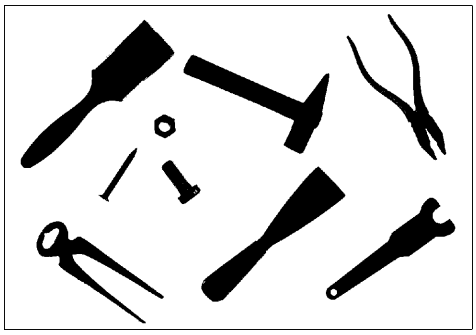
VC3 – Image Processing

Regionen in Binärbildern

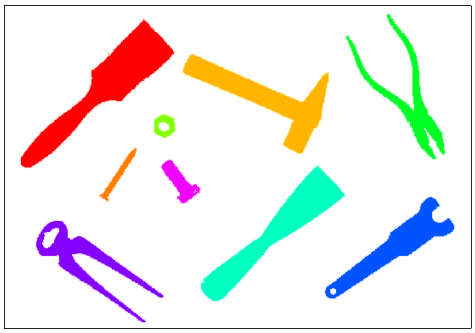
Prof. Dr. Klaus Jung

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Auffinden von Bildregionen

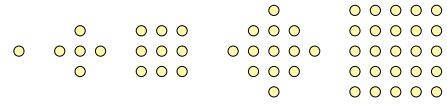


Ziel: Objekte identifizieren



Definition von Zusammenhang über Nachbarschaft

$$N_r(i,j) = \left\{ (u,v) \mid (u-i)^2 + (v-j)^2 \le r^2 \right\}$$



$r^2=0$ $r^2=1$ $r^2=2$ $r^2=4$ $r^2=8$

Vierer-nachbar-schaft **Achter-nachbar-schaft**

Beispiel Nachbarschaft

0 Hintergrund 1 Vordergrund

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

4er Nachbarschaft

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

8er Nachbarschaft

Markierung durch Flood Filling

$$I(u,v) = \begin{cases} 0 & \text{Hintergrund} \\ 1 & \text{Vordergrund} \\ 2,3,\ldots & \text{Regionenmarkierung} \end{cases}$$

1: RegionLabeling(I)
   I: binary image (0 = *background*, 1 = *foreground*)
   The image I is labeled (destructively modified) and returned.
2: Initialize $m \leftarrow 2$ (the value of the next label to be assigned).
3: Iterate over all image coordinates (u, v).
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** the labeled image I.

1. Rekursives Flood Filling

```
8:  FloodFill(I, u, v, label)                          ▷ Recursive Version
9:      if coordinate (u, v) is within image boundaries and I(u, v) = 1 then
10:         Set I(u, v) ← 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.
```

4er Nachbarschaft

- Vorteil: Einfach zu implementieren
- Nachteil: Stack-Speicher schnell erschöpft
  - Nur für kleine Bilder geeignet

2. Depth-First Flood Filling

```
16: FloodFill(I, u, v, label)                          ▷ Depth-First Version
17:     Create an empty stack S
18:     Put the seed coordinate ⟨u, v⟩ onto the stack: Push(S, ⟨u, v⟩)
19:     while S is not empty do
20:         Get the next coordinate from the top of the stack:
                ⟨x, y⟩ ← Pop(S)
21:         if coordinate (x, y) is within image boundaries and I(x, y) = 1
                then
22:             Set I(x, y) ← label
23:             Push(S, ⟨x+1, y⟩)
24:             Push(S, ⟨x, y+1⟩)
25:             Push(S, ⟨x, y−1⟩)
26:             Push(S, ⟨x−1, y⟩)
27:     return.
```


2. Depth-First Flood Filling

- Verwendung einer Datenstruktur
  - Stack von Koordinatenpunkten *(u,v)*
    - Z.B. Java Klasse **`Stack`**
  - Dynamische Speicherallozierung vom Heap
- Vorteil:
  - Keine Probleme mit dem Stack-Speicher
- Nachteil:
  - Stack kann lang werden, wenn die Objekte sehr groß sind

3. Breadth-First Flood Filling

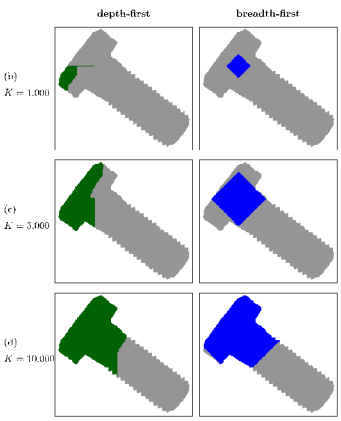
```
28: FloodFill(I, u, v, label)                          ▷ Breadth-First Version
29:     Create an empty queue Q
30:     Insert the seed coordinate ⟨u, v⟩ into the queue: Enqueue(Q, ⟨u, v⟩)
31:     while Q is not empty do
32:         Get the next coordinate from the front of the queue:
                ⟨x, y⟩ ← Dequeue(Q)
33:         if coordinate ⟨x, y⟩ is within image boundaries and I(x, y) = 1
                then
34:             Set I(x, y) ← label
35:             Enqueue(Q, ⟨x+1, y⟩)
36:             Enqueue(Q, ⟨x, y+1⟩)
37:             Enqueue(Q, ⟨x, y−1⟩)
38:             Enqueue(Q, ⟨x−1, y⟩)
39:     return.
```


3. Breadth-First Flood Filling

- Verwendung einer Datenstruktur
  - Queue von Koordinatenpunkten *(u,v)*
    - Z.B. mit Java Klasse **`java.util.LinkedList`**
    - DEQUEUE mit **`removeLast()`**
    - ENQUEUE mit **`addFirst()`**
  - Dynamische Speicherallozierung vom Heap
- Vorteile:
  - Keine Probleme mit dem Stack-Speicher
  - Geringerer Speicherbedarf als bei Depth-First

Vergleich

Original mit Startpunkt

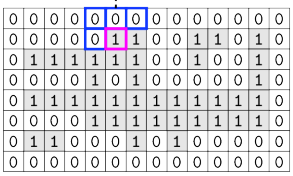


Sequentielle Regionenmarkierung

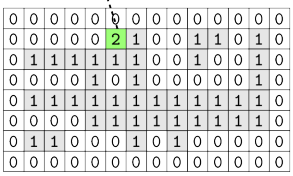
- Auch als „region labeling“ bezeichnet
- „Gleichzeitiges“ Markieren aller Regionen
- Zwei Schritte
  1. Vorläufiges Markieren
  2. Auflösung von Kollisionen

Vorläufiges Markieren (1/3)

(b) only background neighbors

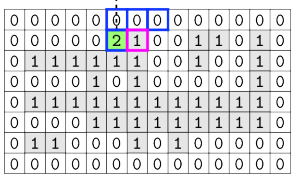


new label (**2**)

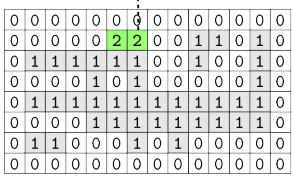


Vorläufiges Markieren (2/3)

(c) exactly one neighbor label

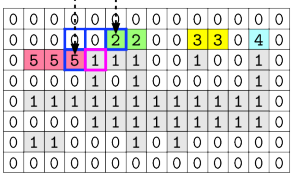


neighbor label is propagated

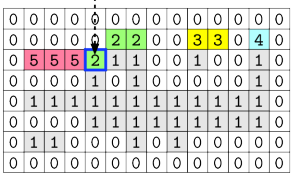


Vorläufiges Markieren (3/3)

(d) two different neighbor labels



one of the labels (**2**) is propagated



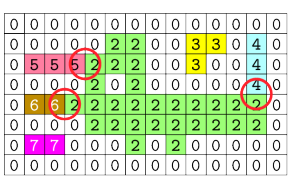
Kollisionen vermerken: Hier (2, 5)

Algorithmus

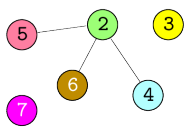
1: SequentialLabeling(I)
   I: binary image (0 = *background*, 1 = *foreground*)
2: Pass 1 – Assign Initial Labels:
3: Initialize $m \leftarrow 2$ (the value of the next label to be assigned).
4: Create an empty set $\mathcal{C}$ to hold the collisions: $\mathcal{C} \leftarrow \{\}$.
5: **for** $v \leftarrow 0 \ldots H\text{–}1$ **do** ▷ H = height of image I
6: **for** $u \leftarrow 0 \ldots W\text{–}1$ **do** ▷ W = 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 $n_k > 1$ **then**
12: set $I(u,v) \leftarrow n_k$
13: **else if** *several* neighbors have label values $n_j > 1$ **then**
14: Select one of them as the new label:
$I(u,v) \leftarrow k \in \{n_j\}$.
15: **for** all other neighbors with label values $n_i > 1$ and $n_i \neq k$ **do**
16: 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 $0, 2, \ldots m\text{–}1$.

Verbleibende Kollisionen



(a)



(b)

Kollisionen Auflösen

17: PASS 2 – RESOLVE LABEL COLLISIONS:
18: Let $\mathcal{L} = \{2, 3, \ldots m{-}1\}$ 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: $\mathcal{R} \leftarrow [\mathcal{R}_2, \mathcal{R}_3, \ldots, \mathcal{R}_{m-1}] = [\{2\}, \{3\}, \{4\}, \ldots, \{m{-}1\}]$, 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 $\mathcal{R}$.

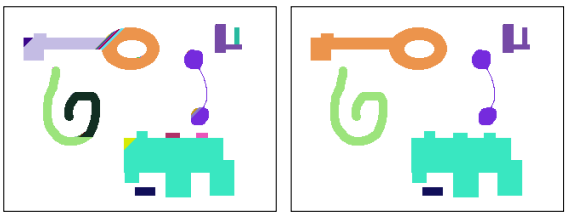
19 © Klaus Jung

Bild neu Indizieren

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.

20 © Klaus Jung

Beispiel



einzelne Regionen

verbundene Regionen

21 © Klaus Jung