Clipping

$\longrightarrow$ ebenfalls grundlegende Aufgabe der Computergraphik

$\longrightarrow$ effiziente Algorithmen notwendig (evtl. Hardware)

Generelle Verfahrensmöglichkeiten:

a) analytisch
   Beispiel: parametrisch gegebene Linie:

$$y = mx + b \qquad x \in [x_0, x_1]$$

   nach Clipping: $x \in [a, b], a \geq x_0, b \leq y_0$

b) bei der Rasterkonvertierung (Scissoring)

Beispiel:

$$
\begin{aligned}
&void\ WritePixel(int\ x,\ int\ y)\\
&\{\\
&\quad if\ ((x_0 \leq x \leq x_1)\ \&\&\ (y_0 \leq y \leq y_1))\\
&\qquad screen[x][y] = color;\\
&\}
\end{aligned}
$$

c) Clipping durch Kopieren

1. Rasterkonvertierung in großes Fenster
2. Bildfenster durch Kopieren erzeugen

Anwendung: Spiele, Kameraschwenk über statische Szene

im weiteren Verlauf des Kapitels:

Betrachtung analytischer Verfahren

Voraussetzung:

- achsenparalleles Rechteck beschreibt Clipping-Region $((x_{min}, y_{min}), (x_{max}, y_{max}))$
- ist das Rechteck allgemein orientiert, kann durch Drehen der Primitive der achsenparallele Fall erzeugt werden

Algorithmus von Cohen & Sutherland

Den beiden Endpunkten p, q einer Strecke ihre Lage bzgl. des Fensters als Bitcode zugeordnet:

- "links" $\longrightarrow$ 0001
- "rechts" $\longrightarrow$ 0010
- "oben" $\longrightarrow$ 1000
- "unten" $\longrightarrow$ 0100

$\Rightarrow$ jeder der neun Bereiche ist durch zwei dieser Mengen gekennzeichnet, die im ersten und zweiten Bit-Paar abgelegt sind

(Punkt links oben: Code 1001)

links oben	oben	rechts oben
links	Clipping-bereich	rechts
links unten	unten	rechts unten

1001	1000	1010
0001	0000	0010
0101	0100	0110

Schnelltest:

Code$(p) \cup$ Code $(q) = \emptyset$,

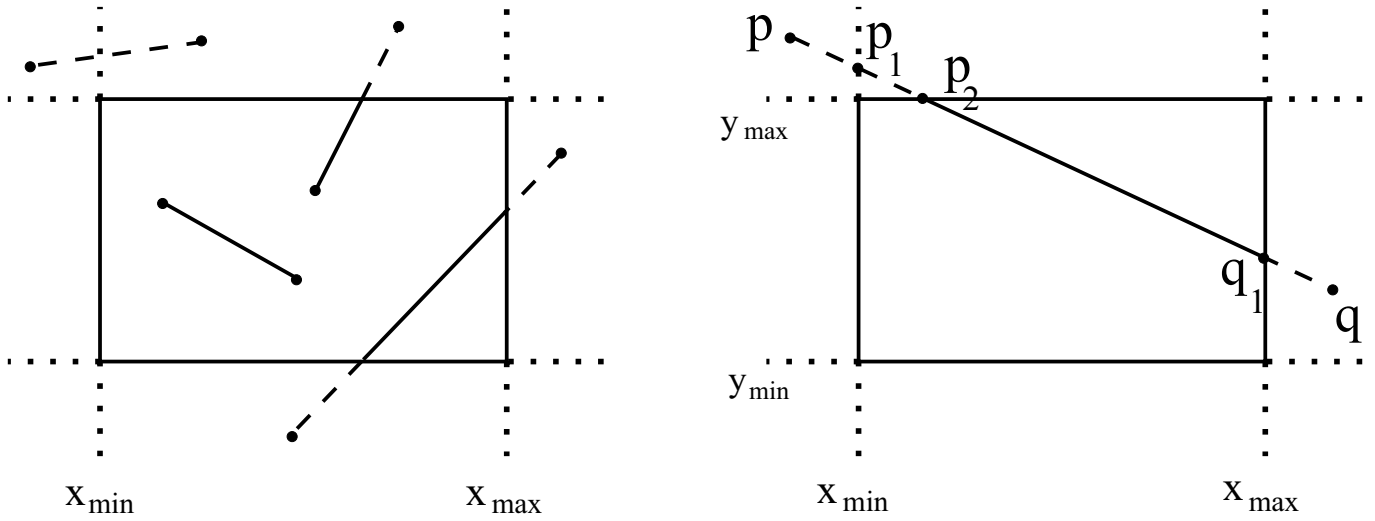
$\Rightarrow$ Strecke vollständig sichtbar

Code$(p) \cap$ Code $(q) \neq \emptyset$,

$\Rightarrow$ Strecke vollständig unsichtbar

Treffen obige Bedingungen nicht zu, dann:

1. Nimm den Endpunkt, dem eine nichtleere Menge zugeordnet ist
2. Ersetze ihn durch Schnittpunkt mit der entsprechenden Begrenzungsgeraden und passe Bereichscode an
3. Wiederhole Algorithmus mit der so gekürzten Strecke



Algorithmus von Cyrus & Beck

Idee:

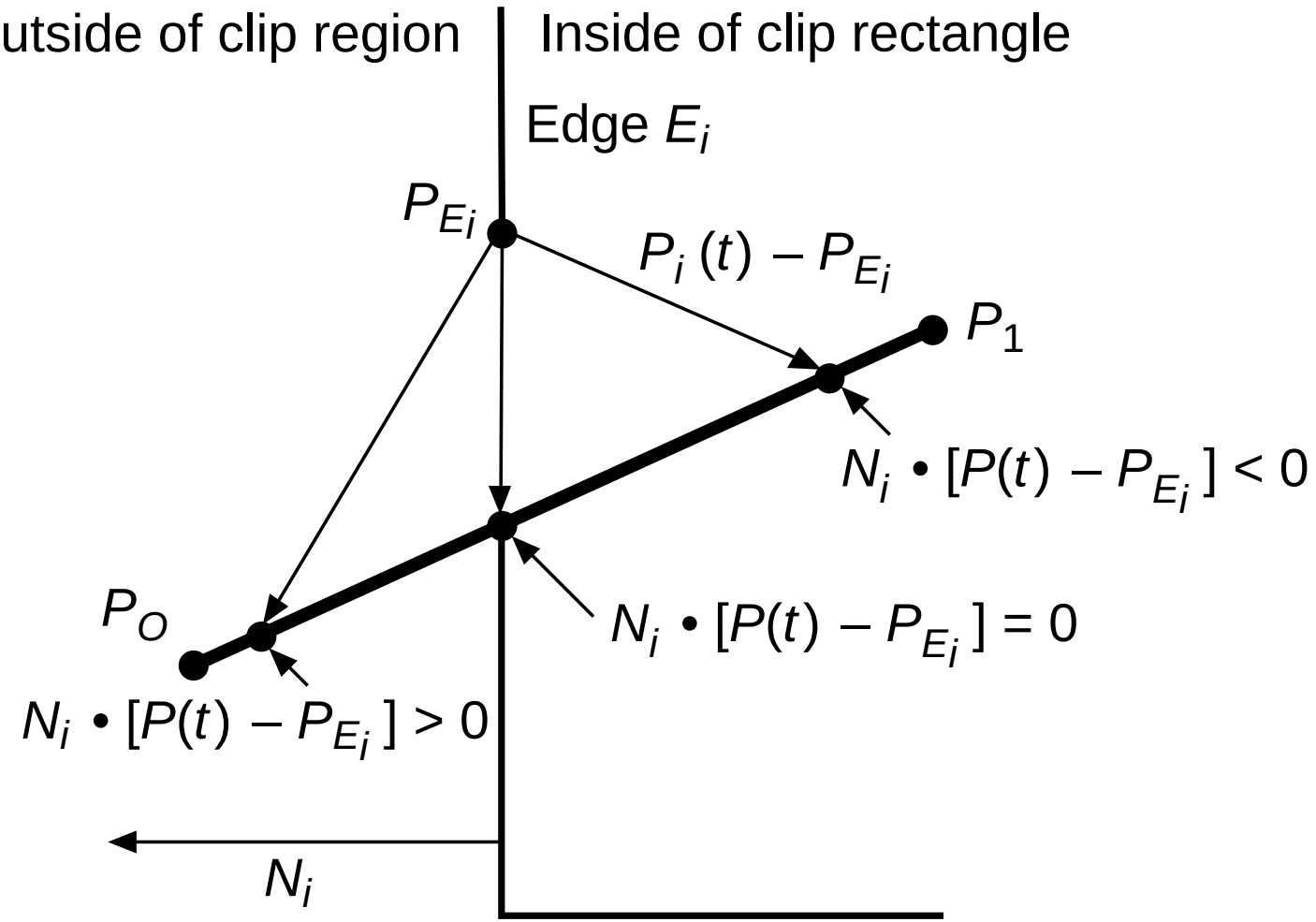
Berechne Schnittpunkte von Strecken mit Kanten in Parameterdarstellung

Strecke: $P(t) = P_0 + t\,(P_1 - P_0)$

$\rightarrow$ Finde t_i für Schnittpunkte mit Kanten

$\rightarrow$ Benutze Skalarprodukt

$$N \cdot (P(t) - P_E) = \begin{cases} < 0 & innen \\ = 0 & Rand \\ > 0 & außen \end{cases}$$



aus: Foley et al.: Computer graphics, principles and practice

$$\begin{aligned} 0 &= N \cdot (P(t) - P_E) = 0 \\ &= N \cdot (P_0 + t\,(P_1 - P_0) - P_E) \\ &= N \cdot (P_0 - P_E) + N \cdot (P_1 - P_0)\,t \end{aligned}$$

Umformen und Auflösen nach t:

$$t = -\frac{N \cdot (P_0 - P_E)}{N \cdot (P_1 - P_0)} = -\frac{N \cdot (P_0 - P_E)}{N \cdot D}$$

$\rightarrow$ ist $N \cdot D = 0$, so sind Kante und Strecke parallel

Wie findet man die "richtigen" Werte von t heraus ?

1. Ordne jeder Kante eine Halbebene zu ($\rightarrow$ innen)
2. Markiere Schnittpunkte:

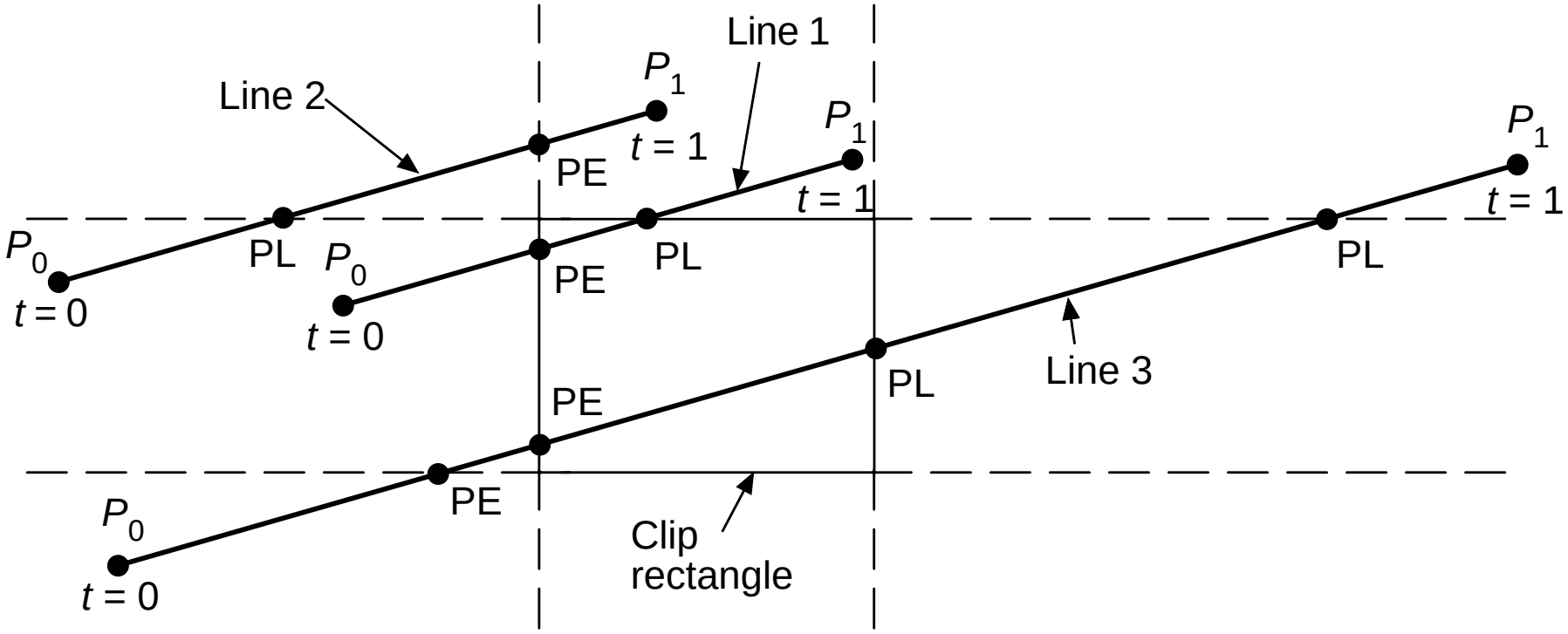
   PL: auf Weg von P_0 nach P_1 wurde Halbebene verlassen
   PE: eine Halbebene wurde betreten

   Berechnung:

   $N \cdot D > 0 \quad \rightarrow \quad$ PL $\quad$ (Winkel $< 90°$)
   $N \cdot D < 0 \quad \rightarrow \quad$ PE $\quad$ (Winkel $> 90°$)

3. Zu zeichnende Strecke ist dann

$$t \in [max(t_{PE}), min(t_{PL})]$$



aus: Foley et al.: Computer graphics, principles and practice

Vereinfachung für achsenparallele Rechtecke:

$$t = -\frac{N\,(P_0 - P_E)}{N \cdot D} = \begin{cases} -\dfrac{x_0 - x_{min}}{x_1 - x_0} & \text{links} \\ -\dfrac{x_0 - x_{max}}{x_1 - x_0} & \text{rechts} \\ -\dfrac{y_0 - y_{min}}{y_1 - y_0} & \text{unten} \\ -\dfrac{y_0 - x_{max}}{y_1 - y_0} & \text{oben} \end{cases}$$

Algorithmus:

begin

Berechne N_i und wähle P_{EI} für jede Kante;

for *jede zu clippende Linie* do

if $P_1 = P_0$ then

Linie ist degeneriert und als Punkt zu clippen

else begin

$t_E := 0; \quad t_L = 1;$

for *jeden möglichen Schnittpunkt mit einer Clipkante* do

if $N_i \cdot D <> 0$ then begin (*wenn nicht kantenparallel*)

berechne t;

nutze Vorzeichen zur Markierung als PE/PL;

if PE then $t_E = \max(t_E, t)$;

if PL then $t_L = \min(t_L, t)$;

end

if $t_E > t_L$ then return nil;

else *return P(t_E) und P(t_L) als gültige Schnittpunkte;*

end

end

Clipping von Polygonen an Rechtecken

Erste Möglichkeit:

→ sequentielles Clippen aller Polygonkanten am Rechteck

→ Problem: nicht zusammenhängende Teile

Zweite Möglichkeit:

→ Sutherland-Hodgman-Algorithmus
(Aufwand: $O(n)$ bei n Polygonkanten)

→ sequentielles Abschneiden der nicht sichtbaren Teile des Polygons am Rechteck

$\longrightarrow$ Resultat:

zusammenhängendes, jedoch nicht einfaches Polygon

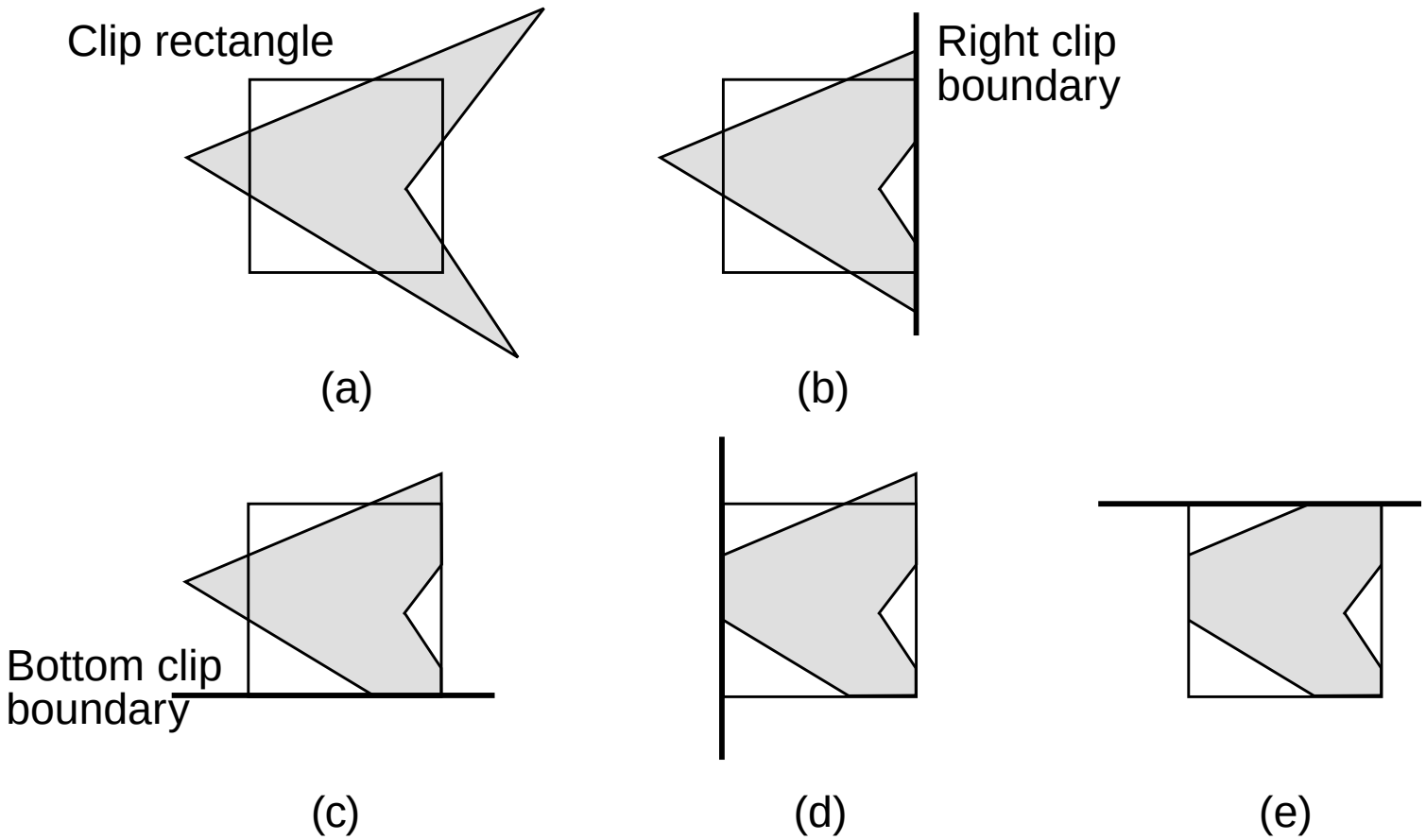
Idee des Sutherland-Hodgman-Algorithmus:

Algorithmus operiert auf Punktliste des Polygons

Operationen auf Punktliste:

1.) nicht sichtbare (außerhalb des Rechtecks liegende) Punkte werden entfernt

2.) Schnittpunkte mit Rechteckkanten werden eingefügt



aus: Foley et al.: Computer graphics, principles and practice

Sutherland-Hodgman-Algorithmus

begin

result = 0; // *(Ergebnis-Polygon leer initialisieren)*

p = *erster Punkt aus* P;

while p *existiert* do begin

if (sichtbar(p,k)) then

result = result $\cup$ {p};

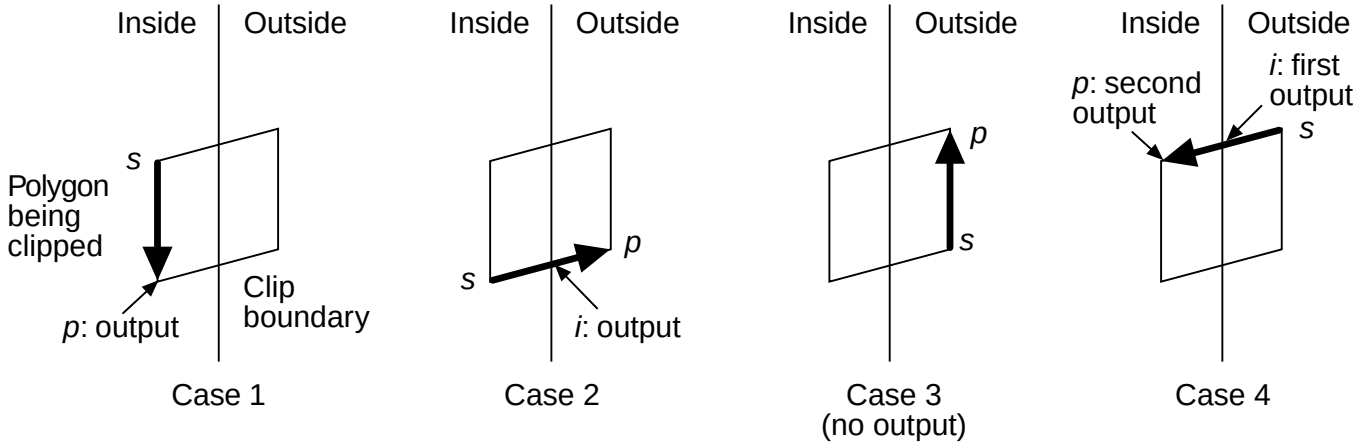
if (schneidet($\overline{psucc(p)}$,k)) then

result = result $\cup$ {schnittpunkt($\overline{psucc(p)}$,k)};

p = succ(p);

end

end



aus: Foley et al.: Computer graphics, principles and practice

Fall	**Beschreibung**
1	Die Polygonkante sp liegt ganz auf der Innenseite der Cliplinie $\Rightarrow$ p zur Liste hinzufügen
2	Die Polygonkante schneidet die Cliplinie von innen nach außen $\Rightarrow$ Schnittpunkt i hinzufügen
3	Die Polygonkante liegt ganz auf der Außenseite der Cliplinie $\Rightarrow$ nichts hinzufügen
4	Die Polygonkante schneidet die Cliplinie von außen nach innen $\Rightarrow$ p und Schnittpunkt i hinzufügen