Computergrafik

Übung: Clipping & Bildoperationen

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

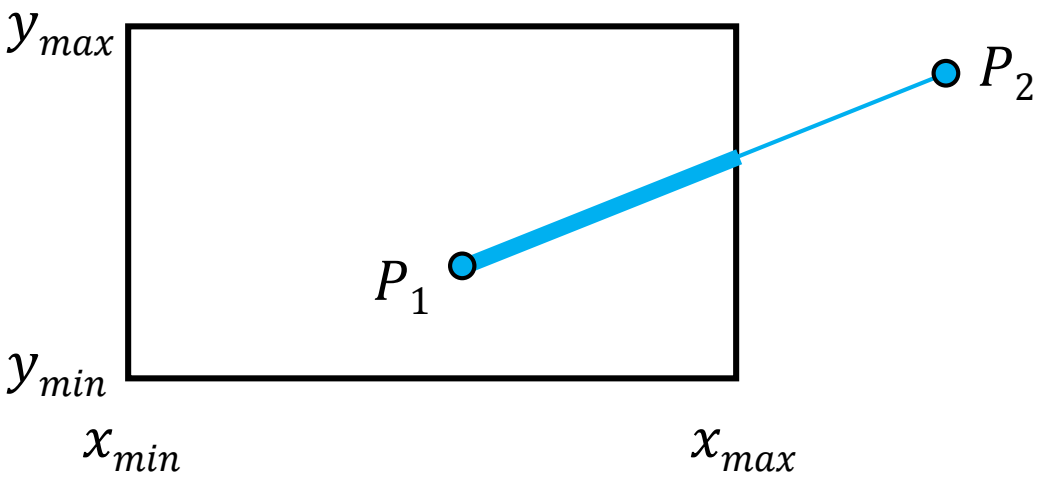
Clipping

- Clipping: Abschneiden von Linien/Polygon-Teilen, die außerhalb des interessanten Bereichs liegen
- Effizient: zeichne nichts außerhalb des Bildschirms/View Frustums, keine Objekte hinter der Kamera (in 3D)

- Clipping von Linien
  - Cohen-Sutherland und α-Clipping
- Clipping von Polygonen
  - Sutherland-Hodgeman

Clipping von Linien

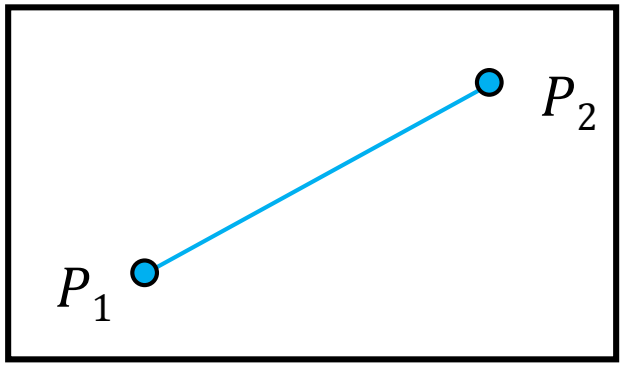
- gegeben
  - eine 2D Linie von $P_1 = (x_1, y_1)$ nach $P_2 = (x_2, y_2)$
  - ein Rechteck definiert durch $(x_{min}, x_{max}, y_{min}, y_{max})$
- bestimme den Teil der Linie innerhalb des Rechtecks.



Clipping Algorithmus von Cohen-Sutherland

Clipping von Linien gegen Rechtecke

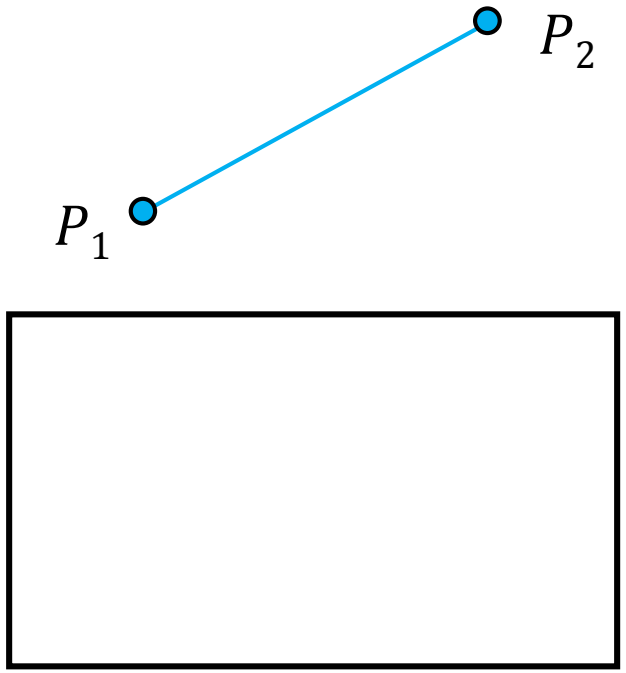
- Einfacher Fall: trivial accept



Clipping Algorithmus von Cohen-Sutherland

Clipping von Linien gegen Rechtecke

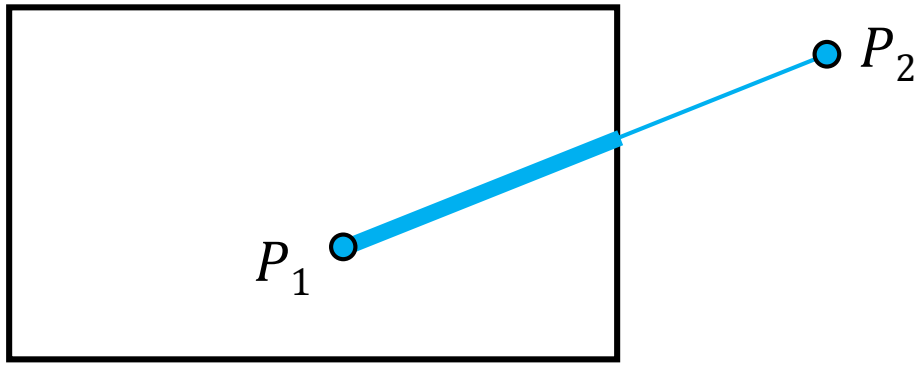
- Noch ein einfacher Fall: trivial reject



Clipping Algorithmus von Cohen-Sutherland

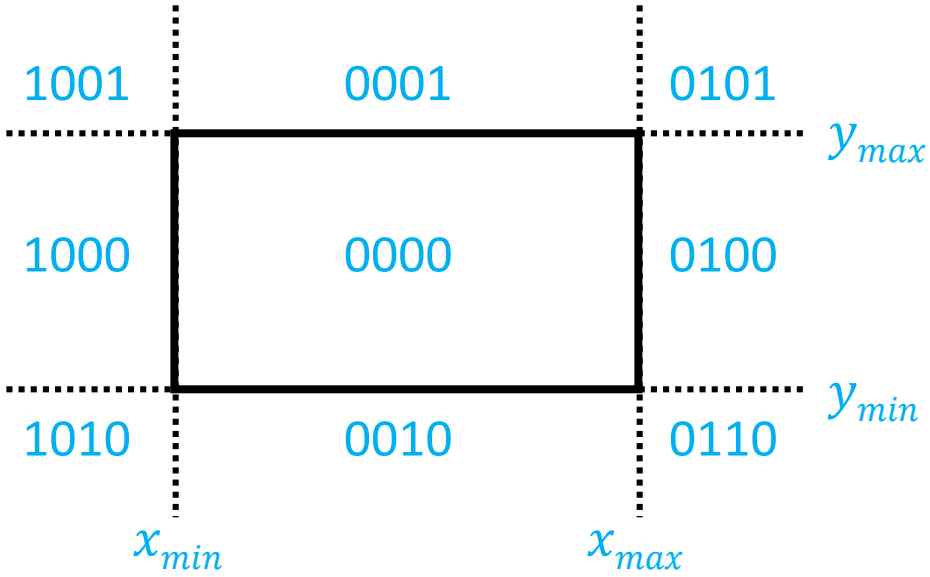
Clipping von Linien gegen Rechtecke

- Ansonsten: Unterteile das Liniensegment



Clipping Algorithmus von Cohen-Sutherland

- unterteile die Ebene in 9 Bereiche



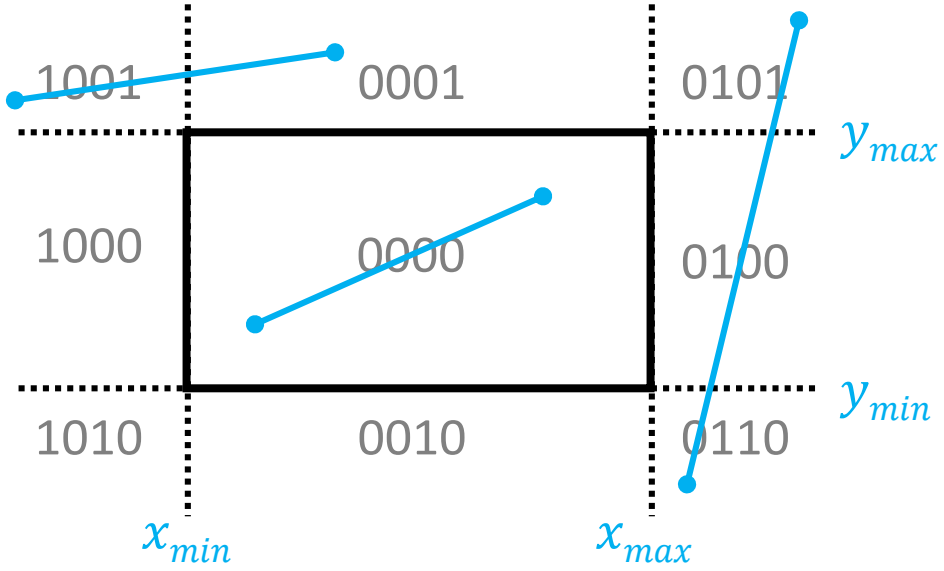
- „Outcode“ =

$x < x_{min}$	$x > x_{max}$	$y < y_{min}$	$y > y_{max}$

  - jedes Bit entspricht einer Kante des Clipping Rechtecks
  - Bit gesetzt ↔ Punkt liegt ausserhalb bzgl. dieser Kante

Clipping Algorithmus von Cohen-Sutherland

Algorithmus zum Clipping von Linien



- trivial accept

  Beide Punkte innen $\Leftrightarrow$ Outcode(P_1) V Outcode(P_2) == 0

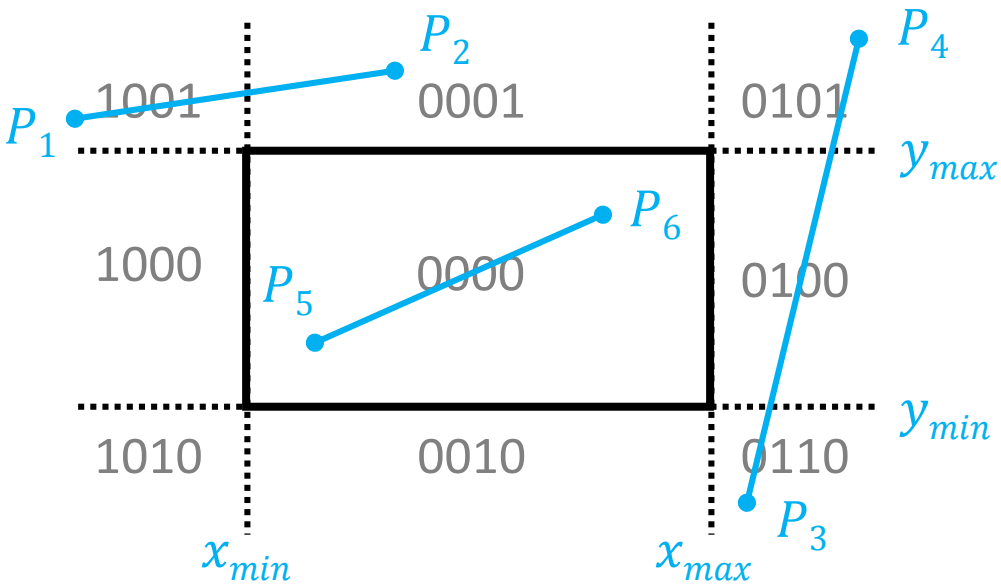
- trivial reject

  Beide Punkte außen bzgl. derselben Kante $\Leftrightarrow$ Outcode(P_1) $\wedge$ Outcode(P_2) $\neq 0$

Clipping Algorithmus von Cohen-Sutherland

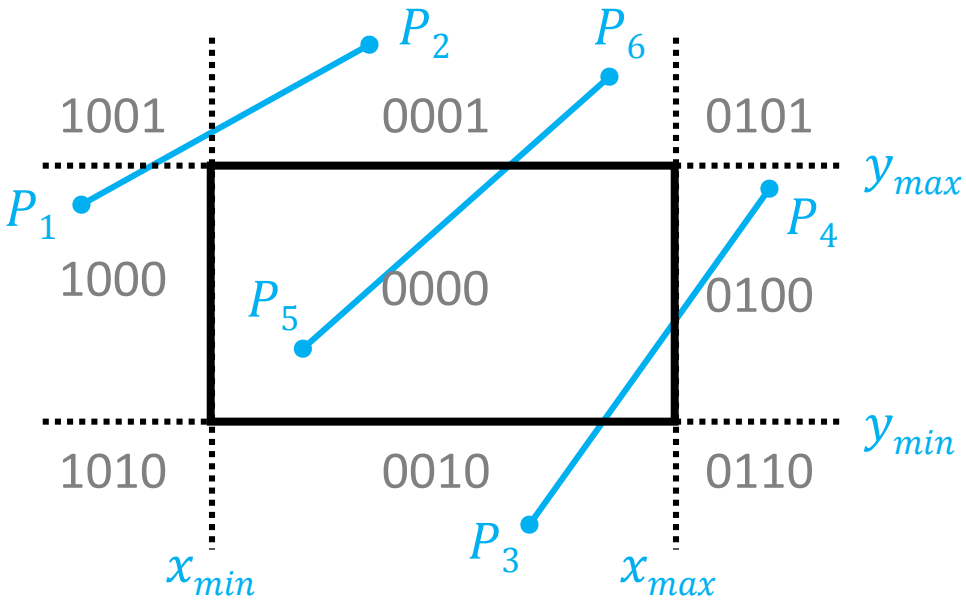
Beispiele

- Outcode (P_1) ∧ Outcode (P_2) = 0001 ≠ 0 ⇒ trivial reject
- Outcode (P_5) ∨ Outcode (P_6) = 0000 == 0 ⇒ trivial accept
- Outcode (P_3) ∧ Outcode (P_4) = 0100 ≠ 0 ⇒ trivial reject



Clipping Algorithmus von Cohen-Sutherland

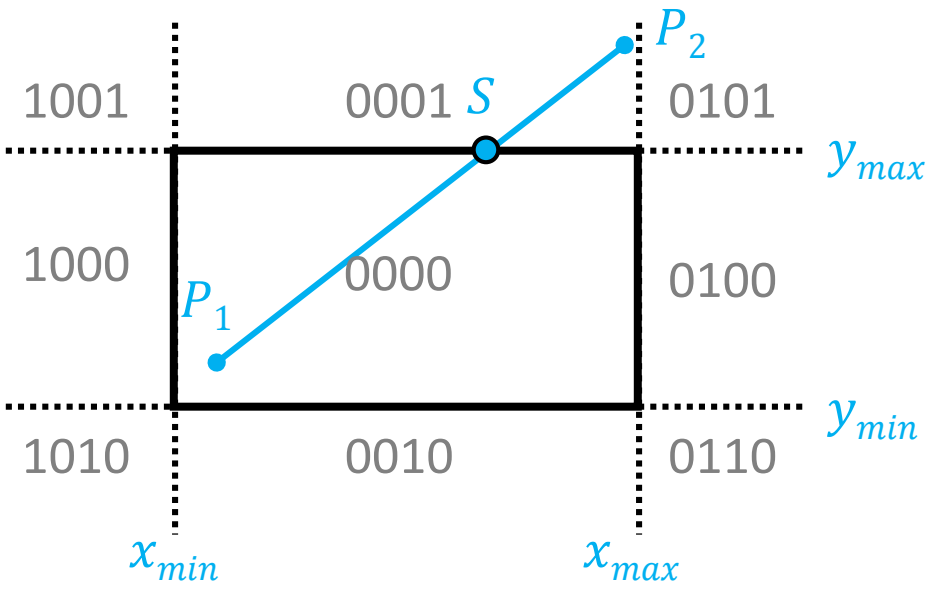
- Nicht-triviale Fälle



Clipping Algorithmus von Cohen-Sutherland

Algorithmus zum Clipping von Linien *cont.*

- Es gibt mindestens eine Kante, für die das Bit in in Outcode(P_1) gesetzt und in Outcode(P_2) nicht gesetzt ist (oder umgekehrt)
- die Linie P_1P_2 schneidet diese Kante
  - berechne den Schnittpunkt S
  - wenn P_1 außerhalb bzgl. dieser Kante liegt
    → verfahre weiter mit SP_2, sonst mit P_1S

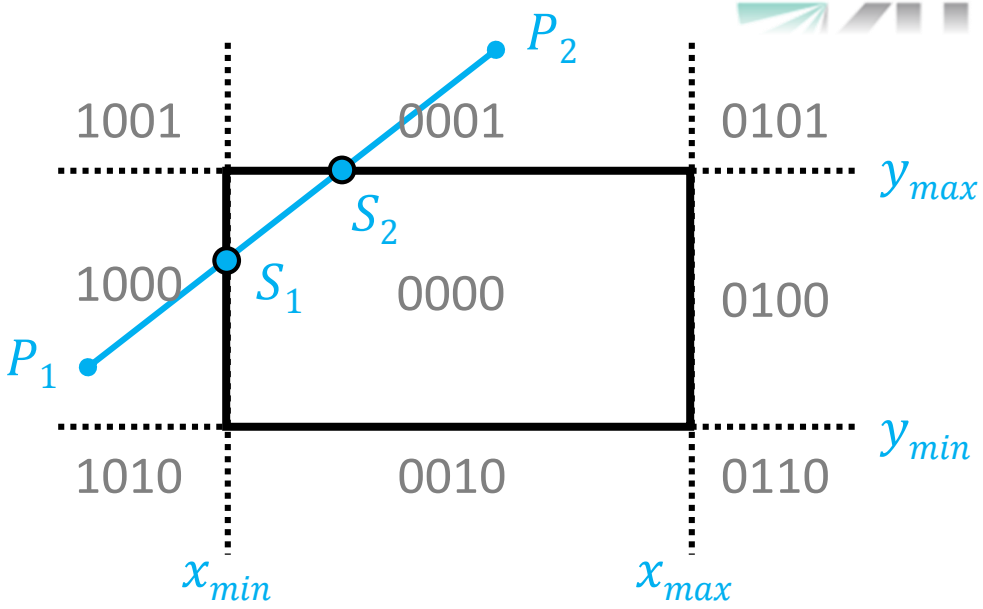


Clipping Algorithmus von Cohen-Sutherland

Beispiel

- Outcode (P_1) = 1000
- Outcode (P_2) = 0001

- Outcode (S_1) = 0000
- Outcode (S_2) = 0000



Outcode (P_1) ∨ Outcode (P_2) ≠ 0	⇒ kein trivial accept	
Outcode (P_1) ∧ Outcode (P_2) == 0	⇒ kein trivial reject	
Outcode (P_1) "left"-Bit gesetzt	⇒ berechne S_1	⇒ weiter mit S_1P_2
Outcode (S_1) ∨ Outcode (P_2) ≠ 0	⇒ kein trivial accept	
Outcode (S_1) ∧ Outcode (P_2) == 0	⇒ kein trivial reject	
Outcode (P_2) "top"-Bit gesetzt	⇒ berechne S_2	⇒ weiter mit S_1S_2
Outcode (S_1) ∨ Outcode (S_2) == 0	⇒ trivial accept (S_1,S_2)	

Clipping optimieren

α-Clipping (nach Cyrus-Beck '78, Liang-Barsky '87)

- Optimierung mittels **W**indow **E**dge **C**oordinates (WEC)
  - $WEC_E(P) < 0 \iff P$ liegt außerhalb bzgl. der Kante E
    - $WEC_{left}(P) = p_x - x_{min}$
    - $WEC_{right}(P) = x_{max} - p_x$
    - $WEC_{bottom}(P) = p_y - y_{min}$
    - $WEC_{top}(P) = y_{max} - p_y$

- Das entsprechende Outcode-Bit ist gesetzt, wenn dazugehörige WEC negativ ist

α-Clipping

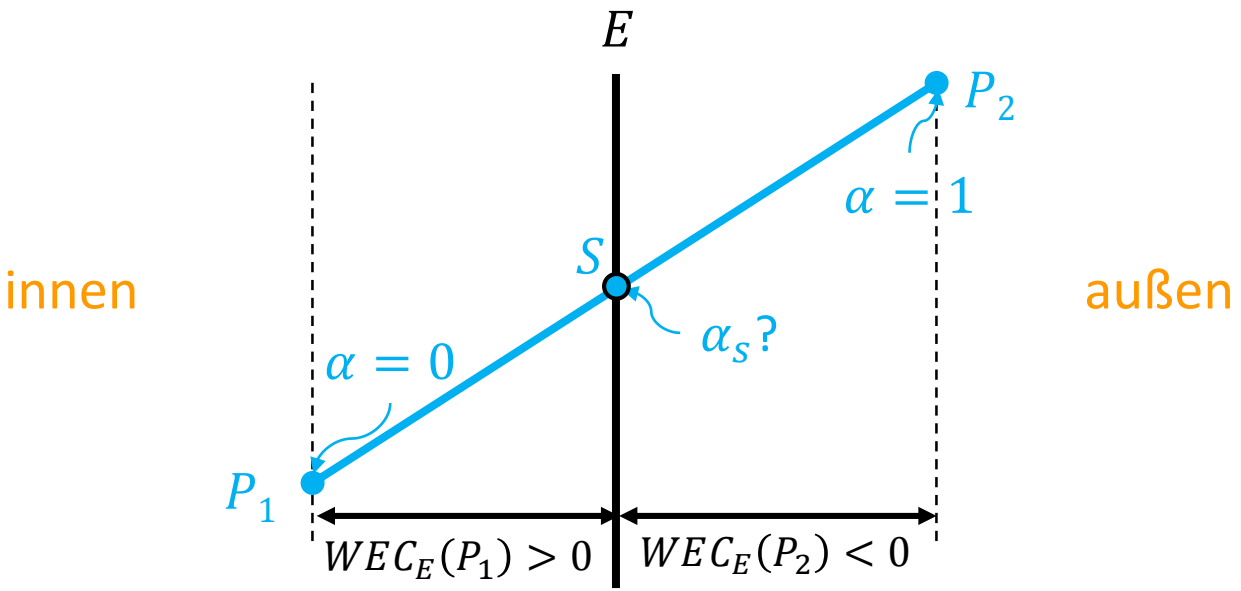
Schnittberechnung mit WECs

- parametrisiere Linie P_1P_2 mit α:

$$P = P_1 + \alpha(P_2 - P_1), \alpha \in [0; 1]$$

- α-Wert für den Schnittpunkt mit einer Kante (Strahlensatz!)

$$\frac{\overline{P_1S}}{\overline{P_1P_2}} = \alpha_s = \frac{WEC_E(P_1)}{WEC_E(P_1) - WEC_E(P_2)}$$



α-Clipping

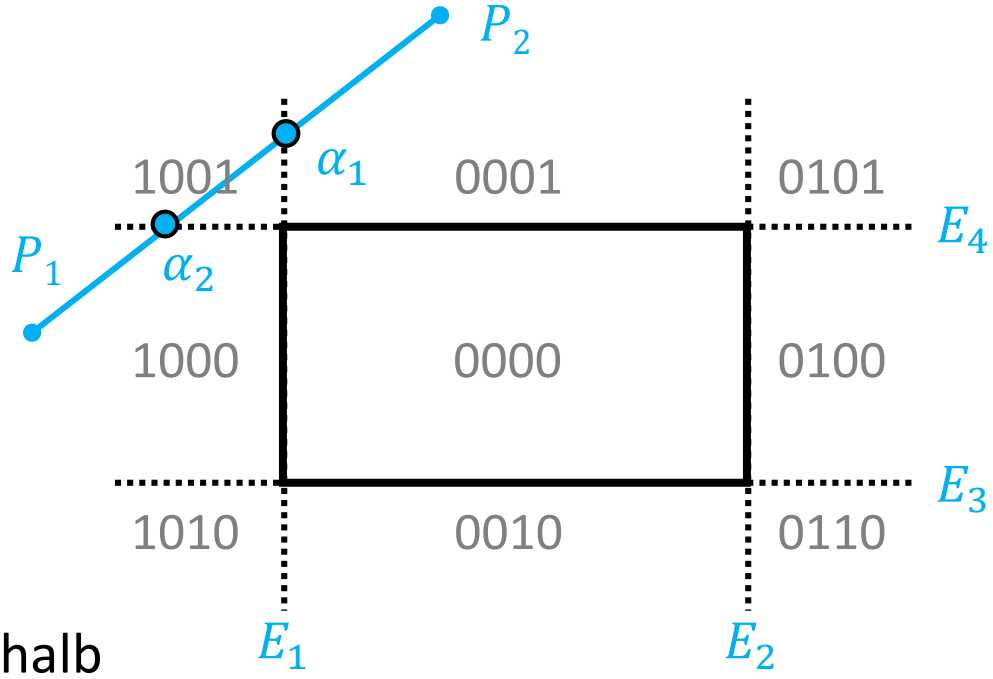
Algorithmus

- berechne WECs von P_1 und P_2
- bestimme Outcodes anhand des Vorzeichens der WECs
- teste auf trivial accept/trivial reject (wie bei Cohen-Sutherland)
- ansonsten verfahre so:
  - setze $\alpha_{min} = 0, \alpha_{max} = 1$
  - für jede Kante E für die das „Outcode$_E$“-Bit für P_1 oder P_2 gesetzt ist
    - berechne $\alpha_s = \frac{WEC_E(P_1)}{WEC_E(P_1) - WEC_E(P_2)}$
    - wenn Outcode$_E(P_1)$
      - dann $\alpha_{min} = \max(\alpha_{min}, \alpha_s)$
      - sonst $\alpha_{max} = \min(\alpha_{max}, \alpha_s)$
    - wenn $\alpha_{min} > \alpha_{max}$, dann liegt die Linie außerhalb
  - geclipptes Liniensegment $\left(P_1 + \alpha_{min}(P_2 - P_1), P_1 + \alpha_{max}(P_2 - P_1)\right)$

α-Clipping

Beispiel 1

- berechne WECs für P_1 und P_2 bzgl. E_1, E_2, E_3, E_4
- bestimme Outcodes
  - Outcode$(P1) = 1000$
  - Outcode$(P2) = 0001$
- kein trivial accept/trivial reject

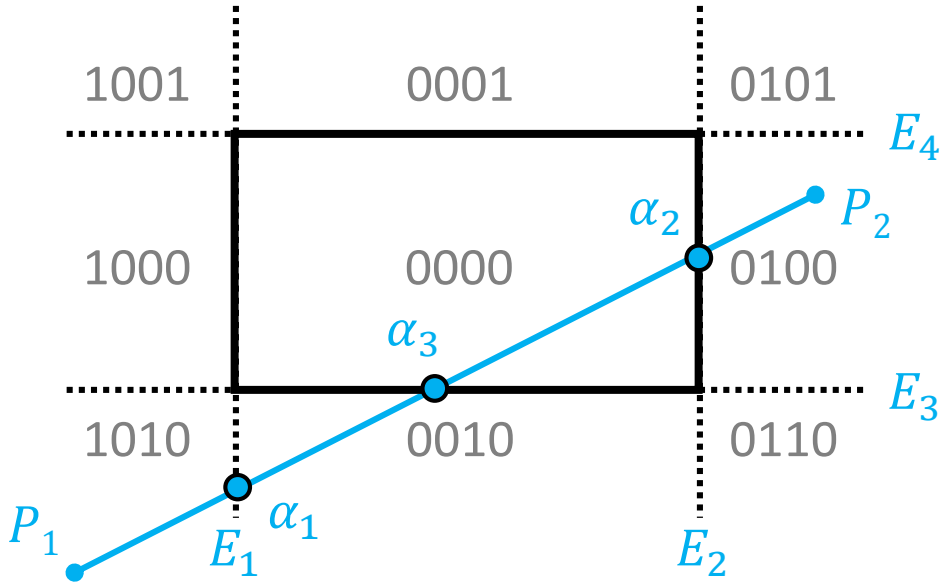


- Clipping:
  - Kante $E_1 \rightarrow \alpha_{min} = \alpha_1$
  - Kante $E_4 \rightarrow \alpha_{max} = \alpha_4$
  - $\alpha_{max} < \alpha_{min} \Rightarrow$ Linie außerhalb

α-Clipping

Beispiel 2

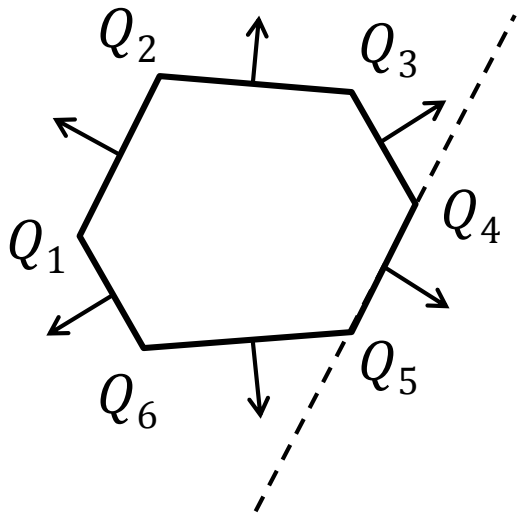
- Kante $E_1 \rightarrow \alpha_{min} = \alpha_1$
- Kante $E_2 \rightarrow \alpha_{max} = \alpha_2$
- Kante $E_3 \rightarrow \alpha_{min} = \alpha_3$
- gebe Liniensegment zurück, denn $\alpha_{min} < \alpha_{max}$



α- Clipping - Konvexe Clipping Regionen

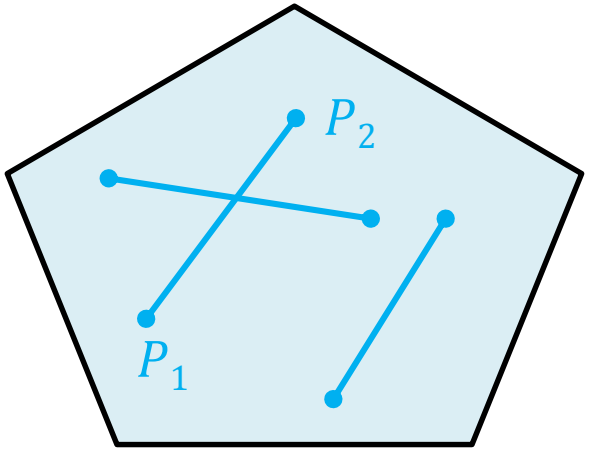
- Für ein Polygon Q_1,…, Q_n ergibt sich
  - n Kanten → n WECs
  - $WEC_i(P)$: gerichteter Abstand von P zur Linie Q_iQ_{i+1}

- Clipping in 3D
  - analog, WEC sind die Abstände zu Clipping-Ebenen

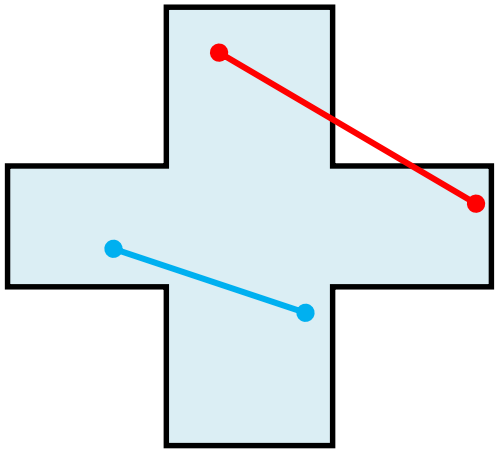


Konvexe Clipping Regionen

- Definition
  - eine Menge von Punkten $\mathrm{M} \subseteq \mathbb{R}^2$ ist konvex, wenn für alle $P_1, P_2 \in \mathrm{M}$ alle Punkte auf der Linie $\overline{P_1 P_2}$ ebenfalls in M liegen



konvex

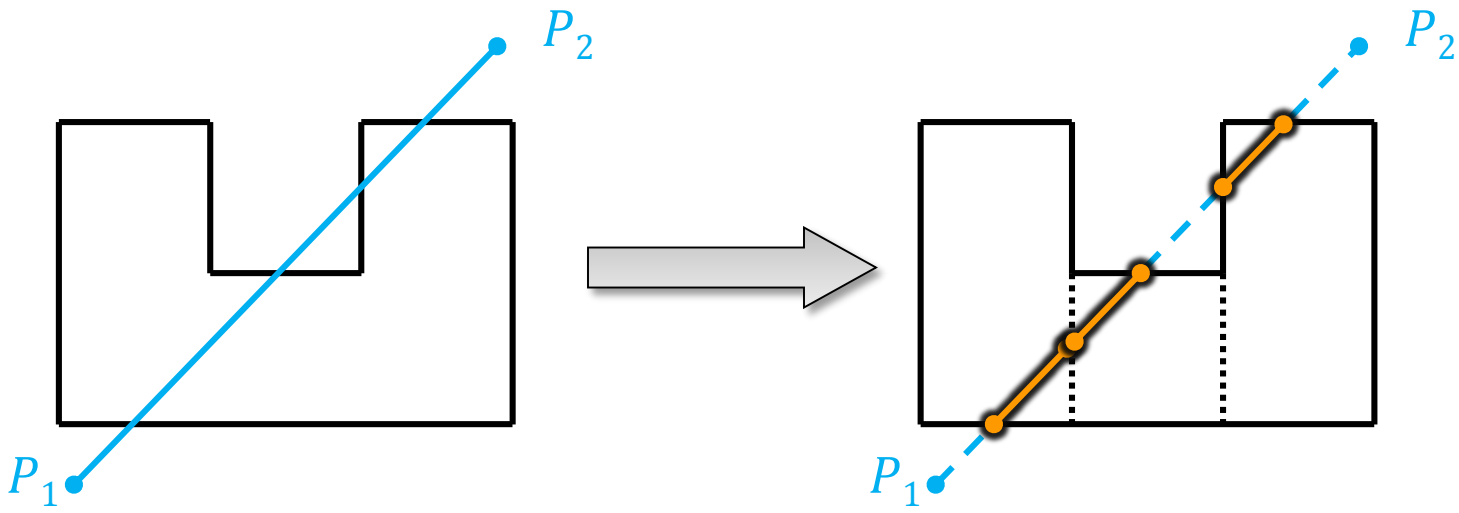


nichtkonvex

- Achtung: nichtkonvex != konkav

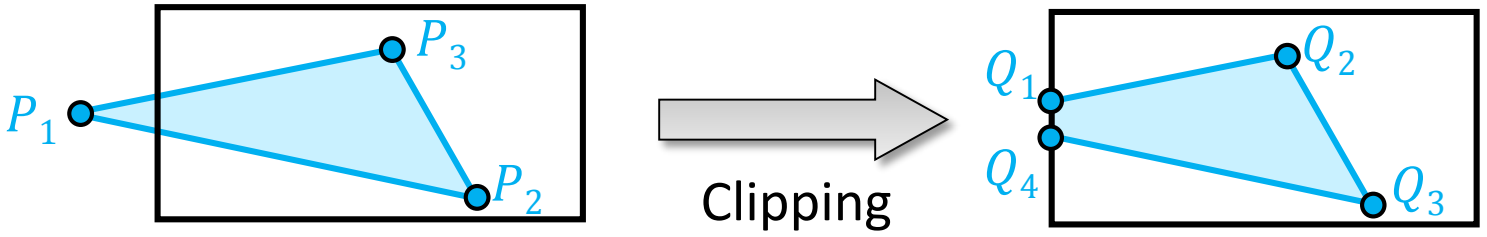
Nichtkonvexe Clipping Regionen

- unterteile in konvexe Teilregionen
- anschließend Clipping gegen jede Teilregion…
- …und zusammensetzen der Liniensegmente



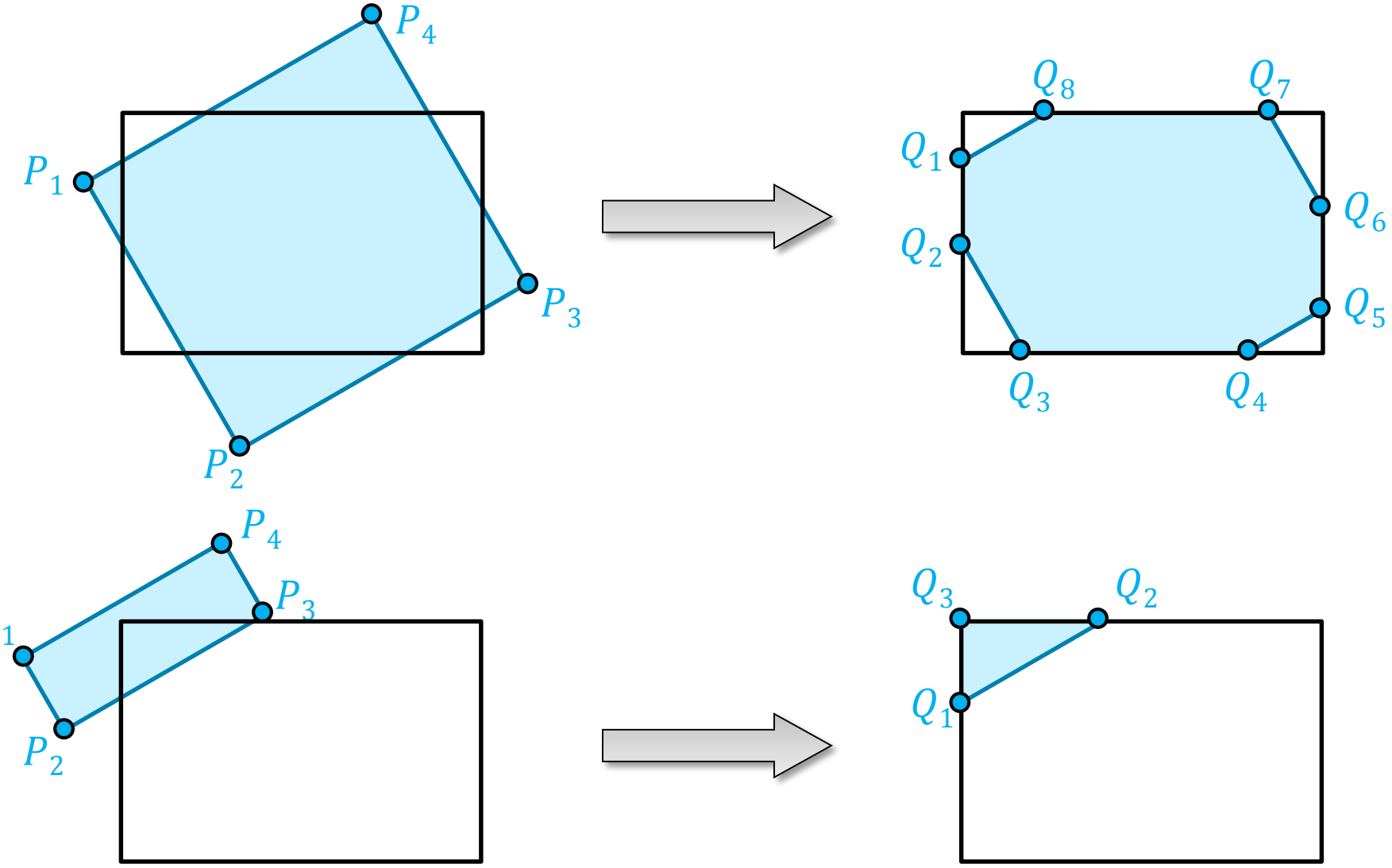
Clipping von Polygonen

- Polygon: Geordnete Liste von Punkten



Clipping von Polygonen

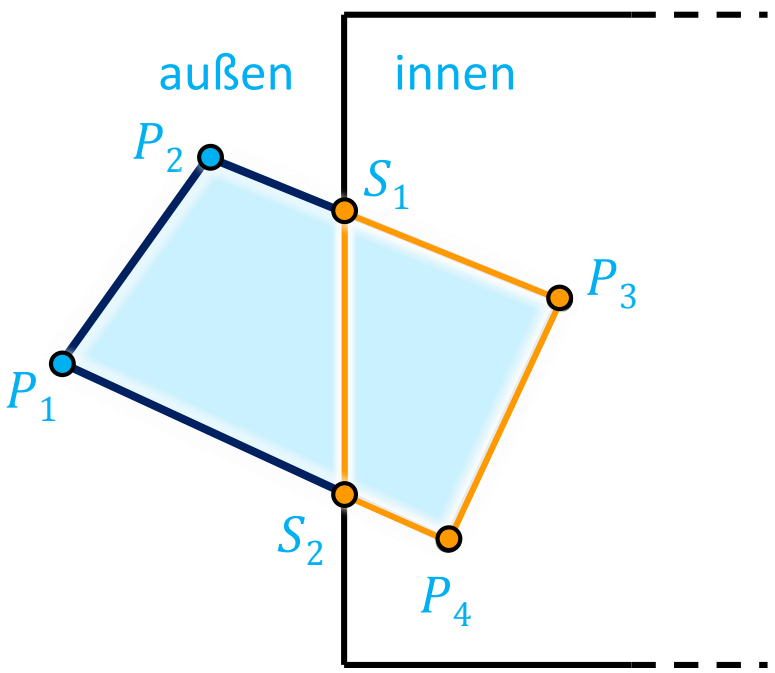
- wir brauchen mehr als nur die Liniensegmente, die durch Clipping der Polygonkanten entstehen → Algorithmus von Sutherland-Hodgeman (1974)



Sutherland-Hodgeman Polygon Clipping

Algorithmus für (konvexe) Regionen in 2D

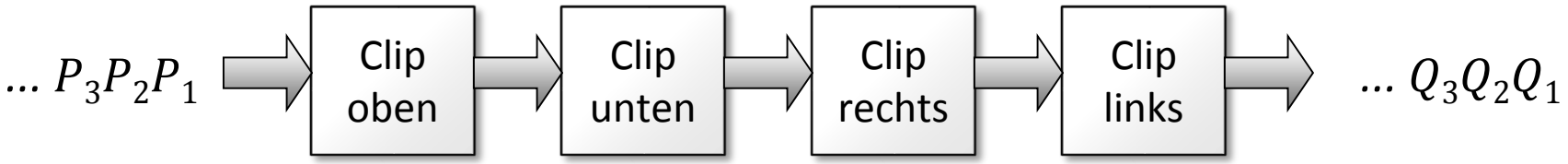
- Clipping gegen eine Kante des Clipping-Polygons nach der anderen
- für jede Kante
  - Eingabe: Liste der Vertizes des Polygons
  - Ausgabe: Liste der Vertizes des resultierenden Polygons
  - Beispiel:
    - Eingabe: P_1, P_2, P_3, P_4
    - Ausgabe: S_1, P_3, P_4, S_2

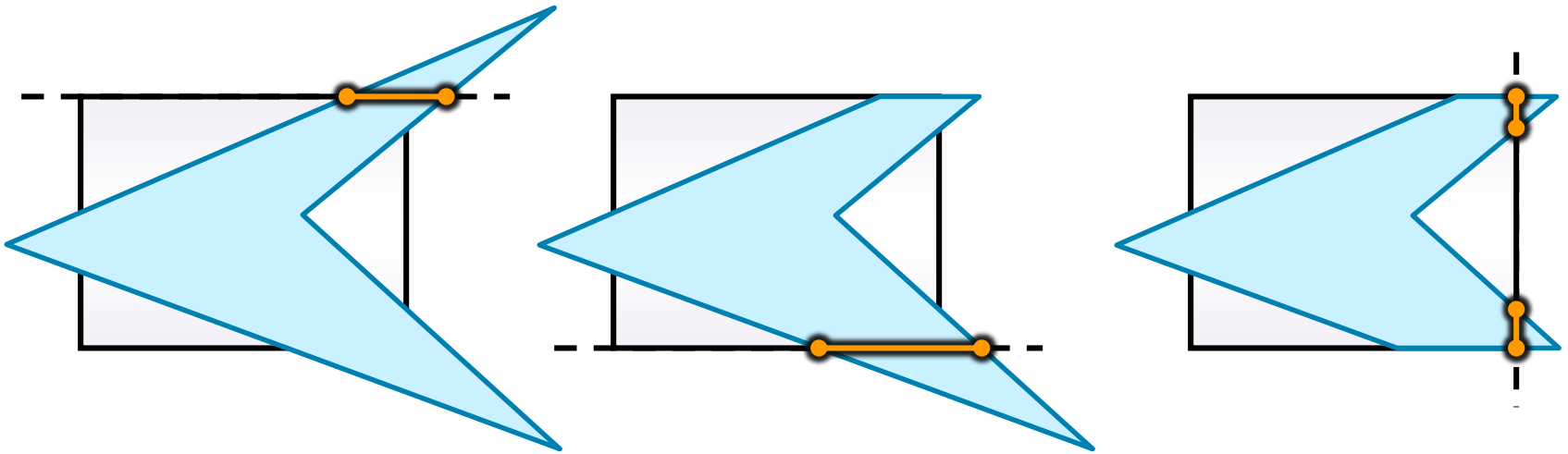


Sutherland-Hodgeman Polygon Clipping

Algorithmus für (konvexe) Viewports in 2D

- Clipping gegen eine Kante des Clipping-Polygons nach der anderen
  - nach jeder Kante erhält man ein geschlossenes Polygon
  - erlaubt Pipelining (wichtig für Hardware-Umsetzungen)

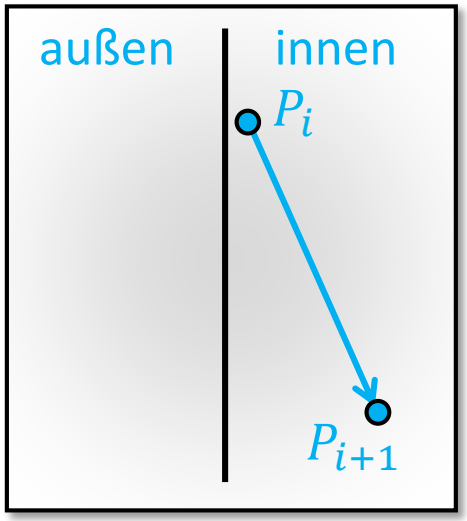




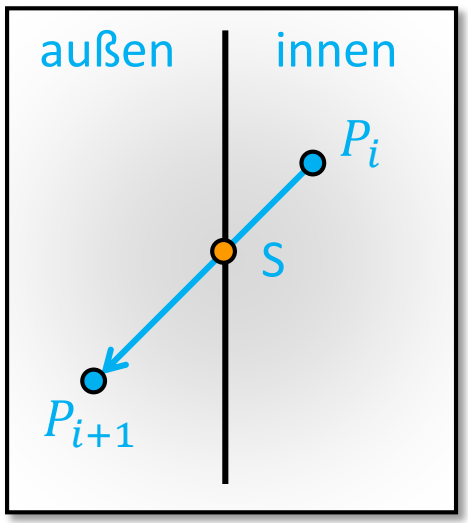
Sutherland-Hodgeman Polygon Clipping

Clipping gegen eine Kante

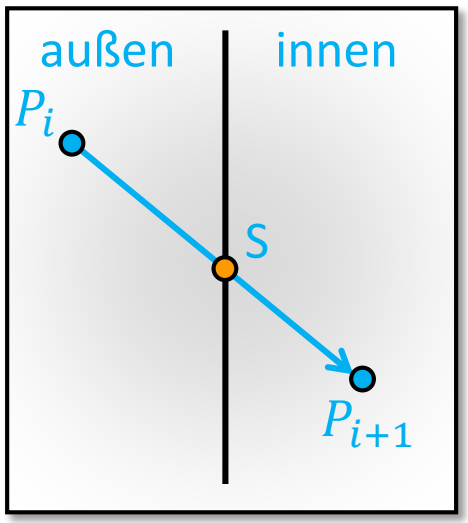
- geg. Liste von Eingabe-Vertizes $P_i, i = 1, \ldots, n$
- betrachte ein Segment $P_i P_{i+1}$ nach der anderen
- klassifiziere P_i bzw. P_{i+1} als innen/außen → 4 Fälle sind zu unterscheiden



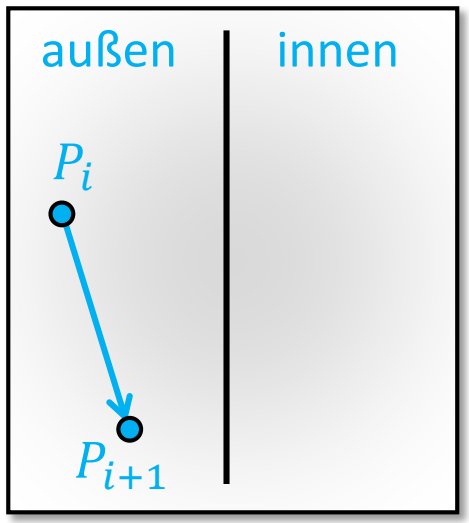
innen → innen:
füge P_{i+1} zur Ausgabe hinzu



innen → außen:
berechne Schnittpunkt S, gebe S aus



außen → innen:
berechne Schnittpunkt S, gebe S und P_{i+1} aus

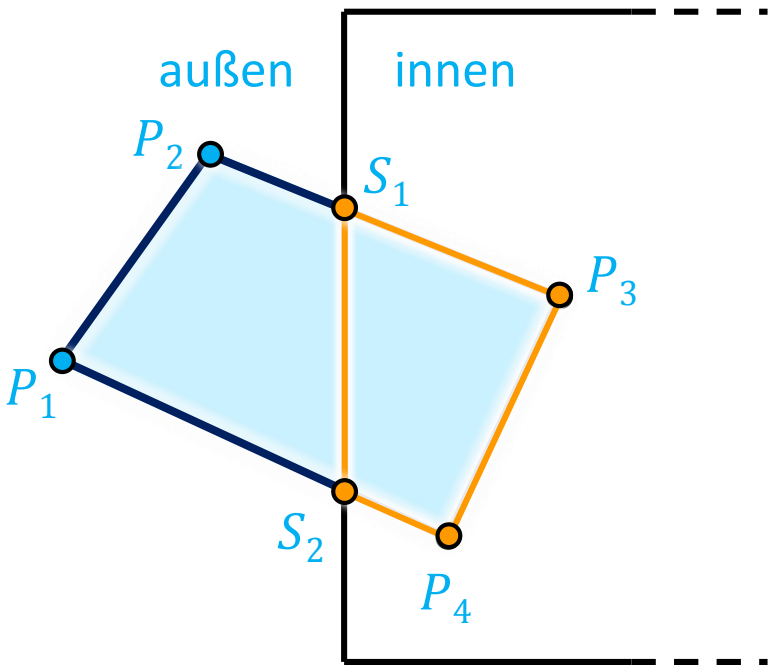


außen → außen:
weiter mit nächstem Segment

Sutherland-Hodgeman Polygon Clipping

Beispiel

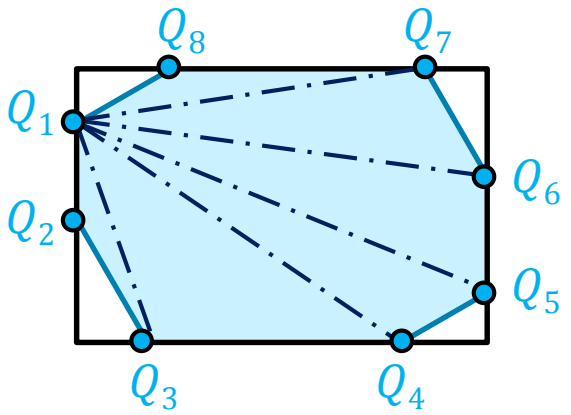
- Eingabe: P_1, P_2, P_3, P_4
- getestete Segmente: $P_1P_2, P_2P_3, P_3P_4, P_4P_1$
- Ausgabe:
  - $P_1P_2 \rightarrow$ nichts
  - $P_2P_3 \rightarrow S_1$ und P_3
  - $P_3P_4 \rightarrow P_4$
  - $P_4P_1 \rightarrow S_2$
  - Segment S_2S_1 schließt das Polygon



Sutherland-Hodgeman Polygon Clipping

Clipping mit Attributen

- Vertex-Attribute (z.B. Texturkoordinaten oder Farben) werden mit und für die Schnittpunkte berechnet
- ist das Eingabepolygon konvex, dann auch das Ausgabepolygon
  - kann bei Bedarf einfach in Dreiecke zerlegt werden:
    Q_1, Q_i, Q_{i+1} mit $1 < i < n$



Clipping in 3D

- Clipping gegen ein konvexes Sichtvolumen
  - statt Clipping gegen Kanten: Clipping gegen Ebenen
  - α-Clipping in homogenen Koordinaten

Computergrafik

Übung: Bildoperationen - Farben & Filter

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

Bildmanipulationen

Wir betrachten folgende Arten von Manipulationen

- Operationen im *Farbraum* („pro Pixel“)
- *Operationen im Farb- und Bildraum*
  - *lineare Filter*: gewichtete Summe benachbarter Pixel-Werte
  - *morphologische Filter*: strukturverändernde Operationen

Helligkeit (Farbraum)

Scharfzeichnung (linearer Filter)

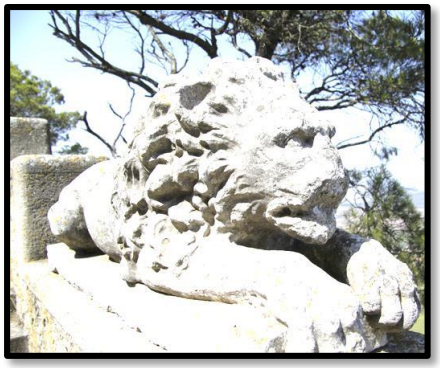
Entfernen kleiner Strukturen (morphologischer Filter)

Bildmanipulationen

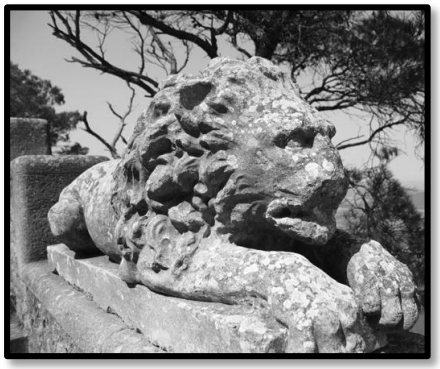
Bildoperationen im Farbraum

- Änderung der Farbe eines Pixels basierend nur auf dem aktuellen Farbwert an dieser Stelle

- Helligkeit $g(r, g, b) = 2(r, g, b)$

- Luminanz $y(r, g, b) = 0.3r + 0.59g + 0.11b$

Beispiel: Änderung des Kontrasts

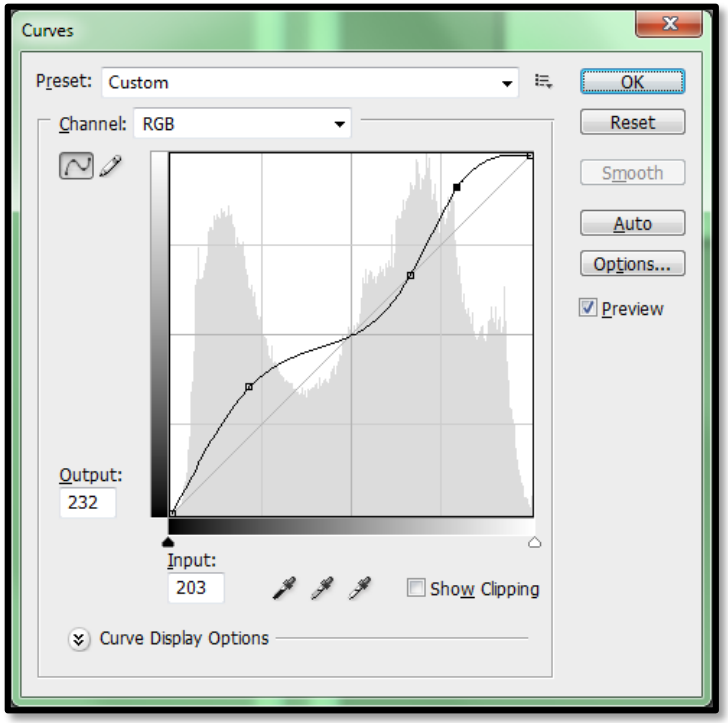
- Kontraständerung: $g(x) = c(x - 0.5) + 0.5, \quad x = (r, g, b)$

$c < 1$

$c > 1$

Gradationskurven, Farbkurven

- visuelle/interaktive Manipulation der Funktion g
- präzise Kontrolle



Operationen im Farb- *und* Bildraum

- die neue Farbe eines Pixels hängt ab von
  - seiner aktuellen Farbe
  - den Farbwerten der Pixelnachbarschaft

- Viele dieser Bildoperationen (sog. Filter) lassen sich durch eine **Faltung** ausdrücken
  - Bsp. Unschärfe, Schärfe, Kantendetektion, Embossing, …

Faltung – lineare Filter

- das Bild ist eine Funktion f, die Positionen eine Farbe zuordnet

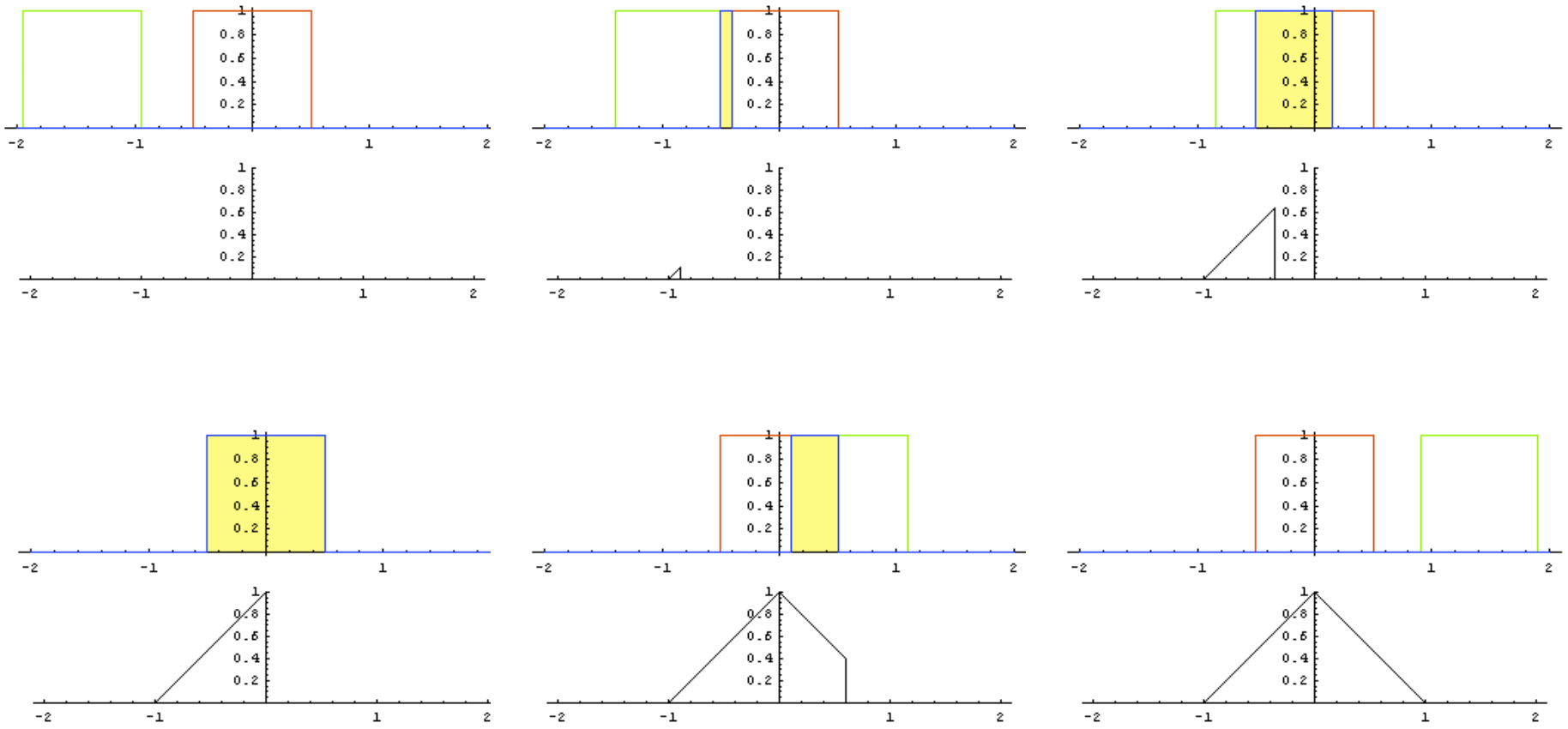
- eine Faltung berechnet die „Überlappung” einer Funktion f mit einer zweiten Funktion g, die gespiegelt über f geschoben wird

$$[f * g](t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

- g nennt man Filterfunktion
- die Faltung zweier Funktionen ist wieder eine Funktion

Faltung

- Faltung zweier Rechteckfunktionen f und g
- das Resultat der Faltung ist eine Dreiecksfunktion (Flächeninhalt des Produkts von f und g für jede Verschiebung)



Diskrete Faltung

- wenn f und g diskret sind (z.B. die Funktionen nur an ganzzahligen Stellen definiert), dann ist die diskrete Faltung

$$[f * g](n) = \sum_{i=-\infty}^{\infty} f(i)g(n-i)$$

- Prinzip
  - die Filterfunktion („Kernel“) g wird mittig um den n-ten Pixel ausgerichtet
  - Gewichtung jedes Pixels des Bildes gemäß des Werts von g an dieser Stelle
  - Aufsummieren der gewichteten Farbwerte ergibt die neue Farbe des n-ten Pixels

Faltung in 2D

- kontinuierlich

$$[f * g](s,t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\sigma,\tau) g(s-\sigma, t-\tau) d\sigma d\tau$$

- diskret

$$[f * g](m,n) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j) g(m-i, n-j)$$

- oft sind Filter $g(.,.)$ nur in einem (kleinen) Intervall ungleich 0

$$[f * g](m,n) = \sum_{i=m-a}^{m+a} \sum_{j=n-b}^{n+b} f(i,j) g(m-i, n-j)$$

  - mit $-a \leq m-i \leq a$ und $-b \leq n-j \leq b$

Bsp. diskrete Faltung in 2D

```
// Falte das Bild "img" mit der Filterfunktion g.
// Der Filterkernel hat die Größe [2a+1, 2b+1]
for (int n = 0; n < img.height; n++) {
  for (int m = 0; m < img.width; m++) {

    float res = 0.0f;
    for (int j = n-b; j <= n+b; j++) {
      for (int i = m-a; i <= m+a; i++) {
        // Achtung: Auf Bereichsüberschreitung achten!
        res += img(i, j) * g(m-i, n-j);
      }
    }

    img(m, n) = res;
  }
}
```

Bsp. diskrete Faltung in 2D

- Annahme: alle Werte außerhalb des Definitionsbereichs von f sind 0
  - man kann aber auch definieren, dass das Bild wiederholt wird, oder
  - man auf den jeweils nächsten Pixelwert zugreift
- der hier verwendeten Kernelmatrix sieht man nicht an, dass sie gespiegelt ist

$$[f * g](m,n) = \sum_{i=m-a}^{m+a} \sum_{j=n-b}^{n+b} f(i,j)g(m-i,n-j)$$

2	3	1
0	5	1
1	0	8

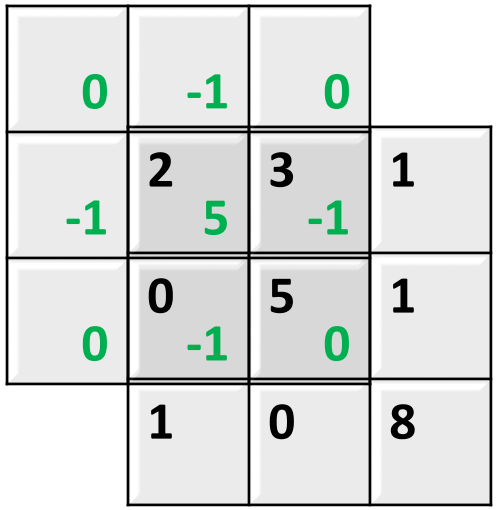
$*$

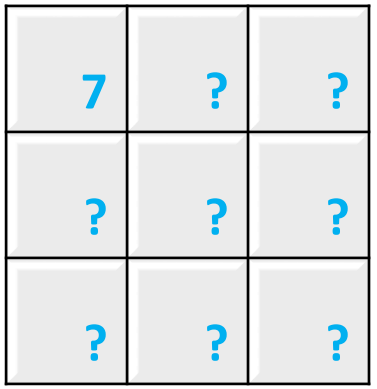
0	-1	0
-1	5	-1
0	-1	0

$=$

7	7	1
-8	21	-9
5	-14	39

Bsp. diskrete Faltung in 2D



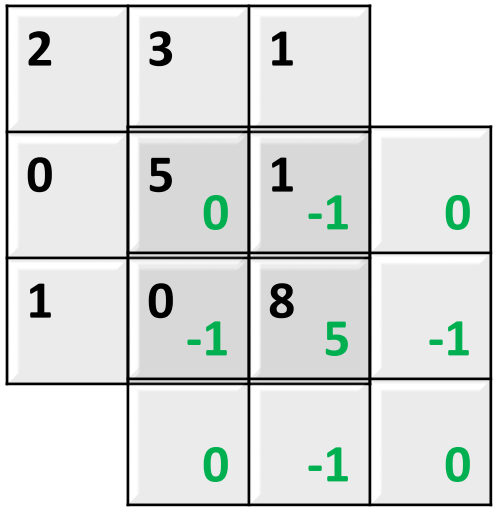


Bsp. diskrete Faltung in 2D

2 0	3 -1	1 0
0 -1	5 5	1 -1
1 0	0 -1	8 0

7	7	1
-8	21	?
?	?	?

Bsp. diskrete Faltung in 2D



7	7	1
-8	21	-9
5	-14	39

Filter: Unschärfe (Blur)

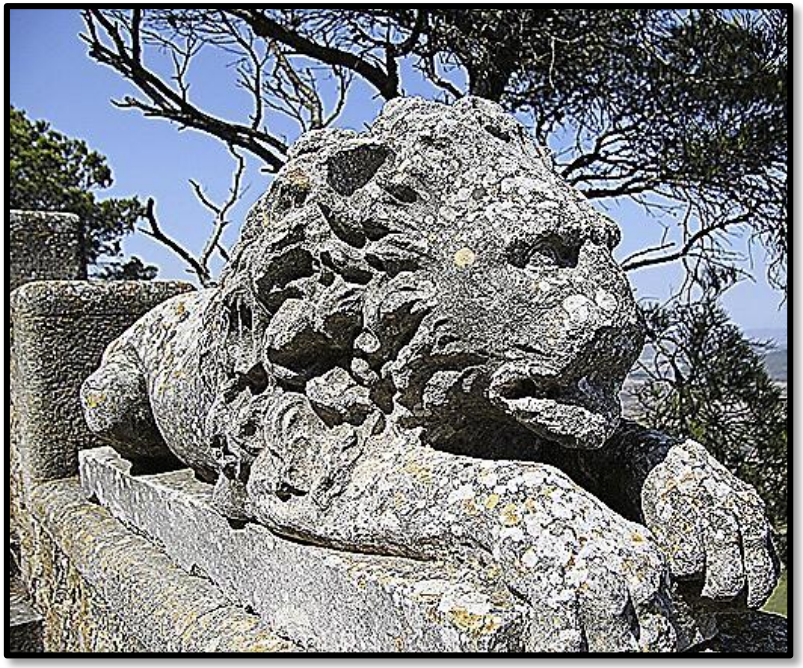
- normalisierter Kernel: die Einträge summieren sich zu 1
  - die Gesamthelligkeit des Bildes bleibt erhalten

$$* \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot 1/9$$

Filter: Schärfe (sharpen)

*

0	-1	0
-1	5	-1
0	-1	0



Filter: Kantendetektion

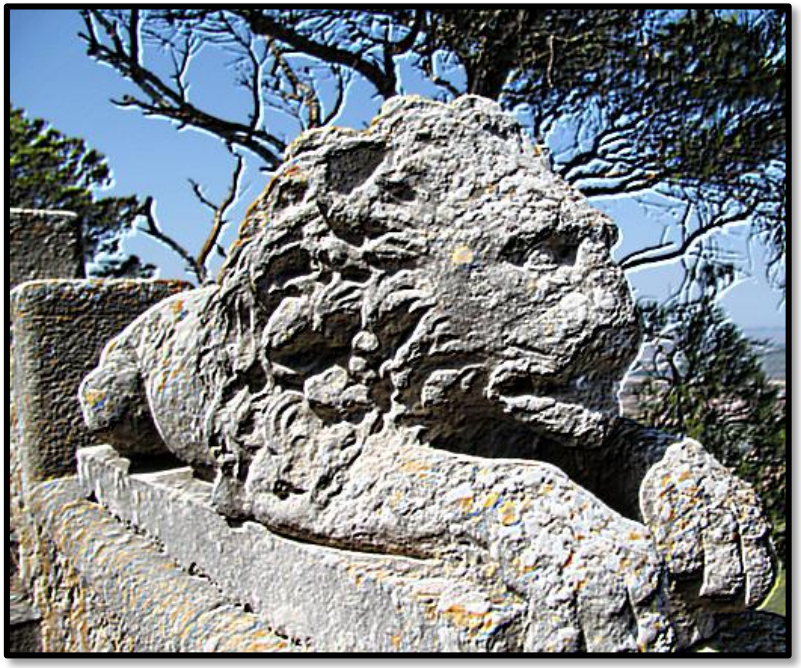
$*$

0	-1	0
-1	4	-1
0	-1	0

Filter: Emboss (Prägen/Stanzen)

*

-1	-1	0
-1	1	1
0	1	1



Morphologische Operationen

- strukturverändernde Operation (ein nicht-linearer Filter)
  - betrachte Menge der Nachbarpixel A (typ. in einem Binärbild)
  - Strukturelement B
  - Operation: setze oder lösche Pixel

0	1	0
1	1	1
0	1	0

B

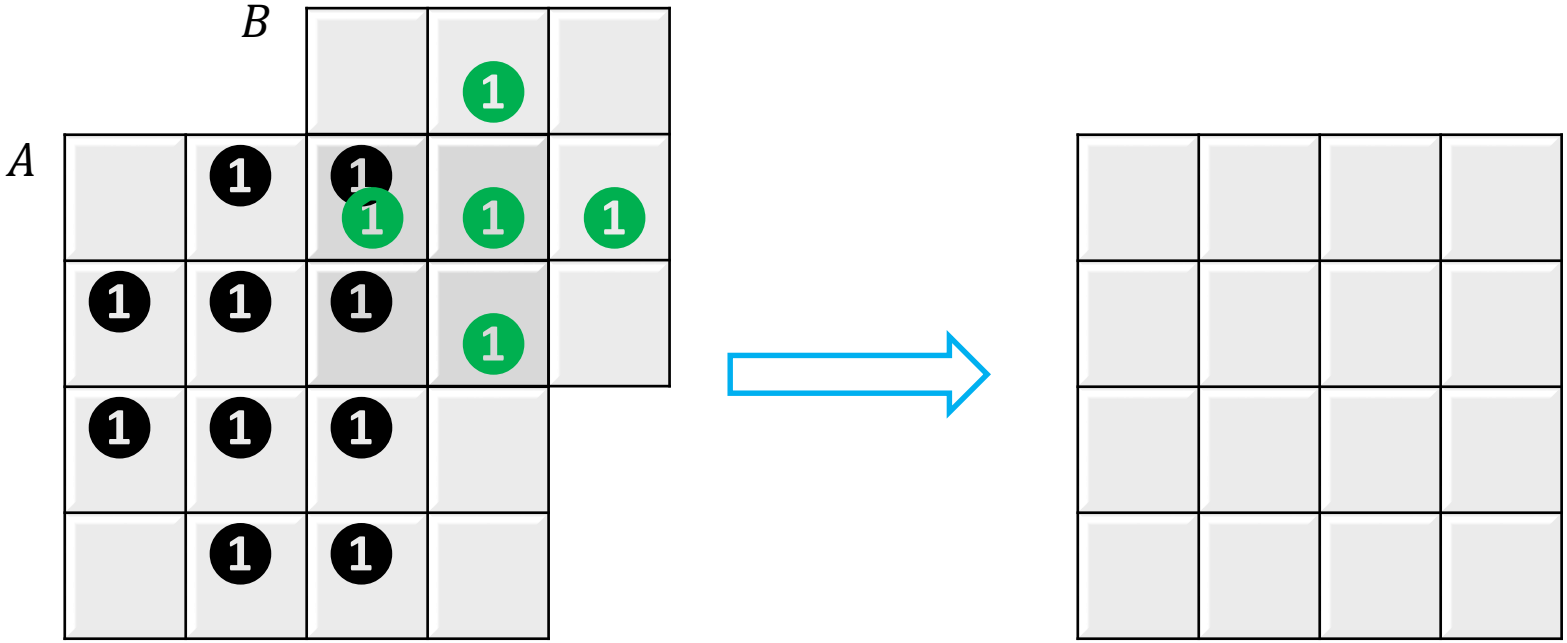
- **Dilatation** (Erweiterung)
  - Menge aller Pixel z, bei denen eine „1“ im Strukturelement mit mind. einer „1“ in A überlappt

- **Erosion** (Abtragung)
  - Menge aller Pixel z, bei denen alle „1“ im Strukturelement mit einer „1“ in A überlappen

Dilatation

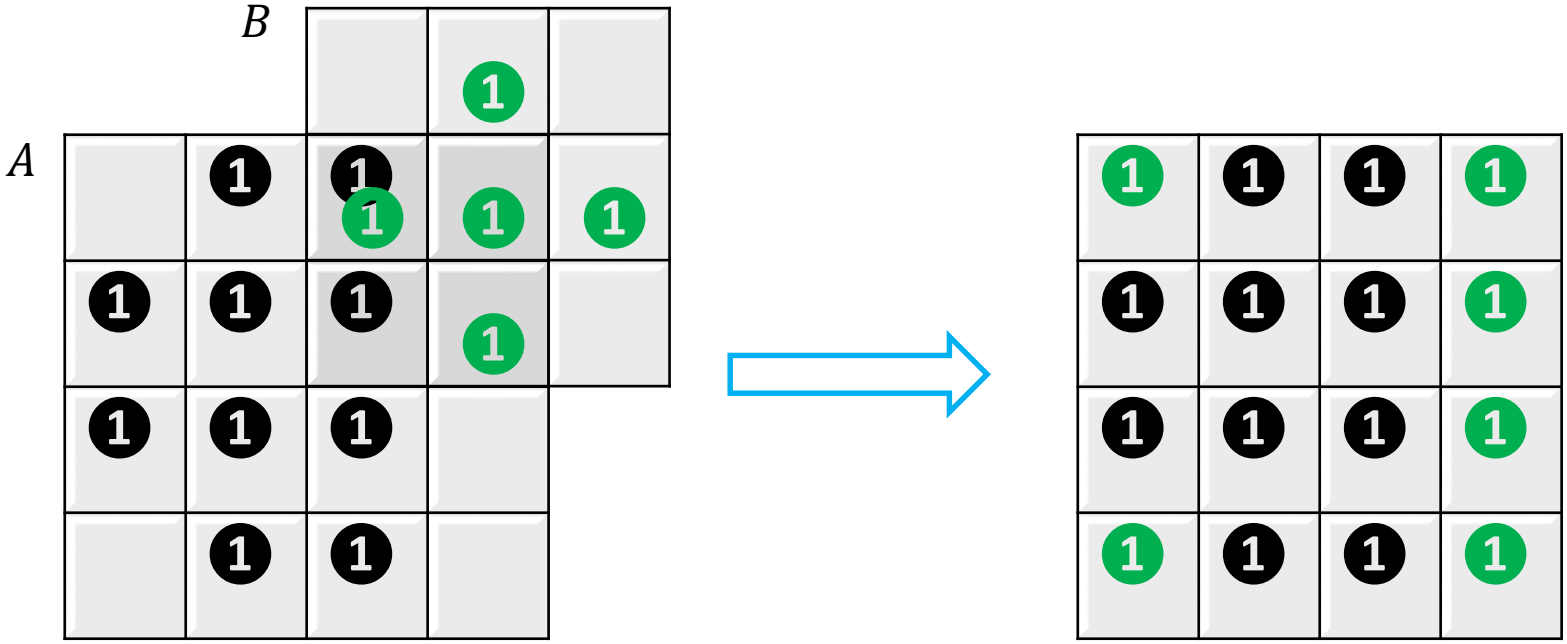
- Menge aller Pixel z, bei denen eine „1“ im Strukturelement mit mind. einer „1“ in A überlappt

- Frage: Welche Pixel sind nach der Dilatation „1“?



Dilatation

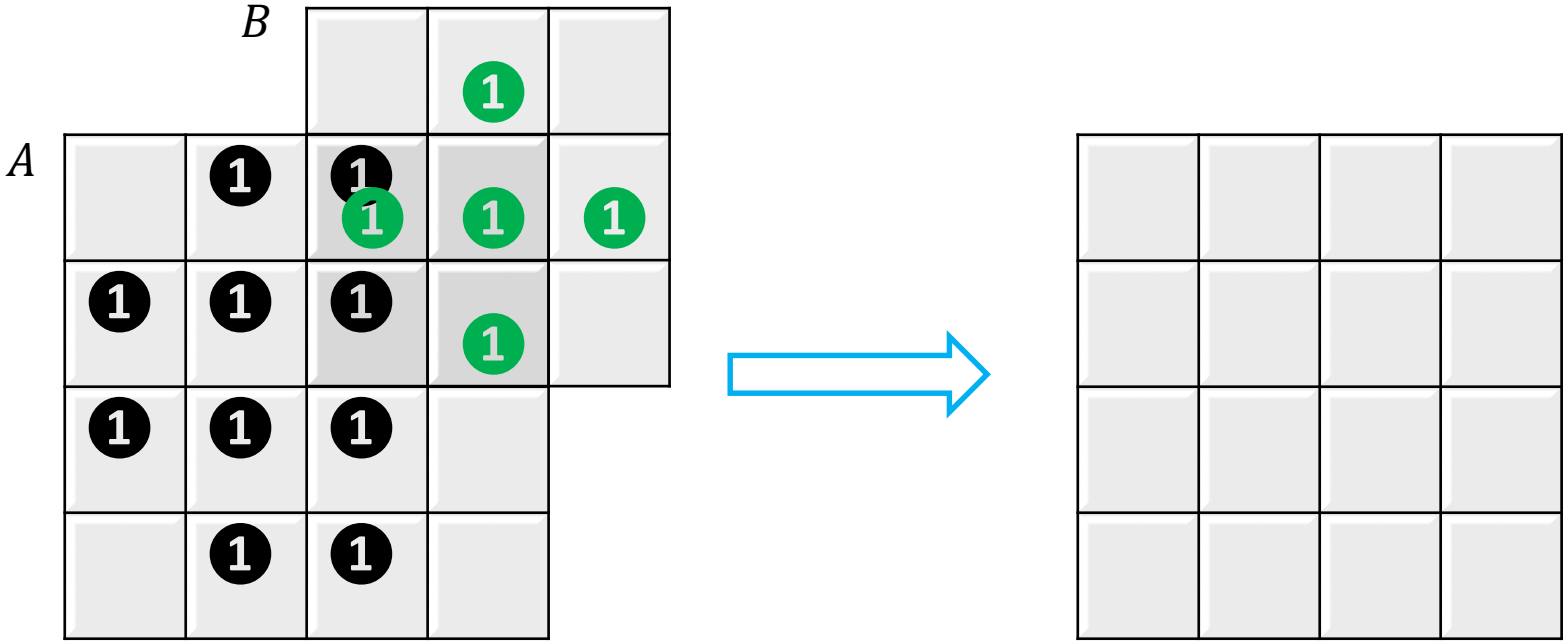
- Ergebnis:



Erosion

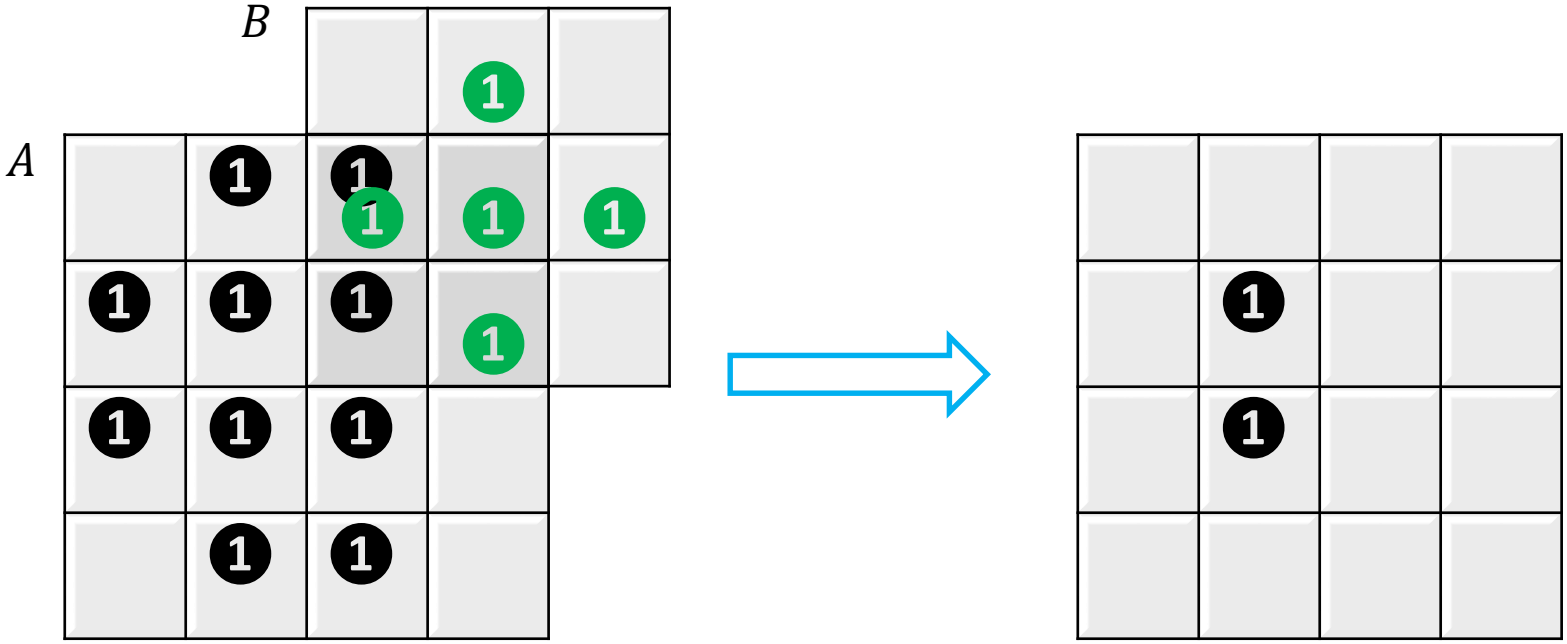
- Menge aller Pixel z, bei denen alle „1“ im Strukturelement mit einer „1“ in A überlappen

- Frage: Welche Pixel sind nach der Erosion „1“?

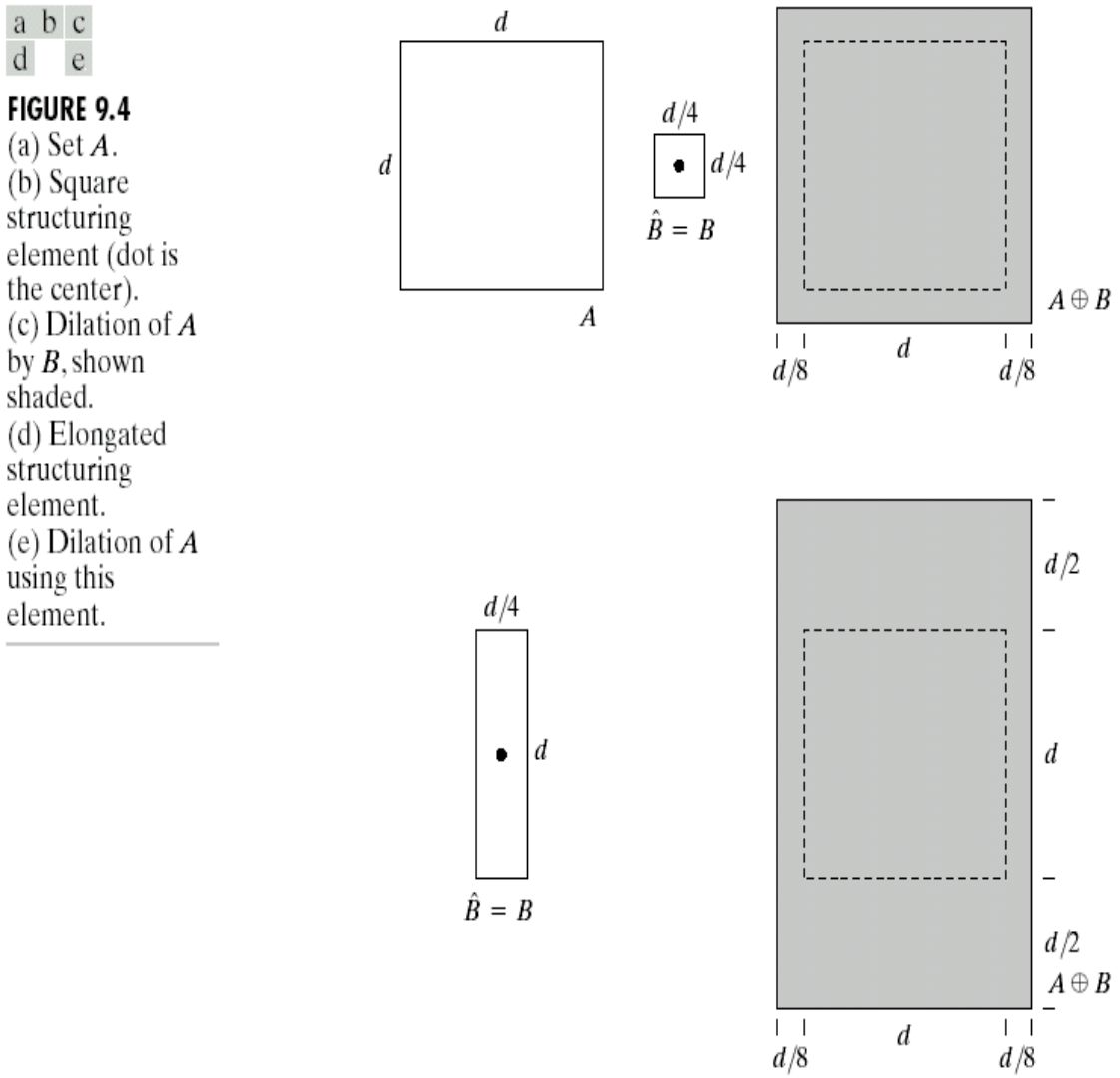


Erosion

- Ergebnis:



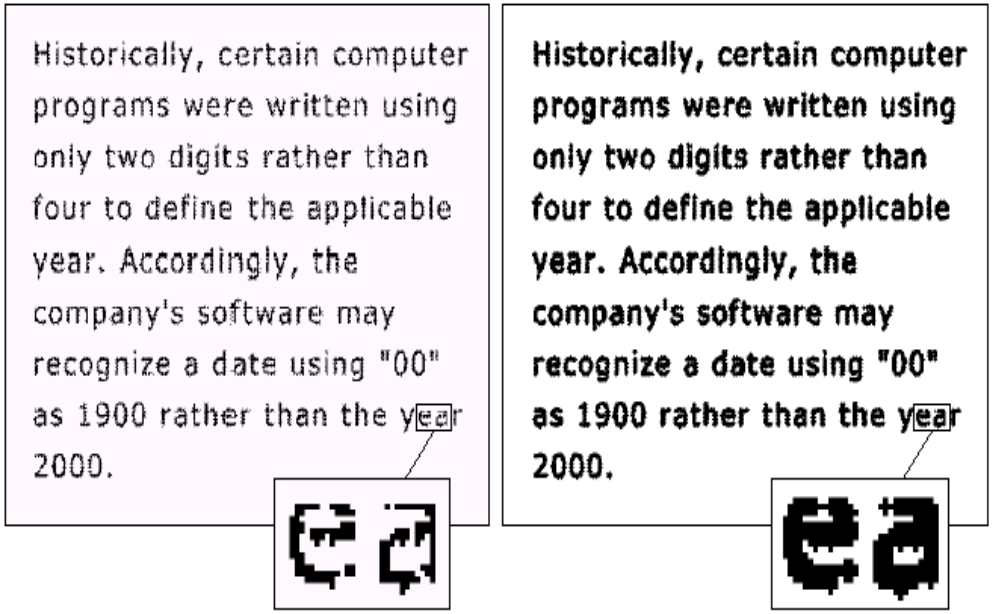
Dilatation (Erweiterung)



a b c
d e

FIGURE 9.4
(a) Set A.
(b) Square structuring element (dot is the center).
(c) Dilation of A by B, shown shaded.
(d) Elongated structuring element.
(e) Dilation of A using this element.

Dilatation (Erweiterung)

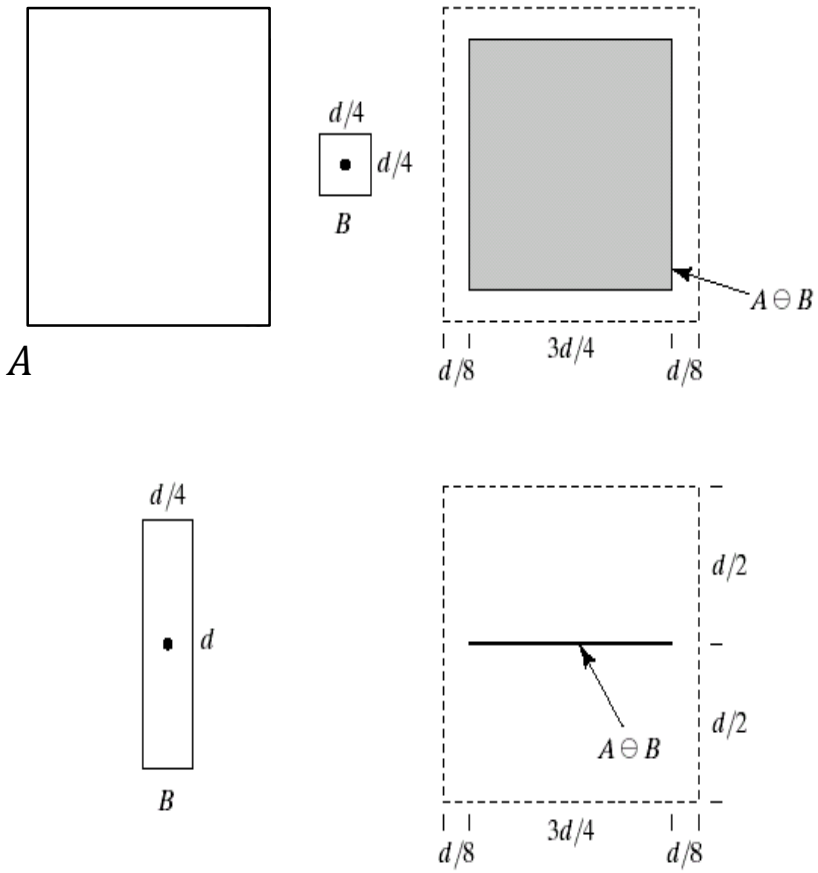


0	1	0
1	1	1
0	1	0

a c
b

FIGURE 9.5
(a) Sample text of poor resolution with broken characters (magnified view). (b) Structuring element. (c) Dilation of (a) by (b). Broken segments were joined.

Erosion



a b c
d e

FIGURE 9.6 (a) Set A. (b) Square structuring element. (c) Erosion of A by B, shown shaded. (d) Elongated structuring element. (e) Erosion of A using this element.

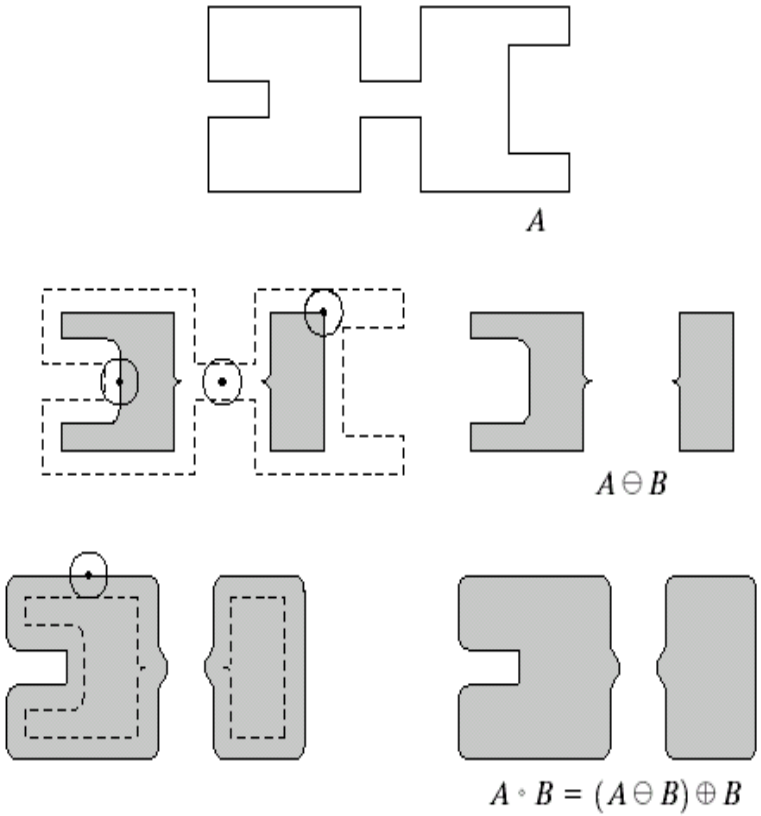
Erosion (Abtragung)

Öffnen und Schließen

- **Öffnen**: glättet die Kontur, entfernt Überstände und Brücken

$$A \odot B = (A \ominus B) \oplus B$$

- Erosion, gefolgt von Dilatation:

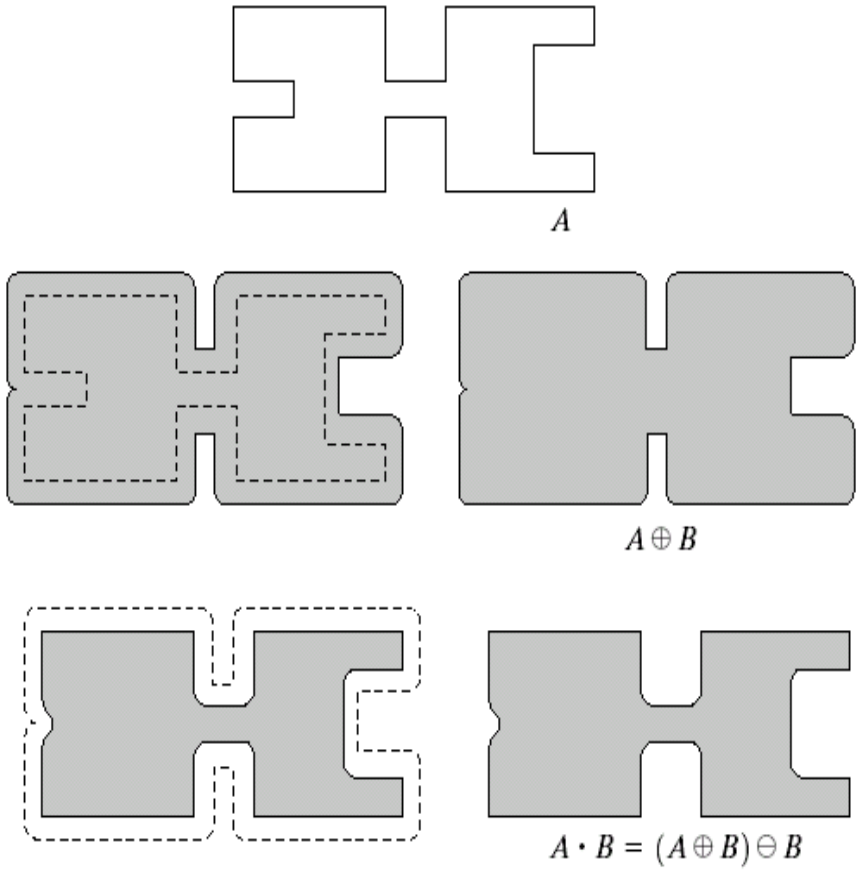


Öffnen und Schließen

- **Schließen**: glättet Teile der Kontur, schließt kleinere Lücken

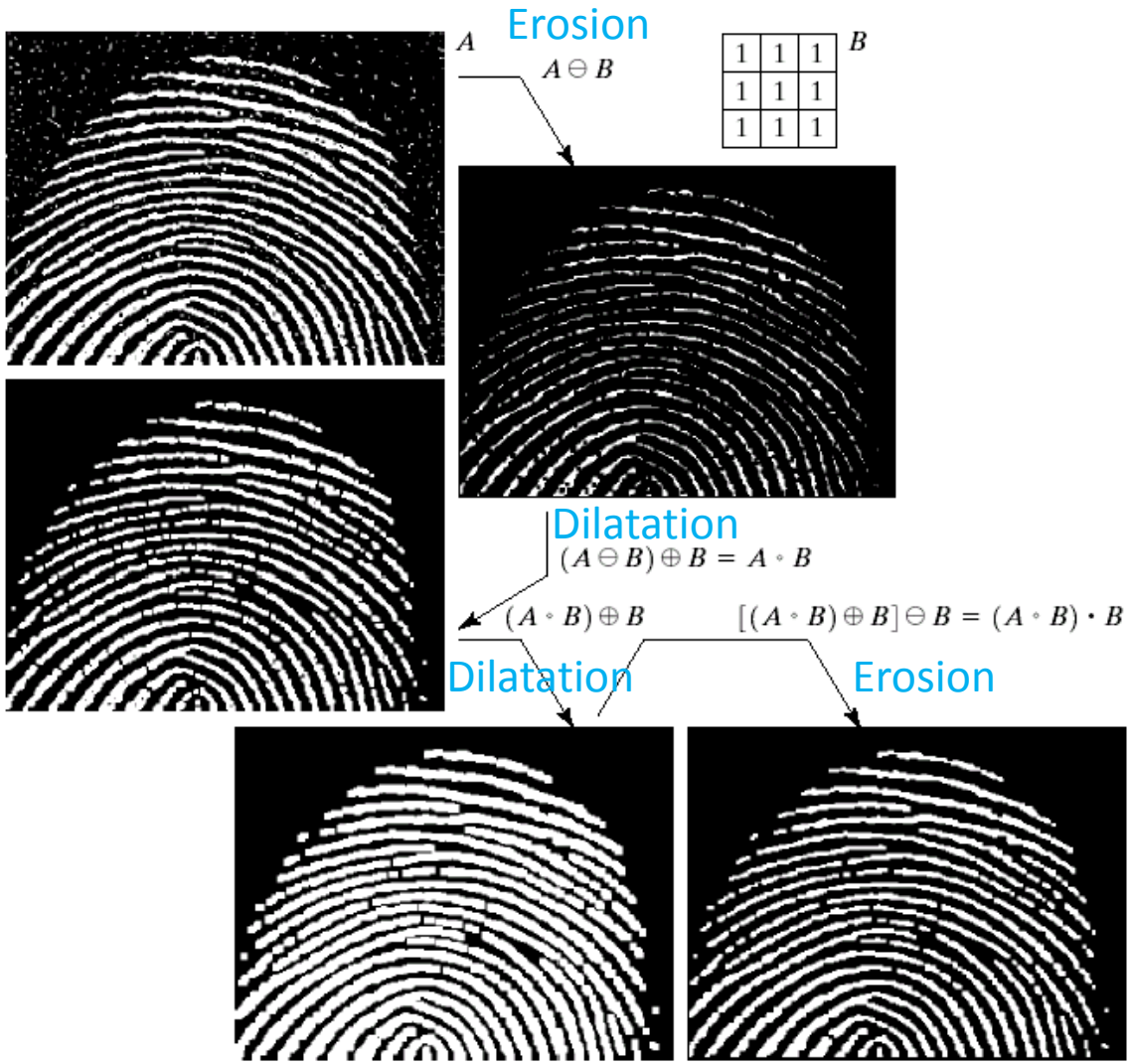
$$A \odot B = (A \oplus B) \ominus B$$

- Dilatation, gefolgt von Erosion



Öffnen und Schließen

- erst Öffnen (Erosion und Dilatation), dann Schließen (Dilatation und Erosion)



a b
d c
e f

FIGURE 9.11
(a) Noisy image. (c) Eroded image. (d) Opening of A. (d) Dilation of the opening. (e) Closing of the opening. (Original image for this example courtesy of the National Institute of Standards and Technology.)

Kontur-Extraktion

- Erosion, gefolgt von einer Subtraktion

$$\beta(A) = A - (A \ominus B)$$

a b
c d

FIGURE 9.13 (a) Set A. (b) Structuring element B. (c) A eroded by B. (d) Boundary, given by the set difference between A and its erosion.