B3. Bilderkennung mit Java

B3.3 HIPR2 Framework

B3.4 Praktisches Beispiel und weitere Operationen auf Bildern

- thresholding
- morphologische Operationen
- region labeling

Literatur:

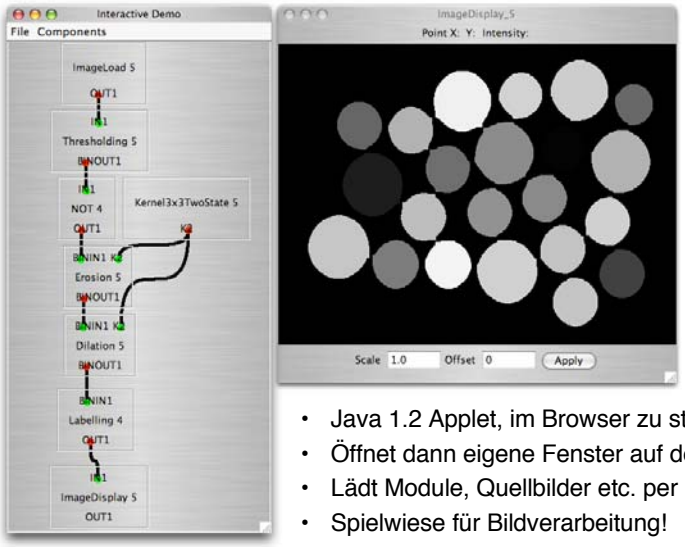
The Hypermedia Image Processing Reference
http://homepages.inf.ed.ac.uk/rbf/HIPR2/
Klaus D. Tönnies: "Grundlagen der Bildverarbeitung"
ISBN 3-8273-7155-4

Bildmaterial in dieser Vorlesung aus HIPR2 und Tönnies entnommen!

HIPR2: The Hypermedia Image Processing Reference

- Entstanden an der Univ. Edinburgh, KI-Labor (Robotik-Anwendungen!)
  - Autoren: Robert Fisher, Simon Perkins, Ashley Walker, Erik Wolfart
- Referenz der 50 meistverwendeten Operatoren in der Bildverarbeitung
  - Detaillierte Beschreibung jedes Algorithmus
  - Java demo für jeden Algorithmus
  - Tips und Beispielbilder zu jedem Operator
- Interaktives Tableau, mit dem man Operatorketten ausprobieren kann.
- Sammlung von Übungsaufgaben, Lexikon der Begriffe in der Bildverarbeitung, weiterführende Literaturhinweise

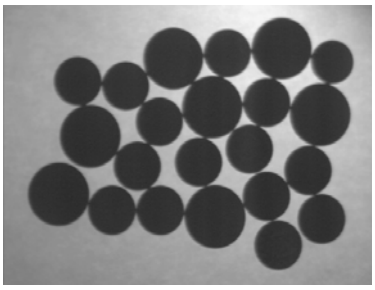
Interaktives Tableau bei HIPR2

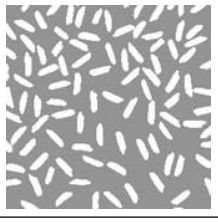


- Java 1.2 Applet, im Browser zu starten
- Öffnet dann eigene Fenster auf dem Desktop
- Lädt Module, Quellbilder etc. per HTTP nach
- Spielwiese für Bildverarbeitung!

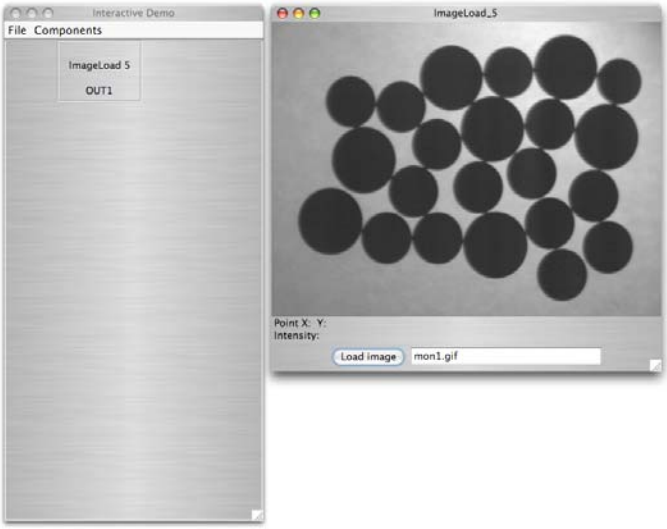
Beispiel für eine Problemstellung in der Bildverarbeitung: Objekte zählen

- Basis: Graustufenbild, auf dem eine Anzahl von Objekten zu sehen ist
  - Hintergrund evtl. gestört
  - Objekte evtl. berührend
  - ..nicht überlappend!
- Ziel: Anzahl der sichtbaren Objekte
- Vorgehensweise: Verarbeitungskette aus Operatoren der Bildverarbeitung
- Soll auf verschiedene Eingangsbilder anwendbar sein.

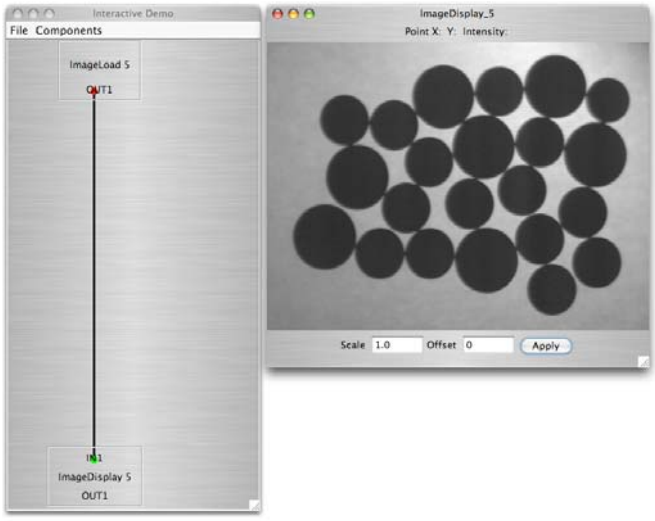




1. Schritt: Bild laden

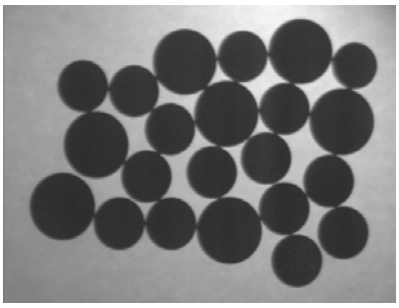


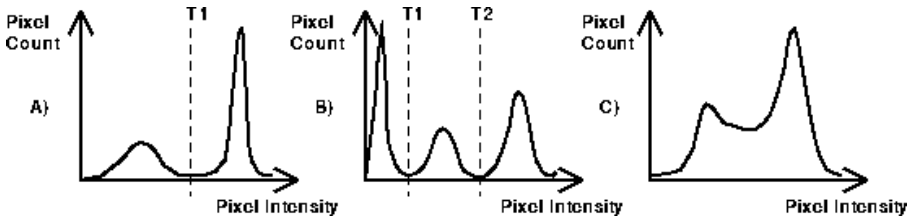
Letzter Schritt: Bild anzeigen



Problem: Segmentierung

- Wo im Bild ist ein Objekt, und wo nicht?
- Einfachstes Verfahren: Thresholding
- Histogrammbasiertes Verfahren
  - Definiere Schwellwerte, ab denen auf Schwarz bzw. Weiß umgeschaltet wird
  - Ausgabe: Binärbild
- Kritische Parameter: die Schwellwerte





Thresholding in Java (aus HIPR2)

```
...
 //applies two thresholds to a greyscale image
 //if a pixel is between the threshold values then colour it white
 //otherwise it should be black

 private int []  apply_two_threshold(int low, int high) {
      int blue = 0;
  for (int i=0; i<src_1d.length; i++){
    blue = src_1d[i] & 0x000000ff;
    if ((blue<= high) && (blue >= low)) {
   dest_1d[i] = 0xffffffff;
    } else dest_1d[i] = 0xff000000;
  }
  return dest_1d;
 }
...
```

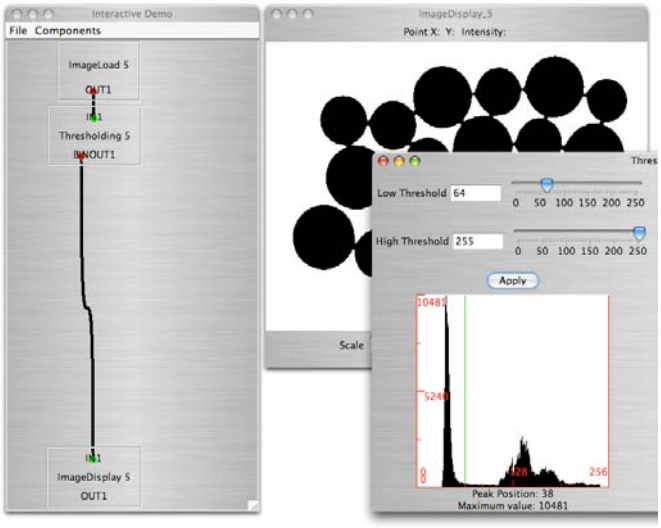
javax.media.jai.operator.ThresholdDescriptor

// Aus der JAI Dcumentation:

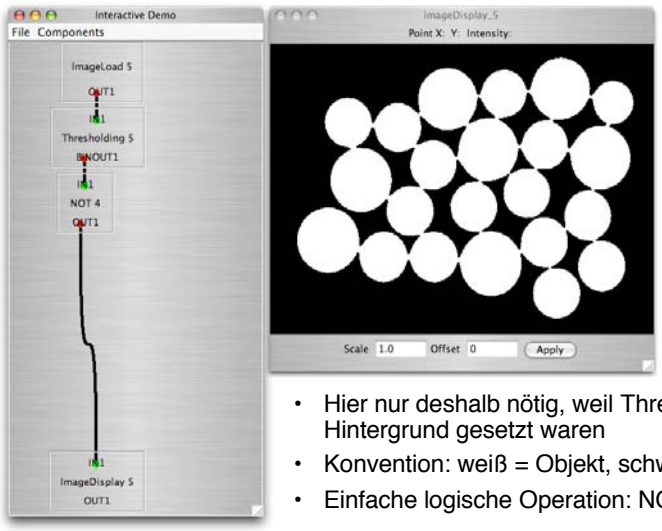
```
lowVal = (low.length < dstNumBands) ?
        low[0] : low[b];
highVal = (high.length < dstNumBands) ?
         high[0] : high[b];
const = (constants.length < dstNumBands) ?
         constants[0] : constants[b];

if (src[x][y][b] >= lowVal && src[x][y][b] <= highVal) {
   dst[x][y][b] = const;
} else {
   dst[x][y][b] = src[x][y][b];
}
```


2. Schritt: Thresholding

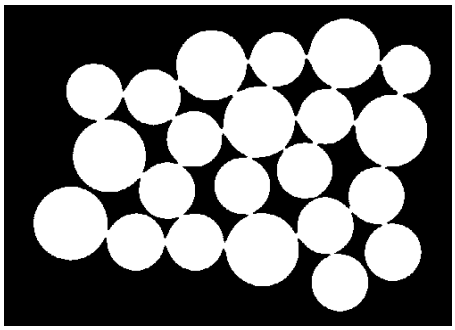


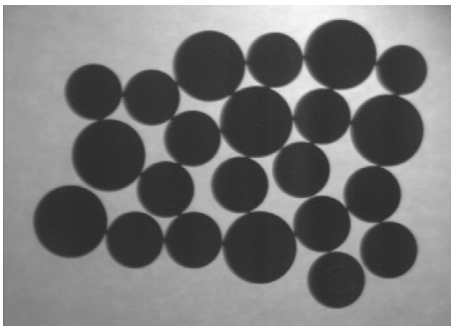
Zwischenschritt: Bild invertieren



- Hier nur deshalb nötig, weil Thresholds für den Hintergrund gesetzt waren
- Konvention: weiß = Objekt, schwarz = Hintergrund
- Einfache logische Operation: NOT

Problem: Objekte verschmolzen





- Immer noch Pixelbrücken zwischen den einzelnen Formen
- Nicht durch andere Threshold-werte zu beseitigen!
- Lösungsansatz: Morphologische Operatoren auf dem Binärbild

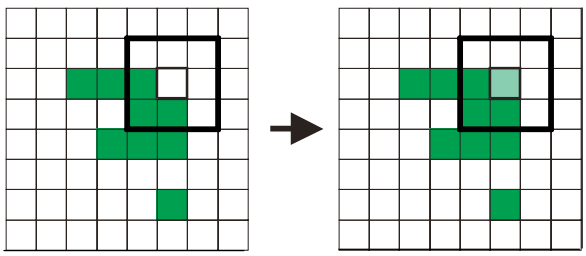
Morphologische Operationen

- Morphologisch: die äußere Gestalt betreffend
- morphologische Operationen:
  - Operationen auf der Gestalt von Objekten
  - ➔ setzt die Extraktion einer Gestalt voraus
  - also: in erster Linie Operation auf Segmenten (d.h., auf Binärbildern)
- Ziel von morphologischen Operationen:
  - Veränderung der Gestalt, um Störungen nach einer Segmentierung zu beseitigen
  - Berechnung von Formmerkmalen
  - Suche nach bestimmten Formen (also: Analyse)

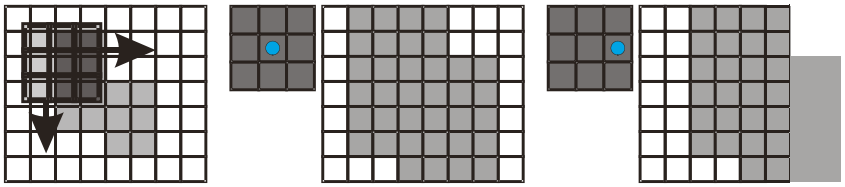
Dilatation

Dilatation (Ausdehnung): $G \oplus S$ mit Strukturelement S

$$g(m,n) = \bigvee_{(m_k,n_k) \in S} b(m + m_k, n + n_k)$$



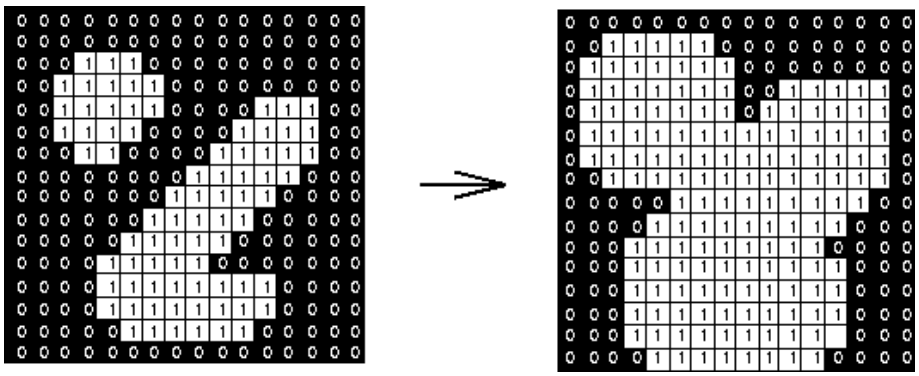
Dilatation



Dilatation wird (wie jede morphologische Operation) für einen **Ankerpunkt** ausgeführt.

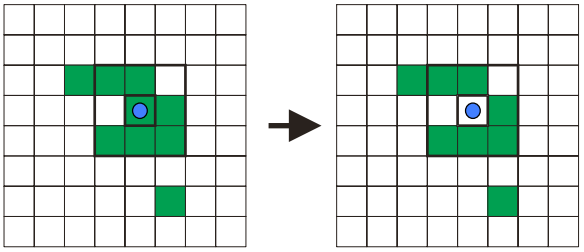
Dilatation: - verbindet Strukturen
- füllt Löcher
- vergrößert

Dilatation mit 3x3 Strukturelement



Erosion

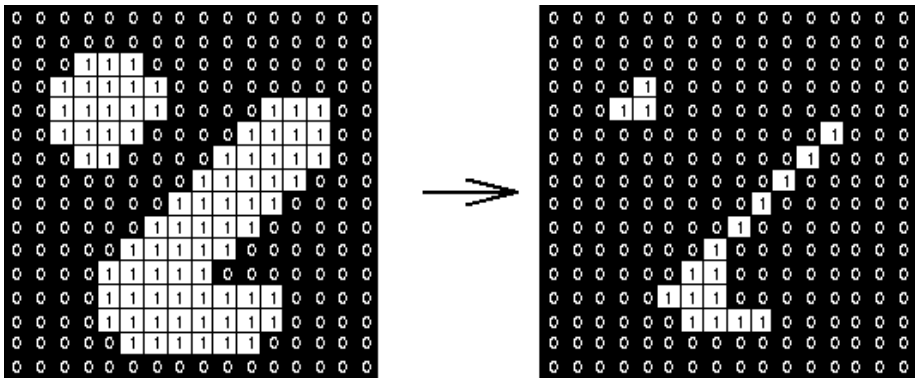
$$g(m,n) = \bigwedge_{(m_k,n_k)\in S} b(m+m_k, n+n_k)$$



Erosion: $G \ominus S$ mit Strukturelement S

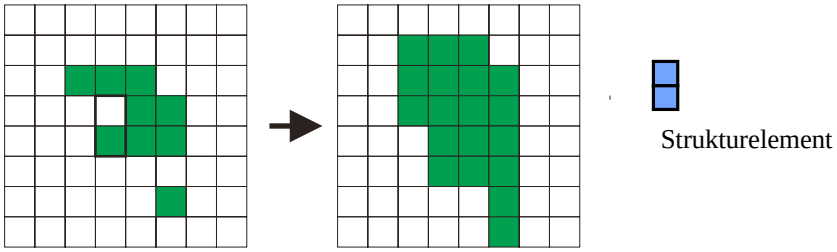
Erosion: - löst Strukturen auf
- entfernt Details
- verkleinert

Erosion mit 3x3 Strukturelement



Strukturelemente

- Ein Strukturelement einer morphologischen Operation entspricht dem Faltungskern bei einer Konvolution.
- Mit einem gezielt geformten Strukturelement können genau definierte Formveränderungen erzeugt werden.



Dilatation

Strukturelement

Morphologische Operatoren in JAI

- javax.media.jai.operator.DilateDescriptor

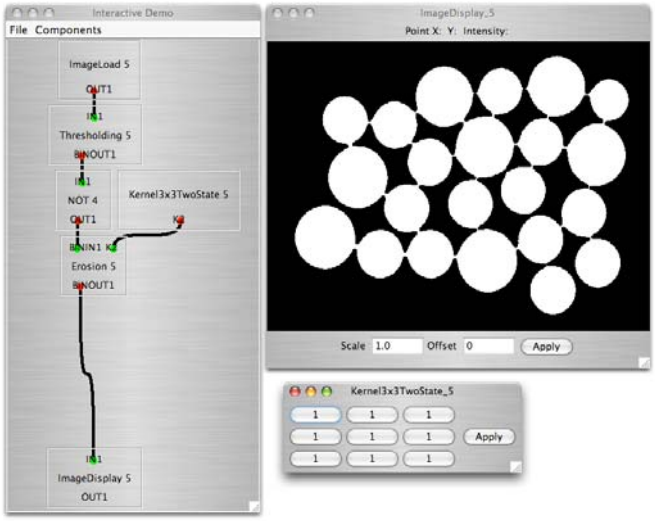
  For a kernel K with a key position (xKey,yKey), the dilation of image I at (x,y) is given by:

  max{ I(x-i, y-j) + K(xKey+i, yKey+j): some (i,j) restriction }
  where the (i,j) restriction means:
  all possible (i,j) so that both I(x-i,y-j) and K(xKey+i, yKey+j)
  are defined, that is, these indices are in bounds.

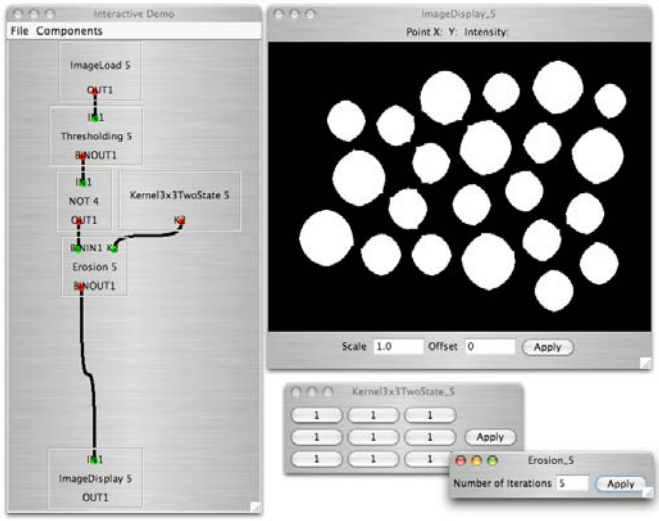
- javax.media.jai.operator.ErodeDescriptor

  max{ f: f + K(xKey+i, yKey+j) <= I(x+i,y+j): all (i,j)}
  "all" possible (i,j) means that both I(x+i,y+j) and K(xKey+i, yKey+j)
  are in bounds. Otherwise, the value is set to 0.
  "f" represents all possible floats satisfying the restriction.

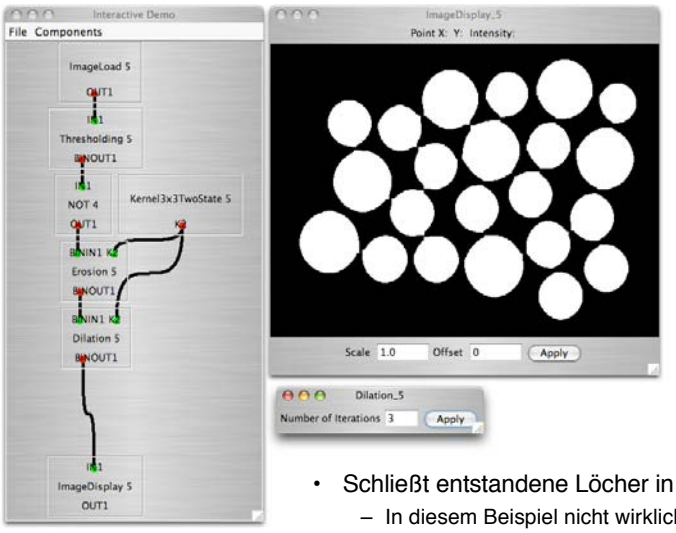
3. Schritt: Erosion: Hmm...



3. Schritt: Erosion, aber 5 mal!

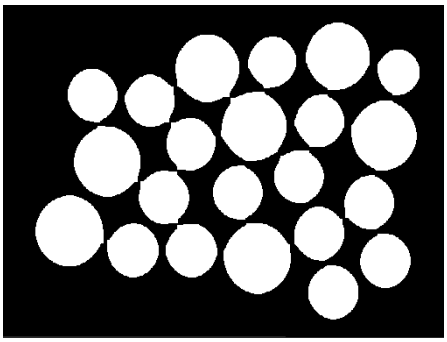


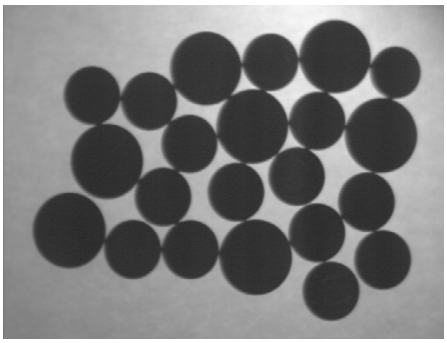
4. Schritt: Dilatation



- Schließt entstandene Löcher in Objekten
  - In diesem Beispiel nicht wirklich nötig ;-)

Zwischenergebnis: klar getrennte Formen





- Aber: noch wissen wir nicht, wie viele Objekte im Bild enthalten sind.
- Lösungsansatz: Region labeling
- Verschiedene Verfahren denkbar
  - anschaulichstes Verfahren: region labeling + flood fill
  - Effizienteres Verfahren: Connected Component Labeling

Region Labeling

- Schwellenwert zerlegt das Bild in Vordergrund und Hintergrundsegmente.
- **Region Labeling** bestimmt Ort und Anzahl aller zusammenhängenden Gebiete im Binärbild *b*:

```
region.initialise() // Region der Größe M,N erzeugen und
label=1             // mit Null initialisieren, Startlabel=1
for (i,j) = 0, (M,N) do       // Doppelschleife über i und j
  if region.labels(i,j) = 0 then // dieser Ort ist noch nicht
                                 Teil einer Region
    label = label+1               // neues Label vergeben
    region.flood_fill(i,j,label)  // zusammenhängendes
                                     Gebiet um (i,j) mit
                                     Label füllen
```

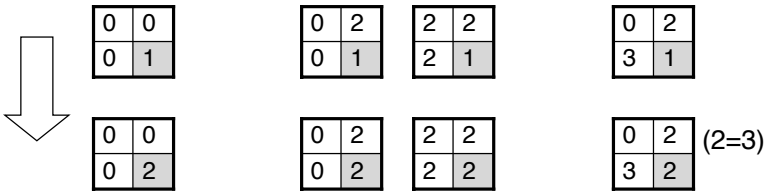

Flood Fill

```
flood_fill(i,j,label)  // Variablen zur Auswertung der
       Zusammenhangsbedingung sind global verfügbar
if f(i,j) erfüllt Zusammenhangsbedingung then
   region(i,j) = label   // Region an (i,j) mit label
                         // versehen
   flood_fill(i-1,j,label) // Nachbarpixel untersuchen
   flood_fill(i,j-1,label)
   flood_fill(i+1,j,label)
   flood_fill(i,j+1,label)
```

Region labeling + flood fill:
- Vollständige Segm.?
- Überdeckungsfrei?
- Zusammenhängend?

Bsp. für Zusammenhangsbedingung:
- hat den gleichen Grauwert wie Saatpunkt

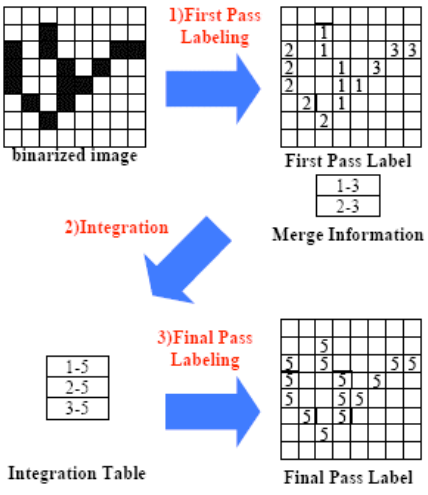
Connected Component Labeling: 1. Schritt



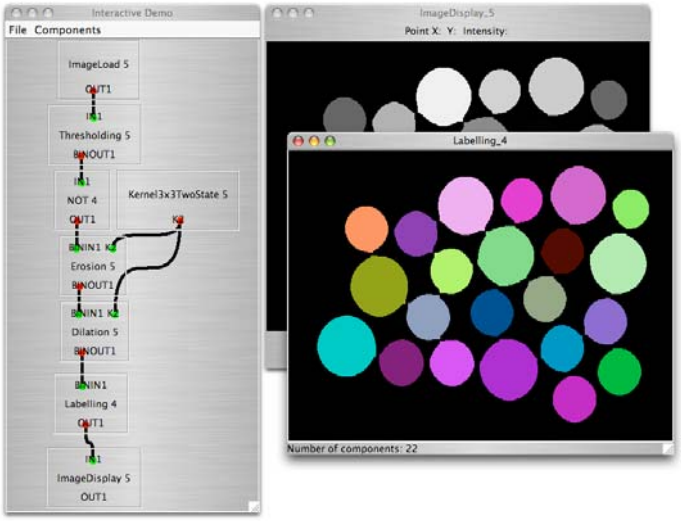
- Bild wird zeilenweise abgescannt
  - Nachbarpixel links und oberhalb wurden schon angeschaut
- Falls Pixel = 1, dann sind 3 Fälle möglich:
  - Alle Nachbarpixel 0 => neue Region gefunden, neues Label vergeben!
  - Ein oder mehrere Nachbarpixel haben das gleiche Label => Pixel gehört zur selben Region, gleiches Label vergeben!
  - Nachbarpixel haben verschiedene Label => Pixel verbindet diese Regionen, ein Label auswählen und Äquivalenz im 2. Schritt!

Connected Component Labeling: 2. Schritt

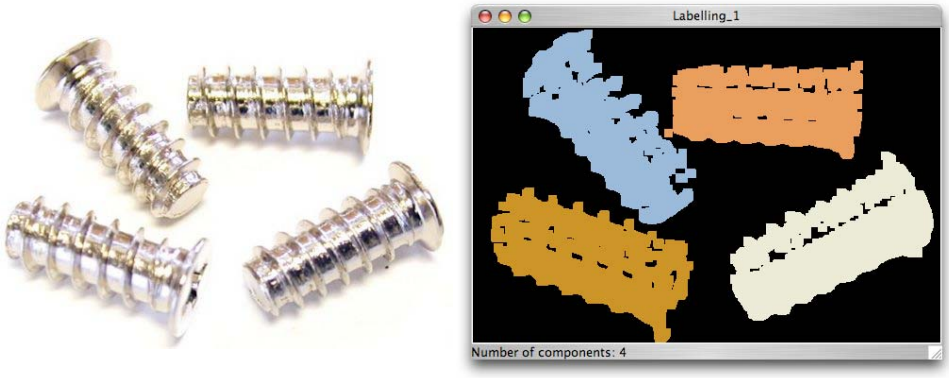
- Label zu Äquivalenzklassen zusammenfassen
- Zweiter Durchlauf zeilenweise über das Bild
- Alle Label einer Klasse durch den Stellvertreter ersetzen
- Ergebnis: zusammen-hängende Regionen haben das gleiche Label
- Abbildung z.B. auf Farbpalette



Vorletzter Schritt: Connected Component Labeling



Anwendung auf anderes Ausgangsbild



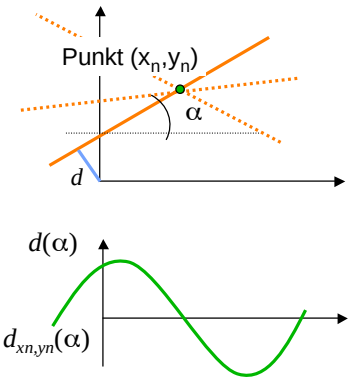
Andere häufig auftretende Probleme

- Objekte einer bestimmten Farbe finden
  - Problem: Lichtbedingungen sind variabel
  - Lösung: nicht in RGB Raum suchen, sondern im HSL Raum, und nach H filtern
- Formen finden, z.B. Kreise oder gerade Linien
  - Hough-Transformation:
    - » Voting(Abstimmungs)-Mechanismus, bei dem jeder Ort in Abhängigkeit der lokalen Information für das Modell stimmt.
    - » entwickelt für Geraden, erweiterbar für beliebige Formen.
    - » Praktisches Beispiel: automatische Rote-Augen-Reduktion

Hough Transformation (HT)

Suche von Geraden in einem Binärbild.

Geradenrepräsentation: $x \cos(\alpha) + y \sin(\alpha) - d = 0$



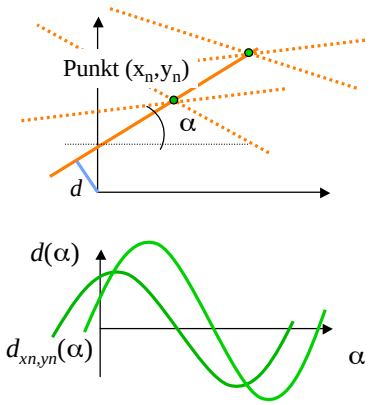
Hough-Transformation:

Suche alle Parameter (α,d) für Geraden, die durch einen Punkt (x_n,y_n) gehen

$$d(\alpha) = x_n \cos(\alpha) + y_n \sin(\alpha)$$

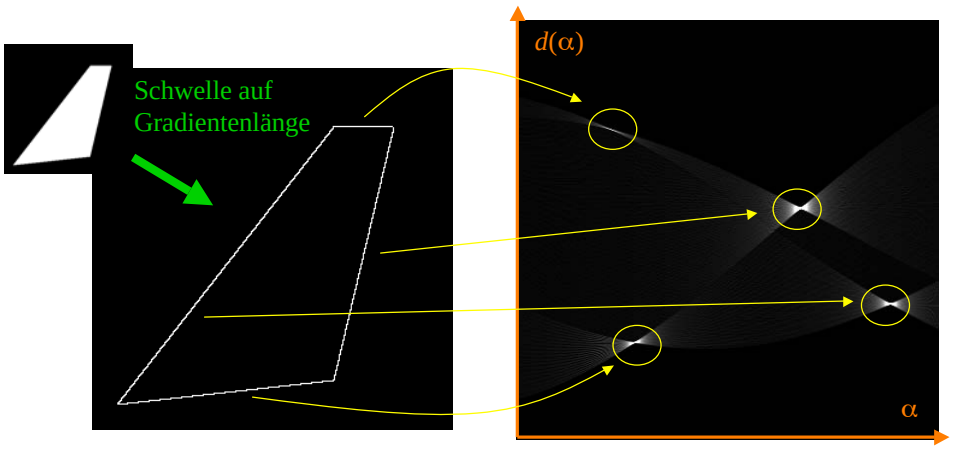
Der Raum, der durch (α,d) aufgespannt wird, heißt Hough-Raum.

Berechnung der HT



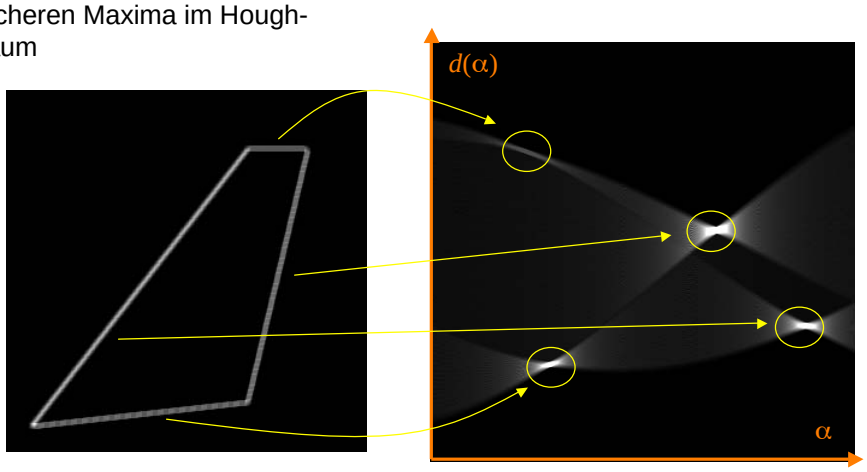
- Erzeugung eines Kantenbilds durch Schwellenwertsetzung auf Gradientenlängen.
- Diskretisierung des (α,*d*)-Raums (Zerlegung in Akkumulatoren)
- Für jeden Punkt x_n,y_n wird eine Kurve im (α ,*d*)-Raum diskretisiert.
- Jeder Akkumulator wird inkrementiert, sobald eine Kurve durch in verläuft.
- Parameter von Linien im Ortsraum sind durch (α,*d*)-Kombinationen gegeben, deren Wert (Stimmenanzahl, votes) nach Ausführung der Transformation am höchsten sind.

Hough Transformation



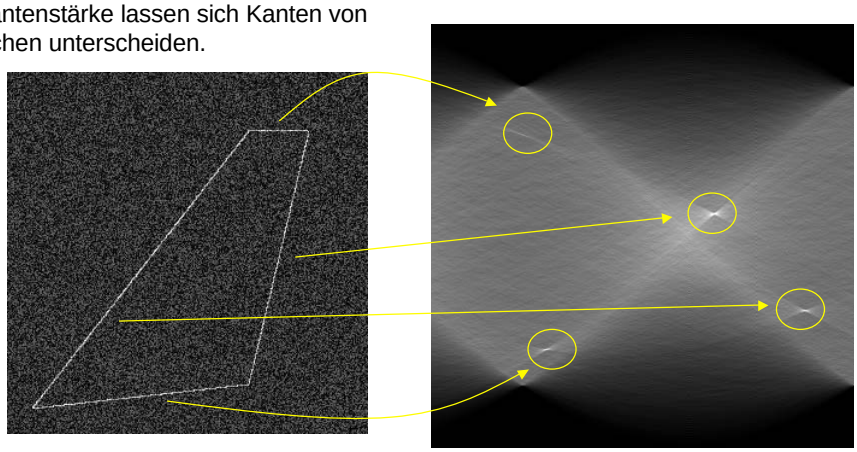
Hough Transformation

Breite Kanten führen zu flacheren Maxima im Hough-Raum



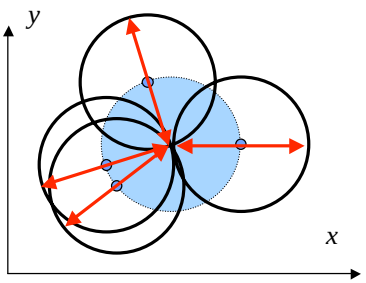
Hough Transformation in nicht-binären Bildern

Bei Gewichtung der Stimmanzahl durch die Kantenstärke lassen sich Kanten von Rauschen unterscheiden.

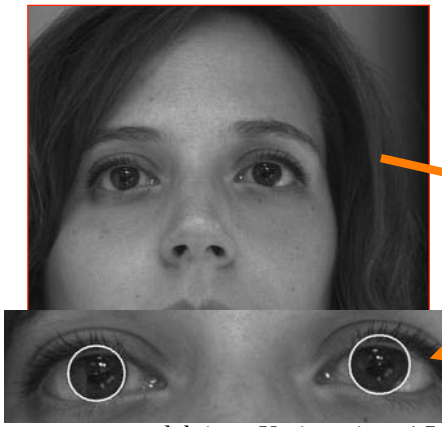


Hough Transformation für Kreise

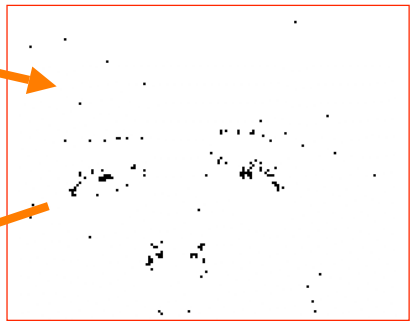
- Kreisgleichung für Kreis mit Mittelpunkt (x_c, y_c) und Radius r: $(x-x_c)^2+(y-y_c)^2-r^2=0$.
- Falls der Radius bekannt ist, ist nur der Verschiebevektor (x_c, y_c) gesucht
  - Hough-Raum = Ortsraum
  - Um jeden Kantenpunkt wird ein Kreis mit Radius r diskretisiert.
- Beschleunigung: Akkumulator wird nur in Distanz r in und entgegen der Gradientenrichtung inkrementiert.



Hough Transformation für Kreise



Akkumulatorzellen mit mehr als 33% der maximalen Stimmenanzahl.



selektierte Kreise mit r=4.5mm

Hough Transformation in HIPR2