10. Bildverbesserung

Bildverbesserungsverfahren haben mit den schon behandelten Bildrestaurierungsverfahren das Ziel gemeinsam, Bildsignale so aufzubereiten, daß sie die relevanten Informationen besser darstellen, d.h. Bildinformationen, die bei einer speziellen Anwendung redundant oder sogar störend sind, zu unterdrücken und dafür wichtige Bildinhalte deutlicher hervorzuheben. Die Bildverbesserungsverfahren sind daher im Gegensatz zu den Bildrestaurierungsverfahren subjektive Verfahren und von der Anwendung abhängig.

Diese Verfahren kann man grob in solche unterteilen, die im Orts- oder Frequenzbereich arbeiten, in lokale und globale, in signalunabhängige und signalabhängige sowie in lineare und nichtlineare Methoden. Die wichtigsten Verfahren sollen im folgenden beispielhaft vorgestellt werden.

10.1 Grauwertmanipulation

Die mit Kameras, Scanner o.ä. gewonnenen Bildsignale nutzen oft, bedingt durch ungünstige Beleuchtungsverhältnisse (Unter- oder Überbelichtung), nicht den ganzen darstellbaren Grauwert-/Farbbereich aus. Solche Bilder wirken dann zu hell, zu dunkel oder zu kontrastarm. Dabei ist zu beachten, daß das menschliche Sehsystem eine logarithmische Kennlinie bezüglich der Helligkeitsempfindung aufweist (vergleiche Kapitel 3 und Abschnitt 5.51). In dunklen Bildbereichen ist daher die Grauwertunterscheidung geringer. Ohne entsprechende Korrektur wirken äquidistant quantisierte Bilder für den Betrachter zu kontrastarm. Als Lösungsmöglichkeit für diesen Fall wurde die Gamma-Korrektur bereits vorgestellt.

Zu helle bzw. zu dunkle Bilder können einfach durch konstante Verschiebung aller Grauwerte abgedunkelt oder aufgehellt werden.

$$g'(x, y) = g(x, y) + c_1$$

mit $g(x, y)$ = Original-Grauwert an der Stelle (x, y), $g'(x, y)$ = neuer Grauwert an der Stelle (x, y) und c_1 = Verschiebungskonstante.

Bei dieser Grauwertverschiebung muß darauf geachtet werden, daß der neue Grauwert nicht außerhalb des darstellbaren Bereiches liegt, da sonst Information verloren geht.

Diese Addition eines konstanten Wertes zu jedem Grauwert verändert den Kontrast nicht. Dies geschieht erst durch Multiplikation mit einem konstanten Faktor ($c_2 > 0$). Dadurch kann eine Spreizung des Ausgangsgrauwertbereichs auf den ganzen darstellbaren Bereich erfolgen. Der Faktor kann größer 1 (z.B. bei dunklen Bildern, Bilder wirken kontrastreicher) oder kleiner 1 (Bilder wirken kontrastärmer) sein, falls der darstellbare Bereich kleiner als der Grauwertbereich des Bildes ist. Eine Kombination dieses Verfahrens mit der Grauwertverschiebung ist ebenfalls möglich.

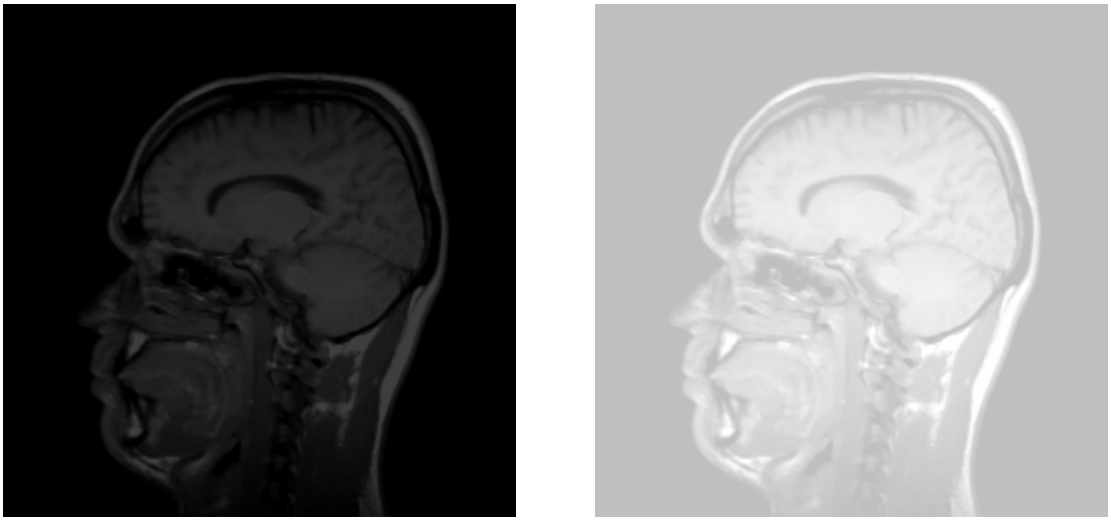


Abbildung 10.1: Alle Grauwerte aus dem linken Bild (Grauwertbereich 0, ..., 63) wurden um den konstanten Wert $c_1 = 192$ verschoben. Der Inhalt des Ergebnisbildes rechts ist zwar besser zu erkennen, jedoch wirkt das Bild noch kontrastarm.

$$g'(x, y) = c_2\, g(x, y) + c_1$$

Folgende Transformation eines Bildes nutzt den ganzen zur Verfügung stehenden Grauwert-/ Farbbereich:

$$g'(x, y) = (g(x, y) - g_{min}) \frac{G_{max} - G_{min}}{g_{max} - g_{min}} + G_{min}$$

mit G_{max}, G_{min} = maximaler und minimaler zur Verfügung stehender Grauwert, g_{max}, g_{min} = maximaler und minimaler im Bild vorkommender Grauwert.

Es handelt sich dabei immer noch um eine lineare, ortsunabhängige und globale Transformation. Denkbar ist auch eine nichtlineare Gewichtung (z.B. logarithmisch) oder eine stückweise lineare Gewichtung, um bestimmte Bereiche stärker zu beeinflussen. Oft werden dabei aber unterschiedliche Grauwerte g des Originalbildes auf einen Wert g' des darstellbaren Bildes transformiert, was einen Informationsverlust bedeutet (vergleiche Abschnitt 5.5.1).

Der Informationsverlust durch die nichtlineare bzw. stückweise lineare Grauwertskalierung kann jedoch auch erwünscht sein, um eine nicht benötigte Bildinformation "auszublenden" bzw. eine Kontrasterhöhung in relevanten Grauwertbereichen zu erhalten.

Einige der häufig verwendeten Funktionen zur nichtlinearen Grauwertskalierung sind im folgenden aufgelistet. Eine Über- oder Unterschreitung des darstellbaren Bereiches muß durch entsprechendes Klipping, d.h. durch Abbildung von Grauwerten, die den Bereich überschreiten, auf den größten Wert bzw. durch Abbildung auf den kleinsten Grauwert, falls der Bereich unterschritten wird, verhindert werden (siehe auch [Pra78]).

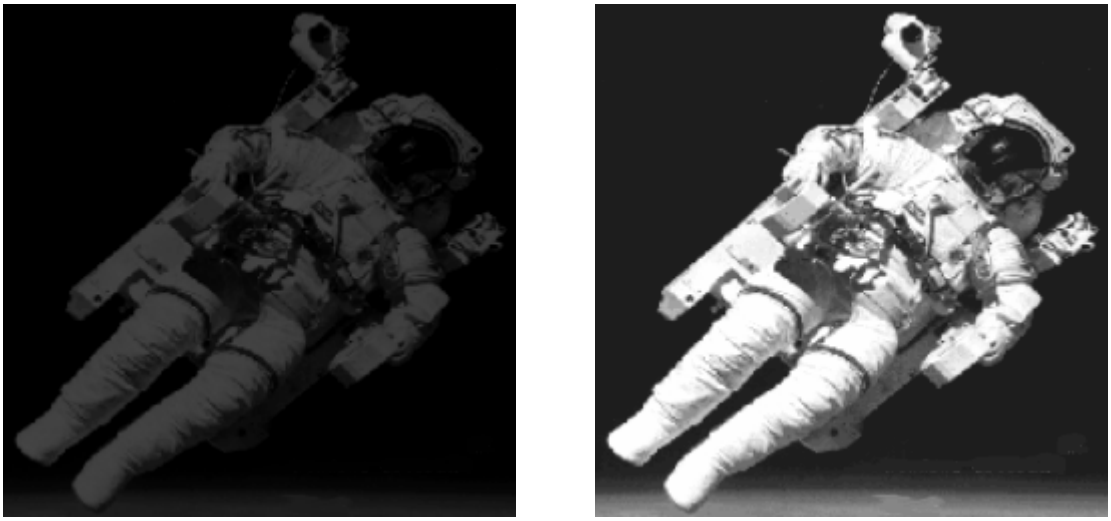


Abbildung 10.2: Alle Grauwerte aus dem linken Bild (Grauwertbereich 0, ..., 63) wurden mit dem Wert c_2 = 3.5 multipliziert und um c_1 = 30 verschoben. Der Inhalt des Ergebnisbildes rechts ist wesentlich besser zu erkennen und wirkt auch kontrastreicher im Vergleich zu der einfachen Grauwertverschiebung.

$$g'(x, y) = c_1 \, g^2(x, y) + c_2$$

$$g'(x, y) = c_1 \, \sqrt{g(x, y)} + c_2$$

$$g'(x, y) = \ln(c_1 \, g(x, y)) + c_2$$

$$g'(x, y) = e^{g(x, y) + c_1} + c_2$$

Bei allen diesen Transformationen muß immer darauf geachtet werden, daß der neue Grauwert noch im darstellbaren Bereich liegt.

Praktisch wird bei der Grauwerttransformation nicht jeder einzelne Bildpunkt nach der Vorschrift transformiert, sondern es wird eine Tabelle aufgebaut, die genauso viele Einträge enthält, wie Grauwerte im Originalbild vorkommen. Die unterschiedlichen Grauwerte der einzelnen Einträge überdecken dann den gewünschten Zielbereich. Mit Hilfe solcher Tabellen (*Look-Up-Table, LUT*) können beliebige Transformationen realisiert werden, da der Wert sich von Eintrag zu Eintrag beliebig ändern kann. Somit ist die gezielte Beeinflussung einzelner Grauwertbereiche möglich.

War bei den bisher vorgestellten Grauwerttransformationen die Transformationsfunktion schon vorgegeben, so ist es auch denkbar, diese aus zwei Bildern (dem Originalbild und dem Zielbild) zu berechnen. Sinn und Zweck einer solchen Grauwertänderung könnte zum Beispiel die möglichst gleiche Darstellung einer Szene sein, die bei unterschiedlichen Beleuchtungsverhältnissen aufgenommen wurde.

```
void Grauwertanpassung(int BildNr)
{
  int z, s;             /* Zeilen-/Spalten-Index                  */
  int Minimum, Maximum; /* Minimaler/Maximaler vorkomm. Grauwert */
  float Faktor;         /* Multiplikator C2 zur Grauwertanpassung*/

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Grauwertanpassung *");

  /* Minimalen und maximalen vorkommenden Grauwert ermitteln     */
  Minimum = WEISS;
  Maximum = SCHWARZ;
  for (z=0; z<Picture[BildNr].Zeilen; z++)
      for (s=0; s<Picture[BildNr].Spalten; s++) {
          if (Picture[BildNr].Bild[z][s] < Minimum)
             Minimum = Picture[BildNr].Bild[z][s];
          else if (Picture[BildNr].Bild[z][s] > Maximum)
             Maximum = Picture[BildNr].Bild[z][s];
      }
  Faktor = ((float)(WEISS - SCHWARZ)) / ((float)(Maximum-Minimum));

  /* Nun die eigentliche Anpassung nach der bek. Formel durchf.  */
  for (z=0; z<Picture[BildNr].Zeilen; z++)
      for (s=0; s<Picture[BildNr].Spalten; s++)
          Picture[BildNr].Bild[z][s] = (unsigned char)
             ((float)(Picture[BildNr].Bild[z][s]-Minimum)*Faktor);
}
```

10.2 Grauwertquantisierung

Eine *Grauwertquantisierung* stellt zwar keine Bildverbesserung dar, ist aber auch eine Grauwertmanipulation und soll der Vollständigkeit halber an dieser Stelle erwähnt werden.

Wie schon bei den Halbtonverfahren (siehe Kapitel 6) deutlich wurde, ist es manchmal notwendig, die Anzahl der zur Verfügung stehenden Graustufen zu reduzieren. Da das Auge sowieso nur eine geringe Graustufenanzahl (siehe Kapitel 3) unterscheiden kann, ist eine Reduktion auf 64 oder 32 Graustufen in der Regel nicht wahrnehmbar.

Der Vorteil dieser Bearbeitung kann Platzersparnis bei der Speicherung des Bildes, die Möglichkeit der Ausgabe auf speziellen Geräten oder die vereinfachte Berechnung von Filterungen u.ä. sein.

Bei der Grauwertquantisierung wird zur Transformation eine Look-Up-Tabelle berechnet, die je nach Anzahl der Quantisierungsstufen mehrere identische Einträge enthält. Wichtig bei solch einer Quantisierung ist, daß Schwarz wieder auf Schwarz und Weiß auf Weiß abgebildet wird. Zur vereinfachten Berechnung dieser Transformationsfunktion kann man dabei von einem fiktiven größeren Grauwertbereich ausgehen.

Werden nur zwei Stufen für die Quantisierung verwendet, so entsteht ein *Binärbild*. Dieser Vorgang wird *Binärisierung* genannt.

Abbildung 10.3: Eine Quantisierung auf 32 Stufen ist im normalen Bild (links) oft noch nicht zu erkennen. Nur im Graukeil sind die Abstufungen zu erahnen. Bei nur 8 Graustufen (rechts) ergeben sich jedoch fiktive Konturen.

```
void Quantisierung(int BildNr, int Stufen)
/* BildNr = das zu bearbeitende Bild                              */
/* Stufen = Anzahl der Quantisierungsstufen >= 2                  */
{
  int z, s;                      /* Zeilen-/Spalten-Index        */
  int Anfang, Ende;              /* Stufen-Werte                 */
  int i;
  int Steps, Anzahl;             /* Groesse der einzelnen Stufen */
  unsigned char MittlererGW;     /* Mittlerer Grauwert der Stufe */
  unsigned char LUT[MAX_COLOR];  /* Tabelle zur einfachen Quant. */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Grauwertquantisierung *");

  Anzahl = MAX_COLOR / (Stufen-1);
  Ende   = Anzahl/2;
  Steps  = 1;
  Anfang = 0;
```

```
    MittlererGW = SCHWARZ;  /* die erste Stufe muss schwarz werden */
    while (Steps < Stufen) {
        for (i=Anfang; i<Ende; i++)  LUT[i] = MittlererGW;
        Anfang       = Ende;       /* "Anfang" und ...              */
        Ende        += Anzahl;     /* "Ende" der aktuellen Stufe    */
        MittlererGW += Anzahl;     /* neuer mittlerer Grauwert      */
        Steps++;
    }
    /* die letzte Stufe jetzt noch auf weiss setzen                 */
    for (i=Anfang; i<MAX_COLOR; i++) LUT[i] = WEISS;

    for (z=0; z<Picture[BildNr].Zeilen; z++)
        for (s=0; s<Picture[BildNr].Spalten; s++)
            Picture[BildNr].Bild[z][s] =
                                LUT[ Picture[BildNr].Bild[z][s] ];
}
```

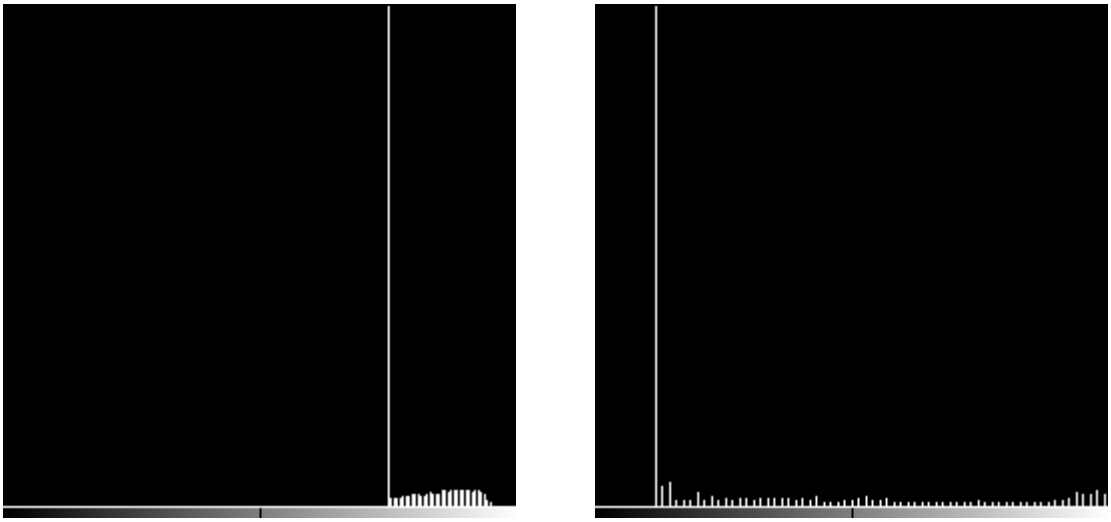


Abbildung 10.4: Die Darstellung der Häufigkeitsverteilung der Grauwerte in einem Bild erfolgt durch das Histogramm. Aufgrund der Form können die Faktoren zur Grauwertmanipulation oder Schwellen zur Segmentierung bestimmt werden. Das linke Bild ist das Histogramm vom Bild 10.1, rechts. Das rechte Bild stellt das Histogramm von Bild 10.2, rechts dar.

10.3 Histogrammdarstellung

Zur Bestimmung der Faktoren c_1 und c_2 ist in der Regel die Auswertung des *Histogramms* nützlich. Das Histogramm ist die Darstellung der Häufigkeitsverteilung der Grauwerte in einem Bild. Auf der horizontalen Achse werden dabei in der Regel die unterschiedlichen diskreten Grauwerte in Form von Zahlen oder einem Graukeil dargestellt. Darüber wird die Häufigkeit des entsprechenden Grauwerts als Balken abgebildet. Normalerweise sind weniger die absoluten Häufigkeitswerte als die relativen Werte von Interesse.

Aufgrund der Form des Histogramms können oft auch Entscheidungen über günstige Schwellwerte zur Segmentierung getroffen werden (vergleiche Kapitel 14).

Für Kontraständerung und Histogrammeinebnung ist das *kumulative Histogramm* von Bedeutung. Es stellt die Anzahl der Bildpunkte mit einem nicht größeren Grauwert als ein vorgegebener dar. Das kumulative Histogramm wird durch Summation der Häufigkeiten der einzelnen Grauwerte aus dem normalen Histogramm, beginnend mit der Häufigkeit von Schwarz, berechnet. Sind die Grauwerte über den gesamten Grauwertbereich in etwa gleich verteilt, so verläuft die Kurve des kumulativen Histogramms diagonal von links unten (schwarz) nach rechts oben (weiß).

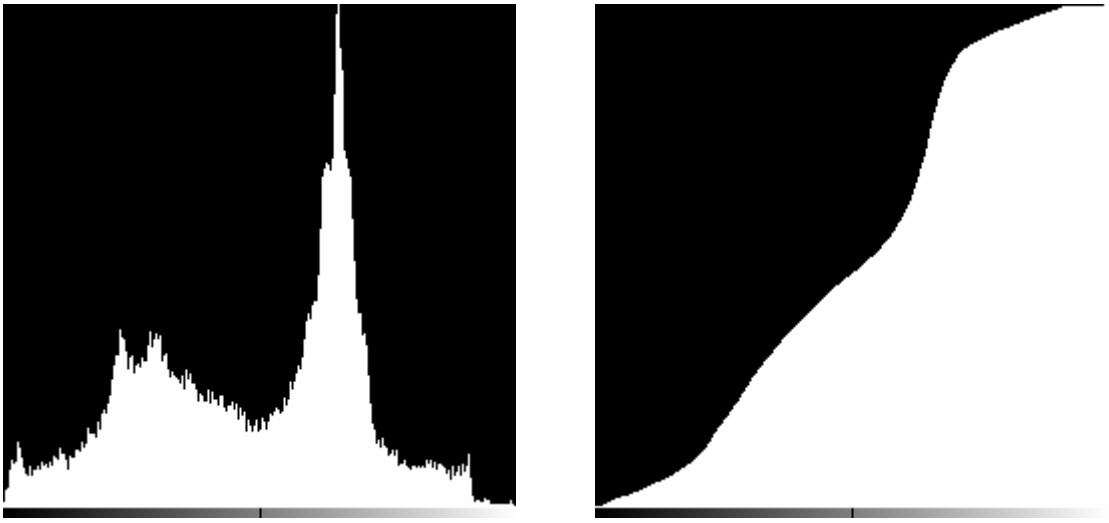


Abbildung 10.5: Das kumulative Histogramm (rechts) wird durch Aufaddieren der Häufigkeiten der einzelnen Grauwerte aus dem normalen Histogramm (links) berechnet.

10.3.1 Histogrammeinebnung

Ein häufig verwendetes globales Kriterium der Histogrammanipulation ist die Forderung, daß die Intensitätswerte des transformierten Bildes möglichst gleichmäßig über den gesamten zur Verfügung stehenden Grauwertbereich verteilt sind. Es wird dabei eine neue Look-Up-Tabelle berech-

net, d.h. ein Vektor *G'(i)*, der die neuen Intensitätswerte enthält, wobei die Grauwerte der zu transformierenden Bildpunkte als Index dienen [Wah84], [O´G88]. Diese Transformation bezeichnet man als *Histogrammeinebnung*.

$$G'(i) = \frac{G_{\max}}{T} H_x(x_i)$$

mit

$$H_x(x_i) = \left(\sum_{x = x_L}^{x_i} h(x) \right) - 0.5[h(x_i) + h(x_L)] \text{ der kumulativen Verteilungsfunktion und}$$

$$T = \left(\sum_{x = x_L}^{x_u} h(x) \right) - 0.5[h(x_u) + h(x_L)] \text{ die Anzahl der Pixel,}$$

$h(x_L)$ bzw. $h(x_U)$ die Häufigkeit des kleinsten (*lower*) bzw. größten (*upper*) Grauwertes sowie G_{max} dem größten möglichen Grauwert.

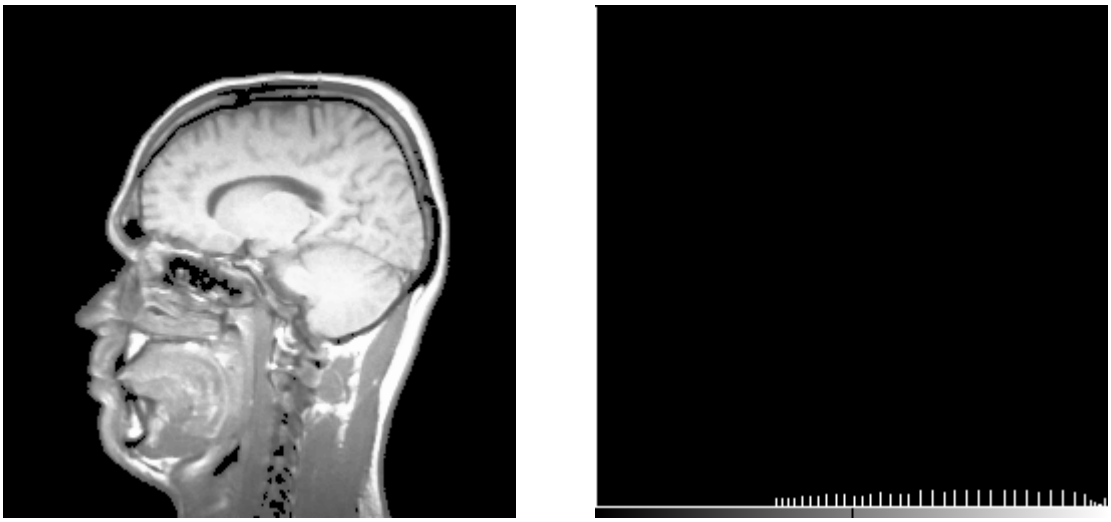


Abbildung 10.6: Die Histogrammeinebnung angewendet auf Abbildung 10.1, links, führt zu optisch guten Ergebnissen. Das zugehörige Histogramm zeigt die Unterschiede zur einfachen Grauwertmanipulation deutlich auf.

Statt zu sagen, daß in einem jeweils konstanten Grauwertintervall etwa gleich viele Bildpunkte enthalten sind, kann man dieses Kriterium noch schärfer formulieren, indem man fordert, daß alle Grauwerte gleich häufig vorkommen. Dies kann jedoch bei der Transformation zu Problemen führen, da die neuen Grauwerte nicht nur in Abhängigkeit des alten Wertes sondern auch in Abhängigkeit der Anzahl in der gerade betrachteten Klasse berechnet werden müssen. Aus diesem Grunde wird diese Art von Histogrammeinebnung normalerweise nur in der zuerst genannten einfacheren Form eingesetzt. Sofern nicht ein einzelner Grauwert stark im Originalbild dominiert, hat man aber

im allgemeinen den Eindruck einer Kontrastverstärkung. Kommt ein Grauwert oder ein eng begrenzter Grauwertbereich im Originalbild sehr häufig vor, so führt die Anwendung der Histogrammeinebnung zu unbefriedigenden Ergebnissen. Der Kontrast in den verbleibenden, weniger häufig vorkommenden Grauwertbereichen wird aufgrund des globalen Kriteriums der Histogrammeinebnung schlechter (siehe Abbildung 10.7).

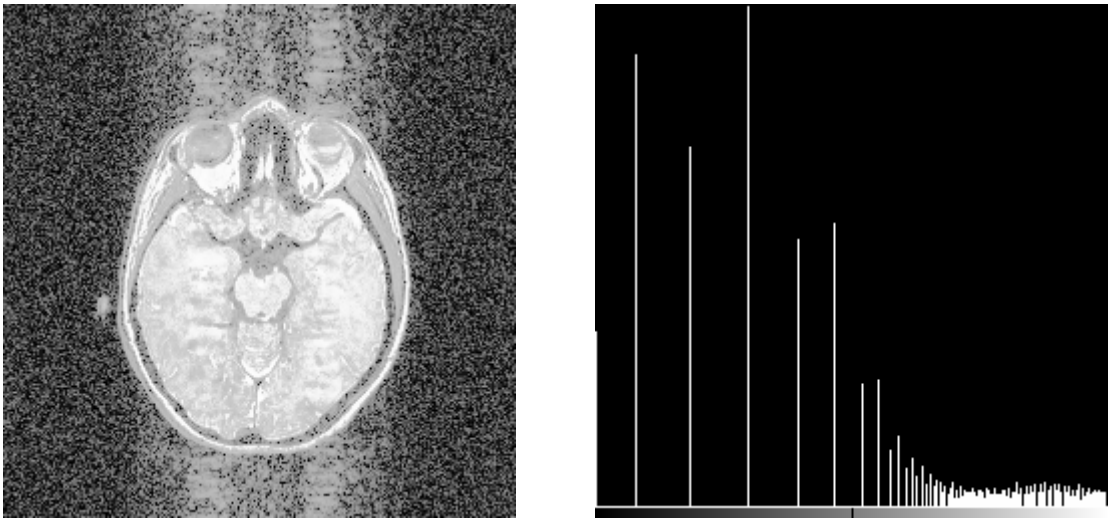


Abbildung 10.7: Ist ein Grauwert oder eng begrenzter Grauwertbereich in einem Bild dominant, so führt eine Histogrammeinebnung aufgrund des globalen Optimierungskriteriums zu optisch schlechteren Ergebnissen. Bei der CT-Aufnahme in diesem Beispiel wirkt sich der relativ große Hintergrund negativ aus. Das zugehörige Histogramm verdeutlicht dies noch einmal.

10.3.2 Adaptive Histogrammeinebnung

Damit sich das dominante Vorkommen eines Grauwertbereiches (z.B. der großflächige Hintergrund oder ungleichmäßig ausgeleuchtete Bilder) nicht auf das gesamte Bild auswirkt, wird bei der *adaptiven Histogrammeinebnung* (im Englischen auch mit *AHE = Adaptive Histogram Equalization* bezeichnet) die Vorlage in kleine Felder unterteilt, die jeweils etwa die Größe der interessierenden Strukturen haben [PAA*87], [VSM88]. In diesen Teilbildern erfolgt im ersten Schritt jeweils eine normale Histogrammeinebnung.

Der eigentliche Grauwert wird dabei durch relative Gewichtung zu den Nachbargebieten aus dem transformierten Grauwert berechnet. Somit entstehen glatte Übergänge zwischen den Feldern. Diese Gewichtung der Transformationskennlinien erfolgt umgekehrt proportional zum Abstand vom Mittelpunkt des aktuellen Feldes und dem der Nachbargebiete. Dadurch nimmt die Wirkung der pro Feld ermittelten Transformationsfunktion mit wachsendem (Euklidschem) Abstand vom Mittelpunkt der Teilbilder ab, die Wirkung der angrenzenden Gebiete entsprechend zu. Als Nach-

barfelder werden nur die direkt angrenzenden vier Felder betrachtet. Neben der Glättung der Übergänge haben gleichzeitig die einzelnen Grauwerte nur einen stark begrenzten Einfluß.

Beispiel:

Gegeben sei ein Bild der Größe $N \times N$ mit k möglichen Graustufen. Das Bild sei unterteilt in Felder der Größe $m \times m$ mit $m < N$. Der neue Grauwert $m(i)$ eines Bildpunktes an der Position (x, y) mit dem Grauwert i berechnet sich dann als bilineare Interpolation:

$$m(i) = a[bm_{--}(i) + (1 - b)m_{+-}(i)] + [1 - a][bm_{-+}(i) + (1 - b)m_{++}]$$

wobei gilt $a = (y - y_-) / (y_+ - y_-)$, $b = (x - x_-) / (x_+ - x_-)$ und $x+$, $x-$, $y+$, $y-$ die (x, y)-Koordinate des Mittelpunktes der entsprechenden Felder, m_{--} der transformierte Grauwert i im Feld links oben, $m_{+-}(i)$ der transformierte Grauwert i im Feld rechts oben, $m_{-+}(i)$ der transformierte Grauwert i im Feld links unten sowie $m_{++}(i)$ der transformierte Grauwert i im Feld rechts unten.

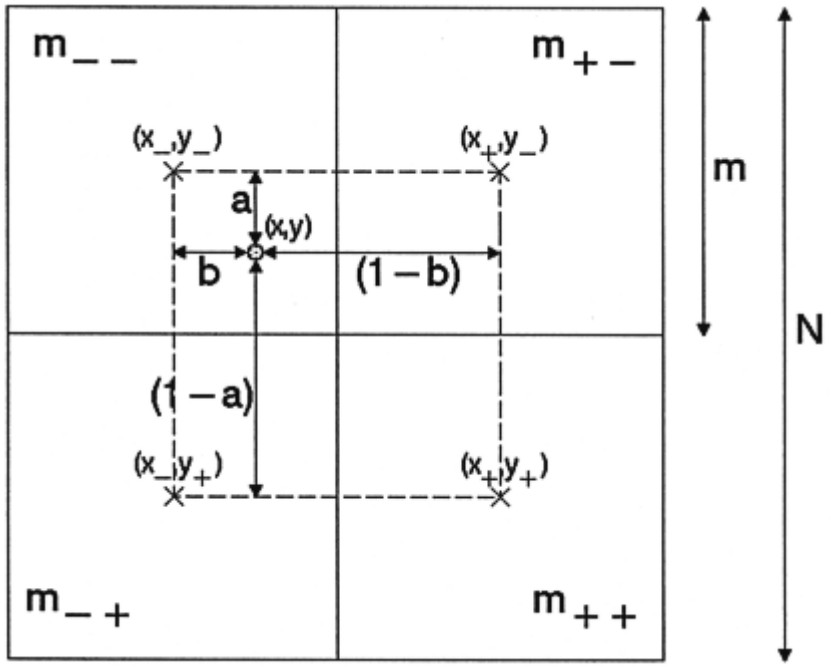


Abbildung 10.8: Der eigentliche Grauwert eines Bildpunktes berechnet sich aus dem in Abhängigkeit vom Abstand zum Feld-Mittelpunkt der Nachbarfelder gewichteten Mittelwert. Andere Nachbarschaftsbeziehungen bei der Berechnung als die hier verwendeten sind ebenfalls denkbar.

Durch diese lokale Histogrammeinebnung und Glättung der Übergänge zwischen den Feldern wird lokal ein optimaler Kontrast erreicht. Für die nachfolgenden Bearbeitungsstufen werden die einzelnen Gebiete gut unterscheidbar. Andererseits wirken sich häufig vorkommende Grauwerte nicht mehr negativ auf das Gesamtbild aus.

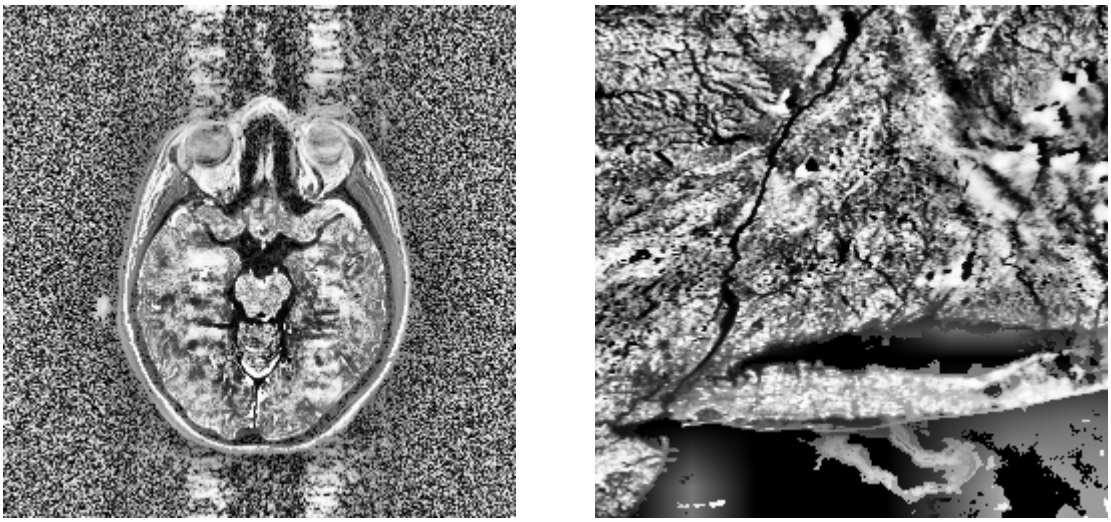


Abbildung 10.9: Bei der adaptiven Histogrammeinebnung (Größe 32×32) machen sich dominante Grauwerte nicht negativ bemerkbar. Selbst in Bereichen mit fast identischen Grauwerten im Originalbild sind durch die lokale Wirkungsweise der Transformationsfunktion gute Ergebnisse zu erzielen. Bei dem CT-Bild (vergleiche Abbildung 10.7) sind Strukturen des Hintergrundes zu erkennen. Nur an den Übergängen zwischen Hintergrund und Objekt wirkt das Bild etwas kontrastarm. Das rechte Bild zeigt eine Satellitenaufnahme von dem Gebiet um New York mit Long Island und dem Hudson River. Durch die adaptive Histogrammeinebnung können lokale Strukturen wie die Zuflüsse zum Hudson oder die Schattierungen im Wasser kontrastreich herausgearbeitet werden. Diese wären mit einer globalen Optimierung nicht sichtbar.

Im Gegensatz zur normalen Histogrammeinebnung existiert bei der adaptiven Histogrammeinebnung ein weiterer Parameter, der das Ergebnis beeinflussen kann, die Feldgröße. Werden die Felder zu groß gewählt, so nähert sich das Ergebnis der Transformation dem der normalen Histogrammeinebnung. An den Übergängen zwischen Objekt und dominantem Grauwert wirken die größeren transformierten Felder kontrastärmer. Ein besseres Ergebnis wird erreicht, wenn die Felder so klein sind, daß sie entweder ganz von den Objekten oder ganz von dem dominanten Grauwert ausgefüllt werden. Auf der anderen Seite wird durch zu kleine Felder Rauschen und jede kleine Grauwertänderung stark verstärkt.

Sinnvolle Transformationen ergeben sich, wenn in jedem Feld mindestens soviel Bildpunkte liegen, wie darstellbare Grauwerte zur Verfügung stehen. Erst dann sind die Häufigkeiten der einzelnen Grauwerte als repräsentativ anzusehen und die Schätzung der Grauwertverteilung aufgrund dieser Stichprobe ist einigermaßen korrekt. Weiterhin sollten die Felder nicht viel größer als die wichtigen Bildinhalte sein. Aufgrund dieser Betrachtungen ergibt sich als Minimalgröße ein Feld von 16×16 und eine sinnvolle Maximalgröße von 64×64 bei den üblichen 256 Graustufen und Bildgrößen von 512×512 Bildpunkten.

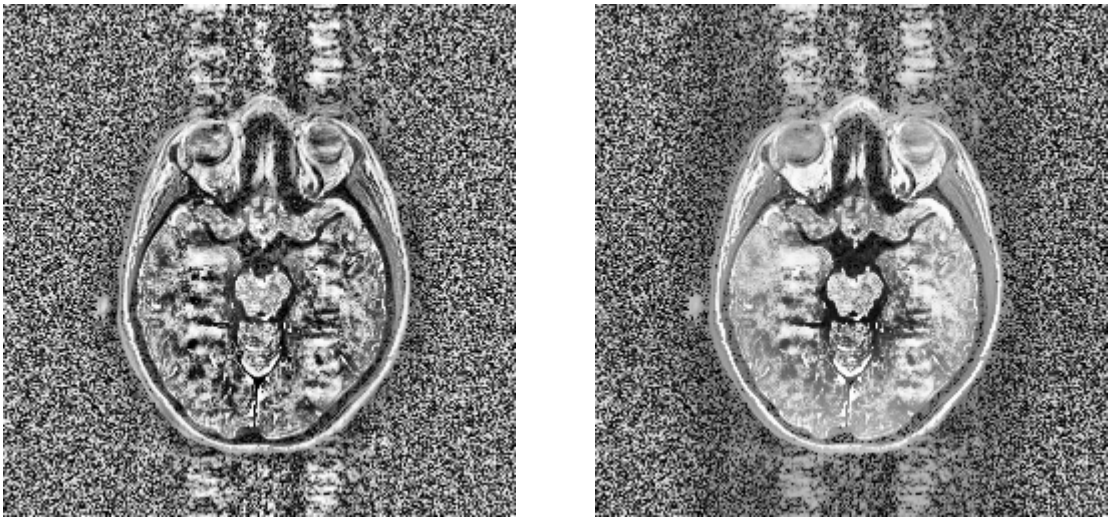


Abbildung 10.10: Das linke Bild wurde mit der adaptiven Histogrammeinebnung bei einer Feldgröße von 16×16, das rechte bei 64×64 erzeugt. Durch die relative kleine Größe wirkt das Bild links zu stark verrauscht. Rechts: das Bild nähert sich im Aussehen schon der normalen Histogrammeinebnung (siehe Abbildung 10.7).

```
  void AHE(int BildNr, int BildNr2, int Fenster)

  /* BildNr  = das zu bearbeitende Quellbild                      */
/* BildNr1 = das Ergebnisbild                                   */
  /* Fenster = Fenstergroesse 16 ... 64, d.h. 16x16 ... 64x64    */
  {
    int z, s;                      /* Zeilen-/Spalten-Index      */
    int Zeile, Spalte;
    unsigned char Tr_Matrix[16][16][256];  /* Transformationsfkt.  */
                                  /* (fuer jeden Block) max. 16x16 */
    int Bloecke;                  /* Anzahl der Bloecke          */
    int Offset_Zeile, Offset_Spalte;
    int i, j, k;
    int X_low, X_high;            /* Maximaler/Minimaler Grauwert  */
    float T, a, b;                /* Hilfsvariablen fuer Interpol. */
    int X_Mitte, Y_Mitte;         /* Mittelpunkt-Koordinaten       */
    unsigned int Histo[MAX_COLOR];/* Histogramm                    */
    unsigned int Kum_Histo[MAX_COLOR]; /* Kumulatives Histogramm   */
    float Abstand_XY[64];         /* Hilfsvektor fuer die Abstaende*/

    ClearTextWindow(35,18,80,25);
    WriteText(35,18,"* Adaptive-Histogrammeinebnung *");

    X_Mitte=Fenster/2;        /* Koordinaten des Blockmittelpunktes*/
    Y_Mitte=Fenster/2;

    Bloecke=Picture[BildNr].Zeilen/Fenster;  /* Anzahl der Bloecke */
```

```
/* Distanzen fuer die Interpolation vorher berechnen           */
for (i=0; i<Fenster; i++)
    Abstand_XY[i] = (float)(i) / (float)(Fenster);

/* Transformations-Funktionen der einzelnen Felder berechnen   */
for (i=0; i<Bloecke; i++) {
    Offset_Zeile = i*Fenster;
    for (j=0; j<Bloecke; j++) {
        Offset_Spalte = j*Fenster;
        /* Histogramm initialisieren                           */
        for (k=0; k<MAX_COLOR; k++) Histo[k] = 0;

        /* Histogramm fuer das aktuelle Feld ermitteln         */
        for (z=0; z<Fenster; z++)
            for (s=0; s<Fenster; s++) {
                Zeile = Offset_Zeile + z;
                Spalte= Offset_Spalte + s;
                Histo[ Picture[BildNr].Bild[Zeile][Spalte] ]++;
            }
        /* Kumulatives Histogramm fuer das Feld aufstellen     */
        Kum_Histo[0] = Histo[0];
        for (k=1; k<MAX_COLOR; k++)
            Kum_Histo[k] = Kum_Histo[k-1]+Histo[k];
        /* Kleinsten vorkommenden Grauwert X_low suchen        */
        k = -1;
        while( !Histo[++k] );
        X_low = Histo[k];
        /* Groessten vorkommenden Grauwert X_high suchen       */
        k = MAX_COLOR;
        while( !Histo[--k] );
        X_high = Histo[k];

        T = (float)(Kum_Histo[MAX_COLOR-1]) -
                    0.5*((float)(X_high + X_low));
        if (fabs(T)>0.0) T=(float)(WEISS)/T;

        for (k=0; k<MAX_COLOR; k++)
            Tr_Matrix[i][j][k] =
                (int)(T * ((float)Kum_Histo[k] -
                                0.5*(float)(Histo[k]+X_low)));
    }
}
```

```
   /* Jetzt die eigentliche Transformation unter Berueecksichtigung*/
   /* der Nachbarfelder durchfuehren.                              */
   /* Der Transformationsbereich ueberlappt 4 Fenster. An der      */
   /* Stelle, wo diese Fenster zusammenstossen, liegt der Mittel- */
   /* punkt des Transformationsbereichs.                           */
   for (i=0; i<Bloecke-1; i++) {
      Zeile=Y_Mitte+i*Fenster;
      for (j=0; j<Bloecke-1; j++) {
         Spalte=X_Mitte+j*Fenster;

         for (z=0; z<Fenster; z++) {
            a = Abstand_XY[z];
            for (s=0; s<Fenster; s++) {
               b = Abstand_XY[s];
               /* Alten Grauwert des Pixels als Index verwenden   */
               Grauwert = Picture[BildNr].Bild[Zeile+z][Spalte+s];
               Picture[BildNr].Bild[Zeile+z][Spalte+s] =
                   (unsigned char)
                   (a * (b*(float)Tr_Matrix[i+1][j+1][Grauwert] +
                       (1.0-b)*(float)Tr_Matrix[i+1][j][Grauwert]) +
                   (1.0-a)*(b*(float)Tr_Matrix[i][j+1][Grauwert] +
                       (1.0-b)*(float)Tr_Matrix[i][j][Grauwert])
                   );
            }
         }
      }
   }
   /* Bis hierher sind nur die innenliegenden Grauwerte trans-    */
   /* formiert worden. Der Rand muss jetzt noch gesondert berueck-*/
   /* sichtigt werden, z.B. durch geeignete Gewichtung mit den    */
   /* noch vorhandenen Nachbarn oder einfach durch Verwendung der */
   /* Ergebnisse der normalen Histogrammeinebnung.                */
}
```

10.4 Medianfilter

Die bisher vorgestellten Methoden der Histogrammanipulation waren rein bildpunktbezogen. Man bezeichnet diese daher auch als *Punktoperationen*. Wird bei der Bildverbesserung noch die Umgebung eines Bildpunktes mitberücksichtigt, so spricht man von *(digitalen) Filtern im Ortsbereich*.

Eines der einfachsten derartigen Verfahren besteht darin, die Grauwerte eines Bildes zu glätten, indem der Grauwert eines jeden Punktes durch den *Mittelwert* der Grauwerte seiner Umgebungs-

punkte ersetzt wird. Aus Gründen der Symmetrie wird meistens eine quadratische Umgebung der Gesamtgröße von 3×3, 5×5 oder mehr Bildpunkten verwendet.

$$g'(x, y) = \frac{1}{(2n+1)^2} \sum_{i=-n}^{n} \sum_{j=-n}^{n} g(x+i, y+j) \quad \text{für festes n} = 1, 2, 3, \ldots$$

Am Bildrand sind besondere Vorkehrungen zu treffen. Dies kann durch Spiegelung des Bildinhaltes an den Bildrändern oder durch Setzen der außerhalb liegenden Bildpunkte auf Schwarz bzw. Weiß. Letzteres führt aber meist zu sichtbaren Fehlern im gefilterten Bild.

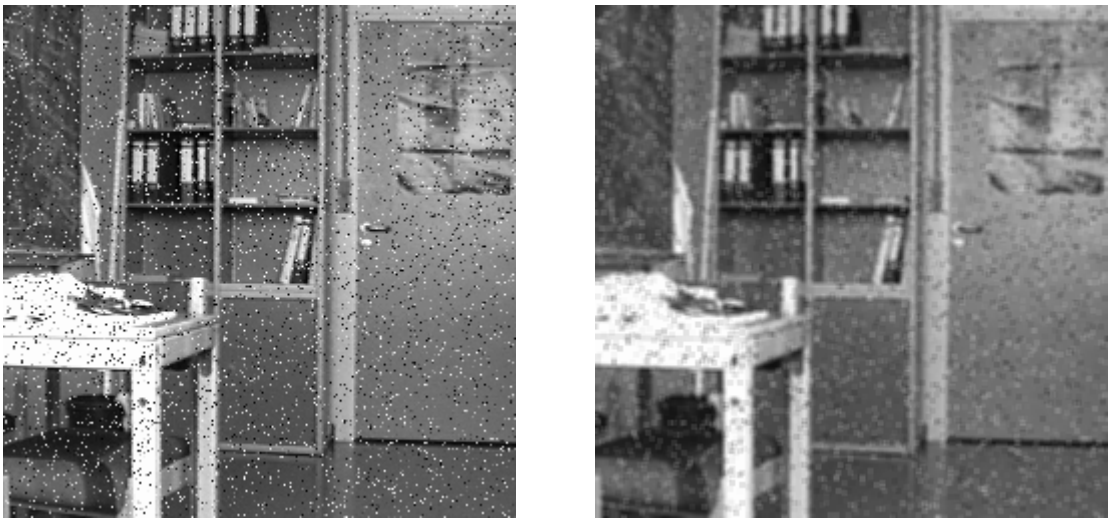


Abbildung 10.11: Einer der einfachsten Filter im Ortsbereich zur Bildglättung ist der Mittelwertfilter. Das verrauschte Originalbild links ist mit einer 3×3-Maske gefaltet und das Ergebnis rechts dargestellt.

Bilder, die mit diesem Operator bearbeitet (gefaltet) werden, wirken im Vergleich zum Original "weicher" oder etwas unschärfer. Der Grauwert-Mittelwert des neuen Bildes ist derselbe wie im alten Bild, wogegen die Streuung im neuen Bild kleiner wird. Das bedeutet, daß das dem Originalbild überlagerte Rauschen durch die Filterung verringert wurde. Strukturen mit einer kleinen Wellenlänge werden jedoch nur in der Amplitude gedämpft, nicht aber komplett herausgefiltert.

Ein großer Nachteil dieser Methode zur Glättung ist die Tatsache, daß das Rauschen nur vermindert, nicht aber komplett beseitigt wird, und daß im neuen Bild durch die Mittelung unter Umständen Grauwerte verwendet werden, die im Originalbild nicht vorhanden waren. Die Verbesserung verrauschter Bildsignale mittels dieser ortsinvarianten Tiefpaßoperation stellt somit nur einen Kompromiß zwischen der Unterdrückung des dem Bildsignal überlagerten Rauschens und der Erhaltung feiner Bildstrukturen dar. Der Mittelwertfilter ist daher kein guter Tiefpaßfilter.

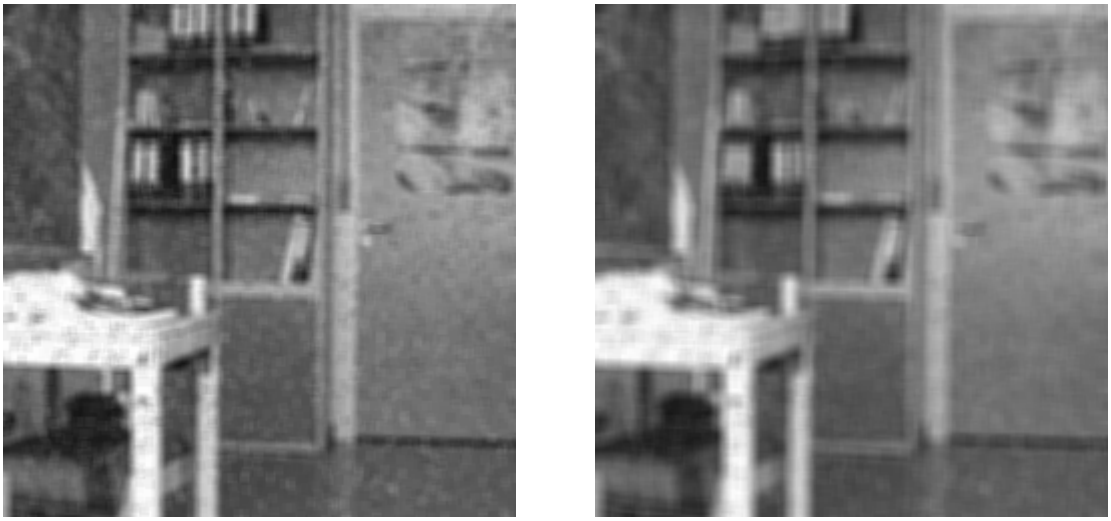


Abbildung 10.12: Bei der Mittelwertfilterung mit größeren Masken wirkt das Ergebnis unschärfer. Links ist das Originalbild aus Abbildung 10.11 mit einer 5×5 Maske, rechts mit einer 7×7-Maske geglättet.

Die Nachteile des Mittelwertfilters werden durch den nichtlinearen *Medianfilter* verhindert. Der Medianfilter wurde Anfang der 70er Jahre zur nichtlinearen Glättung von Signalen vorgeschlagen [HYY79], [GW81]. Allgemein gehört der Medianfilter zu den Rangordnungsfiltern, d.h., die Eingangsdaten w_i mit $w_i \in W$ und W = Wertebereich des Eingangssignals werden sortiert und der w_m-größte Wert der aufsteigend sortierten Folge $f(k)$ mit $k \in W$ wird als Funktionswert genommen. Beim Medianfilter entspricht w_m dem mittleren Wert dieser Folge, dem Median. Als Vorteile dieses Filters seien genannt

- der Erhalt scharfer Kanten,
- die gute Unterdrückung von Störimpulsen,
- die Eignung bei nicht additiven und korrelierten Störungen,
- die einfache, wenn auch unter Umständen zeitaufwendige Berechnung.

Im Zusammenhang mit der Bildverarbeitung ist der Medianfilter wie folgt definiert:

Ist $A(x, y)$ die Matrix-Darstellung eines digitalisierten Bildes, dann ist das Ergebnis des Medianfilters mit einer Fenstergröße von $(2m+1)\times(2n+1)$, mit $m, n = 1, 2, ...$, ein Bild $B(x, y)$. Jedes Element von $B(x, y)$ entspricht dem Median der Grauwerte des Originalbildes, die in einem Fenster der Größe $(2m+1)\times(2n+1)$ berücksichtigt wurden. Das mittlere Element in diesem Fenster wird dabei immer auf den Medianwert gesetzt. Zur Vereinfachung wird meist $m = n$ verwendet. Weiterhin ist aus Geschwindigkeitsgründen $m = n = 1, 2$ oder 3 üblich.

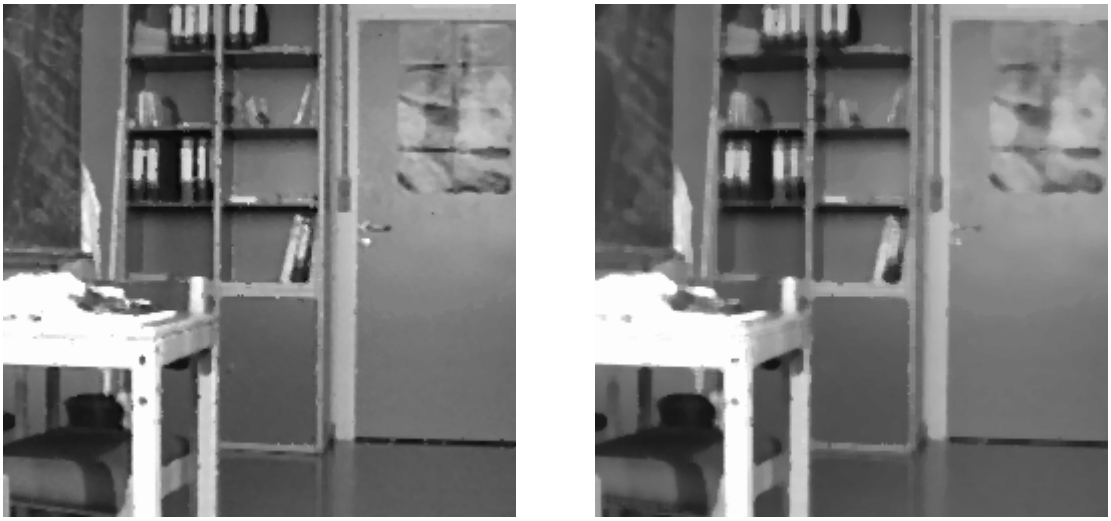


Abbildung 10.13: Bei der Glättung mit dem nichtlinearen Medianfilter erfolgt eine sehr gute Rauschunterdrückung. Links ist das gestörte Bild aus Abbildung 10.11 mit einem 3×3-Median, rechts mit einem 5×5-Median geglättet worden.

Strukturen und Störungen wie z.B. Rauschen bis zu einer Ausdehnung von n Pixel in einer Richtung werden somit durch den Medianfilter vollständig beseitigt. Eine Anwendung dieses Filters reduziert die Varianz der Grauwerte im Bild. Dabei wird meist auch der mittlere Grauwert geändert, jedoch werden keine neuen Grauwerte wie bei dem Mittelwertfilter erzeugt. Durch bestimmte Filterformen (Kreuz-Form, Vektor-Form) können bestimmte Strukturen in der Vorlage besser berücksichtigt und erhalten werden. So ist es unter anderem auch möglich, kleine Lücken in Strukturen mit Hilfe des Medianfilters zu schließen (vergleiche Kapitel 14). Einer der größten Nachteile des Medianfilters ist die laufend notwendige Sortierung der Eingangswerte. Diese Sortierung kann mit bekannten Verfahren wie Quicksort, Bubblesort, mit verketteten Listen usw. durchgeführt werden. Trotz der einfachen Berechnung ergeben sich dadurch bei Softwarelösungen lange Berechnungszeiten. Zur Beschleunigung wird daher neben der geänderten Filterform auch oft als Näherung der Median der Mediane verwendet, d.h. es wird bei einer $(2n+1)\times(2n+1)$-Maske der Median einer jeden Zeile und einer jeden Spalte ermittelt und als Gesamtergebnis der Median dieser $2(2n+1)$ Medianwerte verwendet [Ner81]. Dies ist aber nur eine Näherung des eigentlichen Medianwertes. Insgesamt gelten folgende Rechenregeln bei dem Umgang mit dem Medianfilter:

$$med(c + f(x)) = c + med(f(x))$$

$$med(c\, f(x)) = c\, med(f(x))$$

$$med(f(x)) + med(g(x)) \neq med(f(x) + g(x))$$

Eine schnelle, wenn auch programmtechnisch etwas aufwendigere Möglichkeit der Medianfilterung ohne laufende Neusortierung der Werte ist die sogenannte Histogrammethode [HYY79]. Bei dieser Berechnung wird die Überlappung der einzelnen Filter-Fenster ausgenutzt, denn bei der Filterung fallen nur $(2n+1)$ alte Werte aus der Liste heraus und $(2n+1)$ neue kommen hinzu. Alle anderen $(2n+1)^2-2(2n+1)$ Werte bleiben erhalten. Je größer die Filtermaske, desto größer ist daher der Vorteil dieser Histogrammethode. Als Vorbereitung wird zuerst das Histogramm der $(2n+1)\times(2n+1)$ Werte und das dazugehörige kumulative Histogramm berechnet. Die Stelle, bei der die Häufigkeit von $(2n+1)^2/2$ im kumulativen Histogramm überschritten wird, wird bestimmt. Der Grauwert an dieser Stelle entspricht dem Median. Im nächsten Schritt werden die $(2n+1)$ alten Werte aus dem Histogramm entfernt (die zugehörigen Häufigkeiten erniedrigt) und die $(2n+1)$ neuen Werte eingefügt (die zugehörigen Häufigkeiten erhöht) und das kumulative Histogramm bis zur Schwelle $(2n+1)^2/2$ wieder berechnet. Die Filterung des Originalbildes kann dabei zeilenweise von links nach rechts oder zeilenweise in Schlangenlinie erfolgen. Bei der zeilenweise Abarbeitung von links nach rechts muß am Zeilenanfang jeweils das Histogramm komplett neu berechnet werden.

10.5 Tiefpaßfilter

Wie schon bei der Fourier-Transformation deutlich wurde, entsprechen Kanten und starke Grau-/Farbwertänderungen im Bild den hochfrequenten Anteilen der Fourier-Transformierten. Es besteht daher die Möglichkeit, durch Multiplikation der Fourier-Transformierten eines Bildes im Frequenzbereich mit einer Übertragungsfunktion $H(u, v)$ eine Filterung vorzunehmen.

$$G(u, v) = H(u, v)\, F(u, v)$$

Die inverse Transformation von $G(u, v)$ liefert dann das gefilterte Bild. Für einen *idealen Tiefpaßfilter* ist die Funktion $H(u, v)$ definiert als (siehe Abbildung 10.14)

$$H(u, v) = \begin{cases} 1 & falls\ D(u,v) \le D_0 \\ 0 & sonst \end{cases}$$

wobei D_0 die *Grenzfrequenz* und $D(u, v)$ die Entfernung vom Ursprung ist.

$$D(u, v) = \sqrt{u^2+v^2}$$

Obwohl diese Ideale-Filter-Definition sehr einleuchtend ist, entspricht der steile Abfall nicht den physikalisch realisierbaren Möglichkeiten. Will man die physikalischen Gegebenheiten näher berücksichtigen, so verwendet man oft den Butterworth-Filter, der im folgenden noch näher besprochen wird.

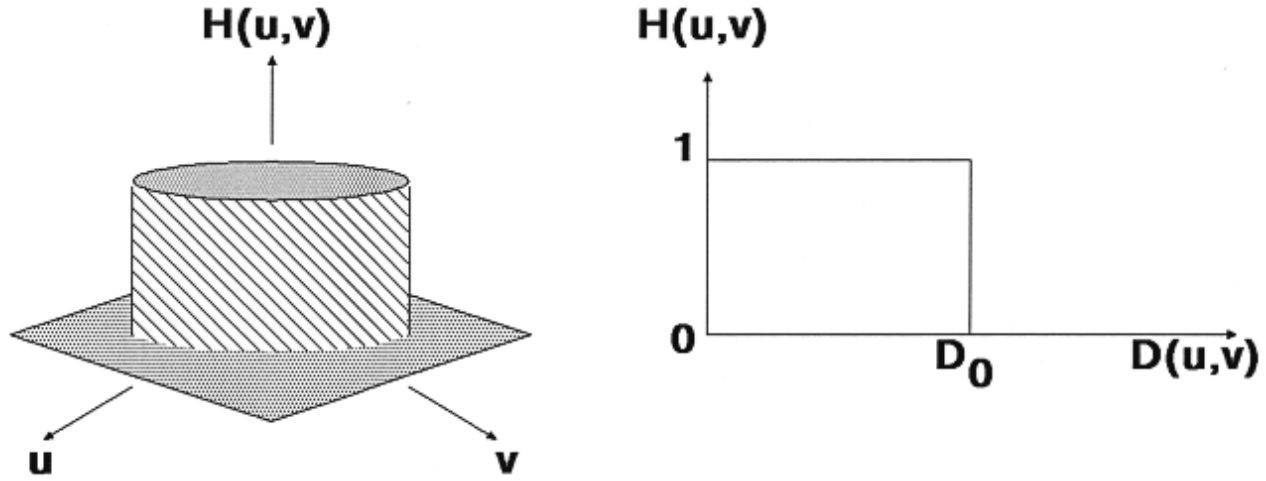


Abbildung 10.14: Die Übertragungsfunktion für einen idealen Tiefpaßfilter mit der Grenzfrequenz D_0.

Während die oben vorgestellte Methode im Frequenzbereich arbeitet, gibt es auch die Möglichkeit, im Ortsbereich zu filtern. Eine einfache Mittelung über eine $n \times n$ Umgebung oder Medianfilterung stellt ebenfalls eine Tiefpaßfilterung dar.

Da bei einem Tiefpaßfilter hohe Frequenzanteile abgeschwächt oder ganz eliminiert werden, erscheint das gefilterte Bild im Vergleich zum Original unschärfer, Rauschen und Störungen ähnlicher Art werden dabei aber beseitigt. Möchte man eine Bildverschärfung, so muß eine Hochpaßfilterung durchgeführt werden.

10.6 Hochpaßfilter

Die Übertragungsfunktion für einen *idealen Hochpaßfilter* im Frequenzbereich ist in Abbildung 10.16 wiedergegeben. Durch solch eine Filterung bleiben Stellen mit abrupten Grauwertänderungen wie Kanten, Rauschen u.ä. deutlicher erhalten. Wird das Ergebnis dieser Filterung zum Ausgangsbild hinzuaddiert, so erhält man eine Kantenverstärkung und Verschärfung des Bildes.

$$H(u, v) = \begin{cases} 1 & falls\, D(u,v) > D_0 \\ 0 & sonst \end{cases}$$

Eine Bildverschärfung kann nicht nur durch Filterung im Frequenzraum und anschließende Rücktransformation sondern auch durch örtliche Differenzoperatoren erreicht werden. Prinzipiell ist dazu jeder Kantenoperator geeignet, da Kanten ja Stellen mit starker Grauwertänderung, also höherfrequente Anteile im Frequenzraum repräsentieren. Eine besonders einfache Methode zur Bildverschärfung wird im nachfolgenden Abschnitt vorgestellt.

Abbildung 10.15: Analog zur Mittelwertfilterung wirkt das Bild nach der Filterung im Frequenzraum unschärfer. Zur Vereinfachung wurde hier nicht ein runder, sondern ein quadratischer Filterkern verwendet.

10.7 Butterworth-Filter

Wie schon erwähnt, repräsentieren die idealen Filter nicht die physikalischen Gegebenheiten. An der Grenzfrequenz werden in Wirklichkeit nicht alle Anteile, die darüber bzw. darunter liegen, abgeschnitten, sondern ab dieser Stelle stärker gedämpft, meist mit exponentiellem Abfall (z.B. mit Hilfe einer Gauß-Funktion). Besonders an Bedeutung gewonnen hat in diesem Zusammenhang die Übertragungsfunktion des *Butterworth-Filters*.

$$H(u, v) = \frac{1}{1+[D(u,v)/D_0]^{2n}} \text{ Tiefpaßfilter}$$

$$H(u, v) = \frac{1}{1+[D_0/D(u,v)]^{2n}} \text{ Hochpaßfilter}$$

Leider ist bei dieser Art von Filter keine Grenzfrequenz in der gewohnten Art und Weise definiert. Man verwendet daher oft die Stelle $D(u, v)$ für D_0, an der $H(u, v) = 0.5$, an der also $H(u, v)$ nur noch 50% seines Maximalwertes (Intensität) besitzt.

10.8 Kantenverstärkung

Zur Kantenverstärkung und damit zur allgemeinen Bildverschärfung kann ein einfacher örtlicher Differenzoperator nach folgender Formel verwendet werden [Knu87]:

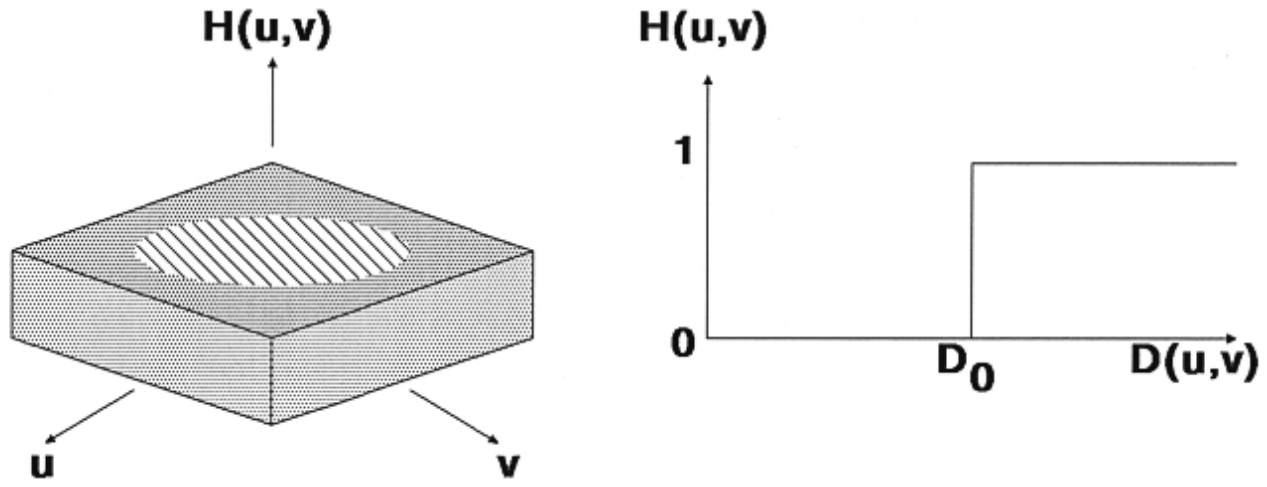


Abbildung 10.16: Die Übertragungsfunktion für einen idealen Hochpaßfilter mit der Grenzfrequenz D_0.

$$A'(i,j)=\frac{A(i,j)-\alpha\overline{A}(i,j)}{1-\alpha}$$

mit $0 \le \alpha \le 1$als "Verstärkungsfaktor" und

$$\overline{A}(i,j)=\frac{1}{9}\sum_{u=i-1}^{i+1}\sum_{v=j-1}^{j+1}A(u,v)$$

als Mittelwert. Für $\alpha = 0$ bleibt das Bild unverändert, wie man leicht aus der Formel ablesen kann. Je größer α wird, desto stärker werden die möglichen Kanten betont. Dieser Filter stellt also keinen direkten Hochpaßfilter dar, sondern er verstärkt die hochfrequenten Anteile. Diese Verstärkung wird im Ortsbereich durchgeführt.

Je näher α an 1 gewählt wird, desto verrauschter wirkt das Ergebnis. Die geeignete Wahl für α ist zwar abhängig vom Originalbild, Werte im Bereich von 0.6 bis 0.9 haben sich aber allgemein als günstig herausgestellt.

```
void EdgeEnhance(int BildNr, int BildNr2)
{
  int s, z, i, j;                /* Zeilen-/Spalten-Index        */
  unsigned char Pixel;           /* Original-Grauwert des Pixels */
  float Zwerg, Ein_Neuntel;      /* Hilfsvariablen               */
  float Alpha;                   /* "Verstaerkungsfaktor"        */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Kantenverstaerkung *");
```

Abbildung 10.17: Bei der Hochpaßfilterung bleiben nur Stellen mit starken Grauwertänderungen (hochfrequente Anteile) unbeeinflußt. Zur Vereinfachung wurde hier nicht ein runder, sondern ein quadratischer Filterkern verwendet.

```
    /* Alpha wird durch den Benutzer im Intervall [0,1) gewaehlt   */
    WriteText(35,22,"Bitte Faktor ALPHA eingeben");
    do {
        gotoxy(35,23);
        Alpha = Read_Float("Wertebereich (0.0-0.99) ", 0.75, ':' );
    } while ((Alpha<0.0) || (Alpha>=1.0));

    Ein_Neuntel = 1.0 / 9.0;

  /* Originalbild zuerst einmal ins Zielbild kopieren wegen Rand */
    for (z=0; z<Picture[BildNr].Zeilen; z++)
        for (s=0; s<Picture[BildNr].Spalten; s++)
            Picture[BildNr2].Bild[z][s]=Picture[BildNr].Bild[z][s];

    /* 1-Pixel breiter Rand bleibt hier unberuecksichtigt          */
    for (z=1; z<(Picture[BildNr].Zeilen-1); z++) {
        for (s=1; s<(Picture[BildNr].Spalten-1); s++) {
            /* Pixel-Werte im 3x3-Fenster aufsummieren             */
            Pixel = Picture[BildNr].Bild[z][s];
            Zwerg = 0;
            for (i=(z-1); i<=(z+1); i++)
                for (j=(s-1); j<=(s+1); j++)
                    Zwerg += Picture[BildNr].Bild[i][j];
            Zwerg = Ein_Neuntel * Zwerg;     /* Mittlerer Grauwert */

            /* Moeglichen neuen Grauwert berechnen und ...         */
            Zwerg = ( (float)Pixel - Alpha*Zwerg ) / (1.0-Alpha);
```

```
            /* ... ggf. noch ein Klipping durchfuehren.            */
            if      (Zwerg>(float)WEISS)   Zwerg=(float)WEISS;
            else if (Zwerg<(float)SCHWARZ) Zwerg=(float)SCHWARZ;

            Picture[BildNr2].Bild[z][s] = (unsigned char)Zwerg;
        }
    }
}
```

Aufgaben

Aufgabe 1

Ordnen Sie den vorgestellten Bildverbesserungsverfahren die Attribute lokal, global, linear, nichtlinear, Anwendung im Ortsbereich, Anwendung im Frequenzbereich zu.

Aufgabe 2

Gegeben ist folgende 3×3-Maske mit identischen Koeffizienten:

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

1. Was bewirkt die Anwendung dieser Filtermaske?
2. Wozu dient der Vorfaktor 1/9
3. Aus welchem Grund ist bei jeder Glättungsmaske die Summe aller Koeffizienten 1?

Aufgabe 3

Gegeben sind zwei Folgen $f = \{0, 7, 8, 2, 1\}$ und $g = \{6, 7, 6, 4, 7\}$. Zeigen Sie, daß

$$med(f) + med(g) \neq med(f + g)$$

ist, d.h. daß der Medianfilter nicht linear ist (med = Medianberechnung).

Aufgabe 4

Nennen Sie tabellarisch die Unterschiede und Vor-/Nachteile zwischen der Medianfilterung und der Filterung durch Mittelwertbildung.

Aufgabe 5

Geben Sie die Gleichung für die Transformation der Grauwerte G eines Bildes an, die den Bereich von 0-50 auf den Bereich 0-150, 50-100 auf 150-200 und 100-170 auf 200-255 transformiert.

Aufgabe 6

Von einem binären Bild (Hintergrund=0, Linien=255) einer technischen Zeichnung weiß man, daß es nur horizontale und vertikale Linien enthält. Geben Sie einen Satz von 3×3-Filtermasken an, mit denen man 1-Pixel große Lücken in diesen Linien entdecken kann.

Wie müssen die Masken aussehen, wenn diese Lücken damit geschlossen werden sollen?

11. Kantendetektion

Versuche haben gezeigt, daß Menschen sich beim Betrachten von Objekten sehr stark auf die Grenzen zwischen mehr oder minder homogenen Regionen konzentrieren. Gegenstände werden meist schon anhand der groben Umrisse erkannt.

Auch bei der Digitalen Bildverarbeitung stellen diese Kanten bzw. Konturen eine wichtige Stufe zur Bildsegmentierung, Objekterkennung und somit zur Bildinterpretation dar.

Wie gut diese Kanten in den Vorlagen gefunden werden, ist abhängig von der Qualität des Bildes, der Art der Kanten und dem verwendeten Algorithmus. Je nach Kantenart eignet sich das eine oder andere Verfahren besser. Die auf dem Gebiet der Bildverarbeitung wichtigsten Operatoren, ihre unterschiedlichen Eigenschaften sowie ihre bevorzugten Einsatzgebiete werden in den nachfolgenden Abschnitten vorgestellt.

11.1 Was ist eine Kante? Kantenmodelle

Bevor Verfahren zur Kantendetektion vorgestellt werden, sollte zuerst der Begriff der *Kante* in der Bildverarbeitung definiert werden. Kanten bei physikalischen Körpern lassen sich leicht beschreiben. Es werden hierbei in der Regel die Objektgrenzen, abrupte Änderungen der Oberflächennormalen oder einfach Änderungen der Materialeigenschaften verstanden.

In der Bildverarbeitung werden die Objekte in den Aufnahmen durch unterschiedlich intensive Grau-/Farbwerte dargestellt. Eine Kante ist dann eine Diskontinuität im Verlauf dieser Intensitätswerte. Bei entsprechender Beleuchtung und Aufnahmeansicht entsprechen die Kanten der physikalischen Körper den "Intensitäts-Kanten" in den Bildern. Daher haben die Kantendetektionsverfahren in der Bildverarbeitung eine bedeutende Rolle. Jedoch können auch Pseudo-Kanten auftreten, wie bei Schatten, ungünstigen Beleuchtungen usw.

Aufgrund der Änderung der Intensitätswerte lassen sich verschiedene Modelle für das Profil einer Kante definieren (vergleiche Abbildung 11.1):

- **ideale Stufenkante**

  Bei einer idealen Stufenkante *S(x)* ändert sich der Funktionswert (Grauwert) von einem zum nächsten Argument. Im diskreten Raster stellt dies die allgemeinste Kantenform dar. Ist der Ort des Wertewechsels in den Nullpunkt ($x = 0$) zentriert, so läßt sich die Stufenkante folgendermaßen beschreiben:

$$S(x)=\begin{cases} a & x<0 \\ b & sonst \end{cases}$$

- **ideale Rampenkante**

  Die ideale Rampenkante *R(x)* ändert, wie der Name schon andeutet, langsam ihren Funktionswert in Abhängigkeit von *x*. Da hier der genaue Ort der Kante nicht so einfach festgelegt werden kann wie bei der Stufenkante, wird dieses Modell für ausgedehnte oder verschmierte Kanten verwendet. Meist wird als Ort der Kante die Mitte der Rampe definiert. Zusammen mit der Breite *w* der Rampe ergibt sich dann:

$$R(x)=\begin{cases} a & x<-w/2 \\ a+(b-a)(x+w/2) & -w/2\le x\le w/2 \\ b & x>w/2 \end{cases}$$

- **ideale Dachkante**

  Die ideale Dachkante *D(x)* ist eine Sonderform der Rampenkante. Neben einem langsamen Anstieg des Funktionswertes erfolgt wieder eine Verminderung nach dem Erreichen des lokalen Maximums. Dieses lokale Maximum wird als Ort der Kante angesehen. Mit *w* ist hier die Gesamtbreite der Dachkante bezeichnet:

$$D(x)=\begin{cases} a & x<-w/2 \\ a+2(b-a)(x+w/2) & -w/2\le x\le 0 \\ a+2(b-a)(w/2-x) & 0<x\le w/2 \\ a & x>w/2 \end{cases}$$

- **ideale Treppenkante**

  Eine ideale Treppenkante *T(x)* setzt sich aus mehreren Stufenkanten zusammen. Als Beispiel wird hier eine Treppenkante, bestehend aus zwei aufeinanderfolgenden Stufenkanten betrachtet. Die eigentliche Position der Kante wird als der Mittelpunkt des Plateaus mit der Breite *w* der Stufe angesehen:

$$T(x)=\begin{cases} a & x<-w/2 \\ b & -w/2\le x\le w/2 \\ c & x>w/2 \end{cases}$$

- **reale Kantenform**

  Die reale Kantenform ist meist eine der oben gezeigten idealen Kanten, überlagert mit Störungen in Form von Rauschen. Oft sind auch noch mehrere ideale Kantenformen in der realen Kante überlagert wiederzufinden. Der genaue Ort läßt sich bei gestörten Kantenformen nicht so einfach definieren. In der Praxis wird daher noch zur genauen Lokalisation die Differenz der Funktionswerte (Grauwerte) von Nachbarpunkten betrachtet (erste und zweite Ableitung).

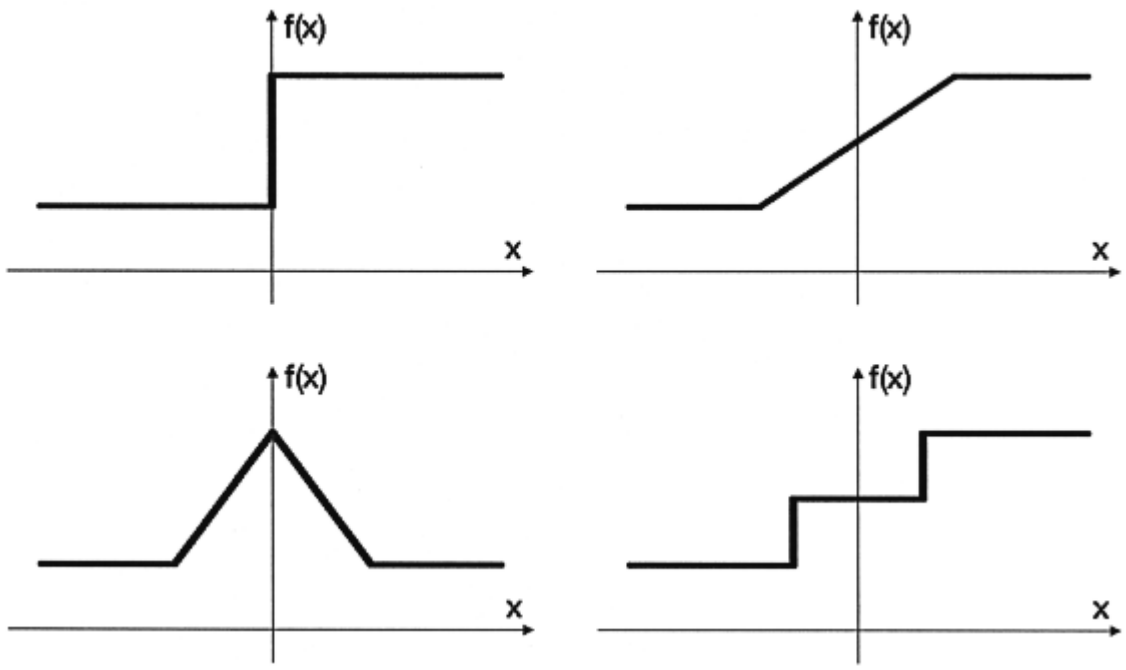


Abbildung 11.1: Die Profile einiger idealer Kantenmodelle. Links oben: die ideale Stufenkante; Rechts oben: die ideale Rampenkante; Links unten: die ideale Dachkante; Rechts unten: die ideale Treppenkante.

11.2 Anforderungen und Schwierigkeiten bei der Kantendetektion

Wenn in der Digitalen Bildverarbeitung eine Kantendetektion mit dem Rechner durchgeführt wird, sollte natürlich eine möglichst geringe Fehlerrate erreicht werden, d.h. tatsächliche Kantenpunkte sollten nicht zurückgewiesen werden und Punkte, die zu keiner Kante gehören, auch nicht als solche erkannt werden. Weiterhin wird verlangt, daß Kanten möglichst an ihrer wirklichen Position detektiert werden (*Lokalisation*). Um ein Verschmieren zu vermeiden, sollte auf eine Kante auch nur eine Antwort kommen. Numerische Kriterien wie z.B. schnelle Berechenbarkeit, Ganzzahl-Arithmetik und auch subjektive Punkte spielen bei den Anforderungen an Kantendetektoren zusätzlich eine wichtige Rolle.

Für die idealen Kantenprofile wurden oben schon Kantenpunkte und deren Lokalisation definiert. Bei den realen Kanten ist diese Definition nicht mehr so einfach. Eine Betrachtung der Kanten im Ortsfrequenzraum zeigt, daß es sich bei Kanten um hochfrequente Bildanteile handelt (siehe Kapitel 8). Diese müssen vom Kantendetektor berechnet werden. Leider ist auch Rauschen von hochfrequenter Natur, so daß Kantendetektoren normalerweise auch empfindlich gegenüber diesen Störungen sind. Die Schwierigkeit besteht also in der Unterscheidung der zufälligen Signalfluktuationen, die durch Rauschen verursacht werden, von den gesuchten Grauwertdiskontinuitäten, die mit realen Kanten korrespondieren.

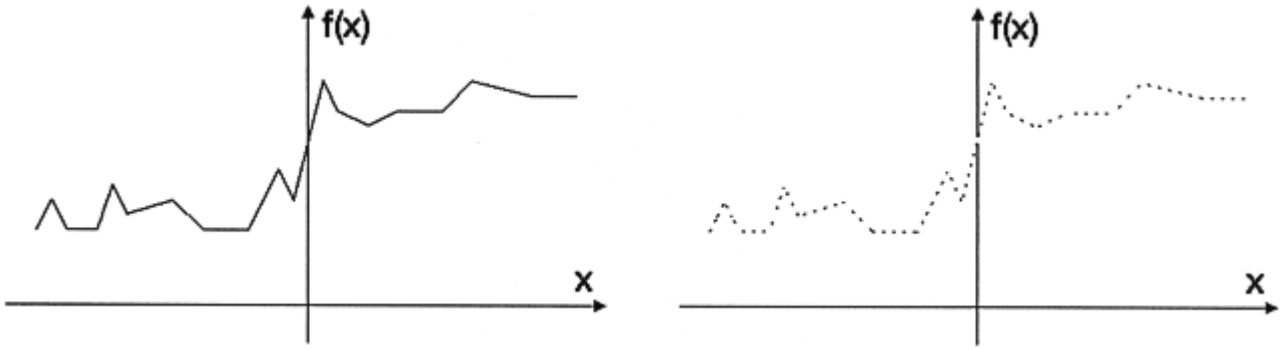
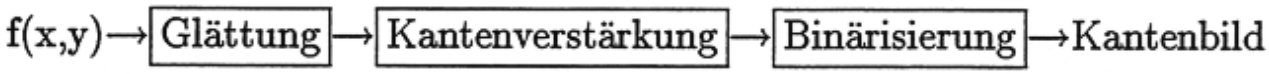


Abbildung 11.2: Eine reale Kante ist meist eine Kombination idealer Kantenprofile, überlagert mit Rauschen. Aufgrund der Diskretisierung und Quantisierung (rechts) geht noch weitere Information verloren.

Um Kanten korrekt zu ermitteln, muß daher vor der eigentlichen Kantendetektion eine Filterung durchgeführt werden, die Rauschen und somit auch feine Details unterdrückt. Das kann aber dazu führen, daß Kanten verschmieren und damit die Güte der Lokalisation verschlechtert wird. Die üblichen Kantendetektoren bestehen daher in der Regel aus einem Glättungs- und einem Kantenverstärkungsteil.



Durch die Glättung werden störende Feinheiten und vor allem Rauschen im Originalbild vermindert. Dies entspricht einer Tiefpaßfilterung.

Die darauffolgende Hochpaßfilterung verstärkt Diskontinuitäten, also mögliche Kanten. Die Stärke der Antwort liefert ein Maß für die Wahrscheinlichkeit des Vorliegens einer Kante.

In der letzten Stufe folgt durch ein Schwellwertverfahren eine Binärisierung. Durch den/die gewählte(n) Schwellwert(e) wird die endgültige Entscheidung über das Vorliegen einer Kante getroffen. Die Höhe der Schwelle richtet sich nach der Art und der "Stärke" der gesuchten Kante. Zur Verwendung eines möglichst universellen Schwellwertes hat es sich als günstig herausgestellt, Extremwerte zu unterdrücken und die anderen Werte auf einen normierten Bereich zu transformieren. Die Extremwerte können z.B. durch Berechnung des Histogramms ermittelt werden. Beispielsweise werden nur die unteren 95% – 98% der Ergebniswerte (Gradientenwerte) für die Transformation z.B. in den darstellbaren Grauwertbereich (zur Visualisierung der Daten) verwendet. Alle darüberliegenden Werte werden geklippt.

Die folgende Prozedur zeigt eine solche Unterdrückung von Extremwerten. Die Transformation auf den darstellbaren Grauwertbereich ist schon zuvor erfolgt.

```
void Skalier_Histo(int BildNr, int Prozent)
/* BildNr  = Nummer des zu bearbeitenden Bildes                  */
/* Prozent = Prozentsatz von Punkten mit hohem Wert, der ueber   */
/*           das kumulative Histogramm geklippt werden soll.     */
{
  int  s, z;                 /* Zeilen-/Spaltenindex             */
  long Histo[MAX_COLOR];     /* Histogramm                       */
  long GesamtPixel, Anzahl;  /* Gesamtanzahl Punkte              */
  float Faktor;              /* Skalierungsfaktor                */

  /* Initialisierung des Histogramms                             */
  for (s=0; s<MAX_COLOR; Histo[s++]=0);

  /* Haeufigkeit der einzelnen Grauwerte ermitteln               */
  for (z=0; z<Picture[BildNr].Zeilen; z++}
      for (s=0; s<Picture[BildNr].Spalten; s++)
          Histo[ Picture[BildNr].Bild[z][s] ]++;

  GesamtPixel = (long)Picture[BildNr].Zeilen *
                (long)Picture[BildNr].Spalten;

  /* Den oberen Prozentsatz an Bildpunkten ermitteln             */
  z=MAX_COLOR;
  Anzahl=Histo[--z];
  while ( ((Anzahl*100/GesamtPixel) < Prozent) && (z>2) )
        Anzahl += Histo[--z];

  /* Jetzt den Skalierungsfaktor fuer die Grauwerte berechnen    */
  Faktor = (float) (MAX_COLOR-1) / (float) (z);

  /* Neue LUT berechnen                                          */
  for (s=0; s<z;         s++) Histo[s] = (int)((float) s * Faktor);
  for (s=z; s<MAX_COLOR; s++) Histo[s] = WEISS;

  /* Zum Schluss eigentliche Umsetzung der Grauwerte vornehmen   */
  for (z=0; z<Picture[BildNr].Zeilen; z++)
      for (s=0; s<Picture[BildNr].Spalten; s++)
           Picture[BildNr].Bild[z][s] =
                              Histo[ Picture[BildNr].Bild[z][s] ];
}
```

11.2.1 Bewertung von Kantendetektoren

Wie schon erwähnt, ist das Ergebnis der Kantendetektoren abhängig von der Kantenart, dem Ausgangsbild, den Störungen usw. Je nach Einsatzgebiet liefert daher der eine oder der andere Operator ein besseres Ergebnis. Doch wie ist die Qualität der Operatoren zu bewerten?

Zum einen kann das Ergebnis visuell bewertet werden. Der Betrachter entscheidet dabei, ob die durch den Kantendetektor gefundenen Kanten den von ihm als Kante wahrgenommenen Bildteilen entsprechen.

Für die weitere Verarbeitung kann unter Umständen zur Segmentierung (siehe Kapitel 14) die Geschlossenheit von Kanten ein wichtiges Bewertungskriterium sein.

Eine andere Bewertungsmöglichkeit besteht darin, in Testbildern, deren (korrekte) Kantenpunkte bekannt sind, die durch den Operator gefundenen Kantenpunkte mit den richtigen Kantenpunkten zu vergleichen. Diese Untersuchung muß natürlich bei unterschiedlichen Stärken von Rauschen vorgenommen werden, um möglichst realistische Bedingungen zu erzeugen. Als sinnvolles quantitatives Maß eignet sich Pratt's *Figure-of-Merit* (FOM) [Pra78], [AP79]. Hierbei wird der gewichtete quadratische Abstand zwischen den tatsächlichen Kantenpunktpositionen und den detektierten gemessen. Als Gewichtungsfaktor α wird im allgemeinen $\alpha = 1/9$ gewählt.

$$\mathrm{FOM} = \frac{1}{\max(I_i, I_a)} \sum_{j=1}^{I_a} \frac{1}{1+\alpha(d(j))^2}$$

Dabei sind I_i und I_a die Anzahl der idealen bzw. aktuell detektierten Kantenpunkte und $d(j)$ der Abstand des j.ten detektierten Kantenpunktes von der tatsächlichen (nächstgelegenen) Kantenposition. Die FOM ist normalisiert, so daß immer gilt $0 < \mathrm{FOM} \leq 1$. FOM = 1 entspricht einer perfekten Überdeckung von idealer und tatsächlich detektierter Kante.

Mit Hilfe des Gewichtungsfaktors und der Normalisierung erfolgt eine ausgewogenere Gewichtung zwischen breiten, verschmierten Kanten in der korrekten Position und dünnen, aber gegenüber der richtigen Position verschobenen Kanten. Dies entspricht einer "natürlicheren" Wertung der einzelnen Fehler als bei dem üblichen quadratischen Abstandsmaß.

Eine Aussage über den Zusammenhang der Kantenpunkte wird mit der FOM nicht gemacht.

11.3 Einteilung der Kantendetektoren

Aufgrund der Vorgehensweise lassen sich die Kantendetektionsverfahren grob in zwei Klassen unterteilen; die der *parallelen Verfahren* und die der *sequentiellen Verfahren*. Die parallelen Verfahren stellen auch oft eine Vorstufe für die sequentiellen dar.

Bei den parallelen Verfahren wird lokal ein Eigenschaftsvektor bestimmt, der Angaben wie Kantenstärke, Kantenrichtung oder Maße für die Kantenform enthält. Da dieser Eigenschaftsvektor nur von der lokalen Bildfunktion abhängt, kann er parallel für alle anderen Bildpunkte berechnet werden. Je nach Verfahren werden die einzelnen Eigenschaften als Kriterium für das Vorliegen eines Kantenpunktes verwendet. Meist ist es nur die Kantenstärke, die in ein Schwellwertverfahren (Binärisierung) eingeht.

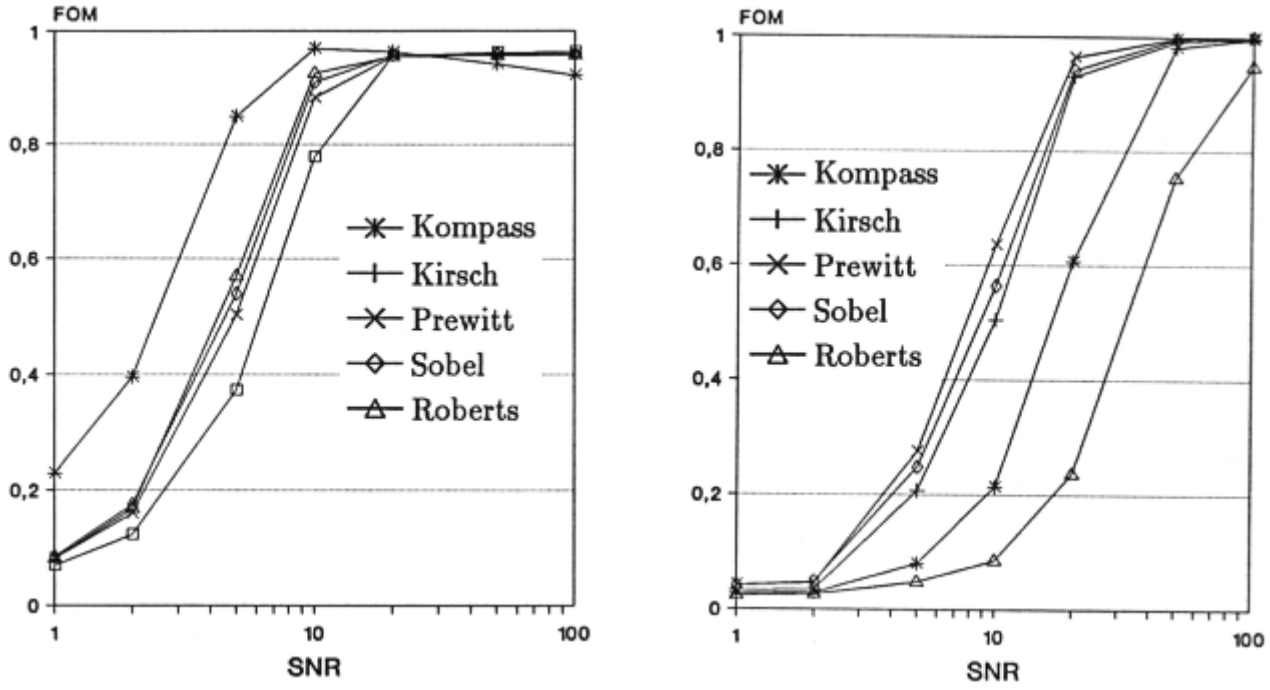


Abbildung 11.3: Die Ergebnisse der FOM-Berechnung verschiedener Operatoren bei unterschiedlichen Signal-zu-Rausch-Verhältnissen (SNR). Links das Ergebnis bei einer kreisförmigen Stufenkante. Rechts bei einer vertikalen Rampenkante (aus [Gru91]).

Die Klasse der parallelen Verfahren kann noch weiter unterteilt werden in

- **Einfache, lokale Operatoren**

  Diese stellen meist eine Approximation der ersten oder zweiten Ableitung der Bildfunktion zusammen mit einer Glättung dar (Roberts, Prewitt, Sobel; siehe Abschnitt 11.4.3).

- **Template-Matching**

  Bei diesen Kantendetektionsverfahren werden Musterkanten in Form von unterschiedlichen Filtermasken verwendet. Die Maske, die am besten mit der unterliegenden Bildfunktion übereinstimmt, repräsentiert die Form der Kante an dieser Stelle. Bekannte Template-Matching-Verfahren sind z.B. der Kompaß-Gradient und der Kirsch-Operator.

- **Optimale Operatoren**

  Wegen des großen Einzugsbereichs könnte man diese Operatoren auch als regionale Operatoren bezeichnen. Die Bezeichnung "optimal" basiert auf der mehr oder weniger mathematischen Herleitung, die sich ihrerseits auf Beobachtungen (z.B. beim menschlichen Sehsystem) und Modellvorstellungen gründet. Zu dieser Gruppe gehören z.B. der Marr-Hildreth-Operator und der Canny-Operator (siehe Abschnitt 11.8.2).

- **Parametrisierte Kantenmodelle**

  Die Masken sind bei diesen Verfahren nicht starr festgelegt, sondern können noch über Parameter variiert werden [Wah94]. Dadurch ist eine teilweise automatische Adaption an unterschiedliche Vorlagen möglich. Grundlage dazu ist die meist realistische Annahme, daß Gradientenbilder und intensitätsskalierte Bilder im Bereich von Objektkanten sehr stark korreliert sind.

- **Morphologische Operatoren**

  Bei dieser Art von Operatoren (Morphologie = Lehre von der Form) wird die Kenntnis über die Form der Objekte ausgenutzt (siehe Kapitel 14).

Bei den sequentiellen Verfahren hängt das Ergebnis des Akzeptanztests für einen möglichen Kantenpunkt von benachbarten Ergebnissen (lokal) oder sogar von weiter entfernten Ergebnissen (regional, global) ab. Es wird dabei a-priori-Wissen über die Struktur der gesuchten Kanten oder Vorwissen über den Bildinhalt ausgenutzt. In der Regel bauen die sequentiellen Verfahren auf den Ergebnissen der parallelen Verfahren auf. Sie werden daher auch gelegentlich als Verfahren zur *Kanten-Nachbearbeitung* bezeichnet.

Im den folgenden Abschnitten werden in erster Linie die wichtigsten parallelen Verfahren vorgestellt und anschließend die Möglichkeiten der Kanten-Nachbearbeitung erörtert.

11.4 Differenzoperator erster Ordnung

Wie schon erwähnt, können Ableitungsoperatoren für die Kantendetektion verwendet werden. Die erste Ableitung hat an einer Kante ein lokales Maximum, die zweite Ableitung einen Nulldurchgang. Im Zweidimensionalen werden die partiellen Ableitungen $\partial f(x,y)/\partial x$ und $\partial f(x,y)/\partial y$ berechnet. Die erste partielle Ableitung ist ein Maß für die Änderung der Grauwerte und somit für die Kantenstärke. Sie ist am stärksten senkrecht zur Ableitungsrichtung. Im kontinuierlichen Fall gilt:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{t \to 0} \frac{f(x+t,y) - f(x,y)}{t}$$

Bei digitalen Bildern ist $t = 1$ und die Ableitung berechnet sich nach der Formel

$$\Delta_x f(x, y) = f(x, y) - f(x\text{-}1, y)$$

$$\Delta_y f(x, y) = f(x, y) - f(x, y\text{-}1)$$

Diese Art der Schreibweise wird als *Rückwärtsgradient* bezeichnet (da der zurückliegende Bildpunkt mit einbezogen wird), im Gegensatz zum *Vorwärtsgradienten*

$$\Delta_x f(x, y) = f(x+1, y) - f(x, y)$$

und zum symmetrischen Gradienten [Jä89]

$$\Delta_x f(x, y) = (f(x+1, y) - f(x-1, y))/2$$

Viele Kantenoperatoren berechnen die Kanten mit Hilfe des *Gradienten*. Der Gradient einer partiell differenzierbaren Funktion zweier Variablen ist durch

$$(\partial f(x,y)/\partial x, \partial f(x,y)/\partial y)$$

definiert. Für den Betrag des Gradienten gilt dann

$$l = \sqrt{(\partial f(x,y)/\partial x)^2 + (\partial f(x,y)/\partial y)^2}$$

sowie für die Richtung

$$\Theta = \arctan\left(\frac{\partial f(x,y)}{\partial y} / \frac{\partial f(x,y)}{\partial x}\right)$$

Der Vektor mit der Richtung Θ und der Länge l wird auch *Gradienten-Vektor* genannt. Im diskreten Fall werden die Funktionen $\Delta_x f(x, y)$ und $\Delta_y f(x, y)$ berechnet und anschließend Betrag und Richtung des Gradienten nach der Formel

$$\sqrt{(\Delta_x f(x,y))^2 + (\Delta_y f(x,y))^2}$$

und $\Delta_x f(x, y)/\Delta_y f(x, y)$ ermittelt.

In der Regel ist der genaue Wert des Gradienten nicht von Interesse, sondern nur seine relative Größe. Der Gradient wird daher meist approximiert, um den Rechenaufwand zu minimieren. Gebräuchlich sind:

$$l_1 = \left|\frac{\partial f(x,y)}{\partial x}\right| + \left|\frac{\partial f(x,y)}{\partial y}\right|$$

$$l_2 = \max\left\{\left|\frac{\partial f(x,y)}{\partial x}\right|, \left|\frac{\partial f(x,y)}{\partial y}\right|\right\}$$

11.4.1 Form der Filtermasken

Die Filterfunktion wird als *Maske* geschrieben, d.h. als eine diskrete Funktion (auf demselben Gitter wie die Bildfunktion), die nur in einem kleinen Bereich ungleich Null ist. Die Form dieses Bereiches wird meist intuitiv festgelegt. Gebräuchlich sind dabei:

- rechteckige,
- kreuzförmige
- näherungsweise kreisförmige Masken.

Meist wird eine rechteckige/quadratische 3×3-Maske mit folgender Indizierung verwendet:

$$\Delta f(x,y)=\begin{pmatrix} f(x-1,y-1) & f(x,y-1) & f(x+1,y-1) \\ f(x-1,y) & f(x,y) & f(x+1,y) \\ f(x-1,y+1) & f(x,y+1) & f(x+1,y+1) \end{pmatrix}$$

Um ein Ausgangsbild in der gleichen Größe wie das Eingangsbild zu erhalten, muß der Definitionsbereich der Bildfunktion f(x, y) ausgedehnt werden, da sonst die Maske bei der Berechnung der Randpunkte den Definitionsbereich verläßt. Dies geschieht analog wie bei den Filtermasken zur Bildverbesserung (vergleiche Kapitel 10) durch Spiegelung des Bildinhaltes an den Bildrändern.

11.4.2 Ein einfacher Differenzoperator

Der *einfache Differenzoperator* ist die direkte Umsetzung des Rückwärtsgradienten in eine Filtermaske und hat folgende Gestalt:

$$\Delta_x f(x,y)=\begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \Delta_y f(x,y)=\begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Die Maske Δ_x spricht vor allem auf vertikale, Δ_y auf horizontale Kanten an. Die Berechnung ist sehr einfach, da nur eine 2×2-Matrix zu betrachten ist (die Erweiterung auf 3×3 ist nur zum besseren Vergleich mit den anderen Operatoren vorgenommen worden) und nur Additionen verwendet werden. Da aber eine Glättung fehlt, ist dieser Operator sehr rauschempfindlich. Das Gradientenbild ist um ein halbes Abtastintervall verschoben, da die eigentliche Stelle für das Ergebnis zwischen den Abtastpunkten liegt. Ebenso wie der Vorwärtsgradient plaziert der Rückwärtsgradient das Ergebnis auf Zwischengitterplätze. Weiterhin ist diese Art von Ableitungsoperatoren nicht isotrop, das bedeutet, daß die Filterantwort nicht in allen Richtungen gleich, sondern am stärksten senkrecht zur Ableitungsrichtung ist.

Um kleine Störungen nicht zu berücksichtigen und das Ergebnis genau auf den betrachteten Bildpunkt zu legen, kann man diesen Operator auf den symmetrischen Gradienten erweitern. Da jeweils nur die übernächste Zeile bzw. Spalte in die Differenzbildung mit einbezogen wird, gehen kleine Störungen benachbarter Zeilen bzw. Spalten nicht in das Ergebnis ein. Damit wird der Operator etwas unempfindlicher gegenüber kleinen Störungen. Dieser Operator ist ebenfalls nicht isotrop.

$$\Delta_x f(x,y)=\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad \Delta_y f(x,y)=\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Zur Erzeugung des binären Kantenbildes werden die berechneten Gradientenwerte mit einem Schwellwert verglichen (siehe Abschnitt 11.2 und Abschnitt 11.10.2).

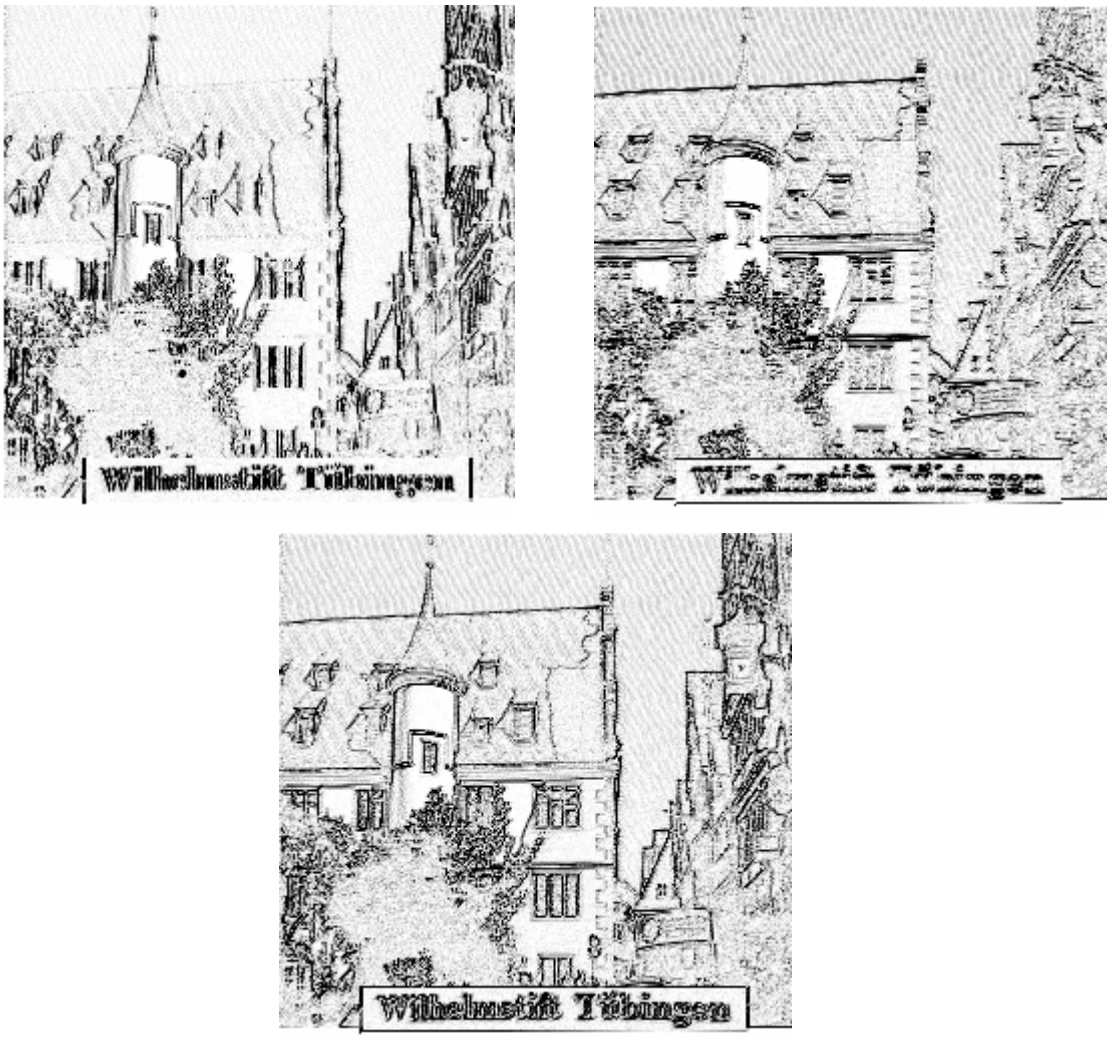


Abbildung 11.4: Differenzierung in x-Richtung (links) und in y-Richtung (rechts) und Darstellung des betragsmäßigen Ergebnisses (unten).

11.4.3 Der Roberts-Operator

Der *Roberts-Operator* berechnet die Differenzen in diagonaler Richtung [RK76]. Daher wird dieser Operator auch oft mit *Roberts-Cross* bezeichnet.

$$\Delta_x f(x,y) = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \Delta_y f(x,y) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Für den Roberts-Operator gelten dieselben Vor- und Nachteile wie für den einfachen Differenzoperator. Der einzige Unterschied ist, daß mit dem Roberts-Operator diagonal verlaufende Kanten besser detektiert werden. Dies kann in der Praxis gelegentlich von Vorteil sein.

Abbildung 11.5: Vergleich des Ergebnisses des symmetrischen Gradienten (links) in x- und y-Richtung mit dem Roberts-Operator (rechts). In beiden Bildern ist das Ergebnis des Gradienten auf den maximal darstellbaren Grauwertbereich normiert.

11.5 Differenzoperator mit einfacher Mittelwertbildung

Wie am Roberts-Operator deutlich wurde, ist die einfache Differenzbildung (erste Ableitung) in der Praxis zu anfällig gegenüber Störungen wie etwa Rauschen. Aus diesem Grunde werden zur Differenzbildung auch die Grauwerte weiterer Nachbarn verwendet und zusätzlich noch gemittelt. Dadurch wird eine Glättungswirkung senkrecht zur Richtung der Differenzbildung erzielt. Erkauft wird diese geringere Rauschanfälligkeit aber mit einer etwas verbreiterten Kante im ungestörten Fall.

11.5.1 Der Prewitt-Operator

Der *Prewitt-Operator* beinhaltet sowohl eine Glättung durch einfache Mittelwertbildung über eine 3-Punkte Nachbarschaft senkrecht zur Differenzierungsrichtung, als auch die Glättungswirkung des symmetrischen Gradienten [RK76], [BB82].

$$\Delta_x f(x,y) = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \qquad \Delta_y f(x,y) = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

```
#define SKALIERUNG 2.0  /* Skalierungsfaktor f. Betragsgradient  */
#define RAND       1    /* Zusaetzlicher Rand der Breite 1 Pixel */

void Prewitt(int BildNr1, int BildNr2)
/* BildNr1 = Originalbild                                        */
/* BildNr2 = Ergebnisbild; skalierter und geklippter Bereich     */
{
  long  z, s;             /* Variablen fuer Zeilen-/Spaltenindex */
  long  Zwerg1, Zwerg2;   /* Variablen fuer Zwischenergebnisse   */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Anwendung des Prewitt-Operators *");

  /* Erzeugung eines 1 Pixel breiten Randes zur einf. Berechnung */
  Erzeuge_Rand(BildNr1, RAND);
  /* Nun die eigentliche Faltung/Kantendetektion durchfuehren    */
  for (z=1; z<(Picture[BildNr1].Zeilen+1); z++}
      for (s=1; s<(Picture[BildNr1].Spalten+1); s++) {
          Zwerg1 = -Picture[BildNr1].Bild[z-1][s-1]
                   -Picture[BildNr1].Bild[z  ][s-1]
                   -Picture[BildNr1].Bild[z+1][s-1]
                   +Picture[BildNr1].Bild[z-1][s+1]
                   +Picture[BildNr1].Bild[z  ][s+1]
                   +Picture[BildNr1].Bild[z+1][s+1];
          Zwerg2 = -Picture[BildNr1].Bild[z-1][s-1]
                   -Picture[BildNr1].Bild[z-1][s  ]
                   -Picture[BildNr1].Bild[z-1][s+1]
                   +Picture[BildNr1].Bild[z+1][s-1]
                   +Picture[BildNr1].Bild[z+1][s  ]
                   +Picture[BildNr1].Bild[z+1][s+1];
          /* Betragsgradienten berechnen. Koennte auch           */
          /* approximiert werden, da genauer Zahlenwert hier     */
          /* nicht von Interesse.                                */
          Zwerg1 = (int)(sqrt((double)(Zwerg1*Zwerg1 +
                                       Zwerg2*Zwerg2))/SKALIERUNG);
          /* Gradient speichern und Extremwerte ggf. klippen     */
          Picture[BildNr2].Bild[z][s] =
                                    (Zwerg1>WEISS) ? WEISS : Zwerg1;
      }
  /* Stellt das Bild im Ursprungsformat ohne Rand wieder her     */
  Loesche_Rand(BildNr1, RAND);

  /* Endgueltige Skalierung des Gradientenbildes mit Klipping    */
  Skalier_Histo(BildNr2, 5);          /* obere 5 Prozent klippen */
}
```

11.5.2 Der Sobel-Operator

Ein sehr häufig verwendeter Kantenfilter ist der Sobel-Operator [Pra78]. Er enthält eine Glättung quer zur Differenzierungsrichtung mit einer (1 2 1) Binomial-Filtermaske, d.h. einer Filtermaske, die über die Binomial-Verteilung berechnet wurde. Die Binomial-Verteilung wird als diskrete Ap-

proximation der Gauß-Verteilung verwendet. Dabei steigt die Approximationsgüte mit der Größe der Maske, d.h. mit der Ordnung des Binoms. Mit dieser speziellen Gewichtung soll die Entstehung von Artefakten bei der Filterung verhindert und die Auswirkung im Ortsfrequenzraum regional begrenzt bleiben (die Gauß-Funktion ist neben der Dirac-Funktion die einzige Funktion, die im Orts- und im Ortsfrequenzraum dieselbe Form hat (vergleiche Kapitel 8, Abschnitte 11.8.1 und 11.8.2).

$$\Delta_x f(x,y)=\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix} \qquad \Delta_y f(x,y)=\begin{pmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{pmatrix}$$

Für die diagonalen Richtungen lassen sich ebenso entsprechende Masken definieren. Die Übertragungsfunktionen unterscheiden sich aber von denen der achsenparallelen Varianten [Jä89].

$$\Delta_x f(x,y)=\begin{pmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{pmatrix} \qquad \Delta_y f(x,y)=\begin{pmatrix}0 & -1 & -2\\ 1 & 0 & -1\\ 2 & 1 & 0\end{pmatrix}$$

Zur einfacheren Berechnung des Sobel-Operators können die Masken zerlegt werden, so daß nur noch Additionen notwendig sind [Hed88].

$$\Delta_x f(x,y)=\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix}=\begin{pmatrix}-1 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 1\end{pmatrix}+\begin{pmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{pmatrix}$$

Die Zerlegung der anderen Maske erfolgt entsprechend.

11.6 Differenzoperator zweiter Ordnung

Bei der Verwendung der zweiten Ableitung zur Lokalisierung von Kanten befindet sich eine mögliche Kante im Nulldurchgang der zweiten Ableitung. Diese zweite Ableitung wird im diskreten Fall wieder approximiert und kann auf die Approximation der ersten Ableitung (Vorwärts- und Rückwärtsgradient) zurückgeführt werden.

11.6.1 Der Laplace-Operator

Wie man bei den Masken für den Sobel-Operator gesehen hat, werden horizontale und vertikale Kanten bevorzugt detektiert. Soll der Operator möglichst richtungsunabhängig (isotrop) sein, so muß eine punktsymmetrische Maske verwendet werden. Dies ist bei Differenzoperatoren zweiter Ordnung möglich. Am bekanntesten ist hierbei der *Laplace-Operator*, der im kontinuierlichen Fall als

$$\nabla^2 f(x,y)=\frac{\partial^2 f(x,y)}{\partial x^2}+\frac{\partial^2 f(x,y)}{\partial y^2}$$

definiert ist. Im diskreten Fall ($t = 1$) wird die zweite Ableitung durch

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\partial (f(x+1,y) - f(x,y))}{\partial x}$$

$$\approx (f(x+1, y) - f(x, y)) - (f(x, y) - f(x-1, y))$$

$$= f(x-1,y) - 2f(x, y) + f(x+1, y)$$

angenähert. Analog errechnet sich die partielle Ableitung nach y. Als grobe Annäherung für den kontinuierlichen Fall lautet die Maske dann

$$\Delta_{xy} f(x,y) = \begin{pmatrix} 1 & -2 & 1 \end{pmatrix} + \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} (-1)$$

Die eigentliche Kantendetektion mit Hilfe dieses Operators kann auf zwei unterschiedliche Arten erfolgen. Zum einen können in der Ergebnismatrix nach der Faltung die Nulldurchgänge gesucht werden, d.h. die Stellen, an denen das Vorzeichen wechselt. Zum anderen kann ein Schwellwertverfahren auf den Betrag der Ergebnisse angewendet werden. Wegen der Einfachheit wird letzteres meist verwendet. Der Operator wird in diesem Fall dann oft als *Absoluter-Pseudo-Laplace-Operator* bezeichnet [Wah84].

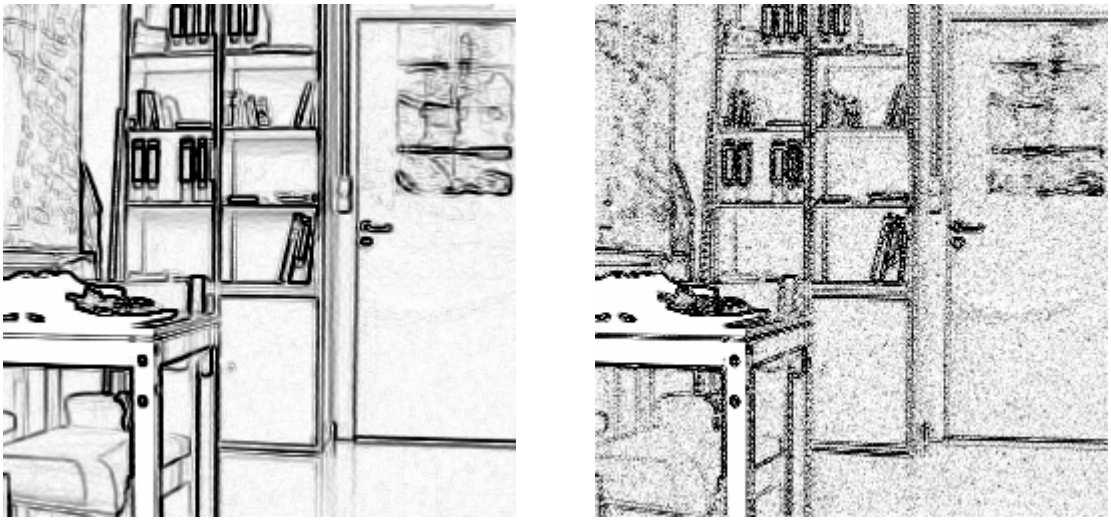


Abbildung 11.6: Ergebnis des Prewitt-Operators (links) und des Laplace-Operators (rechts), angewandt auf dieselbe Bildvorlage. Das Kantenbild wurde mit Hilfe eines Schwellwertverfahrens aus dem Betrags-Ergebnisbild erzeugt. Deutlich ist die wesentlich höhere Rauschempfindlichkeit des Laplace-Operators zu erkennen.

Der Laplace-Operator ist anfälliger gegenüber Bildstörungen und ohne eine vorherige Tiefpaßfilterung zumindest bei Bildern mit mittlerem bis niedrigem Signal-zu-Rausch-Verhältnis relativ

nutzlos. Die hohe Empfindlichkeit gegenüber Störungen wird besonders deutlich, wenn man das Verhalten der obigen Maske gegenüber einzelnen Punkten und Linien im Betrags-Ergebnisbild betrachtet. Punkte werden dabei viermal so stark wie eine Kante, ein Linienende dreimal und eine 1-Pixel breite Linie doppelt so stark detektiert, was normalerweise die Ergebnisse verfälscht. Diese Effekte können aber bei der Mustererkennung unter Umständen positiv ausgenutzt werden. Der Laplace-Operator liefert, angewendet auf eine beliebig geneigte Ebene den Wert 0, obwohl die Gradienten dieser Fläche nicht notwendig den Wert 0 haben.

Die hier vorgestellte gebräuchlichste Approximation des Laplace-Operators ist nur für niederfrequente Bildanteile isotrop [Jä89]. Die Isotropie soll daher folgende Maske verbessern:

$$\Delta_{xy} f(x,y) = \begin{pmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{pmatrix}$$

Eine Reihe weiterer 3×3-Approximationen sind noch bekannt, die aber alle ähnliche Eigenschaften besitzen [Pra78], [Hor86].

$$\Delta_{xy} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \qquad \Delta_{xy} = \begin{pmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{pmatrix} \qquad \Delta_{xy} = \begin{pmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{pmatrix}$$

11.7 Template-Matching

Bei den *Template-Matching-Operatoren* werden die Filtermasken T_i als Sätze von Schablonen (Templates) verwendet, die dann eine diskrete Approximation von Modellkanten in verschiedenen Orientierungen darstellen. Schon der vorgestellte Prewitt- oder Sobel-Operator kann in seiner Form als Template für eine mögliche Kante/Kantenrichtung aufgefaßt werden.

Es wurden verschiedene Verfahren entwickelt, die mit Hilfe eines festen Satzes von Modellkanten, oder mit variabler, vom Bildinhalt selbst abhängiger Schablonenfunktionen eine Kantendetektion durchführen.

Die Richtung einer Kante ergibt sich dann aus der Orientierung der Maske, die die beste Übereinstimmung ergibt. Über die Faltungsoperation wird für jede der Schablonen ein Maß für die Übereinstimmung zwischen der Filtermaske (und damit der zugehörigen Modellkante) und der darunterliegenden Bildfunktion bestimmt. Diejenige Schablone, die an einem Punkt das betragsmäßig größte Filterergebnis F_{out} liefert, bestimmt zum einen die Ergebnishöhe in diesem Punkt und zum anderen die Richtung der potentiellen Kante. Diese kann an der Orientierung der Schablone abgelesen werden.

$$F_{out}(x, y) = \max_j \left| \sum_{k=-m}^{m} \sum_{l=-m}^{m} I_{in}(x-k, y-l) T_j(k,l) \right|$$

Dabei bezeichnet F_{out} das Filterergebnis, I_{in} die Bildfunktion und $(2m+1)$ die Größe der Filtermasken T_j. Über den Index j ist die Filtermaske und damit gleichzeitig die Orientierung festgelegt.

Exemplarisch sei hier der Kompaß-Gradient für 8 Richtungen angegeben:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Nord NO Ost SO

$$\begin{pmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{pmatrix}$$

Süd SW West NW

Weiterhin ist der Kirsch-Operator mit seinen unterschiedlichen Orientierungen eine gängige Maske beim Template-Matching.

$$\begin{pmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{pmatrix}$$

Insgesamt kann jede Filtermaske durch geeignete Wahl der Koeffizienten als Template verwendet werden. In [NB80] wurde der Prewitt-Operator auf eine 5×5-Maske erweitert und die Koeffizienten wurden so berechnet, daß 6 Filtermasken mit den Orientierungen 0, 30, 60, 90, 120 und 150 Grad erzeugt wurden. Wegen der Symmetrie kann damit der gesamte Bereich von 0 bis 360 Grad in Schritten von 30 Grad abgedeckt werden. Damit als Maskenkoeffizienten Ganzzahlen verwendet werden können, wurde die Maske mit dem Faktor 100 skaliert. Als Beispiel resultieren daraus folgende Masken (alle anderen analog):

$$\begin{pmatrix} -100 & -100 & 0 & 100 & 100 \\ -100 & -100 & 0 & 100 & 100 \\ -100 & -100 & 0 & 100 & 100 \\ -100 & -100 & 0 & 100 & 100 \\ -100 & -100 & 0 & 100 & 100 \end{pmatrix} \quad \begin{pmatrix} -100 & 32 & 100 & 100 & 100 \\ -100 & -78 & 92 & 100 & 100 \\ -100 & -100 & 0 & 100 & 100 \\ -100 & -100 & -92 & 78 & 100 \\ -100 & -100 & -100 & -32 & 100 \end{pmatrix}$$

0 Grad 30 Grad

Untersuchungen mit Hilfe der FOM (siehe Abbildung 11.3) zeigen, daß die Qualität der mit den Template-Matching-Operatoren ermittelten Kanten mit den Ergebnissen des Prewitt- oder Sobel-Operators vergleichbar ist. Als einziger Vorteil bestimmt die Verwendung der Schablonen gleichzeitig die lokale Kantenrichtung. Die Genauigkeit hängt von der Anzahl der verwendeten Masken (Orientierungen) und deren Größe ab [Wah84].

11.8 "Optimale" Operatoren

Alle bisher vorgestellten einfachen Differenzoperatoren benutzen lediglich die offensichtlichen Eigenschaften von Kanten, nämlich die hohen Werte der Ableitungen am Ort der Kante bzw. die Homogenität der Gebiete zu beiden Seiten der Kante. Ein bestimmtes Kantenmodell, d.h. quantitative Vorstellungen über das zu erwartende Ergebnis lagen diesen Operatoren nicht zugrunde.

Bei den *optimalen Operatoren* werden nun detaillierte Vorstellungen über die zu suchenden Bildeigenschaften wie Kantenmodell, Maße für die Genauigkeit und Fehler bei der Detektion verwendet. Aufgrund unterschiedlicher Vorstellungen und Schwerpunkte lassen sich damit auch die verschiedensten "optimalen" Operatoren ableiten. Es ist aber klar, daß es sich immer nur um eine Optimalität im Rahmen der Definition der Kante handelt (bedingte Optimalität).

11.8.1 Der Marr-Hildreth-Operator

Der Marr-Hildreth-Operator [MH80], [Hil83] ist ein Bandpaßfilter, also eine Kombination eines Tiefpaßfilters mit einem Hochpaßfilter. Die Hintergrundtheorie basiert auf neurophysiologischen Untersuchungen, bei denen ein Modell der ersten Bildvorverarbeitungsstufen im menschlichen Sehen entwickelt werden sollte.

Unter der Annahme, daß es nicht möglich ist, einen für alle Ortsfequenzbereiche optimalen Kantenoperator zu finden, forderten Marr und Hildreth, daß ein optimaler Operator eine begrenzte Bandbreite haben muß. Diese sollte über entsprechende Parameter einstellbar sein. Aus der Beobachtung, daß Intensitätsänderungen überwiegend örtlich begrenzt sind, wurde die zweite Anforderung der örtlichen Lokalisierung der Kanten im Ausgangsbild des Operators abgeleitet. Der optimale Operator mußte daher die beiden gegensätzlichen Anforderungen der begrenzten Bandbreite im Ortsfrequenzraum und dem begrenzten Einzugsbereich im Ortsraum minimieren.

Diese gegensätzlichen Bedingungen sind schon in der klassischen Unschärferelation der Fourier-Transformation enthalten:

Genaue Lokalisation in einem Raum, sei es auch nur durch steile Kanten,
führt zu einer Verschmierung über weite Bereiche im reziproken Raum.

Bekanntermaßen ist die Gauß-Funktion (Glockenkurve) neben der Dirac-Funktion die einzige Funktion, die im Orts- und im Ortsfrequenzraum ihre Form beibehält und somit zur Minimierung der Forderung verwendet werden kann.

In Anlehnung an die klassische Vorstellung eines Kantenoperators, bestehend aus Glättungsfilter und Ableitungsoperator, wird daher die Gauß-Funktion zur Glättung verwendet.

Als Ableitungsoperator wird aus Effizienzgründen, aber auch gestützt auf die physiologischen Erkenntnisse, der Laplace-Operator als einfacher linearer und rotationssymmetrischer Operator verwendet.

Der erste Schritt beim Marr-Hildreth-Filter ist die Glättung des Ausgangsbildes durch

$$f_{glatt}(x, y) = G_\sigma(x, y) * f(x, y)$$

mit $G_\sigma(x, y)$ als Gauß-Filter mit der Standardabweichung σ und * als Faltungsoperator, wobei gilt

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2}\exp\left[-\left(\frac{x^2+y^2}{2\sigma^2}\right)\right]$$

so daß sich insgesamt bei einer Filtergröße von $(2r+1)\times(2r+1)$

$$f_{glatt}(x, y) = \frac{1}{2\pi\sigma^2}\sum_{i=-r}^{+r}\sum_{j=-r}^{+r}\exp\left[-\left(\frac{i^2+j^2}{2\sigma^2}\right)\right] f(x+i, y+j)$$

ergibt.

Sigma = 1.5
Radius = 6

0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-1	-2	-3	-4	-3	-2	-1	0	0	0
0	0	-1	-4	-8	-13	-15	-13	-8	-4	-1	0	0
0	-1	-4	-11	-21	-27	-27	-27	-21	-11	-4	-1	0
0	-2	-8	-21	-26	-7	9	-7	-26	-21	-8	-2	0
0	-3	-13	-27	-7	71	125	71	-7	-27	-13	-3	0
0	-4	-15	-27	9	125	200	125	9	-27	-15	-4	0
0	-3	-13	-27	-7	71	125	71	-7	-27	-13	-3	0
0	-2	-8	-21	-26	-7	9	-7	-26	-21	-8	-2	0
0	-1	-4	-11	-21	-27	-27	-27	-21	-11	-4	-1	0
0	0	-1	-4	-8	-13	-15	-13	-8	-4	-1	0	0
0	0	0	-1	-2	-3	-4	-3	-2	-1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

Sigma = 2.5
Radius = 6

-3	-6	-11	-16	-20	-23	-24	-23	-20	-16	-11	-6	-3
-6	-12	-19	-25	-29	-31	-31	-31	-29	-25	-19	-12	-6
-11	-19	-27	-31	-29	-24	-21	-24	-29	-31	-27	-19	-11
-16	-25	-31	-26	-8	13	22	13	-8	-26	-31	-25	-16
-20	-29	-29	-8	33	76	95	76	33	-8	-29	-29	-20
-23	-31	-24	13	76	141	169	141	76	13	-24	-31	-23
-24	-31	-21	22	95	169	200	169	95	22	-21	-31	-24
-23	-31	-24	13	76	141	169	141	76	13	-24	-31	-23
-20	-29	-29	-8	33	76	95	76	33	-8	-29	-29	-20
-16	-25	-31	-26	-8	13	22	13	-8	-26	-31	-25	-16
-11	-19	-27	-31	-29	-24	-21	-24	-29	-31	-27	-19	-11
-6	-12	-19	-25	-29	-31	-31	-31	-29	-25	-19	-12	-6
-3	-6	-11	-16	-20	-23	-24	-23	-20	-16	-11	-6	-3

Abbildung 11.7: Eine Filtermaske für den Marr-Hildreth-Operator (links σ = 1.5, rechts σ = 2.5). Die Einträge am Rand der linken Maske sind fast Null, so daß keine nennenswerten Fehler entstehen. In der rechten Maske sind die Randwerte aber noch relativ groß, so daß hier sichtbare Fehler entstehen können. Für das gewählte σ ist die rechte Maske zu klein. Durch die Multiplikation mit einem geeigneten Faktor (=200) wurden alle Einträge in den Ganzzahlbereich transformiert.

Nach der Glättung wird im zweiten Schritt dann mit dem klassischen Laplace-Operator eine Anhebung der hohen Frequenzen durchgeführt.

$$f_{ausgang}(x, y) = \nabla^2 f_{glatt}(x, y)$$

Da beide Filter linear und ortsunabhängig sind, können sie zu einem Filter, dem sogenannten Marr-Hildreth-Operator (*Laplacian-of-Gaussian, LoG*), kombiniert werden. Der Operator wird auch oft wegen seiner charakteristischen Form "Mexikanischer Hut" genannt.

$$f_{ausgang}(x, y) = \nabla^2 (G_\sigma(x, y) * f(x, y)) = (\nabla^2 G_\sigma(x, y)) * f(x, y)$$

$$\nabla^2 G_\sigma(x, y) = \frac{1}{\pi\sigma^4}\left(\frac{(x^2+y^2)}{2\sigma^2}-1\right)\left(\exp\left[-\left(\frac{(x^2+y^2)}{2\sigma^2}\right)\right]\right)$$

Da es insgesamt nur auf die Detektion von Nullstellen ankommt, kann der Vorfaktor $1/\pi\sigma^4$ durch einen beliebigen, für die Berechnung günstigen Skalierungsfaktor ersetzt werden. Die Nulldurchgangssuche erfolgt lokal innerhalb einer 3×3-Umgebung. Dadurch, daß die Nullstelle in der Regel nicht exakt auf das diskrete Raster fällt, wird die Position zu dem nächstgelegenen Bildpunkt gerundet. So entsteht eine Verschiebung der Kante.

Als Parameter wird nur die Standardabweichung σ benötigt, woraus die Filtergröße zwischen $3w$ und $4w$ mit $w = 2\sqrt{2\sigma}$ für die innere positive Filterregion gewählt werden kann (vergleiche 11.7).

Untersuchungen [SB89] haben gezeigt, daß eine Filtergröße von $4w$ ausreichend ist, ein $3w$ großer Filter jedoch schon zu sichtbaren Fehlern führen kann. Dies führt bei üblichen Werten von σ im Intervall [2, 10] zu relativ großen Filtermasken (Weite 10-100). Der LoG zählt daher auch zu den regionalen Operatoren. Die detektierten Kanten sind aufgrund der Nulldurchgangssuche immer 1-Pixel breit (eine Skelettierung ist nicht mehr notwendig; vergleiche Kapitel 12) und bilden immer geschlossene oder am Rand des Bildes endende Kurven, wie in [TP86] bewiesen wurde.

Eine Verschiebung der detektierten Kante gegenüber der tatsächlichen Kantenposition kann entstehen, wenn mehrere Kanten innerhalb des zentralen Bereichs des LoG-Operators (w) liegen, was natürlich abhängig von σ [Ber84], [LB86] und von der Größe der Filtermaske ist.

In verrauschten Bildern können prinzipiell keine Kreuzungspunkte mit einer ungeraden Anzahl sich treffender Kanten detektiert werden. Das Ergebnis an diesen Stellen ist instabil, d.h. extrem rauschabhängig (siehe Abbildung 11.9). Diejenigen Kantenpunkte, die fälschlicherweise nicht als mit dem Kreuzungspunkt verbunden detektiert werden, bilden andere geschlossene Kanten, die offensichtlich oftmals über weite Strecken falsch sind [MC*89].

Gänzlich fehlt diesem Operator ein Maß für die Stärke einer Kante. Leichte Variationen in einer Fläche mit ansonsten konstantem Grauwert werden genauso als Kanten detektiert wie ideale steile Kanten (vergleiche Ergebnisse in Abbildung 11.8). Versuche, über die erste Ableitung eine entsprechende Schwelle anzulegen, funktionieren zwar, führen aber zu Lücken in den ansonsten geschlossenen Kanten.

Wie schon erwähnt, scheint im menschlichen Gehirn der größte Teil der für einen solchen Filter nötigen "Hardware" vorhanden zu sein. Schon Ernst Mach stellte 1865 fest, daß bei der visuellen Wahrnehmung die räumlichen Schwankungen in der Lichtintensität offenbar verstärkt werden (Mach-Band-Effekt). Diese Verstärkung vermutete er in einem neuronalen Mechanismus. Mitte dieses Jahrhunderts konnte dann gezeigt werden, daß die Netzhaut etwas ganz ähnliches tut, wie der

oben beschriebene Zentrum-Umkreis-Filter. Die Differenz der Filterantworten zweier Gauß-Filter mit unterschiedlicher Standardabweichung (*Difference-of-Gaussians, DoG*) beschreiben dieses Verhalten sehr gut. Im Anhang von [MH80] wird gezeigt, daß der Marr-Hildreth-Operator eine Approximation des DoG darstellt.

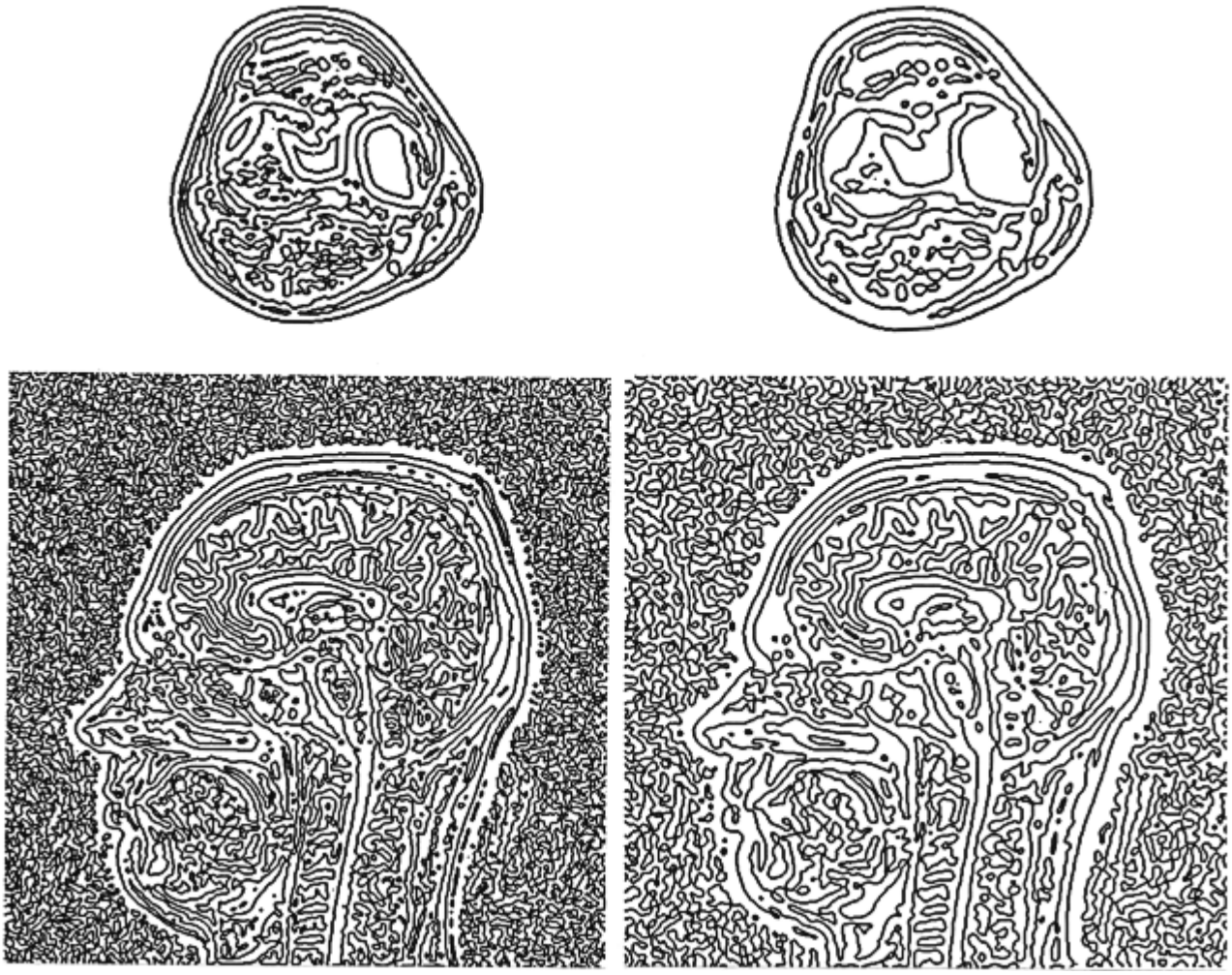


Abbildung 11.8: Die Nulldurchgänge im Filterergebnis des LoG-Operators bei verschieden großer Standardabweichung. Links: $\sigma = 2.0$. Rechts: $\sigma = 3.0$. Oben: CT-Aufnahme aus dem Bereich des Kniegelenks. Unten: MRT-Aufnahme des Kopfes. Deutlich ist bei den unteren Bildern die Rauschempfindlichkeit zu erkennen.

Jede der Nervenfasern, die die Signale zum Gehirn weiterleiten, ist mit einer Gruppe von Photorezeptoren verbunden. Bei bestimmten Sehnervenzellen ist das rezeptive Feld in Zentrum und Umkreis unterteilt. Die Helligkeit im Zentrum des rezeptiven Feldes erregt die Sehnervenzelle, Helligkeit in einem umgebenden Ring hemmt deren Erregung, also genau so, wie es bei der Laplace-Ableitung einer Gauß-Funktion der Fall ist. Man sagt auch oft, das rezeptive Feld hat ein AN-Zentrum und einen AUS-Umkreis. Bei anderen Sehnervenzellen ist es genau umgekehrt. Dies ist notwendig, da Nerven keine Signale mit "negativem"-Vorzeichen weiterleiten können.

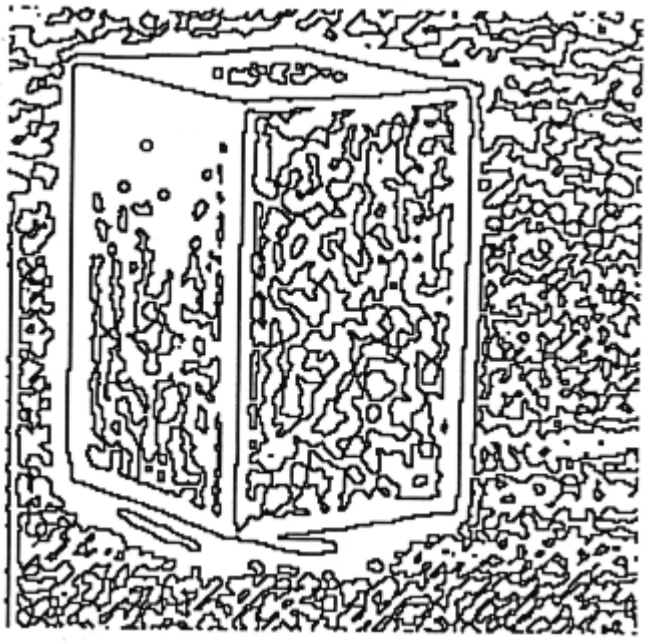


Abbildung 11.9: An Stellen, an denen sich eine ungerade Anzahl von Kanten trifft, entstehen beim Marr-Hildreth-Operator Fehler. In leicht verrauschten Gebieten werden Pseudokanten detektiert

```
#define FAKTOR 200      /* Faktor zur Transform. in Ganzzahlber. */
#define SCHWELLE 5      /* "Mindeststaerke" der Kante            */

void Marr_Hildreth(int BildNr1, int BildNr2)
/* BildNr1 = Originalbild                                       */
/* BildNr2 = Ergebnisbild; binaeres Kantenbild                  */
{
  int Matrix_MH[51][51];  /* Matrix fuer Marr-Hildreth Operator  */
  float Sigma;            /* Standardabweichung                  */
  float Sigma_Quadrat;    /* Sigma*Sigma                         */
  int Radius;             /* Halbe Groesse der Filtermaske       */
  int z, s, x, y;         /* Variablen fuer Zeilen-/Spaltenindex */
  float Zwerg;            /* Variable fuer Zwischenergebnisse    */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Anwendung des Marr-Hildreth-Operators *");

  /* Zuerst Filter-Matrix (=Filterkern) berechnen. Dazu gibt der */
  /* Benutzer die Werte fuer Sigma und den Filterradius ein.     */
  Sigma = Read_Float("Sigma (1.0 - ...)    ", 1.0, '?' );
  /* Filterkern = 4w mit w=2*sqrt(Sigma); Radius=2w              */
  do
     Radius=Read_Int("Reichweite (max. 25)? ",
                                        (int)4*sqrt(Sigma), ':' );
  while ((Radius > 25) || (Radius < 1));
```

```
  /* Die Filtermaske ist nun mit den Werten zu initialisieren    */
  Sigma_Quadrat=Sigma*Sigma;
  for (y= -Radius; y<=Radius; y++)
      for (x= -Radius; x<=Radius; x++) {
          Abstand = (float)(x*x + y*y);
          Zwerg   = FAKTOR * (Abstand/(2.0*Sigma_Quadrat) - 1) *
                             exp(-Abstand/(2.0*Sigma_Quadrat));
          /* Abbildung in den Ganzzahlenbereich und Rundung      */
          Matrix_MH[y+Radius][x+Radius]=
                   (Zwerg>0.0) ? (int)(Zwerg+0.5):(int)(Zwerg-0.5);
      }
  /* Erzeugung eines "Radius"-Pixel breiten Randes zur einf. Ber.*/
  Erzeuge_Rand(BildNr1, Radius);

  /* Nun die eigentliche Faltung der Maske mit dem Bildinhalt    */
  for (z=Radius; z<(Radius+Picture[BildNr1].Zeilen); z++)
      for (s=Radius; s<(Radius+Picture[BildNr1].Spalten); s++) {
          Zwerg = 0.0;
          for (y= -Radius; y<=Radius; y++)
              for (x= -Radius; x<=Radius; x++)
                  Zwerg += Matrix_MH[y+Radius][x+Radius] *
                           Picture[BildNr1].Bild[z+y][s+x];

          /* Ergebnis auf die normalen Grauwerte abbilden und ...*/
          /* ... wenn notwendig klippen.                         */
          Picture[BildNr2].Bild[z-Radius][s-Radius] =
           (Zwerg<SCHWARZ) ? SCHWARZ:((Zwerg>WEISS) ? WEISS:Zwerg);
      }
  Picture[BildNr2].Zeilen = Picture[BildNr1].Zeilen - 2*Radius;
  Picture[BildNr2].Spalten= Picture[BildNr1].Spalten- 2*Radius;

  /* Zum Schluss noch die Nulldurchgangssuche vornehmen und      */
  /* dabei gleich ein Binaerbild erzeugen.                       */
  for (z=0; z<(Picture[BildNr2].Zeilen-1); z++) {
      for (s=0; s<(Picture[BildNr2].Spalten-1); s++) {
          if ((Picture[BildNr2].Bild[z][s]==0) &&
              (Picture[BildNr2].Bild[z][s+1]>=SCHWELLE) ||
              (Picture[BildNr2].Bild[z][s]>=SCHWELLE) &&
              (Picture[BildNr2].Bild[z][s+1]==0)        ||
              (Picture[BildNr2].Bild[z][s]==0) &&
              (Picture[BildNr2].Bild[z+1][s]>=SCHWELLE) ||
              (Picture[BildNr2].Bild[z][s]>=SCHWELLE) &&
              (Picture[BildNr2].Bild[z+1][s]==0) )
             Picture[BildNr2].Bild[z][s]=WEISS;   /* Kantenpixel */
          else
             Picture[BildNr2].Bild[z][s]=SCHWARZ; /* Hintergrund */
      }
      Picture[BildNr2].Bild[z][s]=SCHWARZ;  /* rechter Rand noch */
  }
}
```

11.8.2 Der Canny-Operator

Canny geht bei der Herleitung seiner "optimalen" Operatoren streng mathematisch vor [Can83], [Can86]. Nach der Aufstellung einiger grundsätzlicher Annahmen und Optimierungs-Kriterien (geringe Fehlerrate, gute Lokalisation und nur eine Filterantwort auf eine einzige Kante) wird durch Anwendung von Variationsmethoden und ihrer numerischen Lösung die Operator-Funktion gefunden.

Die so für eine Stufenkante berechnete Filterfunktion approximiert Canny mit Hilfe der ersten Ableitung der Gauß-Funktion. Während sich eine Kante in einer Dimension allein durch eine Koordinate beschreiben läßt, besitzt sie in zwei Dimensionen auch eine Orientierung. Unter Orientierung versteht man dabei die Richtung der Tangente an die Kontur, die durch die einzelnen Kantenelemente beschrieben wird. Diese Orientierung läßt sich mit Hilfe des Gradienten der geglätteten Bildfunktion *f(x, y)* bestimmen, da der Gradient gerade senkrecht zur Kantenrichtung steht, also in Richtung der Normalen $\vec{n}$ zur Kantenrichtung. Im diskreten Fall gilt dies allerdings nur näherungsweise.

Zur Kantendetektion mit dem Canny-Operator wird zuerst die Bildfunktion *f(x, y)* mit den Richtungsableitungen der Gauß-Funktion in x- und y-Richtung gefaltet:

$$D_x(x, y) = \frac{\partial}{\partial x}(G_\sigma(x,y) * f(x,y)) = \frac{\partial G_\sigma(x,y)}{\partial x} * f(x,y)$$

$$D_y(x, y) = \frac{\partial}{\partial y}(G_\sigma(x,y) * f(x,y)) = \frac{\partial G_\sigma(x,y)}{\partial y} * f(x,y)$$

Dabei ist

$$G_\sigma(x\,y) = \exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right)$$

bis auf einen Skalierungsfaktor die Gauß-Funktion mit der Standardabweichung σ und dem Faltungsoperator *. Für die Normierung der Gauß-Funktion verwendet man im allgemeinen den Skalierungsfaktor $1/(\sqrt{2\pi}\ \sigma)^2$.

Damit berechnet sich die erste Ableitung der Gauß-Funktion in x-Richtung als:

$$\frac{\partial G_\sigma(x,y)}{\partial x} = \frac{-x}{\sigma^2}\exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right)$$

Wie sich leicht nachweisen läßt, wird eine Normierung der ersten Richtungsableitung der Gauß-Funktion durch einen Skalierungsfaktor von $\sqrt{2\pi}\ \sigma$ erzielt [Kor88], [WS90]. Die Normierung der Richtungsableitungen ist vor allem bei der Bearbeitung der Bilder in mehreren Auflösungsstufen, d.h. mit verschiedenen Standardabweichungen (*Multiscale-Ansatz, Multiscale Image Processing*; Filterung und Kantendetektion in einer Bildvorlage bei unterschiedlichen Auflösungen) wichtig. In

diesem Zusammenhang sei erwähnt, daß bei unterschiedlichen Auflösungsstufen durch die Gauß-Funktion keine neuen Kanten entstehen.

Der Gradient $\vec{D}$ der Gauß-geglätteten Bildfunktion an der Stelle *(x, y)* lautet:

$$\vec{D}(x,y)=D_x(x,y)\vec{e}_x+D_y(x,y)\vec{e}_y$$

Der Gradient $\vec{D}$ zeigt immer in Richtung des stärksten Anstieges (hier der Gauß-geglätteten Bildfunktion) und steht damit immer senkrecht zur Kantenrichtung am untersuchten Ort *(x, y)*. Die Richtung des Gradienten wird durch die Komponenten D_x und D_y festgelegt.

Der Betrag des Gradienten $\vec{D}$

$$D(x, y) = \left|\nabla(G_\sigma(x,y) * f(x,y)\right| = \sqrt{D_x(x,y)^2+D_y(x,y)^2}$$

entspricht der Größe der Änderung der Gauß-geglätteten Bildfunktion am Ort *(x, y)* und liefert damit ein gutes Maß für die Stärke der Kante.

Für die Verwendung in einem digitalen Rechner muß die Filterfunktion zum einen gekappt werden, da die erste Ableitung der Gauß-Funktion nur asymptotisch gegen Null geht, und zum anderen muß sie abgetastet (diskretisiert) werden. Für die verwendete Filtergröße empfiehlt Canny, die Filterfunktion an der Stelle zu kappen, an der die Filterwerte 0.1% des größten Wertes unterschreiten.

Die Standardabweichung der Gauß-Funktion ist auch hier der einzige veränderbare Parameter der verwendeten Filterfunktion. Durch Vergrößerung der Standardabweichung wird eine größere Glättungswirkung erzielt, mit dem Nachteil einer schlechteren Lokalisation der Kanten. Bei größeren Filtermasken beeinflussen sich benachbarte Kanten, sofern sie gleichzeitig in dem Bereich derselben Maske liegen. Auf diesen Umstand haben schon Marr und Hildreth hingewiesen und aus diesem Grund für die Verwendung möglichst kleiner Filtermasken plädiert. Canny hat diesen Fall durch seine Grundannahme, daß nur eine Kante im Bereich der Filterfunktion liegt, ausgeschlossen. Diese Annahme ist aber nicht realistisch.

Im nächsten Schritt wird beim Canny-Operator die vorhandene Information über die Gradientenrichtung und -stärke in dem Kanten-Nachbearbeitungsverfahren der Non-Maxima-Suppression ausgenutzt (siehe Abschnitt 11.10.1).

In der Version des Non-Maxima-Suppression-Verfahrens von Canny wird die lokale Richtung des Gradienten durch seine Komponenten $D_x(x, y)$ und $D_y(x, y)$ bestimmt. Daraus berechnet sich die lokale Normale $\vec{n}$ zur Kante als (vergleiche Abbildung 11.10):

$$\vec{n}(x,y)=n_x(x,y)\vec{e}_x+n_y(x,y)\vec{e}_y$$

mit den Komponenten

$$n_x(x, y) = \frac{D_x(x,y)}{\sqrt{D_x(x,y)^2+D_y(x,y)^2}} \quad \text{und} \quad n_y(x, y) = \frac{D_y(x,y)}{\sqrt{D_x(x,y)^2+D_y(x,y)^2}}$$

Problematisch ist, daß die Bildfunktion und das Filterergebnis nur auf einem diskreten Gitter vorliegen und daß die Gradientenrichtung im allgemeinen nicht genau in Richtung eines Gitterpunktes aus der 8-Punkte-Umgebung zeigt. Aus diesem Grunde werden beim Canny-Operator die zwei gesuchten angrenzenden Gradientenwerte näherungsweise durch lineare Interpolation der Gradientenwerte, die der Gradientenrichtung am nächsten liegen, berechnet. Mit Hilfe des folgenden Beispiels soll die Funktionsweise des Interpolationsverfahrens erläutert werden:

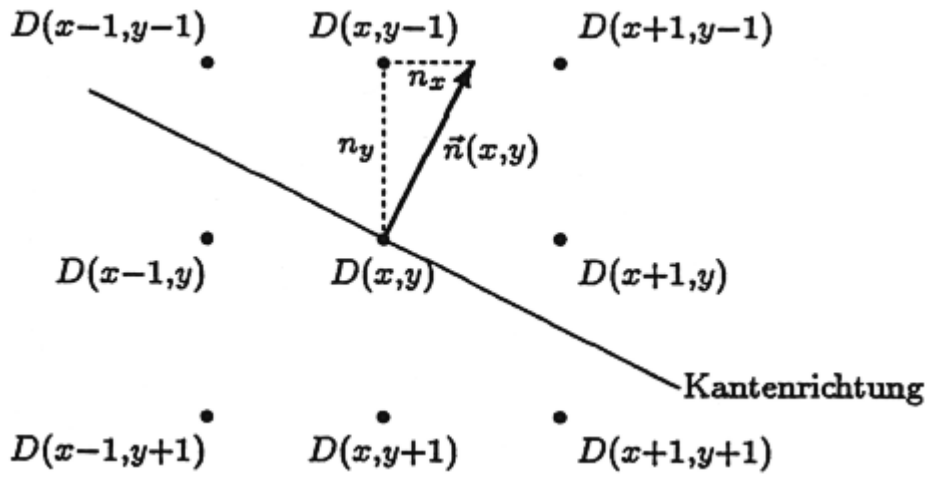


Abbildung 11.10: Suche nach dem lokalen Maximum in Gradientenrichtung.

Entsprechend dem Beispiel in Abbildung 11.10 berechnen sich die beiden interpolierten Gradienten D_1 und D_2 zu:

$$D_1 = \frac{n_x}{n_y} D(x+1, y-1) + \frac{n_y - n_x}{n_y} D(x, y-1)$$

$$D_2 = \frac{n_x}{n_y} D(x-1, y+1) + \frac{n_y - n_x}{n_y} D(x, y+1)$$

Analog ergeben sich die interpolierten Werte D_1 und D_2 für die sieben anderen möglichen Richtungen (Oktanten). Die Auswahl der richtigen Richtung erfolgt durch Fallunterscheidung an jedem Punkte anhand der Normalenrichtung.

Ein Maximum liegt dann vor, wenn $D(x, y) \geq D_1$ und $D(x, y) \geq D_2$ [MD86]. Dieses Vorgehen bedeutet, daß die Kantenpunkte im Gebiet der höheren Intensität gewählt werden.

Nach einer anschließenden Schwellwertbildung der so vorselektierten Kantenpunkte erhält man mit Hilfe des Non-Maximum-Suppression-Verfahrens 1-Pixel breite Kanten, die den Ort der stärksten Grauwertänderung bestimmen. Daher kann dieser Algorithmus auch als intelligentes Skelettierungs-Verfahren angesehen werden (vergleiche Abschnitt 12), das nicht nur die geometrische Anordnung der Punkte wertet wie traditionelle Methoden, sondern auch die Information aus der vorangehenden Kantendetektionsstufe.

Aus der bisher berechneten Menge von möglichen Kantenpunkten muß durch ein geeignetes Schwellwertverfahren entschieden werden, ob es sich bei einem Punkt aufgrund seines Gradientenwertes um einen Kantenpunkt handelt oder nicht. Ein solches Verfahren, das nicht einen festen Wert, sondern einen Bereich verwendet, ist das Hysteresis-Threshold-Verfahren [Can83] (vergleiche Abschnitt 11.10.2).

Die Annahme für die Definition der Optimalität des Canny-Operators, daß sich jeweils nur eine Kante im örtlichen Einzugsbereich des Kantenoperators befindet, ist in der Praxis natürlich nicht haltbar. Dies führt zu Unterbrechungen und falschen Verbindungen vor allem im Bereich von Kreuzungen (siehe Abbildung 11.11). Ein zusätzliches Kanten-Nachbearbeitungsverfahren, das Constraint-Thinning [Gru91], kann diese Lücken teilweise wieder schließen (vergleiche Abschnitt 11.10.3).

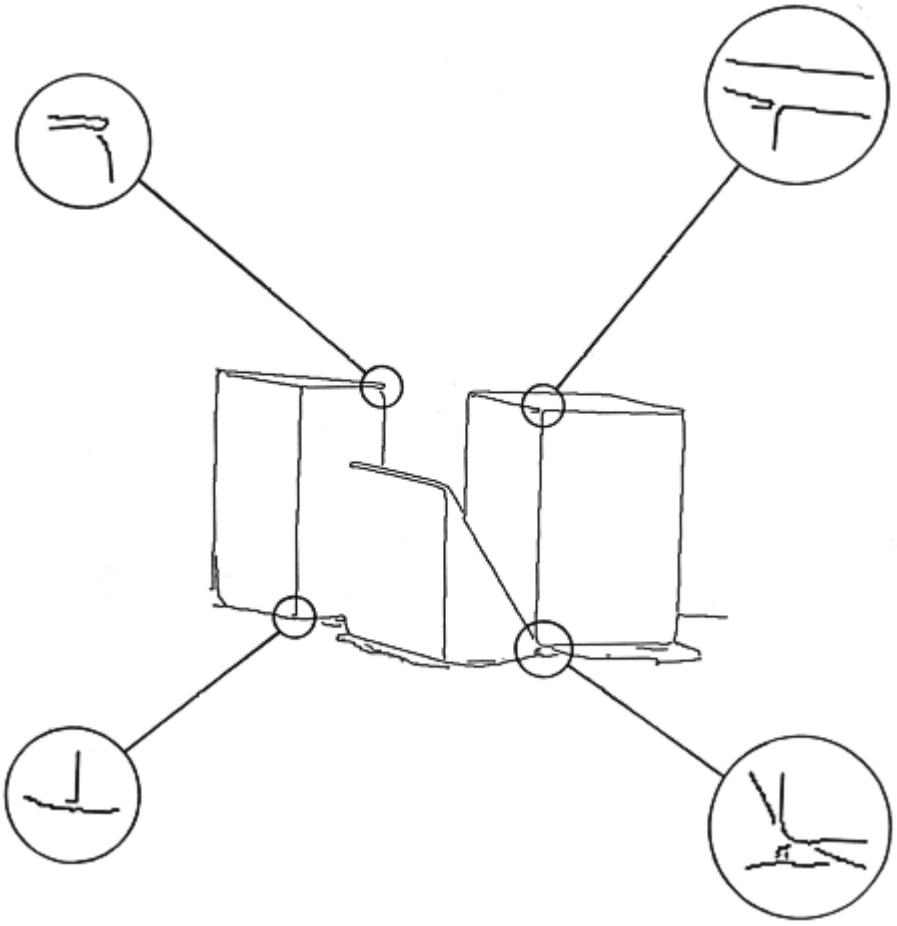


Abbildung 11.11: Beim normalen Canny-Operator entstehen an Stellen, an denen sich mehrere Kanten treffen, kaum sichtbare Lücken.

Aufgrund dieser gesamten Vorgehensweise liefert der Canny-Operator gute Ergebnisse bei der Anwendung auf reale Bilder. Die Verwendung der ersten Ableitung entspricht einer "natürlichen" Definition des Ortes einer Kante. Gleichzeitig ist damit die Rauschempfindlichkeit wesentlich geringer als z.B. bei dem Marr-Hildreth-Operator. Leider wird die Geschlossenheit der Konturen nicht garantiert (vergleiche Abschnitt 11.8.1).

```
/*************************************************************/
/* Aufgrund der Laenge des Algorithmus wird er hier nur im   */
/* prinzipiellen Ablauf beschrieben.                         */
/*************************************************************/

#define MAX_RAND       20   /* maximale Randgroesse um das Bild  */
#define TH_CONSTR_THIN 70   /* Schwellwert fuer Constraint-Thin. */

void Canny(int BildNr1, int BildNr2)
{
  /* separierte Faltungsmasken  G = G1 * G2                    */
  /* Gx = [(-x/Sigma^2)*exp{-x^2/2Sigma^2}]*[exp{-y^2/2Sigma^2}] */
  float G1[2*MAX_RAND+1];      /* Faltungsmasken 1             */
  float G2[2*MAX_RAND+1];      /* Faltungsmasken 2             */

  /* Hilfsfelder fuer die Berechnung; G, Gx, Gy = Gradientenfeld.*/
  float Temp_Pic[Z_RES_MAX+MAX_RAND][S_RES_MAX+MAX_RAND];
  float G[Z_RES_MAX][S_RES_MAX];
  float Gx[Z_RES_MAX][S_RES_MAX];
  float Gy[Z_RES_MAX][S_RES_MAX];
  unsigned char Markierung[Z_RES_MAX][S_RES_MAX];
  unsigned char Kante[Z_RES_MAX][S_RES_MAX];

  float Zwerg;                 /* Zwischenergebnis             */
  float Sigma;                 /* Standardabweichung Sigma     */
  double e, Abstand;
  double Sigma_Quadrat;        /* Sigma * Sigma                */
  int Radius;                  /* Operatorradius               */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Anwendung des mod. Canny-Operators *");

  /* Zuerst separierte Filter-Matrix (=Filterkern) berechnen.  */
  Sigma = Read_Float("Sigma (1.0 - ...)    ", 1.0, '?' );
  do
     Radius=Read_Int("Reichweite (max. 20)? ", 4, ':' );
  while ((Radius > 20) || (Radius < 1));
  Sigma_Quadrat=(double)(Sigma*Sigma);

  /* Eigentliche Berechnung der separierten Faltungsmasken     */
  for (i= -Radius; i<=Radius; i++) {
     Abstand=(double)(i*i);
     e = exp(-Abstand/(2.0*Sigma_Quadrat));
     G1[i+ Radius] = (float)e * -1.0 * i / Sigma_Quadrat;
     G2[i+ Radius] = (float)e;
  }
  /* Erzeugung eines "Radius"-Pixel breiten Randes zur einf. Ber.*/
  Erzeuge_Rand(BildNr1, Radius);
```

```
/* Differenzierung in X-Richtung:                                    */
/*                      1) Faltung in x-Richtung (Ableitung)         */
/*                         Faltung mit der ersten Maske              */
for (z=Radius; z<(Radius+Picture[BildNr1].Zeilen); z++)
    for (s=Radius; s<(Radius+Picture[BildNr1].Spalten); s++) {
        Zwerg = 0.0;
        for (x= -Radius; x<=Radius; x++)
             Zwerg += G1[x+Radius] *
                      Picture[BildNr1].Bild[z][s+x];
        Temp_Pic[z-Radius][s-Radius]=Zwerg;
    }
/* Rand auch um das Zwischenergebnis erzeugen                        */
Erzeuge_Rand(Temp_Pic, Radius);

/* Differenzierung in X-Richtung:                                    */
/*                      2) Faltung in x-Richtung (Glaettung)         */
/*                         Faltung mit der zweiten Maske             */
for (z=Radius; z<(Radius+Picture[BildNr1].Zeilen); z++)
    for (s=Radius; s<(Radius+Picture[BildNr1].Spalten); s++) {
        Zwerg = 0.0;
        for (y= -Radius; y<=Radius; y++)
            Zwerg += G2[y+Radius] * Temp_Pic[z+y][s];
        Gx[z-Radius][s-Radius] = Zwerg;
    }

/* Differenzierung in Y-Richtung:                                    */
/*                      1) Faltung in y-Richtung (Ableitung)         */
/*                         Faltung mit der ersten Maske              */
...

/* Rand auch um das Zwischenergebnis erzeugen                        */
Erzeuge_Rand(Temp_Pic, Radius);

/* Differenzierung in Y-Richtung:                                    */
/*                      2) Faltung in y-Richtung (Glaettung)         */
/*                         Faltung mit der zweiten Maske             */
for (z=Radius; z<(Radius+Picture[BildNr1].Zeilen); z++)
    for (s=Radius; s<(Radius+Picture[BildNr1].Spalten); s++) {
        ...
        /* Bestimmung des Gradienten im untersuchten Punkt           */
        G[z-Radius][s-Radius] = sqrt((double)
                  Gx[z-Radius][s-Radius]*Gx[z-Radius][s-Radius] +
                  Gy[z-Radius][s-Radius]*Gy[z-Radius][s-Radius]);
    }

/****************** Non-Maxima Suppression *********************/
/* G          = Gradientenmatrix                                 */
/* Gx         = Ableitung in horizontaler Richtung               */
/* Gy         = Ableitung in vertikaler Richtung                 */
/* Markierung = Ergebnisarray mit den berechneten Markierungen */
NonMaxSuppression(G, Gx, Gy, Markierung, Picture[BildNr1].Zeilen,
                                         Picture[BildNr1].Spalten);
```

```
    /****************** Hysteresis-Threshold ***********************/
    /* G          = Gradientenmatrix                                */
    /* Markierung = Markierungen aus dem Non-Maxima-Suppression     */
    /* Kante      = Ergebnisarray mit den Kantenpunkten             */
    HysteresisThreshold(G, Markierung, Kante,Picture[BildNr1].Zeilen,
                                          Picture[BildNr1].Spalten);

    /****************** Constraint-Thinning ***********************/
    /* G          = Gradientenmatrix                                */
    /* Kante      = Kantenpunkte aus dem Non-Maxima-Suppression     */
    /* BildNr2    = Ergebnisbild mit alten und zusaetzlichen Punkt.*/
    ConstraintThinning(G, Kante, BildNr2, Picture[BildNr1].Zeilen,
                                          Picture[BildNr1].Spalten);
}
```

11.9 3D-Operatoren

3D-Kantenfilter können durch direkte Übersetzung der 2D-Filter auf drei Dimensionen erzeugt werden. Bei den Differenzoperatoren hat Liu [Liu77] als einer der ersten den Roberts-Operator auf 3D erweitert. Analog sieht die 3×3×3-Version des Prewitt-Operators aus.

Die entsprechend rotierten Filtermasken werden in den drei Koordinatenrichtungen angewendet.

Der Marr-Hildreth-Operator in drei Dimensionen lautet demnach:

$$\nabla^2 G_\sigma(x, y, z) = \frac{1}{\pi\sigma^4}\left(\frac{(x^2+y^2+z^2)}{2\sigma^2}-1\right)\left(\exp\left[-\left(\frac{(x^2+y^2+z^2)}{2\sigma^2}\right)\right]\right)$$

Analog läßt sich der 3D-Canny-Operator definieren [MD86].

Anwendung können solche Operatoren in all den Gebieten finden, in denen 3D-Datensätze anfallen, z.B. bei CT-/MRT-Aufnahmen in der Medizin. Die Abtastverhältnisse sind dann aber von besonderer Wichtigkeit, da oftmals keine isotropen Abtastdichten vorliegen. Dies muß in den Maskenkoeffizienten der Operatoren entsprechend berücksichtigt werden.

Ist beispielsweise das Abtastverhältnis $\Delta x/\Delta y = ¼$, so sind die Filterkoeffizienten in den z-Ebenen ±1 um die zentrale z-Ebene bei dem 3D-Canny-Operator mit einer Standardabweichung von $\sigma = 1$ bereits um den Faktor $\exp(-4^2/2) \approx 0.00033$ niedriger als die Maskenkoeffizienten der zentralen Ebene. Eine Berücksichtigung dieser Werte lohnt sich dann nicht, so daß der Operator wieder zu einer 2D-Version degeneriert.

Im Fall isotroper Abtastung oder hoher Korrelation zwischen den Schichten wird jedoch ein höherer Rauschabstand erzielt als bei den zweidimensionalen Operatoren. Gleichzeitig steigt aber der Rechenaufwand um ein Vielfaches.

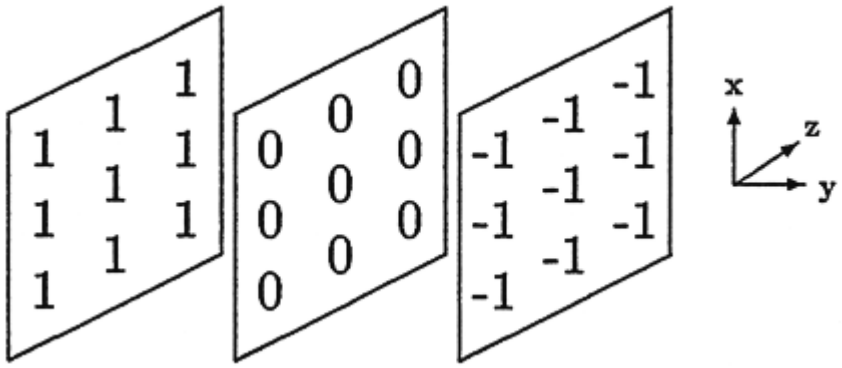


Abbildung 11.12: Eine Maske des 3D-Prewitt-Operators.

11.10 Kanten-Nachbearbeitung

Alle vorgestellten Kantenoperatoren liefern nach verschiedenen Vorverarbeitungsschritten wie Glättung usw. eine Ergebnis-Matrix mit Information über die potentiellen Kantenpunkte, z.B. lokale Kantenstärke und Kantenrichtung.

Die im nachfolgenden besprochenen Verfahren zur *Kanten-Nachbearbeitung* setzen auf diese Information auf und versuchen, diese Daten besser als mit den einfachen Schwellwertverfahren auf den Betragsgradientenbildern auszuwerten. Aufgrund dieser Vorgehensweise können die Kanten-Nachbearbeitungsverfahren aber auch im Zusammenhang mit den Differenzoperatoren verwendet werden.

11.10.1 Non-Maxima-Suppression

Einer der ersten Nachbearbeitungsschritte wurde schon beim Canny-Operator beschrieben; das Beseitigen der Nicht-Kantenpunkte im *Non-Maxima-Suppression*-Verfahren.

In vielen Kantendetektionsverfahren wird die Richtung des Gradienten nicht ausgewertet, sondern aufgrund einer globalen Schwelle z.B. über den Betragsgradienten Punkt für Punkt entschieden, ob an dieser Stelle eine Kante vorliegt oder nicht. Dadurch entstehen je nach Kantenform und -verlauf größere Flächen, in denen eine mögliche Kante liegt. Durch Skelettierung (vergleiche Kapitel 12) wird dann die Mittelachse als Ort der eigentlichen Kante festgelegt.

Bei einem einfachen Non-Maxima-Suppression-Verfahren kann in einer Umgebung um den aktuellen Punkt ungerichtet untersucht werden, ob der Betragsgradient im betreffenden Pixel ein lokales Maximum darstellt. Ist dies nicht der Fall, wird das Pixel gelöscht.

In einer verbesserten Version des Verfahrens wird zusätzlich die Gradientenrichtung ausgenutzt. Der zentrale Gradientenwert wird dazu mit zwei angrenzenden Gradientenwerten verglichen, die in Gradientenrichtung, aber auf verschiedenen Seiten des zentralen untersuchten Punktes liegen. Alle die Punkte, die innerhalb einer 8-Punkte-Umgebung kein lokales Maximum darstellen, werden dann auf Null gesetzt, d.h. für die weitere Bearbeitung unterdrückt. Daher auch die Bezeichnung Non-Maxima-Suppression.

Da in dem diskreten Gitter die Gradientenrichtung im allgemeinen nicht genau in Richtung eines Gitterpunktes aus der 8-Punkte-Umgebung zeigt, muß die Richtung angenähert werden. Als Lösungsmöglichkeit bietet sich die Auswahl von zwei Punkten aus der 8-Punkte-Umgebung an, die der Gradientenrichtung am nächsten liegen [Kor88], oder die näherungsweise Ermittlung der zwei gesuchten angrenzenden Gradientenwerte durch lineare Interpolation der Gradientenwerte (vergleiche Abschnitt 11.8.2), die der Gradientenrichtung am nächsten liegen [Can83]. Als Kantenpunkte werden die Punkte im Gebiet der höheren Intensität gewählt.

Andere Arten von Non-Maxima-Suppression-Verfahren sind denkbar und in [Gen86], [Kor88] und [Lac88] beschrieben.

11.10.2 Hystersis-Threshold

Über das Non-Maxima-Suppression-Verfahren wurde die Anzahl der potentiellen Kantenpunkte erheblich reduziert. Aus dieser Menge kann jetzt durch ein geeignetes Schwellwertverfahren entschieden werden, ob es sich bei einem Punkt aufgrund seines Gradientenwertes um einen Kantenpunkt handelt oder nicht.

Bei dem *Hysteresis-Threshold*-Verfahren wird nicht ein einzelner Schwellwert, sondern ein Intervall verwendet [Can83].

Die Entwicklung dieses Verfahrens beruht auf der Beobachtung, daß die Kanten, die mit Hilfe einfacher Schwellwertverfahren erzeugt wurden, häufig Unterbrechungen aufweisen. Diese Unterbrechungen kommen durch stückweises Über- und Unterschreiten des Schwellwertes zustande. Damit wird die Schwierigkeit der richtigen Wahl *eines* Schwellwertes deutlich, mit dessen Hilfe zutreffende Kanten von Rauschen und schwachen Kanten möglichst weitgehend getrennt werden sollen.

Basierend auf der realistischen Annahme, daß ein Kantenstück, das aus schwachen Kantenpunkten besteht, eher als falsch detektiert angesehen werden muß, als wenn dieses Kantenstück zu einer Kante gehört [NB86], die auch starke Kantenpunkte enthält, gelangt man zu diesem Schwellwertverfahren mit zwei Schwellwerten [Can83]. Punkte, an denen der Gradientenbetrag den hohen Schwellwert TH_{high} überschreitet, werden verwendet, um neue Konturen zu beginnen, während Punkte mit Gradientenbeträgen über dem niedrigeren Schwellwert TH_{low} der Fortsetzung dienen. TH_{low} ist die untere Grenze, unterhalb der die Grauwertdiskontinuitäten entweder dem Rauschen im Bild oder unwichtigen, weil zu schwachen Details zugeschrieben werden.

Das Hysteresis-Threshold-Verfahren kann daher als einfacher Konturverfolgungs-Algorithmus interpretiert werden.

Für die Abschätzung der Wirkung des Hysteresis-Threshold-Verfahrens sind vor allem zwei Beobachtungen interessant:

- Eine Kontur, auf der zwar alle Gradientenbeträge zwischen TH_{high} und TH_{low} liegen, TH_{high} aber nicht erreichen, wird nicht anerkannt.
- Es genügt ein einziger Punkt mit Gradientenbetrag größer TH_{high}, damit eine gesamte Kontur, entlang der sonst kein weiterer Gradientenbetrag TH_{high} übersteigt, anerkannt wird.

Diese so berechneten "sicheren" Kantenpunkte werden als "Kristallisationspunkte" für komplette Kanten verwendet. Aufgrund der Ergebnisse bei verschiedenen Testbildern hat sich als sinnvolles Verhältnis zwischen oberer und unterer Schwelle ein Wert von 2–3 ergeben [Can83], [Gru91]. Die obere Schwelle wird wieder in Abhängigkeit der Gradientenwerte bei gleichzeitiger Unterdrückung von Extremwerten berechnet (vergleiche Abschnitt 11.2).

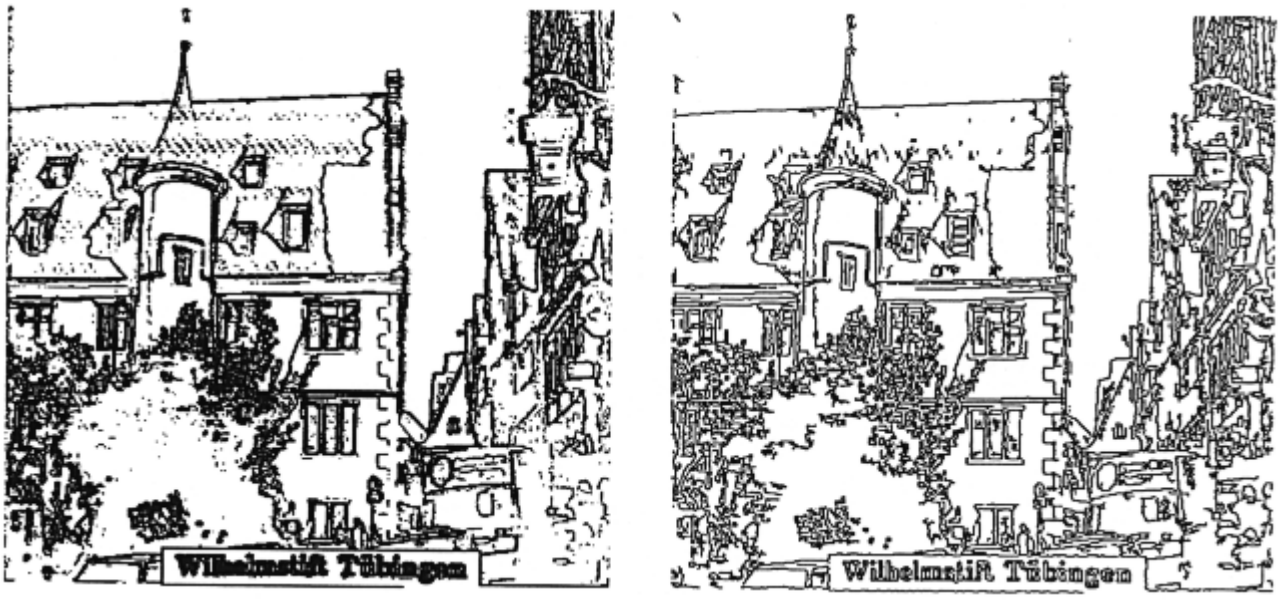


Abbildung 11.13: Vergleich des Ergebnisses des einfachen Schwellwertverfahrens (links) mit dem Hysteresis-Threshold-Verfahren (rechts).

In entsprechenden Untersuchungen [Gru91] erwies sich das Hysteresis-Threshold-Verfahren als sehr robust. Es war selbst bei festen Schwellwerten ohne Änderung erfolgreich auf eine größere Anzahl verschiedener Bilder übertragbar. Die erzeugten Kanten waren über wesentlich längere Strecken nicht unterbrochen, als es selbst bei langwieriger Hand-Optimierung mit Hilfe eines einfachen Schwellwertverfahrens möglich gewesen wäre. Auch gegenüber Änderungen des Verhältnisses zwischen den Schwellwerten erwies sich das Verfahren als recht robust. Prinzipiell bewirkt aber eine zu hohe Wahl von TH_{high}, daß auch lange "gute" Kantenstücke nicht anerkannt werden. Eine

Verminderung von TH_{low} fördert die Verbindung von einzelnen Kantenstücken und führt damit zu langen fortgesetzten Konturen. Im Extremfall eines sehr niedrig gewählten TH_{low} kann es daher dazu kommen, daß einige wenige "Saatpunkte" mit Gradientenbeträgen größer TH_{high} genügen, um praktisch alle Punkte, solange sie nur Gradientenbeträge größer als TH_{low} aufweisen, als Kantenpunkte anzuerkennen.

```
#define MARK_TH1      128   /* Markierungen fuer das ...         */
#define MARK_TH2      254   /* Hystheresis-Threshold-Verfahren   */
#define MARK_EDGE     255
#define TH_LOW         20   /* THlow  fuer Hysteresis-Threshold. */
#define TH_HIGH        90   /* THhigh fuer Hysteresis-Threshold. */

HysteresisThreshold(float Grad[Z_RES_MAX][S_RES_MAX],
                    unsigned char Markierung[Z_RES_MAX][S_RES_MAX],
                    unsigned char Kante[Z_RES_MAX][S_RES_MAX],
                    int Dim_Zeilen, int Dim_Spalten)
/* Grad       = Gradientenmatrix                                  */
/* Markierung = Markierungen aus dem Non-Maxima-Suppression       */
/* Kante      = Ergebnisarray mit den Kantenpunkten (MARK_EDGE)   */
/* Dim_Zeilen = Anzahl der Zeilen des Bildes                      */
/* Dim_Spalten= Anzahl der Spalten des Bildes                     */
{
  unsigned char Temp_Pic1[Z_RES_MAX][S_RES_MAX];  /* Hilfs-Array */
  int z, s;                                /* Zeilen-/Spaltenindex */

  /* Als Kantenpunkte kommen nur solche in Frage, die beim Non-  */
  /* Maxima-Suppression als lokale Maxima markiert wurden und die*/
  /* im Gradientenbild mindestens die untere Schwelle ueber-     */
  /* schreiten.                                                  */
  /* Es wird dabei zwischen Sicheren-Kantenpunkten (> TH_HIGH)   */
  /* und Moeglichen-Kantenpunkten (<= TH_HIGH und > TH_LOW)      */
  /* unterschieden.                                              */
  for (z=0; z<Dim_Zeilen; z++)
      for (s=0; s<Dim_Spalten; s++) {
          if      (Markierung[z][s] == 0) Temp_Pic1[z][s]=0;
          else if (Grad[z][s] > TH_HIGH)  Temp_Pic1[z][s]=MARK_TH2;
          else if (Grad[z][s] > TH_LOW)   Temp_Pic1[z][s]=MARK_TH1;
          else                            Temp_Pic1[z][s]=0;

          Kante[z][s]=0;          /* Ergebnis-Bild initialisieren*/
      }
```

```
  /* Sichere-Kantenpunkte im Hilfs-Array suchen und ausgehend    */
  /* davon die moegliche Kante verfolgen.                        */
  for (z=0; z<Dim_Zeilen; z++)
      for (s=0; s<Dim_Spalten; s++)
          if (Temp_Pic1[z][s] == MARK_TH2)
             /****************** Kantenverfolgung **************/
             /* (z,s)     = Koordinate des sicheren Kantenpunktes*/
             /* Temp_Pic1 = Array mit den sicheren und moeglichen*/
             /*             Kantenpunkten                       */
             /* Kante     = Ergebnis-Array mit den gefundenen    */
             /*             Kantenpunkten (verfolgten).         */
             /*             Die Kantenpunkte werden in beiden   */
             /*             Arrays mit MARK_EDGE gekennzeichnet. */
             Verfolge_Kante(z, s, Temp_Pic1, Kante);
}
```

11.10.3 Das Constraint-Thinning

Wie schon erwähnt, kommt es in Kombination mit dem Non-Maxima-Suppression-Verfahren zu Fehlern, wenn sich mehrere Kanten im Einzugsbereich des Operators befinden. Die starke Kante dominiert in diesem Bereich und schwächere Kanten können nicht mehr ermittelt werden. Bei theoretischen Untersuchungen wird dieser Fehler meist dadurch ausgeschlossen, daß angenommen wird, es liege jeweils nur eine Kante im Bereich des Kantenoperators. Dies ist in der Praxis aber normalerweise nicht erfüllt.

Mit dem *Constraint-Thinning*-Verfahren ist es nun möglich, diese Lücken in den meisten Fällen wieder zu schließen. Das Verfahren ordnet sich folgendermaßen in einen Kantendetektionsprozeß ein:

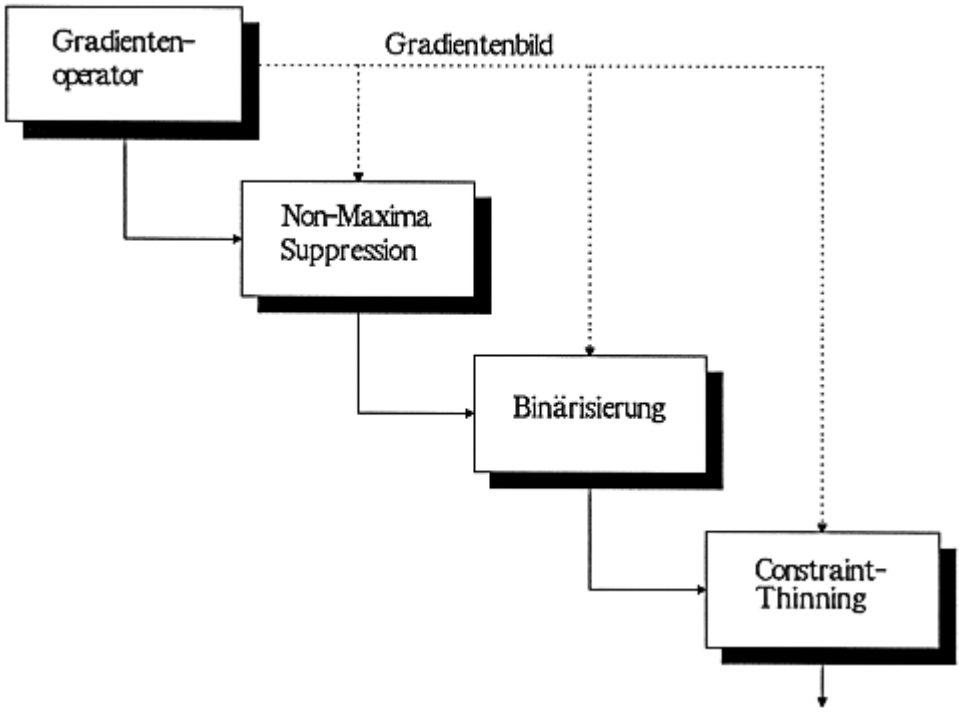


Abbildung 11.14: Einordnung des Constraint-Thinning-Verfahrens.

Mit Hilfe eines zusätzlichen einfachen Schwellwertverfahrens wird ein Binärbild des Gradientenbildes erstellt. Anschließend wird dieses Binärbild im eigentlichen Constraint-Thinning-Prozeß mit einem Skelettierungs-Verfahren skelettiert (siehe Kapitel 12), allerdings unter der Bedingung, daß die in der ersten und zweiten Stufe markierten Maxima nicht gelöscht werden dürfen. Daher auch die Bezeichnung Constraint-Thinning.

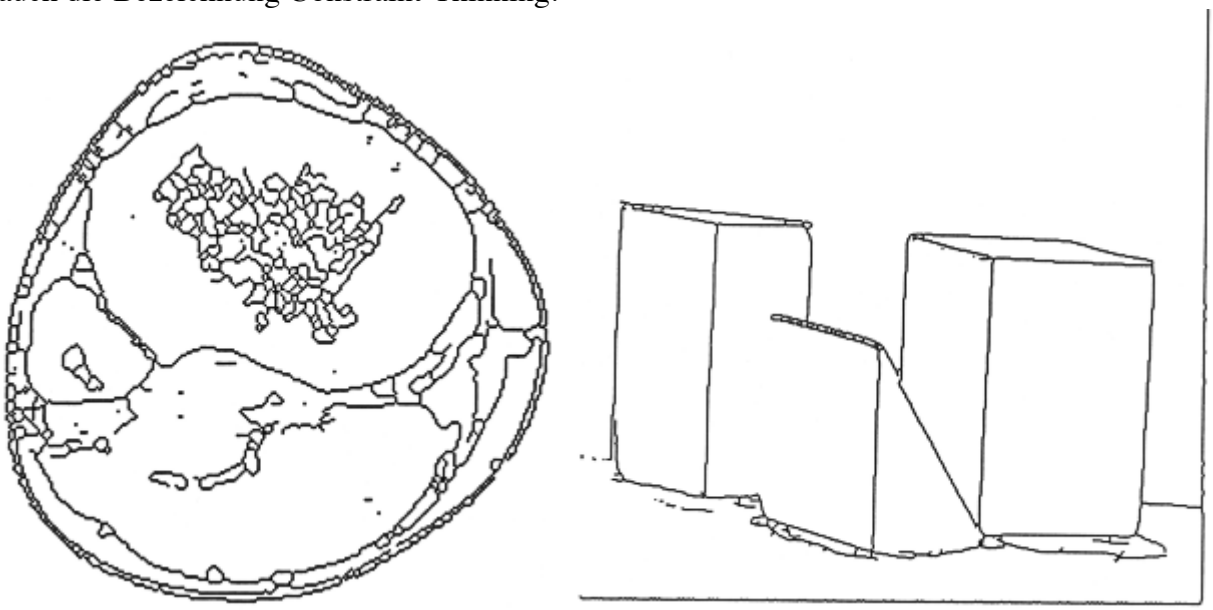


Abbildung 11.15: Ergebnis des Canny-Operators mit dem Constraint-Thinning nach der Anwendung auf eine CT-Aufnahme aus dem Bereich des Knies (links) und auf das Testbild mit der Polyeder-Gruppe (vergleiche Abbildung 11.11).

Auf diese Weise erhält man zum einen die lokalen Maxima als Ortsbeschreibung von Kanten und zum anderen die Mittelachsenkonturen an Orten, an denen vorher keine lokalen Maxima gefunden wurden. Im Idealfall schließen diese zusätzlichen Konturen die Lücken, die beim Non-Maxima-Suppression-Verfahren entstanden sind.

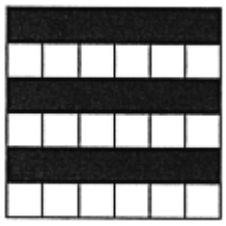
Nachteilig machen sich in Einzelfällen einige fälschlicherweise als Kantenpunkte berechneten Punkte bemerkbar. Kommt es bei der Schwellwertbildung in der Constraint-Thinning-Stufe zu einer teilweisen Überlappung sehr nahe benachbarter Kanten, so kann dies nach dem Verdünnen der Bereiche unter Umständen zu einer Vielzahl an Verbindungen ("Leitereffekt") zwischen den vormals getrennten, aber dicht beieinanderliegenden Kanten führen (vergleiche Abbildung 11.15, links). Dieser Effekt tritt vor allem dann auf, wenn der verwendete Kantenoperator einen großen örtlichen Einzugsbereich hat, was zu einer weiträumigen "Verschmierung" der Kanten führt, wenn der Schwellwert zu niedrig gewählt wurde.

Im allgemeinen hat es sich daher als günstig erwiesen, einen eher hohen Schwellwert für das Constraint-Thinning zu verwenden. Gute Ergebnisse konnten erzielt werden, wenn der Schwellwert der Constraint-Thinning-Stufe etwas niedriger gewählt wurde als der hohe Schwellwert TH_{high} des Hysteresis-Threshold-Verfahrens zur Binärisierung der lokalen Maxima.

Aufgaben

Aufgabe 1

Gegeben ist das folgende Bild mit abwechselnd schwarzen (=0) und weißen (=255) Linien (Wellenlänge=2 Pixel). Geben Sie jeweils das Ergebnis der Faltung mit folgenden beiden Masken (Sobel und Laplace) an. Interpretieren Sie das Ergebnis.



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

0	1	0
1	-4	1
0	1	0

Aufgabe 2

Gegeben sei ein Bildausschnitt aus einem Graukeil:

				...				
0	15	30	45	60	75	90	105	...
0	15	30	45	60	75	90	105	...
0	15	30	45	60	75	90	105	...
0	15	30	45	60	75	90	105	...
0	15	30	45	60	75	90	105	...
0	15	30	45	60	75	90	105	...
				...				

Wenden Sie auf diesen Bildausschnitt den Laplace-Operator an. Wie lautet das Ergebnis?

Aufgabe 3

In einem Bild, das nur aus Hintergrund (Grauwert 240-255) und Text (Grauwert 0-20) besteht, existieren Störungen (Rauschen) in Form von vereinzelten isolierten Punkten (Grauwert 0-50). Wie kann der Laplace-Operator zur Herausfilterung dieser Störungen eingesetzt werden?

Aufgabe 4

Bestimmen Sie die Richtung(en) der 1-Pixel breiten Linie(n), die die stärkste Antwort bei folgenden Liniendetektionsmasken hervorrufen.

a:

0	1	0
-1	0	-1
0	1	0

b:

1	-2	1
-2	4	-2
1	-2	1

12. Skelettierung

Im Hinblick auf Bildinterpretation oder Mustererkennung muß die in den Bildvorlagen vorhandene Information auf die für die weitere Verarbeitung wichtigen Bestandteile reduziert und abstrahiert werden. Die bisher beschriebenen Verfahren, hierbei im besonderen die Kantendetektion, sind ein wichtiger Schritt in diese Richtung. Die detektierten Kanten sollten dabei möglichst genau die einzelnen Gebiete (Umrisse der Objekte) umschließen. Einige Kantenoperatoren liefern jedoch mehrere Pixel breite Kanten, wie z.B. der Prewitt- und der Sobel-Operator, was zu einer größeren Ungenauigkeit und zu Problemen bei der weiteren Verarbeitung führt. Ausgehend von den Kantenmodellen (siehe Abschnitt 11.1) kann man definieren, daß die wirkliche Kantenposition genau in der Mitte der mehrere Pixel breiten Kante liegt. Diese Mitte ist mit geeigneten Verfahren zu berechnen.

Nicht nur bei Kanten, sondern allgemein bei flächenhaften Objekten, kann die Mittellinie, das sogenannte *Skelett*, als abstrahierte Repräsentation der Figur verwendet werden. Die *Skelettierung* (*Thinning*) wird daher oft auch als Mittelachsen- oder Medialachsentransformation bezeichnet.

Die Berechnung solch eines Skeletts geschieht theoretisch durch die Berechnung der Entfernung von Punkten innerhalb der Kante/Figur zum Rand hin. Dadurch werden die nächsten Randpunkte bestimmt. Punkte, die mehrere nächste Randpunkte (gleiche Entfernung) besitzen, bilden die Mittelachse (Skelett) des Objekts (siehe Abbildung 12.1). Dies impliziert, daß Skelettpunkte die Mittelpunkte von Kreisen mit maximalem Radius sind, die vollständig in der Figur liegen.

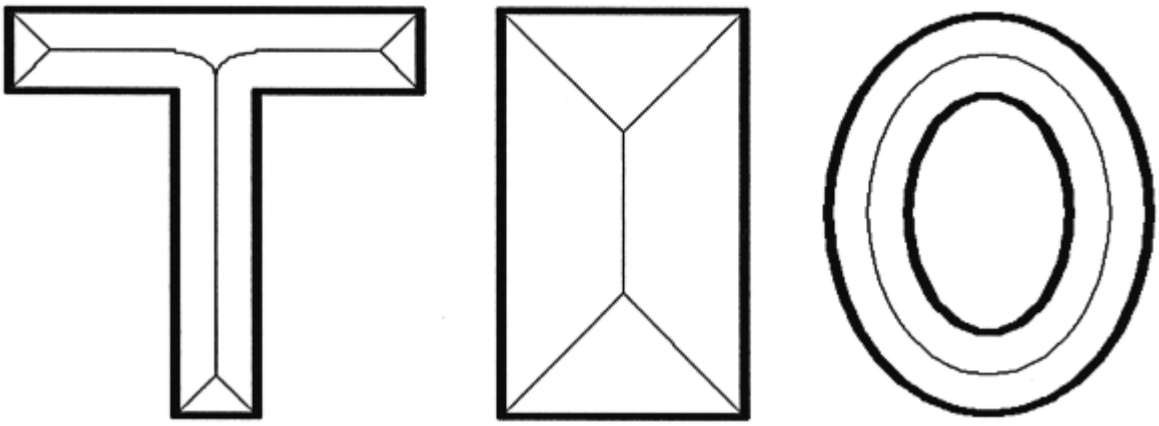


Abbildung 12.1: Das aus der Definition resultierende Skelett von Figuren. Ein Skelettpunkt ist zu mehreren Randpunkten gleich weit entfernt.

Leider ist diese anschauliche Definition nur sehr schwer in ein effizientes Programm umzusetzen, da die Berechnungen enorme Zeit in Anspruch nehmen. Daher gibt es verschiedene Ansätze, das Skelett über eine Art "Erosions-Vorschrift" zu erzeugen. Die einzelnen Verfahren sind unterschiedlich empfindlich gegenüber Störungen, liefern verschiedene Verkürzungen von Strecken und besitzen unterschiedliche Genauigkeiten.

Die Güte eines Skelettierungs-Verfahrens wird nach

- der Dicke des Skeletts (ideal: nur ein Pixel),
- der Genauigkeit, der Abweichung der berechneten von den idealen Skelettpunkten,
- der Beibehaltung der Topologie der Figur (das Skelett einer zusammenhängenden Figur sollte ebenfalls zusammenhängend sein),
- der Möglichkeit der Rekonstruktion der Originalfigur aus dem Skelett durch Anwendung der inversen Skelettierungs-Vorschrift,
- der Empfindlichkeit gegenüber Störungen und
- der Beibehaltung der absoluten bzw. relativen Größen der Figuren (Verkürzung von Strecken)

bewertet. Schon die Berechnung des idealen Skeletts in Abbildung 12.1 zeigt, daß die Beibehaltung der Topologie der Ausgangsfigur in einigen Fällen zu Problemen führen kann. Besonders bei dicken oder fast quadratischen oder kreisförmigen Objekten liefert das Skelett wenig Information über die ursprüngliche Gestalt.

12.1 Ein einfacher Algorithmus

Die Schwierigkeit der programmtechnischen Realisierung der theoretischen Definition eines Skeletts besteht in der Abstandsberechnung eines Objektpunktes zu allen Randpunkten in allen Richtungen. Wird die Anzahl der möglichen Richtungen beschränkt, so kann die Berechnung vereinfacht werden.

Besonders einfach wird es, wenn man sich auf die horizontale oder vertikale Richtung beschränkt. Bei der zeilen- oder spaltenweisen Abtastung der Figur muß dann der Skelettpunkt nur noch als Mittelpunkt einer Strecke berechnet werden. Das Verfahren wird daher auch mit *Scan-Line-Thinning* bezeichnet. Jede Abtastzeile/-spalte, die durch die Figur verläuft, enthält bei dieser Vorgehensweise mindestens einen Skelettpunkt.

Bei der Abtastung ist demnach nur die Position des Wechsels zwischen Hintergrund und Objekt sowie die Position des nächsten Wechsels zwischen Objekt und Hintergrund zu speichern. Die Position des Skelettpunktes ergibt sich dann durch Mittelung der Spaltenpositionen (bei zeilenweiser Abtastung) bzw. der Zeilenpositionen (bei spaltenweiser Abtastung). In Abbildung 12.2 ist diese Vorgehensweise an Beispielen verdeutlicht.

Je nach Ausgangsfigur liefert dieser einfache Skelettierungs-Algorithmus mehr oder minder viele Fehler. Diese betreffen alle die genannten Gütekriterien, von denen als einzige die Skelettstärke von einem Pixel und die Lage des Skeletts erfüllt wird.

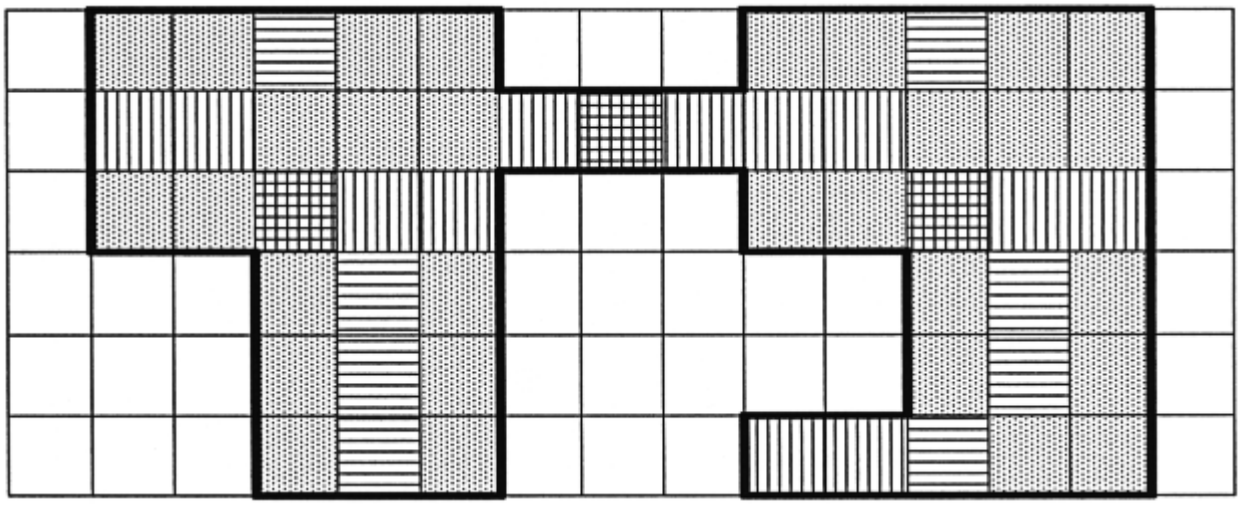


Abbildung 12.2: Das von dem einfachen Skelettierungs-Algorithmus berechnete Skelett. Die horizontal gestreiften Felder sind das Ergebnis der horizontalen Abtastung, die vertikal gestreiften Felder die der vertikalen. Einige Punkte werden bei beiden Abtastrichtungen als Skelettpunkte erkannt. An einigen Stellen liefert dieses Verfahren gute Ergebnisse. In den meisten Fällen entstehen aber Skelettpunkte, die den genannten Anforderungen nicht genügen.

Daß das mit diesem Verfahren erzeugte Skelett einer zusammenhängenden Figur nicht unbedingt zusammenhängend sein muß, zeigt das Bildbeispiel deutlich. Durch die Vereinigung der Ergebnisse von horizontaler und vertikaler Abtastung kann dieser Fehler allerdings auf Kosten neuer topographischer Fehler beseitigt werden. Durch diese Kombination wird eine bessere Berücksichtigung der Umgebung erreicht.

12.2 Verfahren mit 3×3-Masken

Wie das Beispiel des einfachen Skelettierungs-Algorithmus gezeigt hat, ist es für die Berechnung der Mittelachse unbedingt notwendig, die Umgebung eines Objektpunktes zu berücksichtigen. Dann erst kann korrekt entschieden werden, ob es sich um einen möglichen Skelettpunkt handelt oder nicht. Ideal wäre natürlich eine so große Umgebung, daß mindestens ein Randpunkt der Figur in dieser Umgebung liegt. Zur Beschleunigung der Rechenzeit und zur leichteren Handhabung wird bei den meisten Verfahren aber nur eine 1-Punkte-Umgebung in einer 3×3-Maske verwendet. In Abhängigkeit der Konstellation der 8 Nachbarn wird der mittlere Punkt als möglicher Skelettpunkt markiert oder gelöscht. Werden alle 8 direkten Nachbarn in die Berechnung mit einbezogen, so spricht man von einer 8er-Nachbarschaft. Algorithmen zur Skelettierung bei einer 4er-Nachbarschaft finden vereinzelt ebenfalls Anwendung.

Werden alle Kombinationen in der 8er-Nachbarschaft berücksichtigt, so sind $2^8 = 256$ verschiedene 3×3-Masken notwendig. Einige davon scheiden sofort aus, da sie kein Teil eines möglichen Skeletts darstellen. Rund 50 Masken sind aber immer noch notwendig [Kre77]. Aus Symmetriegründen ist es aber möglich, diese Anzahl auf 20 relevante Masken zu reduzieren. Die Erosions-Vorschrift kann daher mit einigen wenigen festen Masken definiert werden. Diese Masken geben an, unter welchen Nachbarschaftsbeziehungen ein mittleres Pixel kein möglicher Skelettpunkt ist und somit gelöscht werden kann. Dies sind zum Beispiel:

0	0	x
0	1	1
x	1	x

0	0	0
x	1	x
1	1	x

0	x	x
0	1	1
0	x	1

wobei 1 = "Rand-/Objektpunkt", 0 = "Hintergrundpunkt" und x = "die Punktzugehörigkeit ist nicht von Interesse" bedeuten. Alle diese Masken werden nun zeilen- und spaltenweise über das Bild gelegt. Wird an einer Stelle eine in einer der Masken dargestellte Kombination gefunden, so kann der mittlere Punkt gelöscht werden, da er keinen Skelettpunkt darstellt. Diese Masken werden solange auf das Bild angewandt, bis in einem kompletten Durchlauf keine Änderung mehr erfolgt ist, d.h. bis ein stationärer Zustand erreicht ist.

Durch diese Art der Skelettierung werden die Objekte in ihren Abmessungen im Skelett etwas kleiner. Wieviel, ist abhängig von der Dicke der Objekte. Ein großer Vorteil dieser 3×3-Masken ist die leichte Implementierbarkeit in Software und auch in Hardware mittels UND-, ODER-Gatter und Invertern.

Die einzelnen Skelettierungs-Algorithmen, die auf solchen 3×3-Masken aufbauen, unterscheiden sich im wesentlichen darin, welche der möglichen Masken in welcher Reihenfolge verwendet wird. Damit nicht nur von einer Seite erodiert wird, müssen die Masken in unterschiedlichen Richtungen über das Bild bewegt werden. Die abwechselnd zeilenweise Bearbeitung von links oben nach rechts unten in einem Durchgang und dann zeilenweise von rechts unten nach links oben bewirkt, daß das resultierende Skelett in etwa in der Objektmitte liegt. Die Anzahl der Iterationsschritte ist abhängig von der maximalen Dicke der Objekte bzw. Kanten.

12.2.1 Der Zhang/Suen-Algorithmus

Trotz der Einfachheit der verwendeten Masken werden die Algorithmen aufgrund der großen Anzahl meist unübersichtlich. Die Abarbeitung dauert durch die vielen Vergleiche und das iterative Vorgehen relativ lange. Betrachtet man die für eine Skelettierungs-Vorschrift in Frage kommenden 3×3-Masken genauer, so werden gewisse Gemeinsamkeiten und Systematiken im Aufbau deutlich. Mit dieser Erkenntnis können alle Masken in einigen wenigen Vorschriften zusammengefaßt werden.

Das Verfahren von Zhang/Suen [ZS84] verwendet die in [SR71] beschriebenen Vorschriften. Eine Erweiterung der Vorschriften soll das Verschieben des Skeletts zu einer Seite hin verhindern. Die Vorgehensweise kann wie folgt beschrieben werden:

- Zuerst baut man eine 3×3-Maske auf und kennzeichnet die einzelnen Nachbarn wie folgt:

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

- Danach definiert man die verschiedenen Nachbarschaftsbeziehungen, bei denen der Mittelpunkt P_1 für ein Skelett keine Rolle spielt.
- Im dritten Schritt werden diese Nachbarschaftsbeziehungen auf das ganze Bild angewandt, in der Regel abwechselnd von links oben nach rechts unten und umgekehrt, um eine Verschiebung des Skeletts durch die lokalen Operationen in eine Richtung zu verhindern.
- Oftmals wird noch eine weitere Skelettierungs-Vorschrift abwechselnd zu der im dritten Schritt verwendeten benutzt. Diese soll die Fehler, die durch die Lokalität der Operationen entstehen, noch weiter vermindern.

Der eigentliche Algorithmus lautet dann wie folgt:

(a) $2 \le N(P_1) \le 6$

(b) $S(P_1) = 1$

(c) $P_2 \cdot P_4 \cdot P_6 = 0$

(d) $P_4 \cdot P_6 \cdot P_8 = 0$

wobei $N(P_1)$ die Anzahl der von Null verschiedenen Nachbarn von P_1 ist (hier wird 0 = "Hintergrundpunkt" und 1 = "Rand-/Objektpunkt" verwendet) und $S(P_1)$ die Anzahl der 0-1-Übergänge in der geordneten Reihenfolge der P_i (i=2, 3, ... 9, 2). Bei der Anwendung der Vorschriften in umgekehrter Richtung (von rechts unten nach links oben) bleiben die Bedingungen (a) und (b) bestehen. Nur (c) und (d) werden ersetzt durch

(c') $P_2 \cdot P_4 \cdot P_8 = 0$

(d') $P_2 \cdot P_6 \cdot P_8 = 0$

Sind alle Bedingungen erfüllt worden, so kann Punkt P_1 als zu löschend markiert werden. Nach jedem vollendeten Schritt werden alle markierten Punkte wirklich gelöscht. Wird bei beiden Durchläufen kein Punkt gelöscht, so ist ein stationärer Zustand erreicht; das Skelett ist ermittelt.

Um eine zu starke Verkürzung von Strecken zu verhindern, kann die erste Bedingung in

(a) $3 \le N(P_1) \le 6$ abgeändert werden.

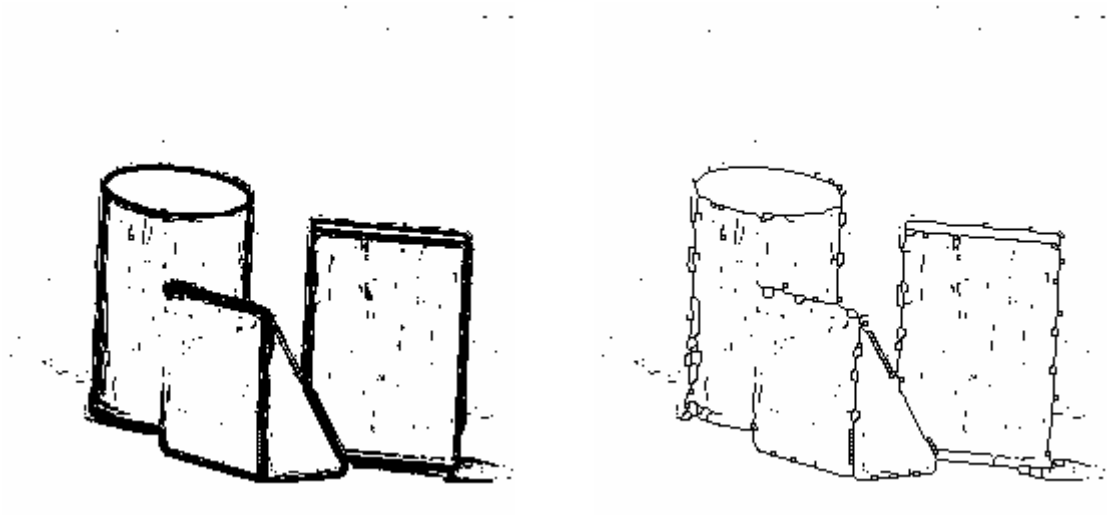


Abbildung 12.3: Das Skelettbild (rechts) wurde aus dem Originalbild (links) nach dem Verfahren von Zhang/Suen berechnet.

Die unterschiedlichen Abwandlungen dieses Verfahrens [Pav82], [LW85], [O'G90] variieren in der Richtung der Anwendung, in der Anzahl der von Null verschiedenen Nachbarn und in kleineren Zusatzbedingungen. Allen Versionen gemeinsam ist aber der Nachteil der Verkürzung von Strecken und die große Empfindlichkeit gegenüber Störungen im Ausgangsbild. Punktuelle Störungen können an einigen Stellen zu einem vollkommen geänderten Aussehen des Skeletts führen. Abhilfe können Verfahren schaffen, die nicht sofort auf einen Punkt Stärke reduzieren, sondern noch die Strukturen von 2-3 Punkten Stärke belassen. Die dabei verwendeten Regeln müssen nicht so viele Sonderfälle berücksichtigen, so daß gleichzeitig eine Glättung punktueller Störungen durchgeführt werden kann. Danach können die geglätteten und verdünnten Strukturen mit einem genaueren Verfahren auf einen Punkt reduziert werden (hierarchisches Vorgehen).

```
#define ERFUELLT   15

void Thinning(BildNr)
{
  long z, s;              /* Zeilen- und Spaltenzaehler          */
  long Zeilenzahl, Spaltenzahl;
  int  i, Anzahl;
  int  Bedingung;         /* zur Zaehlung der Bedingungen a - d  */
  int  Aenderung;         /* Zaehler fuer die geloeschten Pixel  */
  int  Ende;
  int  P[11];             /* die 8 direkten Nachbarn             */

  ClearTextWindow(35,18,80,25);
  WriteText(35,18,"* Skelettierung Zhang/Suen *");
```

```
Spaltenzahl = Picture[BildNr].Spalten;
Zeilenzahl  = Picture[BildNr].Zeilen;

/* Iterativ verduennen nach modifiziertem Zhang/Suen-Algorith. */
Ende = FALSE;
do {
   Aenderung=0;
   for (z=1; z<(Zeilenzahl-1); z++) {
       for (s=1; s<(Spaltenzahl-1); s++) {
           if (Picture[BildNr].Bild[z][s]) {
              /* Nachbarn ermitteln. Zur leichteren Berechnung */
              P[2] = Picture[BildNr].Bild[z-1][s  ];
              P[3] = Picture[BildNr].Bild[z-1][s+1];
              P[4] = Picture[BildNr].Bild[z  ][s+1];
              P[5] = Picture[BildNr].Bild[z+1][s+1];
              P[6] = Picture[BildNr].Bild[z+1][s  ];
              P[7] = Picture[BildNr].Bild[z+1][s-1];
              P[8] = Picture[BildNr].Bild[z  ][s-1];
              P[9] = Picture[BildNr].Bild[z-1][s-1];
              P[10]=P[2];  /* Fuer 0-1-Uebergang von P[9]-P[2] */

              Bedingung=0;
              Anzahl   =0; /* Anzahl der weissen Nachbarn      */
              for (i=2; i<=9; i++)
                  if (P[i]) Anzahl++;

              /* diese erste Bedingung weicht von Zhang/Suen ab*/
              if ((3<=Anzahl) && (Anzahl<=6))   /* 3<=S(P1)<=6 */
                 Bedingung += 1;
              if (!(P[2] && P[4] && P[6]))      /* P2*P4*P6=0  */
                 Bedingung += 2;
              if (!(P[4] && P[6] && P[8]))      /* P4*P6*P8=0  */
                 Bedingung += 4;

              Anzahl=0;
              for (i=2; i<10; i++)
                  if (!P[i] && P[i+1]) Anzahl++;  /* 01-Muster */

              if (Anzahl==1) Bedingung += 8;
              /* Falls alle Bedingungen a-d erfuellt, markieren*/
              if (Bedingung==ERFUELLT) {
                 Picture[BildNr].Bild[z][s] = GRAU;/* Pixel mark.*/
                 Aenderung++;
              }
           }
       }
   }
   Ende = (Aenderung==0);
```

```
/* Nach einem Durchlauf nun alle markierten Pixel loeschen  */
for (z=0; z<Zeilenzahl; z++)
    for (s=0; s<Spaltenzahl; s++)
        if (Picture[BildNr].Bild[z][s]==GRAU)
           Picture[BildNr].Bild[z][s]=SCHWARZ;

/* Nun von rechts unten nach links oben Skelett berechnen   */
Aenderung=0;
for (z=(Zeilenzahl-2); z>=1; z--) {
    for (s=(Spaltenzahl-2); s>=1; s--) {
        if (Picture[BildNr].Bild[z][s]) {
           /* Nachbarn ermitteln. Zur leichteren Berechnung */
           P[2] = Picture[BildNr].Bild[z-1][s  ];
           P[3] = Picture[BildNr].Bild[z-1][s+1];
           P[4] = Picture[BildNr].Bild[z  ][s+1];
           P[5] = Picture[BildNr].Bild[z+1][s+1];
           P[6] = Picture[BildNr].Bild[z+1][s  ];
           P[7] = Picture[BildNr].Bild[z+1][s-1];
           P[8] = Picture[BildNr].Bild[z  ][s-1];
           P[9] = Picture[BildNr].Bild[z-1][s-1];
           P[10]=P[2];  /* Fuer 0-1-Uebergang von P[9]-P[2] */

           Bedingung=0;
           Anzahl   =0; /* Anzahl der weissen Nachbarn      */
           for (i=2; i<=9; i++)
               if (P[i]) Anzahl++;

           /* diese erste Bedingung weicht von Zhang/Suen ab*/
           if ((3<=Anzahl) && (Anzahl<=6))   /* 3<=S(P1)<=6 */
              Bedingung += 1;
           if (!(P[2] && P[4] && P[8]))      /* P2*P4*P8=0  */
              Bedingung += 2;
           if (!(P[2] && P[6] && P[8]))      /* P2*P6*P8=0  */
              Bedingung += 4;

           Anzahl=0;
           for (i=2; i<10; i++)
               if (!P[i] && P[i+1]) Anzahl++;  /* 01-Muster */

           if (Anzahl==1) Bedingung += 8;
           /* Falls alle Bedingungen a-d erfuellt, markieren*/
           if (Bedingung==ERFUELLT) {
              Picture[BildNr].Bild[z][s] = GRAU;/* Pixel mark.*/
              Aenderung++;
           }
        }
    }
}
Ende = (Ende && (Aenderung==0));
```

```
      /* Nach einem Durchlauf nun alle markierten Pixel loeschen  */
      for (z=0; z<Zeilenzahl; z++)
          for (s=0; s<Spaltenzahl; s++)
              if (Picture[BildNr].Bild[z][s]==GRAU)
                 Picture[BildNr].Bild[z][s]=SCHWARZ;

  } while (!Ende);      /* Ende erst, falls keine Aenderung mehr */
}
```

12.2.2 Skelettierung durch "Mittelachsenberechnung"

Im Prinzip stellt diese Überschrift eine Tautologie dar. Jedoch läßt sich damit der Ansatz von Zhang/Wang [SW88] zur Skelettierung am besten beschreiben.

Statt über entsprechende Masken direkt Pixel an dem Konturrand abzutragen, werden die einzelnen Bildpunkte in Abhängigkeit ihrer horizontalen und vertikalen Nachbarn gewichtet. Diese Gewichtung wird solange durchgeführt, bis ein Mindestwert, der bei jeder Iteration entsprechend erhöht wird, nicht mehr überschritten wird.

Im Initialisierungsschritt erhalten alle Objektpunkte den Wert 1. Der Mindestwert wird ebenfalls mit 1 vorbelegt. Jeder Objektpunkt wird dann durch die Summe der jeweiligen Werte der vier orthogonalen Nachbarn ersetzt. Nach dem kompletten Durchlauf wird der Mindestwert mit 4 multipliziert und geprüft, ob mindestens ein neu gewichteter Objektpunkt den Mindestwert überschreitet. Ist dies der Fall, so wird eine neue Gewichtung mit den bereits gewichteten Punkten berechnet. Der Mindestwert wird nach jedem Durchlauf auf das Vierfache vergrößert. Dies geschieht solange, bis dieses Minimum nicht mehr durch die gewichteten Objektpunkte erreicht werden kann.

Aufgrund dieser speziellen Vorgehensweise ergibt sich eine Gewichtung, in der die in etwa in der Mitte einer breiteren Kontur liegenden Bildpunkte am stärksten gewichtet sind.

Zur einfacheren weiteren Berechnung wird die Matrix mit den Gewichtungen in eine normierte Form umgerechnet, wo nur noch die Werte von 0, ..., 4 enthalten sind. Diese neue Matrix wird *Vergleichsmatrix* genannt. Je größer ein Punktwert in dieser Vergleichsmatrix ist, desto näher liegt der Punkt an der Mittelachse. Alle Punkte mit dem Wert 0 können sofort gelöscht werden. Danach werden alle Punkte mit dem Wert 1 untersucht und entfernt, wenn in ihrer direkten Umgebung noch größere Werte vorhanden sind, der Punkt kein Endpunkt ist und der Zusammenhang des Objektes bestehen bleibt. Diese Berechnung wird mit den Werten 2, 3 und 4 ebenfalls durchgeführt. Punkte mit dem Wert 4 sind fast immer Punkte der Mittelachse.

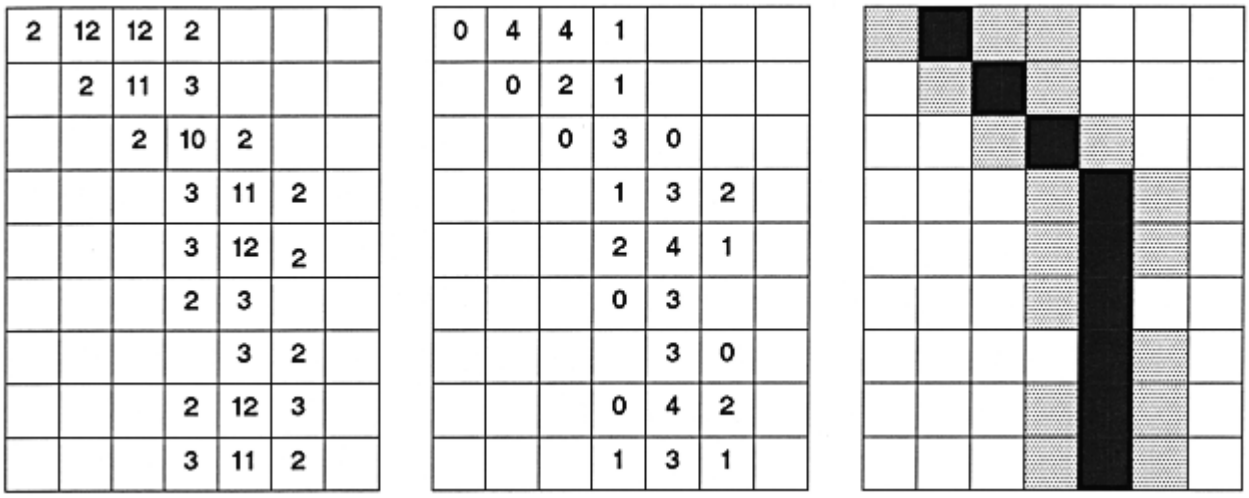


Abbildung 12.4: Das linke Bild zeigt einen Ausschnitt mit den aufsummierten Gewichten. In der Mitte ist die zugehörige Vergleichsmatrix und im rechten Bild das skelettierte Ergebnis zu sehen (aus [ZW88]).

```
void Mittelachsen_Thinning(int BildNr);
/* Ausschnitt aus dem Algorithmus zur Skelettierung durch die    */
/* ''Mittelachsenberechnung'' nach Zhang/Wang 1988               */
{
  int Matrix1[Z_RES_MAX][S_RES_MAX];    /* Vergleichsmatrix      */
  int Matrix2[Z_RES_MAX][S_RES_MAX];    /* Aufsummierte Gewichte */
  int Nachbar[8];                       /* 8 direkte Nachbarn    */
  int Gewicht, Maximum;
  long s,z;                             /* Zeilen-/Spaltenvar.   */
  int Zeilen, Spalten;                  /* Groesse des Bildes    */
  int k, Ende, Mittelachse;

  Zeilen = Picture[BildNr].Zeilen;
  Spalten= Picture[BildNr].Spalten;

  /* Zuerst die Matrix mit den Werten 0 und 1 erzeugen           */
  for (z=0; z<Zeilen; z++)
      for (s=0; s<Spalten; s++)
          Matrix2[z][s]=(Picture[BildNr].Bild[z][s]==WEISS) ? 1:0;
```

```
Ende = FALSE;     /* Kennzeichnung, dass Additionsmatrix ber.  */
Gewicht = 1;
while (!Ende) {
   for (z=0; z<Zeilen; z++) /* Ausgangsbild zuerst duplizieren */
       for (s=0; s<Spalten; s++)
           Matrix1[z][s] = Matrix2[z][s];

   /* aktuelles Gewicht ueber Summe der Nachbarn berechnen     */
   for (z=1; z<(Zeilen-1); z++)
       for (s=1; s<(Spalten-1); s++)
           if (Matrix1[z][s] == Gewicht)
              Matrix2[z][s] = Matrix1[z-1][s] + Matrix1[z][s+1] +
                              Matrix1[z+1][s] + Matrix1[z][s-1];
   Gewicht *= 4;  /* Gewicht nach Regel erhoehen               */
   Maximum = 0;
   for (z=0; z<Zeilen; z++)
       for (s=0; s<Spalten; s++)
           if (Matrix2[z][s] > Maximum)
              Maximum = Matrix2[z][s];
   Ende = (Maximum < Gewicht);     /* Ende-Bedingung erfuellt? */
}

/* Vergleichsmatrix ueber die Additionsmatrix berechnen        */
for (z=1; z<(Zeilen-1); z++)
   for (s=1; s<(Spalten-1); s++) {
      Nachbar[0]=Matrix2[z-1][s  ]; Nachbar[1]=Matrix2[z-1][s+1];
      Nachbar[2]=Matrix2[z  ][s+1]; Nachbar[3]=Matrix2[z+1][s+1];
      Nachbar[4]=Matrix2[z+1][s  ]; Nachbar[5]=Matrix2[z+1][s-1];
      Nachbar[6]=Matrix2[z  ][s-1]; Nachbar[7]=Matrix2[z-1][s-1];

      Mittelachse = TRUE;
      /* Koennte Punkt ein Mittelachsenpunkt sein ?            */
      for (k=0; k<4; k++) {
          if (!((Nachbar[k  ]< Matrix2[z][s]) &&
                 Nachbar[k+4]<=Matrix2[z][s]))  ||
               ((Nachbar[k  ]<=Matrix2[z][s]) &&
                 Nachbar[k+4]< Matrix2[z][s])) )
             Mittelachse=FALSE;

      Gewicht=0;
      for (k=0; k<4; k++) {
          if ((Nachbar[k  ]<=Matrix2[z][s]) &&
              (Nachbar[k+4]<=Matrix2[z][s])) Gewicht++;

      Matrix1[z][s] = (Mittelachse) ? Gewicht : 0;
   }
```

```
   /* Alle Punkte mit dem Gewicht 0 sind automatisch geloescht    */
   Gewicht = 1;
   for (z=1; z<(Zeilen-1); z++)
       for (s=1; s<(Spalten-1); s++)
           if (Matrix1[z][s] == Gewicht) {
              /* Loesche Punkt falls in der Umgebung (3x3) noch   */
              /* ein Punkt mit einem groesseren Gewicht existiert,*/
              /* der Punkt kein Endpunkt ist und der Zusammenhang */
              /* des Objekts erhalten bleibt.                     */
           }

   /* Zum Schluss Skelettpunkte wieder auf WEISS setzen           */
   for (z=0; z<Zeilen; z++)
      for (s=0; s<Spalten; s++)
         Pictur[BildNr].Bild[z][s]=(Matrix2[z][s]) ? WEISS:SCHWARZ;
}
```

Bleibt die Gewichtung in der Vergleichsmatrix ebenso wie die aufsummierten Werte der ersten Matrix erhalten, so kann daraus in etwa die ursprüngliche Breite der Objekte wieder berechnet werden. Ähnliches könnte auch mit der normalen iterativen Skelettierung rekonstruiert werden, sofern die Anzahl der Iterationen gespeichert wird.

```
   /* ... Rueckrechnung der urspruenglichen Dicke der Objektes    */
   /* Matrix1 = Vergleichsmatrix mit den Werten 1...4             */
   /* Matrix2 = Matrix mit den aufaddierten Gewichten             */
   for (z=0; z<Zeilen; z++) {
       Gewicht = 0;
       do {
          Gewicht++;
       while ( (pow(4,Gewicht)<= Matrix2[z][s]) &&
               (Matrix2[z][s] <= pow(4,(Gewicht+1))) );

       /* War der Punkt laut skelettierte Vergleichsmatrix ein    */
       /* Skelettpunkt?                                           */
       if (Matrix1[z][s] == 0)
          Picture[BildNr].Bild[z][s] = SCHWARZ;
       else {                              /* ... dann expandieren */
          for (k=1; k<=Gewicht; k++) {
              Picture[BildNr].Bild[z+k][s  ] = WEISS;
              Picture[BildNr].Bild[z-k][s  ] = WEISS;
              Picture[BildNr].Bild[z  ][s+k] = WEISS;
              Picture[BildNr].Bild[z  ][s-k] = WEISS;
          }
       }
   }
```

12.2.3 Skelettierung mit Hilfe der Euler-Charakteristik

In dem Algorithmus von Neusius/Olszewski [NO90] werden die Bildpunkte als rechteckige Flächen aufgefaßt und die Euler-Charakteristik wird in einer 3×3-Maske berechnet.

Die Euler-Charakteristik faßt topologische Merkmale von Körpern oder Flächen in einer ganzen Zahl zusammen. Im Eulerschen Polyedersatz für dreidimensionale konvexe Körper ist die Zahl eine Konstante:

$$2 = e - k + f$$

mit e = Anzahl der Ecken, k = Anzahl der Kanten und f = Anzahl der Flächen.

Bei der Skelettierung von zweidimensionalen Bildern gibt die Euler-Charakteristik E die Anzahl der zusammenhängenden Bildobjekte innerhalb der 3×3-Maske an. Sie berechnet sich nach

$$E = e - k + f$$

mit e = Anzahl der Ecken der Objektpunkte, k = Anzahl der Kanten der Objektpunkte und f = Anzahl der Objektpunkte. Gemeinsame Ecken oder gemeinsame Kanten von Objektpunkten werden in dieser Formel nur einmal gezählt.

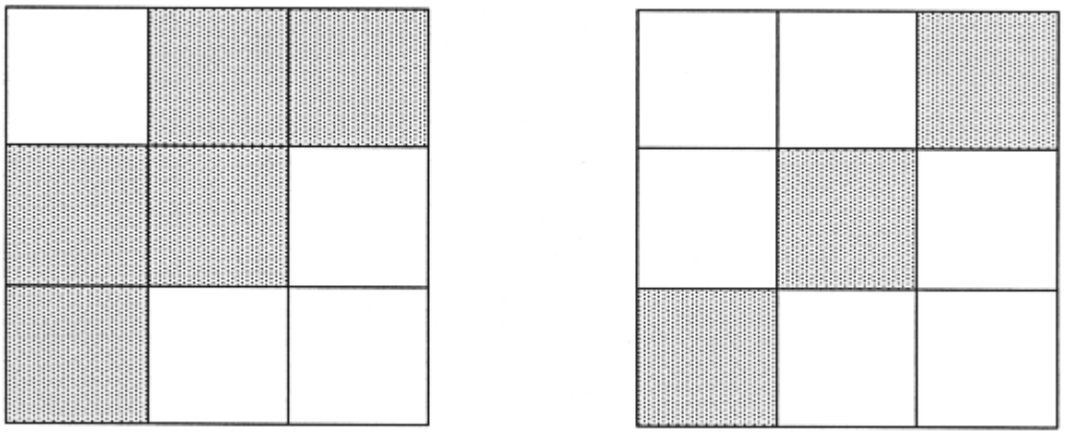


Abbildung 12.5 Die Euler-Charakteristik des linken Bildes ist $E = 1$, auch wenn der mittlere Bildpunkt entfernt wird. Im rechten Beispiel ist der Mittelpunkt der Maske ein Skelettpunkt. Die Euler-Charakteristik ändert sich daher nach dem Entfernen des Bildpunktes von $E = 1$ auf $E = 2$.

Der Mittelpunkt in einer 3x3-Maske kann entfernt werden, wenn die Euler-Charakteristik nach dem Entfernen mit E=1 unverändert bleibt. Damit eine Verschiebung des Skeletts zum Objektrand vermieden wird, kann auch bei diesem Verfahren wieder eine abwechselnde Bearbeitung des Bildes von links oben nach rechts unten und umgekehrt erfolgen. In [NO90] wird noch eine andere Variante beschrieben. Dort werden abwechselnd in dem einen Durchgang alle Punkte mit gerader Zeilen-/Spaltenkoordinate und im anderen mit ungerader bearbeitet. Auch mit diesem Verfahren bleibt der topologische Zusammenhang erhalten.

Beendet wird das Verfahren, wenn nur noch Skelettpunkte oder Endpunkte existieren. Ein Bildpunkt ist Endpunkt, wenn er nur einen Nachbarn besitzt.

12.3 Kontur-Folge-Verfahren

Bei den Kontur-Folge-Verfahren [Kwo88], [Xia88], [Ji89] wird nicht jeder Bildpunkt auf die Eigenschaft hin untersucht, "Skelettpunkt" zu sein, sondern es werden in einem ersten Schritt (Initialisierungsschritt) alle Randpunkte eines jeden Objektes bestimmt. Nur diese Randpunkte werden dann auf ihre Skelett-Eigenschaft hin untersucht. Ist ein Punkt kein Skelettpunkt, so wird er aus der Liste der Randpunkte entfernt und ein oder mehrere Nachbarpunkte können dadurch neue Randpunkte werden.

Die Kontur-Folge-Verfahren sind bei steigender Dicke der Objekte wesentlich schneller als die Verfahren mit den 3×3-Masken. Jedoch können sie nicht parallel, sondern nur sequentiell durchgeführt werden.

12.3.1 Das Verfahren von Ji

Das Verfahren von Ji [Ji89] gehört zu den Kontur-Folge-Verfahren. Im Initialisierungsschritt werden zuerst alle Randpunkte in einem Puffer zwischengespeichert. Randpunkte sind solche Objektpunkte, die in der direkten Nachbarschaft einen oder mehrere Hintergrundpunkte haben (3×3-Maske). Im folgenden werden dann mit Hilfe von 3×3-Masken nur noch diese Konturpunkte daraufhin untersucht, ob es sich um Skelettpunkte handelt. Die gefundenen Skelettpunkte werden in einem Ergebnisbild abgelegt, die anderen Randpunkte gelöscht und deren Nachbarschaft daraufhin getestet, ob neue Randpunkte entstanden sind. Neue Randpunkt werden dann in den Puffer eingetragen.

In einer Erweiterung des Algorithmus [KF90] werden folgende Masken zur Bestimmung der Skelettpunkte aus der Liste der Randpunkte bei horizontalen, vertikalen und diagonalen Verläufen verwendet:

x	0	x
1	p	1
x	0	x

x	1	x
0	p	0
x	1	x

x	0	1
x	p	0
x	x	x

x	x	x
x	p	0
x	0	1

x	x	x
0	p	x
1	0	x

1	0	x
0	p	x
x	x	x

und für Endpunkte die Masken

0	1	0
0	p	0
0	0	0

0	0	0
0	p	1
0	0	0

0	0	0
0	p	0
0	1	0

0	0	0
1	p	0
0	0	0

wobei x in diesen Masken bedeutet, daß der Punkt nicht von Interesse ist, bei 1 handelt es sich um einen Randpunkt und bei 0 um einen Hintergrundpunkt.

Stimmt die Umgebung des Randpunktes p mit einer dieser Masken überein, so stellt p einen Skelettpunkt dar und darf nicht entfernt werden.

Durch diese Art der Definition von Rand- und Skelettpunkten wird verhindert, daß sich das Skelett einer Kontur nach einer Seite (z.B. Abtastrichtung) hin verschiebt.

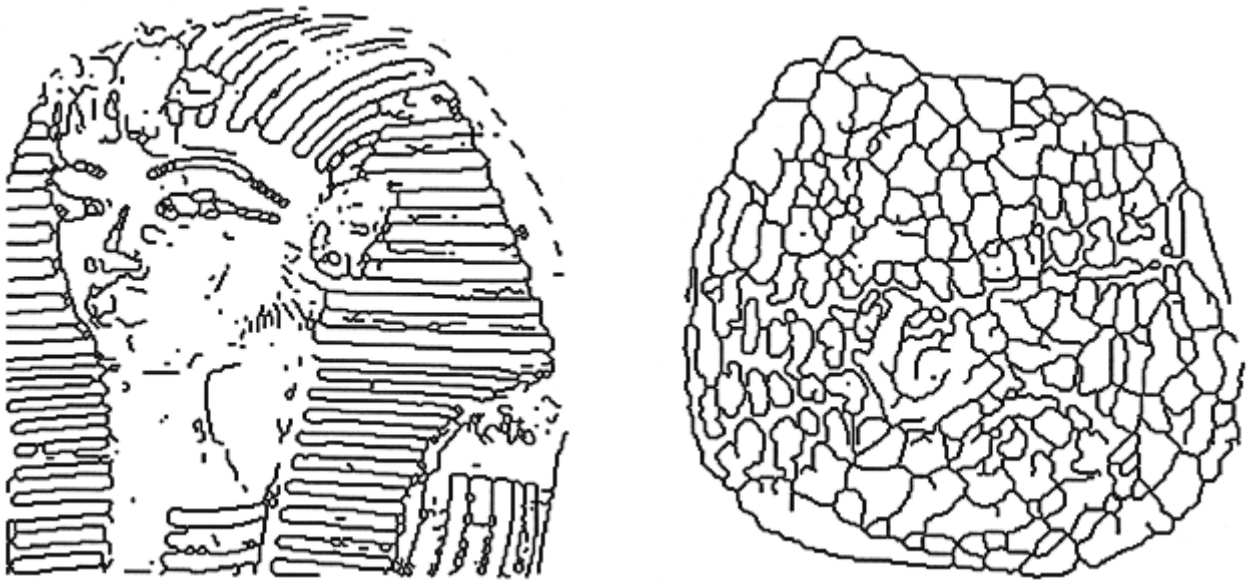


Abbildung 12.6: Ergebnisse der Skelettierung nach dem Verfahren von Ji. An beiden Bildern wurde eine Kantendetektion mit dem Prewitt-Operator vorgenommen und das Binärbild anschließend skelettiert. Das linke Bild zeigt die Totenmaske des Tutanchamun. Für das rechte Bild ist die Aufnahme der Kalkschale einer Ostrakode (Muschelkrebs) verwendet worden (vergleiche Abbildung 14.5). Muschelkrebse spielen als Leitfossilien in der Paläontologie eine besondere Rolle. Die maschenartigen Muster auf der Oberfläche dienen dabei zur Identifizierung [Kol91].

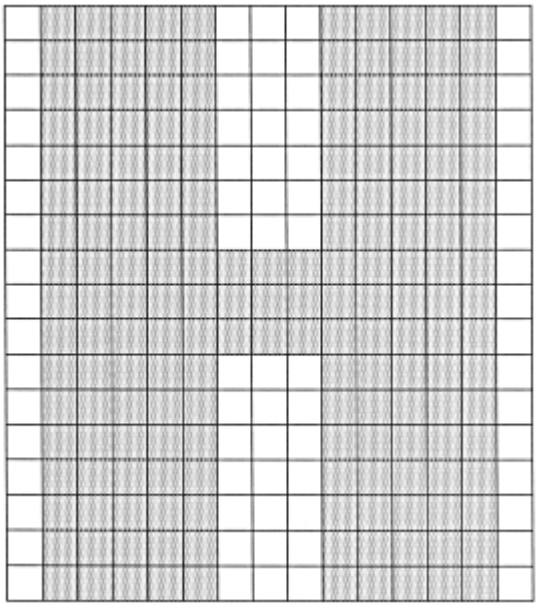
Aufgaben

Aufgabe 1

Wie sieht das ideale Skelett eines Quadrats und eines Kreises aus?

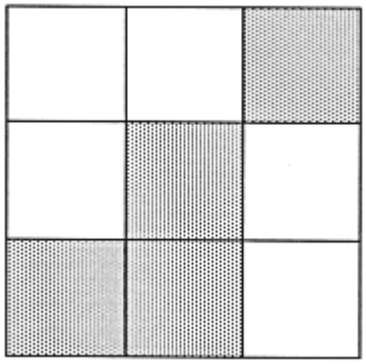
Aufgabe 2

Um wieviel Bildpunkte wird der nachfolgend dargestellte Buchstabe "H" bei der Skelettierung mit 3×3-Masken nach dem Verfahren von Zhang/Suen verkürzt?



Aufgabe 3

Berechnen Sie die Euler-Charakteristik *E* des folgenden Bildes (siehe Abschnitt 12.2.3). Wie ändert sich das Ergebnis nach dem Entfernen des mittleren Punktes? Handelt es sich dabei um einen Skelettpunkt?



13. Vektorisierung

Durch Kantendetektion und Skelettierung der berechneten Kanten werden die Bildinhalte auf die wichtigsten Informationen beschränkt. Trotzdem ist die reduzierte Menge an relevanten Bildpunkten noch zu groß, um eine direkte Bildinterpretation oder Mustererkennung durchzuführen. Die gesuchten Strukturen müssen daher erst mit geeigneten Masken (Templates) extrahiert (siehe Kapitel 14) oder durch Zusammenfassung der Bildpunkte zu komplexeren Objekten (Strecke, Rechteck, Kreis usw.) gebildet werden.

Für den Aufbau solcher geometrischer Primitive aus den binären Bildpunkten ist die *Vektorisierung* ein wichtiger Schritt. Hierbei werden Punkte, die auf einer möglichen Gerade liegen zu einer Strecke bzw. einem Vektor mit Anfangs- und Endpunkt zusammengefaßt. Das resultierende Vektorbild kann zur Suche nach komplexeren Strukturen weiter verwendet oder beliebig transformiert werden.

Die Erzeugung der Vektorinformation kann auf unterschiedliche Art und Weise geschehen. Als Gütekriterien seien hier genannt

- schnelle und einfache Berechnung,
- hohe Genauigkeit,
- möglichst lange resultierende Vektoren und
- Unempfindlichkeit gegenüber kleinen Störungen.

Zwei unterschiedliche Verfahren werden im folgenden beschrieben, wobei das erste nicht direkt eine Liste von Vektoren aus den Bildpunkten erzeugt.

13.1 Der Freeman-Kettenkode

Bei dem Kettenkode nach Freeman [FD77] wird nicht direkt eine Liste von Vektoren erzeugt, sondern die Konturen von Objekten werden beschrieben. Dabei wird ausgenutzt, daß in dem Bildpunktraster jeder Punkt nur 8 direkte Nachbarn besitzt und eine Strecke nur in eine dieser 8 Richtungen verlaufen kann. Die Richtungen werden mit den Zahlen 0-7 kodiert (siehe Abbildung 13.1).

Um den Kettenkode für ein Objekt aufzubauen, wird die Vorlage zeilenweise nach dem ersten Objektpunkt durchsucht. Die Nachbarschaft dieses Punktes wird dann getestet und die Richtung, in der der nächste Nachbar liegt, mit der zugehörigen Richtungszahl kodiert. So beschreibt z.B. die Kette 000066444422 ein zu den Rändern des Bildes paralleles Rechteck.

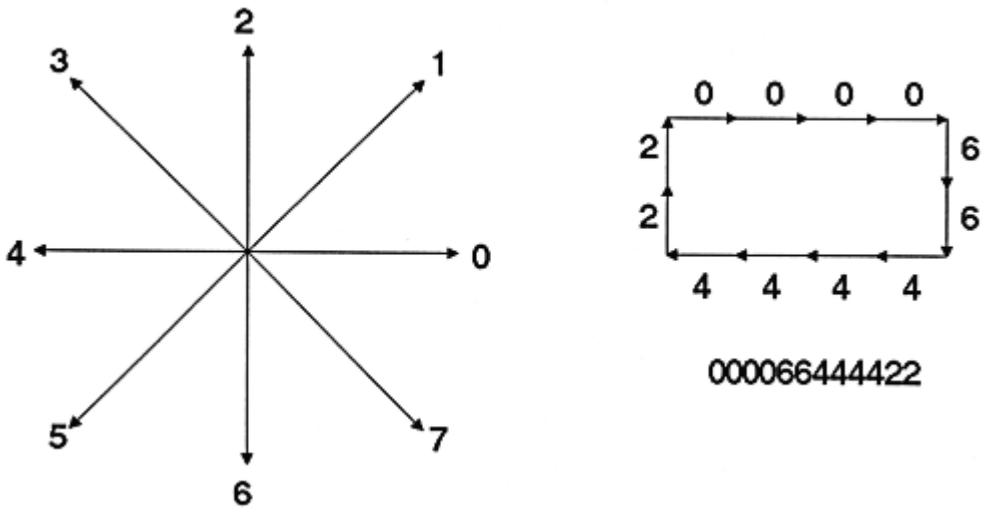


Abbildung 13.1: Die 8 möglichen Richtungen werden mit den Zahlen 0-7 kodiert (links). Durch entsprechende Abtastung der Umrisse von Objekten wird eine Zahlenfolge erzeugt, die die Objektkontur beschreibt (rechts).

Wird bei der Abtastung wieder der Ausgangspunkt erreicht, so ist die Kontur geschlossen. Um alle Richtungsänderungen durch den Kettenkode korrekt zu erfassen, muß das Bild nach einem bestimmten Schema abgetastet werden. Wird das Bild zeilenweise von links oben nach rechts unten hin untersucht, sollten die Objekte im mathematisch negativen Sinn abgetastet werden, damit keine Bildpunkte übersehen werden. Es sind dann nicht mehr alle 8 Richtungen zu untersuchen, sondern maximal noch 6, da die anderen beiden durch die Vorgehensweise bei der Abtastung schon bearbeitet worden sind. Die verbleibenden 6 Richtungen müssen im selben Umlaufsinn wie die Objektabtastung kontrolliert werden. Damit wird jeder mögliche Richtungswechsel korrekt erfaßt (siehe Abbildung 13.2).

Um aus diesen Zahlenkodes Vektoren zu erzeugen, müssen gleiche aufeinanderfolgende Richtungen einfach zusammengefaßt und Anfangs- und Endpunktkoordinaten gespeichert werden. Damit ist eine Vektorisierung der Objektkonturen erreicht.

Aber schon der Kettenkode alleine enthält eine große Informationsmenge. So kann durch das dominante Vorkommen zweier entgegengesetzter Richtungen die Hauptausdehnungsrichtung eines Objektes grob bestimmt werden. Eine Transformation in eine normierte Lage ist somit einfach möglich. Sind alle Richtungen etwa gleichhäufig, so ist das Objekt annähernd kreisförmig. Kommen im wesentlichen nur 4 Richtungen (zwei Richtungen und ihre entgegengesetzten Richtungen) vor, so ist die Form des Objekts in etwa rechteckig. Die Lage kann über die vorkommenden Richtungen einfach ermittelt werden. Ist die Differenz zweier aufeinanderfolgender Richtungszahlen Modulo 8 gleich 2, so existiert an dieser Stelle im Objekt ein rechter Winkel. Einzelne Ausreißer (Störungen) lassen sich ebenso einfach ermitteln und damit auch beseitigen. Damit ist auch gleichzeitig eine Abstraktion des durch den Kettenkode beschriebenen Objektes möglich.

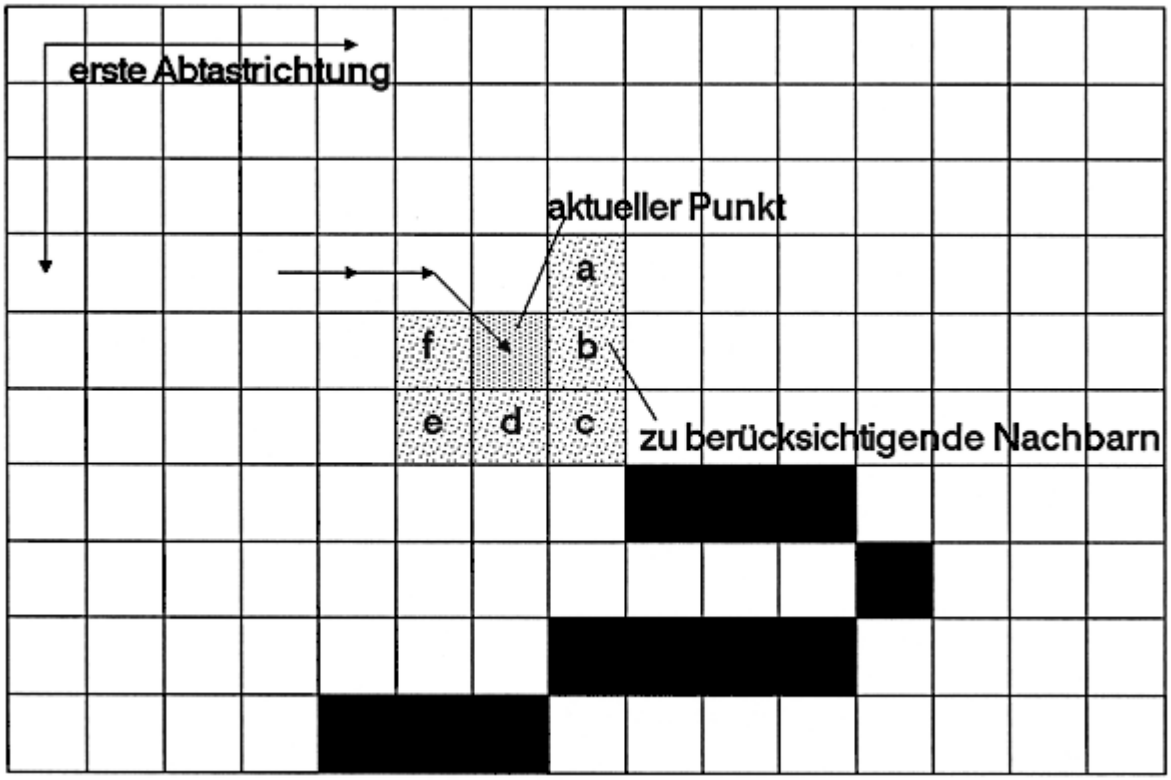


Abbildung 13.2: Wird das Bild bis zum ersten Objektpunkt zeilenweise abgetastet, so müssen für die Erzeugung des Kettenkodes nicht alle Richtungen untersucht werden. Es reicht aus, wenn nur die Nachbarn im Umlaufsinn der Abtastung, ausgehend von der senkrechten Richtung zur aktuellen Richtung untersucht werden. In den Beispielen sind dies jeweils die Nachbarn a, b, ..., f.

Eine weitere Vereinfachung ist durch die Berücksichtigung von nur 4 Richtungen möglich. Dies kann dann interessant sein, wenn im Originalbild, bis auf Störungen, nur diese 4 Richtungen vorkommen.

Im *differentiellen Kettenkode* wird nicht jede Richtung, sondern nur die Richtungsänderung abgespeichert. Dazu sind zwar auch 8 Richtungen zu berücksichtigen, diese sind aber in der Praxis nicht gleichverteilt. Die Richtungsänderungen 0 und ±1 kommen wesentlich häufiger vor als alle anderen. Es liegt daher nahe, diese Richtungen speziell zu kodieren, um insgesamt weniger Speicherplatz zu belegen (vergleiche Abschnitt 15.2). Das Resultat ist der *variable Kettenkode*, bei dem z.B. die Richtungsänderung mit 0=0, +1=01, -1=011, +2=0111, ... kodiert werden kann [Pav82].

Für die Erzeugung des Kettenkodes muß das Objekt nicht unbedingt skelettiert sein. Durch die spezielle Vorgehensweise bei der Abtastung wird immer nur der äußere Rand betrachtet. Der Kettenkode umschließt dann das Objekt komplett. Die umschlossene Fläche muß dann entsprechend markiert werden, damit sie nicht bei der weiteren Abtastung berücksichtigt wird.

Ein Kettenkode ist beendet, wenn entweder der Ausgangspunkt wieder erreicht wurde (geschlossene Kontur) oder kein Bildpunkt in der Nachbarschaft liegt, der nicht schon berücksichtigt wurde. Es ist daher wichtig, alle abgetasteten Bildpunkte entsprechend zu markieren.

13.2 Direkte Vektorisierung

Bei einer direkten Vektorisierung muß die Vorlage ebenfalls abgetastet und müssen benachbarte Bildpunkte daraufhin untersucht werden, ob sie auf einer möglichen Geraden liegen. Bei den meisten Verfahren wird diese Menge von Punkten mit Hilfe der linearen Regression (Methode der kleinsten Fehlerquadrate) oder über den Euklidschen Abstand berechnet [Kre77], [SH79], [CF*84], [KFJ85], [Par88]. Der Rechenaufwand bei diesen Verfahren ist relativ hoch.

In dem hier vorgestellten Verfahren [Ste89] wird daher ein anderer Weg beschritten. Geht man im ersten Schritt davon aus, daß die zu vektorisierende Vorlage aus einer mit dem Computer erstellten Strichzeichnung stammt, so sind die enthaltenen Linien z.B. mit dem Bresenham-Algorithmus oder dem DDA (digital differential analyzer) erzeugt worden [NS81].

Der Gedanke liegt daher nahe, das umgekehrte Verfahren zur Vektorisierung zu verwenden. Wird dieser Algorithmus dann auf reale, skelettierte Binärbilder angewendet, so zeigt es sich, daß diese Vorgehensweise auch bei diesen Vorlagen sinnvoll ist.

13.2.1 Der Algorithmus

Das Kernstück dieses Vektorisierungsverfahrens ist der Bresenham-Algorithmus zum Zeichnen von Vektoren, der hier in einer leicht modifizierten Form zum Vektorisieren verwendet wird.

Es gilt dabei, das Objekt möglichst genau durch Vektoren darzustellen und gleichzeitig möglichst lange Vektoren zu erhalten. Die maximale Genauigkeit wird erreicht, wenn immer nur zwei benachbarte Bildpunkte zu einem Vektor zusammengefaßt werden. Die Vektoren mit zwei Punkten Länge stellen aber weder eine Abstraktion des Objektes noch eine Speicherersparnis dar, da ja für jeden Vektor Anfangs- und Endpunktkoordinaten gespeichert werden müssen.

Daher ist es notwendig eine Ungenauigkeit bei der Vektorisierung zu tolerieren. Wie groß dieser Fehler ist, wird bei diesem Verfahren durch den Fehlerterm aus dem Bresenham-Algorithmus bestimmt.

Die Vorgehensweise kann algorithmisch wie folgt beschrieben werden (das detaillierte Listing ist aufgrund der Länge hier nicht abgedruckt):

```
/* Die am rechten Rand in Klammern befindlichen Zahlen dienen    */
/* zur Referenzierung im Text.                                   */

void Vektorisierung(Bild)
{
  Taste Bild komplett ab
  if (ein Pixel==Kantenfarbe) {
    Suche alle Nachbarn;                                        (1)
```

```
    for (alle Nachbarn) {
      Berechne mit jeweils zwei Punkten eine Hauptrichtung;
      Suche die Nachbarn der Nachbarn in Hauptrichtung (auch
          schon markierte Punkte sind als Nachbarn moeglich);    (2)
      for (alle Nachbarn der Nachbarn) {
        Stelle fuer jeweils 3 erhaltene Punkte eine Hypothese
            fuer eine Gerade auf (Bresenham);
        while (diese Gerade erweiterbar ist) {
          Versuche aus dieser Geradendefinition den naechsten
              Punkt (x,y) zu berechnen;
          if (dieser Punkt (x,y) existiert) {
            Erweitere Gerade;
            Markiere Punkt;
            Aktualisiere Fehlerterm (Bresenham);               (3)
          }
          else {
            Suche Nachbarpunkte in Hauptrichtung der
                bisherigen Geraden;                            (4)
            if (es einen Nachbarpunkt gibt, so dass die neue
                Gerade < EPSILON von der alten abweicht) {
              Erweitere Gerade;
              Markiere Punkt;                                  (5)
            }
            else {
              Gerade ist nicht erweiterbar;                    (6)
              Speichere Anfangs- und Endpunkt;
            }
          }
        }
      }
    }
  }
}
```

Wie man an diesem Algorithmus erkennt, werden alle Bildpunkte berücksichtigt. Einzelne Punkte (die meist nur eine Störung/Rauschen darstellen) werden automatisch herausgefiltert. Auch 2-Punkte lange Strecken gehen verloren, da mit zwei Punkten noch keine mit Hilfe des Bresenham-Algorithmus gezeichnete Strecke in ihrer Richtung genau genug bestimmbar ist (siehe Tabelle 13.1 und Abbildung 13.3).

Ein Punkt kann Startpunkt für mehrere Strecken sein. Aus diesem Grunde werden alle Nachbarn (1) und deren Nachbarn (2) betrachtet, was hier schon eine Hauptrichtung festlegt (durch die 3-Punkte lange Strecke). Eine somit einmal festgelegte Richtung kann nicht mehr geändert werden. Sollte daher ausgerechnet der Anfangspunkt gestört sein (siehe Abbildung 13.4), so weicht der erhaltene Vektor vom ursprünglichen ab, liegt aber immer noch innerhalb der Toleranz (1-Punkte Umgebung).

Länge in Punkten	Maximaler Öffnungswinkel in Grad
2	53.13°
3	36.87°
4	28.07°
5	22.62°
6	18.92°
7	16.26°
8	14.25°
13	9.53°
24	4.98°
59	1.98°
78	1.49°

Tabelle 13.1: Maximaler Öffnungswinkel bei Bresenham-Strecken in Abhängigkeit von der Länge (vergleiche Abbildung 13.3).

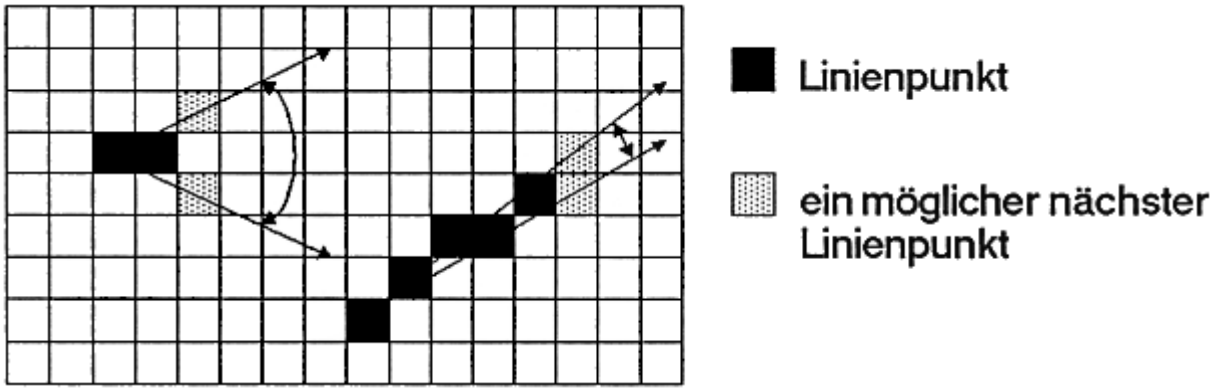


Abbildung 13.3: Der Öffnungswinkel bei Strecken, die mit Hilfe des Bresenham-Algorithmus erzeugt werden.

Diese Verschiebung tritt am stärksten bei horizontalen oder vertikalen Strecken auf, bei denen der erste Punkt etwas versetzt liegt (Abbildung 13.4). Für den Einsatz bei speziellen Vorlagen, wie z.B. technische Zeichnungen o.ä. mit vielen horizontalen/vertikalen Strecken, kann noch die Steigung fast horizontaler/vertikaler Strecken näher untersucht werden, um gegebenenfalls den Anfangspunkt zu korrigieren und somit Verschiebungen zu eliminieren. Dies geschieht einfach durch Fehlerbetrachtung und Vergleich der Strecke, die durch die normale Abtastrichtung erhalten worden ist, mit der, die man durch Abtastung vom Endpunkt her (falls dieser korrekt ist) bekommt.

Wie man in Abbildung 13.4 sieht, entsteht kein Unterschied zwischen Original und Vektorisierung mehr, falls mehr als ein Punkt am Anfang der Strecke von der Hauptrichtung abweicht. Am Ende müssen es mehr als zwei Punkte sein. Die entsprechende Figur wird dann durch mehrere Vektoren repräsentiert.

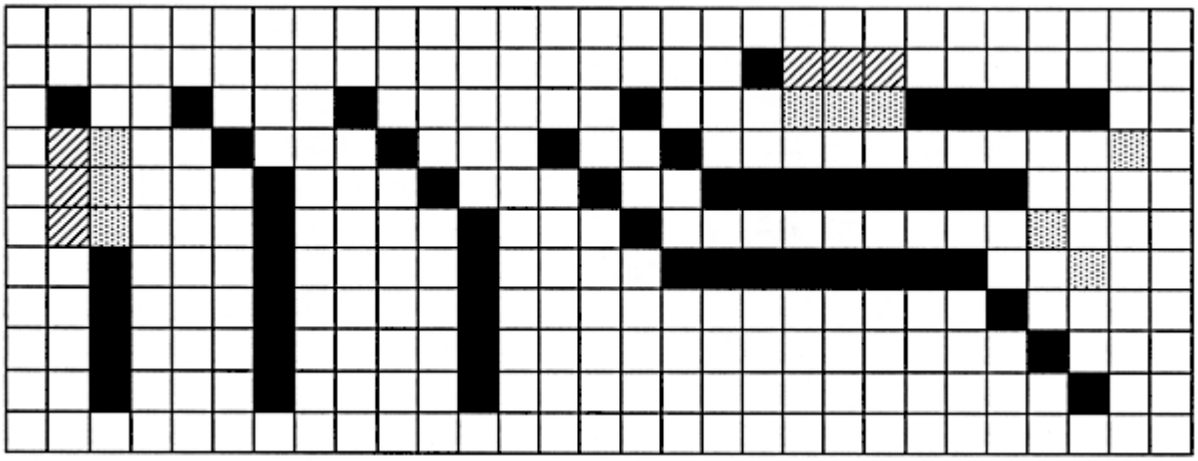


Abbildung 13.4: Ergebnis des Verfahrens bei vertikalen und horizontalen Strecken mit gestörtem Anfangspunkt.

= Übereinstimmung Original-Vektorisierung.

= Neuer Punkt durch die Vektorisierung.

= Alter Punkt, der aber durch die neuen Vektoren nicht mehr gezeichnet wird.

Kann eine 3-Punkte-Strecke verlängert werden ((3) oder (4)), so wird die Hauptrichtung neu (genauer) berechnet und es wird versucht, diese neue Strecke zu erweitern. Dies geschieht solange, bis entweder kein Verlängerungspunkt mehr vorhanden ist, oder bis die Strecke, die durch Hinzunahme des neuen Punktes entsteht, an irgendeiner Stelle um mehr als einen Punkt von den Originalbildpunkten abweicht (6).

Jeder betrachtete Punkt wird markiert (5) und kann für andere Geraden weiter verwendet werden. Damit werden Lücken bei sich schneidenden Geraden vermieden. Jedoch darf der Anfangspunkt eines Vektors noch nicht für einen anderen Vektor verwendet worden sein.

13.2.2 Fehlerbetrachtung

Die Gründe für die Verwendung des Bresenham-Algorithmus zur Vektorisierung sind klar. Das Verfahren ist seit langem bekannt, wird vielfältig eingesetzt und kann schnell und einfach berechnet werden [NS81], [FvD84].

Normalerweise bestimmt eine Fehlergröße e, welcher Punkt als nächster mit dem Algorithmus betrachtet werden soll. Im Bresenham-Algorithmus wird daher bei jeder Iteration die Liniensteigung $\Delta y/\Delta x$ (o.B.d.A. $\Delta x \geq \Delta y$) zu der Fehlergröße addiert. Zuvor muß über das Vorzeichen von e die Rechenrichtung der y-Koordinate des laufenden Punktes festgelegt werden. Ein e mit positivem Vorzeichen bedeutet, daß der Linienverlauf über dem aktuellen Punkt liegt; somit wird die y-Koordinate erhöht und e um eins dekrementiert. Ist e negativ, so bleibt die y-Koordinate unverändert.

Die x-Koordinate wird mit jedem Schritt inkrementiert. Der Abstand, hier also die vertikale Entfernung zwischen dem Linienverlauf und gewähltem Punkt, ist daher immer kleiner oder gleich 0.5.

Für den Fehler e ergeben sich daher:

Initialisierung: $e = \Delta y/\Delta x - 0.5$

Rechenschritt 1: $e = e - 1$

Rechenschritt 2: $e = e + \Delta y/\Delta x$

Durch Multiplikation mit 2 und Δx erhält man folgende Ganzzahl-Rechnungen ($e' = 2\ e\ \Delta x$):

Initialisierung: $e' = 2\Delta y - \Delta x$

Rechenschritt 1: $e' = e' - 2\Delta x$

Rechenschritt 2: $e' = e' + 2\Delta y$

Falls der Algorithmus so formuliert wurde, daß sich Rechenschritt 1 und 2 gegenseitig ausschließen, so lautet der Fehler e' in Rechenschritt 1 mit Berücksichtigung der Größe aus Rechenschritt 2

$$e' = e' - 2\Delta x + 2\Delta y$$

Die Fehlergröße e' kann nun in einem Intervall von $2\Delta x$ liegen, wobei die untere und obere Grenze durch die Steigung der jeweiligen Strecke aus der Initialisierung festgelegt werden.

Steigung	Intervall von e'
Steigung = 0	$e' = -\Delta x$
0 < Steigung < 1	$2(\Delta y - \Delta x) < e' < 2\Delta y$
Steigung = 1	$e' = \Delta x$

Tabelle 13.2: Maximales Intervall des Fehlers e' in Abhängigkeit der Steigung.

Die Schwierigkeit beim Vektorisieren mit Hilfe des Bresenham-Algorithmus liegt nun darin, daß die endgültige Länge der Strecke während der Linienverfolgung noch unbekannt ist. Der Fehlerterm e' muß somit immer dann aktualisiert werden, wenn die Gerade normal erweiterbar ist, d.h. ein Punkt, der nach dem Bresenham-Algorithmus berechnet wurde, auch wirklich als Fortsetzungspunkt der momentanen Strecke im Bild vorhanden ist. Der Fehler zu jedem Punkt wird in einem Fehlervektor abgespeichert. Bei einer möglichen Erweiterung werden die bisherigen Fehlergrößen nicht neu berechnet, sondern es wird nur ab dem neuen Punkt der neue Fehler verwendet. Durch das Abbruchkriterium (siehe unten) ist sichergestellt, daß auch im Anfangsbereich die ermittelte Strecke nicht mehr als ein Punkt von den Originalpunkten abweicht.

Konnte eine Strecke nicht normal nach dem Bresenham-Algorithmus erweitert, sondern mußte ein Nachbarpunkt verwendet werden, dann wird der Fehlerterm e' nicht mehr aktualisiert; Δx bleibt auch unverändert. Der Fehlervektor dieser vollkommen neuen Strecke wird mit dem bisherigen verglichen. Ist die Summe des Fehlers jeweils entsprechender Punkte (die Anzahl der Punkte pro

Strecke ist gleich bzw. um eins größer, wenn sich die Strecken nur um einen Punkt im Endpunkt unterscheiden, da sie von der Größe Δx bestimmt wird. Es gilt: *Anzahl Punkte/Strecke* = Δx + 1) kleiner als $2\Delta x_{alt}$ (vergleiche oben), so weicht die neue Strecke nicht mehr als 0.5 Punkte von der Bresenham-Strecke ab und somit maximal ein Punkt von den Originalbildpunkten. Je nach Strecke kann dadurch eine Bresenham-Strecke durch Suche von Nachbarpunkten mindestens auf die doppelte Länge vergrößert werden, bis das Abbruchkriterium

$$| e'_{alt} + e'_{neu} | \leq | \Delta x_{alt} |$$

in Kraft tritt.

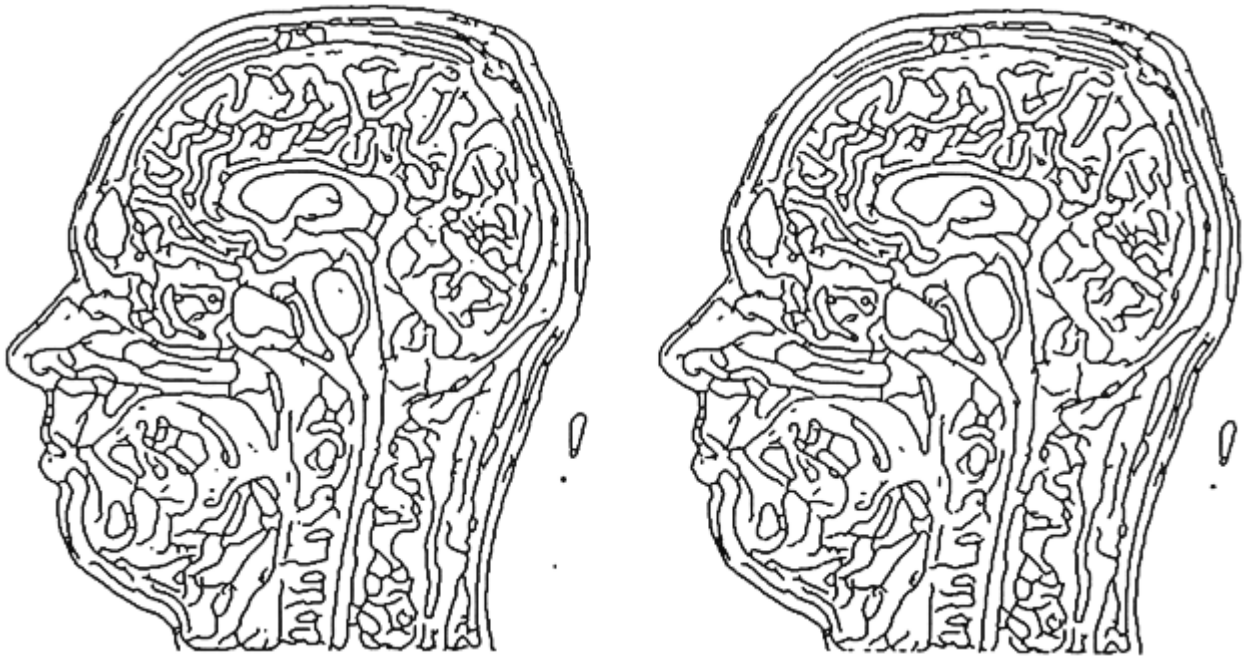


Abbildung 13.5: Originalbild (links, schon skelettiert) und Ergebnis der Vektorisierung. Zu beachten ist das Löschen punktueller Störungen und die glättende Wirkung.

Sollen Strecken genauer vektorisiert werden, so ist der zulässige Fehler zu verringern (z.B. auf 1.5 Δx oder Δx). Dadurch erhält man natürlich mehr kurze Vektoren. Genauso kann man auf der anderen Seite durch Vergrößerung des zulässigen Fehlers eine weitere Glättung erreichen. Die Vektoren weichen dann aber in einigen Fällen mehr als ein Punkt von den Originalpunkten ab.

Durch die anfänglich ermittelten zwei bzw. drei Startpunkte ist die grobe Richtung der Strecke festgelegt. Jedoch sind noch Abweichungen von einem Punkt (keine Richtungsänderungen!) zu der Idealstrecke erlaubt. In solchen Fällen wirkt sich das hier vorgestellte Vektorisierungsverfahren positiv aus, da seine glättende Wirkung zum Tragen kommt.

Dadurch, daß gewisse Schwankungen der Strecken/Punkte erlaubt sind und das Verfahren trotzdem nicht abbricht, sind die erhaltenen Vektoren im Vergleich zu anderen Verfahren [Par88] relativ lang. Bei Tests mit den unterschiedlichsten Bildvorlagen waren die Vektoren durchschnittlich mehr als 5-8 Bildpunkte lang. Dies wirkt sich natürlich direkt auf die erhaltene Anzahl von Vektoren und

die Speicheranforderung aus. Auch die nachfolgende Bearbeitung kann sich vereinfachen, wenn weniger, dafür aber längere Vektoren bei trotzdem hoher Genauigkeit zu verarbeiten sind.

Ebenfalls positiv für die Weiterverarbeitung ist die automatische Vorsortierung der Vektoren, die man durch die systematische Abtastrichtung (von links nach rechts, von oben nach unten) erhält. Befindet sich der Punkt (0,0) links oben im Bild und wird mit x die Spaltenzahl, mit y die Zeilenzahl bezeichnet, so sind die Ergebnisvektoren in lexikographisch aufsteigender Reihenfolge bezüglich des Anfangspunktes sortiert.

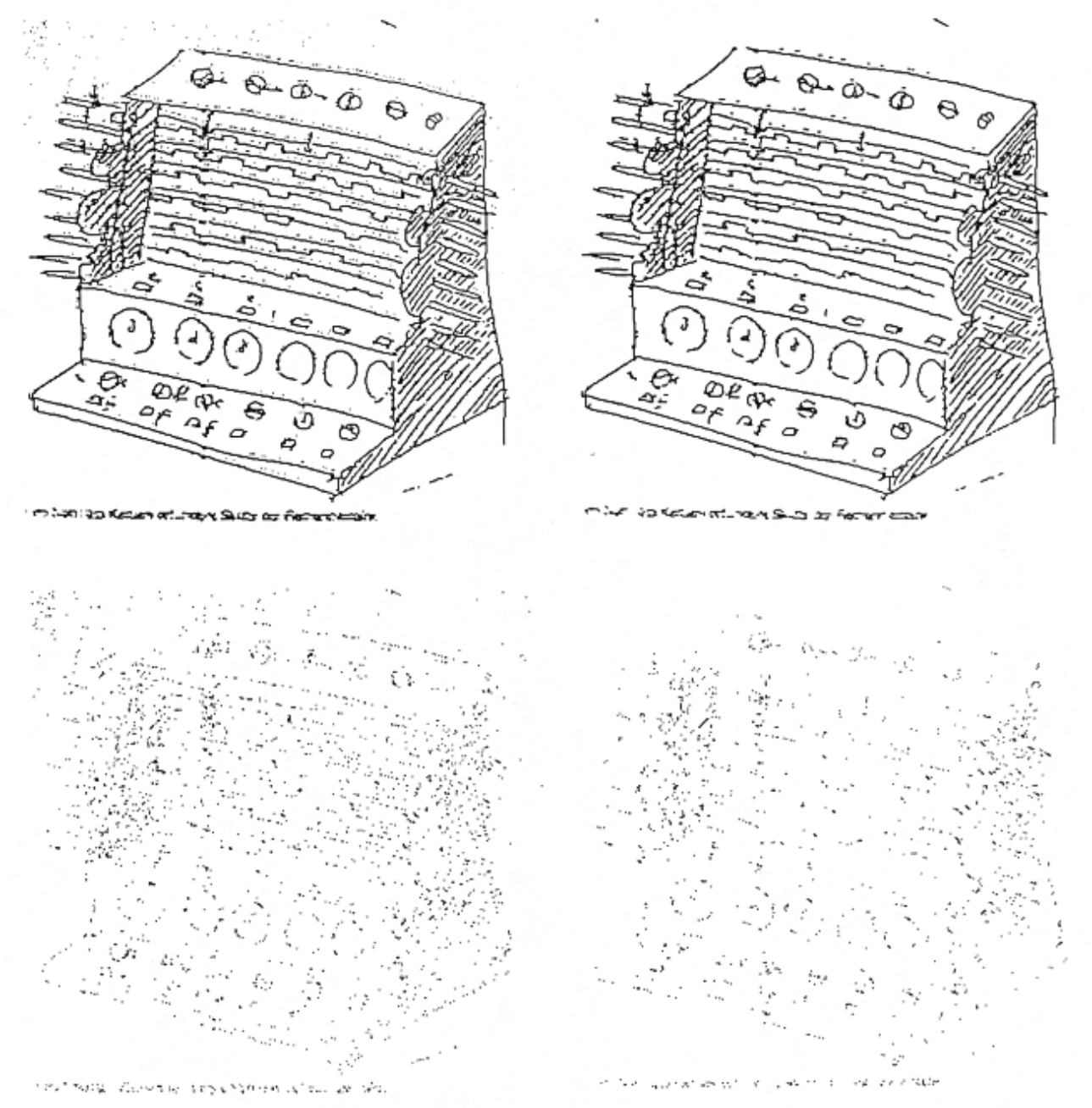


Abbildung 13.6: Originalbild, Vektorbild und Differenzbilder

Das Bild zeigt die Skizze der von Wilhelm Schickard 1623 in Tübingen konstruierten Rechenmaschine.

Links oben: skelettiertes Originalbild.

Rechts oben: Ergebnis der Vektorisierung.

Links unten: Originalbild – Vektorisiertes Bild (unberücksichtigte Punkte, Rauschen).

Rechts unten: Vektorbild – Originalbild (zusätzlich erhaltene Bildpunkte).

Aufgaben

Aufgabe 1

1. Zeichnen Sie die Kurve, deren Kettenkode durch die Folge gegeben ist:

   000000555333

2. Geben Sie die Kodierung dieser Kurve mittels differentiellem Kettenkode an.
3. Charakterisieren Sie die Kettenkodes aller Rechtecke, deren Seiten horizontal bzw. vertikal sind.

Aufgabe 2

Wieso wird bei dem beschriebenen Verfahren zur direkten Vektorisierung eine Störunterdrückung und eine Glättung durchgeführt?