendet werden.

## 14.1 Schwellwertbildung zur Bildsegmentierung

Eines der einfachsten Verfahren zur Bildsegmentierung ist die Anwendung eines *Schwellwertverfahrens*, um aus dem Grauwertbild ein Binärbild oder ein Bild mit reduzierten Graustufen (Quantisierung; siehe Kapitel 10) zu erzeugen. Dies ist vor allen Dingen bei solchen Bildern sinnvoll, bei denen der Bildinhalt schon in "Hintergrund" und "Objekt von Interesse" eingeteilt werden kann. Typische Beispiele dafür sind Dokumente/Texte und Strichzeichnungen.

Jeder Bildpunkt wird dabei in Abhängigkeit einer Schwelle $T$ entweder auf die Hintergrundfarbe (Grauwert $\leq T$) oder auf die Objektfarbe (Grauwert $> T$) gesetzt. Um eine zu starke Verästelung beider Regionen zu verhindern, ist eine vorherige Glättung durch Tiefpaßfilterung (Medianfilter, Mittelwertfilter) empfehlenswert. Da die Gebietszuordnung nur von dem Grauwert des aktuellen Bildpunktes und der Schwelle $T$ abhängt, bezeichnet man diese Segmentierungsverfahren auch als punktorientierte Verfahren [Jä89].

Wie schon bei den digitalen Halbtonverfahren (siehe Kapitel 6) deutlich wurde, existiert keine Schwelle *T*, die universell für alle Bildvorlagen einsetzbar ist. Die Schwelle muß daher für jedes Bild neu berechnet werden.

Besitzt die Vorlage schon binären Charakter, so kann die Schwelle einfach über das Histogramm ermittelt werden. Dieses besitzt dann ein bimodales Verhalten, d.h. es existieren im Histogramm zwei deutlich voneinander unterscheidbare Maxima. Auch bei natürlichen Aufnahmen ist eine solche Histogrammform meist zu erkennen und kann daher zur Schwellwertberechnung verwendet werden (siehe Abbildung 14.1).

Zur genaueren Bestimmung des lokalen Minimums (oder der Minima) ist eine Glättung des erstellten Histogramms durch Mittelwertbildung (siehe Abbildung 14.2) oder Berechnung des Medians von benachbarten Histogrammwerten günstig [Pav82].

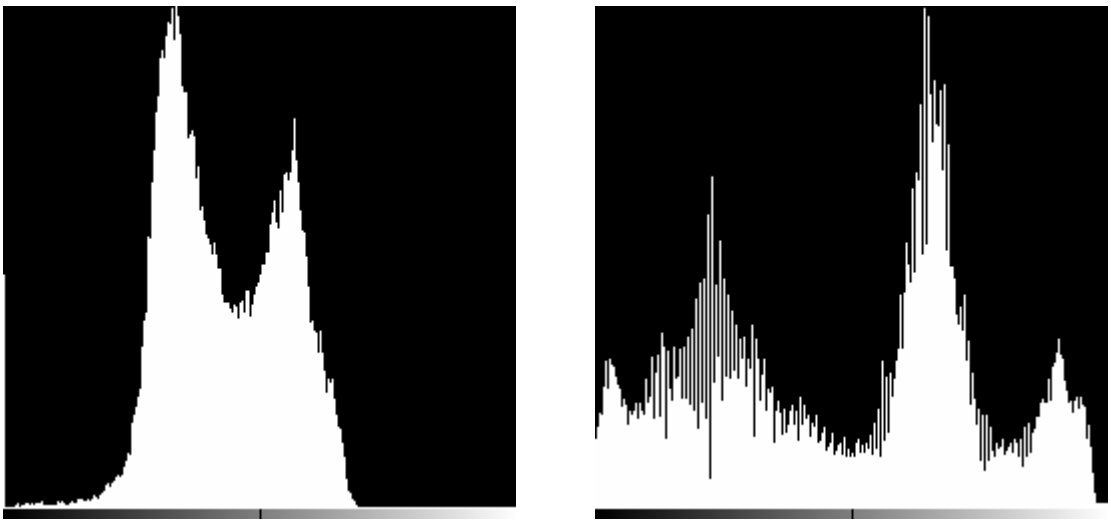

Abbildung 14.1: Über das bimodale Verhalten des Histogramms kann ein an die Bildvorlage adaptierter Schwellwert zur Segmentierung berechnet werden.

Selbstverständlich können auch mehrere unterschiedliche Schwellwerte verwendet werden, falls mehrere lokale Maxima im Histogramm auftreten und diese mit Gebieten von Interesse im Originalbild korrespondieren. Dies entspricht dann einer nicht äquidistanten Quantisierung.

Ist die relative Objektgröße bzw. der Flächenanteil der Objekte am Gesamtbild bekannt, kann der Schwellwert prozentual über das kumulative Histogramm berechnet werden. Dazu werden ausgehend von dem sicheren (bekannten) Objektgrauwert so viele Bildpunkte auf den Objektgrauwert gesetzt, bis der entsprechende Prozentsatz erreicht ist. Alle anderen Bildpunkte werden auf den Hintergrundgrauwert gesetzt. Analog kann natürlich auch vom Hintergrund ausgegangen werden.

Weitere Schwierigkeiten kommen hinzu, wenn Grauwerte des Hintergrunds auch in den Objekten auftreten bzw. umgekehrt. In diesen Fällen können Techniken mit dynamischem Schwellwert verwendet werden. Der Schwellwert *T* wird dabei für jeden Bildpunkt in Abhängigkeit von seiner

Umgebung neu berechnet. Geht man davon aus, daß Hintergrund und Objekt in einem $m \times n$-Ausschnitt etwa gleich häufig auftreten, so kann man den Mittelwert dieses Ausschnittes als Schwellwert verwenden. Trifft diese Annahme nicht zu, ist es möglich, durch Bimodalitätsprüfung des Histogramms einen geeigneten Schwellwert zu finden. Diese Vorgehensweise ist dann ähnlich der adaptiven Histogrammeinebnung (siehe Abschnitt 10.3.2).

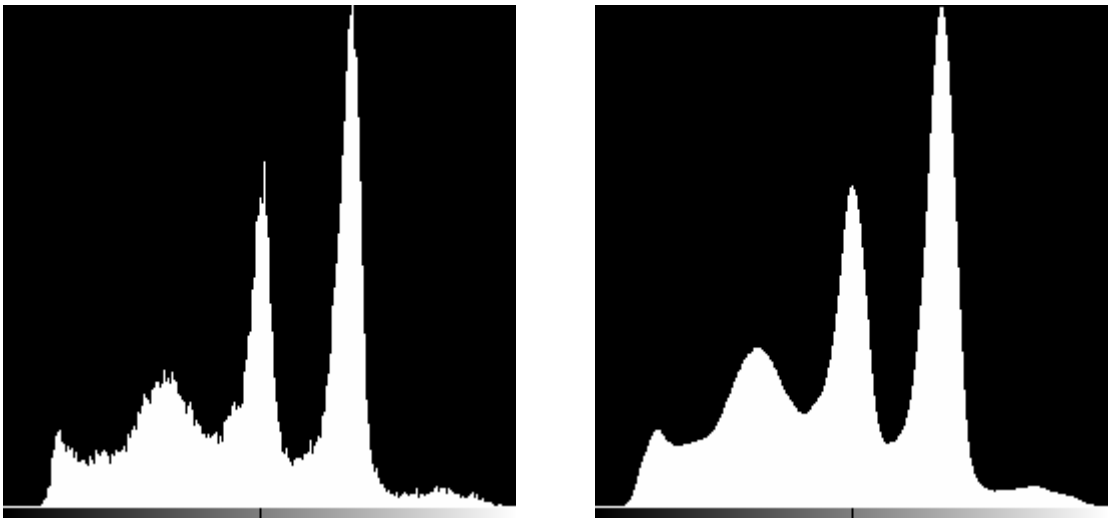

Abbildung 14.2: Die Glättung des Original-Histogramms (links) erfolgt durch Berechnung des gleitenden Mittelwerts. In dem geglätteten Histogramm (rechts) können lokale Minima leichter bestimmt werden.

## 14.2 Segmentierung über Templates

Ist die Anzahl der im Bild vorkommenden unterschiedlichen Objekte eng begrenzt und die jeweilige Form bekannt, so kann aufgrund dieses Vorwissens eine Segmentierung direkt über entsprechende *Schablonen* (*Templates*) erfolgen. Solche Segmentierungsverfahren eignen sich besonders in der industriellen Umgebung, wo mit konstanten Aufnahmeparametern gute Bedingungen für die Bildverarbeitung geschaffen werden können.

Durch die fast ideale Beleuchtung in solchen Einsatzgebieten enthält das zu segmentierende Bild nur noch wenige Graustufen. Die einzelnen Schablonen werden dann in den möglich vorkommenden Orientierungen an alle Positionen auf das Bild gelegt und dann die darunterliegenden Bildpunkte analysiert. Entsprechen diese in der Form und im Grauwert komplett oder zu einem hohen Prozentsatz der Schablone und schließen keine weiteren Bildpunkte mit demselben Grauwert direkt an, so wird das von der Schablone bedeckte Gebiet dem entsprechenden Objekt zugeordnet.

Bei diesem Schritt wird somit nicht nur eine Segmentierung sondern auch schon eine Bedeutungszuweisung vorgenommen. Diese Information kann direkt für die weitere Verarbeitung, z.B. Sortierung oder Auszählung benutzt werden.

Wenn, wie in diesen Einsatzgebieten häufig der Fall, nur wenige unterschiedliche Objekte, oft sogar noch mit fester Orientierung vorkommen, kann die Segmentierung mit einer geringen Anzahl Templates sehr schnell durchgeführt werden.

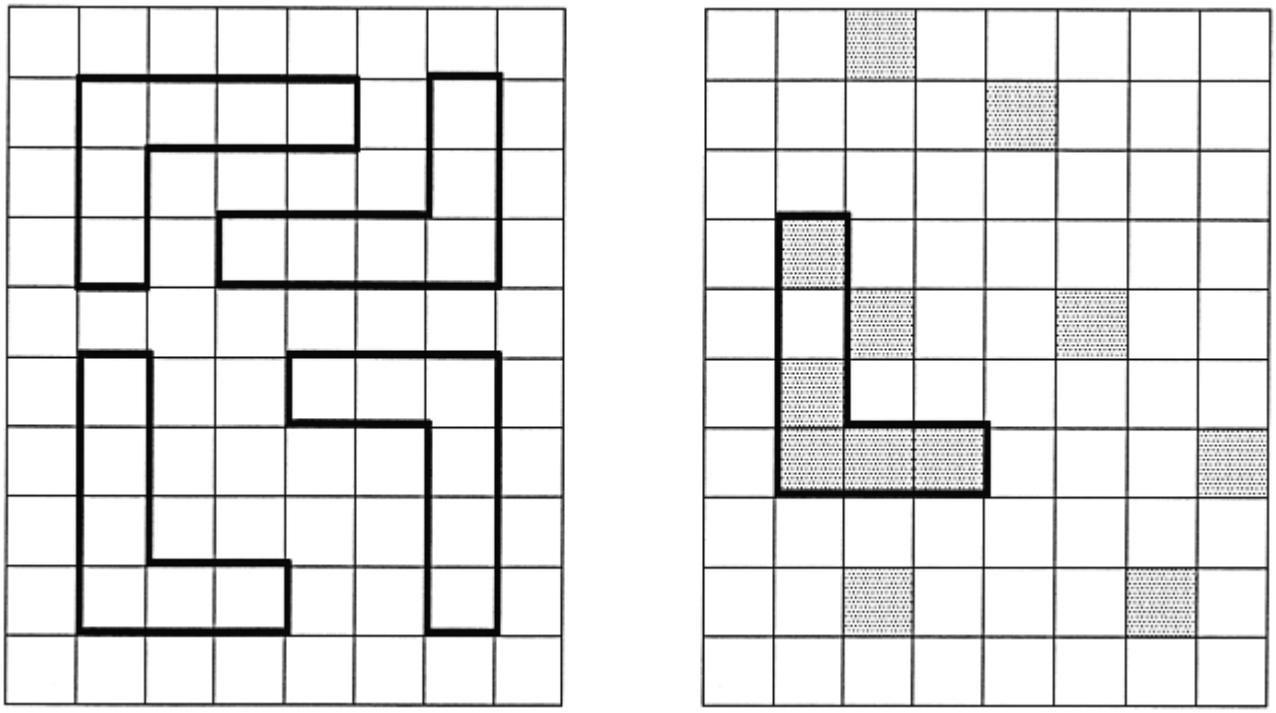

Abbildung 14.3: Segmentierung mit Hilfe von Templates. Alle vorkommenden Formen werden durch entsprechende Schablonen in allen möglichen Orientierungen (links) repräsentiert. Stimmt eine Schablone zum größten Teil mit dem darunterliegenden Bildteil überein (rechts), so wird aufgrund der Information über die Schablone das zugehörige Objekt und seine Lage bestimmt.

## 14.3 Segmentierung mit Hilfe der Kantendetektion

Wie schon im Kapitel 11 beschrieben, entsprechen die in einem Bild detektierten Kanten oft den Konturen bzw. Grenzen der zugehörigen physikalischen Objekte. Kantendetektionsverfahren sind daher für Bildsegmentierung und Bedeutungszuweisung wichtig.

Während bei der Segmentierung über Schwellwerte oder durch Gebietswachstum die einzelnen Objekte als Flächen extrahiert werden, werden mit Hilfe der Kantendetektionsverfahren die Umrandungen der Objekte bestimmt. Wie schon bei den Kantenoperatoren (siehe Kapitel 11) deutlich wurde, muß diese Umrandung nicht unbedingt geschlossen sein.

Kleinere Lücken lassen sich noch leicht mit geeigneten Filtermasken schließen (vergleiche Kapitel 10). Bei größeren Unterbrechungen kann eine Konturverfolgung mit der weiteren Suche in der Hauptrichtung der Kontur bei Lücken zum Erfolg führen.

Die komplett umrandeten Gebiete müssen zum Schluß noch mit Hilfe von Füllalgorithmen [FvD*90] genau bestimmt werden. Unterschiedliche Größen von Füllelementen erlauben hier ebenfalls das Schließen kleinerer Lücken. Das stufenweise Verwenden von zuerst großen Füllmustern und danach immer kleineren bis zum Schluß auf Pixel-Größe kann gleichzeitig eine hohe Genauigkeit erzielen und ein Ausfließen beim Füllen der offenen Objekte verhindern.

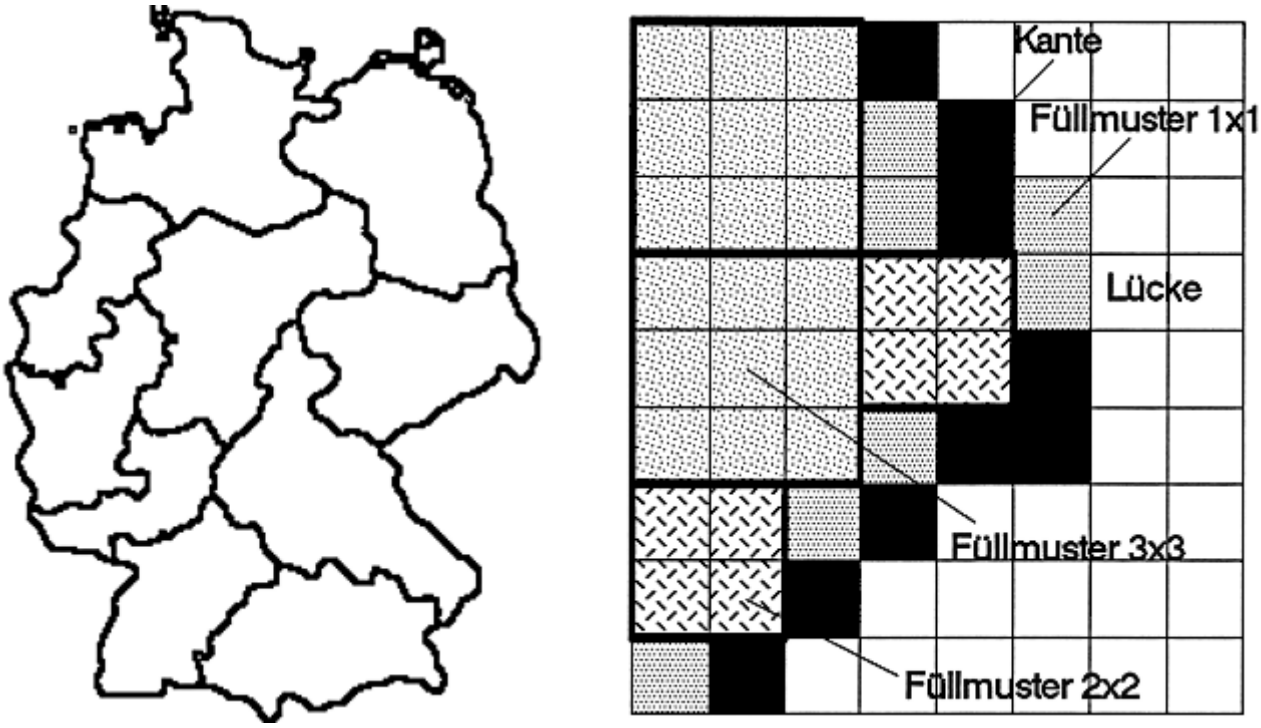


Abbildung 14.4: Das linke Bild zeigt das Ergebnis der Kantendetektion für eine Segmentierung. Rechts ist das stufenweise Vorgehen mit drei unterschiedlich großen Füllmustern bei offenen Umrandungen schematisch dargestellt.

## 14.4 Segmentierung durch Gebietswachstum

Bei der Segmentierung durch *Gebietswachstum* wird in der Hauptsache das Wissen über den Grauwert bzw. die Farbe der gesuchten Objekte ausgenutzt. Im Bild wird dann ein Repräsentant dieser Klasse gesucht und als Ausgangspunkt für das Gebietswachstum verwendet. Danach wird die Umgebung betrachtet und alle die Bildpunkte werden mit einbezogen, die ebenfalls in diese Klasse fallen, d.h. mit ihrem Wert (bis auf einen Toleranzwert) mit dem Vertreter der Klasse übereinstimmen. Diese Segmentierungsmethode gehört zu den regionenorienten Verfahren [Jä89].

Eine Variation des Verfahrens besteht darin, nicht den Wert einzelner Punkte, sondern immer den der ganzen Umgebung zu betrachten. Solange das so berechnete Gebiet noch gleichförmig ist, werden neue Punkte mit einbezogen. Dadurch werden kleine Störungen wie Rauschen unterdrückt. Eine Region *R* wird als gleichförmig bezeichnet, falls für den Wert *f(P)* eines Punktes *P* gilt

$$\max_{P \in R} \left| f(P) - m \right| < T$$

wobei *T* ein anwendungsabhängiger Schwellwert und *m* der mittlere Wert der Region *R* mit *n* Punkten ist.

$$m = \frac{1}{n} \sum_{P \in R} f(P)$$

Sind die Werte der einzelnen Klassen unbekannt, so können diese durch Ermittlung der lokalen Maxima im Histogramm berechnet werden. Aufgrund der Breite der Verteilungskurve kann auch eine sinnvolle Größenangabe zu dem Schwellwert *T* gemacht werden.

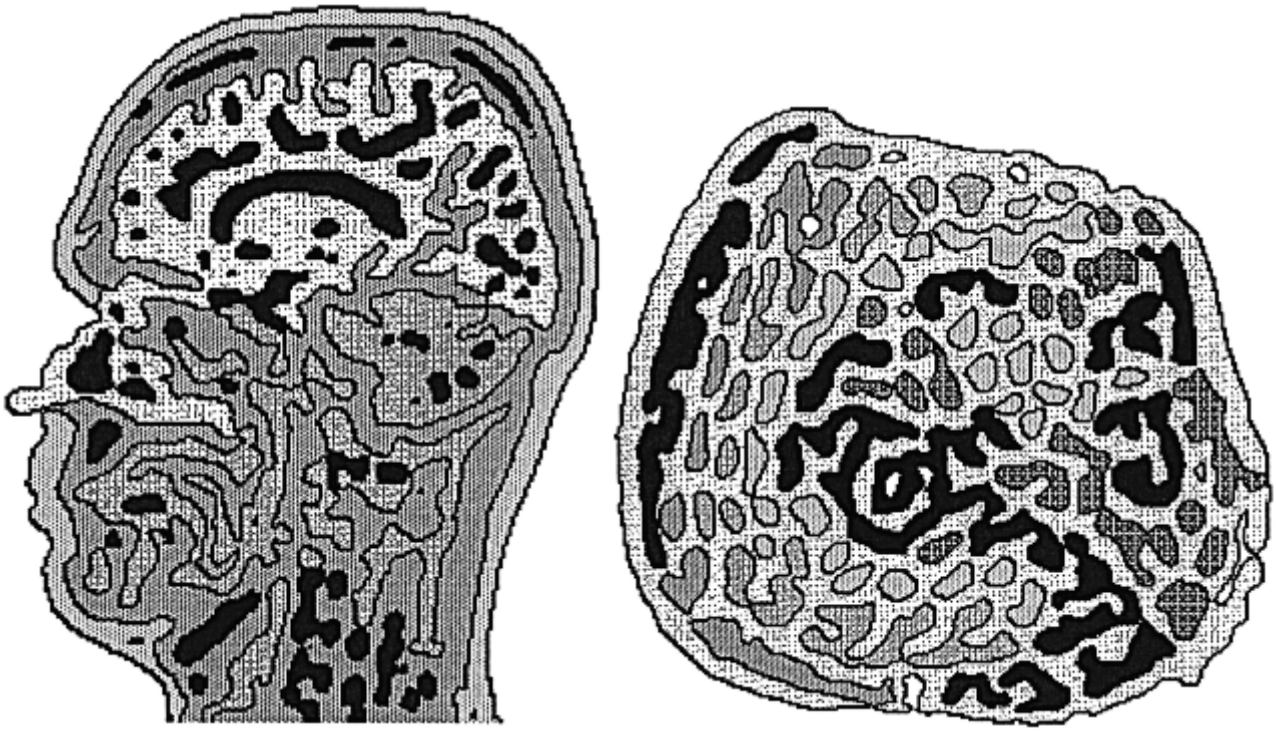

Abbildung 14.5: Segmentierung durch Gebietswachstum bei komplett umschlossenen Gebieten mit unterschiedlichen Grauwerten (vergleiche Abbildung 12.6).

# 14.5 Morphologische Operatoren

Skelettierung (siehe Kapitel 12), Erosion und Dilatation gehören zu den morphologischen Operatoren; ihre Anwendung verändert die Gestalt eines Objekts [Sch89], [BB91]. Die Skelettierung ist dabei eine spezielle Version der Erosion [CH89].

## 14.5.1 Erosion – Dilatation

Die *Erosion* beschreibt einen Vorgang, bestimmte Punkte vom Objektrand bzw. Objektinneren zu entfernen. Dazu ist analog zur Skelettierung die Umgebung des aktuellen Bildpunktes zu betrachten. Im gebräuchlichsten Fall ist diese Umgebung wieder durch eine 3×3-Maske festgelegt. Bei der Erosion wird ein Bildpunkt eines Objekts in das Ergebnis übertragen, falls mindestens $T$ Punkte aus der Maske mit den Bildpunkten übereinstimmen. Meist wird $T = 9$ bei einer 3×3-Maske benutzt. Alle Bildpunkte unter der Maske müssen also entweder Objektpunkte oder Hintergrundpunkte sein.

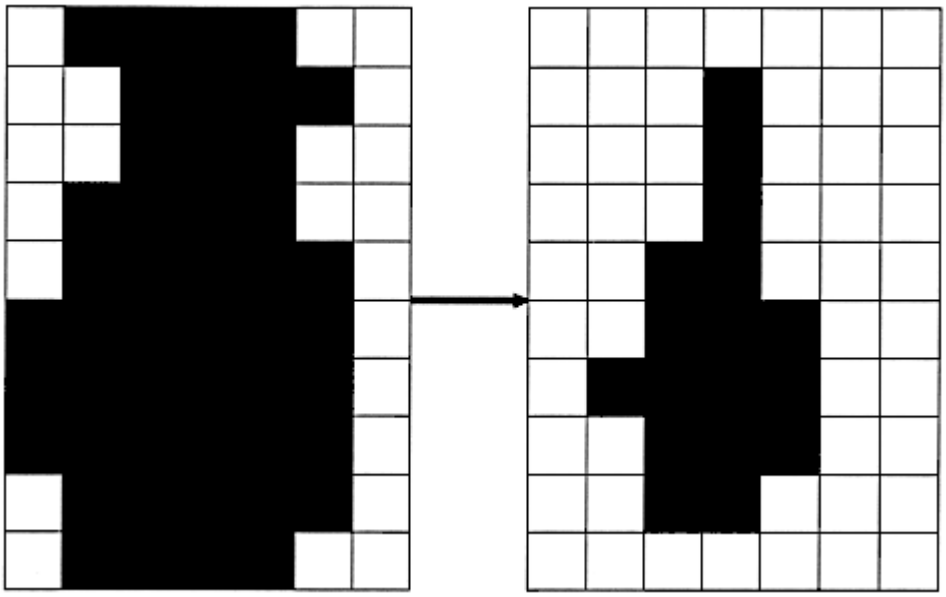

Abbildung 14.6: Das linke Bild wurde nach der Erosionsvorschrift mit einer 3×3-Maske und $T = 9$ bearbeitet. Im rechten Bild ist das Ergebnis dieser Operation zu sehen (Weiß = Hintergrund, Schwarz = Objekt).

Ist $(2n+1) \times (2n+1)$ die Maskengröße und $f(x, y)$ der aktuelle Bildpunkt, so berechnet sich der neue Punkt $f'(x, y)$ in einem Binärbild mit den Werten 0 (=Hintergrund) und 1 (=Objekt) aus

$$f'(x,y) = \begin{cases} 1 & falls \sum_{i=-n}^{n} \sum_{j=-n}^{n} f(x+i, y+j) \geq T \\ 0 & sonst \end{cases}$$

Die Erosion beseitigt somit Störungen. Die Größe der entfernten Störungen ist von der gewählten Schwelle *T* abhängig.

Die *Dilatation* beschreibt, wie der Name schon sagt, einen Wachstums- oder Ausdehnungsvorgang. Auch hier wird innerhalb einer Bildpunktumgebung entschieden, ob der aktuell betrachtete Punkt gesetzt werden soll oder nicht. Um einen Punkt im Ergebnisbild zu setzen, müssen mindestens *T* Bildpunkte unter der Maske mit dem Objekt übereinstimmen. Meist wird *T* = 1 benutzt.

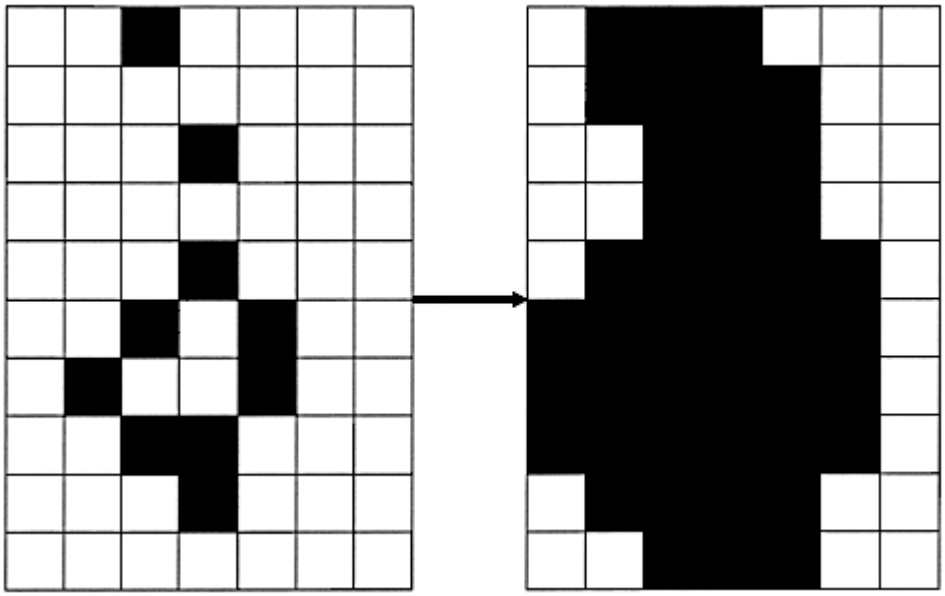

Abbildung 14.7: Das linke Bild wurde nach der Dilatationsvorschrift mit einer 3×3-Maske und T = 1 bearbeitet. Im rechten Bild ist das Ergebnis dieser Operation zu sehen (Weiß = Hintergrund, Schwarz = Objekt).

Die Erosion liefert den Teil des Originalbildes, der komplett von der Maske in Abhängigkeit von *T* bedeckt wird. Das Ergebnis stellt eine Art "Schnittmenge" dar. Die Dilatation hingegen liefert die "Vereinigungsmenge" von Originalbild und Maske, sofern nur mindestens *T* Bildpunkte mit der Maske übereinstimmen. Die Wirkung beider Operatoren ist zwar gegenläufig, die Operatoren sind aber nicht invers, da die morphologischen Operatoren nicht linear sind.

Nicht nur die Größe der Schwelle *T*, auch die Form der Maske beeinflußt stark das Aussehen des Ergebnisbildes. Die Form der Maske ist in der Ergebnisstruktur oder in Teilen davon wiederzufinden. Somit lassen sich gezielt bestimmte Formen bevorzugen (z.B. Kreise durch eine kreisförmige Maske) oder auch unterdrücken.

### 14.5.2 Opening – Closing

Wie schon gezeigt wurde, läßt sich der Effekt einer Erosion mittels einer Dilatation zwar kompensieren, nicht aber korrigieren. Die Wirkung beider Operatoren hebt sich nicht auf. Ebenso ist die Anwendung dieser Operatoren nicht kommutativ. Die Anwendung einer Erosion nach einer Dilatation liefert ein anderes Ergebnis als die Anwendung einer Dilatation auf eine Erosion. Beide Kombinationen können aber jeweils eine spezielle gewünschte Wirkung haben.

Als *Opening* wird die Anwendung einer Erosion gefolgt von einer Dilatation bezeichnet. Diese Kombination dient im wesentlich zur Beseitigung von Störungen in Form vereinzelter Bildpunkte, kleiner Strukturen oder Auswüchse. Die Störungen müssen dabei jeweils kleiner als die Erosions-Maske sein.

Beim *Closing* wird nach einer Dilatation eine Erosion angewendet. Wie die Bezeichnung schon andeutet, werden kleinere Lücken geschlossen. Je nach Schwellwert $T$ werden bei Masken der Größe $(2n+1)\times(2n+1)$ sogar Lücken bis zur Größe $2n$ durch Closing geschlossen.

Die Opening-/Closing-Operationen können auch gezielt dazu genutzt werden, bestimmte Strukturen in einem Bild zu detektieren. Dazu kann eine solche Maskengröße und Maskenform für das Opening verwendet werden, damit garantiert die Objekte von Interesse gelöscht werden. Das Ergebnisbild enthält dann alle größeren Strukturen in einer geglätteten Form sowie den Hintergrund. Dieses Bild ist mit dem Originalbild mittels einer XOR-Funktion zu verknüpfen. In diesem Bild sind die gesuchten Objekte, die Ränder der größeren Strukturen (die durch das Opening erodiert wurden) und kleinere Strukturen wie Störungen. Mit einer weiteren, speziellen Maske wird eine zusätzliche Erosion durchgeführt. Diese Maske berücksichtigt die spezielle Form der gesuchten Objekte und löscht daher alle anderen Strukturen. Eine nachfolgende Dilatation reproduziert die gesuchten Objekte in etwa wieder in Originalgröße. Das Ergebnisbild markiert so die Stellen, an denen im Originalbild die gesuchten Objekte vorkommen.

## 14.6 Split–and–Merge

Die Vorgehensweise des *Split-and-Merge* zur Bildsegmentierung ist ähnlich der Quadtree-Kodierung (siehe Kapitel 15). Das Ausgangsbild wird dazu in vier gleich große Teile – in der Regel Quadrate – aufgeteilt und diese jeweils weiter unterteilt [Pav82], [GW87].

Das Split-and-Merge-Verfahren beginnt bei einer beliebigen Unterteilungstiefe. Zuerst wird untersucht, ob die aktuelle Teilfläche dem Gleichförmigkeitskriterium genügt. Dieses Kriterium kann besagen, daß

- die Fläche im Quadranten denselben Grau-/Farbwert besitzt,
- die Grau-/Farbwerte in einem bestimmten Intervall liegen,

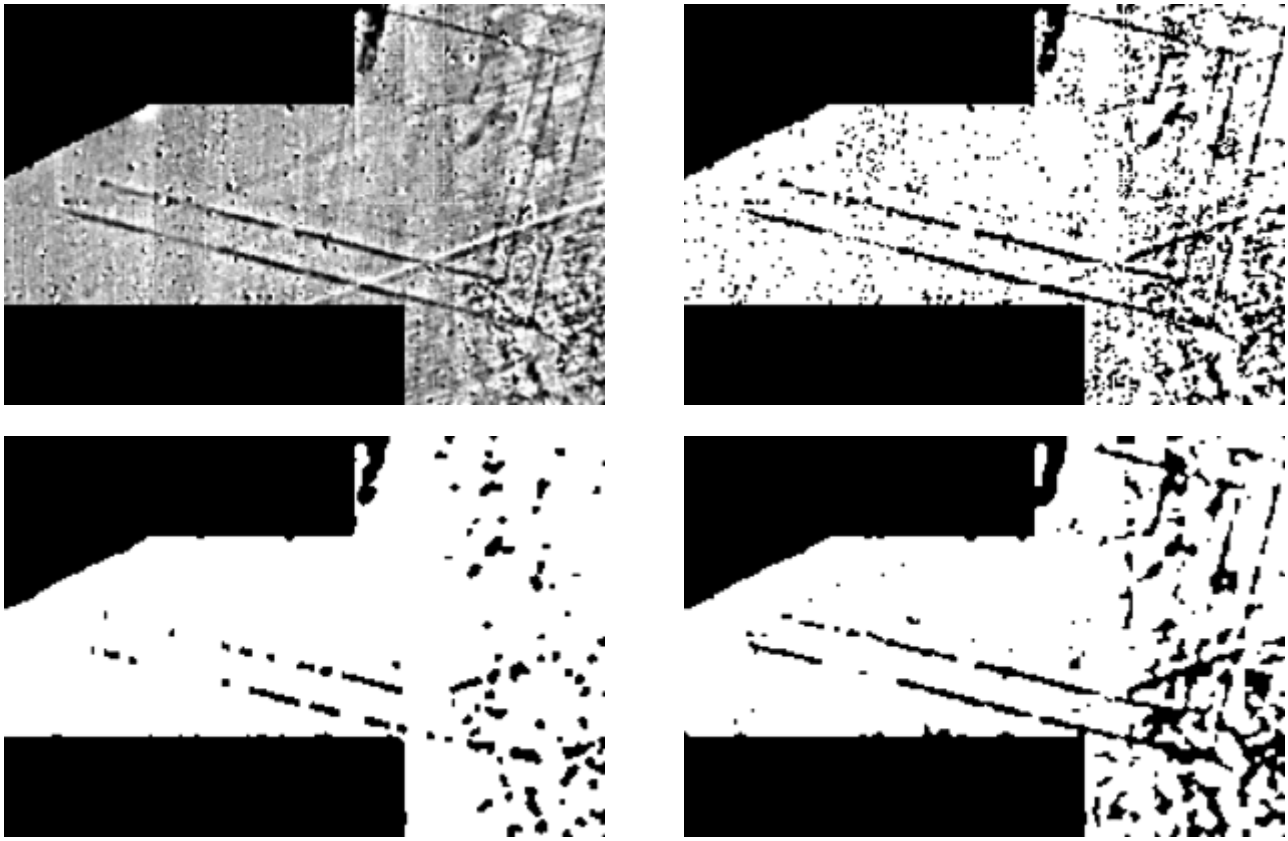

Abbildung 14.8: Beispiel für Opening und Closing

Links unten: das Ergebnis einer Opening-Operation.

Rechts unten: eine Closing-Operation auf dasselbe Beispielbild (rechts oben) angewendet.

Das Originalbild ist links oben zu sehen und zeigt den Grundriß eines römischen Militärlagers bei Hüfingen (Schwarzwald-Baar-Kreis) [JF90]. Die Daten wurden durch geomagnetische Prospektion mit einem Meßintervall von 0.5 m von Dr. H. G. Jansen ermittelt (Größe 120m×120m) und für diese Abbildung freundlicherweise zur Verfügung gestellt.

- eine bestimmte Verteilung der Grau-/Farbwerte in dem Quadranten zu finden ist usw. (siehe Abschnitt 14.4).

Gleichförmige Flächen werden zusammenfügt (*merge*), nicht gleichförmige weiter unterteilt (*split*). Durch diese Vorgehensweise wird indirekt ein Quadtree aufgebaut. Dieser Baum wird aber nicht zur Kodierung o.ä. verwendet, sondern er beschreibt nur die Abtastung und Aufteilung des Originalbildes.

Der Vorteil dieses Verfahrens liegt in der einfachen algorithmischen Implementierung sowie in der Berücksichtigung gleichförmigen Flächen. Wird bis zur höchsten Auflösung unterteilt, so können beliebig geformte Flächen beschrieben werden.

Die jeweils als gleichförmig akzeptierten Flächen werden zusammengefügt und in das Ergebnisbild übertragen. Im Ergebnis und für die Weiterverarbeitung wird der aufgebaute Baum nicht mehr verwendet.

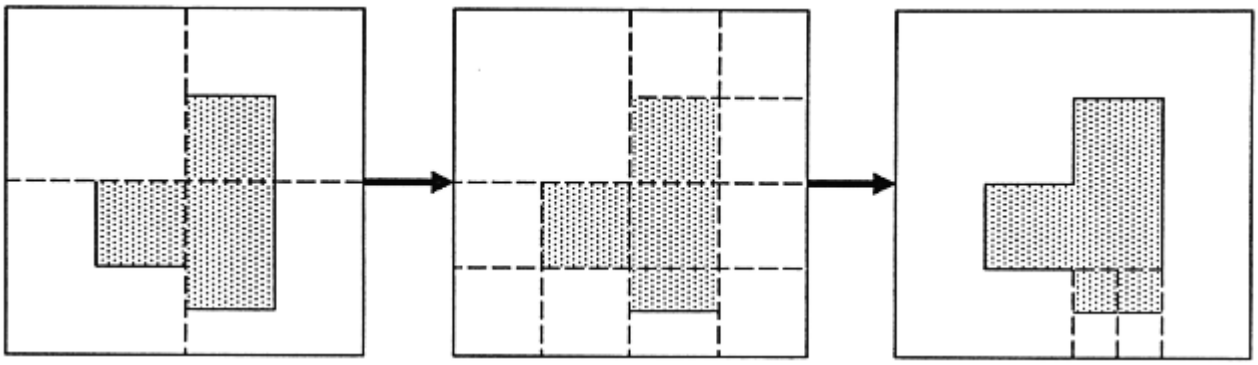

Abbildung 14.9: Vorgehensweise beim Split-and-Merge.

# Aufgaben

## Aufgabe 1

Beschreiben Sie verbal einen Füllalgorithmus, der auch Gebiete füllen kann, deren Umrandung Lücken bis zu zwei Pixel aufzeigt, ohne dabei "auszulaufen".

## Aufgabe 2

Welches Ergebnis erhält man, wenn auf das Originalbild von Abbildung 14.6 eine Erosion mit der Schwelle $T = 6$ vorgenommen wird? Vergleichen Sie das Resultat mit Abbildung 14.6.

## Aufgabe 3

Welches Ergebnis erhält man, wenn auf das Originalbild von Abbildung 14.7 eine Dilatation mit der Schwelle $T = 4$ vorgenommen wird? Vergleichen Sie das Resultat mit Abbildung 14.7.

## Aufgabe 4

Bei der Kantendetektion mit dem modifizierten Canny-Operator entstehen durch das Contraint-Thinning (siehe Abschnitt 11.10.3) Verbindungen ("Leitereffekt") von dicht beieinanderliegenden Kanten. Beschreiben Sie ein Verfahren, das diese Nebeneffekte mittels morphologischer Operatoren beseitigt.

Welche neuen Probleme ergeben sich dadurch?

# 15. Bildkodierung

Mit dem Begriff *Bildkodierung* soll in diesem Zusammenhang nicht die Verschlüsselung von Bilddaten, sondern die Umrechnung der Bildinformation in eine andere Form verstanden werden. Diese neue Form erlaubt es, Bilddaten mit geringerem Speicherbedarf abzuspeichern oder die Übertragungszeiten beispielsweise in einem Netzwerk zu reduzieren.

Daß solche Verfahren in der Praxis notwendig sind, zeigen einige Rechenbeispiele. Ein Grauwertbild mit 512×512 Punkten und 256 Graustufen (8-Bit) benötigt 256 KByte Speicher, ein Echtfarbenbild mit 1280×1024 Punkten und je 8-Bit für Rot, Grün und Blau belegt fast 4 MB Speicher. Soll ein DIN-A4-Farbbild mit einem Scanner mit 300 dpi Auflösung und 24-Bit Farbtiefe (3×8-Bit) digitalisiert werden, so werden rund 26 MB Speicher für dieses Bild benötigt.

Es ist daher notwendig, Verfahren einzusetzen, die diesen Speicherbedarf reduzieren. Dabei werden zwei prinzipielle Vorgehensweisen unterschieden.

1. Datenkompression

   Bei der *Datenkompression* werden die Originaldaten in eine andere Darstellungsform transformiert, die weniger Speicherplatz benötigt. Durch die inverse Transformation kann das Original wieder eindeutig und fehlerfrei erzeugt werden. Die Datenkompression ist eine Redundanz-Reduktion, d.h. es wird lediglich redundante Information eliminiert.

2. Datenreduktion

   Die *Datenreduktion* erzielt eine Speicherplatzersparnis durch das Weglassen nicht relevanter Bestandteile des Originalbildes. Aufgrund dieses Informationsverlustes ist es jedoch nicht mehr möglich, das Originalbild fehlerfrei wiederherzustellen.

Indirekt wurden in den vorangegangenen Kapiteln schon verschiedene Kompressionsverfahren angesprochen, unter anderem die Verwendung von Farbtabellen statt Farbtripeln bei einer geringen Anzahl von Farben.

## 15.1 Kodierung von Binärbildern

Handelt es sich bei den zu kodierenden Vorlagen um Binärbilder, so können in einem Byte 8 Bildpunkte Bit-kodiert verlustfrei gespeichert werden. Dies bedeutet eine Datenkompression um den Faktor 8. Eventuell muß dem komprimierten Bild der Grauwert der beiden Stufen vorangestellt werden. Ähnliches ist auch bei einer reduzierten Anzahl von Graustufen möglich. Bei nur 16 Graustufen können die Grauwerte zweier Bildpunkte in einem Byte gespeichert werden.

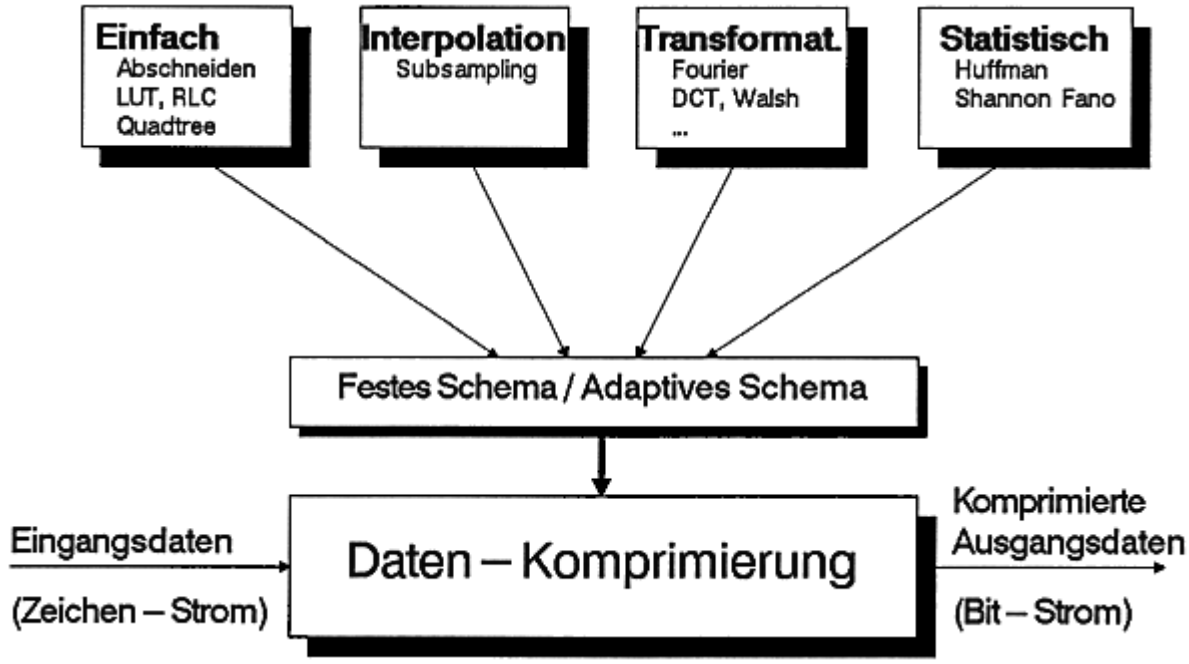


Abbildung 15.1: Verschiedene Komprimierungstechniken für Text- und Bilddaten.

Enthält das zu kodierende Bild nur Strichzeichnungen, so können statt der Speicherung aller Bildpunkte, die einzelnen Vektoren mit Anfangs- und Endpunkt extrahiert und gespeichert werden. Auch die Konstruktion von Richtungsketten (Kettenkodes) kann eine Komprimierung darstellen. Die Vektorisierungsverfahren berücksichtigen in der Regel nicht alle Bildpunkte und Variationen im Verlauf einer Kontur, so daß diese Art der Kodierung nicht verlustfrei ist. Es wird nur die relevante Information gespeichert, aus der das Originalbild angenähert rekonstruiert werden kann. Im Durchschnitt wird mit diesen Verfahren eine Datenreduktion um den Faktor 10-20 erreicht.

Die nachfolgend beschriebenen Verfahren lassen sich selbstverständlich auch bei Binärbildern anwenden.

## 15.2 Huffman-Kodierung

Es ist einleuchtend, daß die bisher vorgestellte Kodierung von Zeichen, bei der für alle Werte die gleiche Anzahl von Bits/Bytes benötigt wird, nicht optimal ist. Günstiger ist es zum Beispiel, wenn häufig vorkommende Symbole durch wenige Bits (kurze Kodeworte) und die seltenen Zeichen durch längere Kodeworte dargestellt werden. Diese Art von Kodierung ist unter dem Namen *Huffman-Kodierung* (David Huffman, 1952) bekannt und wird in ähnlicher Weise im Morse-Alphabet verwendet. Hierbei wird jedes Eingabezeichen in eine Bit-Kette mit einer variablen Anzahl Bits übersetzt. Die Länge der Bit-Kette hängt von der Häufigkeitsverteilung eines Zeichens

in den Eingabedaten ab. Je häufiger ein Zeichen in den Originaldaten vorkommt, desto kürzer ist der Bit-Kode (*VLC = Variable Length Code*) [Str87], [Wal91], [TO92]. Der Huffman-Kode wird auch als kompakter Kode bezeichnet, weil die mittlere Wortlänge kleiner oder gleich der mittleren Wortlänge aller anderen eindeutig dekodierbaren Kodes für dasselbe Eingangssignal ist, d.h. der Huffman-Kode ist ein minimaler Kode.

$$\text{Mittlere Kodelänge} = \sum_{i=1}^{n} P(i)\,L(i)$$

wobei *P(i)* die Wahrscheinlichkeit des Auftretens des Zeichens *i*, *L(i)* die Länge des Zeichens *i* und *n* die Anzahl der verschiedenen Zeichen ist.

Die Eigenschaft der Kompaktheit des Huffman-Kodes kann auch mit Hilfe der Entropie beschrieben werden. Dafür wird der Informationsgehalt *I* eines einzelnen Zeichens *i* definiert als

$$I = ld\left(\frac{1}{P(i)}\right)$$

mit *ld* dem Logarithmus zur Basis 2 und *P(i)* der Wahrscheinlichkeit des Auftretens des Zeichens *i*. Der mittlere Informationsgehalt *H* aller *n* vorkommenden Zeichen berechnet sich dann aus

$$H = \left|\sum_{i=1}^{n} P(i)\,ld\left(\frac{1}{P(i)}\right)\right|$$

Diesen gemittelten Informationsgehalt bezeichnet man als *Entropie*. Die Differenz zwischen mittlerer Kodelänge ($\geq H$) und Entropie ist demnach ein Maß für die Güte der Kodierung. Je mehr sich der Wert für die mittlere Kodelänge dem der Entropie annähert, desto besser ist komprimiert worden.

Nachteilig bei der Huffman-Kodierung wirkt sich die statische Tabelle aus. Eine Anpassung an eine spezielle Vorlage kann nur durch zweimalige Durchsuchung der zu kodierenden Daten erreicht werden. Im ersten Durchgang werden die Häufigkeiten der einzelnen Zeichen ermittelt und daraus die Kodierungstabelle aufgebaut. Erst im zweiten Durchlauf erfolgt die eigentliche Kodierung. Bei dieser dynamischen Methode muß die erzeugte Übersetzungstabelle zuerst abgespeichert bzw. dem Empfänger übermittelt werden. Bei kleinen Datenmengen kann diese Tabelle mehr Speicherplatz als die unkodierten Daten beanspruchen.

Erzeugt wird eine Übersetzungstabelle, indem die vorkommenden Zeichen der Häufigkeit nach sortiert werden (siehe Abbildung 15.2). Die zwei Zeichen mit der geringsten Häufigkeit werden zusammengefaßt und die Wahrscheinlichkeiten aufaddiert. Das rechte dieser beiden Zeichen bekommt als letzten Kode-Teil eine 1, das linke eine 0. Diese neue Liste muß wieder sortiert werden. Danach werden wieder die zwei Zeichen mit der geringsten Wahrscheinlichkeit zusammengefaßt und mit einer zusätzlichen Zahl (0 oder 1) im Kode versehen. Dies wird solange fortgesetzt, bis alle Wahrscheinlichkeiten zu der Gesamtwahrscheinlichkeit 1 zusammengefaßt sind.

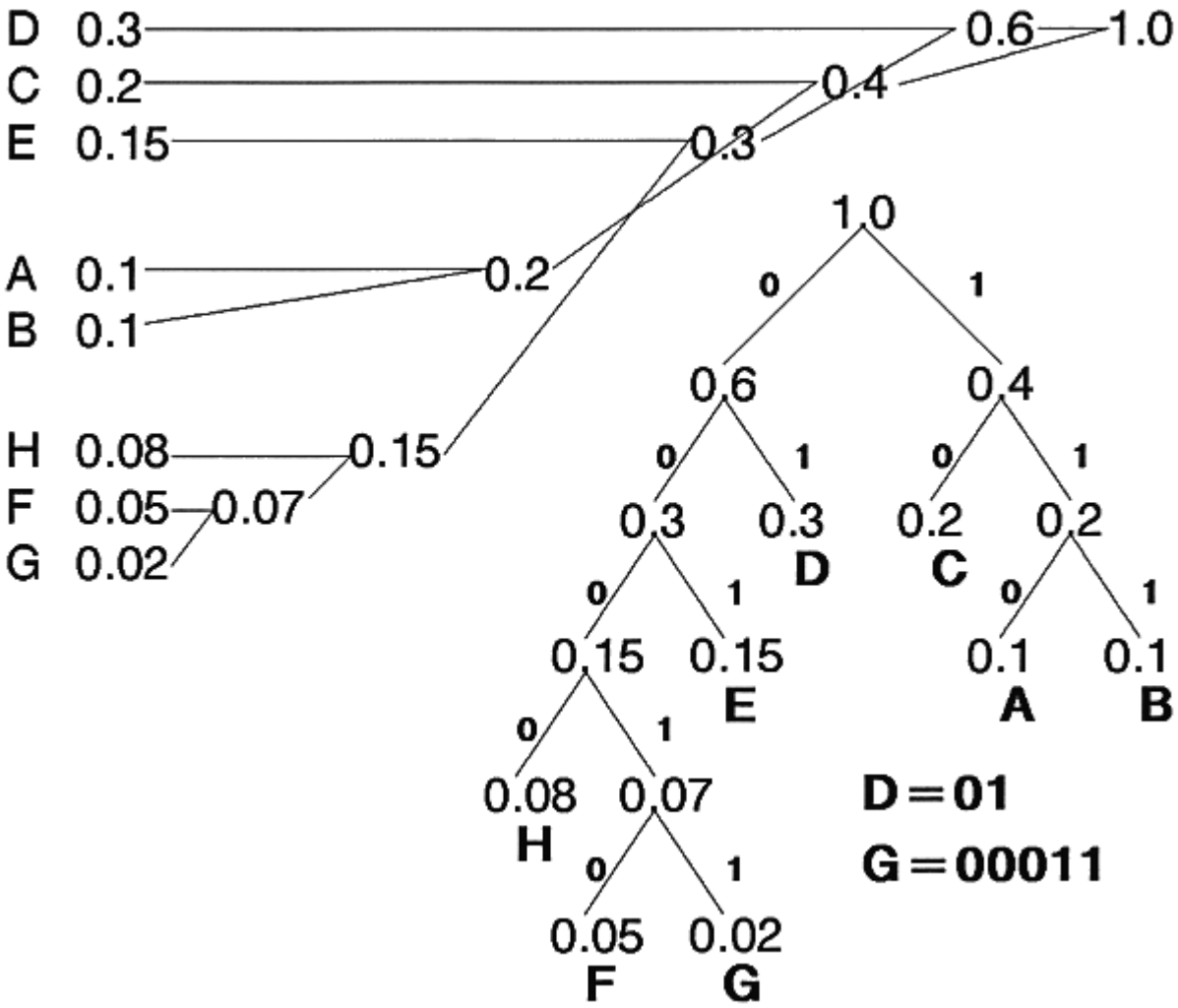


Abbildung 15.2: Erzeugung der Huffman-Kodierung und des zugehörigen Kodebaumes. Bei der Verzweigung nach rechts wird an den Kode eine 1, nach links eine 0 angehängt. Die mittlere Kodelänge in diesem Beispiel beträgt 2.72, die Entropie rund 2.68.

Diese Struktur kann auch im sogenannten Kodebaum dargestellt werden. Die Blätter dieses Binärbaumes sind die einzelnen Zeichen. Die Äste beinhalten die einzelnen Wahrscheinlichkeiten.

Eine adaptive Vorgehensweise kann dieses Kodierungsverfahren optimieren. Dazu ist anfangs die Kodierungstabelle fest vorgegeben. Nach einer gewissen Anzahl von Kodes wird die Tabelle neu berechnet und entsprechend gekennzeichnet an den Empfänger übermittelt. Diese Anpassung kann unter Umständen mehrfach erfolgen.

Bei dem Kompressionsverfahren nach Shannon-Fano [TO92] wird die Tabelle nicht von den Blättern, sondern von der Wurzel aus aufgebaut. Im ersten Schritt werden dazu die Zeichen in zwei Gruppen mit möglichst gleich großen Gesamtwahrscheinlichkeiten aufgeteilt. Die eine Gruppe bekommt als erstes Zeichen des Kodes eine 1, die andere eine 0. Anschließend wird diese Aufteilung für die Zeichen jeder Untergruppe wiederholt, bis jede Gruppe nur noch ein Zeichen enthält. Im Kodebaum entspricht eine Gruppe jeweils einem Zweig.

Neben dem Aufbau der Tabelle ist die Empfindlichkeit gegenüber Störungen einer der großen Nachteile dieser variablen Längenkodierung. Die Störung eines einzelnen Bits führt zur Zerstörung des gesamten nachfolgenden Kodes.

## 15.3 Lauflängen-Kodierung

Bei der allgemeinen *Lauflängen-Kodierung* (*Run Length Coding, RLC*, auch *Run Length Encoding, RLE*) werden nicht die einzelnen Zeichen gespeichert, sondern sich wiederholende Muster werden zusammengefaßt und die Anzahl der Wiederholungen in einem Längenfeld angegeben. Normalerweise beschränken sich die Muster auf einzelne Zeichen [GW87]. Bei Bildern wird diese Kodierungsart meist nur zeilenweise verwendet (eindimensionale Version). Für den Grauwert und die Anzahl wird aus Effizienzgründen jeweils ein Byte benutzt. Diese eindimensionale Version hat den Vorteil, daß sich Störungen nur maximal über eine Zeile bemerkbar machen können. Beim nächsten Zeilenanfang kann der Dekoder wieder neu "synchronisieren". Diese Art der Kodierung wird zum Beispiel bei den FAX-Geräten der Gruppe 3 verwendet.

Bei der zweidimensionalen Version werden zusätzlich statistische Abhängigkeiten zwischen benachbarten Zeilen ausgenutzt. Dabei werden nur die Unterschiede zur vorangegangenen Zeile kodiert übertragen. Die höhere Kompression wird mit einer größeren Störanfälligkeit erkauft.

Eine Lauflängen-Kodierung ist nur dann sinnvoll, wenn mehrere aufeinanderfolgende Zeichen (Bildpunkte) denselben Wert besitzen. Wechselt der Wert von Zeichen zu Zeichen, so benötigen die kodierten Daten doppelt soviel Platz (Zeichen und Anzahl). In Modifikationen der Lauflängen-Kodierung wird daher ein ausgewähltes Zeichen zur Kennzeichnung verwendet, daß ein Lauflängen-Kode folgt. Beim PCX-Format (siehe Anhang C) werden die obersten beiden Bit im Anzahlfeld, das vor dem Zeichenwert gespeichert wird, auf 1 gesetzt. Dadurch ist aber nur maximal eine Anzahl von 64 Zeichen kodierbar. Wird durch die Kodierung keine Komprimierung erzielt, so werden die Zeichen unkodiert gespeichert.

## 15.4 Quadtree-Kodierung

Bei der *Quadtree-Kodierung* von Bildern wird ausgenutzt, daß in den meisten Bildern größere homogene Gebiete, d.h. Gebiete mit gleichem Farbton oder Grauwert existieren. Dazu wird das Originalbild solange rekursiv in jeweils vier Teilflächen aufgesplittet, bis die jeweilige Teilfläche nur noch einen Grauwert bzw. Farbton besitzt. Dieser wird zusammen mit der Flächennummer gespeichert. Für die Teilflächen wird bei quadratischen Bildern auch eine quadratische Form gewählt [GW87].

Aus dieser Information kann das Originalbild fehlerfrei rekonstruiert werden. Für ein Feld werden maximal $log_2(n)$ Stellen benötigt ($n$ = Auflösung), wobei jede Stelle mit $3log_2(n)$ Bit kodiert

werden kann. Daraus läßt sich direkt errechnen, wie groß ein Feld mindestens sein muß, damit sich diese Art der Kodierung lohnt.

Zur Kodierung muß jede durch Aufteilung entstandene Fläche neu auf Homogenität untersucht werden. Dieses bedeutet einen erheblichen Zeitaufwand bei der Kodierung. Bei der Dekodierung hingegen kann das Bild direkt und schnell aufgebaut werden.

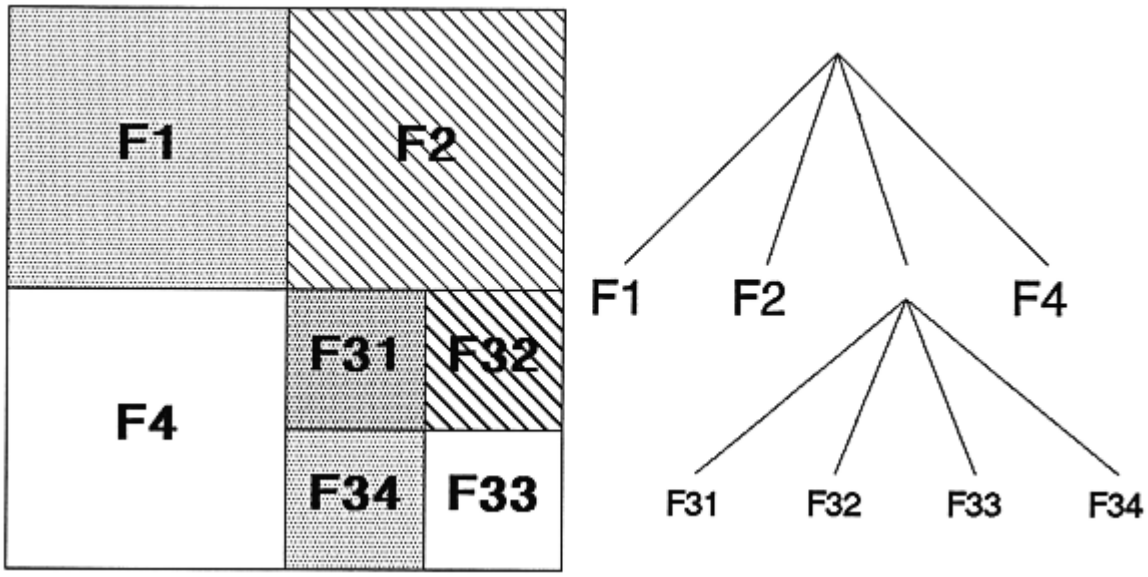


Abbildung 15.3: Quadtree-Kodierung. Alle Punkte eines Quadrats haben denselben Grauwert. Die Nummer des Feldes gibt gleichzeitig die Tiefe der Aufteilung an.

## 15.5 LZW-Kodierung

Die *LZW-Kodierung*, bezeichnet nach den Entwicklern Lempel und Ziv im Jahre 1977 und nach Welch, der 1984 in einer Verbesserung den praktischen Einsatz des Verfahrens beschrieben hat, gehört zu den adaptiven Methoden; die Übersetzungstabelle wird adaptiv während der Kodierung aus den Eingabedaten aufgebaut. Bei diesem Algorithmus erscheinen sich wiederholende Zeichenmuster nur einmal in der Tabelle und bilden sich nur noch durch ihren Tabellenindex ab. Das Verfahren benötigt daher für den Kompressionsvorgang nur einen einzigen Datendurchlauf [Rü89], [Bau91], [Nel91].

Das Verfahren geht davon aus, daß sich Redundanzen innerhalb des Eingabestromes vor allem durch sich wiederholende typische Zeichenketten (sogenannte Phrasen) äußern. Daher wird versucht, solche wiederholt vorkommenden Zeichenketten nicht mehrmals zu senden, sondern sie in der kodierten Ausgabe durch einen Rückbezug zu repräsentieren. Dieser Rückbezug ist ein Index einer parallel zur Kodierung erstellten Tabelle. Ist ein Rückbezug nicht möglich, so müssen die Zei-

chen im Originalformat (als sogenanntes Literal) übertragen und in die Tabelle aufgenommen werden.

Meist wird eine Tabelle mit 4096 Einträgen (12 Bit) verwendet, wobei die ersten 256 Zeichen mit dem normalen ASCII-Kode belegt sind. Die anderen Einträge werden automatisch während der Kodierung erzeugt. Ist die Tabelle voll, so kann sie durch ein spezielles Zeichen gelöscht und mit neuen Einträgen aufgebaut werden. Es ist aber auch möglich, die am wenigsten benutzten Kodes zu entfernen und diese freien Einträge zu verwenden.

Die allgemeine Vorgehensweise beim LZW-Algorithmus läßt sich am besten algorithmisch beschreiben.

```
LZW_Kodierung()
{
  String = Hole_Zeichen(Daten);  /* Zeichen aus dem Eingabestrom */
  while (!EOF) {
      Zeichen=Hole_Zeichen(Daten);
      if ( (String+Zeichen) in der Tabelle)
         String=String+Zeichen;  /* Tabellen-String weiter aufb. */
      else {
         LZW_Kode_Ausgabe(String);
         Tabelle[GroessenIndex++]=String+Zeichen;
         String=Zeichen;
      }
  }
  LZW_Kode_Ausgabe(String);      /* Ausgabe des letzten Kode-Teil*/
}

LZW_Dekodierung()
{
  Kode_1 = Hole_Zeichen(Kodierte_Daten);
  Ausgabe(Kode_1);               /* Erstes Zeichen ist unkodiert */
  while (!EOF) {
      Kode_2=Hole_Zeichen(Kodierte_Daten);
      if ( (Kode_2) nicht in der Tabelle) {
         String=Uebersetzung(Kode_1);
         String=String+Zeichen;
      }
      else {
         String=Uebersetzung(Kode_2);
      }
      Ausgabe(String);
      Zeichen=String[0];         /* paralleler Aufbau der Tabelle*/
      Tabelle[GroessenIndex++]=Kode_1+Zeichen;
      Kode_1=Kode_2;
  }
}
```

| Eingabe | | LZW-Kode (Tabelle) | | Ausgabe |
|---|---|---|---|---|
| **Zeichen** | **ASCII-Kode** | **Tabellenindex** | **zugehörig. String** | **Zeichen** |
| E | 69 | 256 | EI | 69 |
| I | 73 | 257 | IN | 73 |
| N | 78 | 258 | N␣ | 78 |
| ␣ | 32 | 259 | ␣M | 32 |
| M | 77 | 260 | MA | 77 |
| A | 65 | 261 | AL | 65 |
| L | 76 | 262 | L␣ | 76 |
| ␣ | 32 | 263 | ␣E | 32 |
| E | 69 | | | |
| I | 73 | 264 | EIN | 256 |
| N | 78 | 265 | NS | 78 |
| S | 83 | 266 | S␣ | 83 |
| ␣ | 32 | 267 | ␣I | 32 |
| I | 73 | 268 | IS | 73 |
| S | 83 | 269 | ST | 83 |
| T | 84 | 270 | T␣ | 84 |
| ␣ | 32 | | | |
| E | 69 | 271 | ␣EI | 263 |
| I | 73 | | | |
| N | 78 | 272 | INS | 257 |
| S | 83 | 273 | | 83 |

Tabelle 15.1: An einem kleinen Beispiel wird die Vorgehensweise verdeutlicht (␣ steht für ein Leerzeichen):

## 15.6 JPEG-Verfahren

Die Aktivitäten der JPEG (*Joint Photographic Expert Group*), die von der ISO (*International Organization for Standardization*) und der CCITT (*Comité Consultatif International Télégraphique et Téléphonique*) 1984-1987 gegründet wurde, beziehen sich auf die Datenreduktion bei Einzelbildern. Ziel war es, eine Reduktion um den Faktor 10 ohne nennenswert sichtbare Qualitätsverluste zu erhalten, um z.B. Videokonferenzen über langsame Übertragungsmedien (Telefonleitung) zu übertragen [Pet91], [Wal91], CM[92].

Um dies zu erreichen, werden beim *JPEG-Verfahren* unterschiedliche Algorithmen kombiniert. Zur Kodierung eines Farb-Bildes wird im ersten Schritt die Farbinformation von dem RGB-Modell in das YUV-Modell (Luminanz und Chrominanz) umgerechnet. Diese Umrechnung geschieht nach dem CCIR-601-Schema mit folgender Gewichtung der einzelnen Farbanteile:

$$Y = 0.299R + 0.587G + 0.114B$$

$$Cb = 0.1687R - 0.3313G + 0.5B$$

$$Cr = 0.5R - 0.4187G - 0.0813B$$

mit Y=Helligkeit (Luminanz), Cb=Color blueness (Chrominanz; ≈ U), Cr=Color redness (Chrominanz; ≈ V) und R, G, B für Rot, Grün, Blau. Diese Umrechnung alleine bedeutet noch keine Datenreduktion. Wird aber berücksichtigt, daß das menschliche Sehsystem auf Helligkeit empfindlicher reagiert als auf Farbe, so kann mit der Speicherung nur jeweils eines Farbwertes Cb und Cr für ein Feld von 2×2 Bildpunkten eine erste Datenreduktion erreicht werden. Für ein solches Feld werden dann statt 12 Werte nur noch 6, nämlich 4 für die Helligkeit und 2 für die Farbe benötigt. Es erfolgt danach noch eine Quantisierung der Werte. Diese Vorgehensweise, einige Komponenten mit einer geringeren Rate (Y:Cb:Cr = 4:1:1) abzutasten, wird auch als *Subsampling* bezeichnet.

Das so umgerechnete Bild wird dann beim JPEG-Verfahren in Quadrate der Größe 8×8 aufgeteilt. Für jede Komponente Y, Cb und Cr wird dann eine diskrete Kosinus-Transformation (DCT) nach folgender Formel berechnet:

$$C(k,l)=\frac{1}{4}C_k\,C_l\left[\sum_{m=0}^{7}\sum_{n=0}^{7} f(m,n)\cos\frac{(2m+1)k\pi}{16}\cos\frac{(2n+1)l\pi}{16}\right]$$

mit

$$C_k, C_l = \begin{cases} 1/\sqrt{2} & k,l=0 \\ 1 & sonst \end{cases}$$

Durch diese DCT wird das 8×8-Eingangsignal in 64 orthogonale Basis-Signale zerlegt. Jedes Basis-Signal enthält eine der 64 Frequenzen aus dem Spektrum des Eingangssignals. Die Ausgabe der DCT sind nun die 64 Amplitudenwerte dieser Frequenzen.

Wie schon bei den Bildtransformationen deutlich wurde, haben die höheren Frequenzen in der Regel nur eine geringe Amplitude und tragen daher kaum zum Gesamtaussehen des Originalbildes bei. Wird noch eine Quantisierung der Amplitudenwerte durchgeführt, so sind die meisten Werte gleich Null. Diese Quantisierung bestimmt unter anderem den Kompressionsgrad.

Der Amplitudenwert für *C*(0, 0) wird als DC-Koeffizient (*Direct Current*; "Gleichstromanteil") bezeichnet und zeigt an, mit welchem Grad sich die Eingangssignale nicht ändern. Der DC-Anteil ist der Mittelwert über alle 64 Eingangswerte. Die restlichen Amplitudenwerte werden als AC-Koeffizienten (*Alternative Current*; "Wechselstromanteil") bezeichnet. *C*(1, 0) gibt zum Beispiel den Grad der langsamen Änderung (niedrige Frequenz) in horizontaler Richtung an, *C*(7, 7) enthält

den Amplitudenwert der höchsten Frequenz in beiden Richtungen. Die meisten Amplitudenwerte sind Null oder fast Null, so daß es ausreicht, nur den DC-Wert und dessen umgebende AC-Werte für die weiteren Berechnungen zu berücksichtigen.

Wegen der starken Korrelation benachbarter 8×8-Blöcke wird nur für den ersten Block der DC-Anteil und bei nachfolgenden Blöcken nur der Unterschied zum vorherigen übertragen. Der DC-Wert bzw. die Unterschiede werden mit dem Huffman-Kode komprimiert. Die AC-Komponenten werden lauflängenkodiert. Dazu werden zwei Zeichen verwendet. Das erste Zeichen besteht aus 8 Bit. Die oberen 4 Bit sind als Kennzeichnung für eine Lauflängen-Kodierung Null, die unteren 4 Bit geben die Wiederholungen des nachfolgenden Amplitudenwertes an. Zur besseren Auflösung werden für den Amplitudenwert (zweites Zeichen) bis zu 12 Bit verwendet.

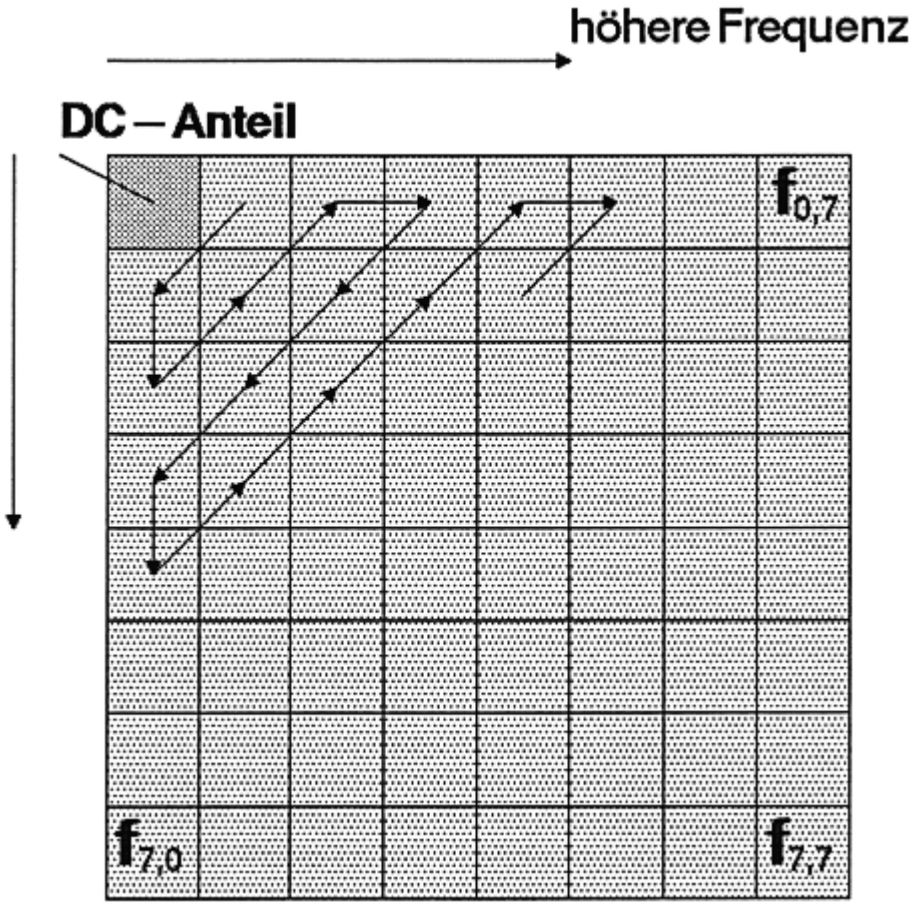


Abbildung 15.4: Die DC-Komponente enthält den Mittelwert über alle 64 Meßwerte. Die AC-Komponenten werden in einem Zickzack-Muster aus dem 8×8-Feld ausgelesen. Die Amplitudenwerte sind dadurch nach der zugehörigen Frequenz sortiert (tiefe Frequenzen zuerst). Aufeinanderfolgende Werte sind fast gleich, was eine bessere Lauflängen-Kodierung ermöglicht.

Der JPEG-Standard definiert die Reduktion von Bildern mit einer Auflösung von bis zu 65536×65536-Bildpunkten. Alle Parameter über die Quantisierung, verwendete Auflösung usw. werden im JPEG-Header abgelegt.

Aufgrund von Untersuchungen an Testbildern kann man folgende Aussagen über Qualität und den visuellen Eindruck von reduzierten Bildern machen:

| | |
|---|---|
| 0.25 – 0.50 Bits/Pixel | annehmbare Qualität |
| 0.50 – 0.75 Bits/Pixel | gute Qualität; oft ausreichend |
| 0.75 – 1.50 Bits/Pixel | sehr gute Qualität; für die meisten Anwendungen ausreichend |
| 1.50 – 2.00 Bits/Pixel | ausgezeichnete Qualität; normalerweise nicht vom Original unterscheidbar |

Die Dekodierung eines nach dem JPEG-Verfahren kodierten Bildes geschieht analog mit der inversen diskreten Kosinus-Transformation.

$$f(m,n)=\frac{1}{4}\left[\sum_{m=0}^{7}\sum_{n=0}^{7} C_k\, C_l\, C(k,l)\cos\frac{(2m+1)k\pi}{16}\cos\frac{(2n+1)l\pi}{16}\right]$$

mit

$$C_k, C_l = \begin{cases} 1/\sqrt{2} & k,l=0 \\ 1 & \textit{sonst} \end{cases}$$

# 15.7 MPEG-Verfahren

Das Ziel der 1988 gegründeten MPEG (*Motion Picture Expert Group*) war es, ähnlich wie bei der JPEG einen Vorschlag für einen Standard zur Komprimierung von Bewegtbildern zu machen. Auch hier stand die Übertragung der Bilder über Telefonleitungen oder ISDN-Leitungen (64 KBit/sec) im Vordergrund. Zu der Videoinformation mußte daher auch zusätzlich Audioinformation berücksichtigt werden [Gal91].

Für dieses Vorhaben mußte eine Kompression von bis zu 200:1 erreicht werden, um mit der maximal möglichen Übertragungsrate von 1.0-1.5 MBit/sec eine Übertragung von Bewegtbildern zu ermöglichen. Gleichzeitig wurde damit dem CD-ROM Rechnung getragen, das mit Datentransferraten von 150 KByte/sec (bei einfacher Geschwindigkeit) in diesen Bereich fällt. Sollen diese Bilder mit der üblichen Bildwiederholfrequenz von 30 Hz (NTSC) oder 25 Hz (PAL) dargestellt werden, so ergibt sich bei einer durchschnittlichen Kompressionsrate von 160:1 eine Auflösung von 352×240 bzw. 352×288 Bildpunkten. Dieses Format wird mit CIF (*Common Intermediate Format*) bezeichnet, die geringere Auflösung von 176×144 Pixel mit QCIF (Quarter-CIF). Die CCITT hat sich für CIF bzw. QCIF als Videoformat und für das Bildtelefon entschieden (CCITT-H.261). Für die Übertragung sind dann mehrere ISDN-Leitungen parallel zu schalten (Px64) [Lio91].

Die eigentliche Kodierung beim MPEG-Verfahren geschieht auch mittels diskreter Kosinus-Transformation (DCT). Hier werden Blöcke der Größe 16×16-Bildpunkten verwendet. Es wird jedoch nicht jedes einzelne Bild mit der DCT kodiert, sondern es werden drei Bildtypen unterschieden:

- I-Bild (*Intra Picture*)
  Dabei handelt es sich um ein komplett mit der DCT komprimiertes Bild. Die kodierte Form enthält die gesamte Information zur Dekodierung und entspricht damit einem normalen JPEG-kodierten Bild.
- P-Bild (*Predicted Picture*)
  Dieses kodierte Bild enthält nur die Unterschiede zum letzten Bild. Auch diese Unterschiede werden mit der DCT kodiert. Das Startbild einer solchen Kette muß natürlich ein I-Bild sein.
- B-Bild (*Bidirectional Prediction Picture*)
  Ein B-Bild wird durch Interpolation zwischen einem vergangenen und zukünftigen Bild der beiden anderen Typen berechnet. Die in dieser Zeit erfolgten Bildänderungen (Bewegungen usw.) werden dazu als gleichmäßig angenommen.

Weitere Komprimierungen können beim MPEG-Verfahren noch durch Abschätzung von Bewegungen und Übertragung des entsprechenden Vektors erfolgen. Für eine schnelle Übersichtsdarstellung können auch Bilder verwendet werden, die nur die DC-Komponenten enthalten.

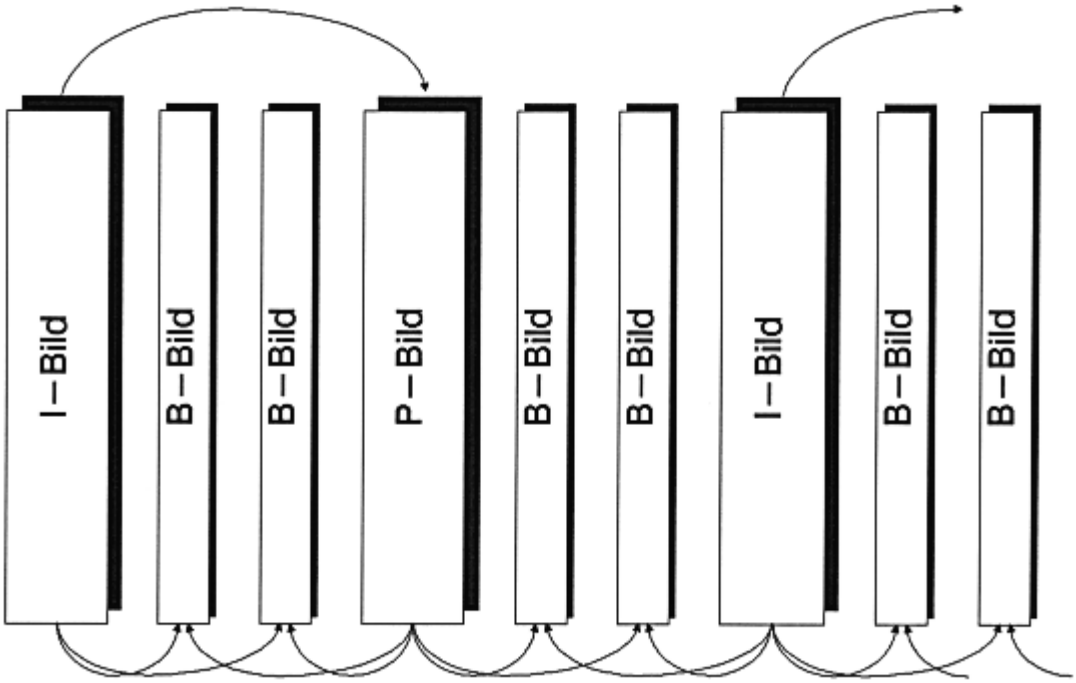


Abbildung 15.5: Beispiel für eine Erzeugung der B-Bilder aus den I- und P-Bildern. Um Bildstörungen durch Übertragungsfehler gering zu halten, wird nach einer Anzahl von P-Bildern immer wieder ein I-Bild gesendet.

Mit diesem MPEG-I-Standard können bei einer Kompression von 200:1 und einer maximalen Übertragungsrate von 1.5 MBit/sec Bilder mit bis zu 360×240 Bildpunkte bewegt dargestellt werden. Die Qualität entspricht dabei fast derjenigen von normalen VHS-Videorekordern. Eine bessere Qualität ist in der MPEG-II-Empfehlung niedergelegt. Bei einer Kompression von bis zu 100:1 und Übertragungsgeschwindigkeiten von 5-10 MBit/sec sollen Auflösungen von bis zu 640×480 Bildpunkten möglich sein. Diese Qualität wäre dann sogar besser als die derzeitige Fernsehqualität.

# Aufgaben

## Aufgabe 1

In einem Bild kommen die folgenden Grauwerte mit der angegebenen Wahrscheinlichkeit vor:

| Grauwert | Wahrscheinlichkeit |
|---|---|
| 0 | 0.35 |
| 46 | 0.05 |
| 50 | 0.1 |
| 120 | 0.05 |
| 128 | 0.15 |
| 177 | 0.1 |
| 193 | 0.05 |
| 255 | 0.15 |

a) Geben Sie für diese Grauwerte eine 3-Bit-Kodierung an.
b) Berechnen Sie die Huffman-Kodierung für diese Werte. Wie groß ist die mittlere Kodelänge und die Entropie?

## Aufgabe 2

Gegeben ist folgendes Bild:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 100 | 100 | 100 | 100 | 80 | 80 | 80 | 80 | 80 |
| 100 | 100 | 100 | 100 | 128 | 92 | 92 | 92 | 92 | 255 |
| 255 | 255 | 100 | 100 | 128 | 92 | 92 | 92 | 80 | 80 |
| 255 | 255 | 255 | 100 | 100 | 70 | 70 | 60 | 50 | 50 |

Geben Sie eine zeilenorientierte Lauflängen-Kodierung (8-Bit Anzahl, 8-Bit Farbe) an. Vergleichen Sie den Speicherbedarf mit dem des unkodierten Bildes.

## Aufgabe 3

Ein Bild mit 256×256 Bildpunkten und 3×8-Bit Farbe soll mit einem vereinfachten JPEG-Verfahren (4:1:1; 12-Bit Tiefe) komprimiert werden. Dazu wird das Bild in 8×8-Bildpunkte große Felder aufgeteilt. Auf diese Felder wird auf den Helligkeitswert (*Y*) eine DCT angewandt. Die Farbwerte (Cb, Cr) werden als Originalwert im 12-Bit Wertebereich gespeichert.

Nur wenige Koeffizienten der DCT sind ungleich von Null. Daher wird eine Quantisierung in 9 Stufen mit 12 Bit Auflösung vorgenommen. Die Wahrscheinlichkeit der einzelnen Stufen gemittelt über alle Felder ist

| Stufe | Wahrscheinlichkeit |
|---|---|
| 1 | 0.4 |
| 2 | 0.2 |
| 3 | 0.08 |
| 4 | 0.08 |
| 5 | 0.06 |
| 6 | 0.05 |
| 7 | 0.05 |
| 8 | 0.04 |
| 9 | 0.04 |

Die einzelnen Stufen werden Huffman-kodiert. Berechnen Sie den Gesamtspeicherbedarf dieses JPEG-komprimierten Bildes und den erzielten Kompressionsfaktor.

# 16. Mustererkennung

Zum Abschluß dieses Buches soll das direkt an die Bildverarbeitung angrenzende Gebiet der *Mustererkennung* angesprochen werden. Aufgrund der aus dem vorverarbeiteten und segmentierten Bild extrahierten Objekte und deren berechneten Merkmale werden die einzelnen Objekte unterschiedlichen Klassen zugeordnet. Diese Klassen werden durch den Anwender oder automatisch durch das System festgelegt.

Bei der unüberwachten Klassifikation werden die Objekte in Mengen mit gleichen Merkmalsausprägungen eingeteilt. Diese Mengen stellen die einzelnen Klassen dar. Die Festlegung des symbolischen Namens der Klassen ("Haus", "Baum", "Buchstabe A" usw.) erfolgt anschließend vom Anwender.

Die überwachte Klassifikation geht von den symbolischen Namen aus und bestimmt in einer Trainingsphase anhand von Prototypen die Merkmale einer jeden Klasse. Aufgrund der gefundenen Merkmale bei der eigentlichen Klassifikation werden die Objekte den einzelnen Klassen mit den bereits vorhandenen symbolischen Namen zugeordnet. Aufgrund der Zuordnung eines Objekts zu einer Klasse macht die überwachte Klassifikation sofort Aussagen wie "Bei diesem Objekt handelt es sich um einen Baum.".

Die meisten Merkmale haben keinen binären Charakter, d.h. die Aussagen über ein Merkmal beschränken sich nicht nur auf "Ist vorhanden" oder "Ist nicht vorhanden". In der Regel wird die Ausprägung eines Merkmals durch einen numerischen Wert $M$ beschrieben, der aus einem Intervall $[0, A]$ stammt. Ist $M = 0$, so ist das Merkmal nicht, bei $M = A$ ist das Merkmal komplett vorhanden. Anhand einer Schwelle muß dann entschieden werden, wie das berechnete Merkmal zu bewerten ist.

Werden mehrere Merkmale zur Klassifikation verwendet, so können diese in einem Merkmalsvektor zusammengefaßt und je nach Relevanz der Einzelnen noch unterschiedlich gewichtet werden. Erst die Gesamtsumme der gewichteten Ausprägungen bestimmt die Klassifikation.

Einige wichtige und einfach zu berechnende Merkmale werden im nächsten Abschnitt beschrieben. Je nach Anwendungsgebiet können diese einzeln oder in einem Merkmalsvektor zusammengefaßt verwendet werden.

## 16.1 Merkmale – Einfache Merkmale

*Merkmale* beschreiben verschiedene Eigenschaften von Objekten. Schon bei der Bildvorverarbeitung und Segmentierung werden indirekt die unterschiedlichsten einfachen Merkmale berechnet. Dazu zählen

- **Umschreibendes Rechteck**
  Als umschreibendes Rechteck wird das das Objekt umschließende Rechteck mit dem kleinsten Flächeninhalt bezeichnet. Zur Vereinfachung wird oft ein achsenparalleles Rechteck erzeugt, das durch einfachen Koordinatenvergleich berechnet wird.

- **Mittlerer Grauwert**
  Schon bei der Kantendetektion und bei der Segmentierung wird der Grauwert eines Bildpunktes zur Berechnung von Objektgrenzen verwendet. Ist die Ausdehnung eines Objekts bekannt, so kann der Grauwert der einzelnen Bildpunkte im Objekt als Merkmal zur Klassifizierung verwendet werden. Um lokale Grauwertvarianzen vernachlässigen zu können, wird der Grauwert gemittelt und nur dieser mittlere Grauwert als Merkmal verwendet.

- **Flächeninhalt**
  Da der genaue Maßstab in der Regel für die einzelnen Bilder unbekannt ist, wird der Flächeninhalt als die Anzahl der Bildpunkte innerhalb der Objektfläche bezeichnet. Schon bei der Segmentierung durch Gebietswachstum oder dem Füllen von umschlossenen Gebieten fällt die Angabe über den Flächeninhalt an.

- **Schwerpunkt**
  Der Schwerpunkt eines Objektes im physikalischen Sinn kann bei der Mustererkennung nicht als Merkmal verwendet werden, da hierfür nicht die notwendigen Informationen (z.B. Dichte) vorliegen. Bei einem Binärbild wird daher als Näherung der Mittelwert der Koordinaten der Objekt-Bildpunkte verwendet. Bei Grauwertbildern erfolgt meist eine Gewichtung der Koordinaten der Objekt-Bildpunkte mit dem korrespondierenden Grauwert.

- **Umfang, Umriß**
  Mit Umfang oder Umriß wird bei diskretisierten Bildern die Anzahl der Randpunkte der Objekte bezeichnet. Beim Freeman-Kettenkode entspricht die Länge der Kette der Anzahl der Randpunkte.

Neben diesen einfachen Merkmalen können noch weitere Merkmale (abgeleitete Merkmale) aus diesen Informationen berechnet werden. Das Ziel ist dabei immer, Merkmale möglichst einfach berechnen zu können und Ergebnisse unabhängig von der Objektlage und Objektgröße zu erhalten.

- **Kompaktheit**
  Als Kompaktheit $K$ wird das Verhältnis von Umfang der Fläche zu ihrem Flächeninhalt bezeichnet.

$$K = \frac{\text{Umfang}^2}{4\pi \ \text{Flächeninhalt}}$$

  Nur bei einem Kreis ist $K = 1$ (kompakteste Figur). Bei allen anderen Objekten ergibt sich ein Wert $K > 1$ [BB91].

Da die Umfangsberechnung ein relativ aufwendiger Rechenschritt ist, wird vereinzelt folgende einfachere Definition benutzt:

$$K_2 = \frac{\text{Fläche umschließendes Rechteck}}{\text{Objekt} - \text{Fläche}}$$

Hier ist $K_2 = 1$, falls es sich bei dem Objekt um ein Rechteck handelt. Bei einem Kreis ergibt sich $K_2 \approx 1.27$. Besonders günstig ist diese Definition der Kompaktheit wenn bekannt ist, welche Objektformen auftreten können. In diesem Fall kann ein Kompaktheitsintervall angegeben werden, in dem die sinnvollen Ergebnisse liegen dürfen (vergleiche Anhang E).

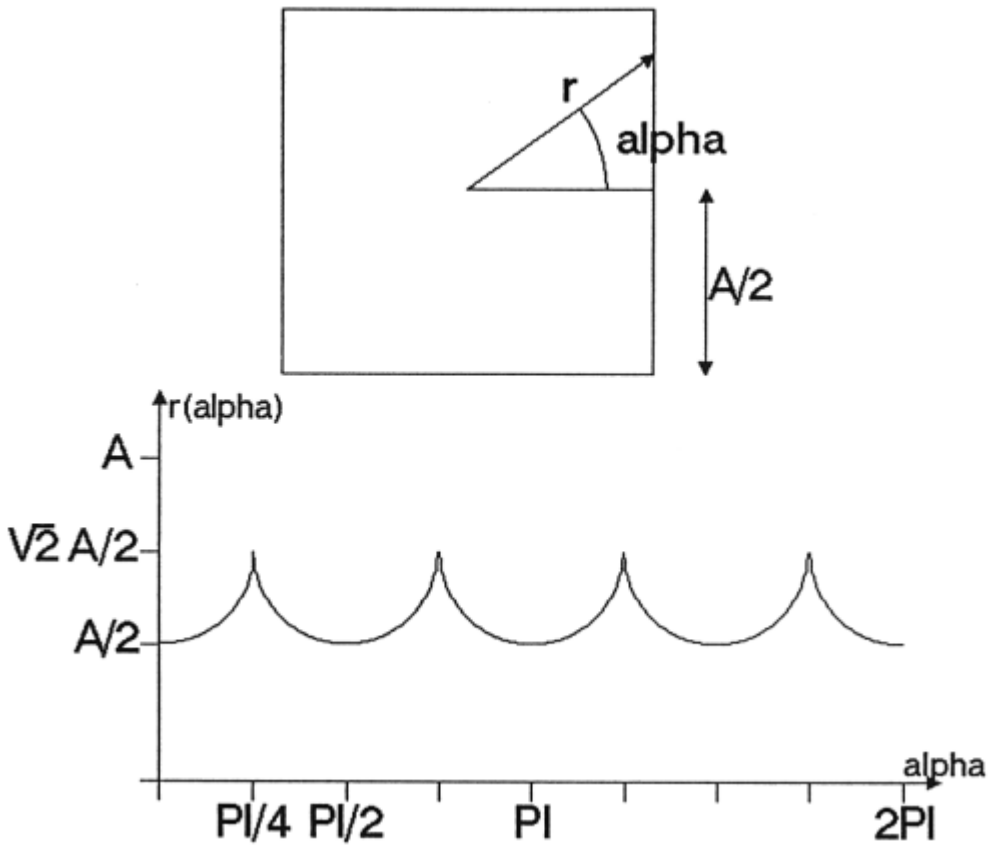


Abbildung 16.1: Der polare Abstand ist der Euklidsche Abstand der Randpunkt zum Schwerpunkt. Die Anzahl der Maxima entspricht der Anzahl der Ecken des Objektes.

- **Polarer Abstand**
  Mit Hilfe des polaren Abstands wird eine lage- und größenunabhängige Beschreibung erzielt. Ausgehend vom Objektschwerpunkt wird in festen Winkelschritten $\Delta\alpha$ der Abstand vom Schwerpunkt zum Objektrand über die Euklidsche Distanz berechnet [GW87]. Dieser Abstand wird normiert auf den Bereich [0, 1] über dem Winkel abgetragen (siehe Abbildung 16.1). Durch die Normierung wird eine Objektgrößenunabhängigkeit erreicht. Mittels einer zyklischen Verschiebung dieser Kurve über den Winkelbereich [0, 360°) bis zum Erreichen einer Normlage kann der Einfluß einer Objektdrehung beseitigt und eine Lageunabhängigkeit

erzielt werden. Die Anzahl der Maxima in der Kurve entspricht der Anzahl der Ecken des Objekts. Die Größe der lokalen Maxima gibt gleichzeitig eine Auskunft über die Ausprägung der Ecken. Diese Informationen können als zusätzliche Merkmale verwendet werden.

- **Lage, Orientierung**
  Über die Lage der lokalen Maxima beim polaren Abstand läßt sich eine Aussage über die Lage des Objekts machen und somit auch der Winkel der Achse des größten/kleinsten Trägheitsmoments bestimmen.

- **Exzentrizität**
  Unter der Exzentrizität $E$ wird das Verhältnis zwischen maximalem und minimalem polaren Abstand verstanden. Ein Kreis besitzt die Exzentrizität $E = 1$. An welchen Stellen und wie oft die Objektform von einem Kreis abweicht, wird durch den Wert $E$ nicht bestimmt.

- **Symmetrie**
  Die Form des umschließenden Rechtecks, die Lage des Schwerpunkts, das Aussehen des Kettenkodes für den Objektrand, die Kurve des polaren Abstands usw. beinhalten jeweils Informationen über die Form und somit auch über eventuelle Symmetrieeigenschaften des untersuchten Objekts (siehe Abbildung 16.1).

Selbstverständlich existieren je nach Anwendung der Mustererkennung noch eine Vielzahl weiterer Merkmale. Diese sind in der Regel aber sehr anwendungsspezifisch (vergleiche Abschnitt 16.4). Beispielhaft seien hier aus dem großen Gebiet der Zeichenerkennung (*Optical Character Recognition, OCR*) einige Merkmale von Buchstaben genannt:

- **Löcher**
  Unter Löchern werden bei der Zeichenerkennung komplett umschlossene Gebiete verstanden. Zum Beispiel besitzen die Buchstaben "B" und "8" zwei Löcher, "P" und "6" jeweils ein Loch, "G" und "2" kein Loch.

- **Endungen**
  Als Endungen werden die frei endenden Buchstabenteile bezeichnet, sofern die Zeichen ohne Serifen dargestellt wurden. Beim Buchstaben "B" gibt es keine Endungen, bei "P" eine Endung, bei "M" zwei, bei "E" drei und bei den Buchstaben "H", "K" und "X" vier Endungen.

- **Kreuzungen**
  Unter Kreuzungen werden die Schnittpunkte der in einem Buchstaben enthaltenen Strecken verstanden. Oft endet ein Teil an dieser Stelle, so daß keine richtige Kreuzung entsteht. Trotzdem werden solche Stellen noch mitgezählt. Das "O" besitzt keine Kreuzung, das "L" eine, das "F" zwei Kreuzungen.

- **Achsenparallele Linien**
  Werden in dem zu untersuchenden Zeichensatz keine geneigten Schriften verwendet, so kann die Anzahl der achsenparallelen Linien als zusätzliches Merkmal verwendet werden. Der Buchstabe "S" besitzt keine achsenparallele Linie, das "R" eine vertikale, das "Z" zwei horizontale, das "H" zwei vertikale und eine horizontale Linie usw.

Selbstverständlich ist diese Auflistung der Merkmale nicht vollständig. Die einzelnen Merkmale hängen stark vom verwendeten Zeichensatz ab.

### 16.1.1 Verwendung der Merkmale

Wie schon zu Beginn dieses Kapitels erwähnt, ist das Ziel der Zuordnung von Merkmalen zu den einzelnen Objekten die Unterteilung der Objekte in disjunkte Mengen (Klassen). Dies wird besonders deutlich bei der Zeichenerkennung. Hier sollte jeweils nur eine Zeichenart in einer Menge vorhanden sein. Ähnliche Zeichen wie "B" und "8" sollen aufgrund ihrer Merkmale getrennt, d.h. unterschieden werden.

Gerade bei der Zeichenerkennung ist diese eindeutige Zuordnung aufgrund der Variationen der Zeichen und der vielen Schriftarten oft nicht möglich. In diesem speziellen Fall können fehlende Buchstaben durch die Suche des gefundenen Teil-Wortes in einem Wörterbuch (Datenbank) ergänzt werden.

Allgemein sollten jedoch immer so viele Merkmale zur Identifizierung herangezogen werden, daß eine eindeutige Aufteilung und Zuordnung gewährleistet ist. Dies sollte mindestens bei den Referenzobjekten in der Trainingsphase der Fall sein.

Die $N$ Merkmale aus einer Trainingsphase spannen einen $N$-dimensionalen Merkmalsraum auf. Es sollten dabei nicht alle möglichen Merkmale verwendet werden, sondern nur so viele und diejenigen, die eine eindeutige Unterscheidung ermöglichen.

Da bei der Klassifikation der Objekte aufgrund der berechneten Merkmale die einzelnen Merkmale nicht immer so ausgeprägt sind wie bei den Referenzobjekten in der Trainingsphase, muß in Zweifelsfällen eine Entscheidung darüber getroffen werden, zu welcher Klasse ein Objekt gehört. In der Regel wird in solchen Fällen eine geometrische Klassifikation durchgeführt, d.h. ein Objekt wird der nächstgelegenen Klasse zugeordnet. Die Entfernung wird auch hier durch die Euklidsche Distanz im Merkmalsraum bestimmt. Eine Gewichtung der einzelnen Entfernungen in Abhängigkeit der Relevanz eines Merkmals für eine genauere Klassifizierung ist denkbar.

## 16.2 Hough-Transformation

Das Grundprinzip der *Hough-Transformation* ist in einer Patentschrift von 1962 niedergelegt [Hou62]. Dort beschreibt P.V.C. Hough die nach ihm benannte Transformation. In [DH72] wurde

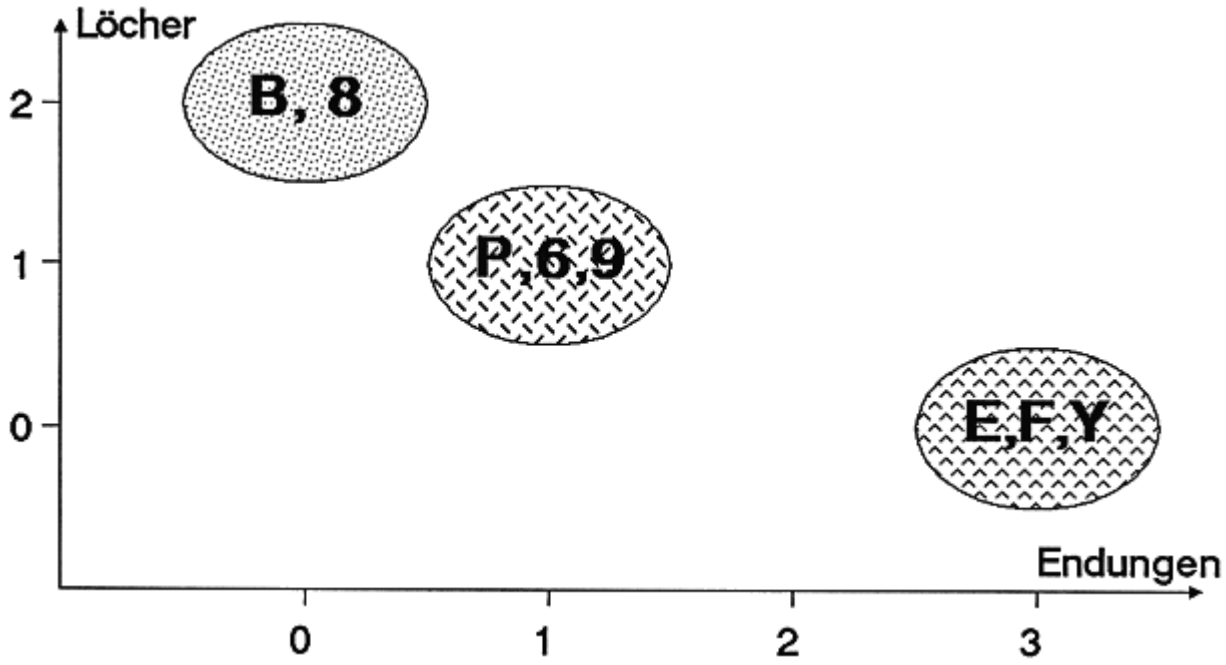


Abbildung 16.2: Beispiel für einen zweidimensionalen Merkmalsraum. Die Anzahl der verwendeten Merkmale läßt noch keine Aussage über die Anzahl der unterscheidbaren Klassen zu. In diesem Beispiel werden mit zwei Merkmalen drei disjunkte Klassen unterschieden.

die Hough-Transformation zur Erkennung von Geraden und – erweitert in [BB82] (*Verallgemeinerte Hough-Transformation*) – zur Erkennung von Kreisen, Ellipsen usw. verwendet.

Ausgehend von einem binärisierten und skelettierten Kantenbild ordnet die Hough-Transformation jedem Kantenpunkt in diesem Binärbild die Geraden (Geradenbüschel) zu, die durch diesen Punkt laufen. Jede Gerade ist durch zwei Parameter spezifiziert, z.B. durch Steigung und Ordinaten-Abschnitt. Den Parameter-Paaren der Geraden entspricht eine Kurve im sogenannten Hough- oder Parameter-Raum. Im Fall der Parametrisierung durch Anstieg und Ordinaten-Abschnitt handelt es sich bei dieser Kurve um eine Gerade.

$$y = ax + b$$

Die Geraden des Geradenbüschels durch den Punkt $(x_1, y_1)$ müssen die Gleichung

$$b = -x_1 a + y_1$$

erfüllen. Im Parameter-Raum (*ab*-Raum; Hough-Raum) beschreibt diese Gleichung für ein festes Wertepaar $(x_1, y_1)$ genau eine Gerade. Umgekehrt korrespondiert jedes Wertepaar $(a_1, b_1)$ im Parameter-Raum mit genau einer Geraden im Bild-Raum. Aus diesem Grund wird die Hough-Transformation auch oft mit *Point-to-Curve-Transformation* oder *Line-to-Point-Transformation* bezeichnet.

Ein anderer, zweiter Punkt $(x_2, y_2)$ (unterschiedlich von $(x_1, y_1)$) beschreibt im Parameter-Raum eine andere Gerade, die die erste im Punkt $(a, b)$ schneidet. Das Wertepaar $(a, b)$ bestimmt genau

eine Gerade im Bild-Raum, mit der Steigung *a* und dem Ordinaten-Abschnitt *b*. Auf dieser Geraden liegen die beiden Punkte ($x_1$, $y_1$) und ($x_2$, $y_2$). Auch alle weiteren Punkte auf dieser Geraden korrespondieren mit Geraden im Parameter-Raum, die durch den Punkt (*a, b*) verlaufen (siehe Abbildung 16.3).

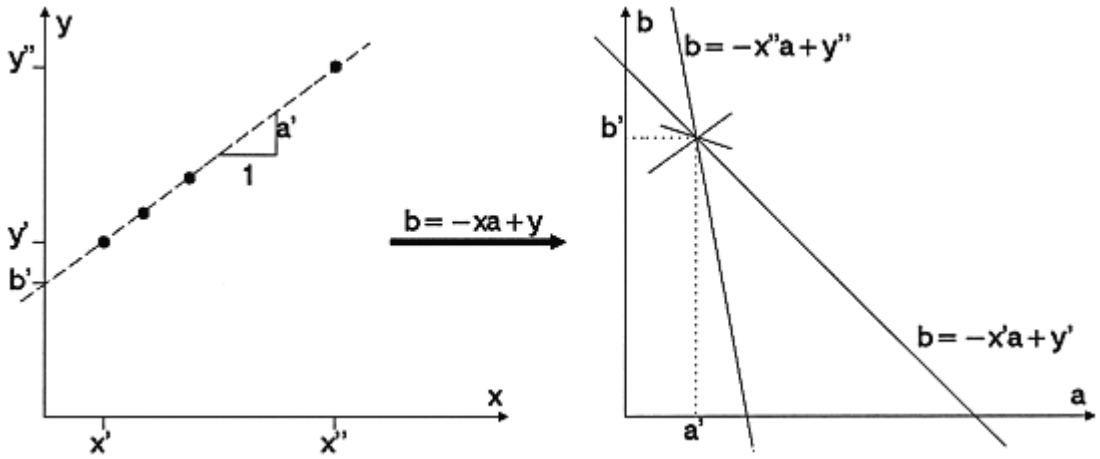


Abbildung 16.3: Die Hough-Transformation ordnet jedem Punkt im Bild-Raum eine Kurve im Parameter-Raum zu. Schneiden sich mehrere Kurven im Parameter-Raum in einem Punkt, so wird dadurch eine Gerade im Bild-Raum spezifiziert.

Damit bei der Schnittpunktsuche im Parameter-Raum keine größeren Gleichungssysteme gelöst werden müssen, wird wie folgt vorgegangen. Zuerst wird der Parameter-Raum diskretisiert. Diese Aufteilung in einzelne Elemente (ähnlich den Bildpunkten in einem digitalisierten Bild) wird mit *Kachelung* bezeichnet, die einzelnen Zellen mit *Akkumulator* oder kurz *Akku*. Die Akkus zählen, wieviele Kurven bei dem Raster im Parameter-Raum durch sie hindurch verlaufen.

Bei der Transformation eines Punktes aus dem Bild-Raum in eine Kurve im Parameter-Raum werden alle Akkus entlang des Verlaufs der Kurve inkrementiert. Schneiden sich zwei Kurven in einem Punkt (Akku) im Parameter-Raum, so wird der Akku an dieser Stelle zweimal inkrementiert. Der Wert des Akkus gibt demnach direkt an, wieviele Kurven sich an dieser Stelle im Parameter-Raum schneiden bzw. wieviele Punkte auf einer möglichen Geraden im Bild-Raum liegen. In Abbildung 16.4 ist ein Ausschnitt der Akku-Werte aus dem Parameter-Raum dargestellt. Deutlich sind lokale Maxima zu erkennen, die mit Geraden im Bild-Raum korrespondieren.

Selbstverständlich geht durch die Diskretisierung des Parameter-Raums Information über die genaue Position der Geraden im Bild-Raum verloren. Je weniger Akku-Zellen der Parameter-Raum besitzt, desto ungenauer wird die Bestimmung. Unterschiedliche Geraden im Bild-Raum entsprechen dann einer Akku-Zelle im Parameter-Raum. Wie fein die Diskretisierung des Parameter-Raums bei einer vorgegebenen Größe des Bild-Raumes sein muß, damit alle im Bild-Raum darstellbaren Geraden noch im Parameter-Raum unterscheidbar sind, wurde in [Ris89] neben der Ausnutzung des Parameter-Raums genauer untersucht. Die Feinheit der Diskretisierung ist natürlich abhängig von der gewählten Geraden-Parametrisierung. Oft wird einfach eine zur Anzahl der Bildpunkte identische Anzahl von Akku-Zellen verwendet.

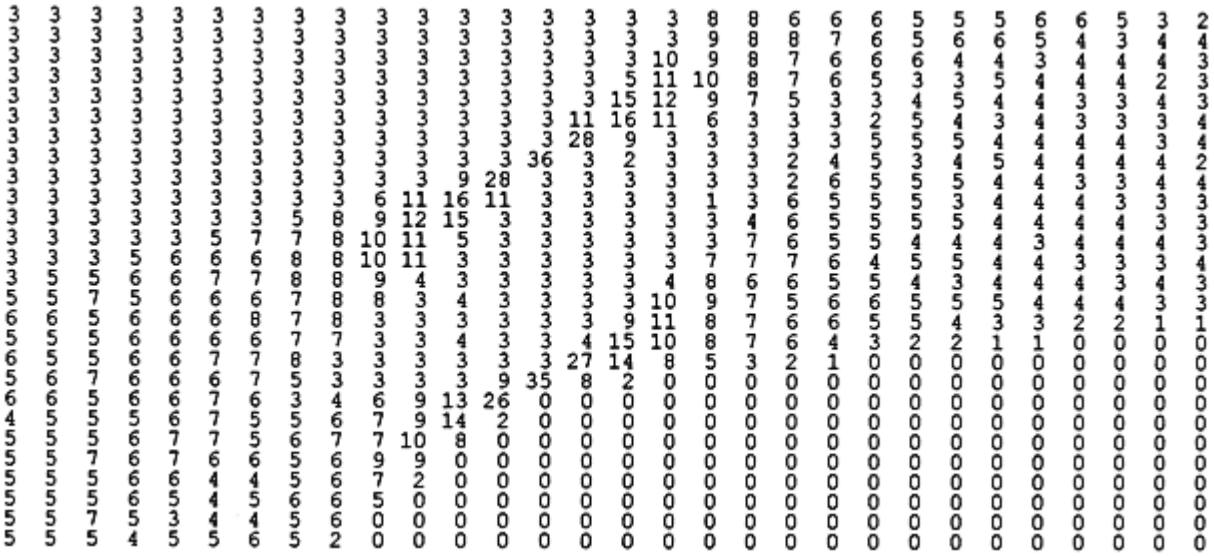

Abbildung 16.4: Ein Ausschnitt aus dem Parameter-Raum. Die Zahlen repräsentieren die Inkremente der einzelnen Akku-Zellen. Die lokalen Maxima korrespondieren mit Geraden im Bild-Raum. Bei der Darstellung des Parameter-Raums als Bild repräsentieren die Graustufen die einzelnen Akku-Werte.

Die oben beschriebene Geraden-Parametrisierung mit Steigung und Ordinate-Abschnitt führt zu Problemen bei fast vertikalen Linien. Die klassische Vorgehensweise (Transformation in nur einen Parameter-Raum, Hough-Raum) wird daher geändert und um einen zweiten Parameter-Raum, den sogenannten *Twin-Hough-Raum* erweitert [Bil87]. Der ursprüngliche Hough-Raum wird dann mit den Geraden mit einer Steigung zwischen –45° und 45°, gefüllt, der Twin-Hough-Raum mit den Geraden mit einer Steigung zwischen 45° und 135°. Diese Unterscheidung wird durch eine Drehung des Bildes um 90°, erreicht. Bei einem quadratischen Bild ($N \times N$) kann dies durch die folgende einfache Koordinatentransformation erfolgen:

$$x' = N - y$$

$$y' = x$$

Die Detektion der gesuchten Geraden im Parameter-Raum beschränkt sich nach der Transformation auf die Suche nach lokalen Maxima. Dies wurde schon bei der Kantendetektion (siehe Kapitel 11) ausführlich behandelt. Ist die Mindestlänge bzw. die Mindestanzahl von Punkten auf einer gesuchten Geraden bekannt, so kann diese Größe als Schwellwert für den Parameter-Raum verwendet werden.

Wurden viele Punkte in den Parameter-Raum transformiert, so kann es aufgrund der Diskretisierung zu einer Verschmierung und nicht eindeutigen Lokalisierung der Maxima kommen. Bei der Anwendung eines Schwellwertverfahrens ergeben sich in solchen Fällen oft ganze Plateaus (*Cluster*). Als eigentliche Position des lokalen Maximums wird dann meist der Schwerpunkt dieser Fläche verwendet.

Als besonders robust gegenüber solchen Ungenauigkeiten hat sich die adaptive Hough-Transformation (*AHT*) erwiesen [RS88]. Hier wird nicht eine feste globale Schwelle zur Bestimmung der lokalen Maxima verwendet, sondern es wird zuerst das globale Maximum im Parameter-Raum bestimmt. Die Auswirkungen der auf der zugehörigen Geraden im Bild-Raum liegenden Punkte werden aus dem Parameter-Raum entfernt, d.h. die Akku-Zellen entlang der einzelnen Kurven werden dekrementiert. Damit wird die Wahrscheinlichkeit des Entstehens von Phantom-Geraden durch falsche Schnittpunkte vermindert. Nach der Detektion dieser ersten Geraden und der Beseitigung ihrer Auswirkungen im Parameter-Raum wird das neue globale Maximum gesucht und analog verfahren. Dieser Vorgang wird solange fortgesetzt, bis eine Schwelle (Mindestgröße für ein globales Maximum) unterschritten wird. Wie Versuche gezeigt haben [Ris89], ist diese adaptive Vorgehensweise sehr robust gegenüber Störungen wie Rauschen oder dicht beieinanderliegenden Geraden.

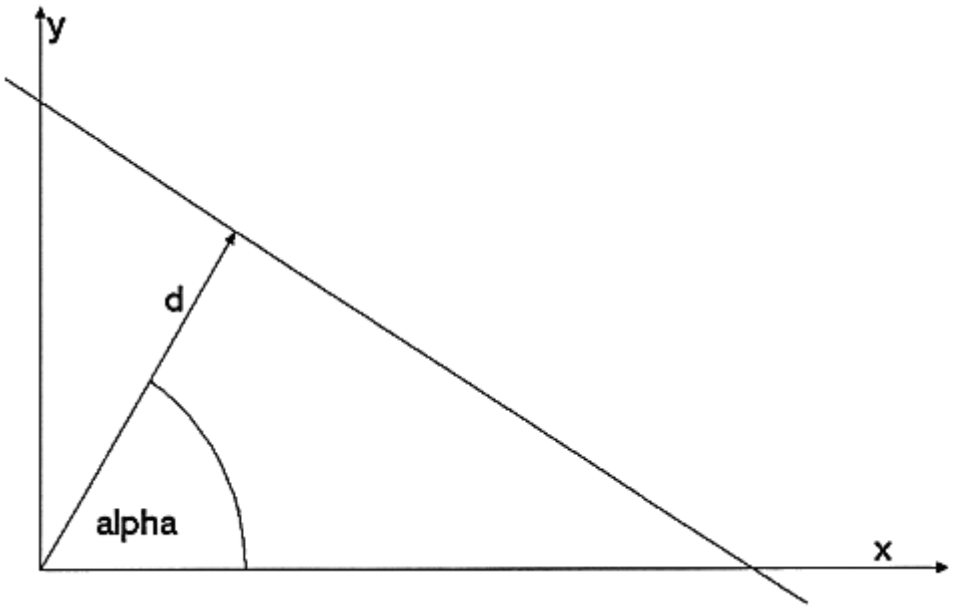


Abbildung 16.5: Die Parametrisierung einer Geraden kann auch über den Abstand *d* zum Ursprung und dem Winkel $\alpha$ zur x-Achse erfolgen. Die Kurve im Parameter-Raum ist dann sinusförmig.

Selbstverständlich sind neben der Parametrisierung durch die Steigung und den Ordinaten-Abschnitt noch andere Parametrisierungen möglich. Die weit verbreitetste ist die in der Hesseschen Normalform (Polarkoordinaten) mit dem Abstand *d* zum Ursprung und dem Winkel $\alpha$ zur x-Achse (siehe Abbildung 16.5).

$$d = x \cos(\alpha) + y \sin(\alpha)$$

Die Kurve im Parameter-Raum ist in diesem Fall sinusförmig. Daher muß dieser Parameter-Raum für dieselbe Geraden-Auflösung im Bild-Raum eine andere Größe haben.

In den folgenden Tabellen sind noch weitere Geraden-Parametrisierungen, die Größe des Parameter-Raums bei einer Bildgröße von $N \times N$ , die Gleichungen für Hin- und Rücktransformation sowie die Ausnutzung des Parameterraumes aufgelistet (aus [Ris89]).

| Art der Parametrisierung | Hin-Transformation | Rück-Transformation |
|---|---|---|
| Steigung *m* und Ordinaten-Abschnitt *c* | $c = -xm + y$ | $y = mx + c$ |
| Steigungswinkel α und Ordinaten-Abschnitt *c* | $c = (-\tan(\alpha))x + y$ | $y = (\tan(\alpha))x + c$ |
| Steigungswinkel α und Abstand d zum Ursprung | $d = x\cos(\alpha) + y\sin(\alpha)$ | $y = (d - x\cos(\alpha))/\sin(\alpha)$ |
| Steigung *a* und Ordinaten-Abschnitt *b* mit Twin-Hough | $b = -xa + y$ | $y = ax + b$ |

Tabelle 16.1: Verschiedene Parametrisierungen der Geraden im Bild-Raum. Je nach Art der Parametrisierung resultieren daraus unterschiedliche Kurvenformen im Parameter-Raum.

Die Größe des Parameter-Raumes und die Auflösung in den beiden Parameter-Achsen bestimmt die Unterscheidbarkeit von Geraden im Bild-Raum. Die gröbste Unterteilung des Parameter-Raumes, in der den Parametern verschiedener im Bild-Raum darstellbarer Geraden immer verschiedene Akku-Zellen entsprechen, wird in [RS88] mit *hifi-Quantisierung* bezeichnet.

Bei der Verwendung von Hough- und Twin-Hough-Raum für ein Bild der Größe $N \times N$ ergibt sich für den Anstieg *b* der Geraden eine Unterteilung in Schritten von

$$\Delta_{hifi}(b) = \frac{1}{N(N-1)}$$

sowie für die Unterteilung des Ordinaten-Abschnitts *a*

$$\Delta_{hifi}(a) = \frac{1}{(N-1)(N-2)}$$

Bei einem Bild der Größe von 256×256 Bildpunkten ergibt sich damit eine Größe der beiden Hough-Räume von jeweils (2*64770)×(3*256*65280) Akku-Zellen. Da in der Praxis aber nicht beliebige Steigungen von Geraden im Bild-Raum vorkommen bzw. das Zusammenfallen von dicht beeinanderliegenden Geraden mit geringer Steigungsdifferenz meist erwünscht ist, reicht eine wesentlich gröbere Aufteilung des Parameter-Raumes aus.

| Art der Parametrisierung | Größe des Parameter-Raums | Ausnutzung des Parameter-Raums |
|---|---|---|
| Steigung *m* und Ordinaten-Abschnitt *c* | $[-N, N]\times[-(N\text{-}1)N, N^2]$ $\cup$ ($N$+1) vertikale Linien | $\lim_{N\to\infty}$ N+1/4N-2 = 25% |
| Steigungswinkel $\alpha$ und Ordinaten-Abschnitt *c* | $[-\arctan(N), \pi/2]\times[-(N\text{-}1)N, N^2]$ $\cup$ ($N$+1) vertikale Linien | $\lim_{N\to\infty}$ (Ausnutzung) = 0% |
| Steigungswinkel $\alpha$ und Abstand *d* zum Ursprung | $[0, \pi - \arctan(1/N)2]\times$ $[-N^2/\sqrt{1+N^2}, \sqrt{2}\,N]$ | $4/\pi(1+\sqrt{2}\approx 52\%$ |
| Steigung a und Ordinaten-Abschnitt *b* mit Twin-Hough | $[-1, 1]\times[-N, 2N]$ je Parameter-Raum | jeder Parameter-Raum zu 50% |

Tabelle 16.2: Die Angabe "Größe des Parameter-Raums" macht nur Aussagen über den kleinsten bzw. größten Wert im Parameter-Raum. Wie fein die Aufteilung der beiden Achsen sein muß, wird damit noch nicht festgelegt.

### 16.2.1 Merkmale im Hough-Raum

Wie die obigen Untersuchungen gezeigt haben, liefert die Parametrisierung mittels Steigung und Ordinaten-Abschnitt in der Kombination mit dem Hough-/Twin-Hough-Raum die beste Ausnutzung des Parameter-Raums. Für die Inkrementierung der Akku-Zellen entlang der Geraden im Parameter-Raum kann z.B. der Bresenham-Algorithmus [NS81] verwendet werden. Dadurch ergibt sich eine einfache und schnelle Berechnung der Hough-Transformation. Die Lage der lokalen Maxima (Cluster) im Parameter-Raum spiegeln bestimmte Merkmale der korrespondierenden Geraden im Bild-Raum wieder [BW84].

Aufgrund der Transformation in zwei Parameter-Räume werden die möglichen Geraden im Ausgangsbild schon in zwei Klassen unterteilt; die mit einer Steigung von –45° bis 45° und die zwischen 45° und 135°.

Lage und Anzahl der lokalen Maxima (Cluster) im Parameter-Raum geben weiterhin Informationen über die Geometrie der Geraden (Linien, Kanten) im Bild-Raum (siehe Abbildung 16.6). Bei der Parametrisierung mittels Steigung und Ordinaten-Abschnitt ist dies:

- **Anzahl der lokalen Maxima**
  Die Anzahl der im Bild-Raum sichtbaren Linien entspricht der Anzahl der lokalen Maxima im Parameter-Raum. Die Maxima von kollinearen Linien überlappen sich im Parameter-Raum und müssen daher mehrfach gezählt werden.

- **Parallele Linien**
  Die Maxima im Parameter-Raum von im Bild-Raum parallelen Linien liegen auf derselben *a*-Koordinate.

- **Ordinaten-Abschnitt**
  Geraden mit demselben Ordinaten-Abschnitt haben ein Maximum im Parameter-Raum auf derselben *b*-Koordinate.

- **Schnittpunkt von Geraden**
  Schneiden sich *n* Geraden im Bild-Raum in einem Punkt, so entsteht eine kollineare Anordnung von *n* lokalen Maxima (Cluster) im Parameter-Raum.

- **Kanten, Ecken**
  Befinden sich im Parameter-Raum *n* kollineare Maxima und schneiden sich die Geraden durch diese *n* kollinearen Anordnungen in einem gemeinsamen Maximum, so entspricht dies einer gemeinsamen Geraden (Kante) auf der an *n* Stellen andere Geraden schneiden.

Mit Hilfe dieser berechneten Merkmale ist es nun möglich, eine Klassifikation durchzuführen und bestimmte Muster bzw. Objekte im Bild zu erkennen. Da sich die hier beschriebene Hough-Transformation nur zur Erkennung von Geraden eignet, sollten die Objekte im Bild geradlinig begrenzt sein. Bei den Bildern kann es sich auch um Aufnahmen von 3D-Objekten wie z.B. Polyeder handeln. Aufgrund der begrenzten Anzahl von Ansichten dieser Körper kann aus der Menge der sichtbaren Kanten und Ecken und ihrer Lage zueinander auf den zugehörigen Polyeder geschlossen werden. Auch eine teilweise Verdeckung der Objekte ist dabei erlaubt [WB86]. Handelt es sich noch um eine relativ geringe Anzahl unterschiedlicher 3D-Objekte, die erkannt werden müssen, so reicht in der Regel eine einzige 2D-Ansicht zur Klassifizierung aus.

Ist es möglich, das Gebiet, in dem sich das gesuchte Objekt befindet, näher zu bestimmen, z.B. durch Berechnung des umschließenden Rechtecks (*Region-of-Interest, ROI*), so kann die Ausprägung der lokalen Maxima erhöht und damit die Genauigkeit und Robustheit gegenüber Störungen weiter verbessert werden.

Obwohl die Hough-Transformation leicht durchzuführen ist, die Suche nach den lokalen Maxima mit bekannten Verfahren aus der Kantendetektion durchgeführt werden kann und die Methode sehr robust gegenüber Störungen wie Rauschen, mehrere Pixel breite Kanten/Geraden usw. ist, besteht doch ein entscheidender Nachteil; die Information über Anfangs- und Endpunkt der Linien im Bild-Raum geht verloren. Die Hough-Transformation liefert als Ergebnis nur Geraden, auf denen die korrespondierenden Original-Strecken liegen.

Durch Vergleich der unter der berechneten Geraden liegenden Punkte im Bild-Raum kann eventuell eine Information über den Anfang und das Ende der Linie zurückgewonnen werden. Dabei müssen aber kleinere Lücken überbrückt werden, was zur Schwierigkeit der Unterscheidung zwischen einer Unterbrechnung der Linie und einer wirklichen Lücke führt. Einer der großen Vorteile der Hough-Transformation, nämlich das automatische Schließen von Lücken wird damit wieder zunichte gemacht.

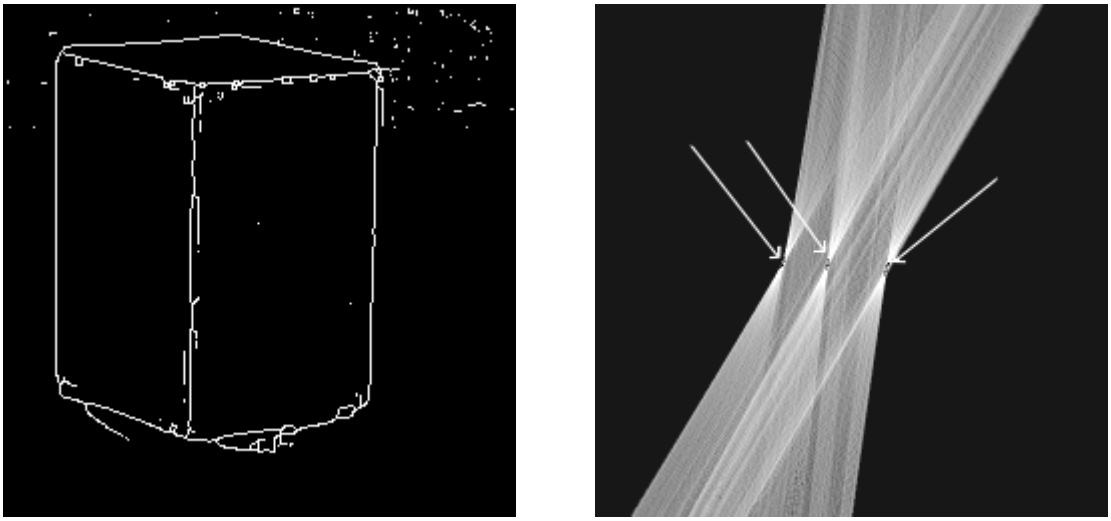

Abbildung 16.6: Das Ergebnis der Hough-Transformation eines skelettierten Binärbildes (links) ist hier am Beispiel des Twin-Hough-Raumes (rechts) dargestellt. Die Werte der Akku-Zellen werden dabei als Graustufen visualisiert. Wegen des besseren Kontrastes ist der Bereich mit dem lokalen Maximum (Pfeil) dunkel dargestellt.

Die Erkennung von Kreisen, Ellipsen usw. mit Hilfe der Hough-Transformation ist mittels einer entsprechenden Parametrisierung ebenfalls möglich [BB82]. Wegen des wesentlich höheren Rechenaufwandes und der großen, mehrdimensionalen Parameter-Räume werden diese Ansätze in der Praxis jedoch selten verfolgt.

## 16.3 Fourier-Deskriptoren

Schon bei der Fourier-Transformation (siehe Kapitel 8) wurde deutlich, daß sich eine Verschiebung im Ortsbereich nur auf die Phase der Fourier-Transformierten, nicht aber auf das Fourier-Spektrum auswirkt. Diese translatorische Invarianzeigenschaft kann bei der Untersuchung von Objektkonturen im Frequenzbereich ausgenutzt werden.

Für die Untersuchungen müssen die Koordinaten $(x, y)$ der Konturpunkte des zu untersuchenden Objektes aus dem Binärbild extrahiert werden. Die Abtastung des Objektrandes erfolgt dabei in mathematisch positivem Umlaufsinn. Der Startpunkt kann beliebig gewählt werden. Werden die Koordinatenwerte $(x, y)$ als komplexe Zahlen $x + iy$ aufgefaßt, so bekommt man bei $N$ Konturpunkten eine Folge $f(i)$ mit $0 \leq i < N$ von komplexen Zahlen. Durch die Anwendung der diskreten Fourier-Transformation auf diese Werte erhält man eine eindeutige Abbildung $F(k)$ mit $0 \leq k < N$ der Kontur im Ortsfrequenzraum. Die Folge von komplexen Zahlen aus den Konturpunktkoordinaten wird dabei als eine Periode einer periodischen Funktion angesehen.

Die aus dieser Transformation resultierenden Fourier-Koeffizienten liefern Informationen über die Objektform [Wah84]. Beispielsweise lassen große Amplitudenwerte bei hohen Frequenzen auf abrupte Konturverläufe schließen.

Durch eine Normierung der Fourier-Koeffizienten kann eine Größen- und Lageunabhängigkeit erreicht werden. Die normierten Fourier-Koeffizienten werden als *Fourier-Deskriptoren* bezeichnet. Diese Fourier-Deskriptoren können somit als Merkmale bei einer Klassifikation verwendet werden [Kol91]. Zur Normierung der Koeffizienten sind folgende Operationen geeignet:

- **Translation**
  *F(0)* bestimmt die Verschiebung des Schwerpunktes der Objektfläche aus dem Ursprung. Durch Nullsetzen von *F(0)* wird das Objekt in den Koordinatenursprung des Ortsbereiches verschoben.

- **Skalierung**
  Eine Multiplikation aller *F(k)* mit einem konstanten Faktor vergrößert oder verkleinert das Objekt und seine Kontur. Wegen der Linearität (siehe Kapitel 8) müssen bei der inversen Transformation die Koordinaten mit demselben Faktor multipliziert werden.
  Die Normierung der Amplitude von *F(1)* normiert die Objektgröße. Der Koeffizient von *F(1)* besitzt (nach *F(0)*) den größten Wert aller Fourier-Koeffizienten, falls die Kontur geschlossen ist und im mathematisch positiven Sinn abgetastet wurde [GW87].

- **Rotation**
  Um die in den Koordinatenursprung verschobene Konturlinie um den Winkel $\alpha$ im Ortsbereich zu drehen, muß jedes Koordinatenpaar *f(i), i* = 0, ... *N* mit dem Faktor $e^{i\,\alpha}$ multipliziert werden. Dies ist identisch mit der Multiplikation der Fourier-Koeffizienten *F(k)* mit $e^{i\,\alpha}$.

- **Startpunkt**
  Da die Konturlinie als eine Periode einer unendlich fortgesetzten periodischen Funktion angesehen wird, entspricht eine Verschiebung des Anfangspunktes auf der Konturlinie um m Punkte einer Multiplikation von *F(k)* mit dem Faktor $e^{(i\,2\pi\,km/N)}$.
  Sowohl die Rotation als auch die Startpunktkorrektur wirken sich auf die Fourier-Koeffizienten aus. Daher kann ein einheitlicher Startpunkt nur ermittelt werden, falls z.B. die normierten Koeffizienten in *F(1)* und in *F(2)* bestimmte Werte annehmen. Durch die Normierung ist dies zwar bei *F(1)* garantiert, nicht aber bei *F(2)*.

Nicht nur die Bestimmung eines einheitlichen Startpunktes führt unter Umständen zu Problemen, auch das diskrete Abtastgitter erzeugt Fehler. Die bisher beschriebenen Zusammenhänge gelten nur bei äquidistanter Abtastung. Da die Konturpunkte jedoch auf einem quadratischen Abtastgitter liegen, variieren die Abtastintervalle um den Faktor $\sqrt{2}$. Daher kann die hier beschriebene Vorgehensweise nur als Näherung angesehen werden.

Der Fehler infolge unterschiedlicher Abtastintervalle wirkt sich insbesondere auf die Phase von *F(k)* aus. Die Auswirkung ist so groß, daß schon einfache Objekte aufgrund dieses Fehlers nicht mehr erkannt werden. Daher beschränkt man sich bei den Merkmalen der Fourier-Deskriptoren häufig nur auf die Amplitude von *F(k)*.

Ist die Anzahl der Konturpunkte eine 2er-Potenz, so kann der FFT-Algorithmus zur Transformation verwendet werden. Diese spezielle Anzahl ist jedoch nur in den seltensten Fällen gegeben und muß im Normalfall künstlich erzeugt werden. Dies kann z.B. durch Aufteilung der Kontur in $2^n$ gleich große Stücke ($n > 0$) und Verwendung jeweils eines Konturpunktes aus diesen Stücken geschehen.

Zur endgültigen Klassifizierung mittels der Fourier-Deskriptoren reichen in der Regel nur einige wenige Werte aus [GW87].

## 16.4 Texturanalyse

Es ist schwierig zu definieren, was genau eine *Textur* ist. Allgemein beschreibt eine Textur die Oberflächenbeschaffenheit (Strukturierung) eines Gegenstandes. Im Bild resultiert dies in einer Strukturierung der Grauwerte. Diese Strukturierung kann sich wiederum aus einzelnen Elementen (*Textureelement, Texel*) aufbauen. Dabei hängt es dann von der Fragestellung ab, ob ein Strukturelement als einzelnes Objekt oder als Texturelement im Zusammenhang mit einem größeren Objekt angesehen wird. Vom menschlichen visuellen System werden Texturen schnell erfaßt und klassifiziert, verbal sind sie aber nur schwer zu beschreiben.

Im CAD-Bereich und bei der Visualisierung von Szenen werden Texturen benutzt, um einzelnen Objekten ein "natürliches" Aussehen zu verleihen, ohne diese Objekte detailgetreu konstruieren zu müssen. Bekannte Beispiele sind Texturen für Gras, Sand oder für Blätter von Bäumen.

Aufgrund des Erscheinungsbildes und auch der Art der Erzeugung werden zwei Gruppen von Texturen unterschieden:

- **Regelmäßige Textur**
  Hierunter wird eine sich wiederholende Anordnung von bestimmten Mustern (Texel) verstanden. Typische Beispiele hierfür sind ein Schachbrett, eine Ziegelsteinwand oder ein Parkettfußboden.

- **Statistische Textur**
  Die Beschreibung und Erzeugung einer statistischen Textur geschieht mit Hilfe einer Zufallsfunktion. Dabei handelt es sich meist um sehr fein strukturierte Muster wie z.B. das Erscheinungsbild von Sand, Gras, Laub usw.

Durch diese Aufteilung wird deutlich, daß eine Texturanalyse mit statistischen Methoden durchgeführt werden muß. Eine weitere Aufteilung dieser beiden Gruppen zeigt dies noch deutlicher.

- **Lokale Textur**
  Bei der lokalen Textur ist das Muster bzw. die Zufallsfunktion abhängig von der Position im Bild (auf dem Objekt). Das Erscheinungsbild dieser Textur variiert daher je nach Position im Bild.

- **Globale Textur**
  Wird für das gesamte Objekt das gleiche Muster bzw. die gleiche Zufallsfunktion verwendet, so wird dies als globale Textur bezeichnet.

- **Hierarchische Textur**
  Eine Textur muß sich im Aufbau nicht auf ein Muster beschränken. Eine hierarchische Organisation ist ebenfalls möglich. Die regelmäßige Textur eines Parkettfußbodens kann als Zusammensetzung von statistischen, lokalen Texturen angesehen werden. Durch eine Zufallsfunktion kann die Strukturierung der einzelnen Holz-Fliesen beschrieben werden. Eventuell unterschiedliche Holzsorten oder Beleuchtungsunterschiede können durch lokale Variationen der Textur realisiert werden. Gelegentlich wird in diesem Zusammenhang auch von Makro- und Mikrotextur gesprochen.

Diese Aussagen über den Aufbau von Texturen legen den Schluß nahe, eine Untersuchung der Texturen im Frequenzraum durchzuführen [Jä89]. Tatsächlich lassen sich damit bestimmte Erscheinungsformen und Muster einfacher erkennen und interpretieren. Dies wurde in ähnlicher Form schon in Abschnitt 16.3 deutlich. Bei der Texturanalyse werden diese Möglichkeiten aufgrund der Komplexität jedoch selten genutzt. Die Analysen beschränken sich in der Regel auf statistische Untersuchungen im Ortsraum.

Eine der verbreitetsten und einfachsten Methode zur Berechnung von Texturmerkmalen ist die Aufstellung von sogenannten *Grauwertübergangsmatrizen*, auch mit *Cooccurrence-Matrizen* bezeichnet.

Eine Cooccurrence-Matrix beschreibt die Grauwertverhältnisse in der näheren Umgebung der einzelnen Bildpunkte. Wegen der Symmetrie reicht es aus, nur vier der 8 möglichen Nachbarn bzw. Richtungen zu betrachten (siehe Abbildung 16.7).

Ist mit *Bild[z][s]* der aktuelle Bildpunkt bezeichnet und soll ein Nachbarbildpunkt im Abstand $d = 1, 2, 3, \ldots$ zur Berechnung der Cooccurrenz-Matrix verwendet werden, so kann dies einer der Nachbarpunkte *Bild[z][s+d]*, *Bild[z+d][s+d]*, *Bild[z+d][s]* oder *Bild[z+d][s-d]* sein, sofern dieser noch im Bild liegt. Am Bildrand sind hier ähnliche Vorkehrungen zu treffen wie bei der Kantendetektion (siehe Kapitel 11).

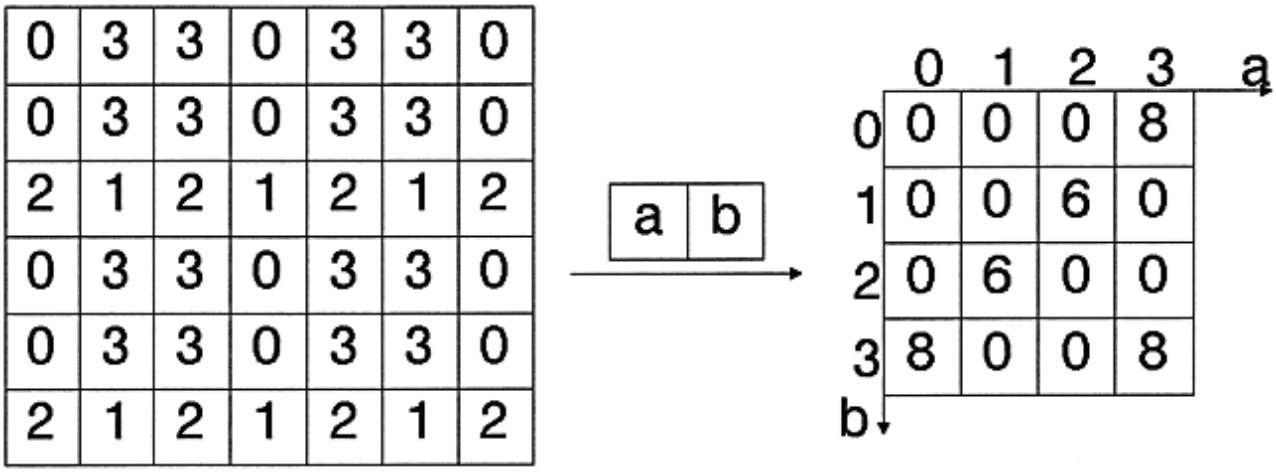


Abbildung 16.7: Bei der Berechnung der Cooccurrenz-Matrix wird der Grauwertunterschied zu einem der vier nachfolgenden Nachbarn verwendet (links). Der Grauwert des aktuellen Bildpunktes ist der eine Index der Cooccurrenz-Matrix, der des Nachbarbildpunktes der andere Index. Der Wert des so indizierten Matrix-Elements wird inkrementiert.

Je nach lokaler Orientierung und Ausdehnung der Muster (Texel) ist die Berücksichtigung des einen oder anderen Nachbarn von Vorteil. Der Abstand *d* wird durch die Größe der Muster bestimmt und liegt in der Regel im Intervall [1, 11].

Die Zähler in der Cooccurrenz-Matrix werden wie bei der Hough-Transformation (siehe Abschnitt 16.2) als Grauwerte interpretiert und entsprechend visualisiert (siehe Abbildung 16.7). Häufig vorkommende Grauwertübergänge werden dabei als hellere Punkte oder Gebiete sichtbar, deren jeweiliger Schwerpunkt als Merkmal der Textur verwendet werden kann.

Da in der Cooccurrenz-Matrix nur die Grauwertübergänge, nicht aber der Ort des Auftretens gespeichert wird, ist eine eindeutige Rücktransformation in das Ausgangsbild nicht möglich. Werden 256 Grauwerte zur Berechnung berücksichtigt, so entstehen relativ große Matrizen. Eine Quantisierung auf z.B. nur 16 Graustufen reduziert den Speicherbedarf für diese Matrizen. Die Merkmal-typische Belegung der Matrix wird durch diese Graustufenreduktion nicht verändert.

Die Belegung der Cooccurrenz-Matrix läßt Schlüsse auf das Aussehen des Ausgangsbildes zu. Langsame Grauwertübergänge im Bild zeichnen sich durch eine starke Belegung der Matrix entlang der Hauptdiagonalen aus. Starke Kontraste im Bild sind durch Einträge deutlich abseits der Hauptdiagonalen gekennzeichnet.

### 16.4.1 Texturmerkmale

Ziel der Texturanalyse ist es, die Muster in einem Bild der Größe $M \times N$ durch einige wenige, aussagekräftige Parameter zu beschreiben. Die Ergebnisse der Untersuchungen im Frequenzraum

oder der Systematiken, die über die Cooccurrenz-Matrizen gefunden wurden, sind solche Parameter.

Weitere einfach zu berechnende Parameter, die als Merkmale verwendet werden können, beschränken sich in der Regel auf statistische Größen [BB82], [Jä89], [BB91], [HB*92]. Diese Merkmale sind rein mathematische Merkmale und lassen in der Regel keinen Schluß auf das Aussehen des Objekts zu. Die Werte werden dabei entweder aus der Cooccurrenz-Matrix oder auch direkt aus dem Bild berechnet. Einige Parameter sind rein mathematische Werte und lassen keine sinnvolle Interpretation des Bildinhaltes zu.

- **Erwartungswert, Mittelwert**
  Ist die Wahrscheinlichkeit des Vorkommens aller Grauwerte *f(x, y)* gleich, so entspricht der Erwartungswert dem mittleren Grauwert (Mittelwert) im Bild.

$$\text{Mittelwert} = \frac{1}{M\,N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

  Bei unterschiedlicher Relevanz der einzelnen Grauwerte können diese noch mit einem Faktor *g* gewichtet werden.
  Werden die Grauwerte in Zeilen- bzw. Spaltenrichtung als die Realisierung zweier unterschiedlicher Zufallsprozesse angesehen, so werden für die Berechnung der Kovarianz und der Korrelation (s.u.) folgende Definitionen verwendet:

$$\text{Mittelwert}_x = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} x\, f(x,y)$$

$$\text{Mittelwert}_y = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} y\, f(x,y)$$

- **Varianz**
  Die Varianz berechnet sich als die Summe der quadratischen Abweichung vom Mittelwert.

$$\text{Varianz} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(\text{Mittelwert} - f(x,y))^2$$

  Die Wurzel aus der Varianz wird mit Standardabweichung bezeichnet. Für die richtungsbezogene Auswertung wird folgende Definition verwendet:

$$\text{Varianz}_x = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(x - \text{Mittelwert}_x)^2\, f(x,y)$$

$$\text{Varianz}_y = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(y - \text{Mittelwerty})^2\, f(x,y)$$

- **Energie**
  Wird der Helligkeitswert eines Bildpunktes als Energiepotential aufgefaßt, so kann die in einem Bild enthaltene Gesamtenergie als

$$\text{Energie} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(f(x,y))^2$$

  berechnet werden.

- **Trägheit**
  Die Trägheit eines Bildes bzw. Objekts ist in diesem Zusammenhang definiert als

$$\text{Trägheit} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}((x-y)^2 f(x,y))$$

- **Entropie**

$$\text{Entropie} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(f(x,y)\log(f(x,y)))$$

- **Homogenität**

$$\text{Homogenität} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\frac{f(x,y)}{1+|x-y|}$$

- **Kovarianz**

  In der Statistik ist die Kovarianz ein Maß für den Zusammenhang zwischen zwei Zufallsprozessen. Bei der Texturanalyse wird dieser Wert als Maß für den Zusammenhang zwischen Bildzeilen und Bildspalten benutzt.

$$\text{Kovarianz} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(x-\text{Mittelwert}_x)(y-\text{Mittelwert}_y)f(x,y)$$

- **Korrelation**

  Die Korrelation gibt weiteren Aufschluß über den Abhängigkeitsgrad zweier Zufallsvariablen.

$$\text{Korrelation} = \frac{\text{Kovarianz}}{\sqrt{\text{Varianz}_x\ \text{Varianz}_y}}$$

Je nach Einsatzgebiet werden die einzelnen Zahlenwerte als Merkmale verwendet. Es wird dabei angenommen, daß eine Textur über das ganze Bild bzw. zu analysierende Objekt verteilt ist. Anhand einer bekannten Referenz-Textur werden diese Werte ebenfalls berechnet und in einem soge-

nannten *Texturvektor* abgespeichert. Dieser Vektor ist die Referenzgröße bei der Untersuchung der unbekannten Texturen. Der Vektor mit den Werten aus den zu klassifizierenden Texturen wird als *Analysevektor* bezeichnet.

Der Analysevektor wird dem ihm "ähnlichsten" Texturvektor zugeordnet. Die Ähnlichkeit kann auch hier über die Summe der Euklidschen Distanzen der einzelnen Merkmale, gegebenenfalls noch unterschiedlich gewichtet, berechnet werden.

Setzt sich ein Bild aus unterschiedlich texturierten Bereichen zusammen, z.B. eine Landschaftsaufnahme mit Himmel, Wolken, Gras, Wasser usw., so ist die Angabe eines Analysevektors für das gesamte Bild nicht sinnvoll. Das Bild muß segmentiert werden und es müssen Analysevektoren für jeden einzelnen Bereich berechnet werden.

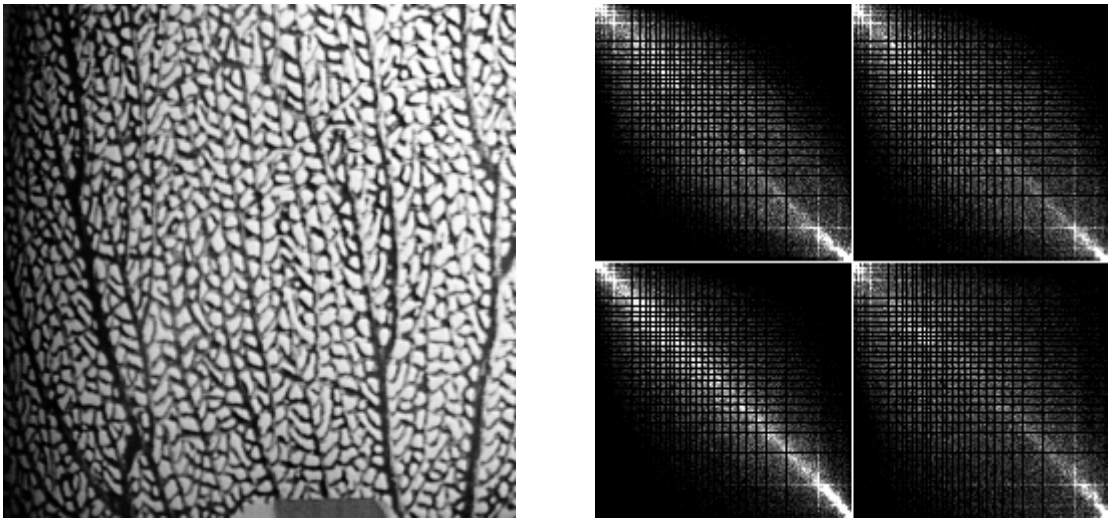

Abbildung 16.8: Das Bild zeigt eine Textur (links) und vier visualisierte Cooccurrenz-Matrizen (rechts; vier Felder). Als Nachbarbildpunkte sind dabei die rechten Nachbarn (Feld links oben), die Nachbarn rechts unten (Feld rechts oben), die Nachbarn unten (Feld rechts unten) und die Nachbarn links unten (Feld links unten) verwendet worden.

Dazu werden in jedem Bildpunkt innerhalb einer Umgebung (3×3, 5×5, ..., 15×15) die Ausprägungen der verwendeten Merkmale (lokaler Mittelwert, lokale Varianz, lokale Homogenität usw.) berechnet. So wird jedem Bildpunkt ein Analysevektor zugeordnet. Die Größe der Umgebung ist dabei abhängig von den zu untersuchenden Texturen und beeinflußt das Ergebnis stark [HB*92]. Die Umgebung sollte mindestens so groß sein, daß genügend Grauwerte erfaßt werden, um signifikante Texturmerkmale zu berücksichtigen.

Die hier beschriebene Vorgehensweise wird besonders dort verwendet, wo klassische Segmentierungsverfahren aufgrund einer besonderen Strukturierung (Texturierung) der einzelnen Gebiete nur mit mäßigem Erfolg eingesetzt werden können. Eines der bekanntesten Beispiele ist die Luftbildauswertung, wo aufgrund des Analysevektors der einzelnen Bildpunkte eine Zuordnung zu den

Klassen "Wald", "Grünfläche", "Wasser", "Acker", "Bebauung" usw. vorgenommen wird. Durch die Verwendung unterschiedlicher Aufnahmen (Rot-Kanal, Grün-Kanal, Blau-Kanal, Infrarot-Kanal) kann die Sicherheit der Klassifikation stark erhöht werden [HB*92].

Sind weitere Informationen über die zu klassifizierenden Texturen vorhanden, z.B. zeichnet sich die Struktur durch Regionen mit einer bestimmten Orientierung aus, so sollte das Bild mit den klassischen Verfahren (siehe Kapitel 14) vorsegmentiert werden. Die eigentliche Klassifikation kann sich dann auf diese Gebiete beschränken und somit störende Bildteile unberücksichtigt lassen.

# Aufgaben

## Aufgabe 1

Entscheiden Sie, ob folgende Merkmale lageabhängig oder lageunabhängig sind.

| Merkmal | lageabhängig | lageunabhängig |
|---|---|---|
| Mittlerer Grauwert | | |
| Flächeninhalt | | |
| Schwerpunkt | | |
| Umfang | | |
| Kompaktheit | | |
| Exzentrizität | | |

## Aufgabe 2

Berechnen Sie mit der angegebenen Formel die Kompaktheit eines Rechtecks mit den Seitenlängen $a = 1$ und $b = 2$.

## Aufgabe 3

Wie groß ist der Zähler der Akku-Zelle eines lokalen Maximums im Hough-Raum idealerweise? Weshalb kann dieser Zahlenwert größer oder kleiner sein?

Wie müßte die Parametrisierung für die Hough-Transformation bei Kreisen aussehen? Welche Dimension hat der Parameter-Raum?

## Aufgabe 4

Gegeben sind die folgenden Grauwerte eines Bildausschnitts. Berechnen Sie die Cooccurrenz-Matrix für diesen Ausschnitt für den horizontalen Grauwertübergang (*Bild[z][s]* und *Bild[z][s+1]*).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 15 | 1 | 10 | 15 | 1 | 10 | 15 | 1 | ... |
| 1 | 10 | 15 | 1 | 10 | 15 | 1 | 10 | ... |
| 10 | 15 | 1 | 10 | 15 | 1 | 10 | 15 | ... |
| 15 | 1 | 10 | 15 | 1 | 10 | 15 | 1 | ... |
| 1 | 10 | 15 | 1 | 10 | 15 | 1 | 10 | ... |
| 10 | 15 | 1 | 10 | 15 | 1 | 10 | 15 | ... |
| .... | ... | ... | ... | ... | ... | ... | ... | ... |

Interpretieren Sie den Aufbau der Cooccurrenz-Matrix.

# A. Definitionen

**Abtastwert**
Eine Zahl, die den Signalwert (Amplitude) zu einem bestimmten Abtastzeitpunkt repräsentiert (→ Quantisierung).

**Abtasttheorem**
Ein Theorem, das besagt, daß jede bandbegrenzte Funktion durch endlich viele Abtastwerte exakt dargestellt werden kann. Die Abtastfrequenz muß dabei mindestens doppelt so groß wie die höchste in der Funktion vorkommende Frequenz sein, damit das Signal eindeutig rekonstruiert werden kann.

**Adaptive Filter**
Eine Klasse von Filtern, deren Koeffizienten in Abhängigkeit von den gerade bearbeiteten Signalen oder den Leistungsfähigkeitskriterien geändert werden.

**ADC**
Die Abkürzung für *Analog to Digital Converter*, deutsch Analog-Digital-Wandler (abgekürzt A/D-Wandler). Das analoge Signal zum Beispiel einer Videokamera wird in einem Bildverarbeitungssystem durch einen A/D-Wandler in digitale Zahlenwerte umgesetzt.

**A/D-Wandlung**
Analog/Digital-Wandlung. Der Vorgang, ein analoges Signal in digitale Daten umzuwandeln. Das Signal wird dabei in festen Zeitabständen abgetastet (→ Diskretisierung) und der analoge Amplitudenwert digitalisiert (→ Quantisierung).

**Alias, Aliasing**
Bei der Signalabtastung dann entstehende Artefakte (→ Artefakt), wenn das abzutastende Signal Komponenten mit einer höheren Frequenz als die halbe Abtastfrequenz enthält.

**Anti-Aliasing**
Die Vorgehensweise, sichtbare Alias-Effekte zu vermindern. In der Regel wird das optische bessere Bild mit einer geringfügigen Reduzierung des Auflösungsvermögens erkauft (→ Alias).

**Artefakt**
In der Bildverarbeitung die Strukturen, die im natürlichen Bild nicht vorkommen und die erst durch die Abtastung und Verarbeitung entstehen. Artefakte sind normalerweise nicht im Ergebnis erwartet.

**Auflösung**
Die Anzahl der Abtastwerte pro Längeneinheit (→ Digitalisierung) in einem digitalisierten Bild in horizontaler bzw. vertikaler Richtung.

Oft wird der Begriff auch nur für die Anzahl der Abtastwerte (z.B. Bildpunkte) ohne zugehörige Längenangabe verwendet.

**Autokorrelation**
Eine mathematische Beschreibung des Grads der Übereinstimmung zwischen zwei Signalabtastungen, auch Bildern oder Bildausschnitten.

**Bandbreite**
Der Frequenzbereich, der von einem analogen System ohne (nennenswerte) Dämpfung übertragen werden kann.

**Bild – analoges, digitalisiertes**
Ein Bild, das aus einer Videoquelle stammt (Kamera, Videorekorder usw.), ist ein kontinuierliches, mit der Zeit in der Amplitude variierendes Signal. Beim digitalen Bild wurde das analoge Signal in gleichmäßigen Zeitabständen $\Delta$ t abgetastet ($\rightarrow$ Diskretisierung), die zeitliche Information in Ortsinformation umgesetzt und die Amplitude der Abtastwerte ($\rightarrow$ Quantisierung) gespeichert. Ein digitales Bild P wird in der Regel in einem zweidimensionalen Feld von Bildpunkten abgespeichert mit $P = \{p(i, j); 1 \leq i \leq i_{max}; 1 \leq j \leq j_{max}\}$ wobei $i_{max}$ und $j_{max}$ die Auflösung des Bildes in den beiden Richtungen ist. Jedem Bildpunkt $p(i, j)$ wird bei Schwarzweißbildern ein Grauwert $0 \leq p(i, j) \leq 255$ zugewiesen.

**Bildvorverarbeitung**
Sammlung von Methoden, die Bildinformationen im Rechner verarbeiten, bzw. die Hervorhebung bestimmter Bildinhalte für die Weiterverarbeitung.

**Bildwiederholfrequenz**
Die Anzahl der Bilder, die komplett pro Sekunde angezeigt/ausgegeben werden können.

**Binär**
Ein Signal, das nur zwei Werte annehmen kann.

**Binärbild**
Ein Bild, das nur noch zwei Farben/Graustufen beinhaltet, in der Regel Schwarz und Weiß.

**Candela**
Basiseinheit der Lichtstärke. Einheitszeichen *cd*.

**CCD**
Die Abkürzung von *Charge-Coupled-Device*. Das CCD ist ein lichtempfindlicher Sensor, der anstelle einer Aufnahmeröhre in eine Video-Kamera eingebaut wird. Wie die Aufnahmeröhre liefert auch das CCD ein Analogsignal, das vor der Verarbeitung im Rechner digitalisiert werden muß.

**CCITT**
Die Abkürzung von *Comité Consultatif International Téléphonique et Télégraphique*. Der "Internationale beratende Ausschuß für den Telegrafen- und Fernsprechdienst" ist beauftragt, über technische, betriebliche und tarifliche Fragen der Fernmeldedienste Studien durchzuführen und Empfehlungen herauszugeben.

**Colour Look Up Table**
→ Farbtabelle, LUT

**Composite Video**
Ein Video-Signal, das alle Farbinformationen in einem Signal enthält. Solche Signale werden in der Fernsehtechnik verwendet.

**CMYK-Farbmodell**
Cyan-Magenta-Yellow-Black-Farbmodell (Türkis-Purpur-Gelb-Schwarz), das für das Drucken von Farben auf Papier verwendet wird (subtraktive Farbmischung, → Farbsystem; vergleiche → RGB).

**Chrominanz**
Farbmetrischer Unterschied (dominante Wellenlänge und Sättigung) zwischen Farben und ihr Bezug zu Weiß bei konstanter Helligkeit. Aus einem RGB-Farbsignal lassen sich die Chrominanzsignale U und V nach der Formel: $U = B - Y$ und $V = R - Y$ berechnen (→ Luminanz).

**Differenzbild**
Das Ergebnis der Subtraktion der Grauwerte zweier Bilder.

**Digitalisierung**
Der Vorgang, ein analoges Signal in ein digitales zu konvertieren. Das Gerät dazu wird Digitalisierer genannt.

**Dilatation**
Bei der Bildverarbeitung bezeichnet die Dilatation das selektive Hinzufügen von Bildpunkten, indem die Umgebung eines jeden Bildpunktes *p(i, j)* des dargestellten Objektes mit einer verschiebbaren Maske verglichen wird und die Punkte der Maske dem Objekt hinzugefügt werden, wenn mindestens ein Punkt der Maske mit dem darunterliegenden Bildpunkt übereinstimmt. Die Dilatation liefert also diejenige Menge von Bildpunkten, die gleich der Vereinigungsmenge von Bildelementen und Maskenelementen sind (→ Erosion). Meist wird eine 3×3-Maske verwendet.

**dpi**
Die Abkürzung von *Dots per Inch*, deutsch Punkte pro Inch. Eine hohe Punktdichte bedeutet für sich allein noch keine hohe Auflösung (→ Auflösung; der Durchmesser eines Punktes muß dazu auch entsprechend klein sein).

**Diskretisierung**
Die Abtastung eines analogen Signals in bestimmten Zeitabständen.

**Dispersion, optische**
Die unterschiedliche Brechung des Lichtes bei unterschiedlichen Wellenlängen in optischen Linsen.

**Dithering**
Ein Halbtonverfahren, das den Schwellwert von Bildpunkt zu Bildpunkt ändert (→ Halbtonverfahren).

**Echtfarben**
Die Darstellung mit mehr Farben und Farbtönen als das menschliche Auge unterscheiden kann. In der Regel bezeichnet man ein System mit jeweils 8-Bit Information für Rot, Grün und Blau als Echtfarbensystem (über 16.7 Millionen Farben).

**Echtzeit**
Operationen, die in realer Geschwindigkeit ablaufen. In der Bildverarbeitung wird von Echtzeit gesprochen, wenn ein komplettes Bild innerhalb der Zeit der Bildwiederholung (→ Bildwiederholfrequenz) bearbeitet werden kann.

**Erosion**
Bei der Bildverarbeitung bezeichnet die Erosion das selektive Löschen von Bildpunkten, indem die Umgebung eines jeden Bildpunktes *p(i, j)* des dargestellten Objektes mit einer verschiebbaren Maske verglichen wird und nur die Bildteile des Objekts erhalten bleiben, die vollständig von der Maske verdeckt werden können. Die Erosion liefert also diejenige Menge von Bildpunkten, die gleich dem Durchschnitt von Bildelementen und Maskenelementen sind (→Dilatation).

**Faltung**
Aus zwei Eingangsfunktionen wird mit entsprechenden Operationen eine dritte erzeugt.

$$f(x)*g(x)=\int_{-\infty}^{\infty} f(\alpha)\,g(x-\alpha)\,d\alpha$$

**Farbe**
Die Helligkeitsinformation von drei Grundfarben sowie deren Mischungsverhältnis.

**Farbfehler**
Die unterschiedliche Empfindlichkeit für Farben oder die unterschiedliche Wiedergabeintensität von Farben.

**Farbsystem, additives**
Ein Farbsystem, bei dem die einzelnen Farben durch Mischen der Grundfarben Rot, Grün und Blau entstehen. Sind alle drei Farben mit der maximalen Intensität vorhanden, so entsteht Weiß.

**Farbsystem, subtraktives**
Ein Farbsystem, bei dem die einzelnen Farben durch Subtraktion der Grundfarben Magenta, Cyan und Gelb entstehen. Sind alle drei Farben mit der maximalen Intensität vorhanden, so entsteht Schwarz.

**Farbtabelle (LUT), Farbpalette**
Eine Liste, die für jede der gleichzeitig darstellbaren Farben die Intensitäten für drei Grundfarben (z.B. Rot, Grün und Blau) beinhaltet. Üblich sind 8-Bit-Farbtabellen (=256 Einträge) mit einer Farbtiefe von ebenfalls jeweils 8-Bit (3×8 Bit). Also können 256 Farben aus 16.7 Millionen möglichen Farbe gleichzeitig dargestellt werden.

**Fenster/Maske**
Der (lokale) Einzugsbereich eines Operators. Die einzelnen Elemente dieses meist quadratischen Feldes werden noch mit unterschiedlichen Gewichten belegt.

**Filterung**
Die Bearbeitung eines Signals/Frequenz mit verschiedenen Methoden um bestimmte Eigenschaften des Signals bzw. andere Frequenzen hervorzuheben oder zu vermindern.

**Frame**
Die Information, die in einem darstellbaren Bild enthalten ist.

**Frame-Grabber**
Ein Zusatzgerät (üblich: Steckkarte im PC), das ein Video-Signal digitalisiert (→ Digitalisieren).

**Gamma-Korrektur**
Die nichtlineare Gewichtung der Helligkeitsinformation bei der Wiedergabe von Bildern auf Monitoren. Die Gamma-Korrektur dient zur Anpassung an das logarithmische Helligkeitsempfinden des Menschen.

**Gradient**
Der Vektor, der aus den Richtungsableitungen an einem Ort $P_0$ gebildet wird. Der Gradient zeigt immer in die Richtung der stärksten Änderung am Ort $P_0$.

**Graustufe, Grauwert**
Variation im Helligkeitsbereioch des Weiß-Lichtes von Schwarz bis Weiß. Die Graustufen können in einem monochromen 8-Bit-Systemvon 0 (=Schwarz) bis 255 (=Weiß) variieren.

**Halbtonverfahren**
Die Methode zur Überführung eines Grauwertbildes in ein Binärbild, bei dem die Graustufen durch unterschiedliche Punktdichten und Punktgrößen simuliert werden (→ Dithering).

**Helligkeit, Grauwert**
Der Zahlenwert, der die Helligkeitsintensität eines Bildpunktes repräsentiert (→ Graustufe).

**Histogramm**
Die graphische Darstellung der Häufigkeit der im Bild vorkommenden einzelnen Grau- oder Farbwerte. Die horizontale Achse repräsentiert die Grauwerte, die vertikale Achsel die Anzahl der Pixel innerhalb eines Grauwertes.

**HLS**
Bezeichnet eine Art der Farbdefinition. HLS kommt vom englischen Hue (=Farbart oder Farbton), Lightness (=Helligkeit), Saturation (=Farbsättigung). Die Farbart nimmt Werte zwischen 0 und 360 Grad an, Sättigung und Helligkeit jeweils zwischen 0 und 1. Die HLS-Definition ist alternativ zur RGB-Definition (→ HSI, → RGB).

**Hochpaßfilter**
Eine Operation im Orts- oder Ortsfrequenzbereich, die niederfrequente Anteile dämpft oder ganz beseitigt (Kantenverstärkung, Kontrastverstärkung).

**HSI**
Bezeichnet eine Art der Farbdefinition. Abkürzung von Hue (=Farbart oder Farbton), Saturation (=Farbsättigung) und Intensity (=Helligkeit) (→ HLS).

**Inverse Filterung**
Die Beseitigung einer bekannten Störung aus einem Bild durch eine Umkehrtransformation.

**Interlaced**
Die Darstellung von Bildern mit je zwei getrennten Halbbildern, um eine höhere Flimmerfreiheit bei geringem Zusatzaufwand zu erhalten. Dies wird durch die Persistenz des Phosphors ermöglicht. Der Vorteil des Halbbild-Verfahrens – auch Zeilensprung-Verfahren genannt – beim Fernsehen ist, daß mit der gleichen Datenmenge (→ Bandbreite) statt 25 ganzen Bildern 50 Halbbilder pro Sekunde übertragen werden. Der Nachteil ist, daß waagrechte Linien oftmals zittern und daß große helle Flächen trotzdem flimmern.

**Isotrop**
Bei der Kantendetektion die Bezeichnung der Eigenschaft einer Filtermaske, auf alle möglichen Kantenrichtungen dieselbe Antwort zu liefern (richtungsunabgängig).

**JBIG**
Abkürzung von *Joint Bilevel Image Coding Group*, eine Gruppe, die sich mit der Normierung der Komprimierung (→ Kompression, → Reduktion) von stehenden Schwarzweißbildern beschäftigt (→ JPEG).

**JPEG**
Abkürzung von *Joint Photographic Expert Group*, eine Gruppe, die sich mit der Normierung der Komprimierung (→ Kompression, → Reduktion) von stehenden Bildern beschäftigt (→ MPEG).

**Kante**
Diskontinuität im Verlauf der Intensitätswerte bzw. Grauwerte eines Bildes.

**Kantenbild**
Ein Bild, das nur noch Punkte aus den beiden Gruppen "Hintergrund" und "Kante" beinhaltet.

**Kompression**
Ein Vorgang, der ein Signal/Bild verlustfrei modifiziert, so daß danach weniger Speicherplatz für dieselbe Information benötigt wird.

**Kontrast**
Verhältnis von hellem zu dunklem Bildpunkt.

**Kontur**
Eine Liste von Vektoren $v_l$, wobei der Endpunkt von $v_l$ den Anfangspunkt von $v_{l+1}$ darstellt. Ist der Endpunkt des letzten Vektors dieser Liste identisch mit dem Anfangspunkt des ersten Vektors, so nennt man die Kontur geschlossen. Eine Kontur repräsentiert in der Regel den Umriß eines Objekts.

**Korrelation**
Der Grad der Beziehung zwischen zwei Signalen/Bildern.

**Luminanz**
Helligkeitswert Y eines Farbsignals (Leuchtdichte). Aus einem RGB-Signal läßt sich sie (subjektive) Helligkeit nach der Formel: $Y = 0.299R + 0.587G + 0.114B$ berechnen (→ Chrominanz).

**LUT**
Look-Up-Table (→ Farbtabelle)

**Merkmalsextraktion**
In der Mustererkennung das Auffinden und das Klassifizieren von vorgegebenen Mustern in einem Bild.

**Monitor**
Auch Bildschirm oder Datensichtgerät genannt. Dient zur Darstellung von Bildern und zur Visualisierung von Informationen.

**MPEG**
Die Abkürzung von *Motion Picture Coding Expert Group*, eine Gruppe, die sich mit der Normung der Komprimierung (→ Komprimierung, → Reduktion) von Bewegtbildern beschäftigt (→ JPEG).

**Nachbarn, direkte**
Menge der Bildpunkte $N_{i,j}$ in der Umgebung eines Pixels $p(i, j) \in P$, mit
$N_{i,j}(P) = \{(m, n) : |m - i| \leq 1; |n - j| \leq 1;$ und $(m, n) \neq (i, j)\}$ . Werden alle 8 Nachbarn bei einem Algorithmus verwendet, so spricht man von 8er Nachbarschaft, werden nur die horizontalen/ vertikalen bzw. die diagonalen benutzt, so bezeichnet man dies als 4er Nachbarschaft.

**Nyquist-Frequenz**
Bei der Signalabtastung die höchste Frequenz des Eingangssignals, die noch fehlerfrei abgetastet werden kann (die Hälfte der Abtastfrequenz), ohne daß also Alias-Effekte entstehen (→ Abtasttheorem).

**Operator**
In der Bildverarbeitung meist die Umschreibung für ein Verfahren.

**Pixel**
Bildpunkt, engl. Picture-Element. Ein Punkt $p(i, j) \in P$ des digitalisierten Bildes $P$. Jedem Bildpunkt $p(i, j)$ wird bei einem Schwarzweißbild ein Grauwert $0 \leq p(i, j) \leq 255$ zugewiesen.

**Quadtree**
Regelmäßige, rekursive Einteilung einer Vorlage in jeweils 4 quadratische Teile zur Komprimierung.

**Quantisierung**
Die Einteilung des analogen Signals (Amplitude) in bestimmte Stufen von Signalstärken (→ Abtastwert).

**Quantisierungsfehler**
Der Fehler, der aufgrund mangelnder (Zahlen-) Auflösung durch die Quantisierungsstufen gemacht wird.

**Rauschen**
Störungen des (Bild-) Signals, die die Amplitude und damit den Grau-/ Farbwert an einzelnen Stellen ändern.

**Reduktion**
Ein Vorgang, der ein Signal/Bild modifiziert, so daß danach weniger Speicherplatz für die Information benötigt wird. Dieser Vorgang ist nicht verlustfrei, jedoch ist der entstandene Fehler in der Regel so gering, daß er vernachlässigt oder toleriert werden kann.

**RGB**
Bezeichnet die Art der Farbdefinition. RGB bezeichnet die Primärfarben der additiven Farbmischung (Rot, Grün, Blau). Die Definition einer Farbe nach dieser Methode erfordert mehr Übung als nach der HLS-Methode (→ HLS).

**RLC, RLE**
Run-Length-Coding (Lauflängen-Kodierung). Das mehrmalige, aufeinanderfolgende Vorkommen desselben Grau-/Farbwertes wird durch die Anzahl des Vorkommens und den Wert kodiert. In der Regel wird ein Byte für die Anzahl und ein Byte für den Grau-/Farbwert verwendet.

**RMS**
Root-Mean-Square (quadratisches Mittel). Die Mittelwertbildung durch die Berechnung der Wurzel aus der Summe der Quadrate der einzelnen Signalwerte.

**Sampling**
Das Abtasten eines analogen Signals (→ Diskretisierung).

**Sättigung**
In einem Echtfarbsystem die Reinheit der Farbe oder wie schwach oder kräftig eine Farbe erscheint (→ HLS).

**Scannen**
Der Vorgang, der ein Bild in ein elektrisches Signal umwandelt, indem ein Sensor über das Bild bewegt wird.

**Segmentierung**
Aufteilung einer Szene in verschiedene Bereiche nach bestimmten Kriterien. Dabei werden implizit diesen Bereichen schon Eigenschaften zugeordnet.

**Sekundärfarben**
Türkis, Purpur und Gelb (→ CMYK).

**Sensorfläche**
Die lichtempfindliche Fläche eines Sensors bei einer Kamera (→ CCD).

**Signal-zu-Rausch-Verhältnis (S/N)**
In einem analogen Video-System das Verhältnis von Spitze-zu-Spitze Schwarz-zu-Weiß-Signal und dem RMS-Wert (→ RMS) des überlagerten Rauschens. In analogen Audio-Systemen das Verhältnis von RMS-Signal zu RMS-Rauschen. Oft wird dies auch statt mit S/N (Signal-to-Noise) mit SNR (Signal-to-Noise-Ratio) bezeichnet.

**Skelett**
Die Repräsentation eines Objektes durch seine Mittelachsen. Bei einem Skelett sind alle Kanten senkrecht zur Kantenrichtung nur ein Pixel breit.

**SNR**
Signal-zu-Rausch-Verhältnis (→ S/N)

**Template**
Eine Schablone, Maske, die bei der Mustererkennung verwendet wird.

**Tiefpaßfilter**
Eine Operation im Orts- oder Ortsfrequenzbereich, die hochfrequente Anteile dämpft oder ganz beseitigt (z.B. zur Rauschunterdrückung).

**Unschärfe**
Die nicht eindeutige Lokalisierung von Kanten. Dies bedeutet gleichzeitig, daß hochfrequente Bildanteile nicht oder nur in geringem Maße vorkommen (→ Tiefpaßfilter).

**Vektor**
Eine Strecke mit einer Richtung und mindestens 2 Pixel Länge.

**Voxel**
Ein dreidimensionaler "Bildpunkt". Ein Voxel hat in der Regel die Form eines Quaders.

**Wellenzahl**
Mit Wellenzahl wird die Anzahl der Wellenlängen pro Einheitslänge bezeichnet und ist damit der reziproke Wert der Wellenlänge $\lambda$. In der Physik wird dieser Wert oft noch mit dem Faktor $2\pi$ multipliziert.

**YUV**
Bezeichnet eine Art der Farbdefinition. Die Farbe wird hier durch die Leuchtdichte (→ Luminanz, Helligkeit) und die Farbart (→ Chrominanz) definiert. Teilweise wird dieses YUV-Modell auch mit YIQ- oder YCbCr-Modell bezeichnet. Diese Art der Farbdefinition wird beim Farbfernsehen verwendet.

# B. Die VGA-Karte

Alle in diesem Buch vorgestellten Verfahren und abgedruckten Routinen wurden unter anderem auf einem PC-kompatiblen Rechner mit VGA-Graphikkarte implementiert. Zum besseren Verständnis der Graphikausgabe soll der Aufbau der VGA-Karte (*VGA=Video-Graphics-Adapter*) und ihre Funktionsweise an dieser Stelle kurz erklärt werden. Gleichzeitig werden dabei Anregungen für spezielle Optimierungen und Erweiterungen oder für die Anpassung an andere Graphikkarten gegeben.

Die verwendeten Adressen und Befehlskodes für die VGA-Karte betreffen alle den weit verbreiteten Tseng-ET3000/ET4000-VGA-Chipsatz.

## B.1 Der technische Aufbau

Die VGA-Graphikkarte besteht im wesentlichen aus fünf Funktionsgruppen und dem Bildschirmspeicher [Alt90], [Mei91b].

Der Bildschirm-Kontroller (*CRT-Controller, CRTC*) steuert den Katodenstrahl und erzeugt die horizontalen und vertikalen Synchronsignale, sowie die Refresh-Signale für den dynamischen Video-Speicher. Im Sequenze-Kontroller (*Sequence-Controller, SC*) wird der Zugriff auf den Bildschirmspeicher der Graphikkarte gesteuert und das Zeitverhalten für die Übertragung der Bildpunkte an den Monitor festgelegt. Der Graphik-Kontroller (*Graphik-Controller, GC*) koordiniert die Kommunikation zwischen CPU, Video-Speicher und Attribut-Kontroller. Der Attribut-Kontroller (*Attribute-Controller, AC*) übernimmt die Farbgebung von Punkten und Buchstaben und sendet die Ausgangsfarbe zum *Digital-Analog-Converter* (DAC). Diese Steuerlogik übernimmt die Kontrolle des Farbsignals, das zum Bildschirm gelangt. Der DAC sorgt für eine Umwandlung des internen Digitalsignals in ein Analogsignal.

## B.2 Die Programmierung

Die ersten VGA-Karten (*Video-Graphics-Adapter*) wurden 1987 der Öffentlichkeit vorgestellt. Diese neuen Karten sollten kompatibel zu dem damaligen EGA-Standard (*Enhanced-Graphics-Adapter*) sein, aber gleichzeitig eine höhere Auflösung und mehr Farben bieten.

Die EGA-Karte bot die Möglichkeit, 16 Farben zu einer Palette (Farbtabelle) zusammenzustellen. Dafür verfügte der Attribut-Kontroller über 16 Paletten-Register, die in sechs Bits die unterschiedlichen RGB-Komponenten bestimmten. Dies ergab gerade $2^6$, also 64 mögliche Farbtöne. Es war auf diese Weise aber unmöglich, zum Beispiel 16 verschiedene reine Blautöne zu erzeugen.

| Bedeutung | I/O-Adresse |
|---|---|
| AC-Index | 03C0h |
| AC-Daten | 03C1h |
| SC-Index | 03C4h |
| SC-Daten | 03C5h |
| GC-Index | 03CEh |
| GC-Daten | 03CFh |
| CRTC-Index | 03D4h |
| CRTC-Daten | 03D5h |
| DAC-Index | 03D7h/03C8h |
| DAC-Daten | 03C9h |

Tabelle B.1: Einige wichtige I/O-Adressen der verschiedenen Bausteine des VGA-Chipsatzes.

Aus Kompatibilitätsgründen findet man auch beim VGA-Konzept den Attribut-Kontroller mit den bekannten 16 Paletten-Registern. Allerdings liefert die VGA-Karte im Gegensatz zu den anderen Graphikkarten ein analoges Ausgangssignal. Dieses wird aus dem digitalen Zahlenwerten im Video-DAC erzeugt. Der Video-DAC enthält 256 jeweils 18 Bit breite Datenregister (Farbregister) und drei 6Bit Digital-Analog-Konverter für die drei Grundfarben Rot, Grün und Blau. Dies ergibt folglich $2^{18}$ (also 262144) mögliche Farbtöne, die theoretisch darstellbare VGA-Palette. Jedoch sind nur jeweils 64 reine Farbtöne und maximal 64 Graustufen realisierbar.

Graphikkarten mit der Echtfarben-Betriebsart (*True-Color*) stellen jedem Pixel 24 Bit an Farbinformation zur Verfügung. Der Video-DAC ist dann ebenfalls 3×8 Bit breit. Damit lassen sich über 16.7 Millionen Farbtöne erzeugen.

Um die Kompatibilität zum EGA-Standard zu wahren, werden die 64 möglichen Farbtöne der EGA-Karte durch eine entsprechende Initialisierung der unteren 64 Datenregister im Video-DAC emuliert. Die Palette im Attribut-Kontroller dient dabei der Adressierung dieser DAC-Register. Da die Palettenregister des Attribut-Kontrollers nur sechs Bit breit sind, können nur 64 Datenregister des Video-DACs adressiert werden.

Der Bildschirmspeicher einer VGA-Karte liegt nach den Vorgaben von IBM in einem 64 Byte großen Segment ab der Adresse A000h im Adreßraum der CPU. Eine VGA-Karte ist aber standardmäßig mit mindestens 256 KByte bestückt. Um bei Auflösungsmodi, die mehr als 64 KByte Bildspeicher benötigen, den gesamten Speicher anzusprechen, wird dieser in einzelne 64 KByte Segmente (Banks) unterteilt. Welches der Video-Segmente jeweils ausgewählt wird, legt man beim ET-4000-Chipsatz über das Segmentauswahlregister an der I/O-Adresse 3CDh fest. Die unteren vier Bits enthalten die Segmentnummer bei Schreib-, die oberen vier Bits diejenigen bei Lesezugriffen. Damit sind insgesamt 16 Segmente à 64 KByte, also insgesamt 1024 Kbyte Bildschirmspeicher adressierbar.

Die Adressierung der einzelnen Segmente (*Banking-Mechnismus*) ist leider nicht standardisiert und muß daher für jeden VGA-Chipsatz entsprechend neu berücksichtigt werden. Beim ET-3000-Chipsatz zum Beispiel werden im Segmentauswahlregister nur jeweils 3 Bit verwendet, also Bit 0-2 für die Segmentnummer bei Schreibzugriffen und Bit 3-5 für die Segmentnummer bei Lesezugriffen. Bit 6 ist bei 64 KByte-Segmenten immer 1 und Bit 7 immer 0. Damit lassen sich nur 512 KByte Bildschirmspeicher adressieren.

Die (lineare) Adresse bezogen auf den unsegmentierten Bildspeicher auf der VGA-Karte selbst ergibt sich zu

```
Adresse = Horiz_Aufl * Zeile + Spalte
```

Die Segmentnummer berechnet sich dann aus

```
Segment = (int) (Adresse / 65536)
Offset  = Adresse - (Segment * 65536)
```

Der Videospeicher ist bei den 16-Farb-Modi in 4 Bit-Planes (Ebenen) aufgeteilt. Jeweils eine Ebene liegt in einem 64 KByte-Segment, von dem in der Regel nur ein Teil des Speichers genutzt wird. Bei dem häufig verwendeten Modus 12h (640×480 Punkte bei 16 Farben) in 4 Ebenen werden nur 38400 Byte (=640*480/8) pro Ebene benötigt (insgesamt 153600 Byte).

Bei den 256-Farb-Modi werden die Punkte byteweise gespeichert und der Videospeicher in die 64 KByte-Segmente aufgeteilt. Somit ist maximal eine Adresse aus 20 Bit für jedes Pixel notwendig, wobei die Bits 16–19 die Segmentnummer repräsentieren und die Bits 0–15 der Offset im normalen Speichersegment ab A000h.

Einige VGA-Karten (z.B. Trident) verwenden zusätzlich den Speicherbereich von B000h-BFFFh, der normalerweise von der Hercules-Graphikkarte (ab B000h) oder der CGA-Karte (ab B800h) belegt ist. So entsteht ein linearer Speicher von 128 KByte.

Die einzelnen Modi werden über das BIOS der VGA-Karte durch den Software-Interrupt 10h mit dem Wert 0 im AH-Register und der Modus-Nummer im AL-Register aufgerufen (bei der Video-7-VGA-Karte steht im AX-Register 6F05h und im BL-Register der Modus) [Mei91b].

Neben dem Setzen des Auflösungsmodus läßt sich noch eine Vielzahl weiterer Funktionen über das VGA-BIOS aktivieren. Die einzelnen Routinen tragen eine Nummer, mit der sie zu identifizieren sind. Vor dem Aufruf des Software-Interrupts wird diese Nummer in das Prozessor-Register AH geladen.

## B.3 Die Initialisierung der Karte

Vor der eigentlichen Ausgabe von Bildern, genauer gesagt vor dem Setzen von Bildpunkten, muß zuerst der gewünschte Video-Modus aktiviert und die Farbtabelle initialisiert werden.

| Modus-Nr. | Typ | Farben | Auflösung | Bemerkung |
|---|---|---|---|---|
| 0/1 | Text | 16 | 360x400 | Standard |
| 2/3 | Text | 16 | 720x400 | Standard |
| 04h | Graphik | 4 | 320x200 | Standard |
| 05h | Graphik | 4 | 320x200 | Standard |
| 06h | Graphik | 2 | 640x200 | Standard |
| 07h | Text | mono | 720x400 | Standard |
| Dh | Graphik | 16 | 320x200 | Standard |
| Eh | Graphik | 16 | 640x200 | Standard |
| Fh | Graphik | mono | 640x350 | Standard |
| 10h | Graphik | 16 | 640x350 | Standard |
| 11h | Graphik | 2 | 640x480 | Standard |
| 12h | Graphik | 16 | 640x480 | Standard |
| 13h | Graphik | 256 | 320x480 | Standard |

Tabelle B.2: Die Standardmodi der VGA-Karte.

Alle PC-C-Compiler enthalten entsprechende Funktionen, um die Register wie benötigt zu initialisieren und die Einstellungen abzufragen.

In der vordefinierten Struktur `pc_cpu` bedeutet der Vorsatz "x" vor dem Register, daß es sich um ein 16 Bit Register, der Vorsatz "h", daß es sich um ein 8 Bit Register handelt. Die Anweisung `intdos()` übernimmt dann den Aufruf des Interrupts. Ihr wird daher zuerst die Nummer des aufrufenden Interrupts, als zweiten Parameter die Inputregister und als dritten Parameter die Adresse der Strukturvariablen, in der die Ergebnisse zurückgeliefert werden sollen, übergeben.

```
void Init_Var(void)    /* Initialisierung der globalen Variablen */
{
  long Index;          /* Index fuer alle Bildpunkte             */
  int z, s;            /* Zeilen- und Spaltenzaehler fuer Icons  */

  clrscr();            /* Bildschirm zuerst komplett loeschen    */

  /* Platz fuer benoetigte Bilder reservieren. Falls erfolglos,  */
  /* dann Abbruch mit Fehlermeldung.                             */
  Pictures=farmalloc((long) (PIC_SIZE*MAX_PICTURE));
  if (Pictures == NULL) {
     WriteText(NO_CURS_POS, NO_CURS_POS,
                "Bilder konnten nicht angelegt werden. Abbruch!");
     exit(1);
  }
```

| Modus-Nr. | Typ | Auflösung | Bemerkung |
|---|---|---|---|
| 1Ch | Graphik | 640x480 | Tecmar VGA |
| 1Dh | Graphik | 800x600 | Tecmar VGA |
| 2Eh | Graphik | 640x480 | Orchid, Genoa, Sigma<br>Tseng ET-3000/4000, VEGA |
| 30h | Graphik | 800x600 | Orchid, Genoa, Sigma<br>Tseng ET-3000/4000, VEGA |
| 38h | Graphik | 1024x768 | Tseng ET-4000 |
| 62h | Graphik | 1024x768 | Trident, nur 8900-Chip |
| 67h | Graphik | 640x480 | Video-7, VEGA |
| 69h | Graphik | 800x600 | Video-7, VEGA |
| 7Eh | Graphik | 1024x768 | CTI, nur 453-Chip |
| | | | |
| 101h | Graphik | 640x480 | VESA |
| 103h | Graphik | 800x600 | VESA |
| 105h | Graphik | 1024x768 | VESA |
| 107h | Graphik | 1280x1024 | VESA |

Tabelle B.3: Einige Modi von VGA-Karten, die über die Standardauflösung hinausgehen und im Rahmen der Bildverarbeitung interessant sein können [Mei91a]. Es können jeweils 256 verschiedene Farben gleichzeitig dargestellt werden.

```
   /* Falls zu wenig freier Hauptspeicher fuer weitere Arbeiten,  */
   /* dann ebenfalls Abbruch.                                     */
   if (farcoreleft()<2*PIC_SIZE) {
      WriteText(NO_CURS_POS, NO_CURS_POS,
                         "Zu wenig freier Hauptspeicher. Abbruch!");
      exit(1);
   }
   /* Alle Bildpunkte aller Bilder mit SCHWARZ initialisieren.    */
   for (Index=0; Index<(PIC_SIZE*MAX_PICTURE); Index++)
       Pictures[Index]=SCHWARZ;

   /* Alle Bildinformation zuruecksetzen.                         */
   for (Index=0; Index<MAX_PICTURE; Index++) {
       Bild[Index].Zeilen =0;
       Bild[Index].Spalten=0;
       Bild[Index].Used   =FALSE;
       Bild[Index].Color  =FALSE;
       for (z=0; z<ICON_SIZE; z++)        /* Alle Icons loeschen. */
           for (s=0; s<ICON_SIZE; s++)
               Bild[Index].Icon[z][s]=SCHWARZ;
   }
   directvideo=FALSE;  /* Textausgabe ueber BIOS im Graphik-Modus */

   Graph_Mode();       /* Graphik-Modus einschalten und ...       */
   Set_LUT();          /* ... Farbtabelle initialisieren.         */
}
```

| Funktionsnummer | Bedeutung |
|---|---|
| 00h | Videomodus setzen |
| 01h | Kursor-Größe festlegen |
| 02h | Kursor-Position festlegen |
| 03h | Kursor-Position lesen |
| 04h | Lichtgriffel lesen |
| 05h | Bildschirmseite wählen |
| 06h | aufwärts rollen |
| 07h | abwärts rollen |
| 08h | Bildschirmzeichen lesen |
| 09h | Bildschirmzeichen setzen 1 |
| 0Ah | Bildschirmzeichen setzen 2 |
| 0Bh | Farbe setzen |
| 0Ch | Bildpunkt setzen |
| 0Dh | Bildpunkt lesen |
| 0Eh | Bildschirmzeichen setzen 3 |
| 0Fh | Status lesen |
| 10h | Parameter lesen und setzen |
| 13h | Zeichenkette ausgeben (ab AT) |
| 1Ah | Graphikinformation |

Tabelle B.4: Im VGA-BIOS sind sehr vielfältige Funktionen realisiert. Diese können einfach über die entsprechende Nummer aktiviert werden [Alt90], [Mei91b].

```
void Graph_Mode(void)              /* Umschaltung auf Graphik    */
{
  pc_cpu.h.ah = 0;                 /* Neuer Video-Modus ueber    */
  pc_cpu.h.al = GRAPH_MODE;        /* Funktionsaufruf 00 setzen  */
  int86(0x10, &pc_cpu, &pc_cpu);   /* und durch Interrupt aktiv. */

  vgaram = MK_FP(0xA000, 0);       /* Far-Pointer auf VGA-Segment*/
}

void Text_Mode(void)               /* Umschaltung auf Text-Modus */
{
  pc_cpu.h.ah = 0;                 /* Text-Modus ueber den       */
  pc_cpu.h.al = TEXT_MODE;         /* Funktionsaufruf 00 setzen  */
  int86(0x10, &pc_cpu, &pc_cpu);   /* und durch Interrupt 10 akt.*/
}
```

```
void Set_LUT(void)                /* Farbtabelle initialisieren */
{
  int i;
  RGB RGB_Tabelle[MAX_COLOR];     /* bis zu MAX_COLOR Eintraege */

  for (i=0; i<MAX_COLOR; i++) {   /* Zuerst Graukeil erzeugen   */
      RGB_Tabelle[i].Rot   = i >> 2;  /* Da nur 6 Bit pro Farbe  */
      RGB_Tabelle[i].Gruen = i >> 2;  /* ergeben sich max. 64    */
      RGB_Tabelle[i].Blau  = i >> 2;  /* Graustufen              */
  }

  RGB_Tabelle[0].Rot   =   0;     /* Index [0] = Schwarz        */
  RGB_Tabelle[0].Gruen =   0;
  RGB_Tabelle[0].Blau  =   0;
  RGB_Tabelle[1].Rot   =  63;     /* Index [1] = Rot            */
  RGB_Tabelle[1].Gruen =   0;
  RGB_Tabelle[1].Blau  =   0;
  RGB_Tabelle[2].Rot   =   0;     /* Index [2] = Gruen          */
  RGB_Tabelle[2].Gruen =  63;
  RGB_Tabelle[2].Blau  =   0;
  RGB_Tabelle[3].Rot   =   0;     /* Index [3] = Blau           */
  RGB_Tabelle[3].Gruen =   0;
  RGB_Tabelle[3].Blau  =  63;
  RGB_Tabelle[7].Rot   =  50;     /* Index [7] = ein Grauton    */
  RGB_Tabelle[7].Gruen =  50;
  RGB_Tabelle[7].Blau  =  50;

  pc_cpu.h.ah=0x10;               /* Aktiviere Modus zum Setzen */
  pc_cpu.h.al=0x12;               /* mehrerer Farb-Palettenreg. */
  pc_cpu.x.bx=0;                  /* Beginne bei Register Nr. 0 */
  pc_cpu.x.cx=MAX_COLOR;          /* aendere MAX_COLOR Register */
  s_cpu.es   =FP_SEG(RGB_Tabelle); /* Segmentadresse LUT-Tabelle */
  pc_cpu.x.dx=FP_OFF(RGB_Tabelle); /* Offsetadresse im Segment   */

  int86x(0x10, &pc_cpu, &pc_cpu, &s_cpu); /* ueber Interrupt 10  */
}
```

Bei der Ausgabe von Bildern auf anderen Systemen, die nicht den Beschränkungen der PC-Umgebung und der VGA-Karte unterliegen, sind diese Routinen entsprechend vereinfacht.

## B.4 Die Graphik-Ausgabe

Ist die VGA-Karte in den Graphik-Modus umgeschaltet, so kann ein Bildpunkt gesetzt werden, indem das zugehörige Segment berechnet und das darin liegende Byte (bzw. die Bits) des Bildpunktes beschrieben wird. Bei der Verwendung von 8 Bit Farbtiefe wird genau ein Byte für jeden Bildpunkt verwendet. Dies vereinfacht die Berechnung. Bei einer Auflösung von 640×480 Bildpunkten und einem Byte pro Bildpunkt werden genau 307200 Byte für den Bildschirmspeicher benötigt.

Die genaue Adresse zu einem Bildpunkt *P(s, z)* (s = Spalte, z = Zeile), bestehend aus der Segmentnummer und dem Index in diesem Segment, berechnet sich bei dieser Auflösung

```
Segment_Nummer = (int)(z * 640 + s) / 65536
Segment_Index  = (z * 640 + s) modulo 65536
```

In C läßt sich diese Berechnung noch vereinfachen, wie die Prozedur `PlotPixel()` zeigt.

Das komplette Bild steht in dem Vektor `Pictures[]`, indem sich auch die anderen verfügbaren Bilder befinden. Über die Bildnummer (`BildNr`), die maximale Bildgröße (`PIC_SIZE`), die Spaltengröße (`Bild[BildNr].Spalten`) sowie die Zeile und Spalte kann die genaue Adresse eines jeden Bildpunktes für jedes definierte Bild in dem Vektor berechnet, dieser Bildpunkt ausgelesen und auf dem Bildschirm ausgegeben werden (vergleiche Anhang B.5).

Da bei der gewählten Auflösung am Bildschirm kein Platz für die überdeckungsfreie Darstellung aller 4 verwendeten Bilder ist, werden nur zwei in der vollen Größe und zusätzlich alle verwendeten Bilder in einer verkleinerten Form (64×64 Pixel) als Icon angezeigt.

```
void Show_Pic(int BildNr, int Position)  /* Anzeige eines Bildes */
{
  unsigned char Color;             /* geklippter Grauwertindex   */
  long s, z;                       /* Variablen Spalten/Zeilen   */
  long Index, PositionOffset;

  /* Position = 1 : Bild wird links oben ausgegeben,             */
  /*          = 2 : Bild wird rechts oben ausgegeben.            */
  PositionOffset = (Position==1) ? 0:POS_OFFSET;

  /* Nur Bild 0 oder 1 werden komplett angezeigt. Alle anderen   */
  /* nur als Icon.                                               */
  if ((BildNr==0) || (BildNr==1)) {
     for (z=0; z<(long)Bild[BildNr].Zeilen; z++) {
         Index = z*(long)Bild[BildNr].Spalten +
                   (long)BildNr*(long)PIC_SIZE;
         for (s=0; s<(long)Bild[BildNr].Spalten; s++) {
             Color = Pictures[Index+s];
             /* Grauwert-Klipping durchfuehren, d.h. falls keine */
             /* Farben gewuenscht werden, die unteren Grauwerte  */
             /* (=Farben) auf SCHWARZ abbilden.                  */
             Color = (Color<CLIP_COLOR) ? SCHWARZ:Color;
             /* Bildpunkt an der gewuenschten Position setzen    */
             PlotPixel(s+PositionOffset, z, Color);
         }
     }

     /* Bild-Platz loeschen falls neues Bild nicht max. Groesse  */
     for (z=(long)Bild[BildNr].Zeilen; z<Z_RES_MAX; z++)
         for (s=0; s<S_RES_MAX; s++)
             PlotPixel(s+PositionOffset, z, SCHWARZ);
     for (s=(long)Bild[BildNr].Spalten; s<S_RES_MAX; s++)
         for (z=0; z<Z_RES_MAX; z++)
             PlotPixel(s+PositionOffset, z, SCHWARZ);
  }
```

```
  Show_Icon(BildNr);      /* Icon wird fuer jedes Bild angezeigt */
}

void PlotPixel(long Spalte, long Zeile, unsigned char Farbe)
{
  long Index;            /* Index bei der linearen Adressierung  */

  int  Segment;          /* Segment in dem der Pixel liegt < 5   */
  unsigned int SegIndex; /* Index im berechneten 64k-Segment     */

  Index    = (long)(SCREEN_SIZE * (long)(Zeile) + Spalte);
  Segment  = Index >> 16;    /* das betreffende 64k-Segment ber. */
  SegIndex = Index & 0xFFFF; /* High-Bits im Segment-Index ausbl.*/

  /* outport(0x3CD, Segment | 0x40);  Segment bei ET-3000        */
  outport(0x3CD, Segment);   /* Segment bei ET-4000 aktivieren...*/
  vgaram[SegIndex]=Farbe;    /* ...und betreffendes Pixel setzen */
}
```

## B.5 Hinweise zur Portierung der Algorithmen

Alle in dem Bildverarbeitungsprogramm verwendeten Routinen sind möglichst systemunabhängig entwickelt worden. Die PC-VGA-spezifischen Eigenarten wurden in wenigen Routinen zusammengefaßt (`Graph_Mode()`, `Text_Mode()`, `Set_LUT()`, `PlotPixel()`). Ansonsten sind alle anderen Routinen hardwareunabhängig und laufen auf allen Rechnern mit einem C-Compiler und genügend nutzbarem Hauptspeicher.

Aufgrund der 64 KByte Segmentierung und des unter DOS auf 640 KByte beschränkten Hauptspeichers wurden die Bilder nicht als zweidimensionales Feld, sondern als Vektor realisiert. In einigen Fällen werden Daten sogar in temporären Feldern auf die Festplatte ausgelagert (z.B. bei der Fourier-Transformation). Bestehen diese einschränkenden Randbedingungen nicht, z.B. auf einem UNIX-System, so können die Bilder als Ergänzung der Struktur `BildInfo` definiert werden.

```
typedef struct                  /* Struktur fuer die Bilder      */
{
int           Zeilen;         /* Aufloesung vertikal = Zeilen  */
  int           Spalten;        /* Aufloesung horizontal = Spalt.*/
  unsigned char Used;           /* Flag, ob Bild benutzt wird    */
  unsigned char Color;          /* Flag, ob Bild Farbe enthaelt  */
  unsigned char Bild[Z_RES_MAX][S_RES_MAX];         /* das Bild  */
  unsigned char Icon[ICON_SIZE][ICON_SIZE];         /* Bild-Icon */
} BildInfo;

#if !defined(MAIN)
    extern BildInfo  Picture[MAX_PICTURE];
#endif
```

Diese Art der Definition wurde schon bei den im Buch abgedruckten Algorithmen verwendet, was zusätzlich die Portierung erleichtern dürfte. Bei der Festlegung der Bildgröße sollte gleich ein entsprechender Rand (z.B. 10 Bildpunkte), der für alle Verfahren ausreichend ist, berücksichtigt werden.

```
...
unsigned char Bild[Z_RES_MAX + RAND][S_RES_MAX + RAND];
...
```

Eine Zeile in der ursprünglichen Form für den PC unter DOS mit

```
Pixel = Pictures[ (long)BildNr*(long)PIC_SIZE      /* Bildnummer */
                + z*(long)Bild[BildNr].Spalten     /* Zeile      */
                + s ];                             /* Spalte     */
```

muß dann in das Format

```
Pixel = Picture[BildNr].Bild[z][s];
```

umgeändert werden. Sollen größere Bilder, mehr Bilder, mehr Icons usw. verwendet werden, so ist dies einfach durch Änderung der Konstanten in der Datei `CONST.H` (siehe unten) und Recompilierung möglich.

Auf verschiedenen Systemen sind einige der Konstanten bereits definiert oder überflüssig. Weitere Hinweise zur Portierung und Optimierung sind in den Kommentaren der einzelnen Programme zu finden.

```
/************* CONST.H  fuer  DOS-PC  mit  VGA-Graphik ***********/
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <io.h>
#include <sys\stat.h>
#include <fcntl.h>
#include <graphics.h>
```

```
/************************* Konstanten-Definition ***************/
#define MAX_PICTURE    4        /* Maximale Anzahl von Bildern   */

#define S_RES_MAX    256        /* Maximale Spalten-Aufloesung   */
#define Z_RES_MAX    256        /* Maximale Zeilen-Aufloesung    */
#define PIC_SIZE   65536        /* =256x256; max. Bild-Groesse   */
#define REDUK_FAKT     4        /* Divisionsfaktor fuer Icons    */
#define ICON_SIZE     64        /* Groesse der Icons; 64x64      */

#define POS_OFFSET   320        /* horiz. Offset fuer Bild Nr. 1 */

#define GRAPH_MODE   0x2E       /* Graphik-Modus 640x480 / 256 F.*/
#define TEXT_MODE    0x03       /* Standard-Text-Modus 720x400   */
#define SCREEN_SIZE   640       /* horiz. Aufloesung Modus 0x2E  */

#define MAX_COLOR    256        /* max. Anzahl verschied. Farben */

#define ROT            0        /* Index fuer die Farben         */
#define GRUEN          1
#define BLAU           2
#define SCHWARZ        0
#define GRAU         128
#define WEISS        255
#define CLIP_COLOR     8

#define LINE_LENGTH  131

#define TRUE           1
#define FALSE          0

#define PCX_HEADERSIZE  128     /* Header-Groesse bei PCX-Bilder */

#define INT_NO_DEFAULT   -32760 /* "unbenutzter" int-Wert        */
#define FLOAT_NO_DEFAULT -1E+20 /* "unbenutzter" float-Wert      */

#define EPS_ERROR         1E-10 /* Genauigkeitsschranke float    */
#define PI         3.14159265359

#define _fmode   O_BINARY       /* Oeffnungs-Modus der Bilddatei */

#define NO_CURS_POS    0
```

```
/************************* Variablen-Definition *****************/
typedef struct                  /* Struktur fuer RBG-Farbanteile */
{
  unsigned char Rot, Gruen, Blau;
} RGB;

typedef struct                  /* Struktur fuer Bildinformation */
{
  int           Zeilen;         /* Aufloesung vertikal = Zeilen  */
  int           Spalten;        /* Aufloesung horizontal = Spalt.*/
  unsigned char Used;           /* Flag, ob Bild benutzt wird    */
  unsigned char Color;          /* Flag, ob Bild Farbe enthaelt  */
  unsigned char Icon[ICON_SIZE][ICON_SIZE];         /* Bild-Icon */
} BildInfo;

#if !defined(MAIN)
extern union  REGS  pc_cpu;     /* Strukturen zur Verwendung ... */
extern struct SREGS s_cpu;      /* ... der CPU-Register.         */
extern unsigned char far *vgaram;   /* Far-Pointer Bildschirmsp. */
extern unsigned char huge *Pictures;/* Huge-Pointer auf Bilder   */
extern BildInfo  Bild[MAX_PICTURE];
#endif
```

# C. Das PCX-Format

Um für den Test der einzelnen Bildverarbeitungsroutinen einen Zugriff auf umfangreiches Bildmaterial zu gewährleisten, wurde das auf dem PC-Sektor sehr verbreitete PCX-Datei-Format zur Kodierung und Abspeicherung der Bilder gewählt.

Damit ist es möglich, Bilder aus vorhandenen Zeichen- oder Malprogrammen oder Aufnahmen von Scannern, die in der Regel auch das PCX-Datei-Format als Ausgabeformat anbieten, zu verwenden. Zusätzlich können somit auch bearbeitete Bilder mit entsprechenden Programmen nachträglich mit Texten versehen oder sogar in Desktop-Publishing-Programmen verwendet werden. Eine Vielzahl von Programmen erlauben die Konvertierung in andere Dateiformate (GIF, TIFF, GEM usw.).

Das PCX-Datei-Format wurde von der Firma Zsoft-Corporation aus Marietta, USA entwickelt. Es war ursprünglich nur zur Speicherung und Übertragung der mit PC-Paintbrush erzeugten Graphiken gedacht, hat aber in der Zwischenzeit eine weite Verbreitung in die unterschiedlichsten Programme gefunden. Es existieren momentan 5 verschiedene Datei-Formate mit der Versionsnummer 0, 2, 3, 4 und 5, die die historische Entwicklung widerspiegeln.

Version 0 (PC-Paintbrush-Version 2.5) enthielt nur Monochrom- oder Vier-Farb-Bilder. Die Version 2 (PC-Paintbrush-Version 2.8) unterstützte die 16 Farben der EGA-Karte und die aktuelle Version 5 (PC-Paintbrush-Version 3.0 und höher sowie PC-Paintbrush IV Plus und Publisher's Paintbrush) kann jetzt 256 unterschiedliche Farben aus 16.7 Millionen oder sogar Echtfarben (24 Bit RGB; je 8 Bit für R,G und B) verarbeiten.

Da im Rahmen dieses Bildverarbeitungsbuches in der Regel nur Grauwertbilder mit bis zu 256 Graustufen Verwendung finden, wird die Version 5 mit 256 verschiedenen Farben (Grautönen) verwendet.

Bilder im PCX-Format können eine Größe von bis zu 65536×65536 Pixel haben. Aus Speicherplatz- und Geschwindigkeitsgründen werden hier nur Ausschnitte von bis zu 256×256 Pixel Größe verwendet. Korrekturfaktoren für die Gamma-Korrektur oder Farbinformationen für andere Farbmodelle außer dem RGB-Modell werden im PCX-Format nicht abgespeichert. Die verwendete Lauflängen-Kodierung kann bei gescannten Bildern unter Umständen eine Vergrößerung der Dateigröße bewirken und damit eine ineffiziente Speicherung der Bildinformation bedeuten.

## C.1 Das PCX-Datei-Format

Ein Bild im PCX-Datei-Format kann man grob in drei unterschiedliche Bereiche einteilen:

- den 128 Byte großen Header,
- das eigentliche Bild und

- die Farbpalette (256 Byte).

Im PCX-Datei-Header sind an fest definierten Stellen Angaben über Bildgröße, Bildart usw. zu finden. Das eigentliche Bild ist in einer Lauflängen-Kodierung abgelegt, wobei bei Echtfarbenbildern die drei Ebenen hintereinander folgen. Bei der 256-Farben-Palette endet das Bild 768 Byte vor Dateiende und die Farbpalette mit 256 RGB-Tripel folgt danach. Da die VGA-Graphik leider nur 64 Stufen für Rot, Grün und Blau unterstützt, müssen die Werte dieser Palette für die Ausgabe auf einer VGA-Graphikkarte durch 4 dividiert werden (vergleiche Anhang B).

| Byte | Anzahl | Bezeichner | Bedeutung |
|---|---|---|---|
| 0 | 1 | Hersteller | immer 10 (0Ah) = Zsoft PCX |
| 1 | 1 | Versionsnummer | Version 0, 2, 3, 4 oder 5 |
| 2 | 1 | Kodierung | 1 = Lauflängen-Kodierung (einzige Möglichkeit) |
| | | | (einzige Möglichkeit) |
| 3 | 1 | Bits pro Pixel | 1, 2, 4 oder 8 Bit (256 Farben) |
| 4 | 8 | Bildgröße | $X_{Min}$, $Y_{Min}$, $X_{Max}$, $Y_{Max}$ mit jeweils zwei Byte; horizontale Größe = $X_{Max}-X_{Min}+1$ vertikale Größe = $Y_{Max}-Y_{Min}+1$ |
| 12 | 2 | horizontale Auflösung | Punkte/Inch beim Ausdruck in horizontaler Richtung (ohne Bedeutung) |
| 14 | 2 | vertikale Auflösung | Punkte/Inch beim Ausdruck in vertikaler Richtung (ohne Bedeutung) |

Tabelle C.1: Der Aufbau des PCX-Datei-Headers, Teil 1. Werden für Zahlenangaben 2 Byte verwendet, so erfolgt zuerst immer die Angabe des High-Byte, dann das Low-Byte.

Aufgrund der Anzahl der Farbebenen läßt sich ablesen, ob eine Farbpalette verwendet wird oder nicht. Wird die Farbpalette verwendet, so kann mit Hilfe der Versionsnummer des PCX-Datei-Formats und der Anzahl der Bits pro Pixel die Art und Lage der Farbpalette (im Header oder am Dateiende) ermittelt werden.

Wird keine Farbpalette verwendet, stellen die Pixelwerte direkt die Farben dar. Setzt sich die Farbe eines Bildpunktes aus den drei Grundfarben Rot, Grün und Blau aus unterschiedlichen Ebenen zusammen, so erfolgt die Speicherung ebenenweise, zuerst die Rot-Ebene, dann die Grün-Ebene und zum Schluß die Blau-Ebene.

| Byte | Anzahl | Bezeichner | Bedeutung |
|---|---|---|---|
| 16 | 48 | Header-Palette | Farbpalette falls weniger als 256 Farben verwendet werden |
| 64 | 1 | reserviert | immer 0 |
| 65 | 1 | Farbebenen | Anzahl 1-4 möglich |
| 66 | 2 | Bytes/Zeile | Anzahl der Bytes/Zeile pro Farbebene |
| 68 | 2 | Interpretation | 1 = Farbe oder s/w; |
| | | der Palette | 2 = Grauwerte |
| 70 | 2 | horizontale Bildschirmgröße | Anzahl horizontaler Bildpunkte - 1 |
| 72 | 2 | vertikale Bildschirmgröße | Anzahl vertikaler Bildpunkte - 1 |
| 74 | 54 | Leer | Leer bis Byte 128 |

Tabelle C.2: Der Aufbau des PCX-Datei-Headers, Teil 2. Werden für Zahlenangaben 2 Byte verwendet, so erfolgt zuerst immer die Angabe des High-Byte, dann das Low-Byte.

Bei zwei Ebenen sind die Farben willkürlich, bei vier Ebenen setzen sie sich aus den Grundfarben und einer zusätzlichen Intensitätsangabe zusammen.

Wie schon erwähnt, erfolgt die Speicherung der eigentlichen Bildinformation in einem Lauflängen-kodierten Format. Die Kodierung wird maximal für jeweils eine Zeile vorgenommen, wobei kein Trennzeichen zwischen den Zeilen angegeben wird. Als Kennzeichen, daß eine Lauflängen-Kodierung folgt, müssen die beiden höchsten Bits des Bytes gesetzt sein. Die unteren 6 Bit geben in diesem Fall die Anzahl und das nachfolgende Byte die Farbe an. Aus dieser Art der Kodierung folgt, daß Pixel mit einem Farbwert größer als 127 immer Lauflängen-kodiert werden müssen. Dabei werden im ungünstigsten Fall pro Bildpunkt zwei Byte benötigt, was zu einer Vergrößerung der Bild-Datei im Vergleich zu einer unkodierten pixelweisen Abspeicherung führt.

| Bits/Pixel/Ebene | Anzahl Ebenen | Bedeutung |
|---|---|---|
| 1 | 1 | monochrom |
| 1 | 3 | 8-Farben |
| 1 | 4 | 16 Farben |
| 2 | 1 | 4-Faben, CGA-Palette |
| 4 | 1 | 16-Farben. EGA-Palette |
| 8 | 1 | 256 Farben, Palette am Dateiende |
| 8 | 3 | 16.7 Millionen Farben (Echtfarben) |

Tabelle C.3: Nähere Erklärung zu der Bedeutung der Bytes 3 und 65 aus Tabelle C.1 anhand einiger Beispiele.

Weiterhin sollte beachtet werden, daß die Bildgrößenangabe aufgrund dieser Speicherung an Byte-Grenzen endet. Werden weniger als 8 Bit pro Pixel verwendet, kann es daher sein, daß die Bilder künstlich auf Byte-Grenze vergrößert werden müssen.

Nach der Bildinformation folgt bei den 256-Farben-Paletten die Palette mit 256 Farbtripeln (Rot, Grün, Blau). Diese Einträge beinhalten nicht nur den unterscheidbaren Bereich von jeweils 64 Stufen, sondern den kompletten Wertebereich von 0 bis 255.

## C.2 Speicherung im PCX-Datei-Format

Da die Abspeicherung eines Grauwertbildes im PCX-Datei-Format einfacher ist als das Einlesen, soll zuerst diese Routine beschrieben werden.

Bei der Abspeicherung wird nur die PCX-Version 5 mit 256 Farben (Graustufen) verwendet. Der Header wird mit den Daten aus der Struktur `BildInfo` gefüllt oder mit entsprechenden Default-Werten belegt. Der Bild-Dateiname ist frei wählbar. Die Lauflängen-Kodierung (RLC) wird immer durchgeführt. Da die obersten beiden Bits als Kennzeichnung für die Kodierung auf 1 gesetzt werden müssen, kann das Vorkommen von maximal 63 identischen aufeinanderfolgenden Bildpunkten kodiert werden. Einzelne Bildpunkte mit einem Grau-/Farbwert größer als 191 müssen nach dieser Formatdefinition immer im RLC-Format kodiert werden.

```
int Save_PCX_Picture()
{
  int Z_Index, S_Index;                    /* PCX-Groessenangaben*/
  int Anzahl;
  unsigned char PCXHeader[PCX_HEADERSIZE]; /* PCX-Bild-Header    */
  unsigned char RLC[2];                    /* Hilfsvektor f. RLC */
  unsigned char LUT[768];                  /* Grauwert-Tabelle   */
  unsigned char Pixel;
  int PCX_fd;                              /* File-Descriptoren  */
  int i, j, z, s, BildNr;
  long Index;                              /* Adresse des Bildes */
  char Filename[LINE_LENGTH];              /* Name Ziel-Datei    */
  char *PCX_Ext  =".pcx";                  /* Extension immer PCX*/

  /*----------------------- Dateiname eingeben ------------------*/
  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Bild im PCX-Format speichern *");

  gotoxy(35,20);
  BildNr=Read_Int("Welches Bild Nr. (0=Abbruch): ", 1, ' ');
```

```
/* Gueltige Bild-Nummer eingegeben? - Und Bild auch vorhanden? */
if ((BildNr<=0) || (BildNr>MAX_PICTURE)) return(FALSE);
BildNr--;                /* die Zaehlung faengt immer bei 0 an */
if (!Bild[BildNr].Used) {
   ShowError("Kein Bild vorhanden.");
   return(FALSE);
}
gotoxy(35,22);
Read_String("Bitte Dateinamen eingeben: ", Filename);
if (!Add_Extension(Filename, PCX_Ext)) {
   ShowError("Fehler beim Anhaengen der PCX-Extension.");
   return(FALSE);
}
/* PCX-Datei neu erzeugen oder alte ueberschreiben.            */
if ((PCX_fd = open(Filename, O_CREAT | O_BINARY |
                    O_TRUNC | O_RDWR, S_IREAD | S_IWRITE)) < 0) {
   ShowError("Fehler beim Erzeugen der PCX-Datei.");
   return(FALSE);
}

/*----------------------- PCX-Header erzeugen -----------------*/
/* Erzeuge PCX-Header mit insgesamt 128 Bytes                  */
/* Bei PCX wird nicht die Anzahl der Bildpunkte sondern der    */
/* Index von ... bis verwendet.                                */
S_Index = Bild[BildNr].Spalten-1;
Z_Index = Bild[BildNr].Zeilen-1;

PCXHeader[ 0] = 10;       /* 10=0Ah = Zsoft PCX file           */
PCXHeader[ 1] =  5;       /* Versionsnr; 256 Farben / 16.7 Mio */
PCXHeader[ 2] =  1;       /* Encoding; 1=PCX-run-length-encod. */
PCXHeader[ 3] =  8;       /* Bits pro Pixel und Ebene          */
PCXHeader[ 4] =  0;       /* Bildgr.: Xmin = Low-/High-Byte    */
PCXHeader[ 5] =  0;
PCXHeader[ 6] =  0;       /* Bildgr.: Ymin = Low-/High-Byte    */
PCXHeader[ 7] =  0;
PCXHeader[ 8] = S_Index-256*(S_Index/256); /* Bildgroesse: Xmax*/
PCXHeader[ 9] = S_Index/256;
PCXHeader[10] = Z_Index-256*(Z_Index/256); /* Bildgroesse: Ymax*/
PCXHeader[11] = Z_Index/256;
                          /* Aufloesung S_Res = Xmax - Xmin + 1*/
                          /*            Z_Res = Ymax - Ymin + 1*/
PCXHeader[12] =  0;       /* Aufloesung beim Drucken; dpi in X */
PCXHeader[13] =150;       /* 150dpi; oft steht hier 300dpi     */
PCXHeader[14] =  0;       /* Aufloesung beim Drucken; dpi in Y */
PCXHeader[15] =150;       /* 150dpi; oft steht hier 300dpi     */

/* Farbtabelle fuer 16 oder weniger Farben initialisieren.     */
for (i=16; i<64; i++) PCXHeader[i] = 0;
```

```
    PCXHeader[64] =  0;       /* Reserviert; immer 0               */
    PCXHeader[65] =  1;       /* Anzahl der Ebenen                 */
    PCXHeader[66] = S_Index+1-256*((S_Index+1)/256); /* Bytes/Zeile*/
    PCXHeader[67] = (S_Index+1)/256;
    PCXHeader[68] =  2;       /* Farbtabelle als Grauton interpr.  */
    PCXHeader[69] =  0;
    PCXHeader[70] =  0;       /* Aufloesung des Bildschirms: unben.*/
    PCXHeader[71] =  0;
    PCXHeader[72] =  0;
    PCXHeader[73] =  0;
    /* unbenutze Bytes auf 0 setzen                               */
    for (i=74; i<128; i++) PCXHeader[i] = 0;

    for (i=0; i<256; i++)
        for (j=0; j<3; j++) LUT[i*3+j]=i;  /* hier: immer Graukeil */

    /* zuerst den zusammengestellten PCX-Header rausschreiben     */
    write(PCX_fd, PCXHeader, PCX_HEADERSIZE);

    /*----------------------- Bild-RLC berechnen ------------------*/
    for (z=0; z<=Z_Index; z++) {
        Index = (long)z * (long)Bild[BildNr].Spalten +
                          (long)BildNr*(long)PIC_SIZE;
        s=0;
        while (s<=S_Index) {
            Pixel=Pictures[Index+s]; /* erstes Vorkommen der Farbe */
            s++;
            Anzahl=1;
            while ((s<=S_Index) && (Pixel==Pictures[Index+s]) &&
                   (Anzahl<63)) {
                Anzahl++;
                s++;
            }
            /* Anzahl>1 oder Farbe>192, dann RLC durchfuehren     */
            if ((Anzahl>1) || (Pixel>=192)) {
               RLC[0] = 0xC0 + Anzahl;/* RLC-Kennung 2-High-Bit=11 */
               RLC[1] = Pixel;        /* danach folgt der Grauwert */
               write(PCX_fd, RLC, 2);
            }
            else {
               RLC[0] = Pixel;       /* ein Pixel mit Grauwert<192 */
               write(PCX_fd, RLC, 1);
            }
        }
    }
    /*----------------------- LUT noch herausschreiben ------------*/
    RLC[0]=12;        /* Kennzeichen fuer Bild-Ende und LUT-Beginn */
    write(PCX_fd, RLC, 1);
    write(PCX_fd, LUT, 768);        /* Farbtabelle (LUT; Graukeil) */
    close(PCX_fd);

    return(TRUE);
}
```

# C.3 Einlesen des PCX-Dateiformats

Das Einlesen eines Bildes im PCX-Datei-Format gestaltet sich komplizierter als das Herausschreiben, da die möglichen unterschiedlichen PCX-Formate mit der unterschiedlichen Anzahl von Farben berücksichtigt werden müssen.

```
int Read_PCX_Picture()
{
  int Anzahl;
  unsigned char PCXHeader[PCX_HEADERSIZE]; /* PCX-Bild-Header    */
  unsigned char RLC[2];                /* Hilfsvektor fuer RLC    */
  unsigned char Pixel;
  int PCX_fd;                          /* File-Descriptor         */
  int i, j, z, s, Bild_Nummer;
  long Index;
  char Filename[LINE_LENGTH];          /* Name Ziel-Datei (.PCX)  */
  char *PCX_Ext  =".pcx";
  int NoFileFound;
  int Z_Reduktion, S_Reduktion, Z_Res, S_Res;
  int Spaltenzahl, S_Cut, Z_Cut;       /* Begrenzung der Bildgr.  */
  unsigned char Temp_Buff[1300];       /* Hilfspuffer f. Einlesen */
  struct ffblk File_Blk;               /* Struktur des DIR-Blocks */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Bild im PCX-Format einlesen *");

  gotoxy(35,20);
  Bild_Nummer=Read_Int("In welches Bild Nr. (0=Abbruch): ",1,' ');

  /* Gueltige Bild-Nummer eingegeben?                               */
  if ((Bild_Nummer<=0) || (Bild_Nummer>MAX_PICTURE)) {
     return(FALSE);
  }
  Bild_Nummer--;                 /* die Zaehlung faengt bei 0 an */

  /* Der Name aller im Verzeichnis vorhandenen .PCX-Dateien ausg.*/
  ClearTextWindow(35,20,80,25);
  i=35;
  j=20;
  NoFileFound = findfirst("*.pcx", &File_Blk, 0);
  while (!NoFileFound) {
     WriteText( i, j, File_Blk.ff_name);
     NoFileFound=findnext(&File_Blk);
     j++;
     if (j==24) {
        j=20;
        i=i+15;
        if (i>70) NoFileFound=TRUE;
     }
  }
```

```
/*----------------------- Dateiname eingeben ------------------*/
gotoxy(35,25);
Read_String("Bitte Dateinamen eingeben: ", Filename);
if (!Add_Extension(Filename, PCX_Ext)) return(FALSE);

if ( (PCX_fd=open(Filename, O_RDONLY | O_BINARY)) == -1) {
   ShowError("Fehler beim Oeffnen der Datei.");
   return(FALSE);
}

/* PCX-Header einlesen, auf Datenformat 5 testen und ...       */
read(PCX_fd, PCXHeader, PCX_HEADERSIZE );
if ( (PCXHeader[0]!=10) || (PCXHeader[1]!=5) ||
     (PCXHeader[2]!= 1) || (PCXHeader[3]!=8) ) {
   ShowError("Falsches Datenformat.");
   return(FALSE);
}

/* ... die Bildmasse berechnen. Byte-Folge: High-Low-Byte      */
S_Res = 256*(int)PCXHeader[ 9]+(int)PCXHeader[ 8]+1;
Z_Res = 256*(int)PCXHeader[11]+(int)PCXHeader[10]+1;

/* Klipping der Zeilen und Spalten auf die max. verf. Groesse  */
if (S_Res>S_RES_MAX) S_Cut=S_RES_MAX;
else                 S_Cut=S_Res;
S_Reduktion=S_Res/S_Cut;
Spaltenzahl=S_Res;
if (S_Reduktion == 1) Spaltenzahl=S_Cut;
if (Z_Res>Z_RES_MAX) Z_Cut=Z_RES_MAX;
else                 Z_Cut=Z_Res;
Z_Reduktion=Z_Res/Z_Cut;

/*----------------------- RLC-Bild einlesen -------------------*/
Index = (long) (Bild_Nummer) * PIC_SIZE;
z=0;
while (z<Z_Res) {
      s=0;
      j=0;
      while (s<S_Res) {
            read(PCX_fd, RLC, 1);
            Anzahl=RLC[0];
            if (Anzahl > 192) {
               Anzahl=Anzahl-192;
               read(PCX_fd, RLC, 1);
               Pixel=RLC[0];
               for (i=0; i<Anzahl; i++) {
                   Temp_Buff[j++]=Pixel;
                   s++;
               }
             } else {
               Temp_Buff[j++]=Anzahl;
               s++;
             }
      }
```

```
/* Ist das Originalbild doppelt so gross wie das         */
/* darstellbare Bild, so wird ueber zwei Punkte gemittelt*/
        if ((z % Z_Reduktion) == 0) {
           if (S_Reduktion==2)
              for (j=0; j<Spaltenzahl; j+=S_Reduktion)
                 Pictures[Index++]=(Temp_Buff[j]+Temp_Buff[j+1])/2;
           else
              for (j=0; j<Spaltenzahl; j+=S_Reduktion)
                 Pictures[Index++]=Temp_Buff[j];
        }
        z++;
  }
  close(PCX_fd);

  /* Genaue Bildabmessungen in Struktur speichern               */
  Bild[Bild_Nummer].Zeilen =Z_Cut;
  Bild[Bild_Nummer].Spalten=S_Cut;
  Bild[Bild_Nummer].Used   =TRUE;
  Bild[Bild_Nummer].Color  =FALSE;

  Make_Icon(Bild_Nummer);
  Show_Pic(Bild_Nummer, Bild_Nummer+1);

  return(TRUE);
}
```

# D. Musterlösungen

## Kapitel 3

### Aufgabe 1

Zu berechnen ist der Winkel $\alpha$ (=Sehwinkel). Um über trigonometrische Funktionen diesen Winkel zu bestimmen, wird er halbiert, so daß ein rechtwinkliges Dreieck mit dem Durchmesser als Ankathete und dem Zapfenradius als Gegenkathete entsteht.

Mit dem Augendurchmesser von 20 mm und dem Zapfenradius von 2.5 µm ergibt sich für $\alpha/2$

$$\frac{\alpha}{2} = \arctan\left(\frac{0.0025}{20}\right)$$
$$= 0.00716°$$
$$\alpha = 0.01432°$$
$$\cong 0.86'$$

Das menschliche Auge hat demnach eine Winkelauflösung von rund einer Bogenminute (daß einzelne Zapfen einen etwas geringeren Durchmesser haben können, sei hier vernachlässigt).

Wenn ein Bild eines Objektes auf der Retina einen Zapfen am Rand gerade noch reizt (analoges Signal), so kann durch eine leichte Verschiebung des Objektes dieser Reiz stärker oder schwächer werden (sakkadische Augenbewegung). Mit dieser "zeitlichen" Information kann das Auge somit solche kleinen Verschiebungen auflösen, die wesentlich geringer als die theoretische Auflösung sind.

### Aufgabe 2

Das menschliche Auge kann im Durchschnitt (in Einzelfällen kann dies stark variieren)

- 30 Graustufen und
- 160 Farben

unterscheiden. Mit Variationen der Helligkeit, Sättigung und Graustufen sind es ca. 7 Millionen Farbtöne.

Bei Grauwert-Bildern ist eine Reduktion auf 32 (2er-Potenz) Graustufen zulässig, ohne daß die optische Qualität leidet. Bei kontinuierlichen Grauwertverläufen (z.B. Graukeil) kann eventuell eine etwas höhere Anzahl notwendig sein.

Bei der Farbwiedergabe genügen maximal 7 Millionen Farbtöne. In der Praxis wird hier noch eine weitere Reduktion möglich sein, ohne daß der Betrachter Unterschiede zu einem entsprechenden Bild mit Echtfarben erkennt. Die Beschränkung des darzustellenden Grauwert-/ Farbbereiches dient letztlich zur Reduktion der anfallenden Datenmenge.

## Aufgabe 2

1. Die Größe des Originaltextes kann nach dem Strahlensatz einfach durch das Verhältnis

$$0.8 / 20 = x / 500 \text{ mm}$$

$$x = 20 \text{ mm}$$

   berechnet werden. Als Textfläche ergibt sich 314 mm².

2. Auf 1 mm² befinden sich ≈ 160000 Zapfen. Die Fovea hat aber nur eine Gesamtfläche von rund 0.5 mm². Dort befinden sich also etwa 80000 Zapfen, auf die der Text abgebildet wird.

$$\begin{aligned} \text{Auflösung} &= (80000 \text{ Zapfen}) / (314 \text{ mm}^2) \\ &\approx 255 \text{ Zapfen / mm}^2 \\ &\approx 164516 \text{ Zapfen / Inch}^2 \\ \Rightarrow \quad &\approx 406 \text{ dpi} \end{aligned}$$

3. Damit ist das menschliche Auge bei diesem Betrachtungsabstand um den Faktor 406/300 ≈ 1.3 besser als übliche Laserdrucker. Aufgrund der nicht immer optimalen Kontrastverhältnisse ist die praktische Auflösung etwas geringer (vergleiche Kapitel 6).

# Kapitel 4

## Aufgabe 1

Unter der Annahme, daß die lichtempfindlichen Bildpunkte der CCD-Kamera ohne Zwischenräume aneinandergrenzen ergibt sich damit eine minimale Objektgröße von

$$\text{Kleinstes Objekt} = 300\text{cm}/512 \approx 5.86\text{cm}$$

Aufgrund von Linsenfehlern (Unschärfe, Dispersion) und da die effektive Sensorgröße kleiner ist, d.h. zwischen den Zellen existieren größere Zwischenräume, ist die Ortsauflösung in der Praxis wesentlich geringer.

## Aufgabe 2

Die Chipfläche des CCD-Sensors ist 338,56mm² groß. Die aktive Sensorfläche hingegen ist nur

$$\text{Aktive Fläche} = 2048 * 2048 * 8\mu\text{m} * 8\mu\text{m} \approx 268.44\text{mm}^2$$

Die aktive Fläche ist also nur ca. 79.3% der realen Fläche. Hochfrequente Bildanteile können daher zwischen den einzelnen Sensorelementen verloren gehen. Dies entspricht einer Tiefpaßfilterung.

## Aufgabe 3

Die optische Dispersion beruht auf der Tatsache, daß Linsen Lichtstrahlen verschiedener Wellenlängen nicht gleich stark brechen, wodurch weißes Licht beim Durchgang durch eine Linse in seine Spektralfarben aufgeteilt wird. Die Stärke des Effektes hängt von der Größe des Winkels ab, um den das Licht gebrochen wird.

Unter Farbfehler wird die unterschiedlichen Durchlässigkeit des Linsenmaterials für verschiedene Spektralfarben verstanden. Die Linsen wirken somit wie ein "Farbfilter". Farbfehler treten bei jedem Winkel gleich stark auf.

## Aufgabe 4

1. Das digitalisierte Dia besteht aus 24000/25 = 960 Zeilen und 36000/25 = 1440 Spalten.
2. Insgesamt sind dies somit 960 * 1440 = 1382400 Bildpunkte.
3. Ein Bildpunkt besteht aus einem RGB-Tripel. Da für die Information zu jeder Farbe ein Byte benötigt wird, belegt der Farbbildpunkt 3 Byte.
4. Zur Speicherung des kompletten Bildes werden 3 * 1382400 = 4147200 Byte benötigt.

# Kapitel 5

## Aufgabe 1

Zwischen dem Schreiben zweier aufeinanderfolgender Bilder muß der Bildpunkt aus dem ersten Bild verloschen sein, bevor der neue Bildpunkt geschrieben wird. Unter "verloschen" wird hier die Unterschreitung der 10%-Leuchtdichte-Schwelle verstanden.

Damit darf die Persistenzzeit 1/Bildwiederholfrequenz, das ist gleich 1/70 Sekunden ≈ 14 ms nicht überschreiten.

## Aufgabe 2

Die Bandbreite des Systems berechnet sich aus der Anzahl übertragener Bildpunkte pro Sekunde.

$$\text{Bandbreite} = 4096 * 3300 * 25\ \text{Hz} = 337.92\ \text{MHz}$$

Alle Bildpunkte müssen 25 Mal pro Sekunde aus dem Bildschirmspeicher ausgelesen werden. Daher ist die maximale Zeit

$$\text{Auslesezeit} < 1/\text{Bandbreite} \approx 3\ \text{ns}$$

## Aufgabe 3

Die Größenangabe 14" des Monitors ist die Angabe über die Bildschirmdiagonale. Die maximal sinnvolle Auflösung berechnet sich aus der maximal unterscheidbaren Anzahl von Bildpunkten. Diese Größe ist durch den Loch-Abstand der Lochmaskenröhre festgelegt.

Aus den Angaben berechnet sich die Breite und Höhe des Monitors wie folgt:

$$(4a)^2 + (3a)^2 = (14 * 25.4 \text{ mm})^2$$

Als Grundgröße *a* ergibt sich somit

$$a \approx 71.12 \text{ mm}$$

und dadurch für die Breite 284.48 mm und die Höhe 213.36 mm. Mit diesen Abmessungen werden bei einem Loch-Abstand von 0.28 mm maximal 1016×762 Bildpunkte unterschiedlich dargestellt. Wird eine höhere Auflösung verwendet, so fallen bei der Wiedergabe Bildpunkte zusammen.

Da in sich die Größenangabe von Monitoren auf die maximale Bildschirmdiagonale und nicht auf den geringeren effektiv nutzbaren Bereich beziehen, ist in der Praxis die maximal sinnvolle Auflösung noch geringer als hier berechnet.

## Aufgabe 4

Die Gamma-korrigierten Grauwerte sind über die bekannte Formel zu berechnen. Die Tabelle D.1 zeigt den interessanten Anfangs- und Endteil der neuen LUT.

| Original-Intensität | Gamma-LUT-Eintrag |
|---|---|
| 0 | 0 |
| 1 | 25 |
| 2 | 34 |
| 3 | 40 |
| 4 | 45 |
| 5 | 50 |
| 6 | 53 |
| 7 | 57 |
| 8 | 60 |
| 9 | 63 |
| ... | ... |
| 50 | 129 |
| 51 | 130 |
| ... | ... |
| 251 | 253 |
| 252 | 254 |
| 253 | 254 |
| 254 | 255 |
| 255 | 255 |

Tabelle D.1: In der neuen LUT stehen statt 256 nur noch 180 unterschiedliche Graustufen zur Verfügung.

## Aufgabe 5

Der Grauwert eines Bildpunktes berechnet sich aus allen sichtbaren Teilen von Objekten, die diesen Bildpunkt überdecken. In dieser Aufgabe also von dem Flächenanteil des Grauwertes der Linie (Objekt) und vom Grauwert der verbleibenden Fläche des Bildpunktes (Grauwert des Bildpunktes oder des Hintergrunds).

Da eine exakte Berechnung der Flächenanteile in der Praxis zu aufwendig ist, wird das Verhältnis über sogenannte Subpixelmasken angenähert. Alle die Subpixel, deren Mittelpunkt durch ein Objekt bedeckt ist, erhalten den Grauwert des Objekts. Die Verhältnisse der Häufigkeiten geben die Verhältnisse zur Mischung des neuen Grauwertes des Bildpunktes wieder. Je feiner die Unterteilung in Subpixel ist, desto genauer werden die einzelnen Flächenanteile approximiert.

Beispiel:
Ein quadratisches Pixel wird in vier quadratische Subpixel unterteilt. Eine Linie mit dem Grauwert $G_1$ verläuft durch das Pixel mit dem Grauwert $G_2$ und bedeckt die Mittelpunkte von 3 Subpixeln. Der neue Grauwert $G$ zur Vermeidung von sichtbaren Alias-Effekten setzt sich dann zusammen aus

$$G = 0.75\ G_1 + 0.25\ G_2.$$

# Kapitel 6

## Aufgabe 1

1. Bei dem einfachen Schwellwertverfahren wird der Grauwert eines jeden Bildpunktes mit der Schwelle $T$ ($T$ = 127) verglichen. Ist er größer, so wird der Punkt auf Weiß (= 255), ansonsten auf Schwarz (= 0) gesetzt. Es ergibt sich dann

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 255 | 0 | 0 | 255 | 0 | 0 | 255 | 0 | 255 |
| 255 | 0 | 0 | 255 | 0 | 0 | 255 | 0 | 255 |
| 255 | 0 | 0 | 255 | 0 | 0 | 255 | 0 | 0 |
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |

2. Bei der Error-Diffusion wird die Dithermatrix über das gesamte Bild geschoben, der Binärwert aufgrund der Schwelle $T$ ($T$ = 127) berechnet und der dabei gemachte Fehler mit der entsprechenden Gewichtung durch die Dithermatrix auf die nachfolgenden (vier Nachbar-) Pixel verteilt. Das Ergebnis ist dann

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 255 | 0 | 0 | 255 | 0 | 0 | 255 | 0 | 255 |
| 255 | 0 | 0 | 255 | 0 | 0 | 255 | 0 | 255 |
| 255 | 0 | 0 | 255 | 0 | 0 | 0 | 0 | 0 |
| 0 | 255 | 0 | 0 | 255 | 0 | 255 | 255 | 0 |

3. Am linken, rechten und unteren Rand kann der Fehler nicht auf 4 Nachbarn, sondern nur auf drei bis ein Nachbar(n) verteilt werden. Entweder wird er so verteilt, als ob immer 4 Nachbarn vorhanden wären, oder die Gewichtung des Fehlers wird am Rand derart geändert, daß immer 100% des Fehlers verteilt werden.
4. Das Element oben links in der Dithermatrix ist ausgespart, da eine Änderung des vorigen Elements keinen Sinn macht. Außerdem sind die horizontalen und vertikalen Elemente (7 & 5) stärker gewichtet als die diagonalen, so daß sich in diese Richtungen früher eine Änderung ergibt, wenn die Original-Grauwerte in der Nähe des Schwellwertes sind. Dadurch wird die Entstehung von horizontalen/vertikalen Strukturen, die das menschliche Sehsystem bevorzugt wahrnimmt, etwas vermindert.

## Aufgabe 2

Bei der Konstruktion von Grautonmustern sollten folgende Randbedingungen beachtet werden:

- Mit den Grautonmustern sollen unterschiedliche Punktgrößen simuliert werden. Daher müssen die Muster dicht zusammenhängend sein.
- Es sollten möglichst keine regelmäßigen Strukturen durch die Muster entstehen.
- Diagonale Strukturen werden vom menschlichen Sehsystem mit geringerer Intensität wahrgenommen als horizontale/vertikale Strukturen. Es sollten daher möglichst die diagonal liegenden Punkte zur Erzeugung der Muster verwendet werden.
- Beim Ausdruck überlappen sich die einzelnen Pixel. Der Schwärzungsgrad ist daher nicht linear.

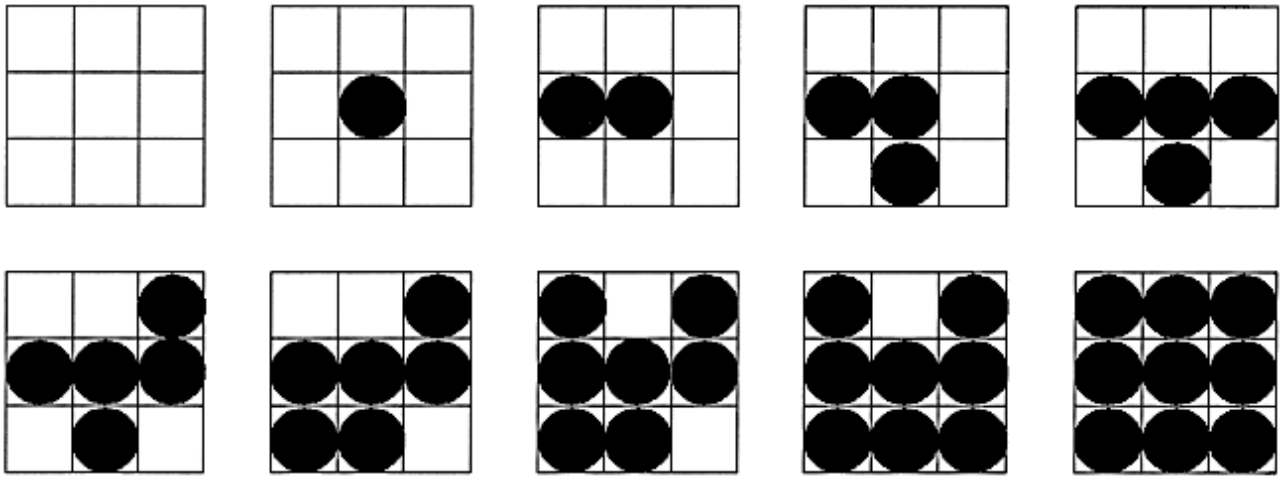

Abbildung D.1: Beispiel für ein Grautonmuster mit 10 Stufen.

# Kapitel 8

## Aufgabe 1

Eine Drehung des Objekts im Ortsraum bewirkt eine Drehung des Fourier-Spektrums im Frequenz-Raum in derselben Richtung und um denselben Winkel.

## Aufgabe 2

Die Fourier-Transformierte von *f(x)*:

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-i2\pi ux}\, dx$$

$$= \int_{0}^{X} A e^{-12\pi ux}\, dx$$

$$= -\frac{A}{i2\pi u}\left[e^{-2\pi ux}\right]_{o}^{X}$$

$$= -\frac{A}{i2\pi u}\left[e^{-2\pi uX} - 1\right]$$

$$= \frac{A}{i2\pi u}\left[e^{2\pi uX} - e^{-2\pi uX}\right]e^{-2\pi uX}$$

Das Fourier-Spektrum berechnet sich dann aus:

$$\left|F(u)\right| = \left|\frac{A}{\pi u}\right| \left|\sin(\pi u X)\right| \left|e^{-i\pi uX}\right|$$

wobei $\left|e^{-i\pi uX}\right| = 1$, so daß

$$\left|F(u)\right| = AX \left|\frac{\sin(\pi u X}{\pi u X}\right|$$

# Kapitel 9

## Aufgabe 1

1. Die Hadamard-Matrix der Größe 4 hat folgendes Aussehen:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

2. Die Hadamard-Transformation einer Folge *f(i)*, $i = 0, 1, 2, 3$ berechnet mit der Matrix als

$$H(0) = f(0) + f(1) + f(2) + f(3)$$

$$H(1) = f(0) - f(1) + f(2) - f(3)$$

$$H(2) = f(0) + f(1) - f(2) - f(3)$$

$$H(3) = f(0) - f(1) - f(2) + f(3)$$

Für die Berechnung werden 12 Additionen benötigt. Durch das paarweise Zusammenfassen von Werten bei der Berechnung läßt sich die Anzahl der Additionen auf 8 erniedrigen.

$$h_0 = f(0) + f(2)$$

$$h_1 = f(0) - f(2)$$

$$h_2 = f(1) + f(3)$$

$$h_3 = f(1) - f(3)$$

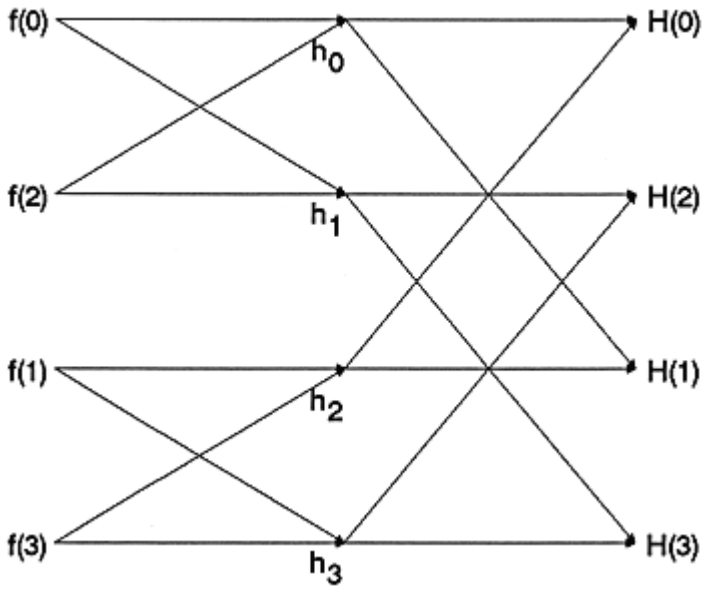


Abbildung D.2 Netzwerk für die Verknüpfung bei der Hadamard-Transformation.

Mit diesen Zwischenwerten werden im zweiten Schritt die endgültigen Werte berechnet:

$$H(0) = h_0 + h_2$$

$$H(1) = h_0 - h_2$$

$$H(2) = h_1 + h_3$$

$$H(3) = h_1 - h_3$$

In beiden Stufen wird dasselbe Verknüpfungsschema verwendet. Das zugehörige Netzwerk ist im folgenden Bild dargestellt. Durchgezogene Linien bedeuten eine Addition der Elemente, gestrichelte Linien eine Subtraktion.

# Kapitel 10

## Aufgabe 1

Die folgende Tabelle zeigt die Lösungen zu dieser Aufgabe. Verschiedene Punkte können nicht klar zur einen oder anderen Gruppe zugeordnet werden.

| Verfahren | lokal | global | linear | nichtlinear | Ortsraum | Frequenzraum |
|---|---|---|---|---|---|---|
| GW-Verschiebung | (x) | (x) | x | | x | |
| GW-Änderung LUT | | x | | x | x | |
| GW-Quantisierung | (x) | (x) | (x) | (x) | x | |
| Histogrammeinebn. | | x | | x | x | |
| Adaptive Histogr. | (x) | (x) | | x | x | |
| Mittelwertfilter | x | | x | | x | |
| Medianfilter | x | | | x | x | |
| Tiefpaßfilter | x | | x | | (x) | (x) |
| Hochpaßfilter | x | | x | | (x) | (x) |
| Kantenverstärkung | x | | x | | x | |

(Die Abkürzung GW steht für Grauwert)

## Aufgabe 2

1. Bei dem Filter handelt es sich um die Maske des Mittelwertfilters. Damit werden Störungen durch Mittelung der Grauwerte vermindert.
2. Der Vorfaktor 1/9 dient zur Normierung. Ohne diesen Faktor würden ansonsten Werte außerhalb des zur Verfügung stehenden Bereiches entstehen.
3. Die Summe aller Koeffizienten ist bei jeder Glättungsmaske 1, damit der mittlere Grauwert im Bild erhalten bleibt.

# Aufgabe 3

Durch einfache Berechnung kann bewiesen werden, daß der Medianfilter nicht linear ist. Es ist

$$f = \{0, 7, 8, 2, 1\} \qquad g = \{ 6, 7, 6, 4, 8\}$$

$$\Rightarrow \quad \text{med}(f) = 2 \qquad \text{med}(g) = 6$$

jedoch ergibt sich aus

$$f + g = \{ 6, 14, 14, 6, 9\}$$

$$\Rightarrow \quad \text{med}(f+g) = 9 \quad \neq \quad \text{med}(f) + \text{med}(g) = 8$$

# Aufgabe 4

| **Median-Filter** | **Mittelwertfilter** |
|---|---|
| + keine neuen Grauwerte | – erzeugt neue Grauwerte |
| + Kanten bleiben erhalten | – Kanten verwischen ("unscharf") |
| – feine Strukturen gehen verloren | – feine Strukturen gedämpft |
| – hohe Rechenzeit | + kurze Rechenzeit |
| + Störimpulse gut unterdrückt | – Störimpulse werden gedämpft |
| – Bildverschiebung um 1 Pixel | |
| | – wirkt optisch schlechter |
| Mittlerer Grauwert ändert sich | Mittlerer Grauwert bleibt gleich |
| + Einfluß durch Filterform möglich | + Einfluß durch Filterform möglich |

# Aufgabe 5

Der neue Grauwert $G'$ berechnet sich nach

$$G' = \begin{cases} 3G & 0 \le G < 50 \\ G + 100 & 50 \le G < 100 \\ \frac{55}{70}(G - 100) + 200 & 100 \le G < 170 \end{cases}$$

# Aufgabe 6

Das Aussehen der Masken ist:

horizontale Lücken detektieren / schließen

¼

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | -2 | 1 |
| 0 | 0 | 0 |

½

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |

vertikale Lücken detektieren / schließen

¼

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | -2 | 0 |
| 0 | 1 | 0 |

½

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |

# Kapitel 11

## Aufgabe 1

Die Form des Sobel-Operators reagiert auf vertikale Kanten. Da es sich hier aber um eine horizontale 1-Pixel breite Linie handelt, wird diese durch den symmetrischen Gradienten in dem Operator nicht erfaßt. Das Ergebnis lautet:

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Der zweite Operator ist eine Variante des Laplace-Operators. Dieser Operator ist näherungsweise isotrop und reagiert daher auf alle Richtungen von Kanten mit einem entsprechenden Ergebnis:

| | | | |
|---|---|---|---|
| -510 | -510 | -510 | -510 |
| 510 | 510 | 510 | 510 |
| -510 | -510 | -510 | -510 |
| 510 | 510 | 510 | 510 |

## Aufgabe 2

Der Laplace-Operator liefert für eine beliebig geneigte Ebene (Graukeil) an allen Stellen den Wert Null.

## Aufgabe 3

Aufgrund des eng begrenzten Grauwertbereiches für die einzelnen Bildanteile ergeben sich auch begrenzte, sich nicht überlappende Antwortbereiche bei der Filterung mit dem Laplace-Operator.

Sobald also als Filterantwort ein Wert im Bereich -1020 ... -910 berechnet wird, ist der mittlere Bildpunkt eine Störung und kann z.B. durch den Mittelwert der anderen 4 berücksichtigten Bildpunkte ersetzt werden.

| Unter der Filtermaske | Antwortbereich | | |
|---|---|---|---|
| nur Hintergrund | -60 | ... | +60 |
| nur Text | -80 | ... | +80 |
| Text und Hintergrund | -315 | ... | -160 |
| | -550 | ... | -400 |
| | -785 | ... | -640 |
| | 235 | ... | 300 |
| Hintergrund und Störung | -1020 | ... | -910 |

## Aufgabe 4

Die erste Filtermaske liefert bei vertikalen Linien der Breite 1 als Ergebnis 2, sofern die Linie komplett durch die Maske geht, ansonsten den Wert 1. Bei horizontalen Linien wird als Antwort -2 oder -1 erzeugt. Bei diagonalen Linien bleibt das Ergebnis immer gleich Null.

Die zweite Filtermaske erzeugt bei horizontalen und vertikalen Linien als Antwort 0, 2 oder -2, je nachdem wieviele Pixel der Maske durch die Linie bedeckt sind. Bei diagonalen Linien wird als Ergebnis 1, 5 oder 6 erzeugt.

# Kapitel 12

## Aufgabe 1

Aufgrund der Definition ist das ideale Skelett eines Kreises genau der Mittelpunkt und das Skelett eines Quadrates ein großes "X". Das nachfolgende Bild zeigt das Ergebnis.

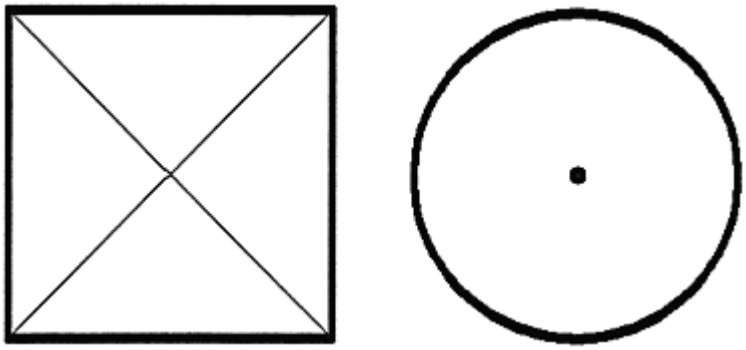

Abbildung D.3: Das Aussehen des Skeletts eines Kreises und eines Quadrats.

## Aufgabe 2

Bei dem Verfahren von Zhang/Suen werden gleichmäßig von allen Seiten die Randpunkte abgetragen, bis nur noch das Skelett übrig bleibt. In jedem Durchlauf werden Punkte von links oben und rechts unten her abgetragen. Die maximale Anzahl der Durchläufe ist daher

$$\text{Anzahl Durchläufe} = \lfloor \text{maximale Objektdicke} / 2 \rfloor$$

Dies ist auch genau die maximale Verkürzung. Die Dicke des Buchstaben "H" ist 5 und somit beträgt die Verkürzung 2 Bildpunkte.

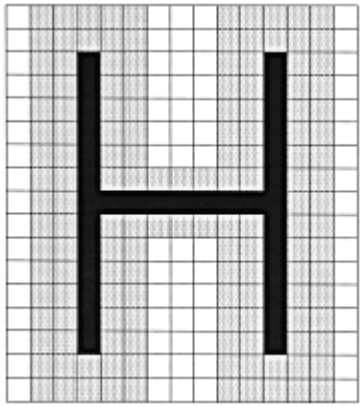

Abbildung D.4: Verkürzung des Buchstabens "H" bei der Skelettierung.

## Aufgabe 3

Die Anzahl der Bildpunkte ist $f = 4$, die Anzahl der Kanten $k = 14$ und die Anzahl der Ecken $e = 11$. Somit ergibt sich für die Euler-Charakteristik

$$E = 11 - 14 + 4 = 1$$

Wird der mittlere Punkt entfernt, so ändern sich die Werte in $f = 3$, $k = 11$ und $e = 10$. $E$ ist dann

$$E = 10 - 11 + 3 = 2$$

Somit handelt es sich bei dem mittleren Punkt um einen Skelettpunkt und darf daher nicht entfernt werden.

# Kapitel 13

## Aufgabe 1

1. Dieser Kettenkode beschreibt ein gleichschenkliges, rechtwinkliges Dreieck.
2. Mit dem differentiellen Kettenkode (nicht variablen) wird die Kurve durch 000000-300-200 beschrieben. Im Bit-kodierten variablen Kettenkode lautet die Beschreibung 000000011111100001111.
3. Achsenparallele Rechtecke werden durch folgende Formel für den Kettenkode charakterisiert:

   $$\textit{Kettenkode} = 0a\ 6b\ 4a\ 2b \quad \text{oder} \quad = 0a\ 2b\ 4a\ 6b$$

   wobei der Schreibweise $0a$ besagt, daß die Richtung 0 a-mal wiederholt wird. Bei diesem Kode kommen nur die Richtungen 0, 2, 4 und 6 vor. Die Gesamtzahl der Kettenelemente ist geradzahlig.

## Aufgabe 2

Bei dem beschriebenen Verfahren zur direkten Vektorisierung werden einzelne Punkte nicht berücksichtigt. Diese einzelnen Punkte stellen nur Störungen (Rauschen) dar. Das Weglassen führt somit zu einer Herausfilterung der Störungen.

Aufgrund eines Fehlerterms *e* wird bei der Vektorisierung entschieden, ob ein neuer Punkt als Verlängerung des bisher berechneten Vektors angesehen werden kann oder nicht. Der maximal zulässige Fehler ist so festgelegt, daß Schwankungen von ±1 Pixel erlaubt sind. Solche Schwankungen führen also nicht zum Abbruch der Vektorisierung. Eine vertikale Linie, die vereinzelt einen Bildpunkt nach rechts oder links versetzt hat, wird als eine einzige vertikale Linie vektorisiert. Damit ist eine Glättung durchgeführt worden.

# Kapitel 14

## Aufgabe 1

Um Lücken in Konturen beim Füllen zu schließen, muß das Füllelement für den Füllalgorithmus mindestens ein Pixel größer sein als die größte Lücke. In dieser Aufgabe muß das Füllelement demnach mindestens die Größe 3×3 haben. Größere Elemente wären zwar auch möglich, würden aber auch zu größeren Ungenauigkeiten führen.

Nachdem ein Objekt nun mit dem Grundelement "gefüllt" wurde, ist das nächstkleinere Element zu verwenden. Dies ist hier ein Füllelement der Größe 2×2. Dieses muß immer direkt an ein bereits vorhandenes größeres Element angelagert werden. Die Fläche kann sich damit maximal um zwei Pixel in allen Richtungen ausdehnen.

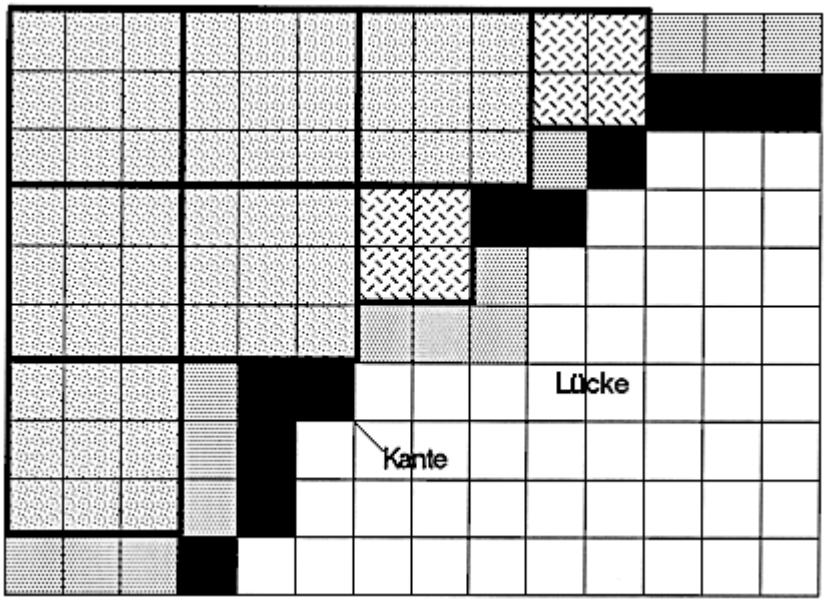


Abbildung D.5: Durch das stufenweise Füllen mit immer kleineren Füllelementen wird ein Auslaufen bei Lücken verhindert. In dem hier gezeigten schlechtesten Fall entstehen nur kleine Auswüchse.

Zum Schluß muß dann mit dem kleinsten Füllelement, dem einzelnen Pixel, der Rest ausgefüllt werden. Diese einzelnen Pixel dürfen sich wiederum nur direkt an die nächstgrößeren Füllelemente anlagern.

Aufgrund dieser Vorgehensweise kann bei Lücken die Füllfläche maximal drei Pixel weit auslaufen (siehe Abbildung D.5).

## Aufgabe 2

(siehe Abbildung D.6)

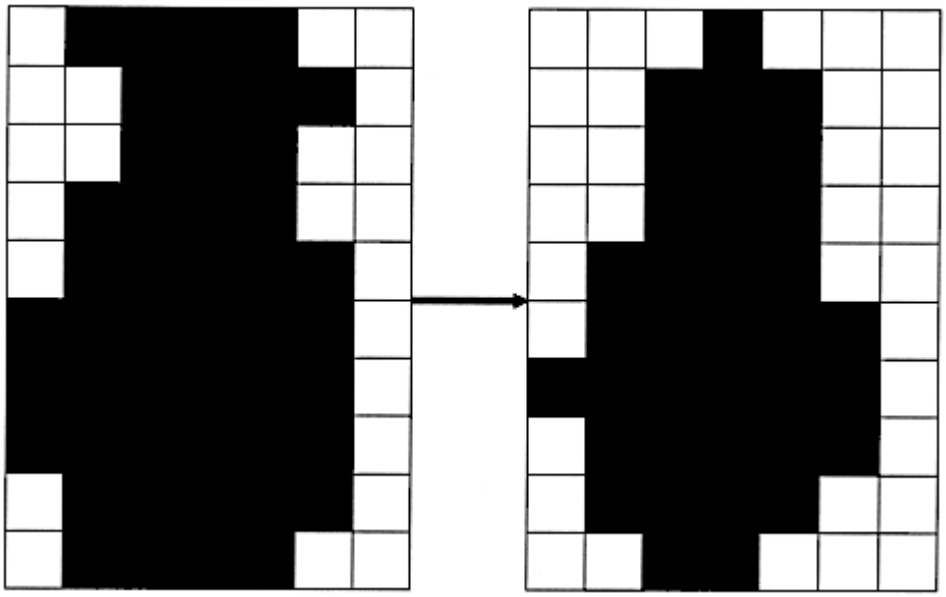

Abbildung D.6: Durch die Erosion mit einer geringeren Schwelle *T* werden nicht so viele Pixel gelöscht. Die Ergebnisstruktur ist größer.

## Aufgabe 3

(siehe Abbildung D.7)

## Aufgabe 4

Da die Verbindungen nur bei dicht beieinanderliegenden Kanten auftritt, könnten die entstandenen kleinen Flächen mit Hilfe einer Dilatation komplett gefüllt werden. Die Schwelle *T* ist dazu entsprechend zu setzen. Die dadurch entstandenen flächenhaften Strukturen sind durch eine Skelettierung wieder auf 1-Pixel Breite zu verdünnen.

Nachteilig bei dieser Vorgehensweise ist das Verschmelzen von zwei Kanten zu einer und die Verschiebung der resultierenden Kante zur Mitte zwischen den beiden Ursprungskanten. Eine größere Ungenauigkeit entsteht. An anderen Stellen können unter Umständen ebenfalls falsche Verbindungen und Verschiebungen entstehen.

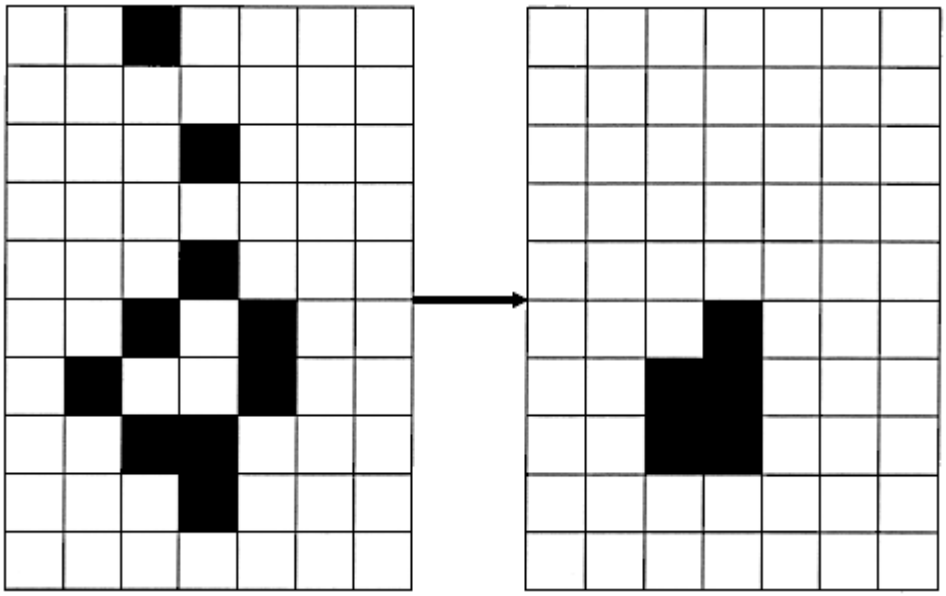

Abbildung D.7: Durch die Dilatation mit einer höheren Schwelle *T* werden nicht so viele Pixel übernommen. Die Ergebnisstruktur ist kleiner.

# Kapitel 15

## Aufgabe 1

Da nur 8 unterschiedliche Werte vorkommen, können die Grauwerte über einen Index mit 3 Bit kodiert werden.

| Grauwert | Index | Bit-Kodierung |
|---|---|---|
| 0 | 0 | 000 |
| 46 | 1 | 001 |
| 50 | 2 | 010 |
| 120 | 3 | 011 |
| 128 | 4 | 100 |
| 177 | 5 | 101 |
| 193 | 6 | 110 |
| 255 | 7 | 111 |

Für die Huffman-Kodierung sollten die Werte nach der Wahrscheinlichkeit sortiert sein. Der Aufbau des Kodebaums geschieht dann nach den allgemeinen Regeln. Das folgende Bild zeigt zwei Arten des Aufbaus. Ein Ast nach links wird dabei mit "0", einer nach rechts mit "1" kodiert.

Mittlere Kodelänge = 0.35*2 + 0.15*2 + 0.15*3 + 0.1*3 + 0.1*3 + 0.05*4 + 0.05*5 + 0.05*5

= 2.75

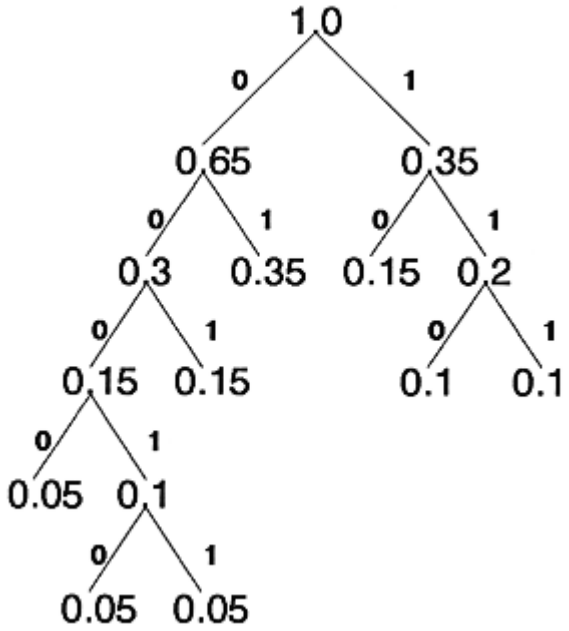


Abbildung D.8: Kodierung mit dem Huffman-Kode durch den Aufbau eines Kodebaums.

Für die Entropie $H$ ergibt sich nach der Formel ein Wert von $H \approx 2.66$. Da die Differenz zwischen mittlerer Kodelänge und Entropie relativ gering ist (2.75 - 2.66 = 0.09), handelt es sich bei dieser Kodierung um eine gute Komprimierung.

## Aufgabe 2

Bei einer zeilenorientierten Lauflängen-Kodierung wird am Zeilenende der Kode abgebrochen, egal ob in der nächsten Zeile derselbe Wert sich fortsetzt oder nicht. Zur besseren Übersicht werden Anzahl und Farbe jeweils durch Komma getrennt in Klammern geschrieben.

(5, 100) (5, 80)

(5, 100) (1, 128) (4, 92) (1, 255)

(2, 255) (2, 100) (1, 128) (3, 92) (2, 80)

(3, 255) (2, 100) (2, 70) (1, 60) (2, 50)

Bei der dritten und vierten Zeile ergibt sich keine Einsparung durch diese Kodierung. Insgesamt werden für das kodierten Bild 32 Bytes und für das Original-Bild 40 Byte benötigt.

## Aufgabe 3

Das Echtfarben-Bild der Größe 256×256 belegt 196608 Byte. Bei der JPEG-Kodierung (4:1:1) und der Quantisierung der DCT-Koeffizienten auf 9 Stufen mit einem Wertebereich von jeweils 12-Bit werden für ein 8×8-Feld noch

9 * 12 Bit + 1 * 12 Bit + 1 * 12-Bit = 132 Bit = 16.5 Byte

benötigt, gegenüber 192 im unkomprimierten Bild.

Werden die 9 DCT-Koeffizienten noch Huffman-kodiert, der Originalwert nur einmal am Bildanfang mitgeteilt und ansonsten nur der Huffman-Kode der Stufe, so ergibt sich eine weitere Datenkompression.

| Stufe | Wahrscheinlichkeit | Huffman-Kode |
|---|---|---|
| 1 | 0.4 | 0 |
| 2 | 0.2 | 100 |
| 3 | 0.08 | 1010 |
| 4 | 0.08 | 1100 |
| 5 | 0.06 | 1101 |
| 6 | 0.05 | 1110 |
| 7 | 0.05 | 1111 |
| 8 | 0.04 | 10110 |
| 9 | 0.04 | 10111 |

Die mittlere Kodelänge ist dabei 2.68, die Entropie ≈ 2.62. Für jedes 8×8-Feld werden je 12-Bit für den Cb- und den Cr-Wert und im Mittel 9*2.68 Bit = 24.12 Bit für die quantisierten und Huffman-kodierten DCT-Koeffizienten. Als durchschnittliche obere Grenze pro Feld ergibt sich damit ein Speicherbedarf von 25+12+12=49 Bit. Das Bild besteht insgesamt aus 32×32-Feldern, so daß sich mit den 9 unkodierten DCT-Koeffizienten als Kodetabelle für die Huffman-Kodierung der Gesamtspeicherbedarf für das JPEG-komprimierte Bild als

$$\text{Gesamtspeicherbedarf} = 9 * 12 \text{ Bit} + 32 * 32 * 49 \text{ Bit} = 50284 \text{ Bit} \approx 6286 \text{ Byte}$$

berechnet. Der Kompressionsfaktor ist ≈31.27.

# Kapitel 16

## Aufgabe 1

Alle aufgelisteten Merkmale sind lageunabhängig.

## Aufgabe 2

Der Umfang des Rechtecks beträgt 6, der Flächeninhalt 2. Die Kompaktheit K ist somit

$$K = \frac{6^2}{4\pi 2} \approx 1.43.$$

## Aufgabe 3

Der Zähler einer Akku-Zelle im Hough-Raum gibt theoretisch genau die Anzahl der Punkte an, die im Bild-Raum auf der durch die Koordinaten der Akku-Zelle definierten Geraden liegen.

Ist der Hough-Raum nicht fein genug unterteilt, so korrespondieren unterschiedliche Geraden im Bild-Raum mit derselben Akku-Zelle im Hough-Raum. Dadurch ist der Zähler der Akku-Zelle größer als theoretisch erwartet.

Ein kleiner Zahlenwert in der Akku-Zelle kann nur durch Ungenauigkeiten beim Berechnen der Geraden im Hough-Raum, entlang derer inkrementiert wird, entstehen. Kreise werden durch drei Parameter bestimmt, den Koordinaten *(x, y)* des Mittelpunktes und den Radius *r*. Demnach lautet die Parametrisierung bei Kreisen

$$(x - y)^2 + (y - b)^2 = r^2$$

Der Parameterraum muß bei drei Parametern dreidimensional sein.

## Aufgabe 4

In dem Ausschnitt treten nur die Grauwertübergänge 15-1, 1-10, 10-15 auf. Demnach werden in der Cooccurrenz-Matrix nur an drei Stellen von Null verschiedene Einträge zu finden sein.

| | 0 | 1 | 2 | ... | 10 | ... | 15 | ... |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | ... |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| ... | | | | | | | | ... |
| 10 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | ... |
| ... | | | | | | | | ... |
| 15 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | ... |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |

Die Anzahl der belegten Stellen in dieser Matrix gibt die Anzahl der unterschiedlichen Grauwertübergänge im Bild an, die Anzahl der unterschiedlichen Koordinaten der Einträge entspricht der Anzahl der Graustufen im Bild. Je weiter die Einträge von der Diagonalen wegliegen, desto kontrastreicher ist das Bild. Je mehr Einträge in dem unteren oder rechten Bereich der Cooccurrenz-Matrix liegen, desto heller ist das Bild.

Da der Ausschnitt nur 42 Bildpunkte (7×6) für die Berechnung enthält und jeder der drei Grauwertübergänge gleich oft vorkommt, ist auf ein regelmäßiges Muster im Originalbild zu schließen, bei dem die Grauwerte abwechselnd vorkommen. Aufgrund der Größe des Bildausschnitts können nur 7 Grauwertübergänge pro Zeile berücksichtigt werden. Die Anzahl der Zeilen des Ausschnittes ist ein Vielfaches der Anzahl der vorkommenden Grauwerte. Die Kenntnis über die Anzahl der Grauwertübergänge und die Anzahl der Zeilen erlaubt die Aussage, daß die Muster einer Zeile in der nächsten zyklisch verschoben auftauchen.

# E. Beispiel für ein Mustererkennungssystem

Als Abschluß soll an einem Beispiel verdeutlicht werden, wie die einzelnen in diesem Buch vorgestellten Bildverarbeitungsroutinen im Zusammenhang mit einem Mustererkennungssystem kombiniert und über die Zwischenergebnisse gesteuert werden können.

Das Beispiel ist aus dem Bereich der Medizin gewählt. Im ärztlichen Bereich gehören Blutbilduntersuchungen zur täglichen Routine. Diese monotone Arbeit soll nun durch ein Mustererkennungssystem unterstützt und erleichtert werden.

Die Bilddaten werden von einer auf einem Mikroskop befestigten CCD-Kamera geliefert, digitalisiert und dem Rechner als Grauwertbilder (Größe 512×512 Bildpunkte, 256 Graustufen) zugeführt. Mit Hilfe der Bildverarbeitung soll das Bild soweit aufbereitet werden, daß möglichst genaue Angaben über die Anzahl der roten und weißen Blutkörperchen, die relative Größe der roten Blutkörperchen und eventuell vorhandene unbekannte Zellarten gemacht werden.

Je nach Untersuchung sind verschiedene Vergrößerungsstufen am Mikroskop notwendig. Damit die aktuelle Vergrößerung nicht dem Mustererkennungssystem mitgeteilt werden muß, sollte sich dieses über einen festen Bereich automatisch anpassen. Die Größe der roten Blutkörperchen liegt daher in den Aufnahmen zwischen 5% und 25% der Gesamtbildgröße.

Die einzelnen Blutbestandteile sind so auf dem Objektträger verteilt, daß es zu möglichst wenigen Überlappungen kommt. Jedoch ist diese Vereinzelung nicht immer gewährleistet. Eine weitere Schwierigkeit ist die nicht ganz gleichmäßige Ausleuchtung des Bildes (am Rand dunkler als in der Bildmitte; siehe Abbildung E.1).

## E.1 Vorwissen – Merkmale

Für eine gute Mustererkennung muß möglichst viel Vorwissen in das System eingearbeitet oder in Datenbanken abgelegt werden. Gleichzeitig sollte aber möglichst wenig Information verwendet werden, damit das System möglichst universell einsetzbar ist.

Die bei diesem Beispiel der Blutbilduntersuchung vorhandene Information beschränkt sich daher auf folgende Punkte:

- der Aufnahmehintergrund ist hell;
- in den Aufnahmen ist eine repräsentative Anzahl von roten Blutkörperchen vorhanden (mindestens 10 Stück);
- rote Blutkörperchen sind annähernd kreisförmig, durch die Aufnahmeverzerrung hier leicht ellipsoid;
- die Größenstreuung der roten Blutkörperchen ist gering;

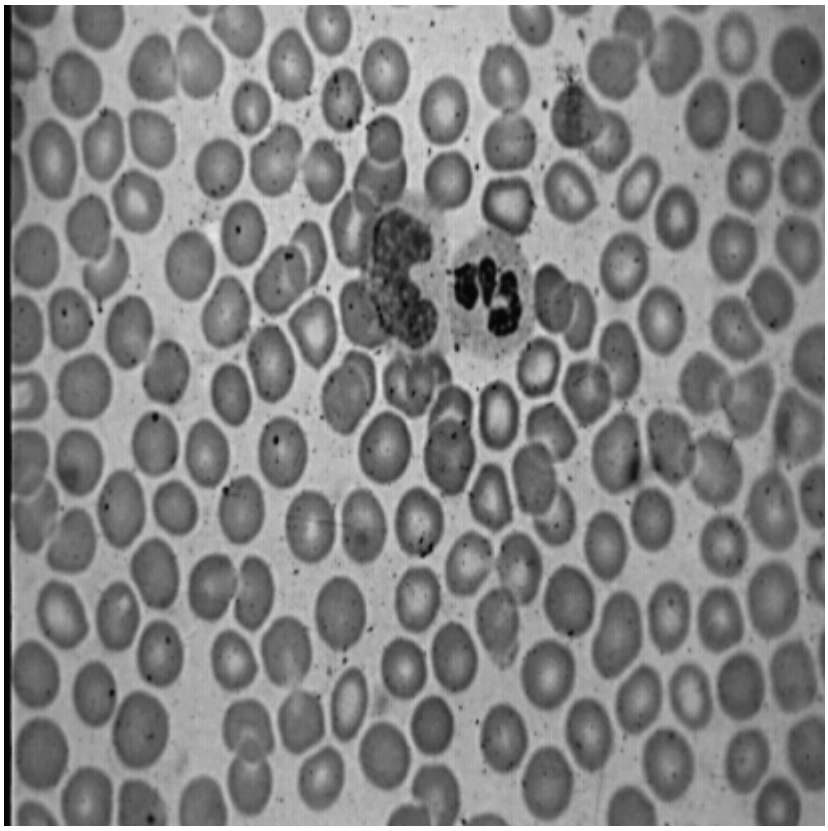

Abbildung E.1: Ein Beispiel für eine Aufnahme des Blutbildes mit einer Größe von 512×512 Bildpunkten. In der Bildmitte sind weiße Blutkörperchen zu erkennen. Ansonsten sind nur rote Blutkörperchen und einige kleinere Störungen in der Aufnahme enthalten. In der Mitte ist die Aufnahme etwas heller als am Rand. Die verwendete Kamera liefert keine quadratischen, sondern rechteckige Bildpunkte (Verhältnis 4:3), so daß das Bild leicht vertikal gestreckt ist.

- rote Blutkörperchen haben gut definierte Kanten, d.h. sie heben sich gegenüber dem Hintergrund deutlich ab.

Weitere, von der speziellen Aufnahme abhängige Informationen müssen während der Verarbeitung automatisch gewonnen und eventuell zur Steuerung der Verarbeitungsschritte verwendet werden.

## E.2 Bildvorverarbeitung

Aufgrund des geringen Vorwissens über die Aufnahme und die Bildbestandteile müssen zuerst sichere Referenzobjekte im Bild gesucht werden. Diese kann man dann zur Erzeugung standardisierter Bedingungen für die weiteren Analysen verwenden.

Da die roten Blutkörperchen sich relativ gut vom Hintergrund abheben, kann eine einfache Kantendetektion die gewünschte Separation der Referenzobjekte bewirken.

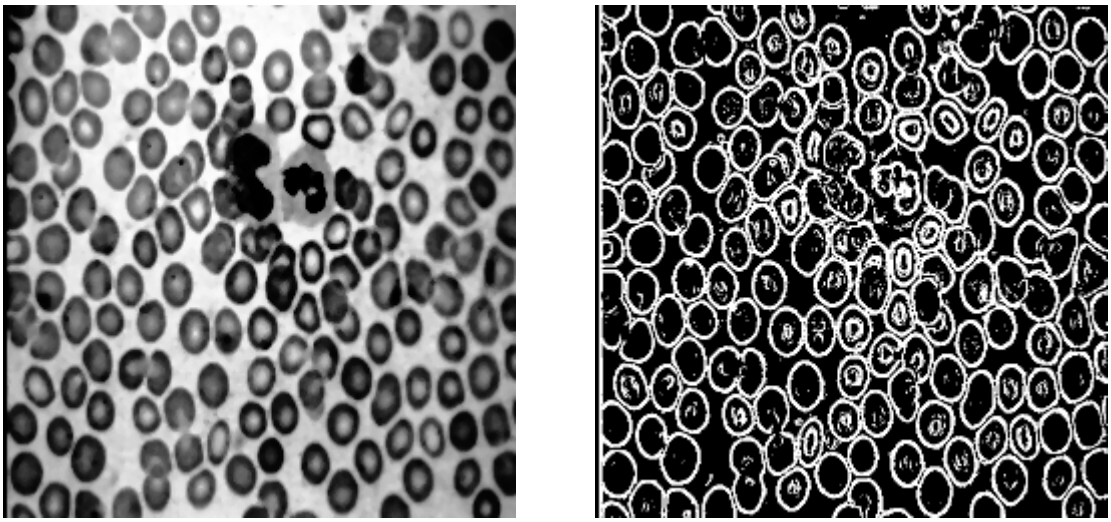

Abbildung E.2: Die Grauwertspreizung auf den verfügbaren Grauwertbereich ergibt in etwa konstante Bedingungen für die spätere Kantendetektion. Die Medianfilterung beseitigt kleine Störungen und vermindert somit die Fehlerquote. Die Gradientenwerte aus der Kantendetektion werden auf ein festes Intervall skaliert, so daß eine konstante Schwelle für die Binärisierung verwendet werden kann.

Die Spreizung der im Bild vorkommenden Grauwerte auf den gesamten zur Verfügung stehenden Grauwertbereich ermöglicht in etwa konstante Gradientenwerte an den Kanten der Blutkörperchen. Damit diese Kanten möglichst ohne Störungen, vor allem ohne Lücken berechnet werden, wird vorher eine Medianfilterung mit einem 3×3-Filter durchgeführt. Nachfolgend wird das Gradientenbild mit dem Prewitt-Operator berechnet. Der maximal vorkommende Betrags-Gradientenwert ist aufgrund der Maske des Prewitt-Operators und des Grauwertbereichs bekannt.

Um für die Binärisierung auch in etwa konstante Verhältnisse zu haben, wird das kumulative Histogramm der Betrags-Gradientenwerte aufgestellt und die unteren 95% der Werte auf ein festes Intervall [0, $GRAD_{MAX}$] skaliert. Alle größeren Werte werden auf $GRAD_{MAX}$ gesetzt. Als Schwelle für das Binärbild wird der relativ hohe Wert von 0.75 $GRAD_{MAX}$ verwendet. Dies soll bewirken, daß in diesem ersten Schritt auch wirklich nur echte rote Blutkörperchen gefunden werden.

## E.3 Lernphase

Das Binärbild ist Ausgangspunkt für den zweiten Schritt, die Lernphase. Mit einem Füllalgorithmus werden die einzelnen umschlossenen Flächen im Bild gefüllt und gleichzeitig dabei die relative Größe (in Bildpunkten) sowie die Extrempunkte berechnet. Über die Extrempunkte kann automatisch das umschließende Rechteck und somit die "Kompaktheit" als

Kompaktheit = Rechteck-Fläche / Füll-Fläche

bestimmt werden. Die Kompaktheit beträgt nach dieser vereinfachten Definition bei einem Kreis etwa 1.27.

Alle gefüllten Flächen,

- deren umschließendes Rechteck fast quadratisch ist und deren
- Kompaktheit zwischen 1.1 und 1.35 liegt,

sind potentielle rote Blutkörperchen. Die Mittelwerte für Größe, Ausdehnung sowie Kompaktheit können mit diesen Daten berechnet werden. Eventuelle Ausreißer, d.h. in einem oder mehreren Merkmalen stark vom Mittelwert abweichende Objekte, werden gelöscht und die jeweiligen Mittelwerte neu berechnet. Selbstverständlich muß dazu eine repräsentative Anzahl von roten Blutkörperchen vorhanden sein. Dies ist aber in den Vorbedingungen schon festgelegt. Wird trotzdem eine zu geringe Anzahl gefunden, so muß diese Lernphase abgebrochen und eventuell mit einem geringeren Schwellwert das Betrags-Gradientbild binärisiert werden. Sollte auch dies nicht zu einem eindeutigen Ergebnis führen, muß die weitere Untersuchung abgebrochen werden.

Die doppelte mittlere Größe der roten Blutkörperchen wird als Fenstergröße für eine adaptive Histogrammeinebnung des Median-gefilterten Bildes verwendet. Diese Histogrammeinebnung soll die unterschiedliche Beleuchtung im Bild ausgleichen.

## E.4 Identifizierung

Im Bild sind durch Vorverarbeitung und Lernphase gleichmäßige Bedingungen geschaffen worden. Die in diesen Schritten berechneten bildabhängigen Größen für die roten Blutkörperchen werden nun zur Identifizierung aller Objekte im Bild verwendet.

Die genaue Identifikation geschieht in einer adaptiven Vorgehensweise. Aus dem jetzt gleichmäßig ausgeleuchteten Bild wird über den Prewitt-Operator wieder das normierte Betrags-Gradientenbild berechnet und mit einer hohen Schwelle (0.75 $GRAD_{MAX}$) das Binärbild erzeugt. In diesem werden durch Füllalgorithmen die einzelnen komplett umschlossenen Gebiete gefüllt und gleichzeitig daraufhin untersucht, ob diese Gebiete die in der Lernphase berechneten Merkmale der roten Blutkörperchen besitzen.

Sind noch relativ viele Bildteile unerkannt geblieben, so wird mit einer geringeren Schwelle (z.B. 0.6 $GRAD_{MAX}$) eine Binärisierung des Betrags-Gradientenbilds vorgenommen und alle neu umschlossenen Gebiete werden auf die Merkmale hin untersucht. Diese Binärisierung mit der verminderten Schwelle wird solange durchgeführt, bis entweder keine neuen Objekte in einem Schritt erkannt werden oder eine Mindestschwelle (z.B. 0.2 $GRAD_{MAX}$ ) unterschritten wird.

Neben der einfachen Bewertung der Merkmale mit "vorhanden" – "nicht vorhanden" kann auch eine zahlenmäßige Gewichtung erfolgen. Diese drückt dann die Wichtigkeit eines Merkmals aus und zeigt, wie gut es ausgeprägt ist. Die Gesamtsumme (> *MINDESTSUMME*) der Bewertung der einzelnen Merkmale entscheidet dann über die Akzeptanz als rotes Blutkörperchen.

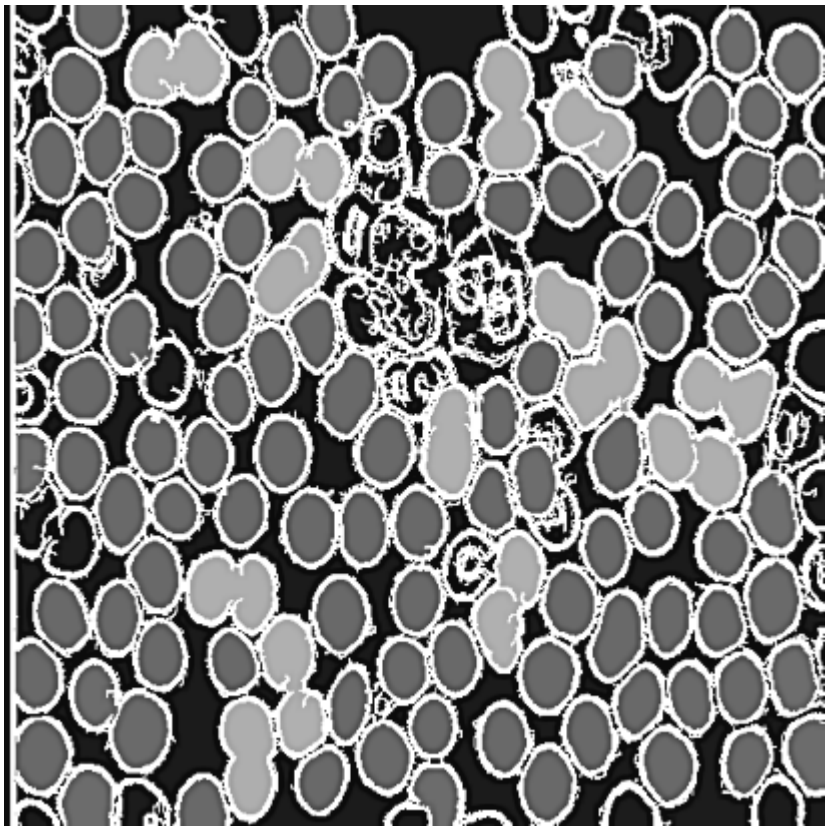

Abbildung E.3: Ergebnis am Ende der Identifizierung. Der Hintergrund ist schwarz. Dunkel gefüllte Objekte sind als einzelne rote Blutkörperchen, grau gefüllte als doppelte identifiziert worden. Alle verbleibenden Objekte sind Teile von Blutkörperchen (am Bildrand), mögliche weiße Blutkörperchen (Bildmitte) oder Störungen bzw. unbekannte Bestandteile.

Wegen der guten Bildqualität war diese Art der Bewertung in diesem hier beschriebenen System nicht notwendig. In Abbildung E.3 ist das Ergebnis dargestellt. Bei allen dunkel gefüllten Objekten handelt es sich um einwandfrei erkannte rote Blutkörperchen. Grau gefüllte Bestandteile sind zwei sich überlappende rote Blutkörperchen. Die Differenzierung erfolgt hier in der Hauptsache aufgrund der Abmessungen des umschließenden Rechtecks. Nicht gefüllte Teile sind entweder unerkannte rote Blutkörperchen (unerkannt wegen der Form oder weil sie am Rand nur teilweise im Bild liegen), Störungen oder weiße Blutkörperchen (Bildmitte).

Die gewünschten Angaben über die Absolutzahl der roten Blutkörperchen und ihre relativen Abmessungen können schon jetzt gemacht werden. Zur Detektion der weißen Blutkörperchen werden alle eindeutig identifizierten Bildbestandteile mit einer Umrandung von zwei Bildpunkten aus dem Binärbild gelöscht. Ebenfalls werden die Teile gelöscht, die direkt mit dem Bildrand verbunden sich. Hierbei handelt es sich in der Regel um Teile von Blutzellen. Für die Analyse sind aber nur vollständig sichtbare Bestandteile sinnvoll. Zurück bleibt ein Binärbild, das im wesentlichen die in den weißen Blutkörperchen aufgrund deren Texturierung detektierten Kanten und einige kleinere Störungen enthält. Nach einer Skelettierung werden die Störungen durch Erosion (Schwelle $T = 7$) beseitigt. Im Bereich der weißen Blutkörperchen bleiben aufgrund der feinen Verästelung der Kanten noch Bestandteile bestehen. Eine Mehrfachanwendung der Dilatation ergibt eine Maske (Abbil-

dung E.4 links), die die weißen Blutkörperchen komplett überdeckt. Aus dem Binärbild oder dem Originalbild wird jetzt der Teil mit den weißen Blutkörperchen extrahiert. Diese können gezählt und weiter analysiert werden. Im Bildbeispiel hat das Mustererkennungssystem zwei weiße Blutkörperchen detektiert.

Die Berechnung der Größe der weißen Blutzellen sowie die relative Häufigkeit der Bestandteile und die Markierung eventuell nicht interpretierbarer Teile schließt die automatische Analyse ab.

Abbildung E.4: Das linke Bild zeigt die Maske, die durch die Dilatation entstanden ist. Diese kann als Schablone zum Ausstanzen der relevanten Bereiche aus dem markierten Binärbild (Mitte, links) verwendet werden. Das Entfernen der bereits erkannten Bestandteile innerhalb dieser groben Schablone liefert eine Form (Mitte, rechts), die die gesuchten Bildteile relativ gut beschreibt (rechts).

## E.5 Schlußbemerkung

Obwohl die Gesamtbeschreibung dieses Systems relativ komplex wirkt, sind die verwendeten Verfahren einfach und können schnell abgearbeitet werden. Ein Gesamtdurchlauf ist auf einem 386er-PC-basierten System innerhalb 1-2 Minuten möglich.

Die Berührung einzelner Blutbestandteile ist durch das Füllen komplett umschlossener Flächen kein Problem. Überlappen sich mehr als zwei Blutkörperchen, so kann es eventuell zu falschen Ergebnissen kommen. Fehler können auch die am Bildrand liegenden Objekte, die nur teilweise sichtbar sind, verursachen. Abhilfe kann hier das Löschen aller Objekte am Rand schaffen, die nicht komplett umschlossen sind.

Können die Randbedingungen verbessert werden (z.B. gleichmäßige Ausleuchtung), so können einzelne Schritte übersprungen werden. Die Abarbeitungsgeschwindigkeit wird erhöht. Ist die Auflösung des Mikroskops bei der Aufnahme bekannt und somit auch die relative Größe der einzelnen Blutbestandteile, so kann die Lernphase verkürzt und die Identifizierung bezüglich der Objektgröße konkretisiert werden.

# F. Programme zum Buch

Die in diesem Buch "Bildverarbeitung in der Praxis" abgedruckten Listings sowie die ebenfalls kostenlos erhältlichen Programme dürfen nur für den privaten Gebrauch, für die Ausbildung oder für Forschungs- und Lehrzwecken im Zusammenhang mit diesem Buch verwendet werden.

Eine kommerzielle Nutzung der Programme, auch nur von Teilen, ist nur mit schriftlicher Genehmigung der Firma Dr. R. Steinbrecher möglich. Der unbefugte Gebrauch der Programme kann zu Schadensersatzansprüchen und strafrechtlicher Verfolgung führen.

Der Hersteller haftet weder für direkt noch für indirekt entstandene Schäden, die aus der Nutzung der Software entstanden sind. Dies gilt auch dann, wenn die Programme in einem fehlerhaften Zustand geliefert wurden.

Der Hersteller hat sich bemüht, die in dieser Anleitung und den Programmen enthaltenen Informationen korrekt anzugeben, jedoch behält er sich das Recht vor, gegebenenfalls Änderungen, Erweiterungen und Korrekturen vorzunehmen, ohne zu einer allgemeinen Bekanntgabe verpflichtet zu sein.

## F.1 Einführung

Punkte, die bei Büchern mit abgedruckten Programmlistings immer wieder stören, sind zum einen die Fehler in den Listings, zum anderen die mühsame Arbeit diese Programme abzutippen um sie entsprechend auszuprobieren.

Beim Schreiben der ersten Version des Buches "Bildverarbeitung in der Praxis" wurde parallel ein kleines Bildverarbeitungsprogramm (DBV) entwickelt, mit dem fast alle im Buch abgedruckten Bilder erstellt wurden. Gleichzeitig wurden die Original-Programme in das Buch übernommen, um so möglichst fehlerfreie Listings zu erhalten. Trotzdem kann keine vollkommene Fehlerfreiheit garantiert werden. Hinweise auf Fehler werden dankbar entgegengenommen. Die Programme erheben keinen Anspruch auf Vollständigkeit oder optimaler Programmierung. Vielmehr sollen die einzelnen Module dem Interessenten einen schnellen und leichten Einstieg in das Gebiet der Digitalen Bildverarbeitung ermöglichen.

Das Bildverarbeitungsprogramm ist komplett lauffähig. Aus technischen Gründen und auch aus Zeitmangel konnten (noch) nicht alle, im Buch angesprochenen Verfahren implementiert werden. So fehlen noch der Canny-Operator, verschiedene Skelettierungsalgorithmen, die Hadamard- sowie die Hough-Transformation. An verschiedenen Stellen in den Menüs und den einzelnen Programmen sind daher Hinweise und Freiräume für Erweiterungen vorgesehen. Sollten Sie solche Erweiterungen vornehmen, so würde ich mich über eine Kopie des neuen Moduls oder einen Hinweis freuen (E-Mail: DBV-Buch@RST-Software.de).

Ursprünglich wurden die Programme auf einem UNIX-System entwickelt und dann auf die 640 KByte-beschränkte-DOS-Welt portiert. Dabei mußten einige Kompromisse eingegangen werden. So wird teilweise die Festplatte als zusätzlicher Speicher verwendet. Das Ziel, einer möglichst großen Gruppe von Interessenten eine umfangreiche Sammlung von Bildverarbeitungsroutinen zur Verfügung zu stellen, konnte damit aber erreicht werden. Eine Portierung auf Windows (unter C/C++ und Java) ist schon seit längerer Zeit begonnen, aus Zeitmangel aber noch nicht beendet.

Zum Betrieb des Halbtonprogramms `HALBTON.EXE` und des Bildverarbeitungsprogramms `DBV.EXE` werden unter DOS mindestens 500 KByte freier Hauptspeicher und 1 MByte freier Festplattenplatz benötigt. Zur Installation muß nur der Inhalt der Diskette/CD/Verzeichnis auf die Festplatte kopiert bzw. die gepackten Dateien dort entpackt werden. Die Programmteile zum Halbtonprogramm und zum Bildverarbeitungsprogramm sollten in getrennten Verzeichnissen gespeichert werden.

Die Entwicklung unter DOS wurde mit dem TURBO-C-Compiler (Version 1.5 bzw. 2.0 von Borland) durchgeführt. Es wurden keine speziellen Eigenarten dieser Compilerversion verwendet, so daß andere C-Compiler unter DOS ohne große Änderungen eingesetzt werden können. Es müssen lediglich die hardwarenahen Teile in `CONST.H` und in `SHOW_PIC.C` (Graphik-Ausgabe) angepaßt werden.

## F.2 Der Disketten-/CD-Inhalt

Die erhältliche Diskette, CD bzw. die gepackten Dateien aus dem Internet enthalten folgende Dateien/Verzeichnisse:

```
README             Hinweise zum aktuellen Stand und Disketteninhalt
ACROBAT   <DIR>    Acrobat Reader 5.0
DBV-BUCH  <DIR>    Unterverzeichnis mit PDF-Dateien des Buches
BILDER    <DIR>    Unterverzeichnis mit den PCX-Bildern
HALBTON   <DIR>    Unterverzeichnis mit allen Teilen zum
                   HALBTON-Programm (Source und EXE)
DBV       <DIR>    Unterverzeichnis mit allen Teilen zum
                   DBV-Programm (Source und EXE)
```

Bitte lesen Sie die Datei `README`, da dort aktuelle Informationen und Hinweise abgelegt sind. Das Unterverzeichnis `BILDER` enthält PCX-Bilder im PCX-Format 5 und der Auflösung von 256x256 Bildpunkten mit bis zu 256 Graustufen. Im einzelnen sind dies:

```
BLUT     PCX    66058  Blutbild mit roten und weißen Blutkörperchen.
GRAUKEIL PCX    82817  Ein Graukeil mit 256 Graustufen.
                       Gutes Testobjekt für Halbtonverfahren.
GRUPPE4  PCX    65349  Eine Gruppe mit Polyedern. Sehr gut für die
                       Kantendetektion und Segmentierung.
KOPF     PCX    31835  Eine MRT-Aufnahme des Kopfes (vertikal).
```

```
KOPF99    PCX    65500  Eine MRT-Aufnahme des Kopfes (horizontal). An
                        beiden Bildern können die Effekte der
                        Histogramm-Operationen studiert werden.
MARKT     PCX    75151  Eine leicht vertikal gestreckte Aufnahme des
                        Marktplatzes in Tübingen.
NEW_YORK  PCX    60396  Die Aufnahme eines Satelliten von dem Gebiet
                        um New York (Long Island).
OSTRAKOD  PCX    46300  Die Kalkschale eines Muschelkrebses (Ostrakode).
STIFT     PCX    81238  Wilhelmstift in Tübingen. Dieses Bild enthält
                        viele, für die Bildverarbeitung interessante
                        Strukturen.
TEXTUR1   PCX    72513  Eine Textur mit Kieselsteinen und ...
TEXTUR2   PCX    66240  .. eine weitere mit einem Farnblatt.
ZIMMER    PCX    63405  Die Aufnahme eines Arbeitszimmers mit ebenfalls,
                        viele, für die Bildverarbeitung interessante
                        Strukturen.
```

Als Testbilder sind prinzipiell alle Bilder im PCX-Format 5 verwendbar. Durch eine entsprechende Änderung der Einleseroutine `READ_PCX.C` können auch andere PCX-Formate verarbeitet werden. Die Auflösung ist beliebig. Größere Vorlagen werden beim Einlesen durch Überspringen einzelner Zeilen/Spalten automatisch verkleinert, kleinere Bilder bleiben so erhalten. Einige Routinen sind auf quadratische Bilder ausgelegt bzw. auf Bilder mit einer Größe einer 2er-Potenz. Dies ist dann entsprechend zu beachten. Einfache Testbilder können auch direkt im Bildverarbeitungsprogramm `DBV` erzeugt werden.

Die Ergebnisbilder werden wieder als PCX-Bilder im PCX-Format 5 abgespeichert und können daher problemlos in Textverarbeitungssysteme o.ä. integriert werden.

In den beiden anderen Unterverzeichnissen befindet sich neben den einzelnen C-Files jeweils

- ein `MAKEFILE`,
- eine Datei mit den benötigten C-Bibliotheken `LIBS`,
- eine Datei mit den zusammenzulinkenden Dateien `FILES` (nur bei `DBV`),
- ein Header-File mit den Definitionen und Deklarationen der Konstanten und globalen Variablen `CONST.H`,
- eine Datei mit den Prototypen der einzelnen Prozeduren und Funktionen `PROTOTYP.H`,
- je eine Datei für die Bildausgabe im Graphik-Modus 12h und 2Eh `SHOW_PIC.12H`, `SHOW_PIC.2EH` und
- ein ausführbares Programm (gepackt; selbstentpackend) im Graphik-Modus 12h, das keinen Koprozessor benötigt.

Für andere Konfigurationen bzw. Graphikmodi müssen die Dateien umkopiert und der Kode neu kompiliert werden.

## F.3 Compiler-/Linker-Optionen

Zur Vereinfachung der Compilierung wurde das MAKE-Programm verwendet. In einer Datei (default: MAKEFILE) werden die entsprechenden Angaben für den Compiler und Linker gemacht. Der Aufbau einer solchen Datei ist in den entsprechenden Handbüchern zu dem verwendeten Compiler nachzulesen. Hier seien nur kurz die wichtigsten Punkte erwähnt:

```
# MAKEFILE fuer TURBO-C
#
# Version vom 19.08.1992
#
PATH=\LANGUAGE\C
#
# EXE-File erzeugen
#
HALBTON.EXE: HALBTON.OBJ OWN_LIB.OBJ SHOW_PIC.OBJ ... ORDERED.OBJ
   TLINK $(PATH)\LIB\C0L HALBTON READ_PCX ... ,HALBTON, ,@LIBS

#
# Alle Abhaengigkeiten definieren
#
HALBTON.OBJ: HALBTON.C CONST.H PROTOTYP.H
   $(PATH)\TCC -1 -I$(PATH)\INCLUDE -N -Z -f87 -ml -K -O -d -c HALBTON.C
...

HP.OBJ: HP.C CONST.H PROTOTYP.H
   $(PATH)\TCC -1 -I$(PATH)\INCLUDE -N -Z -f87 -ml -K -O -d -c HP.C
```

Zeilen mit einem #-Zeichen am Anfang, bedeuten Kommentare. Mit PATH wird eine Umgebungsvariable definiert. Hier ist entsprechend der Pfad für Ihren C-Compiler anzugeben. Das Programm HALBTON.EXE setzt sich aus den angegebenen einzelnen OBJ-Files zusammen. Beim Programm DBV.EXE sind die OBJ-Files in der Datei FILES aufgelistet.

Alle einzelnen C-Programme werden im Large-Modus compiliert, daher ist für den Linker TLINK zuerst die Datei C0L.OBJ anzugeben, danach die anderen Objekt-Files, immer ohne die Dateierweiterung OBJ. Gegen Ende der Zeile mit den Linker-Angaben wird der Name des ausführbaren Programms HALBTON.EXE und danach die Steuerdatei mit den benötigten Bibliotheken @LIBS angegeben. Der Inhalt der Datei LIBS ist

```
\LANGUAGE\C\LIB\EMU  \LANGUAGE\C\LIB\MATHL \LANGUAGE\C\LIB\CL ...
```

Die Namen der einzelnen Bibliotheken werden hier komplett mit Pfad angegeben. Die Pfad-Angabe ist wieder durch den entsprechenden Pfad bei Ihrem C-Compiler zu ersetzen. `EMU` ist der Emulator für den Koprozessor. Ist ein Koprozessor vorhanden, so kann die `FP87`-Bibliothek verwendet werden. Die mathematischen Operationen laufen dann schneller ab und das ausführbare Programm wird etwas kleiner, da nicht mehr die Emulation benötigt wird.

Die Optionen für den C-Compiler sind jeweils in der Zeile mit `TCC` angegeben. Dabei bedeuten

| | |
|---|---|
| -1 | es soll 80286-Kode erzeugt werden |
| -I | der Pfad für die Include-Dateien |
| -N | der Stack-Überlauf soll geprüft werden |
| -Z | die Prozessor-Register sollen zur Optimierung verwendet werden |
| -f87 | es werden 8087-Floating-Point-Routinen benötigt |
| -ml | als Speichermodell soll das Large-Modell verwendet werden |
| -K | char-Variablen sind default unsigned |
| -O | es soll eine Sprung-Optimierung vorgenommen werden |
| -d | doppelte Strings sollen eliminiert werden |
| -w | alle möglichen Warnungen sollen ausgegeben werden |
| -c | es soll nur ein OBJ-File erzeugt werden. |

Entsprechende Optionen sind bei anderen Compilern ebenfalls anzugeben.

Beim Aufruf des MAKE-Programms wird automatisch nach einer Datei `MAKEFILE` gesucht und die dort aufgeführten Befehle ausgeführt. Mit MAKE sind noch weitere Steuerungen und Vereinfachungen möglich.

## F.4 Die Graphik-Modi 12h und 2Eh

Im Anhang B wurde die VGA-Karte und ihre unterschiedlichen Modi bereits angesprochen. Der 256-Farben-Modus (2Eh) und die zugehörige Ansteuerung der ET-4000-Karten wurden mit Beispielen ausführlich erläutert.

Im 16-Farben-Modus (12h) werden jeweils 4-Bit (ein Nibbel) für einen Bildpunkt verwendet. Da der Farb-Index 7 von vielen Programmen (u.a. Turbo-C) anscheinend als Standardfarbe für Textausgaben verwendet wird, wird dieser LUT-Eintrag mit weiß belegt. Farb-Index 0 bleibt in diesem Modus auf schwarz, Index 1 wird mit rot belegt und nur für die Farbausgabe verwendet.

Die Möglichkeit, auch Farbe in den Bildern zu verwenden, ist zur Hervorhebung einzelner Strukturen, für besseren Kontrast und Falschfarben gedacht. Momentan wird dies noch nicht im Programm verwendet. Es könnten damit z.B. die einzelnen Cluster im Hough-Raum bei der Hough-Transformation farblich gekennzeichnet werden.

Das Einschalten des Graphik- bzw. Text-Modus erfolgt bei 12h analog zum Modus 2Eh. Die Farbtabelle (LUT) enthält nur maximal 16 Einträge, die entsprechend initialisiert werden. Es wird bei dieser Farbtabelle nicht die volle Helligkeit der einzelnen Stufen bzw. von weiß verwendet, da die Stufe 1 (=fast schwarz) nicht verfügbar ist. Insgesamt sind also die im Modus 12h dargestellten Bilder etwas dunkler als im Modus 2Eh.

Weitere Hinweise sind im ausführlich kommentierten Listing enthalten. Umfangreiche Informationen finden Sie auch in der aufgeführten Literatur.

```
void Graph_Mode(void)
{
  pc_cpu.h.ah = 0;                /* Neuer Video-Modus                    */
                                  /* Graphik-0x12 = 640x480 bei 16 Farben */
  pc_cpu.h.al = 0x12;             /* ueber Funktionsaufruf 00 setzen ...  */
  int86(0x10, &pc_cpu, &pc_cpu);  /* .. und durch Interrupt 10 aktivieren.*/
  vgaram = MK_FP(0xA000, 0);      /* Erzeugt Far-Pointer auf VGA-Segment  */
}

void Text_Mode(void)
{
  pc_cpu.h.ah = 0;                /* Text-Modus ueber den Funktionsaufruf */
  pc_cpu.h.al = TEXT_MODE;        /* 00 setzen und ...                    */
  int86(0x10, &pc_cpu, &pc_cpu);  /* ... durch Interrupt 10 aktivieren.   */
}

void Set_LUT(void)
{
  int i;
  rgb RGB_Tabelle[MAX_COLOR];     /* Farbtabelle mit MAX_COLOR Eintraegen */
  unsigned char AC_Index[17];     /* Zuweisung DAC-Tabelle zu AC-Tabelle  */

  RGB_Tabelle[0].Rot   =   0;     /* Ueber Index [0] wird Scharz erzeugt  */
  RGB_Tabelle[0].Gruen =   0;
  RGB_Tabelle[0].Blau  =   0;
  RGB_Tabelle[1].Rot   =  63;     /* Ueber Index [1] wird Rot angesprochen*/
  RGB_Tabelle[1].Gruen =   0;
  RGB_Tabelle[1].Blau  =   0;
```

```
  /* Farbtabelle von Eintrag 2 bis 15 als abgedunkelten Graukeil aufbauen  */
  for (i=2; i<16; i++) {
      RGB_Tabelle[i].Rot   = (unsigned char)(i*4-1);
      RGB_Tabelle[i].Gruen = (unsigned char)(i*4-1);
      RGB_Tabelle[i].Blau  = (unsigned char)(i*4-1);
  }
  /* Vertauschen von Eintrag Nr. 7 (==Grau) und Nr. 15 (==Weiss)           */
  RGB_Tabelle[15].Rot   =  RGB_Tabelle[7].Rot;
  RGB_Tabelle[15].Gruen =  RGB_Tabelle[7].Gruen;
  RGB_Tabelle[15].Blau  =  RGB_Tabelle[7].Blau;

  /* Es wird nicht die volle Intensitaet verwendet um das Fehlen der Stufe */
  /* 1 (==Fast Schwarz) zu verbergen. Intensitaet 59 == Weiss              */
  RGB_Tabelle[7].Rot   =  (unsigned char)59;
  RGB_Tabelle[7].Gruen =  (unsigned char)59;
  RGB_Tabelle[7].Blau  =  (unsigned char)59;

  pc_cpu.h.ah=0x10;               /* Aktiviere Modus zum Setzen mehrerer..*/
  pc_cpu.h.al=0x12;               /* ... Farb-Palettenregister (LUT)      */
  pc_cpu.x.bx=0;                  /* Beginne bei Register Nummer 0 und .. */
  pc_cpu.x.cx=16;                 /* ... aendere die 16 Register          */
  s_cpu.es   =FP_SEG(RGB_Tabelle); /* Segmentadresse der LUT-Tabelle       */
  pc_cpu.x.dx=FP_OFF(RGB_Tabelle); /* Offsetadresse in diesem Segment und..*/

  int86x(0x10, &pc_cpu, &pc_cpu, &s_cpu); /*..ueber Interrupt 10 aktivieren*/

  pc_cpu.h.ah =0x10;              /* DAC-Bereich zuordnen (Page-Mode)     */
  pc_cpu.h.al =0x13;
  pc_cpu.h.bl =0x00;              /* Page-Mode festlegen und ...          */
  pc_cpu.h.bh =0x01;              /* 16 Bloecke mit je 16 DAC-Registern ..*/
  int86(0x10, &pc_cpu, &pc_cpu);  /* ... durch Interrupt 10 aktivieren.   */

  pc_cpu.h.ah =0x10;
  pc_cpu.h.al =0x13;
  pc_cpu.h.bl =0x01;              /* Im Page-Mode den DAC-Registerblock ..*/
  pc_cpu.h.bh =0x00;              /* ... Nummer 0 anwaehlen und ...       */
  int86(0x10, &pc_cpu, &pc_cpu);  /* ... durch Interrupt 10 aktivieren.   */

  for (i=0; i<16; i++) AC_Index[i]=i;  /* Zuordnung AC-Index zu DAC-Index  */
  AC_Index[16]=6;                 /* Letzter Eintrag ist das Overscan-Reg.*/

  pc_cpu.h.ah=0x10;               /* Aktiviere Modus zum Setzen aller ... */
  pc_cpu.h.al=0x02;               /* ... AC-Palettenregister (LUT)        */
  s_cpu.es   =FP_SEG(AC_Index   ); /* Segmentadresse der AC-Index-Tabelle  */
  pc_cpu.x.dx=FP_OFF(AC_Index   ); /* Offsetadresse in diesem Segment und..*/
  int86x(0x10, &pc_cpu, &pc_cpu, &s_cpu); /*..ueber Interrupt 10 aktivieren*/
}
```

```
void PlotPixel(long Spalte, long Zeile, unsigned char Farbe)
{
  unsigned int Index;             /* Index bei der linearen Adressierung  */

  Index = (unsigned int) (Zeile * 80) + (Spalte>>3);        /* Pixel-Index */

  /* Abbilden der 256 Graustufen auf 15(16) Stueck. Index 1 ist mit Rot bel*/
  Farbe = (Farbe>31) ? (Farbe>>4):( (Farbe>=CLIP_COLOR) ? 0:Farbe );

  /* Index 7 ist auf Weiss gesetzt fuer die Text-Ausgabe. Index 15 enthaelt*/
  /* den Grauwert von Index 7. Daher Vertauschung notwendig.               */
  Farbe = (Farbe==7) ? 15:( (Farbe==15 ? 7:Farbe) );

  outport(0x3CE, 0);              /* Set/Reset-Register im Graphic ...    */
  outport(0x3CF, Farbe & 0x0F);   /* ... Controller mit Farbe laden.      */
  outport(0x3CE, 1);              /* Enable Set/Reset-Register und ...    */
  outport(0x3CF, 0xF);            /* ... alle vier Ebenen freigeben.      */

  outport(0x3CE, 8);              /* Bit-Mask-Register mit den zu ...     */
  outport(0x3CF, 128>>(Spalte & 0x07));  /* ... setzenden Bits laden.     */

  vgaram[Index]=vgaram[Index];    /* Dummy-Zugriff zum Pixel setzen.      */

  outport(0x3CF, 0xFF);           /* Bit-Mask-Register wieder freigeben   */
  outport(0x3CE, 1);              /* Enable Set/Reset-Register und ...    */
  outport(0x3CF, 0);              /* ... Ansprechen der Ebenen sperren.   */
}
```

Das normalerweise zweidimensionale Bild wird aufgrund der 64 KByte-Segmentierung im Programm als Vektor verarbeitet. Die vier verwendeten Bilder sind nacheinander im Vektor `Pictures` abgelegt. Um einen bestimmten Bildpunkt anzusprechen, muß daher die Bildnummer, die Zeile, die Spalte sowie die Spaltenanzahl bekannt sein. Der Zugriff ist in der Graphik auf der nächsten Seite dargestellt. Ebenso die Numerierung der Bilder und die zugehörige Position auf dem Bildschirm.

Selbstverständlich kann die Adressierung über ein entsprechendes Makro vereinfacht bzw. besser lesbar gestaltet werden (vergleiche `AHE.C`, der Zugriff auf `Tr_Matrix`). Dies könnte statt

```
Pixel=Pictures[(long)BildNr*PIC_SIZE + z*Bild[BildNr].Spalten + s];
```

dann mit dem Makro

```
#define BildPunkt(BildNr, z, s) \
        Pictures[(long)BildNr*PIC_SIZE+z*Bild[BildNr].Spalten+s]
```

als

```
Pixel=BildPunkt(BildNr, z, s);
```

geschrieben werden. Dies entspricht dann fast dem normalen Zugriff, wenn die Bilder als zweidimensionale Felder definiert werden könnten. Leider werden bei jedem Zugriff die Multiplikationen für den Offset durchgeführt, so daß verschiedene Optimierungen nicht mehr möglich sind.

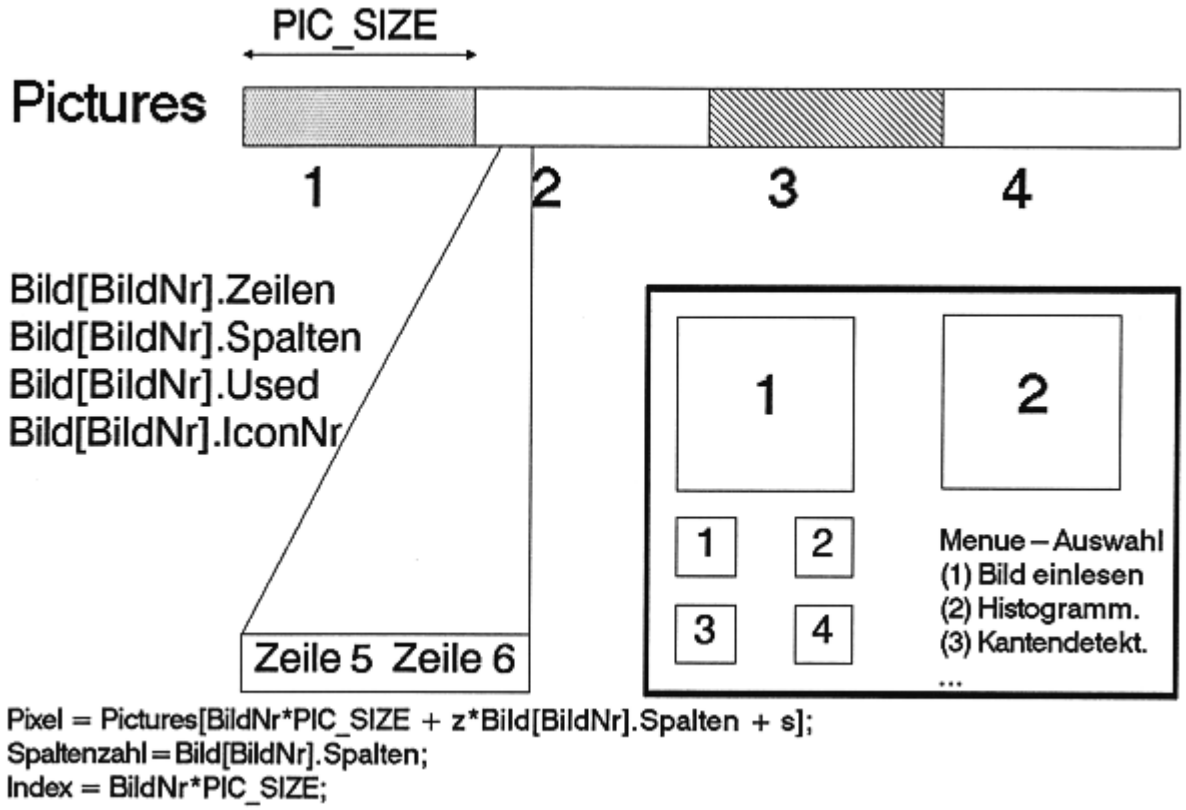


Abbildung F.1: Aufbau des Vektors `Pictures` und der Zusammenhang mit der Darstellung auf dem Bildschirm.

## F.5 Digitale Halbtonverfahren

Alle wichtigen digitalen Halbtonverfahren sind in dem Programm `HALBTON.EXE` zusammengefaßt. Eine Erweiterung um neue Verfahren ist problemlos möglich. Teilweise sind andere Verfahren schon als Kommentar in den Programmen enthalten. Alle verfügbaren Halbtonverfahren und ihre Position im Menübaum sind in der Graphik auf der nächsten Seite aufgeführt.

Nach dem Aufruf von `HALBTON` werden zuerst alle, im aktuellen Verzeichnis verfügbaren PCX-Dateien angezeigt. Es können nur Bilder im PCX-Format 5 verarbeitet werden.

Wurde ein Bild erfolgreich eingelesen, so schaltet das Programm in den Graphik-Modus, zeigt das Bild links oben auf dem Bildschirm an und gibt das Hauptmenü aus. Die Menüauswahl geschieht über die entsprechende Zahl in Klammer. Eventuell erscheint danach ein neues Untermenü.

Wurde das gewünschte Verfahren ausgewählt, so erscheint nach einiger Zeit rechts oben auf dem Bildschirm das Halbtonbild. Es vermittelt schon einen Eindruck des Aussehens des ausgedruckten

Ergebnisses. Bei der Auswahl werden günstige/sinnvolle Verfahren schon als Default-Wert vorgegeben.

Zum Schluß wird gefragt, ob ein HP-File erzeugt werden soll. Es wird hierbei eine Graphik-Datei im HP-Laserjet-II-Format aufgebaut. Diese kann auf den meisten Laserdruckern oder Tintenstrahldruckern ausgegeben werden. Es besteht die Möglichkeit das Halbtonbild mit 75, 150 oder 300 dpi auszudrucken. Weiterhin kann das Bild gedreht und/oder invertiert werden.

Die HP-Datei erhält automatisch die Erweiterung `.HP` und kann dann mit dem Befehl

```
COPY  <Dateiname>.HP  /B  PRN
```

gedruckt werden.

Aufgrund der Überlappung der Tonerpartikel beim Ausdruck ist es von der Qualität besser, wenn die Bilder größer, d.h. in einer geringeren Auflösung ausgedruckt und dann eventuell auf einem guten Photokopierer verkleinert werden. Die Auflösung von Photokopierern liegt über den üblichen 300 dpi bei Laserdruckern.

Die Größe eines Bildes kann einfach ausgerechnet werden. Ein Bild mit 256x256 Punkten und 300 dpi Auflösung wird mit ca. 2.17cm x 2.17cm gedruckt. Bei 150 dpi ist die Größe ca. 4.34cm x 4.34cm. Wurden Verfahren mit 5 Graustufen verwendet, so ist das jeweilige Bild in den Abmessungen doppelt so groß, bei 10 Graustufen dreimal so groß. Ein Bild der Größe 256x256 Punkten mit Error-Diffusion und 10 Graustufen bei einer Auflösung von 150 dpi ist demnach ca. 13cm x 13cm groß, und es paßt gerade noch auf ein DIN-A4-Blatt.

Sollen andere Drucker für die Ausgabe verwendet werden, so ist nur die Datei `HP.C` entsprechend umzuschreiben.

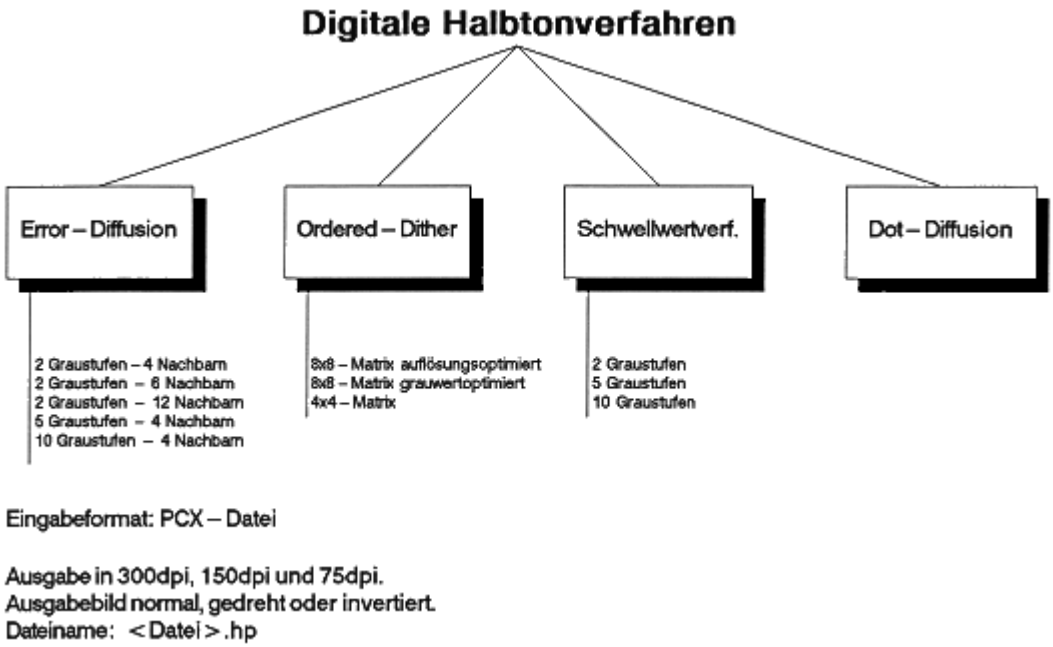


Abbildung F.2: Digitale Halbtonverfahren. Menüstruktur des Programms `HALBTON.EXE`.

# F.6 Digitale Bildverarbeitung (DBV)

Das Hauptprogramm der Digitalen Bildverarbeitung (`DBV.EXE`) ist ebenfalls komplett menügesteuert. Über die entsprechenden Zahlen werden die einzelnen Programmpunkte ausgewählt. Default-Werte, die mit ENTER/RETURN einfach übernommen werden können, stehen in eckigen Klammern.

Das Erscheinungsbild des Programms ist auf der nächsten Seite dargestellt, die Menüstruktur und wo sich die einzelnen Routinen befinden auf der übernächsten Seite in einer Übersichtsgraphik.

Der Menübaum kann bis zu vier Stufen tief sein. Dies wird in der Überschrift der einzelnen Menüs durch >-Zeichen angedeutet. Im Hauptmenü ist der Text in >>> ... <<< eingeschlossen, im ersten Untermenü in >> ... << usw. Ist der endgültige Programmpunkt erreicht, so steht die Überschrift zwischen zwei Sternen * ... *.

Eine besondere Bedeutung hat der Eingabewert -99 im Hauptmenü. Nach dieser Eingabe wird ein Hardcopy des kompletten Bildschirminhalts in eine PCX-Datei angefertigt. Dies ist aber nur im Modus 2Eh sinnvoll, da auf die einzelnen Punkte nach dem entsprechenden Schema zugegriffen wird. Ein Hardcopy im Modus 12h führt zu Fehlern und unsinnigen Ergebnissen. Die Routine `GetPixel()` müßte dazu entsprechend umgeändert werden.

Aufgrund des beschränkten Hauptspeichers wird die Festplatte als zusätzlicher Speicher verwendet. Es sollte daher mindestens 1 MByte freier Festplattenplatz vorhanden sein. In folgenden Routinen werden (temporäre) Binär-Dateien angelegt:

| | | |
|---|---|---|
| DCT.C | DCT.tmp | Zwischenwerte der Diskreten-Cosinus-Transformation |
| FFT.C | Imag1.tmp | Imaginärwerte der Fouriertransformation (Transformation der Zeilen) |
| | Real1.tmp | Realwerte der Fouriertransformation (Transformation der Zeilen) |
| | Imag2.tmp | Imaginärwerte der Fouriertransformation (Transformation der Spalten) |
| | Real2.tmp | Realwerte der Fouriertransformation (Transformation der Spalten) |
| MARR_HIL.C | MH_MATRIX.DAT | Hierbei handelt es sich nicht um eine Binär-Datei, sondern um eine Textdatei mit allen Werten der Faltungsmaske der Marr-Hildreth-Kantendetektion. |

Alle Dateien mit der Endung `.tmp` sind temporär und werden nach korrekter Beendigung der entsprechenden Routinen wieder gelöscht.

Im Anhang C im Buch wurde das Einlesen und Abspeichern von PCX-Bildern zum besseren Verständnis Byte-weise vorgenommen. Als Optimierung ist in dem beiliegenden Programmkode die Einleseroutine derart geändert worden, daß immer ein größerer Block eingelesen und dekodiert

wird. Damit wird der System-Verwaltungsaufwand reduziert und das Einlesen, besonders unter Multi-Tasking-Systemen (OS/2, UNIX, usw.), stark beschleunigt. Die Speicherroutine kann ebenfalls entsprechend modifiziert werden.

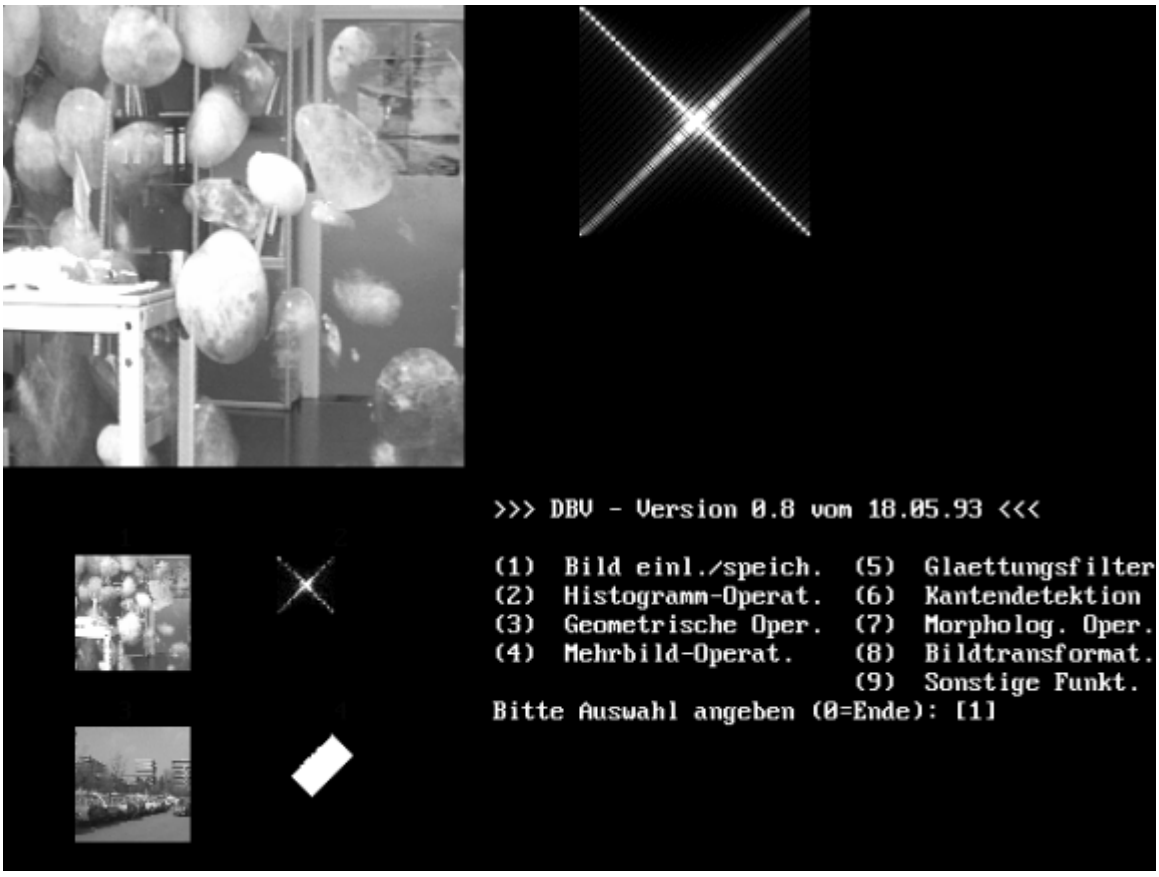


Abbildung F.3: Benutzungsoberfläche des DBV-Programms.

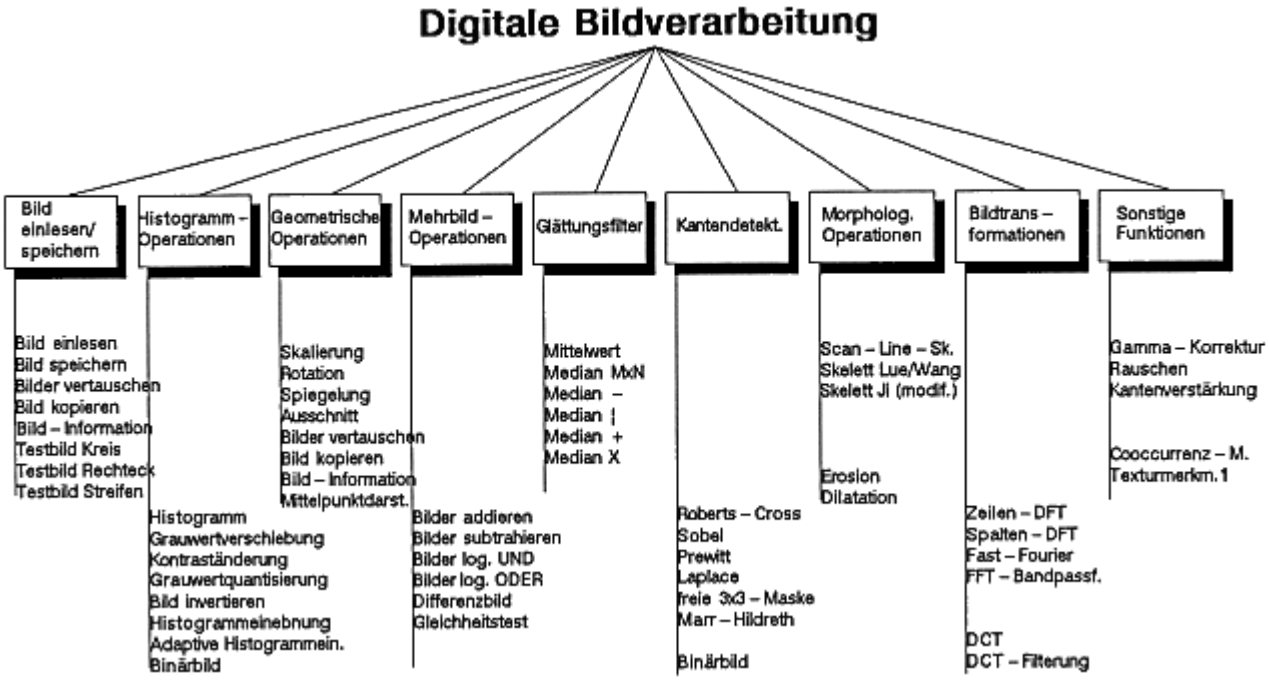


Abbildung F.4: Menüstruktur des DBV-Programms.

Auf einige Besonderheiten der C-Compiler unter DOS sei an dieser Stelle noch einmal hingewiesen.

1. **struct ffblk}**

   Mit Hilfe der Funktionen `FINDFIRST()` und `FINDNEXT()` werden Dateinamen mit ihren Attributen in die Struktur vom Typ `ffblk` geschrieben. Dies wird nur im Programm `READ_PCX.C` verwendet. Die Namen der gefundenen PCX-Files werden auf dem Bildschirm ausgegeben.

2. **farmalloc(), farfree()**

   Diese spezielle Art der Speicherallokierung gibt es nur unter DOS. Bei anderen Betriebssystemen ist das normale `malloc()` bzw. `free()` zu verwenden.

   Mit diesen far-Funktionen können größere Blöcke (größer als 64 KByte) reserviert werden. Diese werden nicht vom Stack, sondern vom Heap, der den gesamten freien Speicherplatz repräsentiert, benutzt. Diese Funktionen finden in den Routinen `AHE.C`, `DBV.C`, `INIT_VAR.C` und `THIN_JI.C` Verwendung.

# Literaturverzeichnis


[AH77] H.C. Andrews and B.R. Hunt. *Digital Image Restoration*.
Prentice-Hall Inc., 1977.

[Alt90] Martin Althaus. *Power-Grafik - mit EGA und VGA.*
*DOS*, (Nummer 6-10):170–173, 88–98, 138–150, 148–156, 178–189, 1990.

[AP79] Ikram E. Abdou and William K. Pratt. *Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors*.
*Proceedings of the IEEE*, 67(5):753–763, Mai 1979.

[Bö87] Klaus D. Bösing. *Über Aspekte der visuellen Wahrnehmung und der optischen Eigenschaft idealer Oberflächen*.
Technischer Bericht 87/12, Technische Universität Berlin; Institut für Technische Informatik; Computer Graphics & Computer Vision, 1987.

[Bau91] Frank Bauernpöppel. *Imploding ... freeezing ... done*.
*c't*, (10):278–286, Oktober 1991.

[BB82] Dana H. Ballard and Christopher M. Brown. *Computer Vision*.
Prentice-Hall Inc., Englewood Cliffs, NJ 07632, 1982.

[BB91] H. Bässman and Ph. W. Besslich. *Bildverarbeitung Ad Oculos*.
Springer, 1991.

[BD90] Sabine Behrens and Joachim Dengler. *Analysing the Structure of Medical Images with Morphological Size Distributions*.
In *10th International Conference on Pattern Recognition*, Seite 886–890. IEEE Computer Society Press, Juni 1990.

[Ber84] Valdis Berzins. *Accuracy of Laplacian Edge Detector*.
*Computer Vision, Graphics, and Image Processing*, 27:195–210, 1984.

[Bil87] Hans Peter Biland. *The Recognition and Volumetric Description of Three-Dimensional Polyhedral Scenes by Analysis of Hough-Space Structures*.
Dissertation, Swiss Federal Institute of Technology Zurich, ETH Zürich, 1987.

[Bli89a] James F. Blinn. *Dirty Pixels*.
*IEEE Computer Graphics and Applications*, Seite 100–105, Juli 1989.

[Bli89b] James F. Blinn. *Return of the Jaggy*.
*IEEE Computer Graphics and Applications*, Seite 82–89, März 1989.

[Bli89c] James F. Blinn. *What We Need Around Here Is More Aliasing*.
*IEEE Computer Graphics and Applications*, Seite 75–79, Januar 1989.

[Bri89] E. Oran Brigham. *FFT Schnelle Fourier-Transformation*.
R. Oldenbourg, 4. Auflage, 1989.

[BW84] Hans Peter Biland and Friedrich M. Wahl. *Understanding Hough Space for Poylhedral Scene Decomposition*.
Technischer Bericht, IBM Zurich Research Laboratory, 8803 Rüschlikon, März 1986.

[Can83] John Francis Canny. *Finding Edges And Lines In Images*.
Technical Report 720, MIT Artificial Intelligence Laboratory, May 1983.

[Can86] John Francis Canny. *A Computational Approach to Edge Detection*.
*IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[CF*84] U. Cugini, G. Ferri, P. Mussio, and M. Protti. *Pattern-Directed Restoration and Vectorization of Digitized Engineering Drawings*.
*Computers & Graphics*, 8(4):337–350, 1984.

[CH89] Yung-Sheng Chen and Wen-Hsing Hsu. *A Systematic Approach for Designing 2-Subcycle and Pseudo 1-Subcycle parallel Thinning Algorithms*.
*Pattern Recognition*, 22(3):267–282, 1989.

[CM87] Fergus W. Campbell and Lamberto Maffei. *Kontrast und Raumfrequenz*.
In Manfred Ritter, *Wahrnehmung und visuelles System*, Seite 132–139. Spektrum der Wissenschaft, 1987.

[CM92] Christoph Cavigioli and Gerhard Moosburger. *Weniger ist oft mehr*.
*Elektronik*, (24):32–43, 1992.

[Coo86] R. L. Cook. *Stochastic Sampling in Computer Graphics*.
*ACM Transactions in Graphics*, 5(1):51–72, Januar 1986.

[DH72] Richard O. Duda and Peter E. Hart. *Use of the Hough Transformation To Detect Lines and Curves in Pictures*.
*Communications of the ACM, Graphics and Image Processing*, 15(1):11–15, Januar 1972.

[ES87] J. Encarnação and W. Straßer. *Computer Graphics*,
R. Oldenbourg, 1987.

[FD77] Herbert Freeman and Larry S. Davis. *A Corner-Finding Algorithm for Chain-Coded Curves*. *IEEE Transactions on Computers*, Seite 297–303, März 1977.

[FvD84] J.D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*.
Addison Wesley, 1984.

[FvD*90] J.D. Foley, A. van Dam, S.T. Feiner, and J.F. Hughes. *Computer Graphics - Principles and Practice*.
Addison Wesley, 1990.

[Gal91] Didier Le Gall. *MPEG: A Video Compression Standard for Multimedia Applications*. *Communications of the ACM*, 34(4):46–58, April 1991.

[Gen86] Michael A. Gennert. *Detecting half-edges and vertices in images*. In *Conference on Computer Vision and Pattern Recognition*, Seite 552–557. IEEE, 1986.

[Gru91] Thomas Grunert. *Untersuchung, Klassifizierung und Bewertung von Kantendetektoren zur Segmentierung, im besonderen Kontext von NMR-Bildern*. Diplomarbeit, Universität Tübingen, Auf der Morgenstelle 10, C9, Mai 1991. WSI/GRIS.

[GW81] Neal C. J.R. Gallagher and Gary L. Wise. *A Theoretical Analysis of the Properties of Median Filters*. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29(6):466–471, Dezember 1981.

[GW87] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing*. Addison-Wesley, 2. Auflage, 1987.

[Har84] Robert M. Haralick. *Digital Step Edges from Zero Crossing of Second Directional Derivatives*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):58–68, Januar 1984.

[HB*92] Axel Hildebrand, Christof Blum, Georg Rainer Hofmann, and Rüdiger Strack. *Verarbeitung und Visualisierung von multispektralen Satellitendaten*. *Informatik Forschung und Entwicklung*, 7:106–114, 1992.

[Hec82] Paul Heckbert. *Color Image Quantization for Frame Buffer Display*. *Computer Graphics*, 16(3):297–307, Juli 1982.

[Hed88] K.H. Hedengren. *Decomposition of Edge Operators*. In *9th International Conference on Pattern Recognition*, Seite 963–965. IEEE, November 1988.

[Hil83] Ellen C. Hildreth. *The Detection of Intensity Changes by Computers and Biological Vision Systems*. *Computer Vision, Graphics, and Image Processing*, 22:1–27, 1983.

[Hof91] Josef Hoffmann. *Redundanz raus*. *c't*, (6):126–132, Juni 1991.

[Hor86] Berthold Klaus Paul Horn. *Robot Vision*. McGraw-Hill Book Company, New York, 1986.

[Hou62] P.V.C. Hough. *Methods and Means for Recognizing Complex Patterns*. U.S. Patent 3069654, 1962.

[HYY79] Thomas S. Huang, Goerge J. Yang, and Tang Gregory Y.
*A Fast Two-Dimensional Median Filtering Algorithm*.
*IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-27(1):460–465, Februar 1979.

[Jä89] Bernd Jähne. *Digitale Bildverarbeitung*.
2. Auflage. Springer, 1991.

[Jer77] Abdul J. Jerri. *The Shannon Sampling Theorem - Its Various Extensions and Applications: A Tutorial Review*.
*Proceedings of the IEEE*, 65(11):1565–1596, November 1977.

[JF90] H. G. Jansen and G. Fingerlin. *Geomagnetische Prospektion an einem ungewöhnlichen Holzbau römischer Zeit in Hüfingen, Schwarzwald-Baar-Kreis*.
In *Archäologische Ausgrabungen in Baden-Württemberg 1990*, Seite 97–101, Theiss.

[Ji89] Yu Ji. *Ein neues Verfahren zur Verdünnung von Binärbildern*.
In *Mustererkennung 1989*, Seite 72–76. GI, Springer, Oktober 1989.

[JN82] Javier Jimenez and Jose L. Navalon. *Some Experiments in Image Vectorization*.
*IBM Journal of research and development*, 26(6):724–734, November 1982.

[JR76] J.F. Jarvis and C.S. Roberts. *A New Technique for Displaying Continous Tone Images on a Bilevel Display*.
*IEEE Transactions on Communications*, Seite 323–330, August 1976.

[KF90] Uwe Kloos and Jürgen Fechter. *Optical Character Recognition*.
Praktikums Mustererkennung, 1990. WSI/GRIS, Tübingen.

[KFJ85] James T. Kuehn, Jeffrey A. Fessler, and Siegel Howard Jay.
*Parallel Image Thinning and Vectorization on PASM*.
In *IEEE, computer vision and pattern recognition*, Seite 368–374. IEEE Computer Society Press/North-Holland, Juni 1985.

[KL92] David C. Kay and John R. Levine. *Graphics File Formats*.
Windcrest / McGraw-Hill, 1992.

[Knu87] Donald E. Knuth. *Digitale Halftones by Dot Diffusion*.
*ACM Transactions on Graphics*, 6(4):245–273, Oktober 1987.

[Kol91] Rupert Kolb. *Computergestützter Mustervergleich von digitalisierten Bildern am Beispiel von Muschelkrebsen (Ostrakoden)*.
Diplomarbeit, Universität Tübingen, Auf der Morgenstelle 10, C9, April 1991. WSI/GRIS.

[Kor88] Axel F. Korn. *Toward a Symbolic Representation of Intensity Changes in Images*.
*IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(5):610–625, September 1988.

[Kre77] Thomas Kreifelts. *Skelettierung und Linienverfolgung in Raster-Digitalisierten Linienstrukturen*.
In H.-H. Nagel, *Digitale Bildverarbeitung*, Seite 223–231. Gesellschaft für Informatik, Springer. GI/NTG 28.-30. März 1977, München.

[KS90] M. Kirby and L. Sirovich. *Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, Januar 1990.

[Kwo88] Paul C.K. Kwok. *Thinning Algorithm by Contour Generation*.
*Communications of the ACM*, 31(11):1314–1324, November 1988.

[Lac88] Vinciane Lacroix. *A Three-Module Strategy for Edge Detection*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(6):803–810, November 1988.

[LB86] Wolfram H. H. Lunscher and Michael P. Beddoes. *Optimal Edge Detector Design I: Parameter Selection and Noise Effects*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(2):164–177, März 1986.

[Len89] Reimar Lenz. *Digitale Kamera mit CCD-Flächensensor und programmierbarer Auflösung bis zu 2994x2320 Bildpunkten pro Farbkanal*.
In *Mustererkennung 1989*, Seite 52–59. GI, Springer, Oktober 1989.

[Lio91] Ming Liou. *Overview of the px64 kbits/s Video Coding Standard*.
*Communications of the ACM*, 34(4):59–63, April 1991.

[Liu77] Hsun K. Liu. *Two- and Three-Dimensional Boundary Detection*.
*Computer Graphics and Image Processing*, 6:123–134, 1977.

[LW85] H.E. Lü and P.S.P. Wang. *An Improved Fast Parallel Thinning Algorithm for Digital Patterns*. In *IEEE, computer vision and pattern recognition*, Seite 364–367. IEEE, Computer Society Press/North-Holland, Juni 1985.

[MBM81] Klaus D. Mörike, Eberhard Betz, and Walter Mergenthaler. *Biologie des Menschen*. Quelle & Meyer, 1981.

[MC*89] E. De Micheli, B. Caprile, P. Ottonello, and V. Torre. *Localization and Noise in Edge Detection*. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(10):1106–1117, Oktober 1989.

[MD86] Olivier Monga and Rachid Deriche. *A new three dimensional boundary detection*.
In *8. International Pattern Recognition Conference*, Seite 739–740. IEEE, Paris, 27.-31. Oktober 1996

[Mei91a] Peer Meier. *Modus-Kennung der Super-VGAs*.
*c't*, (6):249–252, Juni 1991.

[Mei91b] Peer Meier. *Schöne bunte Welt – Hardwarenahe Programmierung von VGA-Karten*.
*c't*, (5+6):338–346, 292–302, April, Juni 1991.

[MH80] D. Marr and E. Hildreth. *Theory of edge detection*.
In *Proceedings of Royal Society of London*, Seite 187–217, Great Britain, 1980. The Royal Society. Volume 207.

[NB80] Ramakant Nevatia and K. Ramesh Babu. *Linear Feature Extraction and Description*.
*Computer Vision, Graphics, and Image Processing*, 13:257–269, 1980.

[NB86] Vishvjit S. Nalwa and Thomas O. Binford. *On Detecting Edges*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):699–714, November 1986.

[Nel91] Mark R. Nelson. *LZW-Datenkompression*.
*PC+Technik*, (2):76–80, 1991.

[Ner81] Patrenahalli M. Nerendra. *A Separable Median Filter for Image Noise Smoothing*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(1):20–29, Januar 1981.

[Nie82] Klaus Niederdrenk. *Die endliche Fourier- und Walsh-Transformation mit einer Einführung in die Bildverarbeitung*.
Friedr. Vieweg & Sohn, 1982.

[NO90] Christian Neusius and Jan Olszewski. *Verdünnisierung*.
*c't*, (10):396–402, Oktober 1990.

[NS81] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*.
McGraw-Hill, 1981. 10. Auflage.

[O'G88] Lawrence O'Gorman. *A Note on Histogram Equalization for Optimal intensity Range Utilization*. *Computer Vision, Graphics and Image Processing*, 41:229–232, 1988.

[O'G90] Lawrence O'Gorman. *k x k Thinning*.
*Computer Vision, Graphics and Image Processing*, 51:195–215, 1990.

[PAA*87] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld.
*Adaptive Histogram Equalization and Its Variations*.
*Computer Vision, Graphics, and Image Processing*, Seite 355–368, 1987.

[Par88] J.R. Parker. *Extracting Vectors From Raster Images*.
*Computers & Graphics*, 12(1):75–79, 1988.

[Pav82] Theo Pavlidis. *Algorithms for Graphics and Image Processing*. Springer, 1982.

[Pet91] Wolfgang Petersen. *Videokompression konkret*. *PC+Technik*, (3):34–37, 1991.

[Pra78] William K. Pratt. *Digital Image Processing*. John Wiley & Sons, 1978.

[Rü89] Bodo Rüscamp. *Datenkonzentrat - Arbeitsweise der LZW-Datenkompression*. ix - UNIX Computer Magazine, (3):53–55, Mai 1989.

[Ris88] Thomas Risse. *Yet Another Line Parametrization for Hough Transform*. In H. Bunke, O. Kübler, and P. Stucki, *Proceedings of the 10th DAGM Symposium on pattern recognition*, September 1988, Informatik Fachberichte 180}, Seite 142–150. Springer.

[Ris89] Thomas Risse. *Hough Transform for Line Recognition: Complexity of Evidence Accumulation and Cluster Detection*. *Computer Vision, Graphics and Image Processing*, 46:327–345, 1989.

[RK76] Azriel Rosenfeld and Avinash C. Kak. *Digital Picture Processing*. Computer Science and Applied Mathematics. Academic Press, 1976.

[RS88] Thomas Risse und Rainer Steinbrecher. *Bewertung der Hough-Transformation als Hilfsmittel zur Objekt-Erkennung*. Technischer Bericht, WSI/GRIS, Auf der Morgenstelle 10, C9, Tübingen, März 1988.

[RW89] Cl. Romanova and U. Wagner. *A VLSI Architecture for Anti-Aliasing*. In *Proceedings of the Fourth Eurographics Workshop on Graphics Hardware*. Eurographics, Springer, 1989.

[SB89] G. E. Sotak and K. L. Boyer. *The Laplacian–of–Gaussian Kernel: A Formal Analysis and Design Procedure for Fast, Accurate Convolution and Full–Frame Output*. *Computer Vision, Graphics, and Image Processing*, 48:147–189, 1989.

[Sch89] Robert J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley & Sons, 1989.

[SG79] J. Sklansky and V. Gonzales. *Fast Polygonal Approximation of Digitized Curves*. In *Pattern Recognition and Image Processing*, Seite 604–609. IEEE Computer Society, 1979. 6.-8. August, Chicago, Illinois.

[SR71] R. Stefanelli and A. Rosenfeld. *Some Parallel Thinning Algorithms for Digital Pictures*. *Journal of the ACM*, 18(2):255–264, April 1971.

[SS87] Rod Salmon and Mel Slater. *Computer Graphics, System & Concepts*. Addison-Wesley, 1987.

[Ste89] Rainer Steinbrecher. *Vektorisierung von Rasterbildern*.
Technischer Bericht, WSI/GRIS, 7400 Tübingen, Januar 1989.

[Str87] Frank Streichert. *Informationsverschwendung - Nein danke*.
*c't*, (1):90–97, Januar 1987.

[Stu81] P. Stucki. *Image Processing for Documentation*.
Technical Report, IBM Zurich Research Laboratory, 8803 Rüschlikon, September 1981.

[TO92] Erwin M. Thurner and Monika Otter. *Komprimieren ohne Datenverlust*.
*Elektronik*, (23+25):134–137, 38–44, 1992.

[TP86] Vincent Torre and Tomaso A. Poggio. *On Edge Detection*.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(2):147–163, März 1986.

[VSM88] Albert M. Vossepoel, Berend C. Stoel, and A. Peter Meershoek.
*Adaptive Histogram Equalization Using Variable Regions*.
In *9th International Conference on Pattern Recognition*, Seite 351–353. IEEE, November 1988.

[Wah84] F.M. Wahl. *Digitale Bildsignalverarbeitung*.
Nachrichtentechnik 13. Springer, 1984.

[Wal91] Gregory K. Wallace. *The JPEG Still Picture Compression Standard*.
Communications of the ACM}, 34(4):31–44, April 1991.

[WB86] Friedrich M. Wahl and Hans-Peter Biland. *Decomposition of Polyhedral Scenes in Hough Space*.
In *International Conference on Pattern Recognition*, Seite 78–84. IEEE, Oktober 1986.

[WS90] Donna J. Williams and Mudarak Shah. *Normalized Edge Detector*.
In *10th. International Conference on Pattern Recognition*, Seite 942–946. IEEE, 6.-21. Juni 1990.

[Xia88] Yun Xia. *Minimizing the Computing Complexity of Iterative Sequential Thinning Algorithm*.
In *9th International Conference on Pattern Recognition*, Seite 721–723. IEEE, November 1988.

[ZS84] T.Y. Zhang and C.Y. Suen. *A Fast Parallel Algorithm for Thinning Digital Patterns*.
Communications of the ACM, 27(3):236–239, März 1984.

[ZW88] Y.Y. Zhang and P.S.P. Wang. *A Maximum Algorithm for Thinning Digital Patterns*.
In *9th International Conference on Pattern Recognition*, Seite 942–944. IEEE, November 1988.

# Stichwortverzeichnis