

§1 Hardwaregrundlagen

§2 Transformationen und Projektionen

2.4 Projektionen

2.5 Perspektivische Projektionen

2.6 Parallele Projektionen

2.7 Umsetzung der Zentralprojektion

2.8 Weitere Projektionen

2.9 Koordinatensysteme, Frts.

2.10 Window to Viewport

2.11 Clipping

§3 Repräsentation und Modellierung
von Objekten

§4 Rasterung

§5 Visibilität und Verdeckung

§6 Rendering

§7 Abbildungsverfahren (Texturen, etc.)

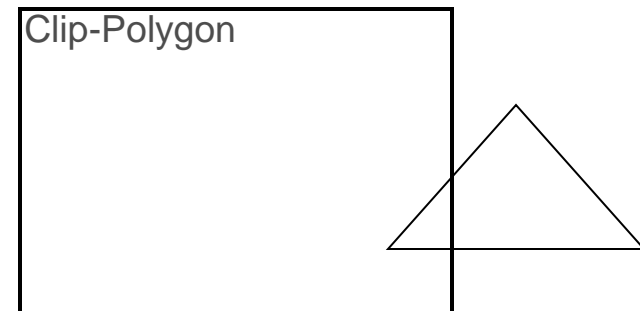
§8 Freiformmodellierung

Anhang: Graphiksprachen und Graphikstandards

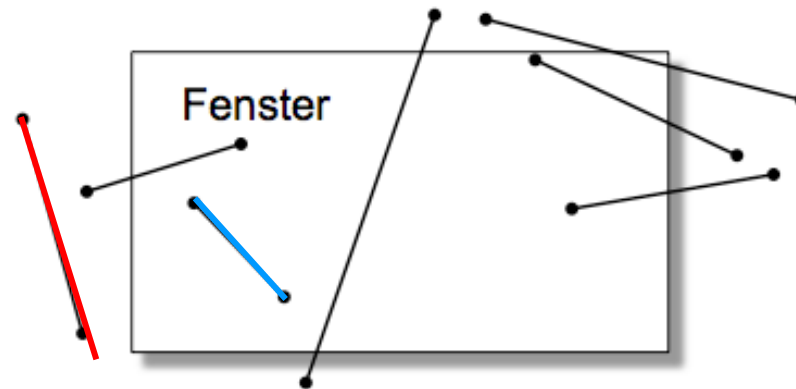
Anhang: Einführung in OpenGL

Weitere Themen: Netze, Fraktale, Animation, ...

- Objekte in der Bildebene werden **innerhalb** eines Fensters dargestellt.
- Alle außerhalb des Fensters liegenden Objektteile müssen **abgeschnitten** werden (**Clipping** am Fensterrand).
- Dieses Fenster wird **Clip-Polygon** genannt.
- Clip-Polygone sind **typischerweise Rechtecke**.
- Können auch **andere Geometrie** haben
- Nichtkonvexe oder nicht einfache Polygone sind **problematisch**

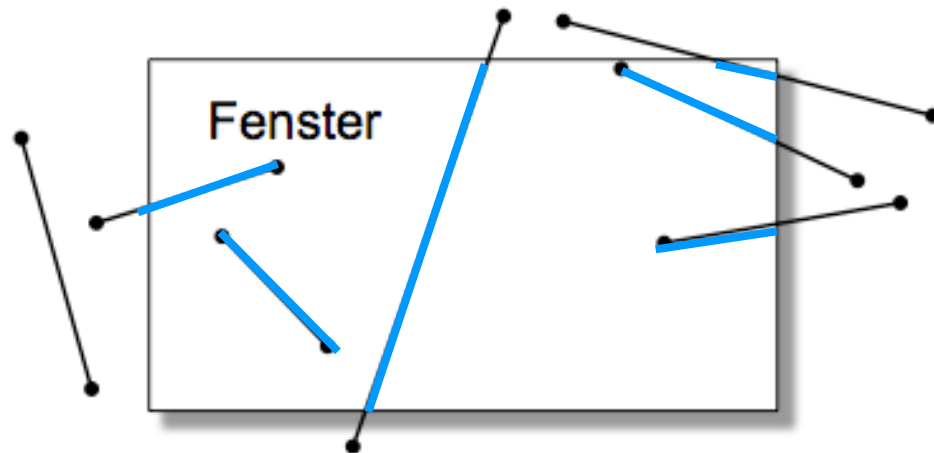


- An einem **rechteckigen, achsenparallelen** Fenster
- Offensichtlich gibt es **3 Fälle**, von denen zwei trivial sind:
 - beide Endpunkte **innerhalb des Fensters** \Rightarrow Linie zeichnen
 - beide Endpunkte **oberhalb oder unterhalb oder links oder rechts des Fensters** \Rightarrow Linie nicht zeichnen



Drei Fälle (Frts)

- Sonst:
 - Die **Schnittpunkte der Linie** mit dem Fensterrand anhand der Geradengleichungen berechnet
 - Daraus die **sichtbare Strecke** bestimmen



Liang-Barsky-Algorithmus

- Fensterkanten als **implizite Gerade**

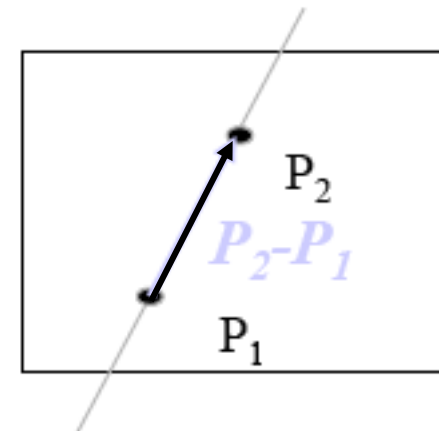
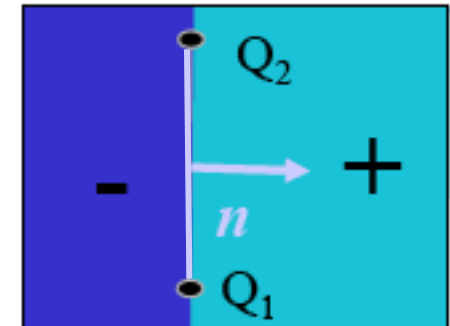
$$Q_1 = (x_1, y_1), Q_2 = (x_2, y_2)$$

$$n = (\Delta y, -\Delta x) = (y_2 - y_1, -(x_2 - x_1))$$

$$P = (x, y)$$

$$E(P) = n \cdot (P - Q_1) = n \cdot P - n \cdot Q_1$$

- Normale n zeigt ins **Innere**
- Liniensegmente **parametrisch** darstellen: $I(t) = P_1 + t \cdot (P_2 - P_1)$



Liang-Barsky-Algorithmus

Fallunterscheidung:

(1) P_1 und P_2 liegen außen

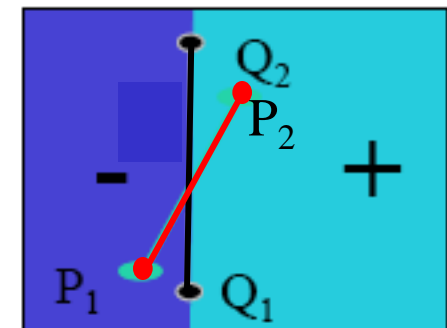
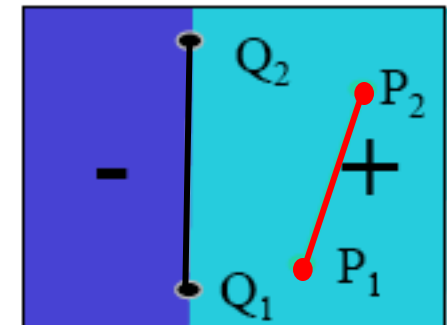
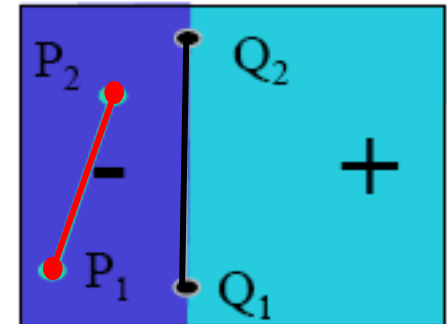
$$E(P_1) \leq 0, E(P_2) \leq 0$$

(2) P_1 und P_2 liegen innen

$$E(P_1) \geq 0, E(P_2) \geq 0$$

(3) P_1 und P_2 liegen auf verschiedenen Seiten

$$E(P_1) < 0, E(P_2) > 0, \text{ bzw. } E(P_1) > 0, E(P_2) < 0$$



Liang-Barsky-Algorithmus

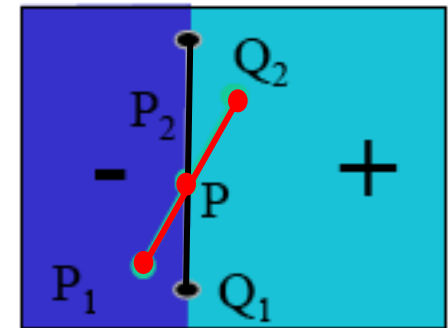
- Fall 3: Schnittpunkt P muss berechnet werden
- Parametrische Gleichung in implizite Gleichung einsetzen:

$$E(P_1 + t \cdot (P_2 - P_1)) = 0$$

$$\Leftrightarrow (P_1 + t \cdot (P_2 - P_1)) \bullet n - Q_1 \bullet n = 0$$

$$\Leftrightarrow t = \frac{(Q_1 \bullet n - P_1 \bullet n)}{(P_2 - P_1) \bullet n}$$

$$\Leftrightarrow P = P_1 + \frac{(Q_1 \bullet n - P_1 \bullet n)}{(P_2 - P_1) \bullet n} \cdot (P_2 - P_1)$$



Cohen-Sutherland Line-Clipping Algorithmus

- Kern des Algorithmus ist ein schnelles Verfahren zur **Bestimmung der Kategorie** einer Linie (innerhalb, außerhalb, schneidend).
- Es sei ein Fenster (xmin, ymin, xmax, ymax) gegeben, dessen begrenzende Geraden (Halbebenen) die Bildebene in **neun Regionen** unterteilen.
- Jeder Region ist ein eindeutiger 4-Bit-Code (**Outcode**) zugeordnet, der Auskunft über deren Lage in Bezug auf das Fenster gibt
- Im **3D** sind es 27 Regionen (3^3) und ein **6-Bit-Outcode**

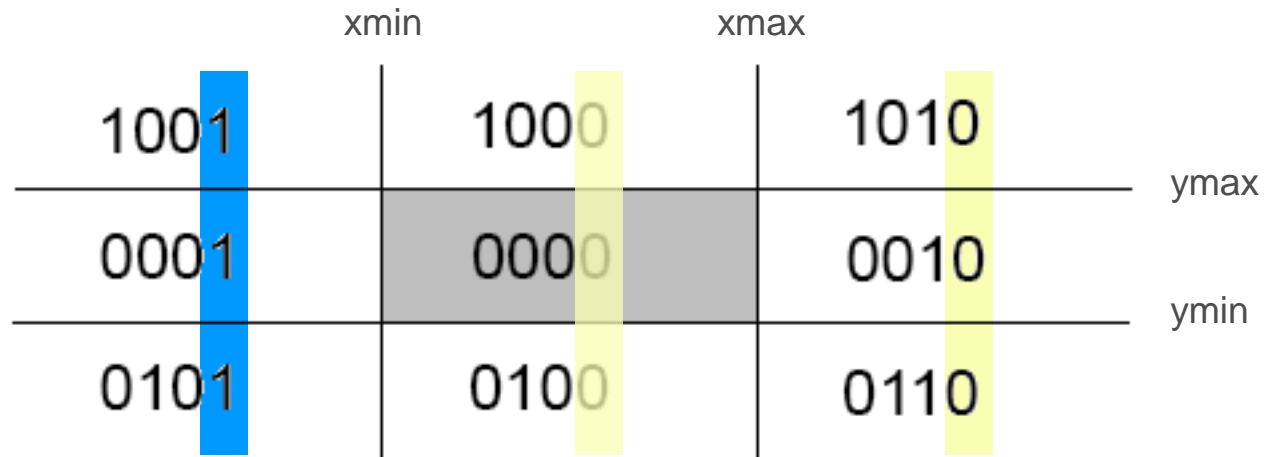
Cohen-Sutherland Line-Clipping Algorithmus

- **Outcode-Bits gesetzt**, falls Eckpunkt in Region liegt
 - Bit 0: ... links des Fensters $x < x_{min}$
 - Bit 1: ... rechts des Fensters $x > x_{max}$
 - Bit 2: ... unterhalb des Fensters $y < y_{min}$
 - Bit 3: ... oberhalb des Fensters $y > y_{max}$

	xmin	xmax	
1001	1000	1010	
0001	0000	0010	y _{max}
0101	0100	0110	y _{min}

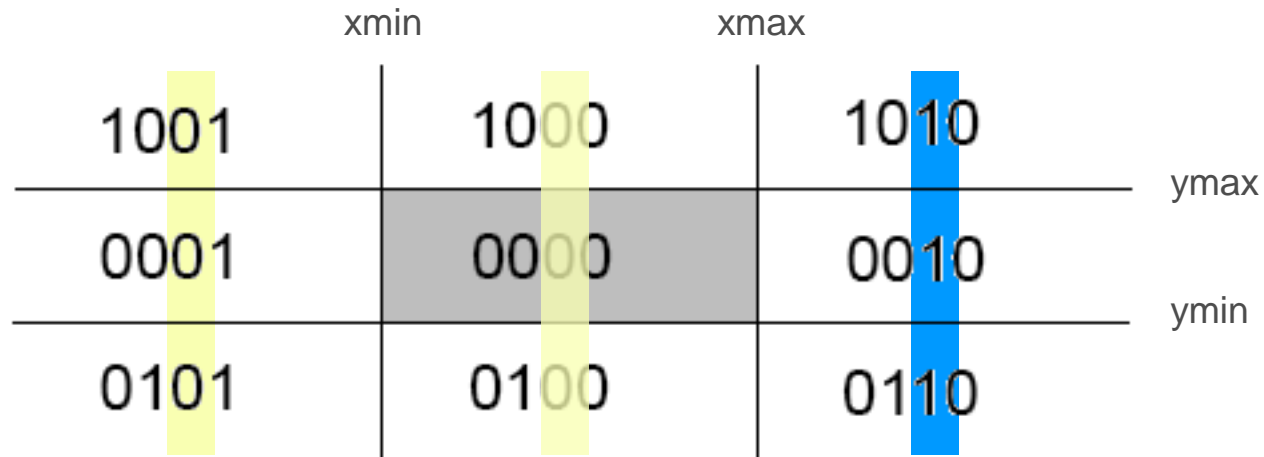
Cohen-Sutherland Line-Clipping Algorithmus

- **Outcode-Bits gesetzt**, falls Eckpunkt in Region liegt
 - Bit 0: ... links des Fensters $x < x_{min}$
 - Bit 1: ... rechts des Fensters $x > x_{max}$
 - Bit 2: ... unterhalb des Fensters $y < y_{min}$
 - Bit 3: ... oberhalb des Fensters $y > y_{max}$



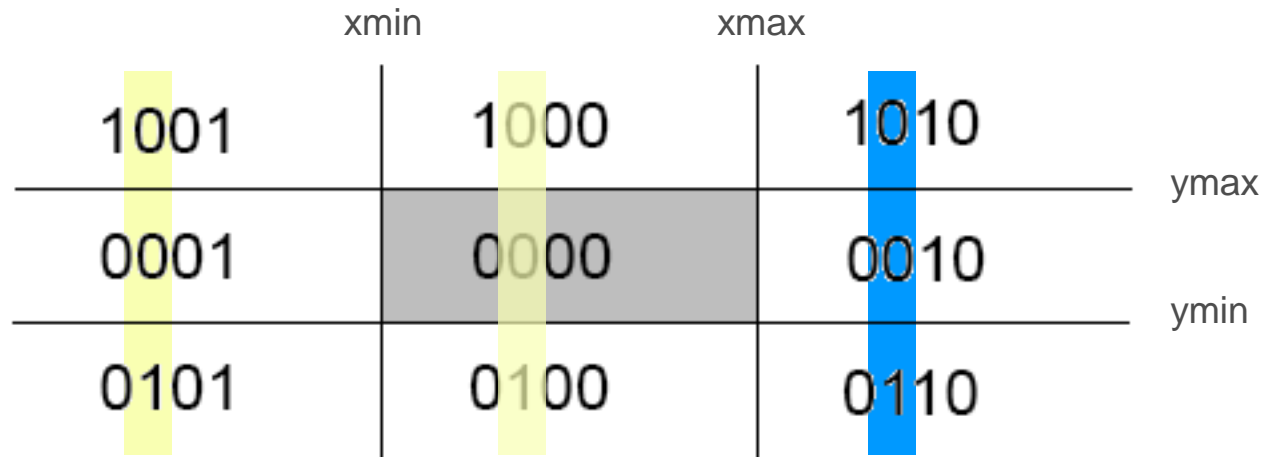
Cohen-Sutherland Line-Clipping Algorithmus

- **Outcode-Bits gesetzt**, falls Eckpunkt in Region liegt
 - Bit 0: ... links des Fensters $x < x_{min}$
 - Bit 1: ... rechts des Fensters $x > x_{max}$
 - Bit 2: ... unterhalb des Fensters $y < y_{min}$
 - Bit 3: ... oberhalb des Fensters $y > y_{max}$



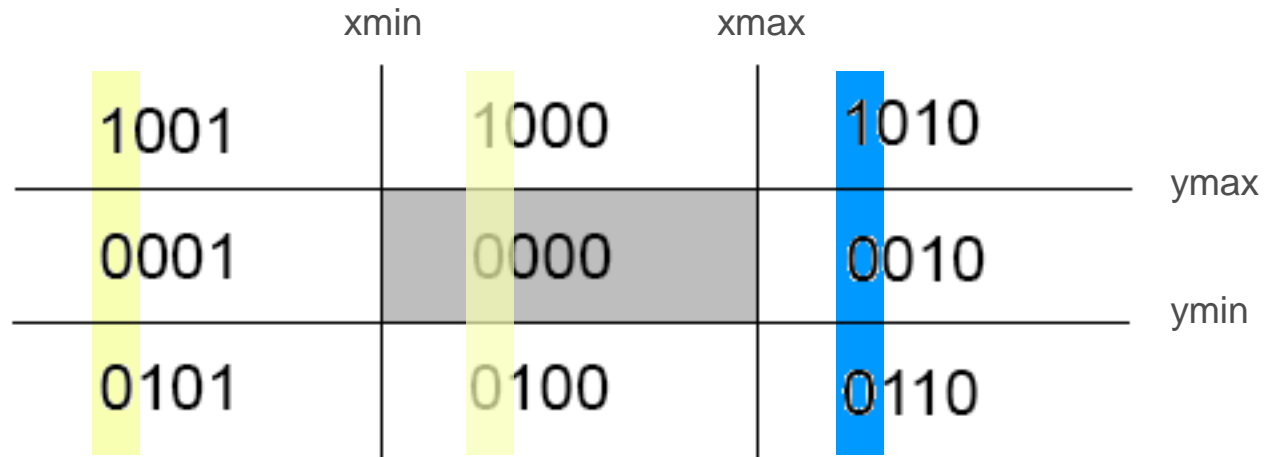
Cohen-Sutherland Line-Clipping Algorithmus

- **Outcode-Bits gesetzt**, falls Eckpunkt in Region liegt
 - Bit 0: ... links des Fensters $x < x_{min}$
 - Bit 1: ... rechts des Fensters $x > x_{max}$
 - **Bit 2: ... unterhalb des Fensters** $y < y_{min}$
 - Bit 3: ... oberhalb des Fensters $y > y_{max}$



Cohen-Sutherland Line-Clipping Algorithmus

- **Outcode-Bits gesetzt**, falls Eckpunkt in Region liegt
 - Bit 0: ... links des Fensters $x < x_{min}$
 - Bit 1: ... rechts des Fensters $x > x_{max}$
 - Bit 2: ... unterhalb des Fensters $y < y_{min}$
 - **Bit 3: ... oberhalb des Fensters** $y > y_{max}$



Cohen-Sutherland Line-Clipping Algorithmus

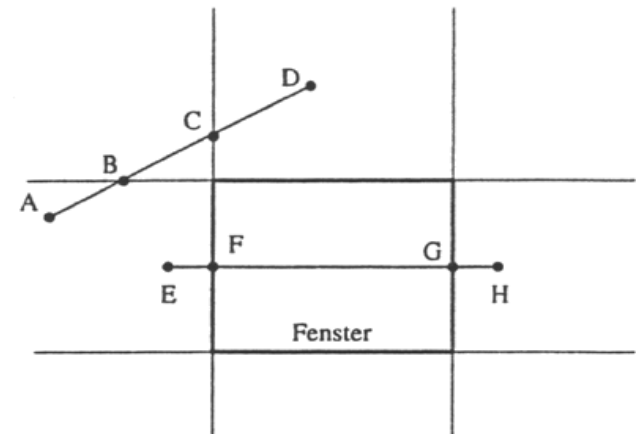
- Bestimmt Outcodes für Endpunkte einer Linie. Dann gilt:
 - Linie liegt vollständig außerhalb des Fensters, falls der **Durchschnitt (AND-Verknüpfung) der Codes beider Endpunkte** von Null verschieden ist: **Trivial Reject**
 - Linie liegt komplett im Fenster, wenn **beide Endpunkte den 4-Bit-Code 0000** besitzen (OR-Verknüpfung ist Null): **Trivial Accept**
 - Sonst:
 - Linie nacheinander mit den das Fenster begrenzenden Geraden schneiden und jeweils in zwei Teile zerlegen
 - Außerhalb des Fensters liegende Teil kann sofort entfernt werden

Cohen-Sutherland Line-Clipping Algorithmus

Beispiele nichttrivialer Fälle

- Linie AD: Codes 0001 und 1000 \Rightarrow Schnittberechnung
Schnitt mit linker Fenstergrenze liefert C \Rightarrow **eliminiere AC**
Punkte C und D liegen oberhalb des Fensters \Rightarrow **eliminiere CD**
- Linie EH: Codes 0001 und 0010 \Rightarrow Schnittberechnung
Schnitt mit linker Fenstergrenze liefert F \Rightarrow **eliminiere EF**.
Für FH ist eine Schnittberechnung mit der rechten Fenstergrenze notwendig, die den Punkt G liefert \Rightarrow **eliminiere GH**

Punkte **F und G** liegen innerhalb des Fensters \Rightarrow FG wird gezeichnet



Cohen-Sutherland Line-Clipping Algorithmus

Spezialfälle und Beschleunigungen

- Bei **senkrecht oder waagrecht** verlaufenden Linien muss nur gegen die y- bzw. x-Grenzen getestet und geschnitten werden.
- Falls **genau ein Endpunkt** innerhalb des Fensters liegt, gibt es nur einen Schnitt mit dem Fensterrand.
- Einige Schnittoperationen führen nicht zu Schnittpunkten am Fensterrand
 - Jedes Bit korrespondiert genau zu einem der Fensterränder.
 - Betrachte nur Fensterränder deren zugehöriges Bit in den zwei Endpunkt-Codes **unterschiedlich gesetzt** ist.

Cohen-Sutherland Line-Clipping Algorithmus

Spezialfälle und Beschleunigungen

- Vermeidung der aufwändigen Schnittpunktberechnung durch Bisektionsmethode:
 - Linien, die weder ganz außerhalb, noch ganz innerhalb des Fensters liegen, werden **so lange unterteilt**, bis ihre Länge kleiner als ein Pixel ist.
 - Bei $2^{10}=1024$ Pixel in einer Zeile bzw. Spalte erfordert dies **maximal 10 Unterteilungen** (\Rightarrow Mittelpunktalgorithmus).
 - In Hardware ist diese Variante wegen **schneller Division durch 2** (Bitshift) und ihrer Parallelisierbarkeit schneller als eine direkte Schnittpunktberechnung.

- Polygone begrenzen Flächen
⇒ Polygon-Clipping muss **wieder geschlossene Polygone** liefern, also ggf. Teile des Fensterrandes einbauen
- Ein einfacher Algorithmus würde **jede Seite** gegen die Fenster clippen.
- Wenn eine Seite das Fenster verlässt, wird der **Austrittspunkt mit dem Wiedereintritt** verbunden
⇒ Ecken können zu Problemen führen

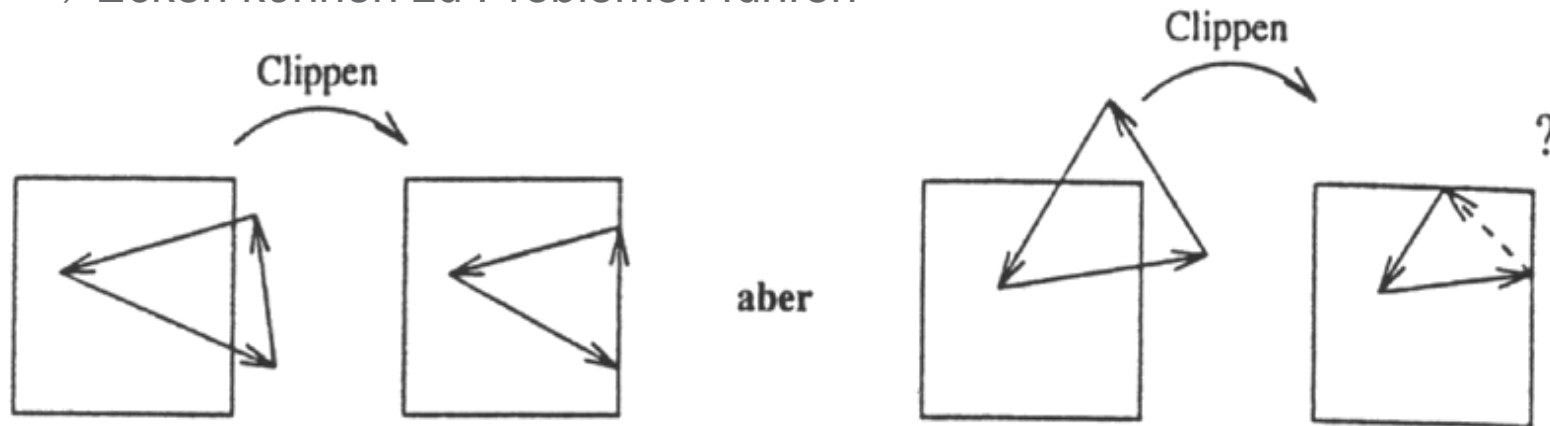
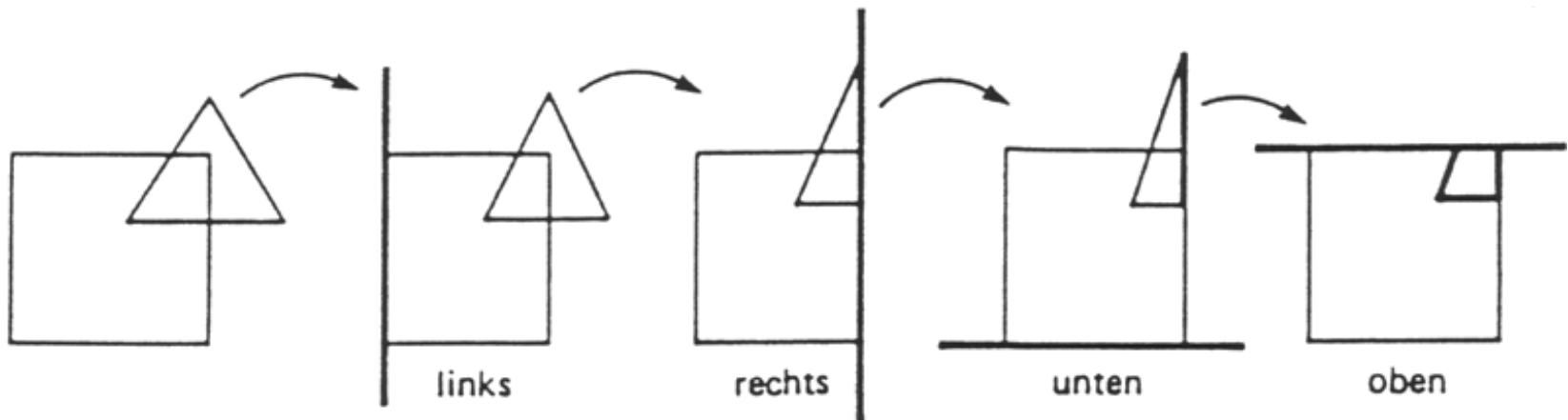


Abb.: Einfügen von Fenstergrenzen beim Polygon-Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

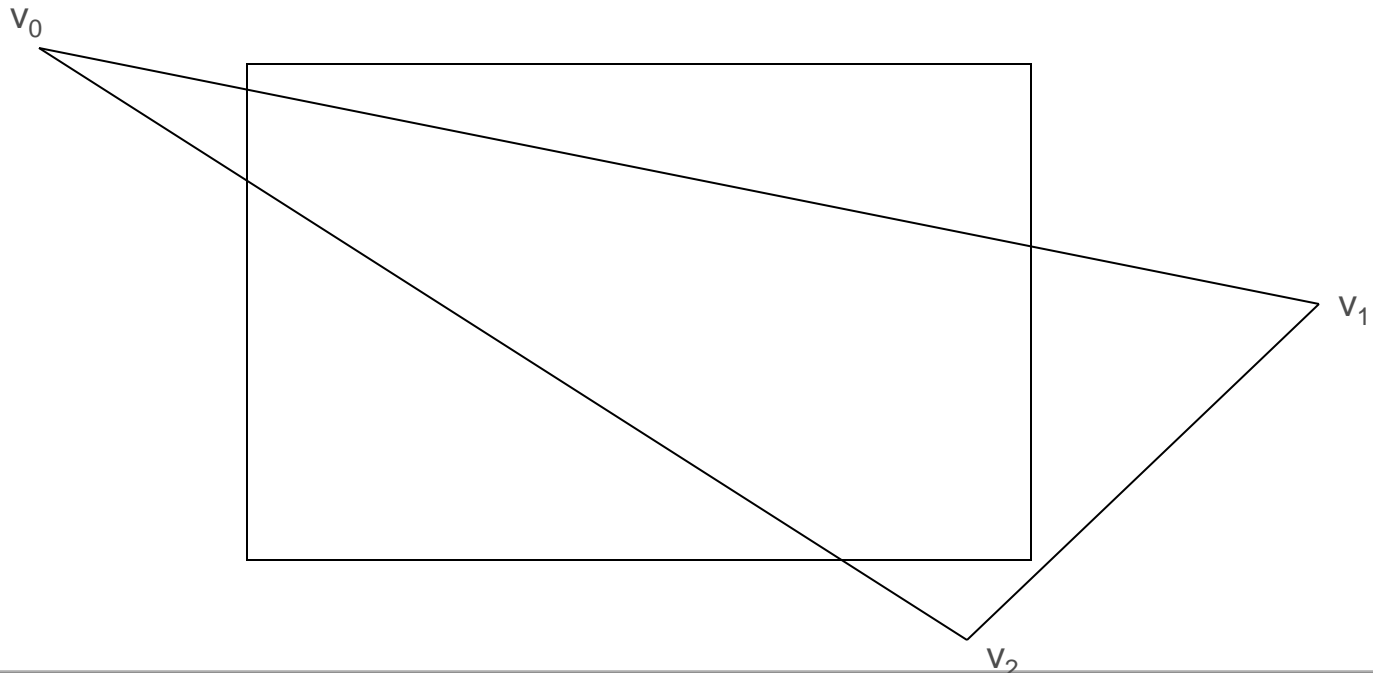
- Problem: Clippen jeder Polygonseite gegen alle 4 Fensterseiten
- Vollständige Clippen des Polygons gegen **eine Fensterseite nach der anderen** zum Ziel.



- Die Zwischenergebnisse müssen gespeichert werden.

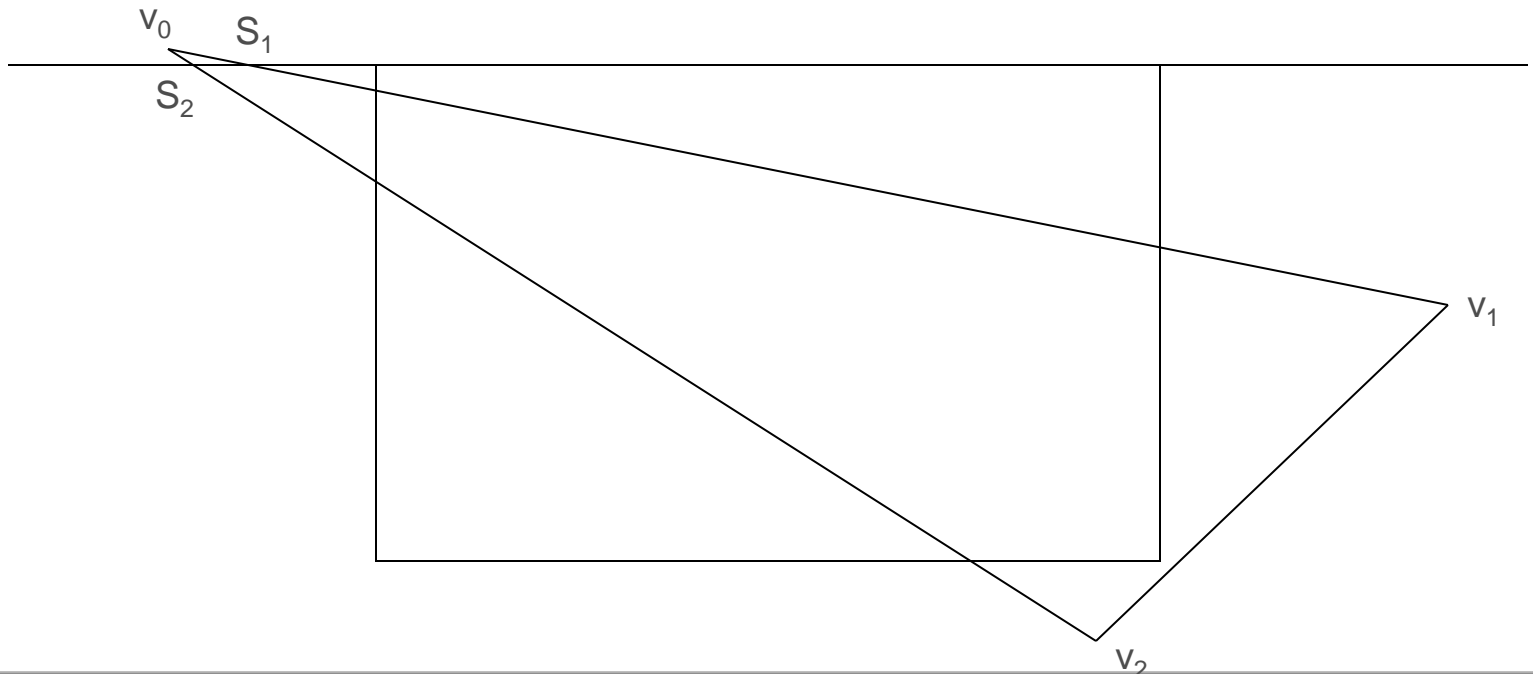
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Ausgangssituation: aktuelles Polygon $\{v_0, v_1, v_2\}$



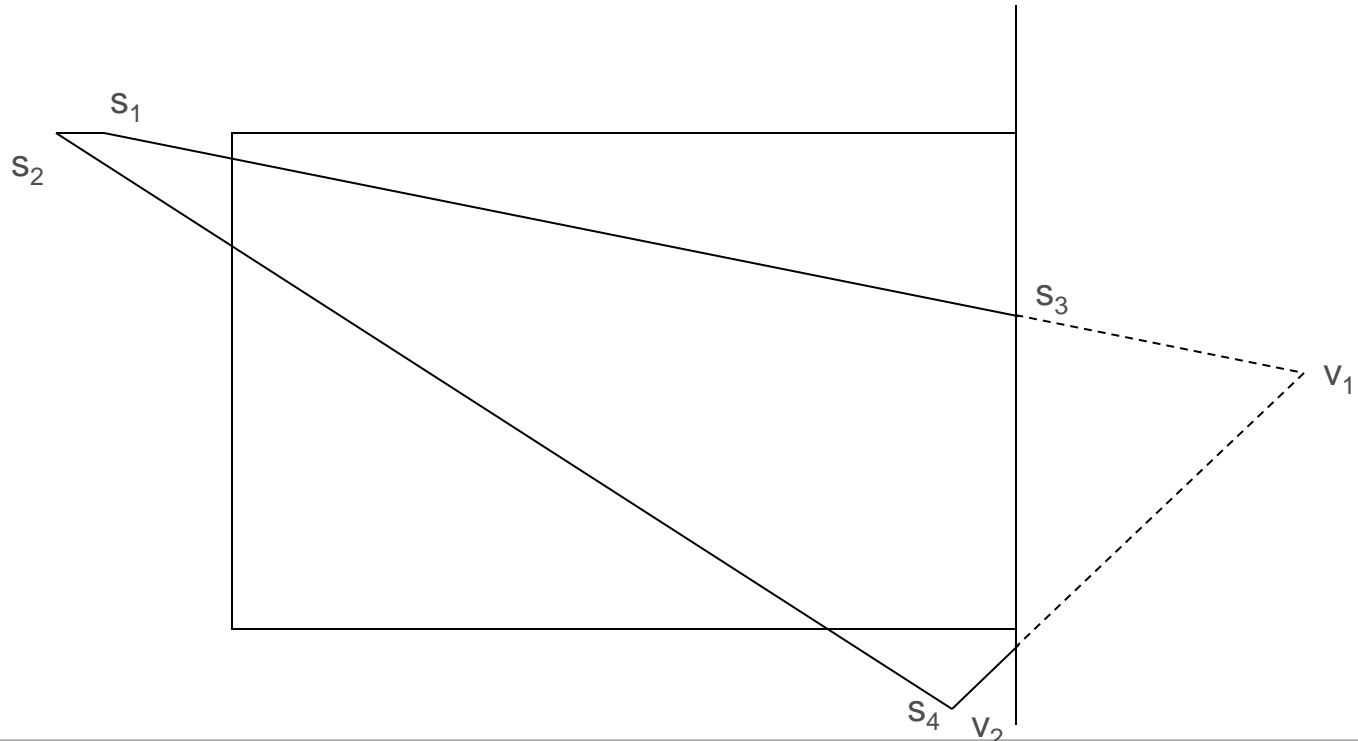
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Clip oben: aktuelles Polygon $\{s_1, v_1, v_2, s_2\}$



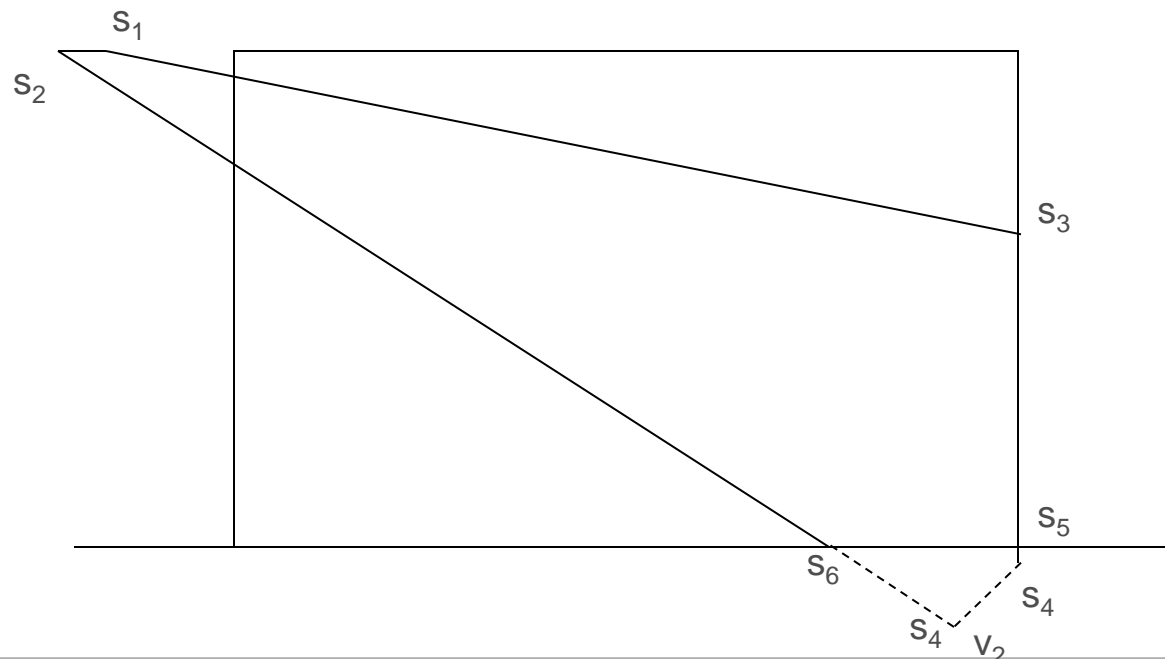
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Clip rechts: aktuelles Polygon $\{s_1, s_3, s_4, v_2, s_2\}$



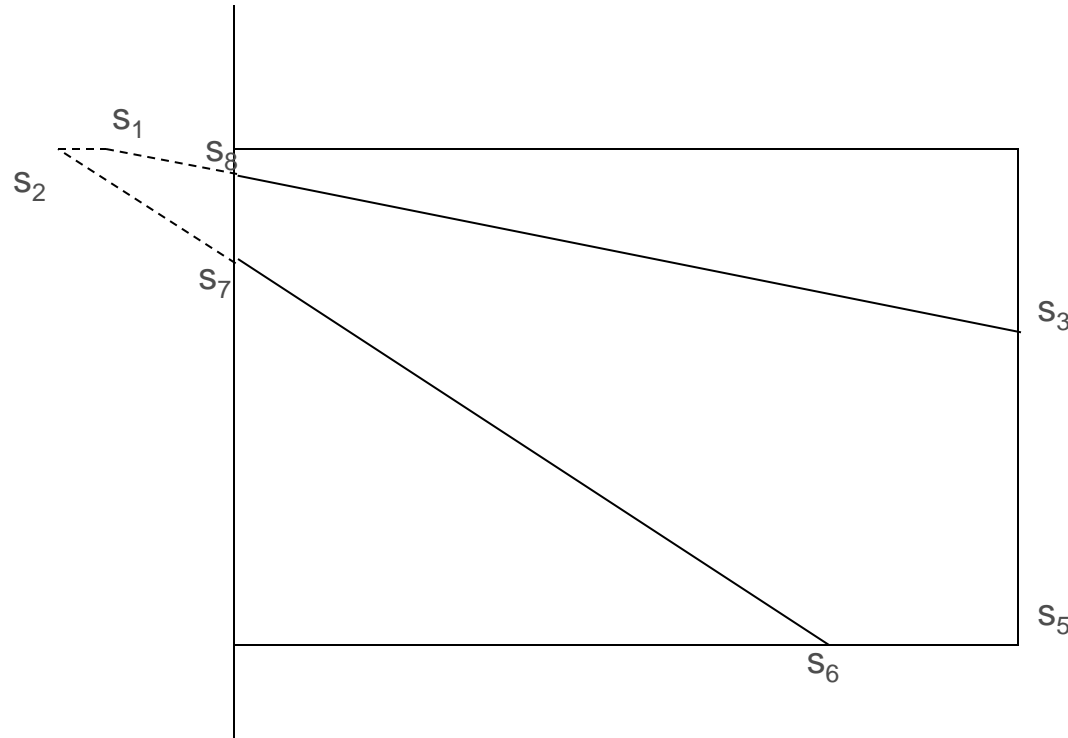
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Clip unten: aktuelles Polygon $\{s_1, s_3, s_5, s_6, s_2\}$



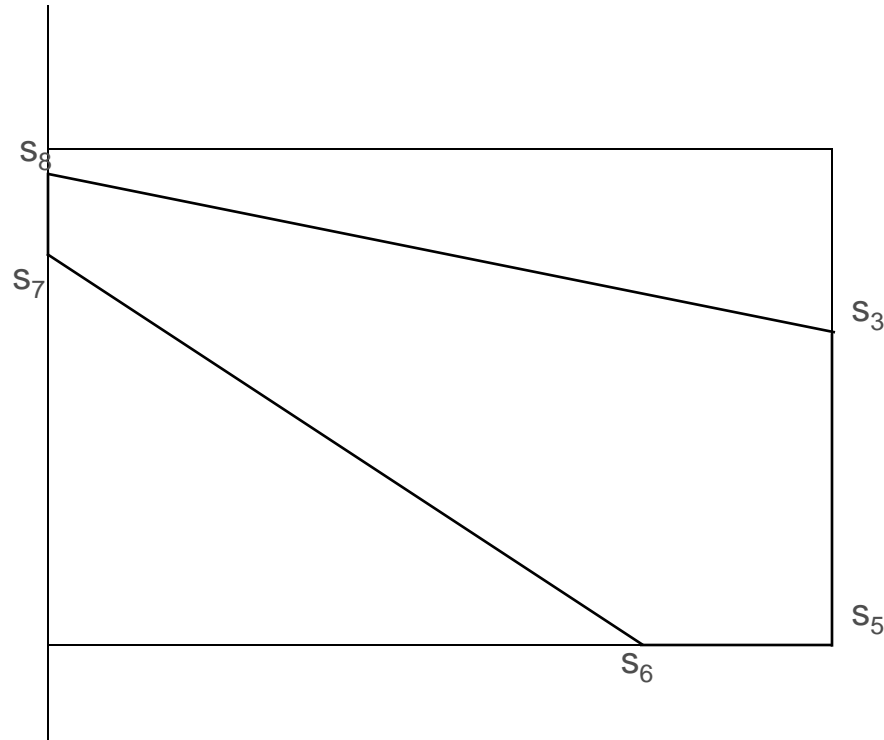
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Clip links: aktuelles Polygon $\{s_8, s_3, s_5, s_6, s_7\}$

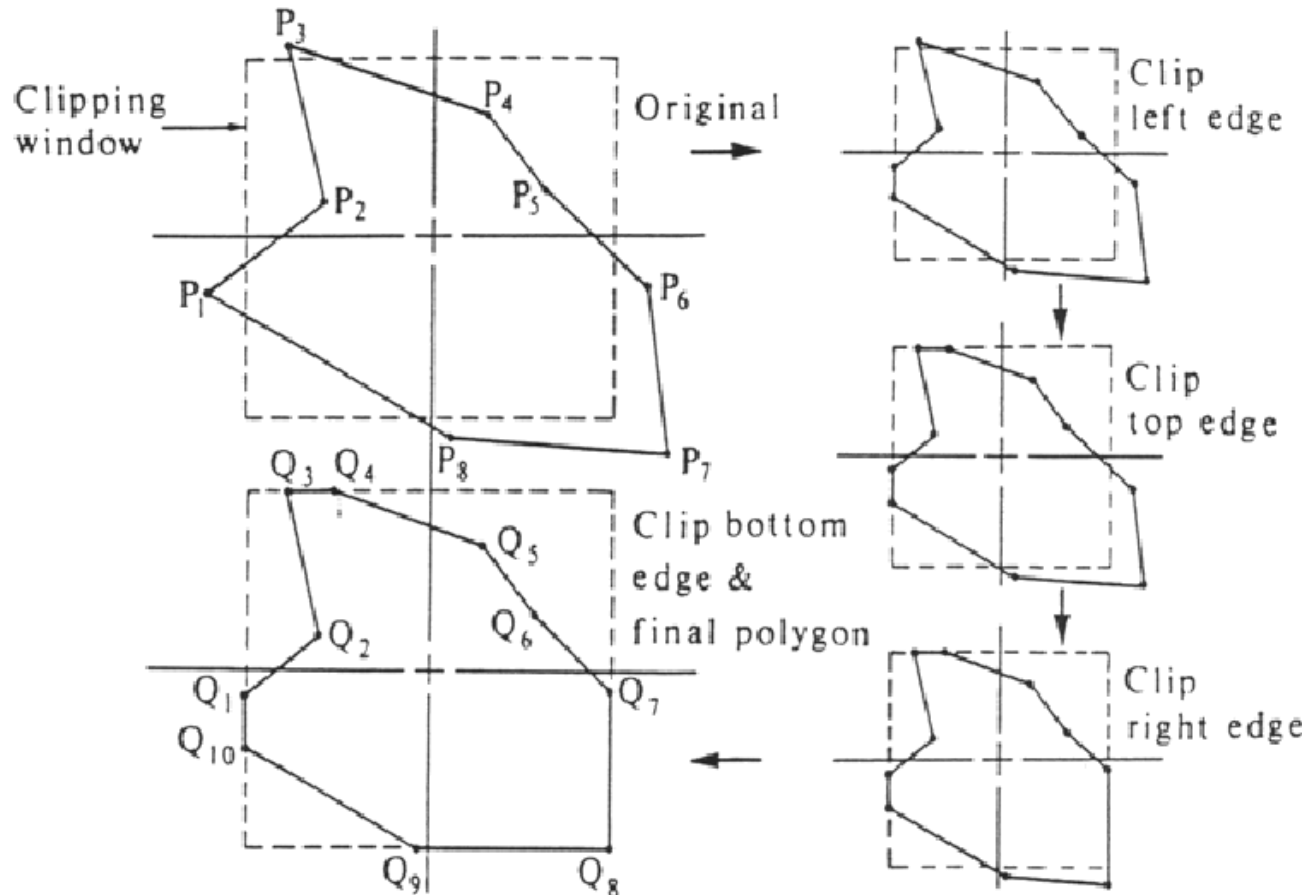


Sutherland-Hodgman Polygon-Clipping Algorithmus

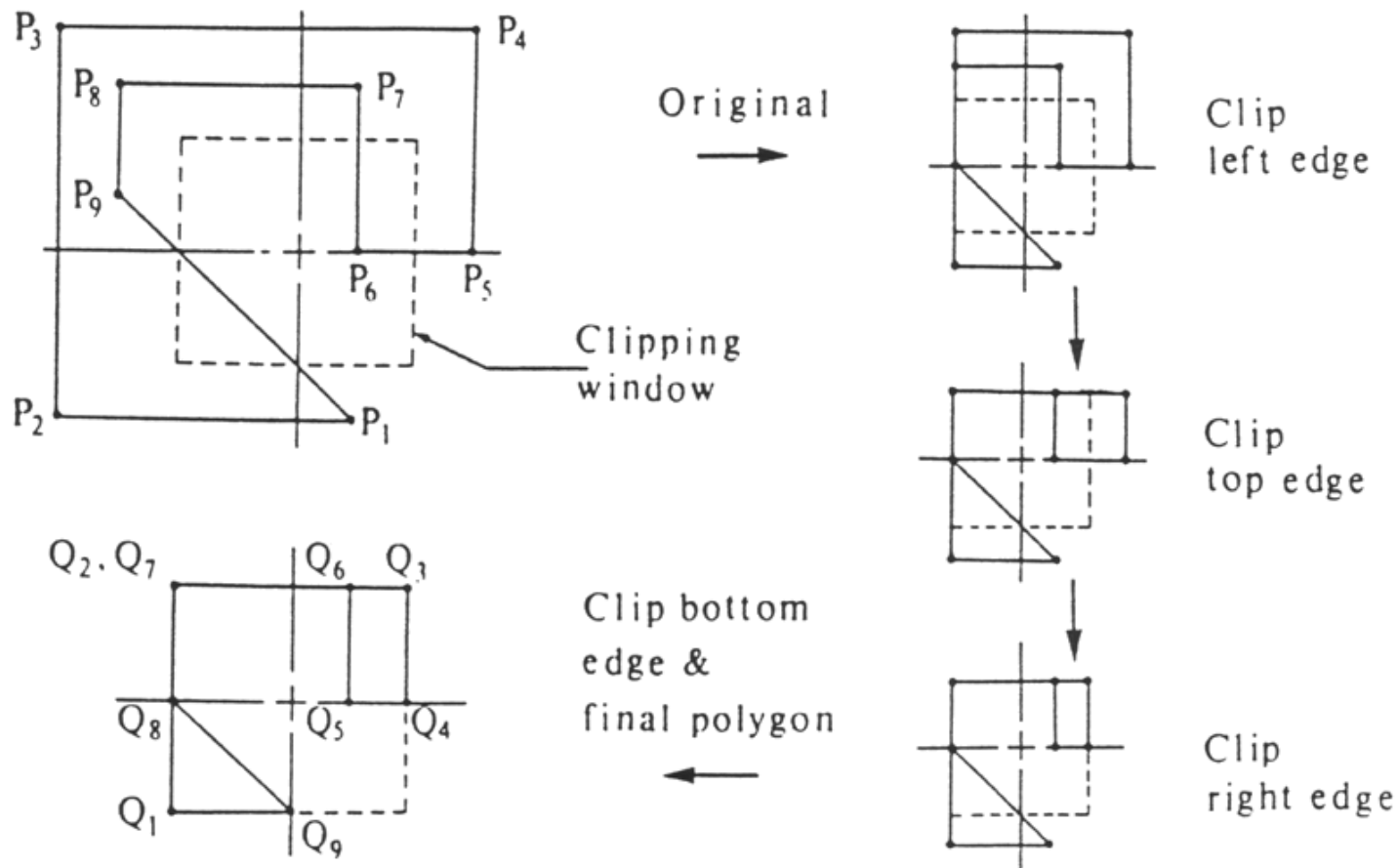
- Polygon nach Clipping: $\{s_8, s_3, s_5, s_6, s_7\}$



Sutherland-Hodgman Polygon-Clipping Algorithmus

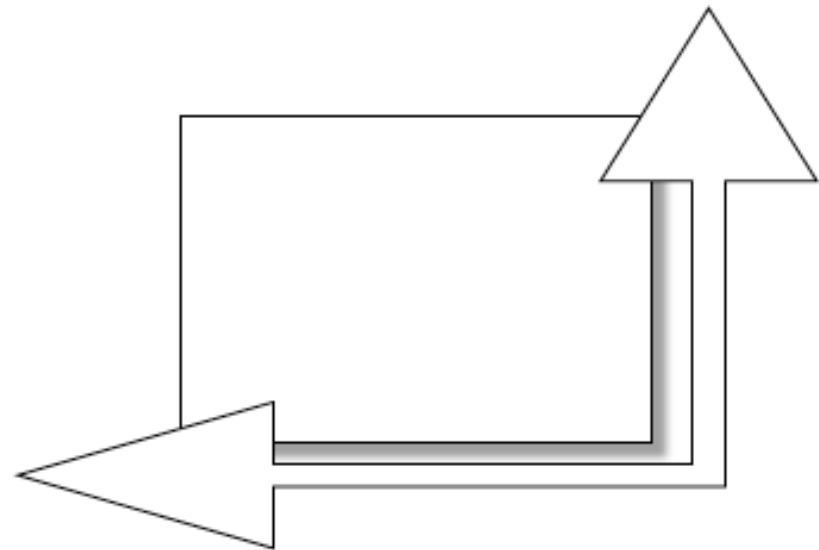
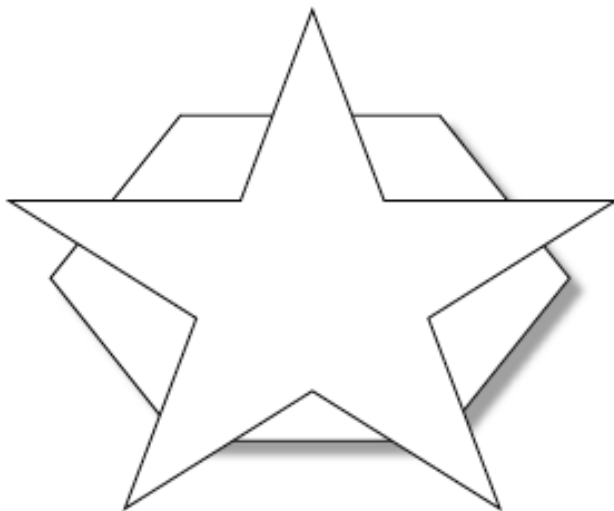


Sutherland-Hodgman Polygon-Clipping Algorithmus



Sutherland-Hodgman Polygon-Clipping Algorithmus

- Wenn Fenster konvexes Polygon ist
 - Und Kandidat einfaches Polygon
- ⇒ Ergebnis ist immer ein **geschlossener Kantenzug**



Andere Polygon-Clipping-Verfahren

- Vatti-Algorithmus: Scan-Line [CACM 35 1992]
- Greiner/Hormann [ACM TOG 17(2),1998]

- Clipping nach Transformation in das normalisierte Sichtvolumen (NDC)
- Clippingverfahren lassen sich einfach in 3D übertragen: Clippen an sechs Halbebenen statt vier
- Alternativ:
Clipping nach Projektion in die zweidimensionale Bildebene \Rightarrow w-Clipping