


Regionen in Binärbildern

Industrielle Bildverarbeitung, Vorlesung No. 9¹

M. O. Franz

¹ falls nicht anders vermerkt, sind die Abbildungen entnommen aus Burger & Burge, 2005. 

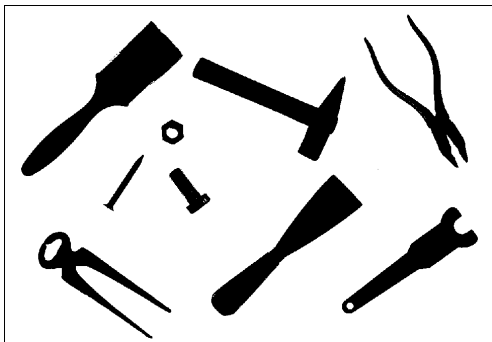
Übersicht

- 1 Auffinden von Bildregionen
- 2 Konturen von Regionen
- 3 Repräsentation von Bildregionen
- 4 Eigenschaften binärer Bildregionen

Übersicht

- 1 **Auffinden von Bildregionen**
- 2 Konturen von Regionen
- 3 Repräsentation von Bildregionen
- 4 Eigenschaften binärer Bildregionen

Regionen in Binärbildern



Aufgaben:

- Auffinden von verbundenen Regionen gleicher Pixelfarbe
- Extraktion von Konturen
- Repräsentation der Bildregionen
- Beschreibung der Regionen durch geeignete Kennzahlen

Auffinden von Bildregionen

Regionenmarkierung (*region labeling* oder *coloring*):

- Bestimmung der Anzahl der Regionen im Bild
- Bestimmung der Position der Regionen im Bild
- Bestimmung der Zugehörigkeit der einzelnen Pixel zu den Bildregionen

Vorher festzulegen: Art der Nachbarschaft (4er oder 8er) \Rightarrow führt zu unterschiedlichen Regionen.

Ergebnis der Regionenmarkierung:

$$I(u, v) = \begin{cases} 0 & \text{Hintergrundpixel (background)} \\ 1 & \text{Vordergrundpixel (foreground)} \\ 2, 3, \dots & \text{Regionenmarkierung (label)} \end{cases}$$

Regionenmarkierung durch *Flood Filling* (1)

```

1: REGIONLABELING(I)
   I: binary image (0 = background, 1 = foreground)

2:   Initialize  $m \leftarrow 2$  (the value of the next label to be assigned).
3:   Iterate over all image coordinates  $\langle u, v \rangle$ .
4:     if  $I(u, v) = 1$  then
5:       FLOODFILL(I, u, v, m)      ▷ use any of the 3 versions below
6:        $m \leftarrow m + 1$ .
7:   return.

```

```

8: FLOODFILL(I, u, v, label)      ▷ Recursive Version
9:   if coordinate  $\langle u, v \rangle$  is within image boundaries and  $I(u, v) = 1$  then
10:    Set  $I(u, v) \leftarrow label$ 
11:    FLOODFILL(I, u+1, v, label)
12:    FLOODFILL(I, u, v+1, label)
13:    FLOODFILL(I, u, v-1, label)
14:    FLOODFILL(I, u-1, v, label)
15:   return.

```

entspricht rekursiver Tiefensuche bei Graphenalgorithmen. Vorsicht:
Erschöpfung des Stack-Speichers bei großen Regionen

Regionenmarkierung durch *Flood Filling* (2)

```
16: FLOODFILL( $I, u, v, label$ ) ▷ Depth-First Version
17:   Create an empty stack  $S$ 
18:   Put the seed coordinate  $\langle u, v \rangle$  onto the stack: PUSH( $S, \langle u, v \rangle$ )
19:   while  $S$  is not empty do
20:     Get the next coordinate from the top of the stack:
        $\langle x, y \rangle \leftarrow$  POP( $S$ )
21:     if coordinate  $(x, y)$  is within image boundaries and  $I(x, y) = 1$ 
       then
22:       Set  $I(x, y) \leftarrow label$ 
23:       PUSH( $S, \langle x+1, y \rangle$ )
24:       PUSH( $S, \langle x, y+1 \rangle$ )
25:       PUSH( $S, \langle x, y-1 \rangle$ )
26:       PUSH( $S, \langle x-1, y \rangle$ )
27:   return.
```

entspricht sequentieller Tiefensuche bei Graphenalgorithmen. Da der Stack im Heapspeicher angelegt wird, besteht prinzipiell keine Größenlimitierung.

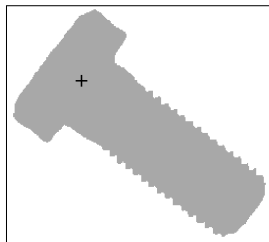
Regionenmarkierung durch *Flood Filling* (3)

```
28: FLOODFILL( $I, u, v, label$ )                                ▷ Breadth-First Version
29:   Create an empty queue  $Q$ 
30:   Insert the seed coordinate  $\langle u, v \rangle$  into the queue: ENQUEUE( $Q, (u, v)$ )
31:   while  $Q$  is not empty do
32:     Get the next coordinate from the front of the queue:
33:      $\langle x, y \rangle \leftarrow$  DEQUEUE( $Q$ )
34:     if coordinate  $\langle x, y \rangle$  is within image boundaries and  $I(x, y) = 1$ 
35:     then
36:       Set  $I(x, y) \leftarrow label$ 
37:       ENQUEUE( $Q, \langle x+1, y \rangle$ )
38:       ENQUEUE( $Q, \langle x, y+1 \rangle$ )
39:       ENQUEUE( $Q, \langle x, y-1 \rangle$ )
40:       ENQUEUE( $Q, \langle x-1, y \rangle$ )
41:   return.
```

entspricht sequentieller Breitensuche bei Graphenalgorithmien.
Ebenfalls keine Größenlimitierung, geringerer Speicherbedarf.

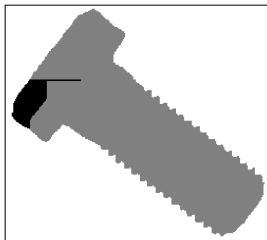
Beispiel: *Flood Filling* (1)

(a)
Original

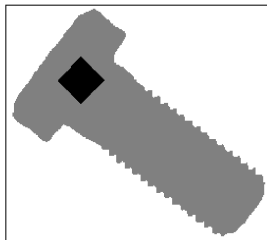


(b)
 $K = 1.000$

depth-first

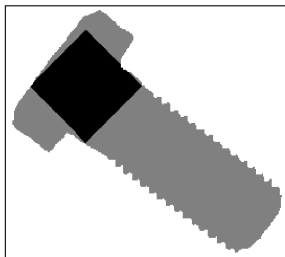
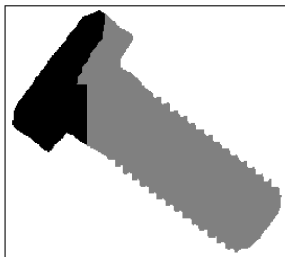


breadth-first

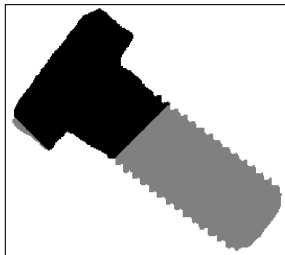
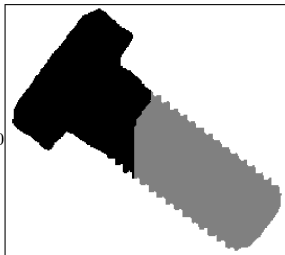


Beispiel: *Flood Filling* (2)

(c)
 $K = 5.000$



(d)
 $K = 10.000$



Sequentielle Regionenmarkierung

Verfahren läuft in 2 Schritten ab:

- 1 **Schritt 1- Vorläufige Markierung:** Beim ersten Durchlauf werden alle Labels aus der linken/oberen Nachbarschaft übernommen.

$$\mathcal{N}_4(u, v) = \begin{array}{|c|c|c|} \hline & N_2 & \\ \hline N_1 & \times & \\ \hline & & \\ \hline \end{array}$$

$$\mathcal{N}_8(u, v) = \begin{array}{|c|c|c|} \hline N_2 & N_3 & N_4 \\ \hline N_1 & \times & \\ \hline & & \\ \hline \end{array}$$

Gleichzeitig werden das Aufeinandertreffen von Regionen mit unterschiedlichen Labels (Kollisionen) gespeichert.

- 2 **Schritt2 - Auflösung der Kollisionen:** Zusammenhängende Regionen werden miteinander verschmolzen.

Häufig eingesetzt wegen moderatem Speicherbedarf, Verschmelzung der Regionen ist aber komplex.

Sequentielle Regionenmarkierung: Schritt 1

```

1: SEQUENTIALLABELING( $I$ )
    $I$ : binary image ( $0 = \text{background}$ ,  $1 = \text{foreground}$ )

2:   PASS 1 – ASSIGN INITIAL LABELS:

3:   Initialize  $m \leftarrow 2$  (the value of the next label to be assigned).
4:   Create an empty set  $\mathcal{C}$  to hold the collisions:  $\mathcal{C} \leftarrow \{\}$ .
5:   for  $v \leftarrow 0 \dots H-1$  do                                ▷  $H = \text{height of image } I$ 
6:     for  $u \leftarrow 0 \dots W-1$  do                            ▷  $W = \text{width of image } I$ 
7:       if  $I(u, v) = 1$  then do one of:
8:         if all neighbors are background pixels (all  $n_i = 0$ ) then
9:            $I(u, v) \leftarrow m$ .
10:           $m \leftarrow m + 1$ .
11:         else if exactly one of the neighbors has a label value
12:            $n_k > 1$  then
13:             set  $I(u, v) \leftarrow n_k$ 
14:         else if several neighbors have label values  $n_j > 1$  then
15:           Select one of them as the new label:
16:              $I(u, v) \leftarrow k \in \{n_j\}$ .
           for all other neighbors with label values  $n_i > 1$  and
            $n_i \neq k$  do
             register the pair  $\langle n_i, k \rangle$  as a label collision:
              $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle n_i, k \rangle\}$ .

```

Remark: The image I now contains label values $0, 2, \dots, m-1$.

Beispiel: Vorläufige Markierung (1)

(a)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 Hintergrund
1 Vordergrund

(b) nur Hintergrundnachbarn

0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	1	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0	
0	0	0	0	1	0	1	0	0	0	0	0	1	0	
0	1	1	1	1	1	1	1	1	1	1	1	1	0	
0	0	0	0	1	1	1	1	1	1	1	1	1	0	
0	1	1	0	0	0	1	0	1	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	

neues Label (2)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	1	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Beispiel: Vorläufige Markierung (2)

(c) genau 1 Nachbar-Label

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	1	0	0	0	1	1	0	1	0	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Nachbar-Label wird übernommen

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	1	1	0	1	0	0	0
0	1	1	1	1	1	1	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(d) 2 Nachbarn tragen versch. Labels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2	2	0	0	3	3	0	4	0	0
0	5	5	5	1	1	1	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

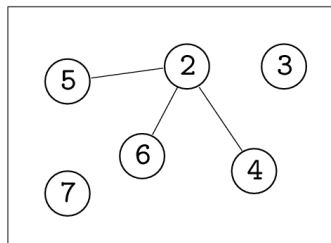
ein Label wird übernommen

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2	2	0	0	3	3	0	4	0	0
0	5	5	5	2	1	1	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Beispiel: Vorläufige Markierung (3)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	3	3	0	4	0
0	5	5	5	2	2	2	0	0	3	0	0	4	0
0	0	0	0	2	0	2	0	0	0	0	0	4	0
0	6	6	2	2	2	2	2	2	2	2	2	2	0
0	0	0	0	2	2	2	2	2	2	2	2	2	0
0	7	7	0	0	0	2	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a)



(b)

Sequentielle Regionenmarkierung: Schritt 2

- 17: PASS 2 – RESOLVE LABEL COLLISIONS:
- 18: Let $\mathcal{L} = \{2, 3, \dots, m-1\}$ be the set of preliminary region labels.
- 19: Create a partitioning of \mathcal{L} as a *vector of sets*, one set for each label value: $\mathcal{R} \leftarrow [\mathcal{R}_2, \mathcal{R}_3, \dots, \mathcal{R}_{m-1}] = [\{2\}, \{3\}, \{4\}, \dots, \{m-1\}]$, so $\mathcal{R}_i = \{i\}$ for all $i \in \mathcal{L}$.
- 20: **for all** collisions $\langle a, b \rangle \in \mathcal{C}$ **do**
- 21: Find in \mathcal{R} the sets $\mathcal{R}_a, \mathcal{R}_b$ containing the labels a, b , resp.:
 $\mathcal{R}_a \leftarrow$ the set which currently contains label a
 $\mathcal{R}_b \leftarrow$ the set which currently contains label b
- 22: **if** $\mathcal{R}_a \neq \mathcal{R}_b$ (a and b are contained in different sets) **then**
- 23: Merge sets \mathcal{R}_a and \mathcal{R}_b by moving all elements of \mathcal{R}_b to \mathcal{R}_a :
 $\mathcal{R}_a \leftarrow \mathcal{R}_a \cup \mathcal{R}_b$
 $\mathcal{R}_b \leftarrow \{\}$

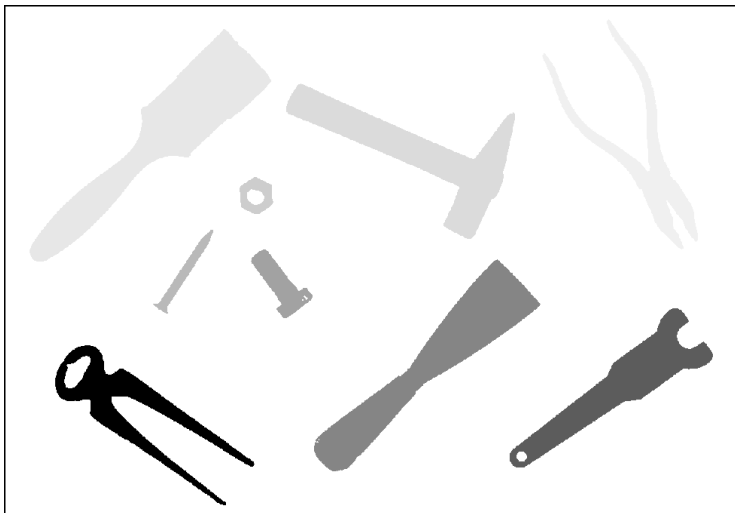
Remark: All equivalent label values (i.e., all labels of pixels in the same region) are now contained in the same sets within \mathcal{R} .

Sequentielle Regionenmarkierung: Schritt 2

- 24: PASS 3: RELABEL THE IMAGE:
- 25: Iterate through all image pixels (u, v) :
- 26: **if** $I(u, v) > 1$ **then**
- 27: Find the set \mathcal{R}_i in \mathcal{R} which contains label $I(u, v)$.
- 28: Choose one unique, representative element k from the set \mathcal{R}_i
 (e.g., the minimum value, $k \leftarrow \min(\mathcal{S})$).
- 29: Replace the image label: $I(u, v) \leftarrow k$.
- 30: **return** the labeled image I .

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	3	3	0	2	0
0	2	2	2	2	2	2	0	0	3	0	0	2	0
0	0	0	0	2	0	2	0	0	0	0	0	2	0
0	2	2	2	2	2	2	2	2	2	2	2	2	0
0	0	0	0	2	2	2	2	2	2	2	2	2	0
0	7	7	0	0	0	2	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

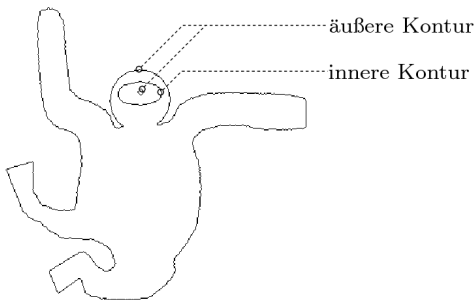
Beispiel: fertige Regionenmarkierung



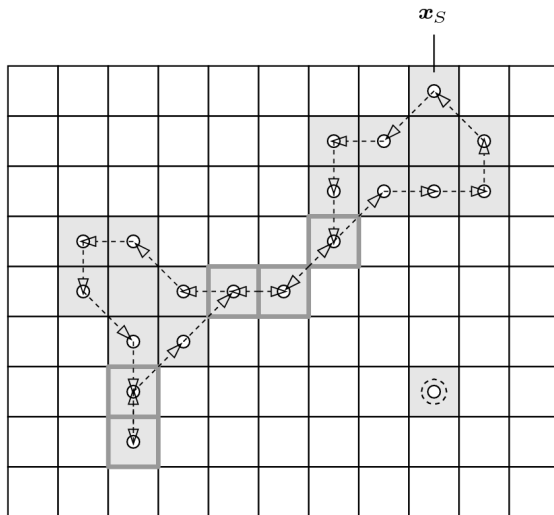
Übersicht

- 1 Auffinden von Bildregionen
- 2 Konturen von Regionen**
- 3 Repräsentation von Bildregionen
- 4 Eigenschaften binärer Bildregionen

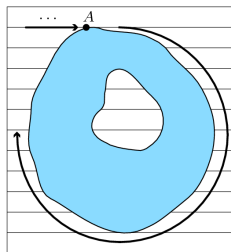
Äußere und innere Konturen



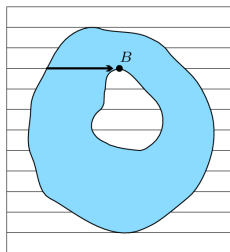
Repräsentation einer Kontur durch eine geordnete Folge von Pixelkoordinaten



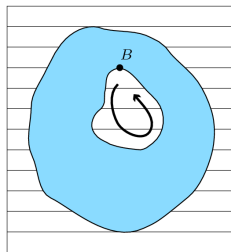
Kombinierte Regionenmarkierung und Konturfindung



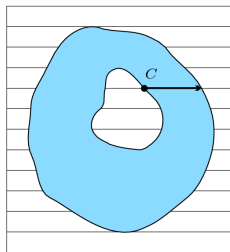
(a)



(b)



(c)



(d)

Algorithmus (1)

```

2:   Create an empty set of contours:  $\mathcal{C} \leftarrow \{\}$ 
3:   Create a label map  $LM$  of the same size as  $I$  and initialize:
4:   for all  $(u, v)$  do
5:      $LM(u, v) \leftarrow 0$  ▷ label map  $LM$ 
6:    $R \leftarrow 0$  ▷ region counter  $R$ 

7:   Scan the image from left to right, top to bottom:
8:   for  $v \leftarrow 0 \dots N-1$  do
9:      $L_c \leftarrow 0$  ▷ current label  $L_c$ 
10:    for  $u \leftarrow 0 \dots M-1$  do
11:      if  $I(u, v)$  is a foreground pixel then
12:        if  $(L_c \neq 0)$  then ▷ continue existing region
13:           $LM(u, v) \leftarrow L_c$ 
14:        else
15:           $L_c \leftarrow LM(u, v)$ 
16:          if  $(L_c = 0)$  then ▷ hit new outer contour
17:             $R \leftarrow R + 1$ 
18:             $L_c \leftarrow R$ 
19:             $\mathbf{x}_S \leftarrow (u, v)$ 
20:             $C_{\text{outer}} \leftarrow \text{TRACECONTOUR}(\mathbf{x}_S, 0, L_c, I, LM)$ 
21:             $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_{\text{outer}}\}$  ▷ collect new contour
22:             $LM(u, v) \leftarrow L_c$ 
23:          else ▷  $I(u, v)$  is a background pixel
24:            if  $(L \neq 0)$  then
25:              if  $(LM(u, v) = 0)$  then ▷ hit new inner contour
26:                 $\mathbf{x}_S \leftarrow (u-1, v)$ 
27:                 $C_{\text{inner}} \leftarrow \text{TRACECONTOUR}(\mathbf{x}_S, 1, L_c, I, LM)$ 
28:                 $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_{\text{inner}}\}$  ▷ collect new contour
29:               $L \leftarrow 0$ 
30:   return  $(\mathcal{C}, LM)$ . ▷ return the set of contours and the label map

```

Algorithmus (2)

```

1: TRACECONTOUR( $\mathbf{x}_S, d_S, L_C, I, LM$ )
    $\mathbf{x}_S$ : 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:  $(\mathbf{x}_T, d) \leftarrow \text{FINDNEXTNODE}(\mathbf{x}_S, d_S, I, LM)$ 
4: APPEND( $C, \mathbf{x}_T$ )                                ▷ add  $\mathbf{x}_T$  to contour  $C$ 
5:  $\mathbf{x}_p \leftarrow \mathbf{x}_S$                             ▷ previous position  $\mathbf{x}_p = (u_p, v_p)$ 
6:  $\mathbf{x}_c \leftarrow \mathbf{x}_T$                             ▷ current position  $\mathbf{x}_c = (u_c, v_c)$ 
7:  $done \leftarrow (\mathbf{x}_S = \mathbf{x}_T)$                     ▷ isolated pixel?
8: while ( $\neg done$ ) do
9:    $LM(u_c, v_c) \leftarrow L_C$ 
10:   $(\mathbf{x}_n, d) \leftarrow \text{FINDNEXTNODE}(\mathbf{x}_c, (d+6) \bmod 8, I, LM)$ 
11:   $\mathbf{x}_p \leftarrow \mathbf{x}_c$ 
12:   $\mathbf{x}_c \leftarrow \mathbf{x}_n$ 
13:   $done \leftarrow (\mathbf{x}_p = \mathbf{x}_S \wedge \mathbf{x}_c = \mathbf{x}_T)$     ▷ back at starting position?
14:  if ( $\neg done$ ) then
15:    APPEND( $C, \mathbf{x}_n$ )                                ▷ add point  $\mathbf{x}_n$  to contour  $C$ 
16: return  $C$  .                                     ▷ return this contour

```


Algorithmus (3)

```

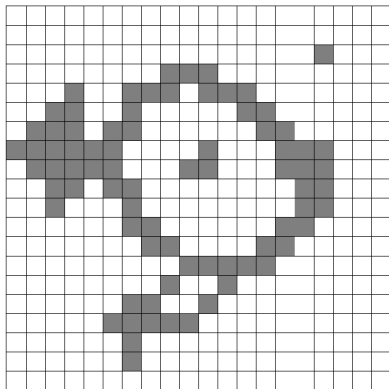
17: FINDNEXTNODE( $\mathbf{x}_c, d, I, LM$ )
     $\mathbf{x}_c$ : original position,  $d$ : search direction,  $I$ : image,  $LM$ : label map
18:   for  $i \leftarrow 0 \dots 6$  do                                ▷ search in 7 directions
19:      $\mathbf{x}' \leftarrow \mathbf{x}_c + \text{DELTA}(d)$                     ▷  $\mathbf{x}' = (u', v')$ 
20:     if  $I(u', v')$  is a background pixel then
21:        $LM(u', v') \leftarrow -1$                             ▷ mark background as visited (-1)
22:        $d \leftarrow (d + 1) \bmod 8$ 
23:     else                                                    ▷ found a non-background pixel at  $\mathbf{x}'$ 
24:       return  $(\mathbf{x}', d)$ 
25:   return  $(\mathbf{x}_c, d)$  .                                     ▷ found no next node, return start position

```

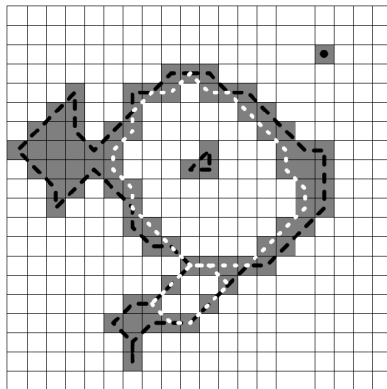
26: $\text{DELTA}(d) = (\Delta x, \Delta y)$, wobei

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

Beispiel Konturverfolgung (1)



(a)



(b)

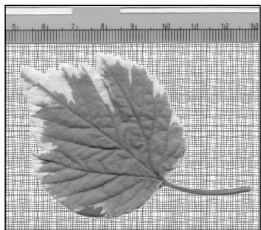
Beispiel Konturverfolgung (2)



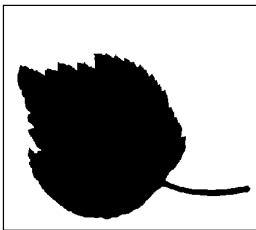
Übersicht

- 1 Auffinden von Bildregionen
- 2 Konturen von Regionen
- 3 Repräsentation von Bildregionen**
- 4 Eigenschaften binärer Bildregionen

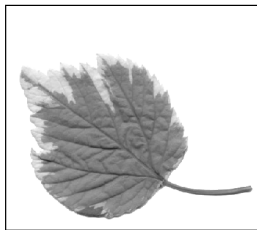
Matrixrepräsentation und Lauflängencodierung



(a)



(b)



(c)

Bitmap

	0	1	2	3	4	5	6	7	8
0									
1			x	x	x	x	x	x	
2									
3					x	x	x	x	
4		x	x	x		x	x	x	
5	x	x	x	x	x	x	x	x	x
6									

RLE

$\langle row, column, length \rangle$

→

$\langle 1, 2, 6 \rangle$

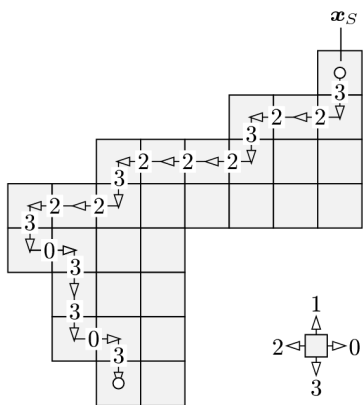
$\langle 3, 4, 4 \rangle$

$\langle 4, 1, 3 \rangle$

$\langle 4, 5, 3 \rangle$

$\langle 5, 0, 9 \rangle$

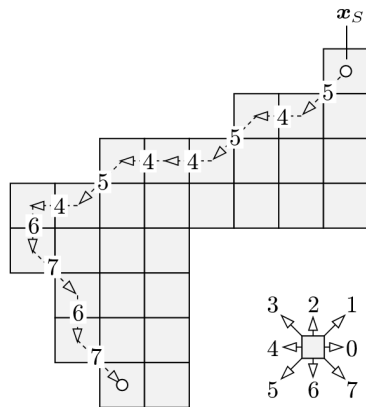
Chain Codes



4-Chain Code

3223222322303303...111

length = 28



8-Chain Code

54544546767...222

length = $18 + 5\sqrt{2} \approx 25$

Differentielle Chain Codes

Problem: Vergleich zweier durch *Chain Codes* dargestellter Regionen ist schwierig, da

- abhängig vom Startpunkt
- Drehung um 90° führt zu völlig anderem *Chain Code*.

Umwandlung *absoluter* Chain Code $C = (c_0, c_1, \dots, c_{M-1})$ in *differentiellen* Chain Code $C' = (c'_0, c'_1, \dots, c'_{M-1})$:

$$c'_i = \begin{cases} (c_{i+1} - c_i) \bmod 8 & \text{für } 0 \leq i < M-1 \\ (c_0 - c_i) \bmod 8 & \text{für } i = M-1 \end{cases}$$

Voriges Beispiel:

$$C_{\mathcal{R}} = (5, 4, 5, 4, 4, 5, 4, 6, 7, 6, 7, \dots, 2, 2, 2)$$

$$C'_{\mathcal{R}} = (7, 1, 7, 0, 1, 7, 2, 1, 7, 1, 1, \dots, 0, 0, 3)$$

Shape Numbers

Problem: Differentielle Chain Codes bleiben zwar bei Drehungen um 90° unverändert, aber sie hängen immer noch von Startpunkt ab.

Idee: Repräsentation als Zahlen zur Basis 4 oder 8 (**Shape Numbers**). Zyklisch vertauschen, z.B.

$$C' = (0, 1, 3, 2, \dots, 9, 3, 7, 4) \quad C' \rightarrow 2 = (7, 4, 0, 1, \dots, 0, 3)$$

und maximale Ähnlichkeit suchen. Aber:

- Drehungen um beliebige Winkel
- Skalierungen
- Verzerrungen
- Veränderung an Teilen

führen zu starken Abweichungen in dieser Repräsentation.

Übersicht

- 1 Auffinden von Bildregionen
- 2 Konturen von Regionen
- 3 Repräsentation von Bildregionen
- 4 Eigenschaften binärer Bildregionen**

Eigenschaften binärer Bildregionen

Nächstes Mal....