LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Prof. Dr. Andreas Butz | Prof. Dr. Ing. Axel Hoppe

Dipl.-Medieninf. Dominikus Baur
Dipl.-Medieninf. Sebastian Boring

Übung: Computergrafik 1

Konvolutionen
Morphologische Operationen
Hough-Transformation

Organisatorisches

- Blatt 8 ist zweiwöchig zu bearbeiten (d.h. Abgabe am 26.6., 12 Uhr)
- Die nächste Übung (d.h. kommender Freitag 19.6., übernächster Montag 22.6.) entfällt

Besprechung Blatt 7

Wichtiges zum `QImage`:

- `bits()` liefert einen Pointer vom Typ `unsigned char`

```
unsigned char* imgPtr = image.bits();
```

- Pixel sind immer als BGR gespeichert (**nicht** RGB), d.h. `imgPtr[0]` ist der Blau-Wert, `imgPtr[1]` ist der Grün-Wert, usw.
- Der Alpha-Kanal (falls vorhanden) entspricht `imgPtr[3]`.
- `depth()` liefert die Anzahl der **Bits** (nicht Bytes) pro Pixel. Daher gilt für die Größe des Bildes in Bytes (bei 8 bits pro Kanal):

```
int size = image.width() * image.height() * image.depth() / 8;
```

- `copy()` erlaubt eine Kopie des (Sub-)Bildes:

```
QImage copied = image.copy(0, 0, image.width(), image.height());
```

Graustufenfilter:

```
QImage GrayscaleFilter::apply(QImage image) {
    // run filter code here!
    QImage newImage = image.copy(0, 0,
        image.width(), image.height());

    unsigned char* ptr = newImage.bits();
    unsigned char* end = ptr + newImage.width()
        * newImage.height() * newImage.depth() / 8;

    for(unsigned char* i = ptr; i < end;
        i += image.depth() / 8) {
        int grayVal = (int)fmax(0.0, fmin(255.0,
            (round(0.2126 * i[2]) + round(0.7152 * i[1])
             + round(0.0722 * i[0]))));

        i[2] = grayVal;
        i[1] = grayVal;
        i[0] = grayVal;
    }
    return newImage;
}
```

Invertierungsfilter:

```
QImage InvertFilter::apply(QImage image) {
    // run filter code here!
    QImage newImage = image.copy(0, 0,
        image.width(), image.height());

    unsigned char* ptr = newImage.bits();
    unsigned char* end = ptr + newImage.width()
        * newImage.height() * newImage.depth() / 8;

    for(unsigned char* i = ptr; i < end;
        i += image.depth() / 8) {
        for(int j = 0; j < image.depth() / 8; j++) {
            i[j] = 255 - i[j];
        }
    }

    return newImage;
}
```

HSL-Filter:

```
QImage HSLFilter::apply(QImage image) {
    // run filter code here!
    QImage newImage = image.copy(0, 0,
        image.width(), image.height());

    unsigned char* ptr = newImage.bits();
    unsigned char* end = ptr + newImage.width()
        * newImage.height() * newImage.depth() / 8;

    for(unsigned char* i = ptr; i < end; i += image.depth() / 8) {
        if(!(inRange(i[2], i[1], i[0]))) {
            i[0] = (unsigned char)0;
            i[1] = (unsigned char)0;
            i[2] = (unsigned char)0;
        }
    }
    return newImage;
}
```

HSL-Filter (`inRange`-Funktion):

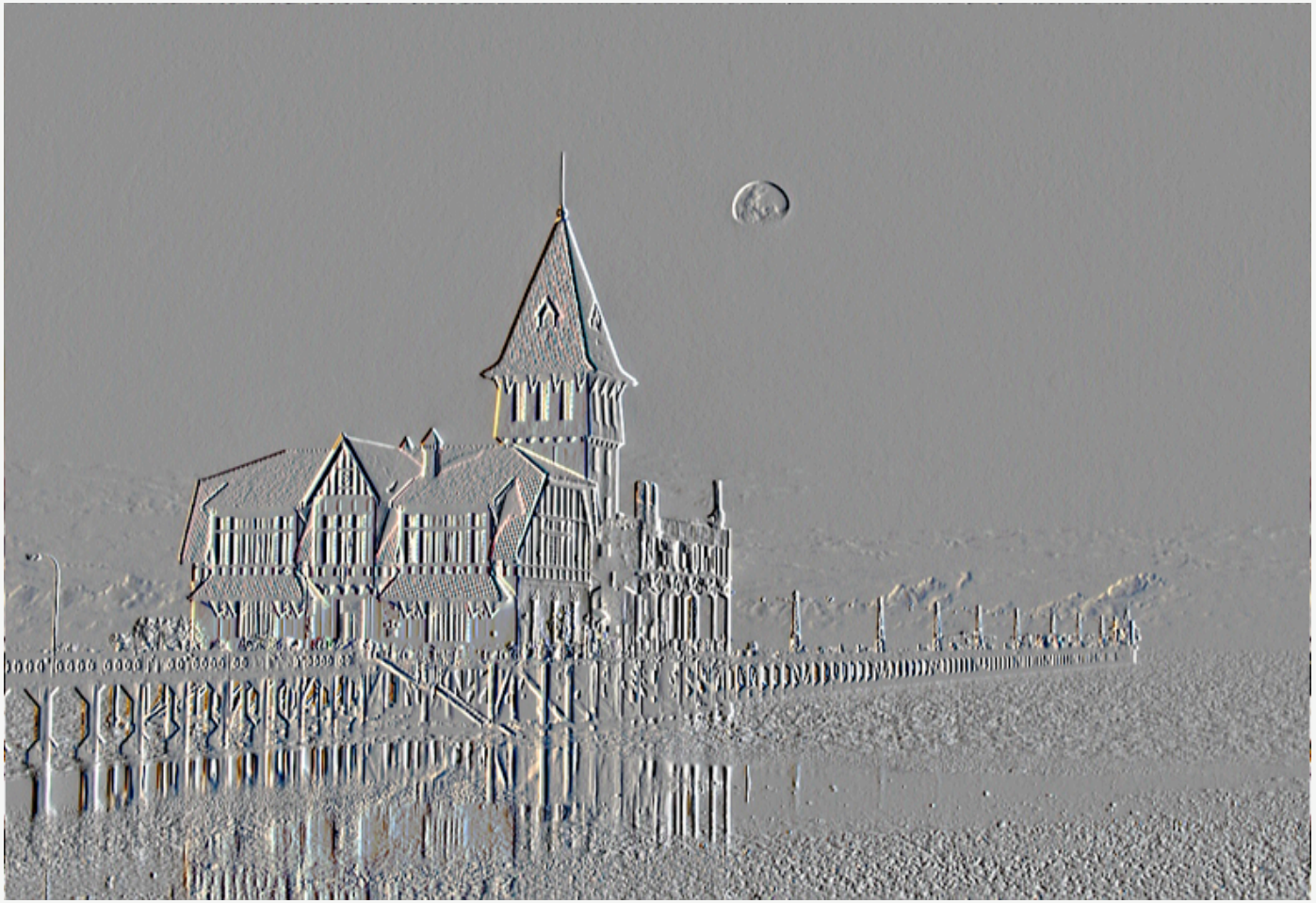
- überprüft, ob ein gegebener RGB-Wert innerhalb des vorgegebenen HSL-Bereiches ist:
  - `TRUE`: H ist innerhalb [hMin, hMax], S ist innerhalb [sMin, sMax] und L ist innerhalb [lMin, lMax]
  - `FALSE`: Falls **mindestens einer** der Werte nicht im Bereich ist
- Ist ein Wert im Bereich bleibt das Pixel unverändert,
- Falls nicht, wird das Pixel schwarz gesetzt

Konvolutionen

Konvolutionen

(Quelle: 'Again' by J's (http://www.flickr.com/photos/61815799@N00/1762152357/)

Konvolutionen



(Quelle: 'Again' by J's (http://www.flickr.com/photos/61815799@N00/1762152357/)

Konvolutionen

- Bisherige Bildmanipulationen (Farben) zielten nur auf einzelne Pixel ab
- Dadurch ergab sich eine Nichtbeachtung der Räumlichkeit von Bildern
- Objekte, Vordergrund und Hintergrund aber für die Wahrnehmung sehr wichtig
- Deswegen lassen sich bessere Ergebnisse durch Betrachtung der Nachbarschaft von Pixel erreichen

-1	0	1
-1	0	1
-1	0	1

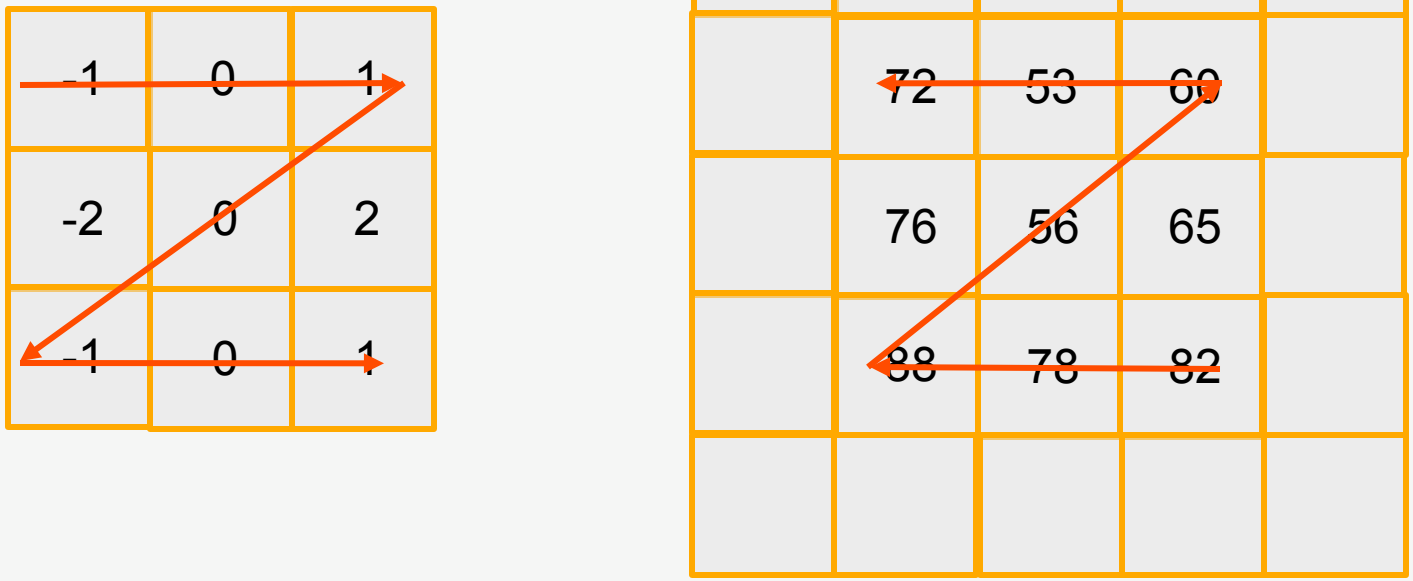
Konvolutionen

- Gewichtete Summen über den Grau- oder die einzelnen Farbwerte eines Pixels und dem seiner Nachbarn
- Koeffizienten zur Gewichtung werden in einer Matrix mit ungerader Dimension angegeben (Kernel)
- Konvolution wird zum Filtern von Bildern verwendet (z.B. Schärfefilter)
- Korrelation wird zum Erkennen von Features in Bildern benutzt

-1	0	1
-1	0	1
-1	0	1

(Quelle: Efford - Digital Image Processing using Java)

Berechnung von Konvolutionen



$$g(x,y) = (-1*82) + (1*88) + (-2*65) + (2*76) + (-1*60) + (1*72) = 40$$

(Quelle: Efford - Digital Image Processing using Java)

Konvolutionen

- Konvolution algorithmisch:

$h[m-1][n-1];$

$m_2 = \left\lfloor \frac{m}{2} \right\rfloor, n_2 = \left\lfloor \frac{n}{2} \right\rfloor;$

$sum = 0;$

for $k = -n_2$ to n_2 do

  for $j = -m_2$ to m_2 do

    $sum = sum + h(j + m_2, k + n_2) f(x - j, y - k)$

  end for

end for

$g(x, y) = sum;$

(Quelle: Efford - Digital Image Processing using Java)

Konvolutionen

- Konvolution (Faltung) mathematisch:

$$g(x,y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j,k)(x-j, y-k)$$

$$m_2 = \left\lfloor \frac{m}{2} \right\rfloor, \; n_2 = \left\lfloor \frac{n}{2} \right\rfloor$$

- h ist der Konvolutionskern, g das Ergebnisbild

(Quelle: Efford - Digital Image Processing using Java)

Konvolutionen – Probleme

- Farbwerte haben bestimmte Grenzen: Farbangaben dürfen nicht negativ oder größer als z.B. 255 (8 Bit) sein.
- Eine Konvolution kann dazu führen, dass diese Grenzen überschritten werden

- Einfache Lösung:
  - Abschneiden!
  - Notwendig, führt allerdings zu Informationsverlust
- Kernel normalisieren:
  - Kernel so anlegen, dass die Summe der Komponenten 1 ergibt

-1	1	1
-1	1	1
-1	1	1

-1 + 1 + 1 +
-1 + 1 + 1 +
-1 + 1 + 1 = 3

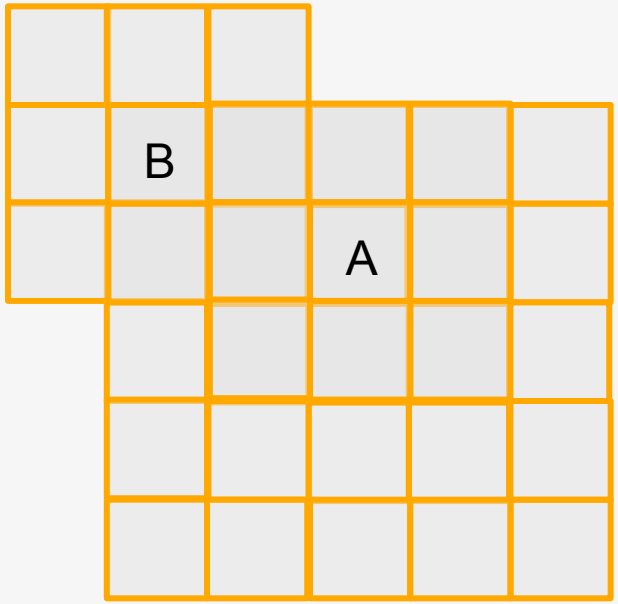
-1/3	1/3	1/3
-1/3	1/3	1/3
-1/3	1/3	1/3

(Quelle: Efford - Digital Image Processing using Java)

Konvolutionen – Probleme

- Was passiert mit Pixeln am Bildrand?
- Die Konvolution nur teilweise auszuführen erzeugt normalerweise Artefakte

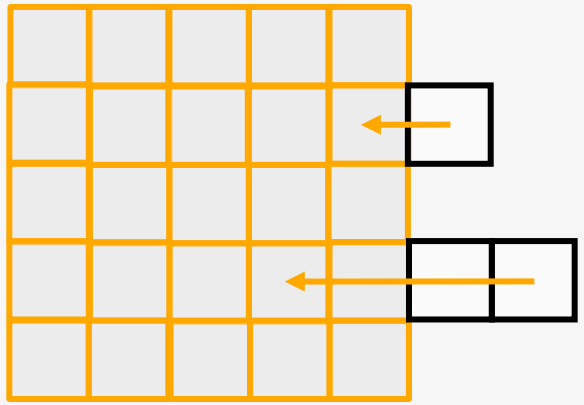
- Lösungen:
  - Randpixel nicht mit der Konvolutions-funktion bearbeiten
  - Rand abschneiden
  - Mehrere Kernel:
    (3x3) -> (3x3), (2x2),(2x3), (3x2)



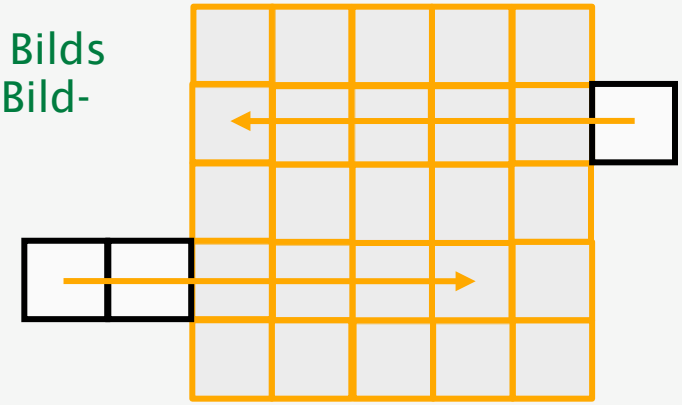
(Quelle: Efford - Digital Image Processing using Java)

Konvolutionen – Probleme

- Weitere Lösungen:
- Reflected Indexing
  - Anstelle von Pixeln die ausserhalb des Bilds liegen werden die gespiegelten Entsprechungen innerhalb benutzt
    (d.h. Bild (100, 100),
    Pixel (-2, 99) => (2, 99))



- Circular Indexing
  - “Wrap around”: Pixel die außerhalb des Bilds liegen werden von der jeweils anderen Bild-seite genommen
    (d.h. Bild (100, 100),
    Pixel (-2, 99) => (98, 99))



(Quelle: Efford - Digital Image Processing using Java)

Reflected Indexing

Circular Indexing

Konvolutionen – Probleme

- Weiteres Problem: Performance!
  - n x n - Kernel und Circular Indexing: n^2 Multiplikationen und Additionen pro Pixel (billige Digicam: 8 Megapixel)
- Lösungen:
  - Kernel aufteilen in eindimensionale Arrays der Länge n und dann zuerst spalten-, dann zeilenweise anwenden (z.B. Gaussian Blur)
  - Aber: Nicht alle Kernel lassen sich teilen!
  - Hardwarebeschleunigung

(Quelle: Efford - Digital Image Processing using Java)

Wichtige Kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Tiefpass / Blur

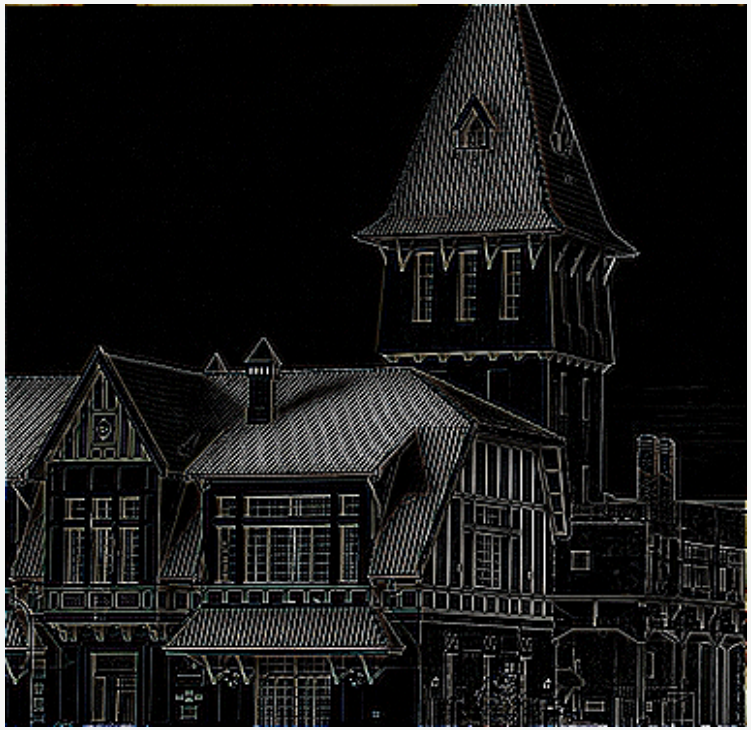
Wichtige Kernel

-1	-1	-1
-1	9	-1
-1	-1	-1

Hochpass

Wichtige Kernel

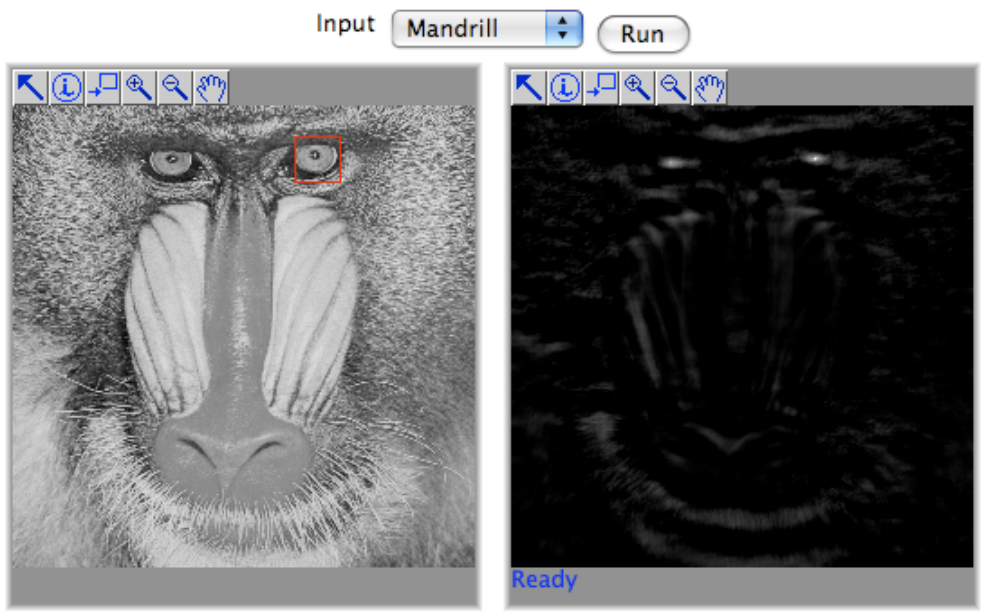
0	1	0
1	−4	1
0	1	0



Kantenerkennung

Korrelation

- Korrelation kann dazu benutzt werden, Bildausschnitte zu finden
- Der Kernel entspricht dem Muster das gesucht werden soll
- Die Korrelationsfunktion benutzt den Kernel 1:1 (ohne Spiegelung)
- Helle Bereiche nach Anwendung der Funktion sind ähnlich zum Muster (im Graustufenbild)

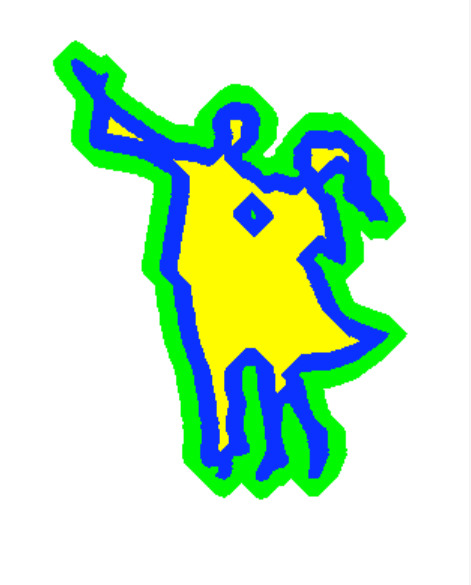


(Quelle: http://bigwww.epfl.ch/demo/templatematching/tm_correlation/demo.html)

Morphologische Operationen

Morphologische Operationen

- Operationen auf der Gestalt von Objekten. Setzt die Extraktion einer Gestalt voraus (Segmentierung)
- Ziele:
  - Veränderung der Gestalt um Störungen zu beseitigen
  - Berechnung von Formmerkmalen
  - Suche nach bestimmten Formen (also: Analyse)
- Beispiele: Erosion (gelb), Dilatation (grün)



(Quelle: http://en.wikipedia.org/wiki/Morphological_image_processing)

Morphologische Operationen

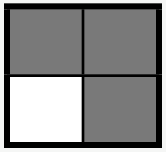
- Nach Segmentierung erhalten wir einen Vorder- und Hintergrund des Bilds. Daraus bilden wir eine binäre Variante mit Vordergrund = 1 und Hintergrund = 0.
- Jetzt brauchen wir ein sog. Strukturelement (structuring element), eine binäre n x n Matrix (ähnlich Konvolutionskernel)
- Ein Strukturelement enthält zusätzlich einen Ankerpunkt, der meist innerhalb des Strukturelements liegt
- Einfache Operationen wie Erosion und Dilatation haben nur Einsen im Strukturelement

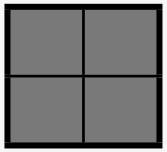
$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

(Quelle: http://en.wikipedia.org/wiki/Morphological_image_processing)

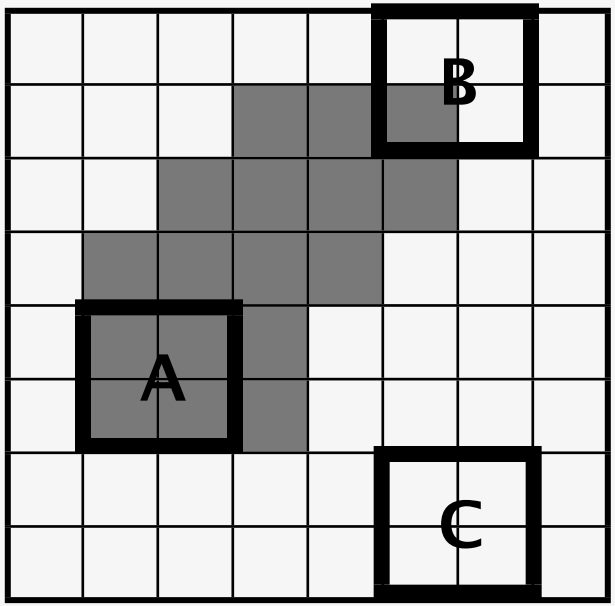
Morphologische Operationen

- Das Strukturelement wird pixelweise auf das Ausgangsbild gelegt (der Ankerpunkt des Strukturelements liegt dabei auf dem aktuellen Pixel)
- Jetzt werden alle *Einsen* im Strukturelement mit dem Bild verglichen.
- Folgende Ergebnisse sind möglich:
  - Komplette Übereinstimmung (*fit*)
  - Teilweise Übereinstimmung (*hit*)
  - Keine Übereinstimmung (*miss*)

s1: 

s2: 

	A	B	C
s1	fit	miss	miss
s2	fit	hit	miss



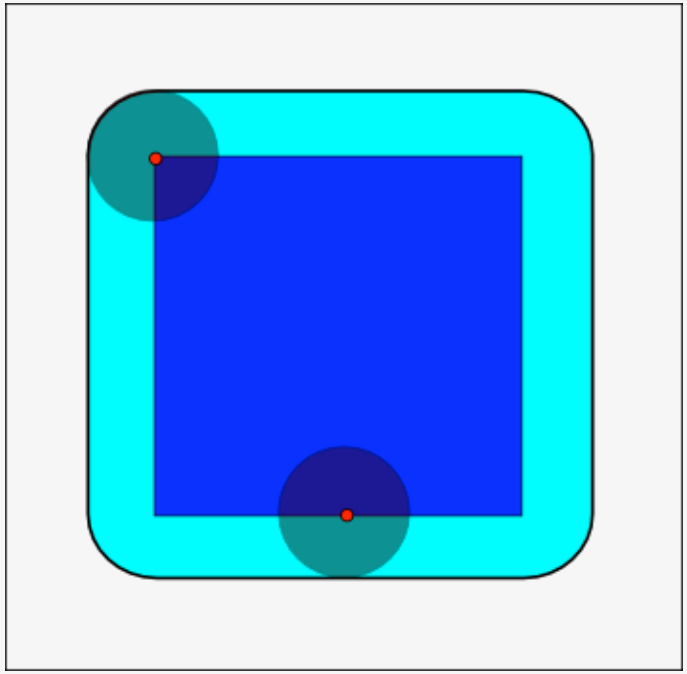
(Quelle: Efford - Digital Image Processing using Java)

Dilatation

- Je nach Ergebnis kann der aktuelle Pixel auf 1 oder 0 gesetzt werden.
- Der Dilatationsoperator ist folgendermaßen definiert:

$$g(x,y) = \{ \begin{array}{ll} 0 & \text{falls s f verfehlt (miss)} \\ 1 & \text{sonst} \end{array}$$

- s ist dabei das Strukturelement und f ein Vordergrundpixel
- Definiert für beliebige n x n Matrizen
- Durch die Dilatation werden existierende Flächen im Bild ausgedehnt (als würde das Strukturelement am Rand entlanggezeichnet) und Löcher gefüllt



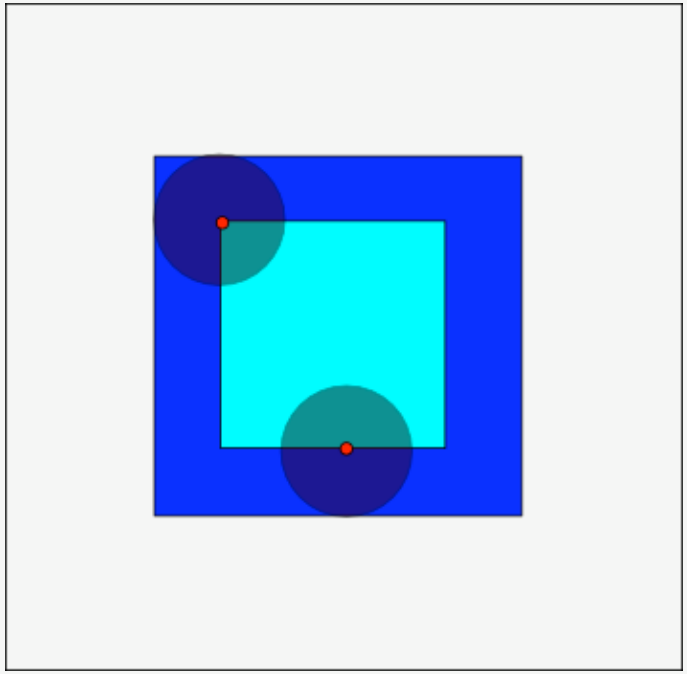
(Quellen: Efford - Digital Image Processing using Java, http://en.wikipedia.org/wiki/Morphological_image_processing)

Erosion

- Der Erosionsoperator ist folgendermaßen definiert:

$$g(x,y) = \begin{cases} 1 & \text{falls s mit f vollständig übereinstimmt (fit)} \\ 0 & \text{sonst} \end{cases}$$

- s ist dabei das Strukturelement und f ein Vordergrundpixel
- Definiert für beliebige n x n Matrizen
- Durch die Erosion werden Flächen verkleinert und Löcher vergrößern sich



(Quellen: Efford - Digital Image Processing using Java, http://en.wikipedia.org/wiki/Morphological_image_processing)

Morphologische Operationen

- Mit anderen Regeln und Strukturelementen lassen sich weitere Effekte erzeugen. Auch Kombinationen einfacher morphologischer Operationen sind möglich.
- Dilatation und Erosion sind eigentlich nur für binäre Bilder definiert, es gibt allerdings Anpassungen für Graustufenbilder.
- Diese können auf Farbbilder angewendet werden indem die Helligkeit des Bilds angepasst wird

(Quelle: http://en.wikipedia.org/wiki/Morphological_image_processing)

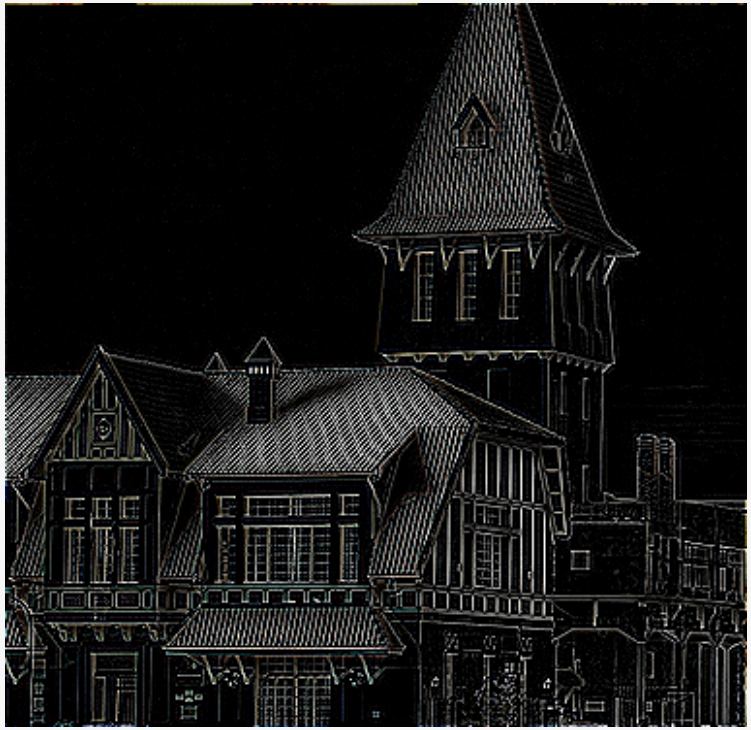
Dilatation

Dilatation

Hough-Transformation

Hough Transform

- Kantenerkennung, z.B. durch Konvolutionsoperationen, liefert ein Bild das (hoffentlich) nur noch Kanten zeigt
- Operationen wie Korrelation können zur Mustererkennung eingesetzt werden
- Die Hough-Transformation ermöglicht das (algorithmische) Finden von geometrischen Körpern (Linien, Kreisen) im Bild

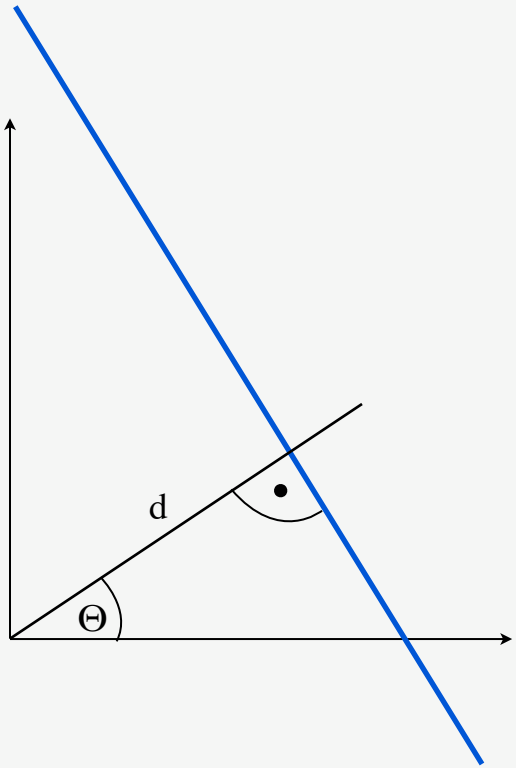


Hough Transform

- Grundidee: Wir suchen nach parametrischen Darstellungen geometrischer Körper und lassen die Pixel des Bilds darüber “abstimmen” welche dieser Darstellungen zutrifft
- Einfachste Variante: Erkennung von Geraden
- Die gängige parametrische Darstellung von Geraden

  $$y = m * x + t$$

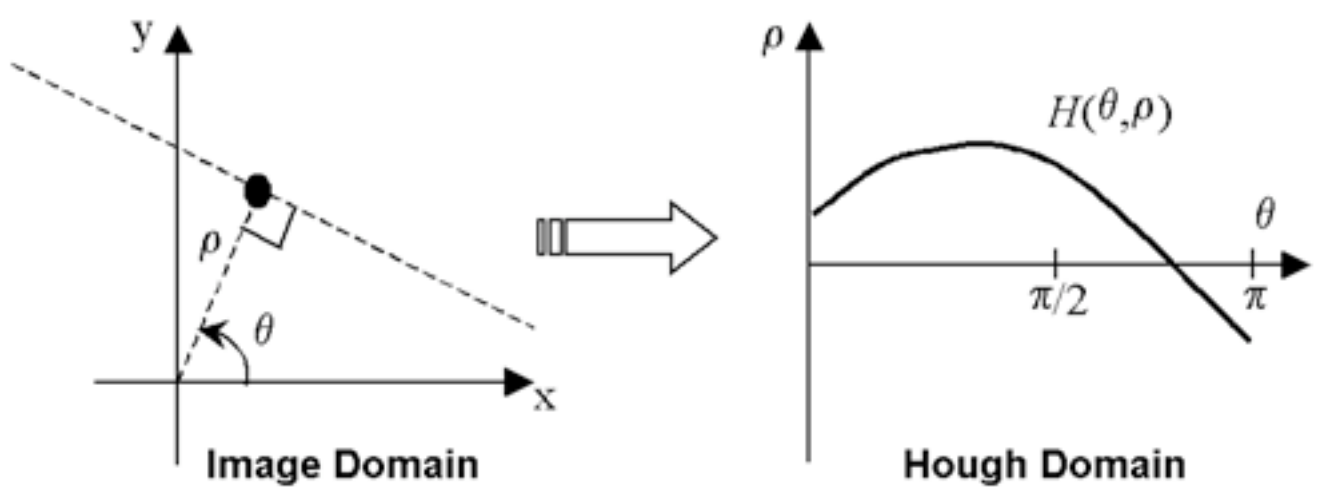
  funktioniert leider nur für Geraden deren Steigung nicht unendlich ist (d.h. die nicht senkrecht sind).
- Daher bilden wir für jede Gerade die Normale die durch den Ursprung geht und den resultierenden Schnittpunkt
- Die Beschreibung sind die Polarkoordinaten des Schnittpunkts, Abstand d und Winkel Θ



(Quelle: http://en.wikipedia.org/wiki/Hough_transform)

Hough Transform

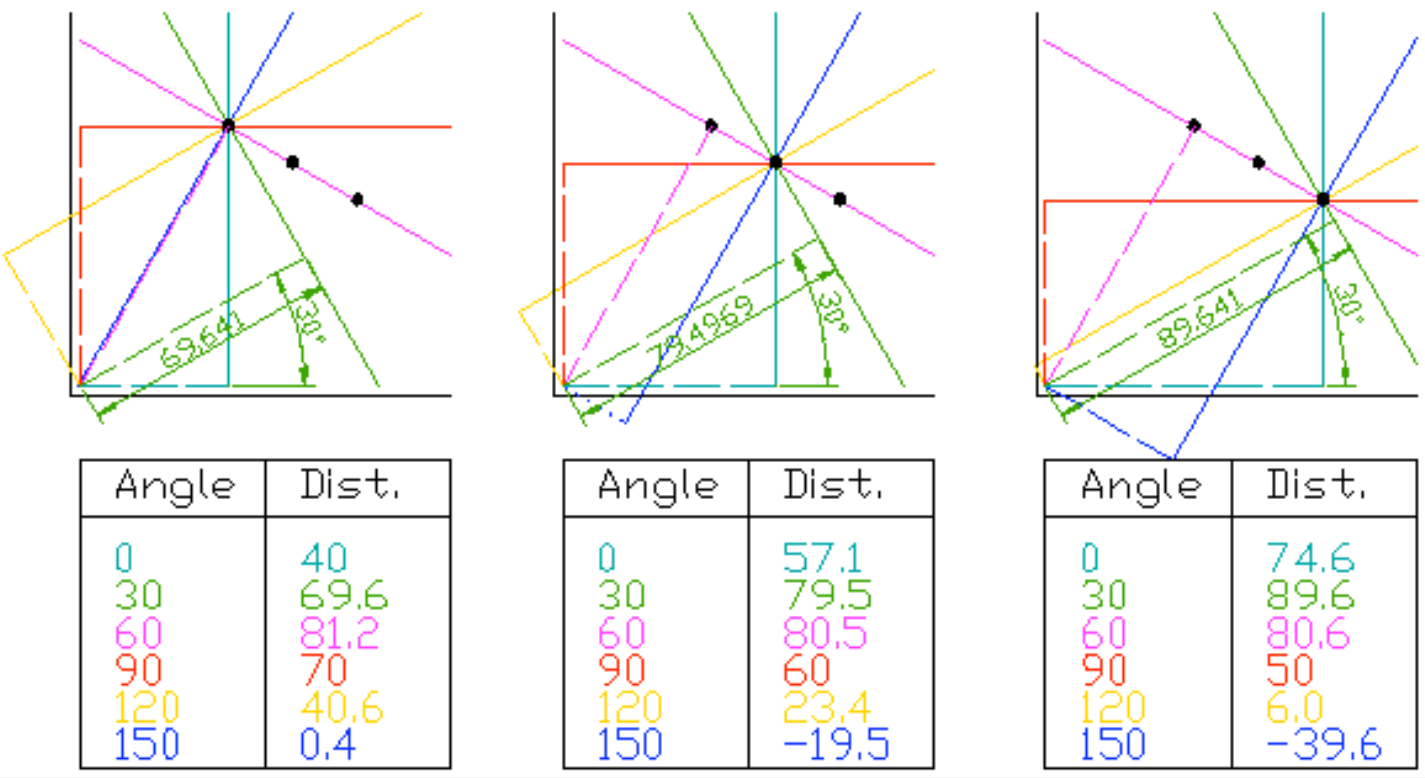
- Die Darstellung der Linien in (d, Θ) bildet den sog. Hough-Raum und die Transformation in diesen heißt Hough-Transformation
- Wenn wir von den einzelnen Bildpunkten ausgehen bilden diese die Schnittpunkte für ein unendliches Geradenbündel
- Dieses Geradenbündel durch einen Punkt entspricht im Hough-Raum einer sinusförmigen Kurve



(Quelle: http://physics.nyu.edu/grierlab/idl_html_help/H17.html)

Hough Transform

- Um Geraden zu erkennen bestimmen wir für jeden (Kanten-)Bildpunkt dessen Kurve im Hough-Raum



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

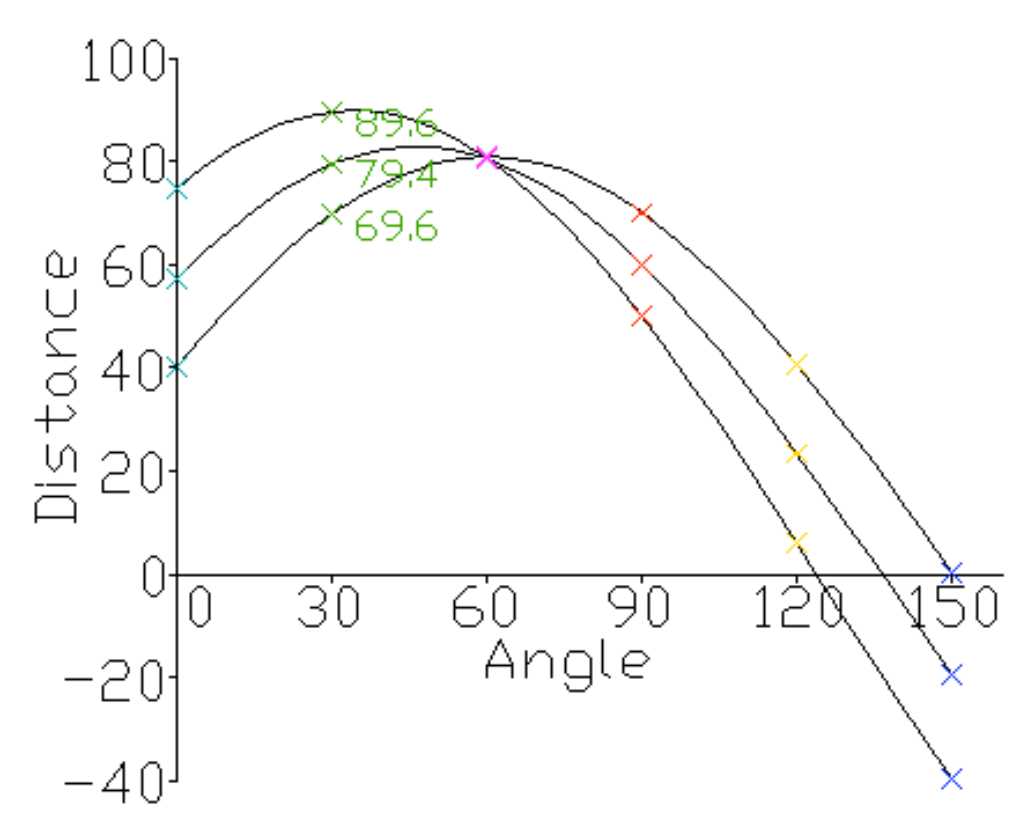
Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

(Quelle: http://en.wikipedia.org/wiki/Hough_transform)

Hough Transform

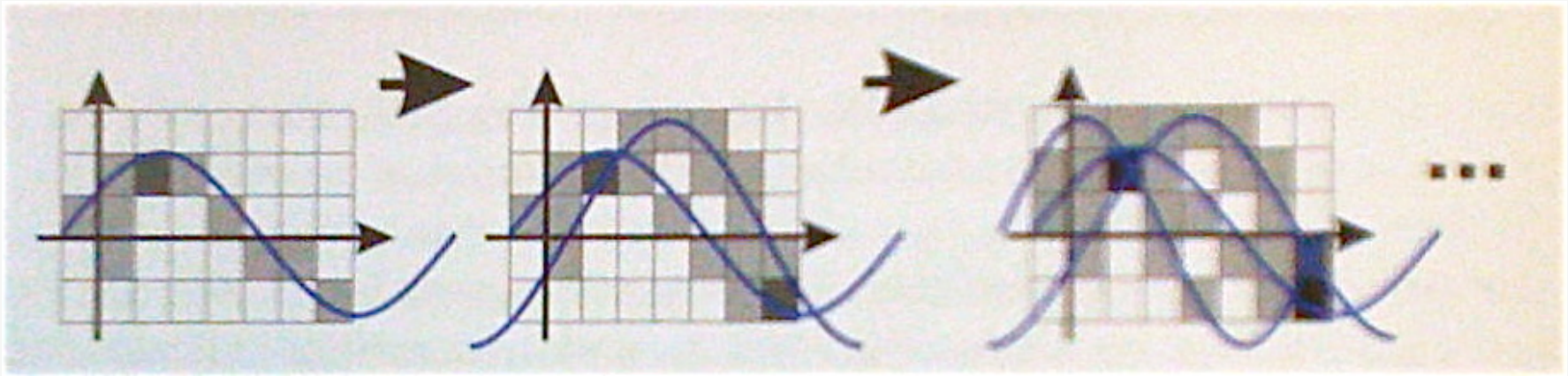
- Die Schnittpunkte dieser Kurven geben die Lage von Geraden an - jeder Schnittpunkt steht dabei für eine Gerade



(Quelle: http://en.wikipedia.org/wiki/Hough_transform)

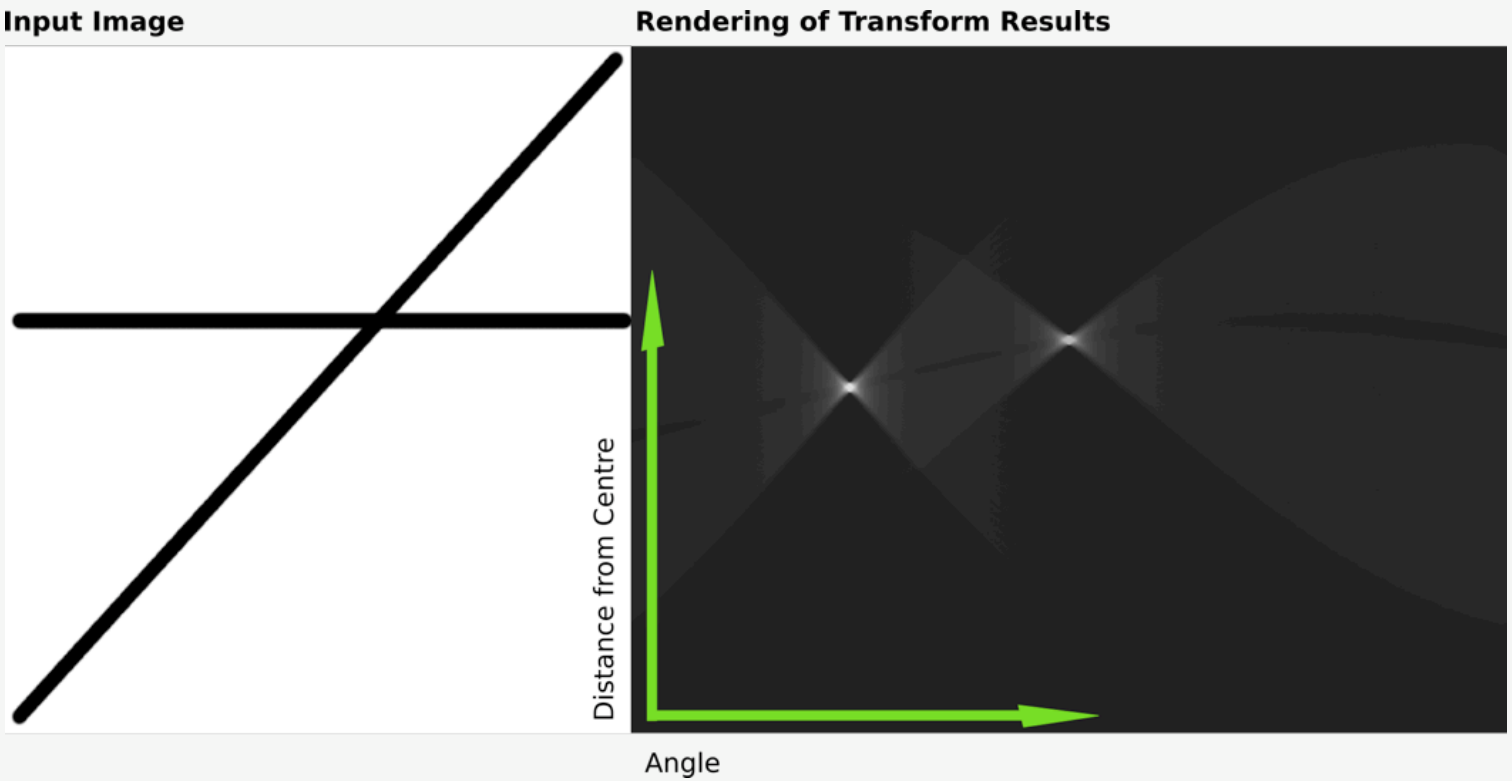
Hough Transform

- Um die Laufzeit zu verbessern wird der Hough-Raum diskretisiert und in ein festes Raster eingeteilt
- Die einzelnen Rasterpunkte heißen Akkumulatorzellen und werden jedesmal im Wert erhöht wenn eine Kurve sie durchläuft
- Die lokalen Maxima sind die Parameter der gefundenen Geraden



(Quelle: Tönnies - Grundlagen der Bildverarbeitung)

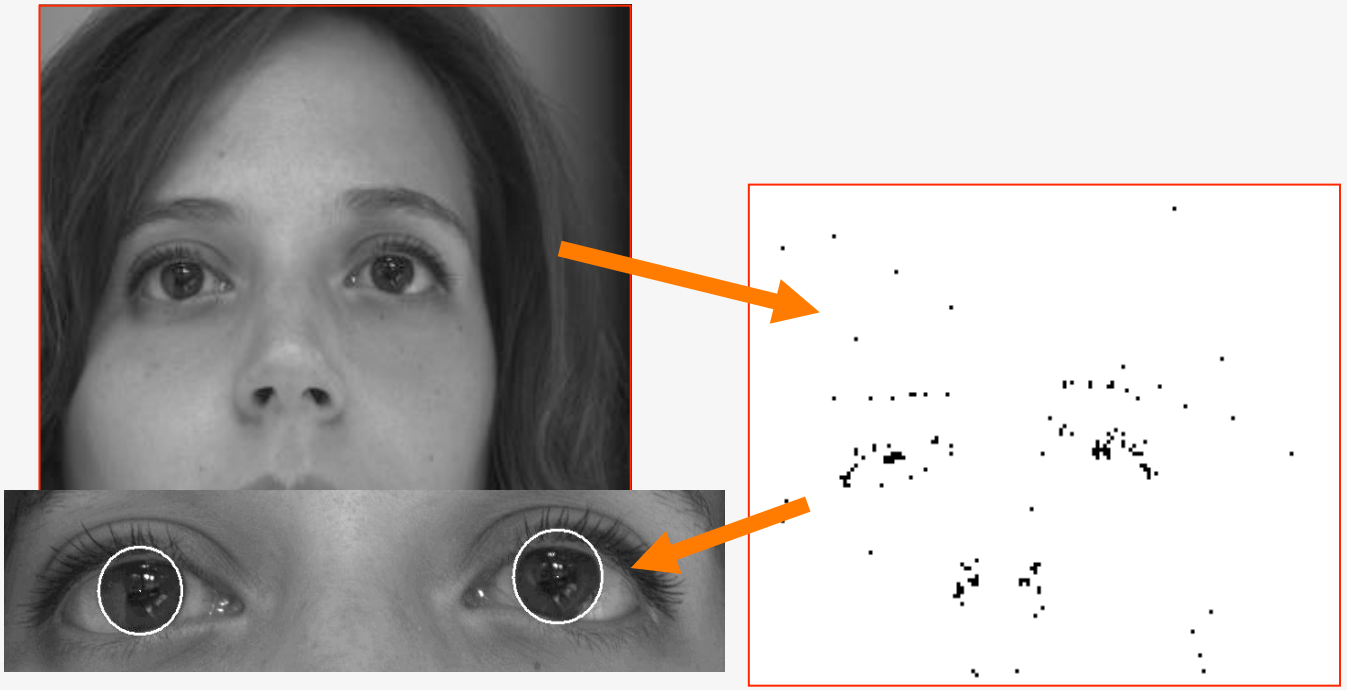
Hough Transform



(Quelle: http://en.wikipedia.org/wiki/Hough_transform)

Hough Transform

- Hough-Verfahren funktionieren am Besten auf binären Bildern, aber auch bei Graustufen (natürlich mit entsprechendem Rauschen)
- Gleiches Prinzip für andere Körper und Parameter möglich, z.B. Mittelpunkt und Radius bei Kreisen



(Quelle: Tönnies - Grundlagen der Bildverarbeitung)

Weiterführende Literatur:

- Nick Efford: “Digital Image Processing – a practical introduction using Java”, ISBN-13: 978-0201596236
- Klaus D. Tönnies: “Grundlagen der Bildverarbeitung”, ISBN-13: 978-3827371553

- http://doc.trolltech.com/4.5/
- http://www.qtsoftware.com/products/