Konturen von Regionen

Prof. Dr. Klaus Jung

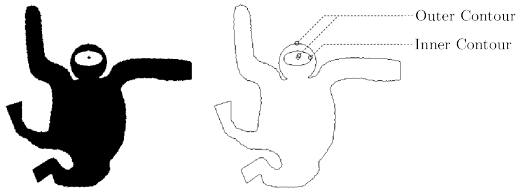
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Inhalt

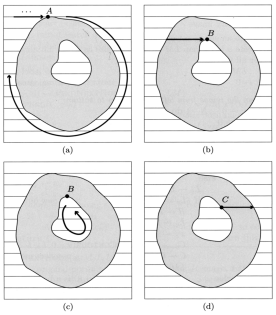
1. Kontoren finden mit Algorithmus nach
   - Burger, Wilhelm; Burge, Mark James *Digitale Bildverarbeitung, Eine Einführung mit Java und ImageJ*
2. Kontoren finden mit Algorithmus nach
   - *potrace* (Peter Selinger, 2003)
   - http://potrace.sourceforge.net/
   - Artikel mit Erläuterung des Algorithmus: http://potrace.sourceforge.net/potrace.pdf

Konturen von Regionen

- Innere und äußere Konturen
  - Beliebig viele Löcher
  - Kleinere Regionen innerhalb von Löchern



1. Kombinierte Regionenmarkierung und Konturfindung



- (a) Übergang Hintergrund auf nicht markierten Vordergrund
  - Neues Label
  - Äußere Kontur finden (Label eintragen
  - Angrenzenden Hintergrund auf –1
- (b) Übergang Hintergrund auf markierten Vordergrund
  - Markierung fortsetzen
- (c) Übergang markierter Vordergrund auf nicht markierten Hintergrund
  - Innere Kontur finden (Label eintr.)
  - Angrenzenden Hintergrund auf –1
- (d) wie (b)

Algorithmus

```
1:  CombinedContourLabeling (I)
        I: binary image
2:  Create an empty set of contours: C ← {}
3:  Create a label map LM of the same size as I and initialize:
4:  for all (u, v) do
5:      LM(u, v) ← 0                                  ▷ label map LM
6:  R ← 0                                             ▷ region counter R
7:  Scan the image from left to right, top to bottom:
8:  for v ← 0 ... N−1 do
9:      L_c ← 0                                       ▷ current label L_c
10:     for u ← 0 ... M−1 do
11:         if I(u, v) is a foreground pixel then
12:             if (L_c ≠ 0) then                     ▷ continue existing region
13:                 LM(u, v) ← L_c
14:             else
15:                 L_c ← LM(u, v)
16:                 if (L_c = 0) then                 ▷ hit new outer contour
17:                     R ← R + 1
18:                     L_c ← R
19:                     x_S ← (u, v)
20:                     C_outer ← TraceContour(x_S, 0, L_c, I, LM)
21:                     C ← C ∪ {C_outer}             ▷ collect new contour
22:                     LM(u, v) ← L_c
23:         else                                      ▷ I(u, v) is a background pixel
24:             if (L_c ≠ 0) then
25:                 if (LM(u, v) = 0) then            ▷ hit new inner contour
26:                     x_S ← (u−1, v)
27:                     C_inner ← TraceContour(x_S, 1, L_c, I, LM)
28:                     C ← C ∪ {C_inner}             ▷ collect new contour
29:                 L_c ← 0
30: return (C, LM).                                   ▷ return the set of contours and the label map
```

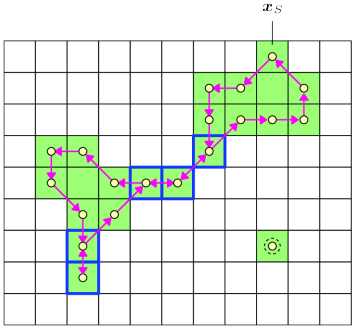
Gleiche Größe wie das Bild *I(u,v)*.
Wird mit Labeln L_c gefüllt.
Wir arbeiten nicht „in place“.

(b) und (d)

(a)

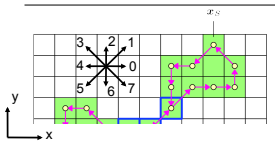
(c)

Pfad einer Kontur (1/2)

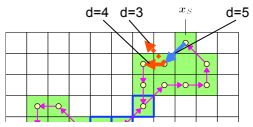


- Breite der Region kann 1 Pixel werden
  - Kriterium für das Erreichen des Startpunktes x_s definieren!
- Singuläre Pixel

Pfad einer Kontur (2/2)



Finden eines neuen Konturpunktes mit Richtung d_{neu} = (d+6) mod 8



d	Δx	Δy
0	1	0
1	1	1
2	0	1
3	-1	1
4	-1	0
5	-1	-1
6	0	-1
7	1	-1

Quelle: Kai Uwe Barthel

Kontur verfolgen

```
1:  TraceContour(x_S, d_S, L_C, I, LM)
        x_S: start position
        d_S: initial search direction
        L_C: label for this contour
        I: image
        LM: label map
2:  Create an empty contour C
3:  (x_T, d) ← FindNextNode(x_S, d_s, I, LM)
4:  Append(C, x_T)                          ▷ add x_T to contour C
5:  x_p ← x_S                               ▷ previous position x_p = (u_p, v_p)
6:  x_c ← x_T                               ▷ current position x_c = (u_c, v_c)
7:  done ← (x_S = x_T)                      ▷ isolated pixel?
8:  while (¬done) do
9:      LM(u_c, v_c) ← L_C
10:     (x_n, d) ← FindNextNode(x_c, (d+6) mod 8, I, LM)
11:     x_p ← x_c
12:     x_c ← x_n
13:     done ← (x_p = x_S ∧ x_c = x_T)      ▷ back at starting position?
14:     if (¬done) then
15:         Append(C, x_n)                  ▷ add point x_n to contour C
16: return C .                              ▷ return this contour
```

Gleiche Größe wie das Bild *I(u,v)*. Wird mit Labeln L_c gefüllt

Neuen Konturpunkt finden

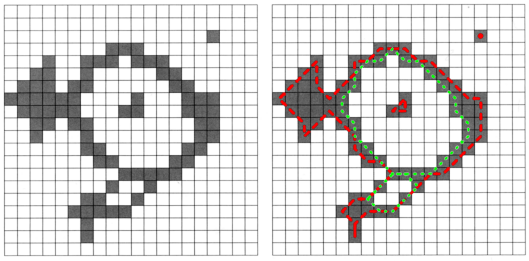
```
17: FindNextNode(x_c, d, I, LM)
        x_c: original position, d: search direction, I: image, LM: label map
18:     for i ← 0 ... 6 do                  ▷ search in 7 directions
19:         x' ← x_c + Delta(d)             ▷ x' = (u', v')
20:         if I(u', v') is a background pixel then
21:             LM(u', v') ← −1             ▷ mark background as visited (−1)
22:             d ← (d + 1) mod 8
23:         else                            ▷ found a non-background pixel at x'
24:             return (x', d)
25:     return (x_c, d) .                   ▷ found no next node, return start position
```

26: Delta(d) = (Δx, Δy), wobei

d	0	1	2	3	4	5	6	7
Δx	1	1	0	−1	−1	−1	0	1
Δy	0	1	1	1	0	−1	−1	−1

-1 verhindert, dass eine Kontur mehrmals gefunden wird.

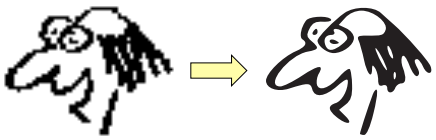
Beispiel



(a) (b)

2. Konturen finden nach *potrace*

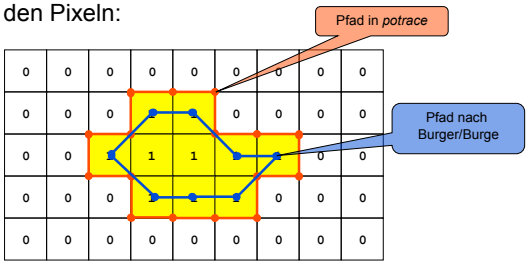
- Ziel:
  - Vektorisierung nach *potrace* implementieren
- Vektorisierung
  - Umwandeln Rastergrafik → Vektorgrafik



Pixel Bezier-Kurven

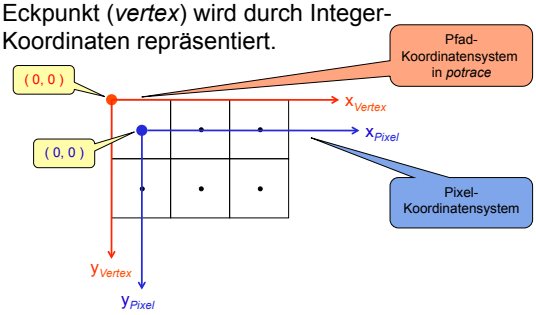
Darstellung der Konturen

- Bei *potrace* sind die Konturen Pfade zwischen den Pixeln:



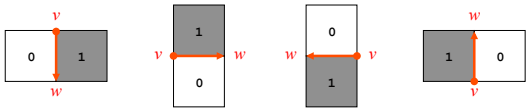
Koordinaten Pixel / Eckpunkte

- Eckpunkt (*vertex*) wird durch Integer-Koordinaten repräsentiert.



Definition: Kante (*edge*)

- Zwei Eckpunkte (*vertices*) v, w bilden eine Kante (*edge*), genau dann, wenn
  - Euklidischer Abstand(v, w) = 1 und
  - Verbindung $\overline{vw}$ von v nach w separiert zwei Pixel so, dass
    - Vordergrund (schwarz) ist links
    - Hintergrund (weiß) ist rechts

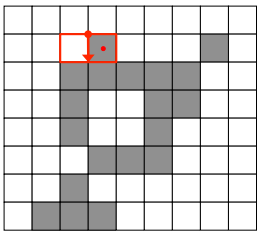


Definition: Pfad (*path*)

- Die Sequenz $p = \{v_0,...,v_n\}$ heißt Pfad, wenn gilt:
  - Für jedes $i = 0,...,n-1$ ist $\overline{v_i v_{i+1}}$ eine Kante
  - Keine Kante kommt doppelt vor
- Ein Pfad $p = \{v_0,...,v_n\}$ heißt geschlossen (*closed*), wenn
  - $v_n = v_0$

Pfad bestimmen (1)

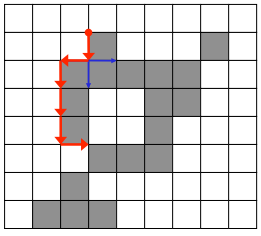
- Finde den ersten Vordergrundpixel
  - *Scanline* Reihenfolge



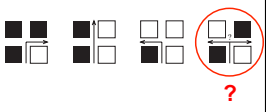
- Erster schwarzer Pixel mit Vorgänger
- Setze zwei Eckpunkte $\overline{v_0 v_1}$, so dass eine Kante entsteht

Pfad bestimmen (2)

- Kanten hinzufügen
  - Drei potentielle Richtungen

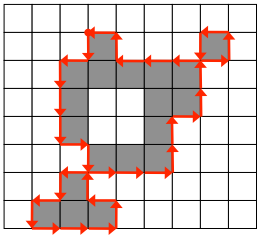


- Wähle die Richtung so, dass schwarzer Pixel links zur Laufrichtung liegt



Pfad bestimmen (3)

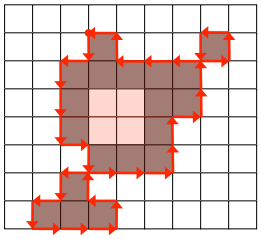
- Abbiegevorschrift
  - Optionen: In Laufrichtung immer links / immer rechts



- Wähle *immer rechts*, um 8er Nachbarschaft zu erreichen
- Behandle Rand wie weißen Pixel
- Fertig, wenn am Anfang angekommen

Pfad bestimmen (4)

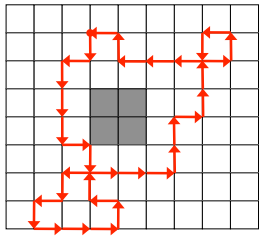
- Alle Pixel im Innern invertieren
  - Ziel: Innere Konturen finden



- Innere Pixel

Pfad bestimmen (5)

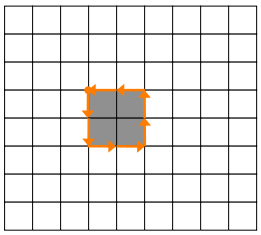
- Alle Pixel im Innern invertieren
  - Ziel: Innere Konturen finden



- Innere Pixel invertiert

Alle Pfade bestimmen (6)

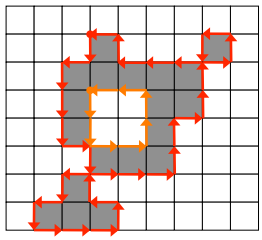
- Pfadsuche wiederholen



- Wiederhole Suche nach einem Pfad, bis kein schwarzer Pixel mehr im Bild

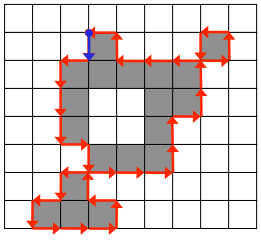
Ergebnis

- Zwei Konturen gefunden



Innere Pixel invertieren

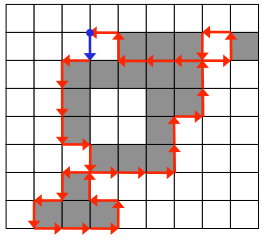
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist größer geworden

Innere Pixel invertieren

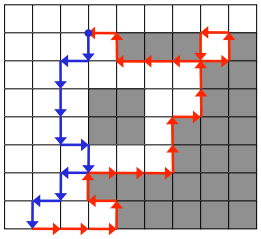
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist größer geworden
- → Invertiere Zeile ab aktueller x-Position

Innere Pixel invertieren

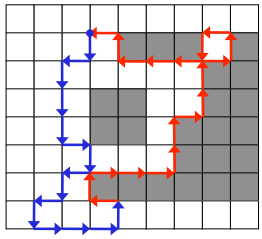
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist größer geworden
- → Invertiere Zeile ab aktueller x-Position

Innere Pixel invertieren

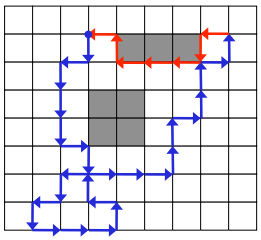
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist kleiner geworden
- → Invertiere Zeile ab aktueller x-Position

Innere Pixel invertieren

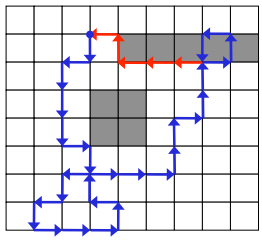
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist kleiner geworden
- → Invertiere Zeile ab aktueller x-Position

Innere Pixel invertieren

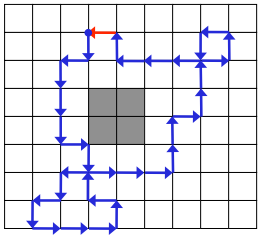
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist größer geworden
- → Invertiere Zeile ab aktueller x-Position

Innere Pixel invertieren

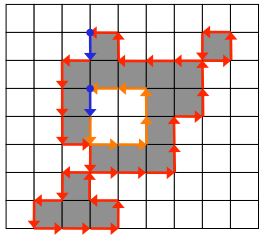
- Folge den gefundenen Eckpunkten
  - Immer wenn sich y-Koordinate ändert, invertiere „Rest der Zeile“



- y ist kleiner geworden
- → Invertiere Zeile ab aktueller x-Position

Innere und Äußere Kontur

- Bestimmen durch Farbe im *Originalbild*
  - Betrachte erste Kante



- Schwarz ist in Laufrichtung links: → Äußere Kontur
- Schwarz ist in Laufrichtung rechts: → Innere Kontur

Abbiegevorschrift

- Durch das Invertieren der Pixel:
  - *Immer rechts* bekommt andere Bedeutung
- Mögliche Modifikation:
  - *Immer schwarz*
  - Definiere Richtung basierend auf der Farbe im *Originalbild* so, dass schwarze Pixel zusammenhängen

Zusammenfassung

- Verschiedene Algorithmen zum Finden von Konturen
  - Grobe Idee meistens klar
  - Aufwand steckt im Detail
- Definition des Zusammenhangs ist wichtig
  - 4er oder 8er Nachbarschaft
- Verschiedene Definition von Kanten
  - Auf den Pixeln laufend
  - Zwischen den Pixeln laufend