Bildregionen und Konturen

12. Februar 2006

Seminar Bildsegmentierung und Computer Vision WS05/06

Heike Zierau
23.01.2006

Inhaltsverzeichnis

1 Auffinden von Bildregionen

1.1 Binärbilder

Binärbilder $I : \mathbb{N}^2 \rightarrow \{0, 1\}$ sind Bilder, deren Pixel nur zwei verschiedene Werte annehmen können, d.h. wir unterscheiden nur zwischen Vordergrund und Hintergrund.

$$I(x,y) = \left\{ \begin{array}{ll} 0, & Hintergrund \\ 1, & Vordergrund \end{array} \right\}$$

Eine *verbundene binäre Bildregion* ist eine Gruppe aneinandergrenzender Vordergrundpixel, wobei die Pixel durch die *Vierernachbarschaft* oder durch die *Achternachbarschaft* miteinander verbunden sind.

- **Vierernachbarschaft**:

$$N(x,y) = \begin{array}{|c|c|c|} \hline & N_2 & \\ \hline N_1 & (\mathrm{x,y}) & N_3 \\ \hline & N_4 & \\ \hline \end{array}$$

- **Achternachbarschaft**:

$$N(x,y) = \begin{array}{|c|c|c|} \hline N_2 & N_3 & N_4 \\ \hline N_1 & (\mathrm{x,y}) & N_5 \\ \hline N_8 & N_7 & N_6 \\ \hline \end{array}$$

Um die Bildregionen in Binärbildern finden zu können, müssen wir wissen, welche Pixel zu welcher Region gehören, wie viele Regionen es im Bild gibt und wo sich diese befinden. Dies führt uns zum Begriff der *Regionenmarkierung*, die normalerweise in einem Schritt die genannten Aufgaben durchführt. Allgemein werden dabei schrittweise zueinander benachbarte Pixel zu Regionen zusammengefügt und alle Pixel innerhalb einer Region erhalten eine eindeutige Identifikationsnummer, auch *„Label"* genannt.

Im Folgenden werden zwei Verfahren vorgestellt, Flood Filling und die sequentielle Regionenmarkierung. Wir nehmen für beide Verfahren an, dass das binäre Ausgangsbild nur aus den Werten 0 (Hintergrund) und 1 (Vordergrund) besteht und alle natürlichen Zahlen ≥ 2 mögliche Kandidaten für ein Label sind:

$$I\,(x,y) = \left\{ \begin{array}{ll} 0, & Hintergrund \\ 1, & Vordergrund \\ 2,3,..., & Labels \end{array} \right\}$$

1.2 Flood Filling

Die Idee des Flood Fillings ist, dass man eine einzelne Bildregion von einem gegebenen Startpunkt aus in alle Richtungen ausfüllt, ähnlich der Ausbreitung einer Flutwelle. Das heißt, man sucht ein noch unmarkiertes Vordergrundpixel, also $I\,(x,y) = 1$, weist diesem ein neues Label zu und besucht anschließend alle nach der Vierer- oder Achternachbarschaft angrenzenden Pixel der Region, die dabei das gleiche Label wie das erste Pixel erhalten.

Algorithm 1 Flood Filling

$FloodFill(I, x, y, label)$
IF(x, y) innerhalb der Bildgrenzen AND $I(x, y) = 1$ THEN
    $I(x, y) := label$
    $FloodFill(I, x + 1, y, label)$
    $FloodFill(I, x, y - 1, label)$
    $FloodFill(I, x, y + 1, label)$
    $FloddFill(I, x - 1, y, label)$

Es gibt verschiedene Möglichkeiten diesen Algorthmus zu implementieren, eine intuitive wäre eine Rekursive, jedoch lassen sich analog dazu ebenso nicht-rekursive Implementierungen realisieren.

Die hier angeführte rekursive Implementierung des Flood-Filling-Algorithmus, vgl. Algorithmus 1 , beruht auf der Vierernachbarschaft.

Dieser Algorithmus ist ein sehr einfacher Ansatz, man braucht keine besondere Datenstruktur zur Verwaltung der bereits besuchten Pixel, bzw. der Nachbarpixel. Das Problem ist nur, dass durch die entstehende Baumstruktur mit dem Startpunkt als Wurzel die Rekursionstiefe proportional zur Größe der Region zunimmt, so dass der Speicher auf dem Stack sehr bald erschöpft ist. Daher können Bilder mit mehr als 200x200 Pixel nicht mehr bearbeitet werden.

Iterative Verfahren können durch einen eigenen Stack für jede Prozedur ganz analog zum rekursiven implementiert werden, wobei der Stack die noch unmarkierten Pixel verwaltet.

Eine weitere iterative Variante wäre anstatt eines Stacks die noch unmarkierten Pixel mithilfe einer Queue zu verwalten.

Jedoch möchte ich hier nicht weiter auf iterative Implementierungen von Flood Filling eingehen, sondern ein anderes iteratives Verfahren vorstellen, die sequentielle Regionenmarkierung.

1.3 Sequentielle Regionenmarkierung

Die sequentielle Regionenmarkierung ist eine nichtrekursive Technik, auch bekannt als *„region labeling"*, bei der die Bearbeitung des Bildes in zwei Schritte unterteilt ist. Zuerst wird das Bild sequentiell zeilenweise durchlaufen und jedes Vordergrundpixel vorläufig markiert. Im zweiten Schritt werden die eventuell kollidierenden Markierungen innerhalb einer Region aufgehoben und die zusammengehörenden Teilregionen durch die Markierung mit einem gemeinsamen Label miteinander verbunden.

(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

Abbildung 1: Binärbild mit drei Regionen

1.3.1 Beispiel

Schritt 1:
Wir durchlaufen das Binärbild, das in Abbildung 1 gegeben ist, zeilenweise, bis wir zum ersten unmarkierten Vordergrundpixel gelangen (vgl. Bild b) Abbildung 2). Das hat nur Hintergrundnachbarn, da wir nur die Nachbarn betrachten können, die wir schon besucht haben, also die oben und links. Wir erzeugen ein neues Label = 2 und das Pixel erhält den Wert 2. Das nächste Pixel hat genau einen markierten Vordergrundnachbarn. Wir übernehmen das Label des Vordergrundnachbarn. Wir führen das Verfahren in diesem Sinne weiter, bis wir auf das erste Pixel stoßen, das zwei Nachbarn mit verschiedenen Labels hat. Wir müssen uns für ein Label entscheiden.

Schritt 2:
Haben wir das gesamte Bild so markiert, treten drei Kollisionen im mittleren Objekt auf, die wir beheben müssen, vgl. Abbildung 3. Anhand des Graphen erkennt man auch den Zusammenhang der mittleren Region.

Wie in Abbildung 4 soll das Bild am Ende aussehen. Die Kollosionen wurden aufgelöst und das kleinste gemeinsame Label für die gesamte Region gewählt.

1.3.2 Union-Find

Eine effiziente Möglichkeit, die Kollisionen zu beheben, ist das Prinzip von *Union-Find.*

Abbildung 2: Beispiel Binärbild mit sequentieller Regionenmarkierung bearbeitet

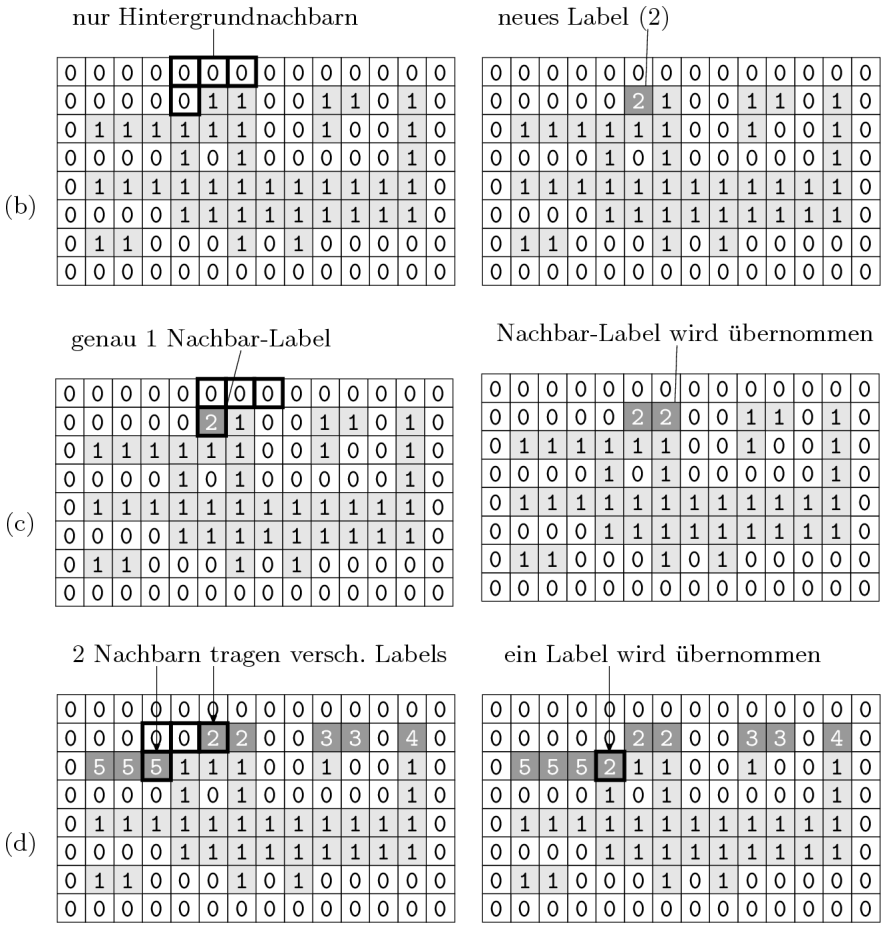
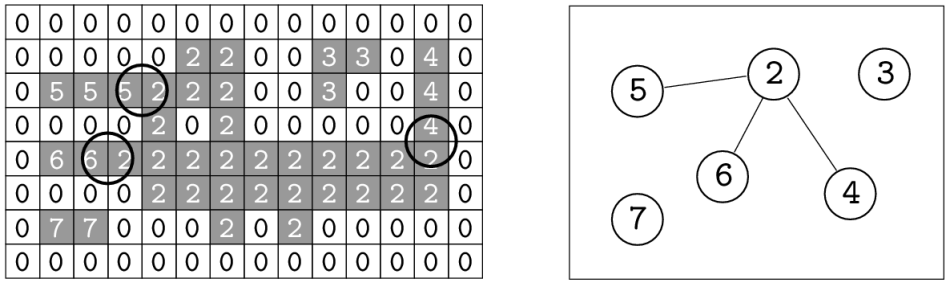


Abbildung 3: Sequentielle Regionenmarkierung - nach vorläufiger Markierung



(a) (b)

Abbildung 4: Sequentielle Regionenmarkierung - Ergebnis nach Schritt 2

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

Algorithm 2 Union-Find - Init

```
PROCEDURE Init
  FOR i := 0TO n − 1 DO a[i] := i
```

Algorithm 3 Union-Find - Find

```
PROCEDURE Find(i)
  IF i = a[i] THEN RETURN i
  ELSE
    a[i] := Find(a[i])
    RETURN a[i]
```

Es wird eine Menge von Daten verwaltet, in unserem Fall die Menge der Bildpunkte $\{v_1, ..., v_n\}$ mit $v_i = (x_j, y_k), j\epsilon\{1, ..., m\}, k\epsilon\{1, ...s\}$eines Bildes der Größe $m \cdot s$ mit $n = m \cdot s$.

Zunächst versteht man jeden Bildpunkt v_i als einelementige Menge, d.h. jedes Element v_i zeigt auf sich selbst. Durch die Operation Union werden die Mengen v_i so vereinigt, dass ein System von Teilmengen von $\{v_1, ..., v_n\}$ entsteht.

Die Funktionen Union und Find sind dabei wie folgt definiert:

- $Find(v_i)$: liefert das kanonisches Element der Menge zurück, in der sich der Punkt v_i befindet.
- $Union(v_i, v_j)$: vereinigt die beiden Mengen, die v_i und v_j enthalten.

Auf einer $n-$elementigen Grundmenge lassen sich mit Hilfe eines Arrays $a[0...n-1]$ die Operationen *Union* und *Find* sehr einfach implementieren. Das Array a entspricht dabei der Menge der Bildpunkte $\{v_1, ..., v_n\}$, wobei die Zeilen des Bildes zu einem einzigen Array aneinandergefügt werden.

Man initialisiert $a[i] = i$, d.h. i ist das kanonische Element der Menge $\{i\}$, also jedes Element des Arrays a verweist auf sich selbst, vgl. Algorithmus 2.

Man findet das kanonische Element der Menge, die das Element i enthält, durch u.U. mehrfaches sondieren von $a[i], a[a[i]], ...$bis für ein k gilt $a[k] = k$, d.h. das kanonische Element bildet immer die Baumwurzel, vgl. Algorithmus 3.

Durch den Befehl $a[i] = Find(a[i])$ wird die Verzweigungsstruktur des Baumes so umgebaut, dass sich die Wege bis zur Wurzel verkürzen, denn in alle angesprochenen $a[i]$-Werte wird ein direkter Verweis auf die Wurzel eingetragen.

Algorithm 4 Union-Find - Union

PROCEDURE *Union(i,j)*
  $c1 = Find(i)$
  $c2 = Find(j)$
  RANDOM z IN $\{0, 1\}$
  IF $z = 0$ THEN $a[c1] := c2$
  ELSE $a[c2] := c1$

Bei der Union-Operation, vgl. Algorithmus 4, entscheidet der Zufall, ob die Menge i an die Menge j gekoppelt wird oder umgekehrt.

Statt den Zufall entscheiden zu lassen, kann man auch die Anzahl der Elemente einer Menge speichern und bei der Union-Operation die kleinere an die größere Menge hängen.

Die Komplexität einer beliebigen Folge von n Union- und Find-Operationen lässt sich durch eine Amortisationsanalyse mit $O(nlog^*n)$ abschätzen, wobei

$$log^*n = min\{k | \underbrace{log...log}_{k-mal} n \leq 1\}.$$

log^*n wächst sehr sehr langsam: für $n \leq 10^{19728}$ ist $log^*n \leq 5$, d.h. die amortisierte Komplexität ist so gut wie konstant.

1.3.3 Beispiel

Das Bild in Abbildung 5 a) wird als Array dargestellt, indem die Zeilen des Bildes zu einem Array aneinandergefügt werden. Die Elemente des mit $a[i] = i$ initialisierten Arrays gehören dabei zu den entsprechenden darunterliegenden Arrayelementen des Bildarrays. Durch die Pfeile werden die Bildpunkte des Arrays mit den entsprechenden im Bild gekennzeichnet.

Das Bild wird wie bei der sequentiellen Regionenmarkierung zeilenweise durchlaufen, bis der erste Pixelwert 1 ist. Wir kennzeichnen vorläufig das bereits besuchte Pixel mit rot (Bild b)). Das 6. Arrayelement wird durch den Verweis auf sich selbst mit Hilfe des Pfeiles gekennzeichnet, damit wir wissen, es ist ein bereits besuchter Vordergrundpixel. Das nächste noch nicht besuchte Vordergrundpixel hat ebenfalls nur Hintergrundnachbarn (Bild c)). Wir kennzeichnen es vorläufig mit der neuen Farbe grün und das 8. Arrayelement verweist ebenfalls auf sich selbst, mit Hilfe des Pfeiles gekennzeichnet. In Bild d) haben wir genau einen roten Vordergrundnachbarn. Wir verweisen

Abbildung 5: Union-Find - bis zur Kollision

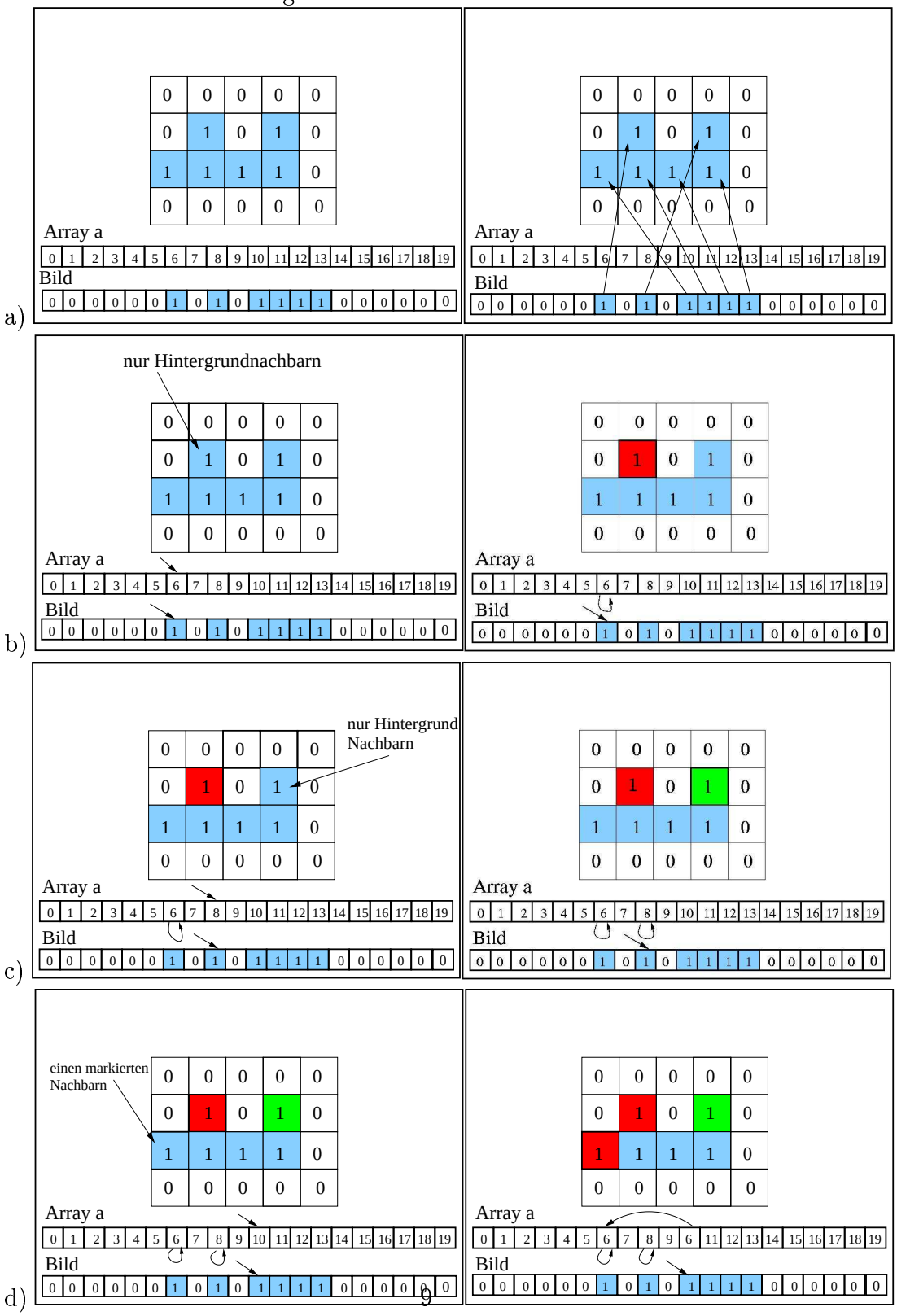
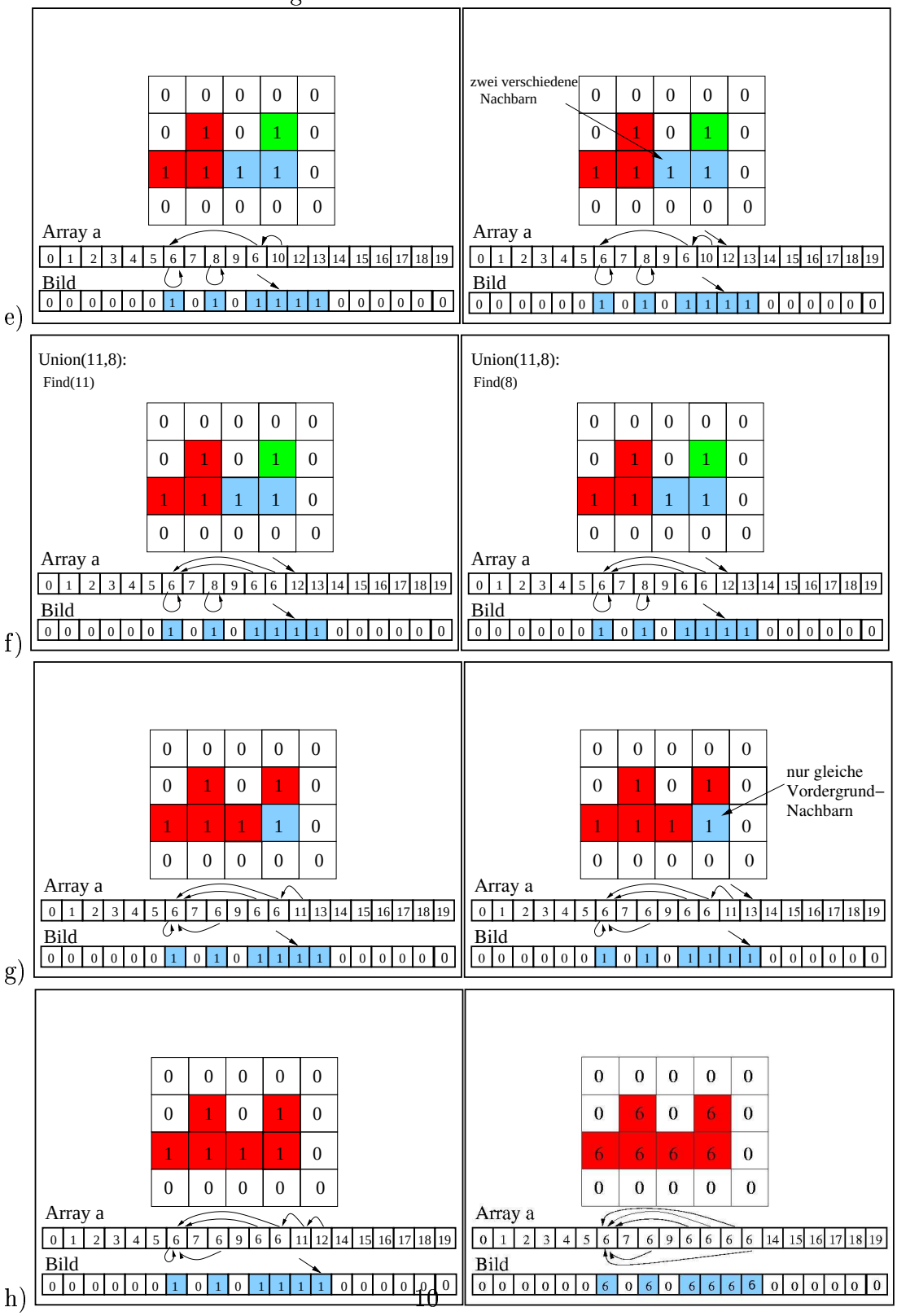


Abbildung 6: Union-Find - Kollosion lösen



vom 11. Arrayelement auf das 6. da die beiden Elemente offensichtlich zu einer Region gehören. Beim nächsten Vordergrundpixel verweisen wir vom 12. Arrayelement auf das 11., da das jetzt betrachtete Pixel zu der roten Region gehört.

In e) tritt wieder die Situation zweier verschiedener Nachbarn auf. Nun brauchen wir die Operationen Union und Find. Nach Ausführung der Find-Operation verweisen alle angesprochenen Elemente der Menge 6 auf das kanonische Element, vgl. Abbildung 6, Bild f). Durch Find(8) verändert sich nichts, da es nur ein Element der grünen Menge gibt, d.h. 8 ist bereits das kanonische Element. Durch $Union(11, 8)$ werden die beiden Mengen vereint, d.h. das 8. Element verweist nun auch auf das 6 (Bild g)). Das betrachtete 12. Element verweist wie gehabt auf das 11. Das letzte Vordergrundpixel hat wieder nur Vordergrundnachbarn gleicher Farbe. Es bekommt einen Verweis auf das vorhergehende und die rote Farbe.

Durch eine letzte Find-Operation, Bild h), verweist nun jedes Element des Objektes auf das kanonische Element. Wir können nun das Bildarray durchlaufen und bei jeder 1 den Wert des Arrays a an der jeweiligen Stelle eintragen.

1.4 Kombiniertes Verfahren

Bisher haben wir zwei Verfahren kennengelernt die Regionen in Binärbildern erkennen.

Eine weitere effiziente Möglichkeit Regionen in Binärbildern zu finden ist ein kombiniertes Verfahren aus sequentieller Regionenmarkierung und traditioneller Konturverfolgung.

Man unterscheidet bei Konturen zwischen äußeren und inneren Konturen, vlg. Abbildung 7. Jede Bildregion hat nur eine äußere Kontur, kann jedoch beliebig viele Löcher enthalten, deren Ränder innere Konturen sind. Dabei können Konturen zum Teil die Breite von nur einem Pixel haben, wenn äußere und innere Konturen sich quasi überschneiden.

Der Vorteil des kombinierten Verfahrens ist, dass in nur einem Bilddurchlauf sowohl äußere, als auch innere Konturen identifiziert werden, die Datenstruktur nicht komplex und das Verfahren im Vergleich zum rekursiven Flood Filling sehr effizient ist.

Der folgende Algorithmus ist leicht verständlich, jedoch ist die Implementierung unter Umständen recht aufwendig.

Das Bild wird wie bei der sequentiellen Regionenmarkierung sequentiell zeilenweise durchlaufen, um sicherzustellen, dass alle Pixel besucht und markiert wurden.

Abbildung 7: Binärbild mit äußeren und inneren Konturen

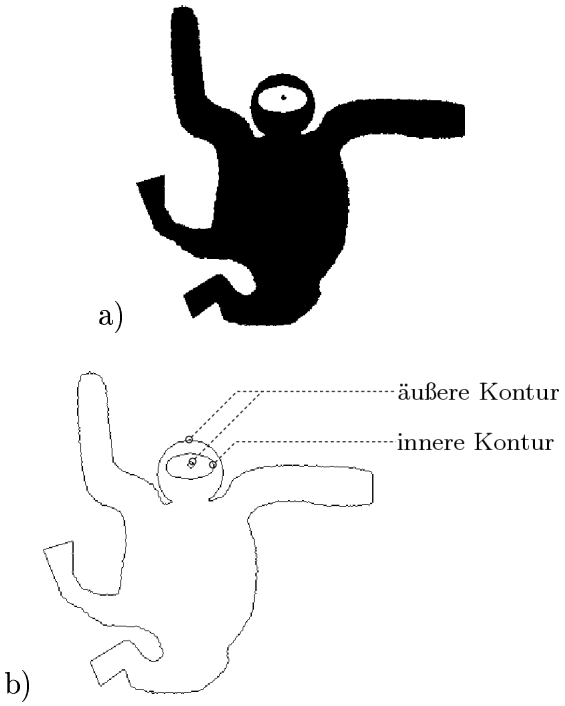
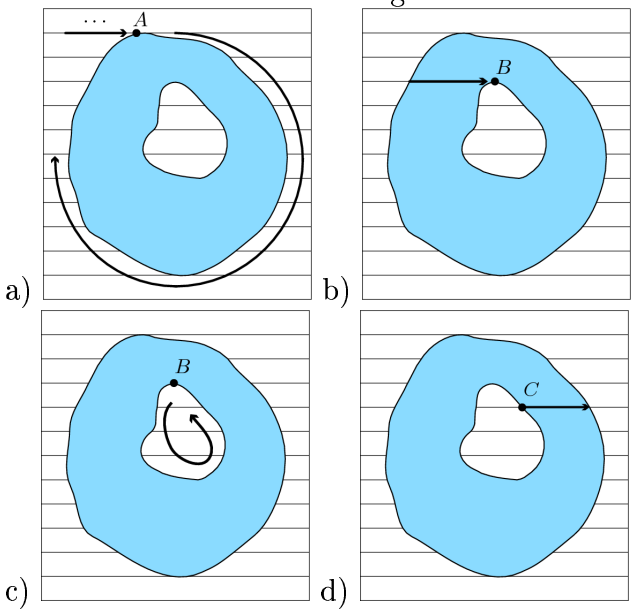


Abbildung 8: Kombiniertes Verfahren



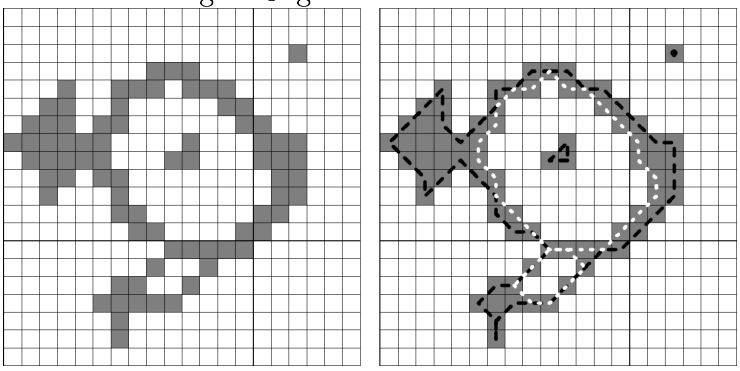
An der aktuellen Bildposition können drei verschiedene Situationen auftreten (vgl Abbildung 8):

1. der Übergang eines Hintergrundpixels auf ein noch nicht markiertes Vordergrundpixel. Das heißt wir haben die äußere Kontur einer neuen Region gefunden. Es wird ein neues Label erzeugt und die zugehörige äußere Kontur im Uhrzeigersinn umfahren, sowie alle Randpunkte der Region mit dem Label markiert. Die unmittelbar angrenzenden Hintergrundpixel bekommen den Wert -1.

2. der Übergang eines Vordergrundpixels auf ein noch nicht markiertes Hintergrundpixel. Das heißt wir haben eine innere Kontur gefunden. Ausgehend von dem Vordergrundpixel wird die innere Kontur gegen den Uhrzeigersinn umfahren und die Pixel der Region mit dem gleichen Label des Vordergrundpixels markiert. Die angrenzenden Hintergrundpixel werden mit -1 markiert.

3. oder der Übergang eines Vordergrundpixels auf ein noch nicht markiertes Vordergrundpixel. Das heißt wir befinden uns innerhalb einer Region. Das neue Vordergrundpixel bekommt den gleiche Label zugewiesen wie das vorhergehende.

Abbildung 9: Orginalbild und mit äußeren und inneren Konturen



1.4.1 Beispiel

In Abbildung 9 bezeichnen die schwarzen Linien die äußeren Konturen, die weißen die inneren Konturen. Innere und äußere Konturen überschneiden sich zum Teil.

2 Konturenverstärkung

2.1 Grauwertbilder

Grauwertbilder sind Bilder, die durch die Funktion

$$B : \mathbb{N}x\mathbb{N} \rightarrow [0, g_{max}]$$

beschrieben werden können. Ebenso kann ein RGB-Farbbild (red-green-blue) in seine drei Farbanteile zerlegt und durch B beschrieben werden, indem die einzelnen Rasterbilder übereinander gelegt werden.

Um Bilder erkennen zu können, ist es wichtig, Konturen zu finden, sprich Begrenzungslinien von Bildobjekten verstärken oder sonstige Kontraste zu verringern. Dabei muss die Funktion B entsprechend bearbeitet werden.

2.2 Konturenverstärkung durch Differenzenbildung

Um Konturen verstärken zu können ist ein erster Ansatz Masken zu verwenden. Es wird eine quadratische $(2n+1)\,x\,(2n+1)$ Matrix M, die Maske, zentriert über den betrachteten Bildpunkt gelegt. Es ist wichtig, dass die Kantenlänge der Matrix ungerade ist, damit der betrachtete Bildpunkt genau in der Mitte der Maske liegt. Nun werden die Bildelemente mit den

sie jeweils überdeckenden Matrixelementen multipliziert. Die Summe dieser Produkte ergibt den neuen Bildwert.

Die Matrizen M können folgendermaßen aussehen:

Zur Auffindung von Konturen in x-, bzw. y-Richtung wendet man die Masken G_x, bzw. G_y an.

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},$$

Zur Auffindung von Konturen in Diagonalenrichtung sind die Masken D_1 und D_2 hilfreich.

$$D_1 = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}, D_2 = \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

Damit haben wir vier Masken, die die verschiedenen Richtungen der Achternachbarschaft beschreiben.

Verläuft eine Bildkontur nun zum Beispiel entlang einer vertikalen Richtung, liegt sie irgendwann bei Anwendung der Maske G_x auf dem mittleren Spaltenvektor, dem Nullvektor. Eine Kante bedeutet verschiedene Graustufenwerte der Pixel rechts und links von ihr (bei wie in diesem Fall einer vertikal verlaufenden Kante). Somit ergibt sich ein Wert $\neq 0$ bei der Multiplikation und Summation der Maskenwerte mit den Bildwerten, so dass der mittlere Kantenpunkt einen betragsmäßig größeren Wert erhält als im Orginalbild.

Eine Kante bedeutet also einen Anstieg der Grauwerte in eine Richtung. Unter dieser Beobachtung können wir die Masken wie folgt herleiten:

Wir interpretieren das Grauwertbild als eine differenzierbare Funktion

$$B : \mathbb{R}^2 \to \mathbb{R}$$

und betrachten den Gradienten$(B_x, B_y)^T$ in einem Punkt (x, y)des Bildes. Ist$(B_x, B_y)^T$ betragsmäßig groß, heißt das, es gibt eine Steigung im Punkt (x, y), wir haben also einen Kantenpunkt gefunden.

Über die angenäherten partiellen Ableitungen von B_x und B_y können wir den Gradienten im Punkt (x, y) bestimmen

$$B_x = \frac{\partial B}{\partial x} = 0.5(B(x+1, y) - B(x-1, y))$$

$$B_y = \frac{\partial B}{\partial y} = 0.5(B(x, y+1) - B(x, y-1))$$

Nehmen wir eine größere Umgebung für die Berechnung der Ableitungen, erhalten wir die folgende Gleichung

$$B_y = \frac{\partial B}{\partial y} = \frac{1}{8}((B(x-1,y+1) - B(x-1,y-1))$$

$$+2(B(x,y+1) - B(x,y-1)) + (B(x+1,y+1) - B(x+1,y-1))).$$

Dies entspricht genau der Anwendung von G_y im Punkt (x, y) multipliziert mit einem Faktor $\frac{1}{8}$.

Entsprechend lassen sich die Masken G_x, D_1 und D_2 ermitteln.

Zur Kantenverstärkung lässt sich folgende Maske aus der Summe der zweiten partiellen Ableitungen nach x und y bestimmen

$$\frac{\partial^2 B}{\partial x^2} + \frac{\partial^2 B}{\partial y^2} \cong ((B(x+1,y) - B(x,y)) - (B(x,y) - B(x-1,y)))$$

$$+((B(x,y+1) - B(x,y)) - (B(x,y) - B(x,y-1))) =$$

$$= B(x+1,y) + B(x-1,y) + B(x,y-1) + B(x,y-1) - 4B(x,y)$$

$$=: L$$

Zieht man nun die Maske L von dem Bildpunkt ab, erhält man eine Maske, die zur Verstärkung der Kanten benutzt werden kann.

$$\Rightarrow \quad I - L = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

2.3 Beispiel

Mit Hilfe eines kleinen Programmes lassen sich die verschiedenen Masken auf ein Bild, vgl. Abbildung 10, anwenden.

Bei Anwendung der Maske $I - L$, vgl. Abbildung 11, ist nach dem ersten Programmdurchlauf nur eine minimale Verbesserung erkennbar. Nach dem zweiten Durchlauf ergeben sich seltsame Strukturen am Rande des E's, was sich bei weiteren Programmdurchlaufen noch verschlimmert. Daher finde ich diese Maske bei dem Beispielbild nicht sehr hilfreich.

Bessere Ergebnisse liefert die Anwendung der Masken G_x und G_y, vgl. Abbildung 12. Im ersten Bild bei Anwendung der Maske G_x kann man sehr

Abbildung 10: Orginalbild (mit dem Gaußfilter geglättet):

Abbildung 11: Bei Anwendung von $I - L$

Orginalbild 1. Programmdurchlauf 2. Programmdurchlauf

Abbildung 12: Anwendung von G_x und G_y

mit G_x

mit G_y

mit $G_x + G_y$

Abbildung 13: Anwendung von D_1 und D_2


mit D_1


mit D_2


mit $D_1 + D_2$

gut sehen, wie die senkrecht verlaufenden Kanten gefunden werden, die waagrecht verlaufenden dagegen schwarz werden. Bei Anwendung der Maske G_y ist genau das umgekehrte der Fall, die waagrechten Kanten sind weiß, die senkrechten wurden nicht entdeckt.

Auch die Anwendung der Masken D_1 und D_2, vgl. Abbildung 13, liefern ein ähnliches Ergebnis in der Summe wie G_x+G_y, bei den Einzelanwendungen kann man sehen, dass die diagonalen Kanten je nach Maske nicht gefunden werden.

Verlaufen Kanten in andere Richtungen als genau vertikal, horizontal oder diagonal, bekommt man mit diesen Masken Schwierigkeiten, die Kanten zu finden. Dafür sind weitere Verfahren notwendig, vgl. den Vortrag *Kantenextraktion: klassische Ansätze* von C. Wagner.

▷

Literatur

[1] W. Burger, M. J. Burge „Digitale Bildverarbeitung", Springer 2005.

[2] A. Janser, W. Luther, W. Otten „Computergraphikund Bildverarbeitung", Vieweg 1996

[3] www.imagingbook.com

[4] U. Schöning „Algorithmik", Spektrum Akademischer Verlag 2001