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

Filtern im Frequenzraum
Segmentierung

Organisatorisches

- Klausuranmeldung
  - Bitte meldet euch bis 23.07.2009 in Uniworx zur Klausur an!
- Kommendes Übungsblatt (Klausurvorbereitung) möglicherweise schon früher (weiterhin keine Abgabe)

Übungsblatt 8

Übungsblatt 8

- Konvolutionsfilter:
  - Kernel ist quadratisch
  - Kann global definiert werden -> Übergabe an Superklasse:

```
#include "convolution_filter.h"

int edgeKernel[9] = {0, 1, 0,
                     1, -4, 1,
                     0, 1, 0};

class EdgesFilter : public ConvolutionFilter {
public:
    EdgesFilter()
        : ConvolutionFilter(edgeKernel, 3) {

    }

    virtual ~EdgesFilter() { }
};
```

- Konvolutionsfilter:
  - Divisor ist die Summe der Elemente des Kernels
  - Der Divisor kann 0 sein, z.B.:
    - `kernel = {0, -1, 0, -1, 4, -1, 0, -1, 0};`
    - `divisor = -1-1+4-1-1 = 0;`
  - Ist dies der Fall, wird der Divisor auf `1` gesetzt.
  - Wandert ein Teil des Kernels außerhalb des Bildes, so werden nur die Elemente berücksichtigt, die **innerhalb** des Kernels sind, z.B.:
    - `kernel = {`**`0, -1,`** `0,` **`-1, 4,`** `-1, 0, -1, 0};`
    - **Hervorgehobene** Elemente sind **innerhalb** des Bildes
    - `divisor = -1-1+4 = 2;`
  - **Wichtig:** Liegen Farbwerte außerhalb des Spektrums (bei RGB kleiner 0 oder > 255), so müssen diese abgeschnitten werden:

```
destPtr[2] = (r > 255) ? 255 : ((r < 0) ? 0 : r);
destPtr[1] = (g > 255) ? 255 : ((g < 0) ? 0 : g);
destPtr[0] = (b > 255) ? 255 : ((b < 0) ? 0 : b);
```

Übungsblatt 8

- Morphologische Filter:
  - **Dilatation:** Finde das Maximum **jedes** Farbanteils **innerhalb** des Strukturelements und setze diese Werte auf das Ankerpixel.
  - **Erosion:** Finde das Minimum **jedes** Farbanteils **innerhalb** des Strukturelements und setze diese Werte auf das Ankerpixel.
  - Quadratischer und kreisförmiger Kernel:

```
int quadraticKernel[81] = {1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1,
                           1, 1, 1, 1, 1, 1, 1, 1, 1};
```

```
int circularKernel[81] = {0, 0, 0, 1, 1, 1, 0, 0, 0,
                          0, 0, 1, 1, 1, 1, 1, 0, 0,
                          0, 1, 1, 1, 1, 1, 1, 1, 0,
                          1, 1, 1, 1, 1, 1, 1, 1, 1,
                          1, 1, 1, 1, 1, 1, 1, 1, 1,
                          1, 1, 1, 1, 1, 1, 1, 1, 1,
                          0, 1, 1, 1, 1, 1, 1, 1, 0,
                          0, 0, 1, 1, 1, 1, 1, 0, 0,
                          0, 0, 0, 1, 1, 1, 0, 0, 0};
```

- Hough-Transformation:
  - Größe des Hough-Raums:
    - **Höhe:** Anzahl der Schritte pro Grad mal 180
    - **Breite:** Diagonale des Bildes (gemessen vom Mittelpunkt können Linien eine maximale Entfernung “Hälfte der Diagonale haben - wir lassen aber negative Entfernungen zu, da nur 180 Grad beim Winkel betrachtet werden).
  - Berechnung pro Pixel:

```
for(int theta = 0; theta < m_houghHeight; theta++) {
    int radius = (int)(m_cosMap[theta] * x
        - m_sinMap[theta] * y) + halfHoughWidth;
    if((radius < 0) || (radius >= houghWidth)) {
        continue;
    }

    houghMap[theta][radius]++;
}
```

Übungsblatt 8

- Hough-Transformation:
  - **Zunächst:** Finden des Maximums im Hough-Raum
  - Finden der Linien:
    - Gehe über den gesamten Hough-Raum
    - Sobald ein Punkt über 50 % des Maximums liegt, muss noch im Umkreis nach einem eventuell **höheren** Wert gesucht werden
    - Sobald das korrekte lokale Maximum gefunden wurde, kann die Linie bestimmt und gezeichnet werden.
    - Diese Schritte werden dann je nach Anzahl der Linien wiederholt
  - Zeichnen der Linien:
    - Gegeben ist bereits der Winkel und der Abstand
    - **Idee:**
      - Verschiebe das Koordinatensystem auf den Bildmittelpunkt
      - Drehe es nun um den Winkel der Linie und verschiebe es dann um den Abstand (lokale x-Achse)
      - Zeichne die Linie nun entlang der y-Achse

Übungsblatt 8

- Hough-Transformation:
  - Linien sind höchstens so lang wie die Diagonale (hier `houghWidth` bzw. `halfHoughWidth` für jeweils die Hälfte der Diagonale)

```
QPainter painter(&newImage);
painter.setPen(Qt::red);
painter.translate(newImage.width() / 2, newImage.height() / 2);

for(int i = 0; i < (int)m_detectedLines.size()
     && i < m_linesToDetect; i++) {
    HoughLine* line = m_detectedLines.at(i);

    painter.rotate(-line->theta());
    painter.translate(line->radius(), 0);

    painter.drawLine(0, -halfHoughWidth, 0, halfHoughWidth);

    painter.translate(-line->radius(), 0);
    painter.rotate(line->theta());
}

painter.translate(-newImage.width() / 2, -newImage.height() / 2);
painter.end();
```

Filtern im Frequenzraum

Filtern im Frequenzraum

- Beim letzten Mal: Konvertierung eines Bilds in den Frequenzraum per FFT
- Dadurch lassen sich bestimmte Filteroperationen (Hoch-, Tief-, Bandpass) deutlich effizienter durchführen
- Anwendung der Filter im Frequenzraum kann deutlich anschaulicher sein (Vergleiche Tiefpassfilter vs. Anwendung des entsprechenden Konvolutionskernels)
- Generell können Konvolutionsfilter sowohl im Orts- als auch Frequenzraum durchgeführt werden

Filtern im Frequenzraum

- Filtern wird durch Pixel Multiplikation des Spektrums mit einer „Filter Transfer Function“ realisiert:

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

- F(u,v) ist das Bild im Frequenzraum, H(u,v) ist die Fouriertransformation des Original-Kernels

- Achtung: Manipulation der Phase kann Bildinformationen zerstören!
- Deswegen sind die meisten Filter sog. „zero-phase-shift filter“, d.h. sie verändern nur Amplitude, nicht Phase

(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Zur Konversion zwischen Orts- und Frequenzraum können Fouriertransformation und inverse FT genutzt werden
- Das Konvolutionstheorem sagt folgendes aus:

$$f * h \Leftrightarrow FH$$

- Dabei steht die linke Seite für die Konvolution im Ortsraum (f Bild, h Kernel) und die rechte für die gleiche Konvolution im Frequenzraum (F Bild, H Kernel)
- Übergang von links nach rechts per FT bzw. Inverser FT
- Wichtig: Beide Konvolutionsanwendungen sind äquivalent!

(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Generelles Vorgehen:
  - Konversion des Bilds in den Frequenzraum per FT (f => F)
  - Konversion des Konvolutionskernels in den Frequenzraum per FT (h => H)
  - Multiplikation von F mit H (F * H)
  - Umwandlung des Bilds in den Ortsraum per inverser FT (F' => f')

(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Performancevergleich:
- Konvolution im Ortsraum: O (N^4)
- Konvolution im Frequenzraum:
  - f => F: O(N^2 * log(N)) (FFT)
  - h => H: O(N^2 * log(N)) (FFT)
  - F * H: O(N^2)
  - F' => f': O(N^2 * log(N)) (Inverse FFT)

- Filtern im Frequenzraum lohnt sich:
  - falls mehrere Filter nacheinander angewendet werden (Konversion und Rückkonversion muss nur einmal durchgeführt werden)
  - falls N sehr groß ist

(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Beispielfilter: Tiefpassfilter (Weichzeichner):
- Im Ortsraum hatten wir diesen Konvolutionskernel:
- Im Frequenzraum müssen wir nur alle Frequenzbereiche, die einen bestimmten Abstand vom Zentrum des Spektrums haben abschneiden:

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

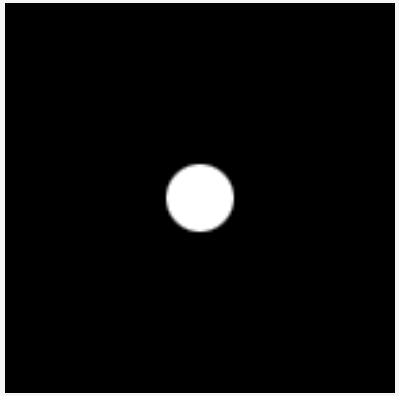
$$H(u,v) = \begin{cases} 1 & ,r(u,v) \leq r_0, \\ 0 & ,r(u,v) > r_0, \end{cases}$$

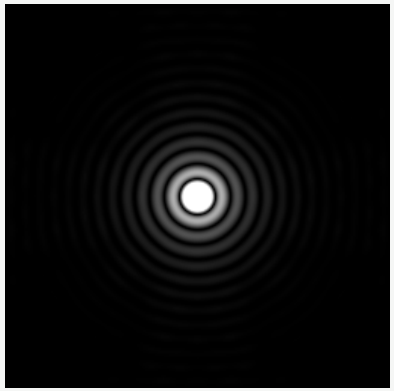
$$r(u,v) = \sqrt{u^2 + v^2}$$

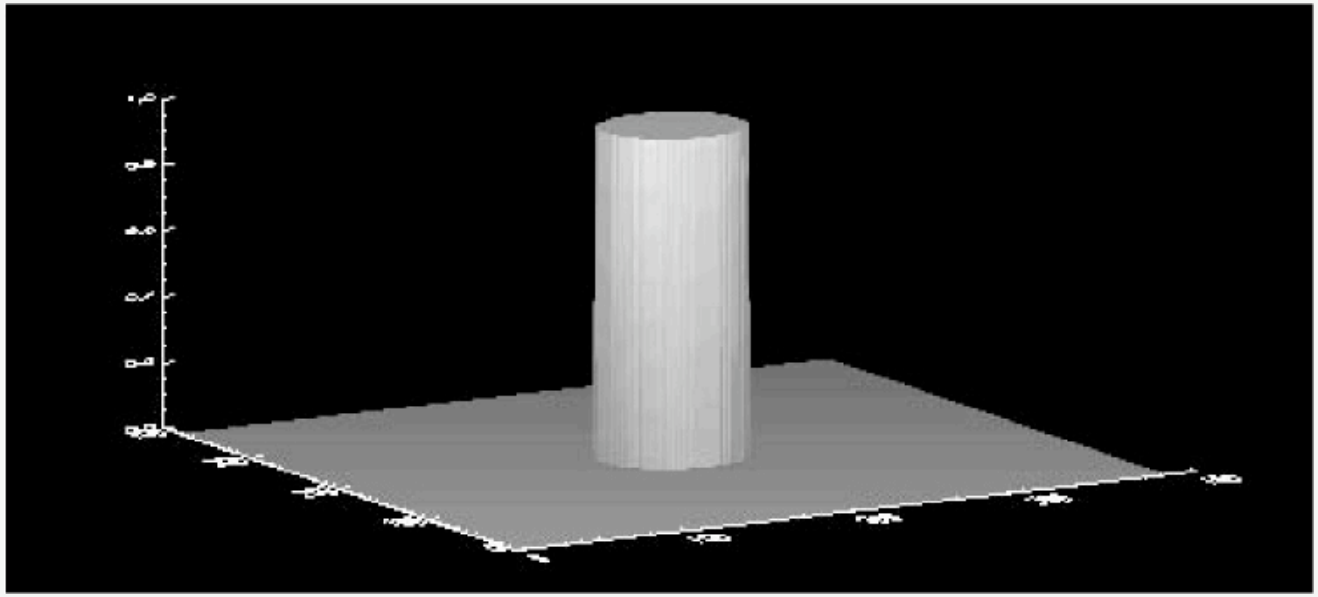
(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Beispielfilter: Tiefpassfilter (Weichzeichner)



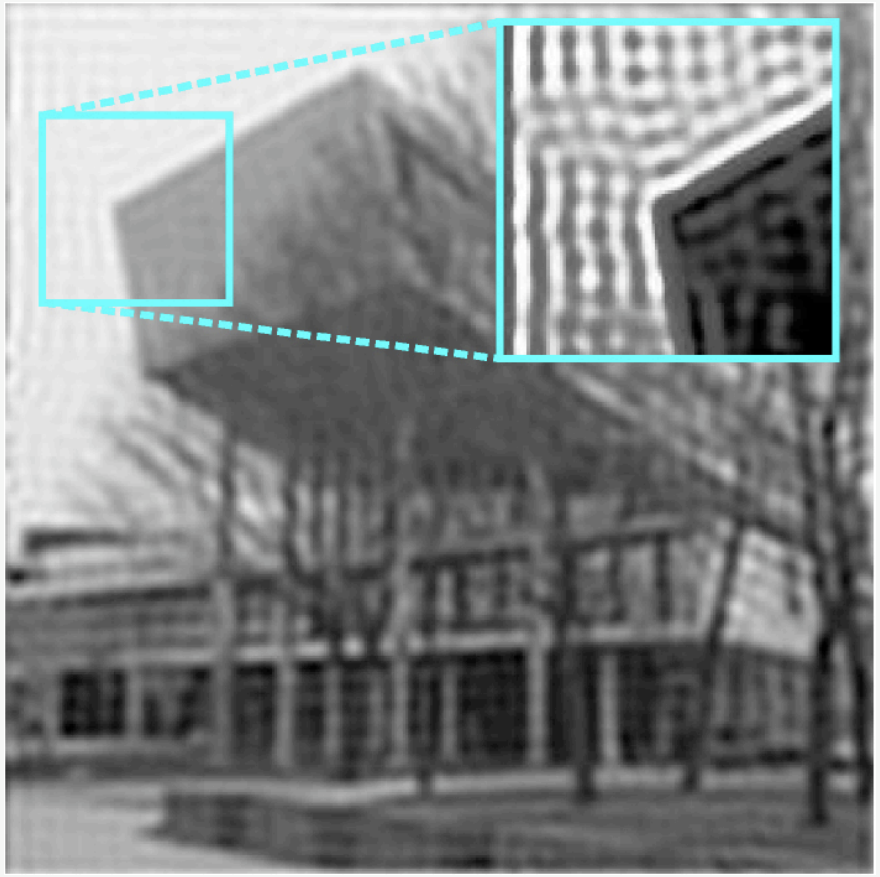




(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

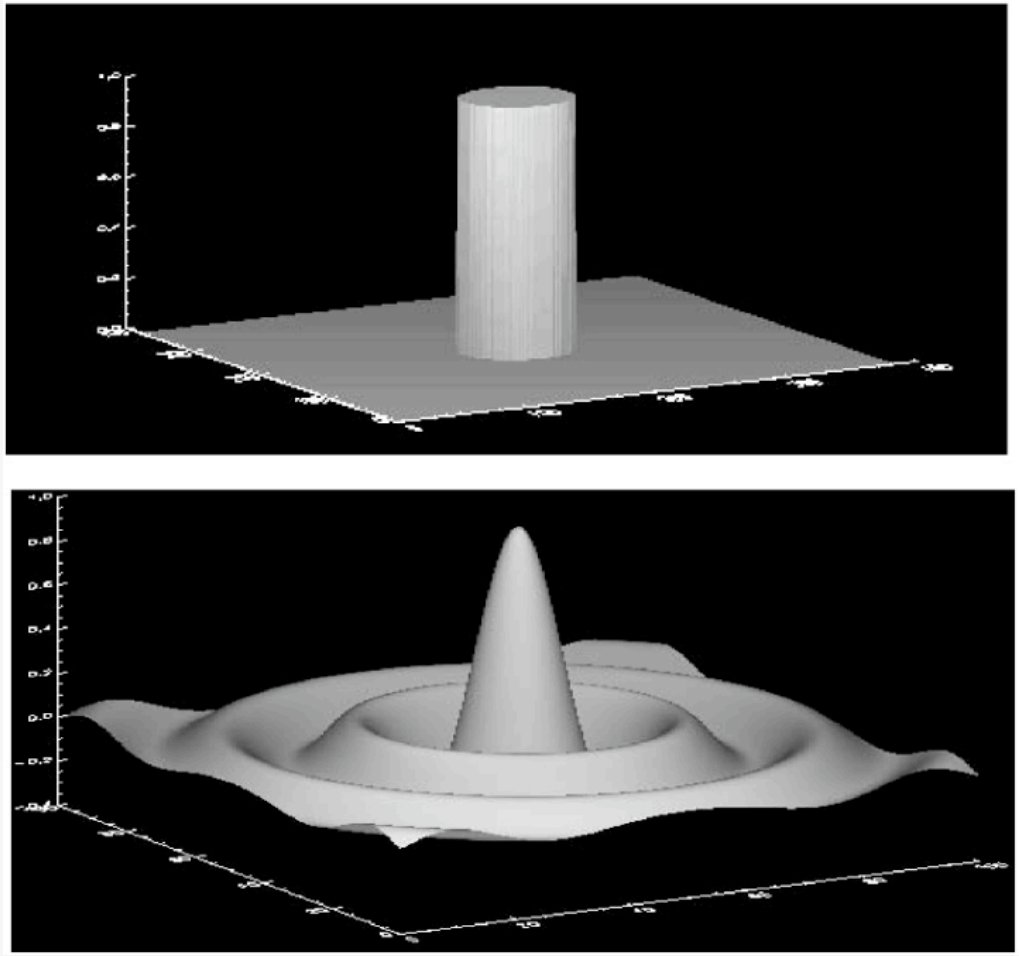
- Beispielfilter: Tiefpassfilter (Weichzeichner)
- Problem: Artefakte! (ringing)



(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

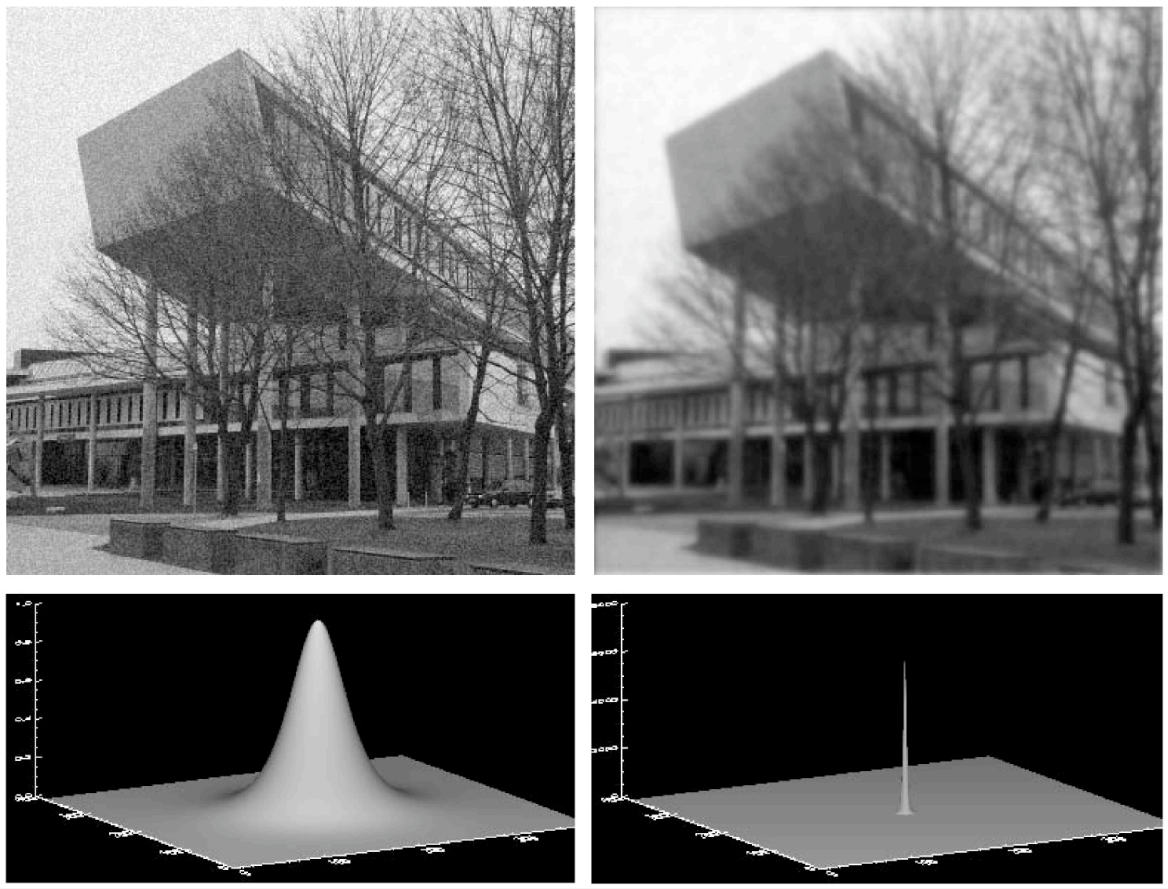
- Grund für die Artefakte:
- Das perfekte Abschneiden im Frequenzraum ist äquivalent zu dem untenstehenden Konvolutionskernel
- Dessen negative Anteile erzeugen an Kanten im Bild Verzerrungen



(Quelle: Efford - Digital Image Processing in Java)

Filtern im Frequenzraum

- Lösung: Butterworth-Filter (weiche Übergänge im Frequenz- und Ortsraum)



(Quelle: Efford - Digital Image Processing in Java)

Segmentierung

Segmentierung

(Quelle: http://www.flickr.com/photos/tauntingpanda/14782257/)

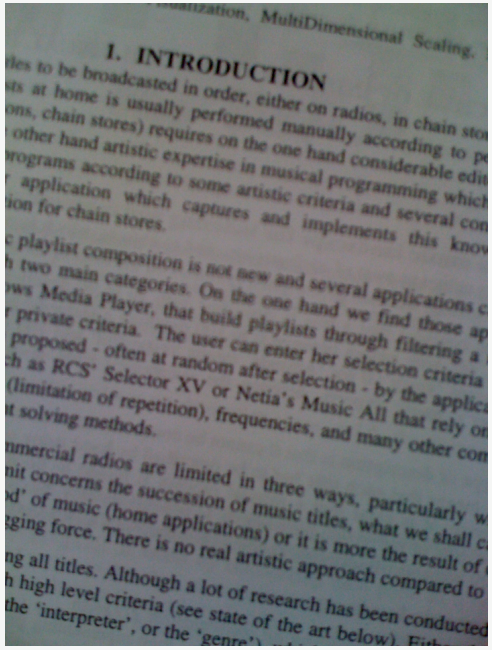
Segmentierung

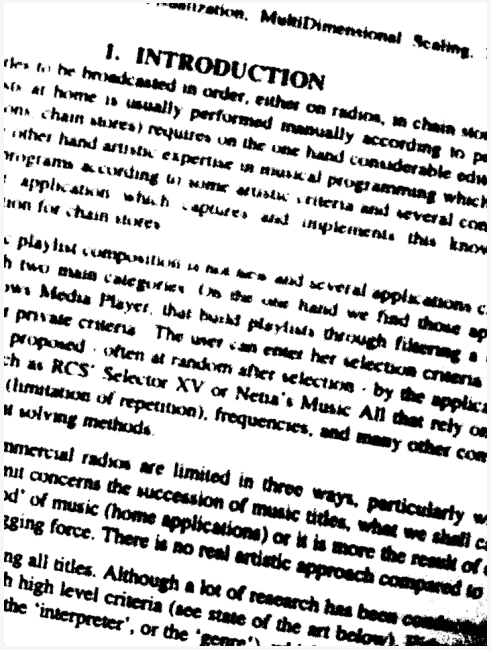
- Segmentierung zerlegt Pixelmengen in Bedeutungsträger, d.h. Umwandlung in binäres Bild (Vordergrund: 1, Hintergrund: 0)
- Bindeglied zwischen low-level (Pixel) und high-level (Objekte) Operationen
- Methoden lassen sich grob in zwei Kategorien unterteilen:
  - Non-contextual: Ausnutzung von globalen Eigenschaften (z.B. Grauwert)
  - Contextual: Ausnutzung der Beziehungen zwischen Pixeln
- (Meist) Einfach für Menschen, schwierig für Maschinen
- Aktives Forschungsgebiet (Bilderkennung, OCR)

(Quelle: Efford - Digital Image Processing in Java)

Thresholding

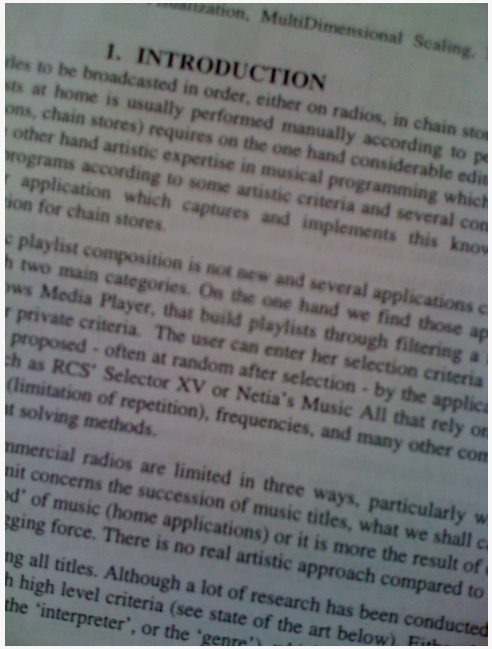
- Sehr einfache Segmentierungstechnik: Thresholding (Schwellenwertbildung)
- Basierend auf dem Grauwert eines Pixels wird er in Vordergrund oder Hintergrund eingeteilt
- Dabei kann entweder oberhalb eines bestimmten Grauwerts abgeschnitten oder ein bestimmtes Band benutzt werden





Thresholding

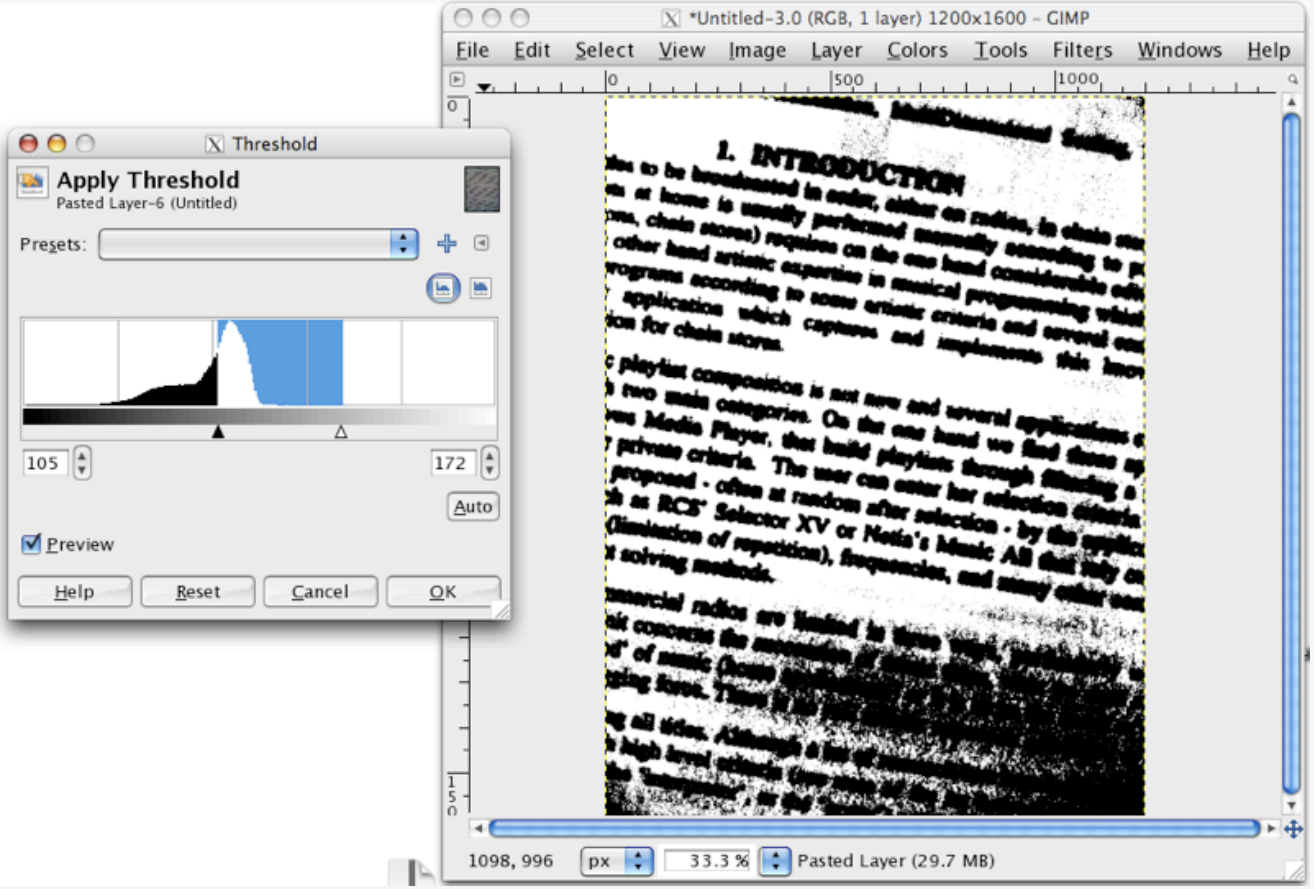
- Funktioniert recht gut bei simplen Bildern wie Texten.
- Aber:
  - Nachbarschaft eines Pixels wird nicht beachtet
  - Funktioniert nicht wenn Vordergrundobjekte unterschiedliche Helligkeiten haben
  - Threshold muss richtig gewählt werden:



Thresholding

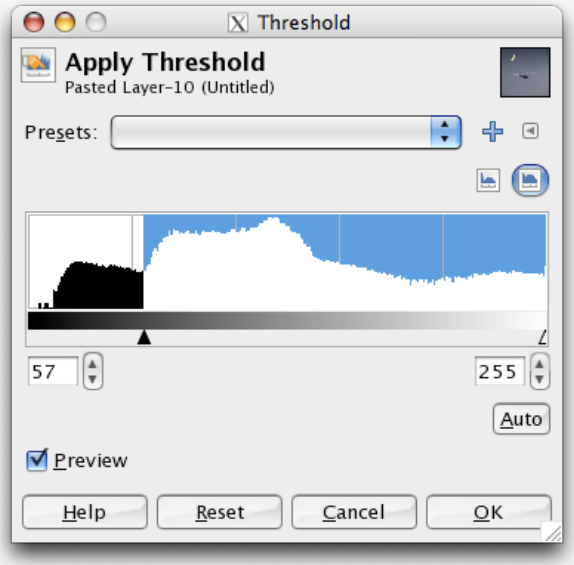
Thresholding

- Um den richtigen Threshold zu finden kann man den Benutzer diesen interaktiv auswählen lassen (z.B. in Photoshop, Gimp)



Thresholding

- Alternative: Analyse des Bildhistogramms
- Haben zwei Objekte klar unterscheidbare Grauwerte (= Helligkeit) enthält das zugehörige Histogramm Minima
- Diese Minima können als Thresholds genutzt werden



(Quellen: Tönnies - Grundlagen der Bildverarbeitung, http://www.flickr.com/photos/visbeek/3163916678/)

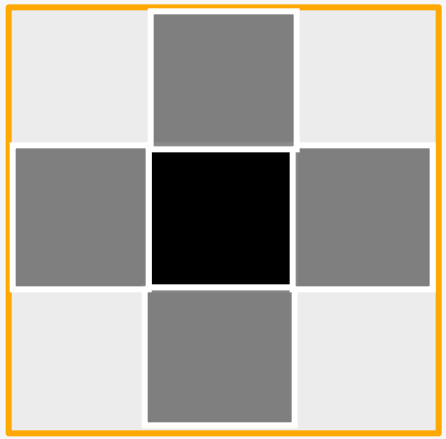
Segmentierung

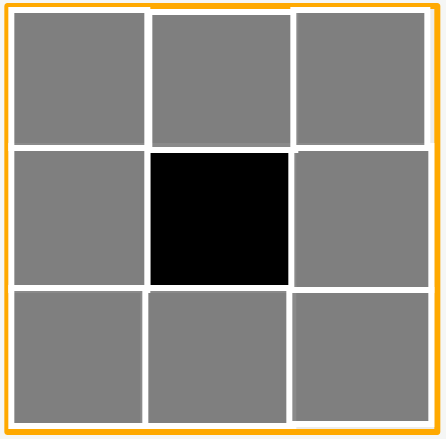
- Verfahren, die ebenfalls den Kontext in Betracht ziehen, können bessere Ergebnisse erzielen.
- Zusätzlich lässt sich damit der Vordergrundbereich in verschiedene, getrennte Objekte mit eigenen “Labels” aufteilen
- Um solche Labels für Objekte speichern zu können reicht ein binäres Bild nicht mehr
- Möglichkeiten sind Graustufenbilder (eine Graustufe pro Objekt) oder zusätzliche Datenstrukturen

(Quelle: Efford - Digital Image Processing in Java)

Nachbarschaft

- Wichtig bei kontextuellen Verfahren: Die Nachbarschaft eines Pixels
- Verbreitete Nachbarschaftsmaße: 4er und 8er Nachbarschaft

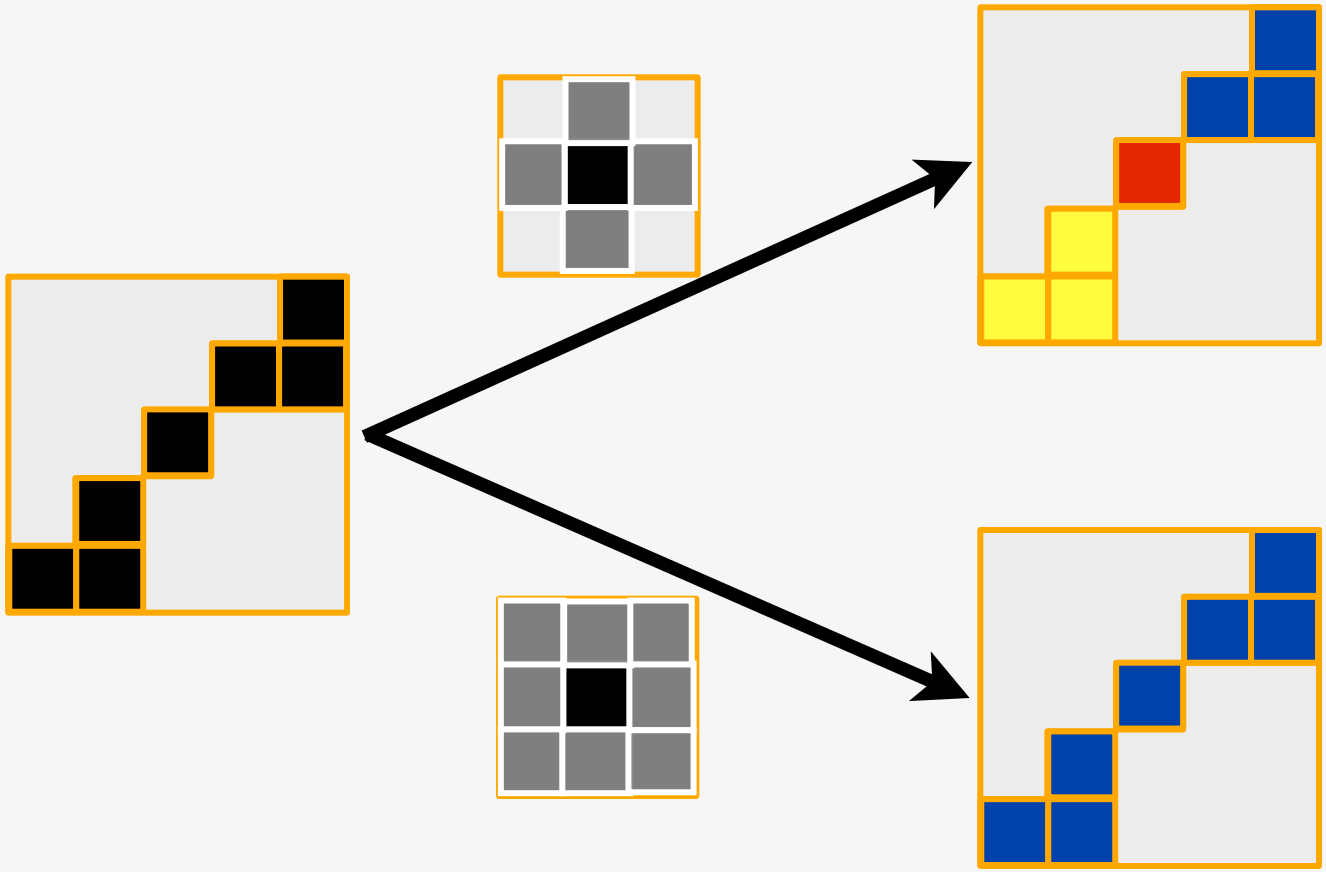




(Quelle: Efford - Digital Image Processing in Java)

Nachbarschaft

- Je nach verwendetem Nachbarschaftsmaß ergeben sich unterschiedliche Aufteilungen:

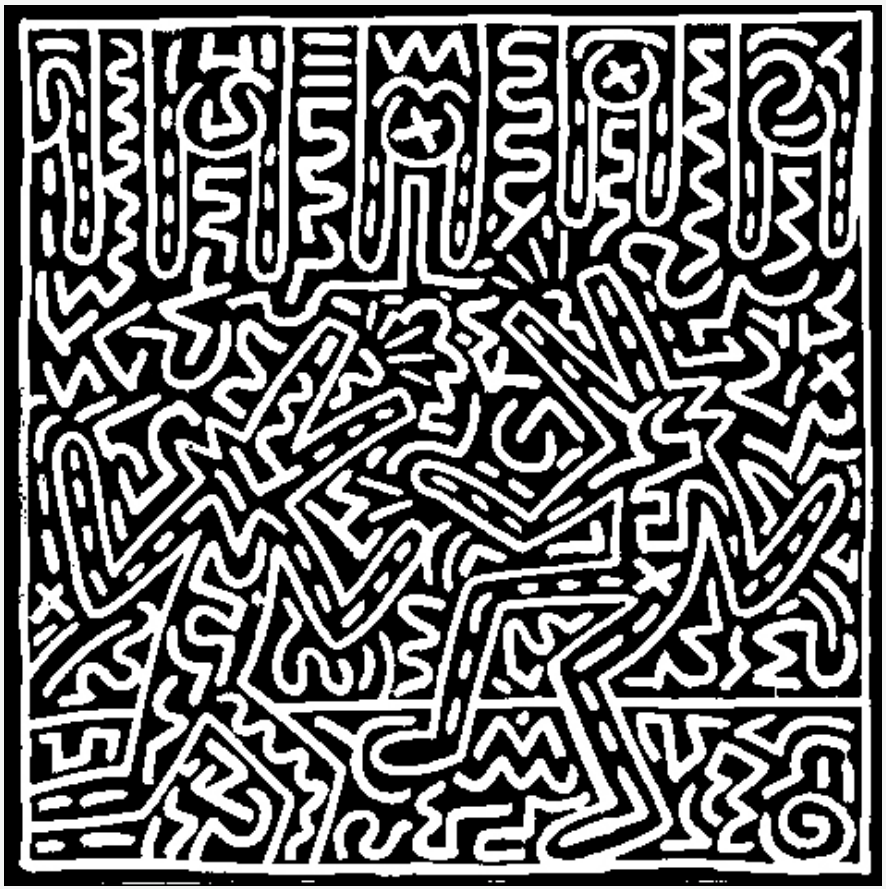


Grassfire

- Ein einfacher Algorithmus zur Aufteilung eines Binärbilds in Objekte ist Grassfire:
  - Scannt Bild zeilenweise
  - Beim Auffinden eines Nicht-Hintergrundpixels wird die rekursive Labeling Funktion aufgerufen.
  - Labeling Startet „Buschfeuer“:
  - Startpixel wird verbrannt (d.h. auf Hintergrund gesetzt)
  - 4- oder 8-Nachbarn werden besucht
  - Ist einer der Nachbarn im Vordergrund wird rekursiv weiter gezündelt.
  - Am Ende wird die Labelnummer erhöht und der Scan-Vorgang geht weiter

(Quelle: Efford - Digital Image Processing in Java)

Grassfire



Interaktive Segmentierung

- Segmentierung zerlegt Pixelmengen in Bedeutungsträger
- Modell: Erwartete Bedeutung
- => Widerspruch!
- Modellbasierte Segmentierung:
  - Vorwissen wird zur Suche verwendet
  - Modellwissen wird interaktiv vom Benutzer eingebracht
  - Instanzen eines Objektes werden im Bild gesucht

Region Growing

Region Growing

Region Growing

- Grassfire funktioniert nur auf Binärbildern
- Region Growing ist ein iterativer Flood-Fill Algorithmus mit einem (interaktiv) festgelegten Startpunkt und funktioniert auf beliebigen Bildern
- Voraussetzung ist eine überprüfbare Homogenitätsbedingung (z.B. Grauwert):

$$H(R) = \begin{cases} TRUE & ,|f(j,k) - \mu_R| \leq \Delta, \\ FALSE & ,sonst \end{cases}$$

$$H(R) = \begin{cases} TRUE & ,|f(j,k) - f(m,n)| \leq \Delta, \\ FALSE & ,sonst \end{cases}$$

(Quelle: Efford - Digital Image Processing in Java)

Region Growing

- Annahmen und Ziele:
  - Homogenität innerhalb der Region ist größer als außerhalb (es gibt Kanten).
  - Selektion eines einzelnen Gebietes.
  - Homogenitätsverhältnisse an anderen Orten interessieren nicht.
- Algorithmus:
  - Im Grunde genommen identisch zu Grassfire/Labelling
  - 4er oder 8er Nachbarschaft möglich
  - Wird Grauwert (μ_R) verwendet muss der durchschnittliche Grauwert bei jedem Pixel angepasst werden

Region Growing

- Beispiel:
  - Zwei Seeds, 8er Nachbarschaft, Grauwertdurchschnitt mit $\Delta = 3$

0	0	5	6	7
1	1	5	8	7
0	1	6	7	7
2	0	7	6	6
0	1	5	6	5

0	0	5	6	7
1	1	5	8	7
0	1	6	7	7
2	0	7	6	6
0	1	5	6	5

0	0	5	6	7
1	1	5	8	7
0	1	6	7	7
2	0	7	6	6
0	1	5	6	5

(Quelle: Efford - Digital Image Processing in Java)

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