



UNIVERSITÄT
LEIPZIG

Viewport und Clipping

COMPUTERGRAPHIK

Inhaltsverzeichnis

7 Viewport und Clipping

7.1 Viewport

7.2 Clipping

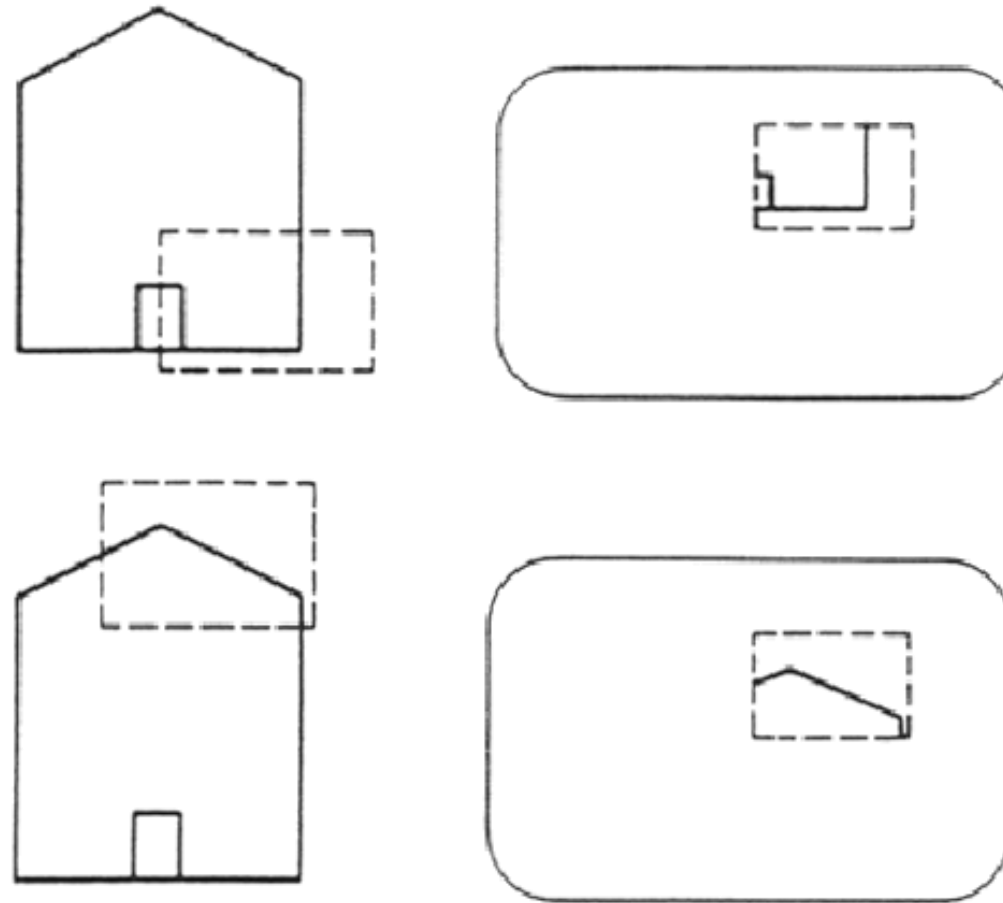
7.1 Viewport

- Window:
 - Definiert Sichtfenster in der Bildebene
 - Definiert, welcher Teilbereich der Szene abgebildet werden soll
 - Auch „View Window“ genannt
- Viewport:
 - Definiert Bildschirmbereich in dem der Inhalt eines Windows dargestellt werden soll

7.1 Viewport

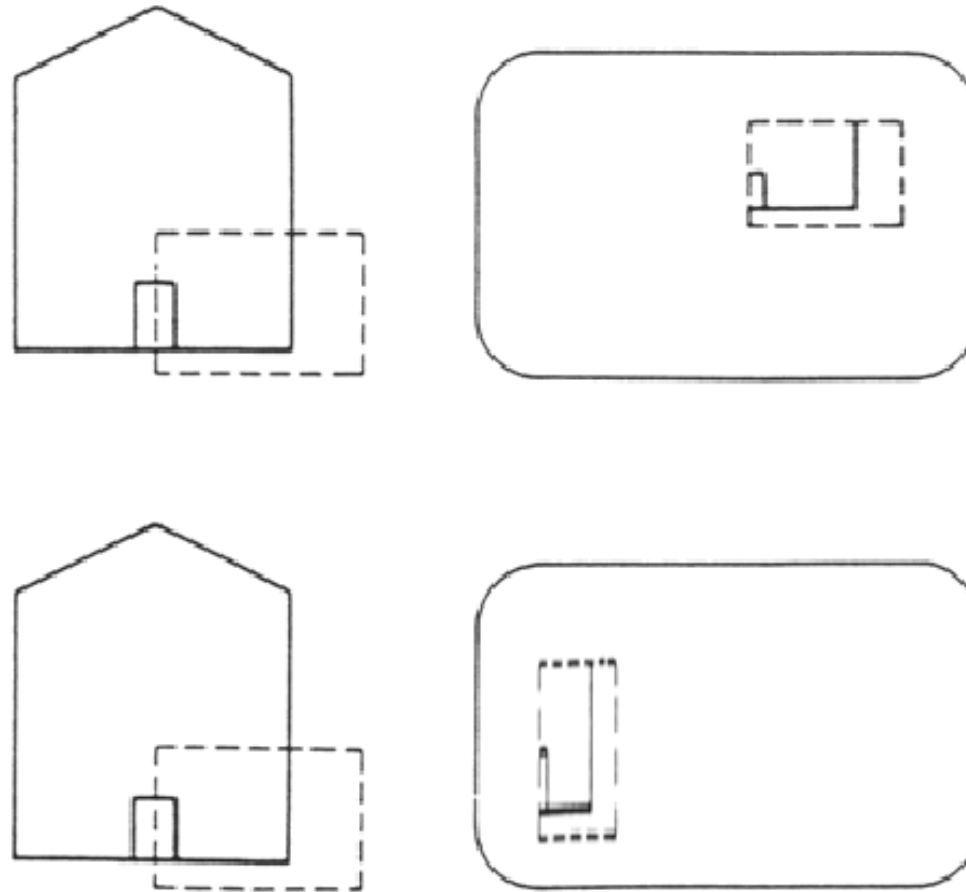
- In der Regel sind sowohl Window als auch Viewport an den Koordinatenachsen ausgerichtete rechteckige Gebiete
- Window-Viewport-Transformation (Windowing Operation):
 - elementare Verschiebungen und Skalierungen

7.1 Viewport



Verschiedene Fenster, dieselben Viewports

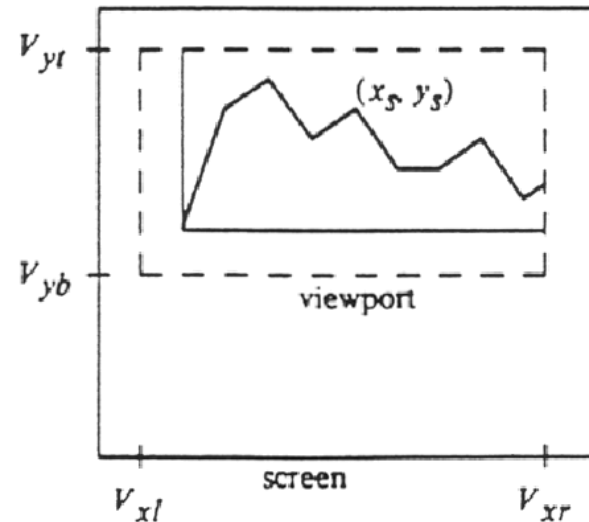
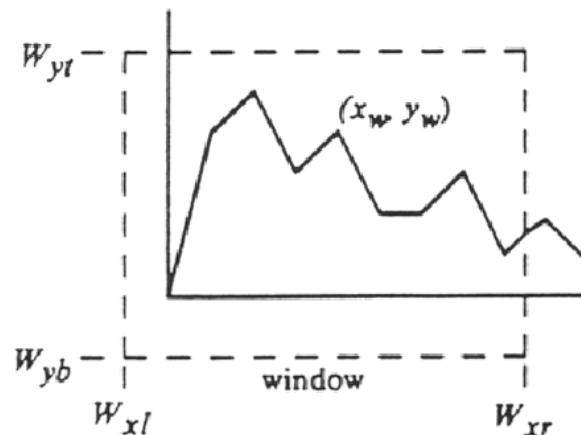
7.1 Viewport



Dieselben Fenster, verschiedene Viewports

7.1 Viewport

- Koordinaten des Windows:
 $W_{xl}, W_{xr}, W_{yb}, W_{yt}$
- Punktkoordinaten im Fenster (window):
 x_W, y_W
- Koordinaten des Viewports im Bildschirmkoordinatensystem
 $V_{xl}, V_{xr}, V_{yb}, V_{yt}$
- Punktkoordinaten auf dem Bildschirm (screen):
 x_S, y_S



7.1 Viewport

Transformation

1) Verschiebung

in den Koordinatenursprung

$$\begin{aligned}x &= x_W - W_{xl} \\ y &= y_W - W_{yb}\end{aligned}$$

2) Skalierung

auf die gewünschte Größe

$$\begin{aligned}x' &= \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} \cdot x \\ y' &= \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} \cdot y\end{aligned}$$

3) Verschiebung

an die gewünschte Stelle

$$\begin{aligned}x_S &= x' + V_{xl} \\ y_S &= y' + V_{yb}\end{aligned}$$

7.1 Viewport

Transformation

Zusammenfassung zu

$$\begin{aligned}x_S &= a \cdot x_W + b \\ y_S &= c \cdot y_W + d\end{aligned}$$

⇒ Punkttransformation durch
zwei Multiplikationen und
zwei Additionen

a, b, c, d fest für alle
Punkttransformationen

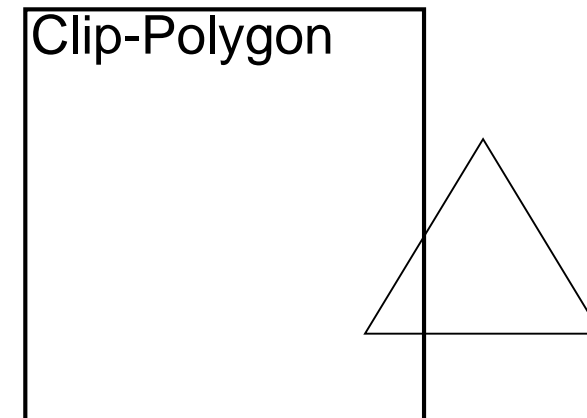
$$\begin{aligned}a &:= \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} \\ c &:= \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}}\end{aligned}$$

$$\begin{aligned}b &= V_{xl} - a \cdot W_{xl} \\ d &= V_{yb} - c \cdot W_{yb}\end{aligned}$$

7.2 Clipping

Clip-Polygon

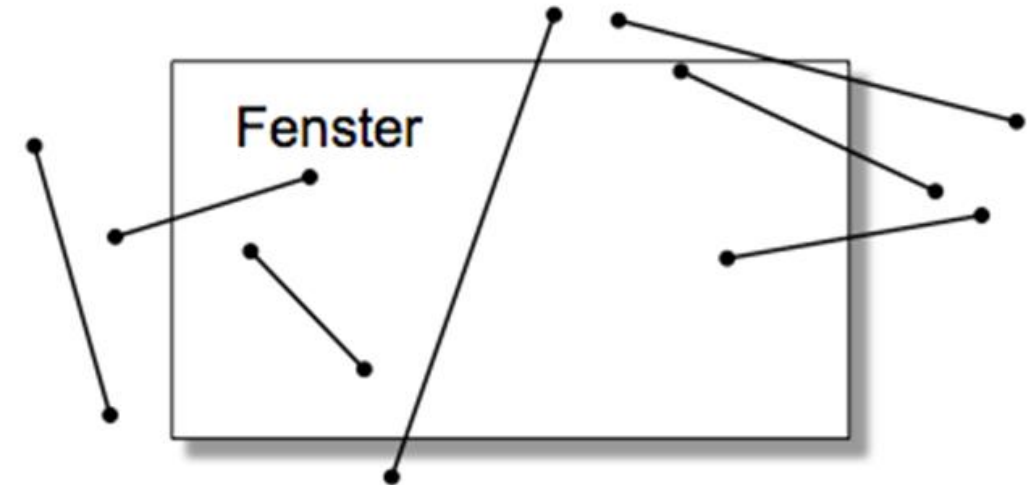
- Objekte in der Bildebene werden innerhalb eines Fensters dargestellt
- Alle außerhalb des Fensters liegenden Objektteile werden abgeschnitten:
Clipping am Fensterrand
- Das Fenster wird Clip-Polygon genannt
 - typischerweise Rechtecke
 - andere Geometrie möglich
 - Nichtkonvexe oder nicht einfache Polygone sind problematisch



7.2 Clipping

Linien

- Clip-Polygon
 - rechteckig
 - achsenparallel
- Fallunterscheidung
 1. beide Endpunkte innerhalb des Fensters
⇒ Linie zeichnen
 2. beide Endpunkte oberhalb oder unterhalb oder links oder rechts des Fensters
⇒ Linie nicht zeichnen

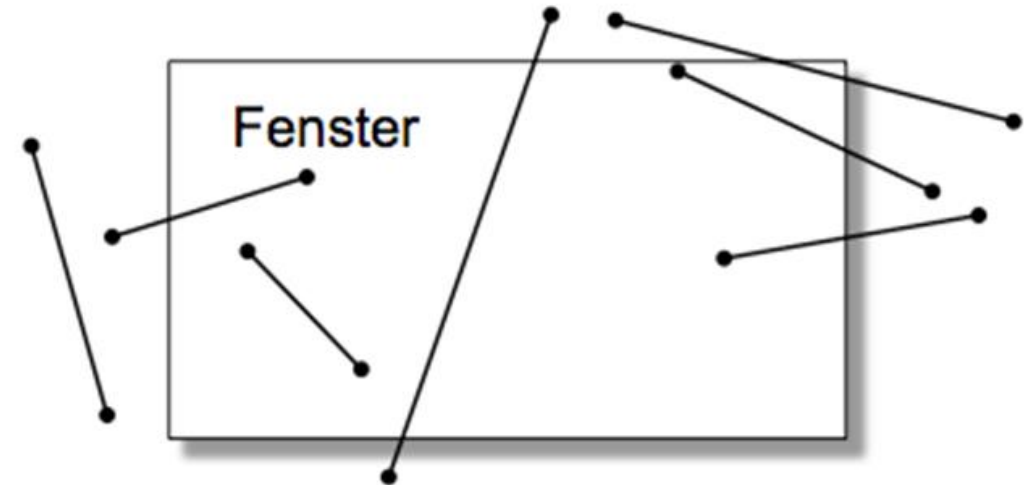


7.2 Clipping

Linien

3. Sonst

- Berechne die Schnittpunkte der Linie mit dem Fensterrand anhand der Geradengleichungen
- Bestimme daraus die sichtbare Strecke



7.2 Clipping

Liang-Barsky-Algorithmus

- Fensterkanten als implizite Gerade

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

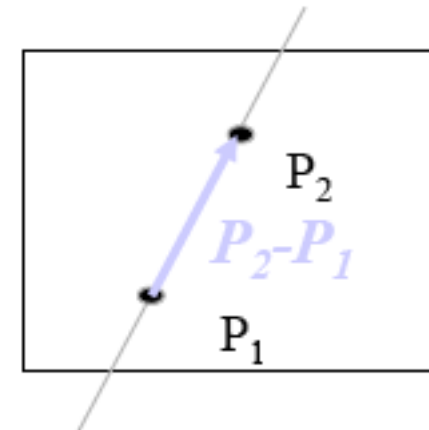
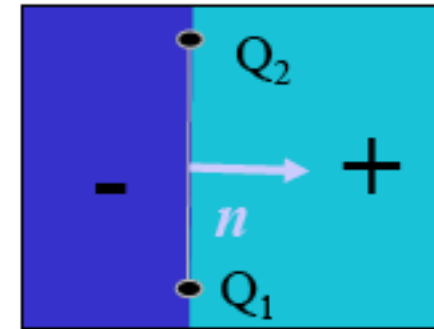
$$n = (\Delta y, -\Delta x) = ((y_2 - y_1), -(x_2 - x_1))$$

$$P = (x, y)$$

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

- Normale n zeigt ins Innere
- Parametrische Darstellung der Liniensegmente

$$l(t) = P_1 + t \cdot (P_2 - P_1)$$



7.2 Clipping

Liang-Barsky-Algorithmus

– Fallunterscheidung:

1) P_1 und P_2 liegen außen

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

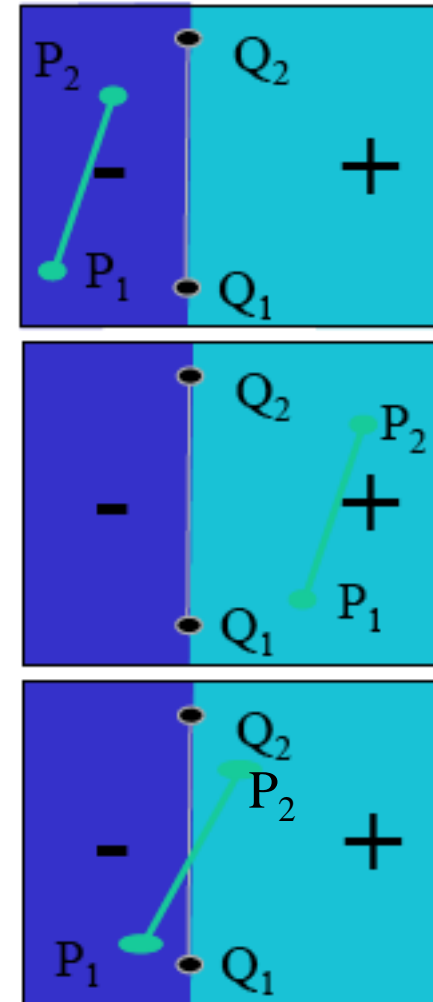
2) P_1 und P_2 liegen innen

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

3) P_1 und P_2 liegen auf verschiedenen Seiten

$$E(P_1) < 0, E(P_2) > 0$$

$$E(P_1) > 0, E(P_2) < 0$$



7.2 Clipping

Liang-Barsky-Algorithmus

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

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

$$\Leftrightarrow$$

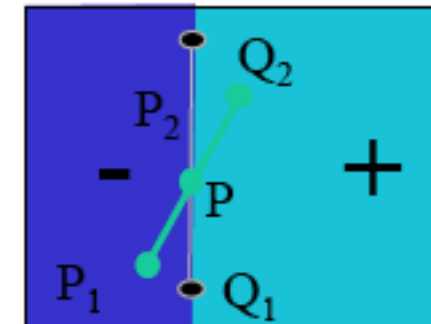
$$(P_1 + t \cdot (P_2 - P_1)) \circ n - Q_1 \circ n = 0$$

$$\Leftrightarrow$$

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

$$\Leftrightarrow$$

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



7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Kern des Algorithmus ist ein schnelles Verfahren zur Bestimmung der Kategorie einer Linie
 - innerhalb
 - außerhalb
 - schneidend
- Gegeben:
 - Ein Fenster: $(x_{min}, y_{min}, x_{max}, y_{max})$ 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

7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Outcode-Bits werden gesetzt, falls der Eckpunkt in der jeweiligen Region liegt

| Bit | Ort | Bedingung |
|-----|------------------------|----------------|
| 0 | links des Fensters | $x < x_{\min}$ |
| 1 | rechts des Fensters | $x > x_{\max}$ |
| 2 | unterhalb des Fensters | $y < y_{\min}$ |
| 3 | oberhalb des Fensters | $y > y_{\max}$ |

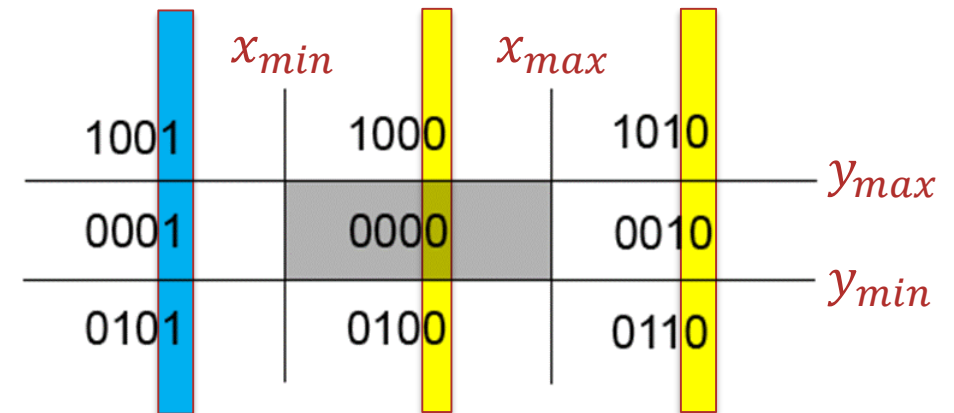
| | | | |
|------|------------|------------|------------|
| | x_{\min} | x_{\max} | |
| 1001 | 1000 | 1010 | |
| 0001 | 0000 | 0010 | y_{\max} |
| 0101 | 0100 | 0110 | y_{\min} |

7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Outcode-Bits werden gesetzt, falls der Eckpunkt in der jeweiligen Region liegt

| Bit | Ort | Bedingung |
|-----|------------------------|----------------|
| 0 | links des Fensters | $x < x_{\min}$ |
| 1 | rechts des Fensters | $x > x_{\max}$ |
| 2 | unterhalb des Fensters | $y < y_{\min}$ |
| 3 | oberhalb des Fensters | $y > y_{\max}$ |

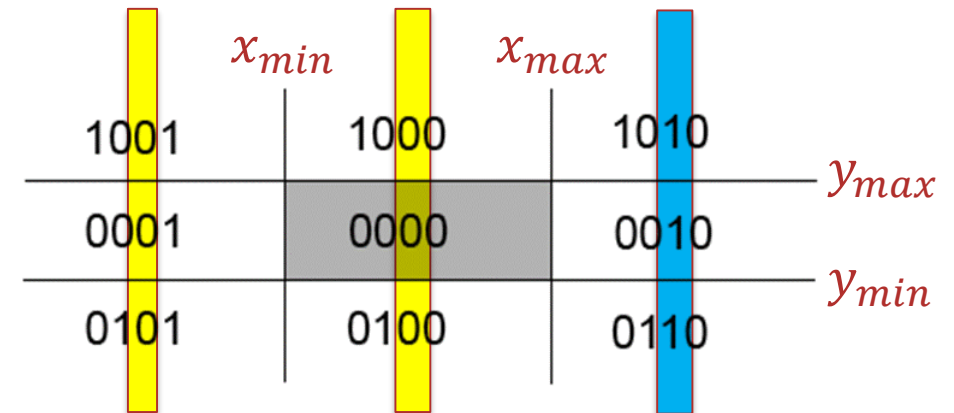


7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Outcode-Bits werden gesetzt, falls der Eckpunkt in der jeweiligen Region liegt

| Bit | Ort | Bedingung |
|-----|------------------------|----------------|
| 0 | links des Fensters | $x < x_{\min}$ |
| 1 | rechts des Fensters | $x > x_{\max}$ |
| 2 | unterhalb des Fensters | $y < y_{\min}$ |
| 3 | oberhalb des Fensters | $y > y_{\max}$ |

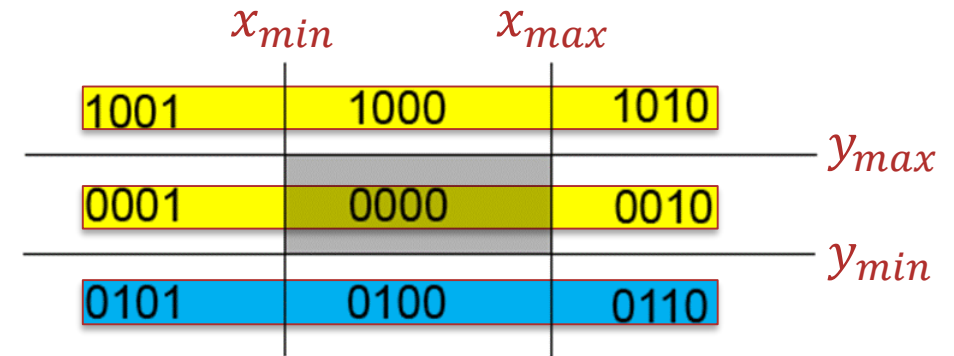


7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Outcode-Bits werden gesetzt, falls der Eckpunkt in der jeweiligen Region liegt

| Bit | Ort | Bedingung |
|-----|-------------------------------|----------------|
| 0 | links des Fensters | $x < x_{\min}$ |
| 1 | rechts des Fensters | $x > x_{\max}$ |
| 2 | unterhalb des Fensters | $y < y_{\min}$ |
| 3 | oberhalb des Fensters | $y > y_{\max}$ |

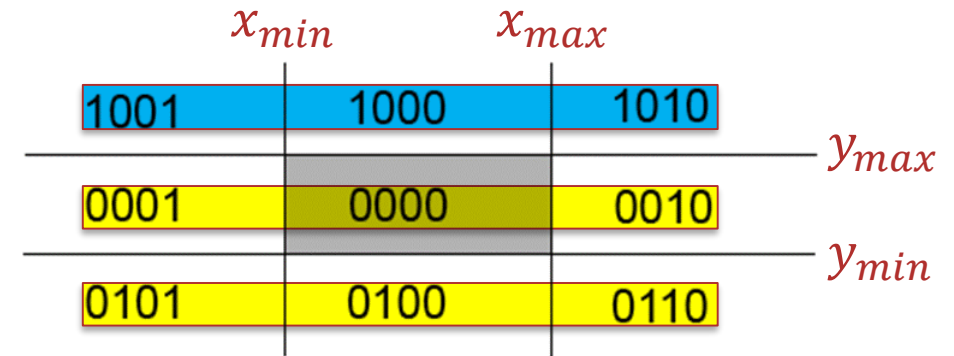


7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

- Outcode-Bits werden gesetzt, falls der Eckpunkt in der jeweiligen Region liegt

| Bit | Ort | Bedingung |
|-----|------------------------|----------------|
| 0 | links des Fensters | $x < x_{\min}$ |
| 1 | rechts des Fensters | $x > x_{\max}$ |
| 2 | unterhalb des Fensters | $y < y_{\min}$ |
| 3 | oberhalb des Fensters | $y > y_{\max}$ |



7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

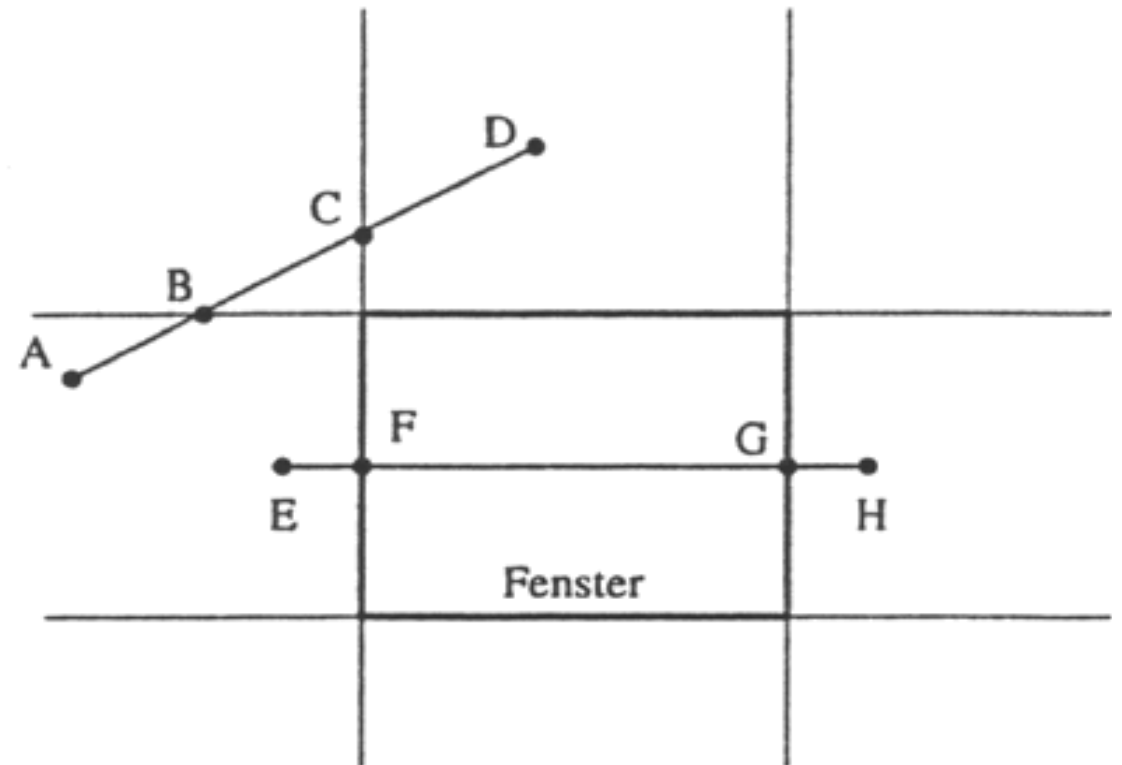
- Bestimme Outcodes für Endpunkte einer Linie
 - Linie liegt vollständig außerhalb des Fensters (Trivial Reject):
Durchschnitt der Codes beider Endpunkte ist von 0 verschieden (AND-Verknüpfung $\neq 0$)
 - Linie liegt komplett im Fenster (Trivial Accept):
Beide Endpunkte besitzen den 4-Bit-Code 0000 (OR-Verknüpfung $= 0$)
- Sonst:
 - Schneide die Linie nacheinander mit den das Fenster begrenzenden Geraden
→ es entstehen zwei Linien
 - Außerhalb des Fensters liegende Teile können sofort entfernt werden

7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

Linie AD

- Codes 0001 und 1000
 - Schnitt mit linker Fenstergrenze liefert
 - Punkte A und C liegen links: eliminiere AC
 - Punkte C und D liegen oberhalb: eliminiere CD

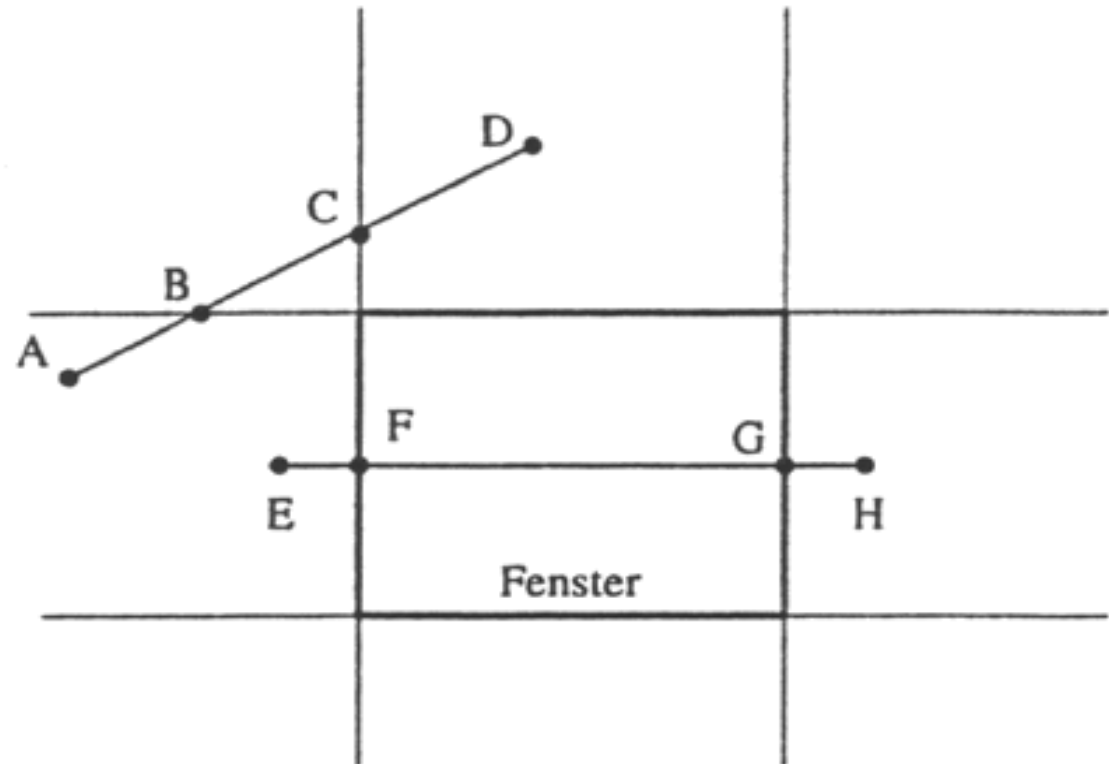


7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus

Linie EH

- Codes 0001 und 0010
 - Schnitt mit linker Fenstergrenze liefert F
 - eliminiere EF
 - behalte FH
 - Schnitt von FH mit der rechten Fenstergrenze liefert G
 - eliminiere GH
 - Punkte F und G liegen innerhalb des Fensters
 - FG wird gezeichnet



7.2 Clipping

Cohen-Sutherland Line-Clipping Algorithmus: Spezialfälle & 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 Schnittooperationen 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

7.2 Clipping

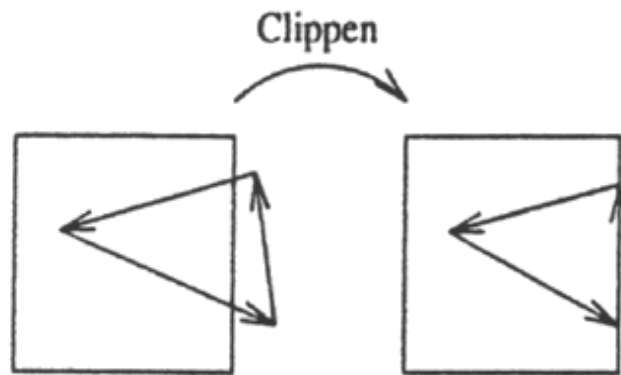
Cohen-Sutherland Line-Clipping Algorithmus: Spezialfälle & Beschleunigungen

- 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 schneller als eine direkte Schnittberechnung
 - schnelle Division durch 2 (Bitshift)
 - Parallelisierbarkeit

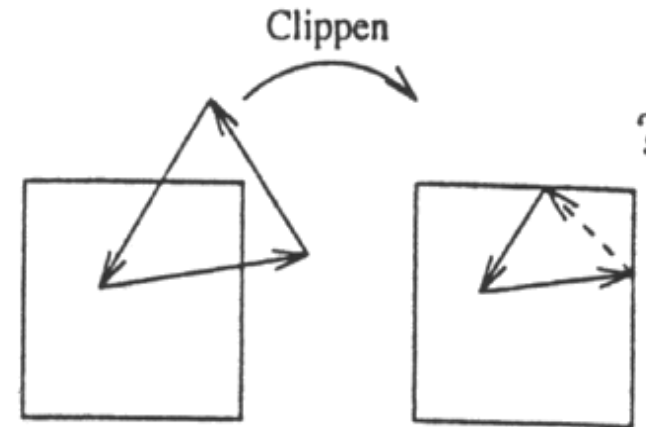
7.2 Clipping

Polygone

- Polygone begrenzen Flächen
 - Polygon-Clipping muss wieder geschlossene Polygone liefern
 - Teile des Fensterrandes müssen eingebaut werden
- 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



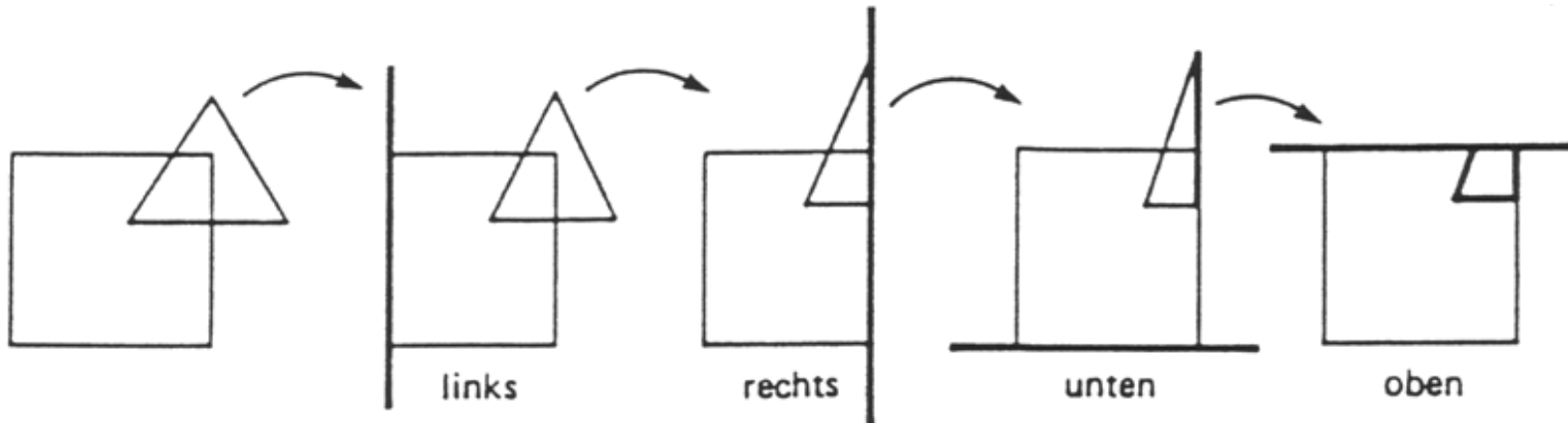
aber



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

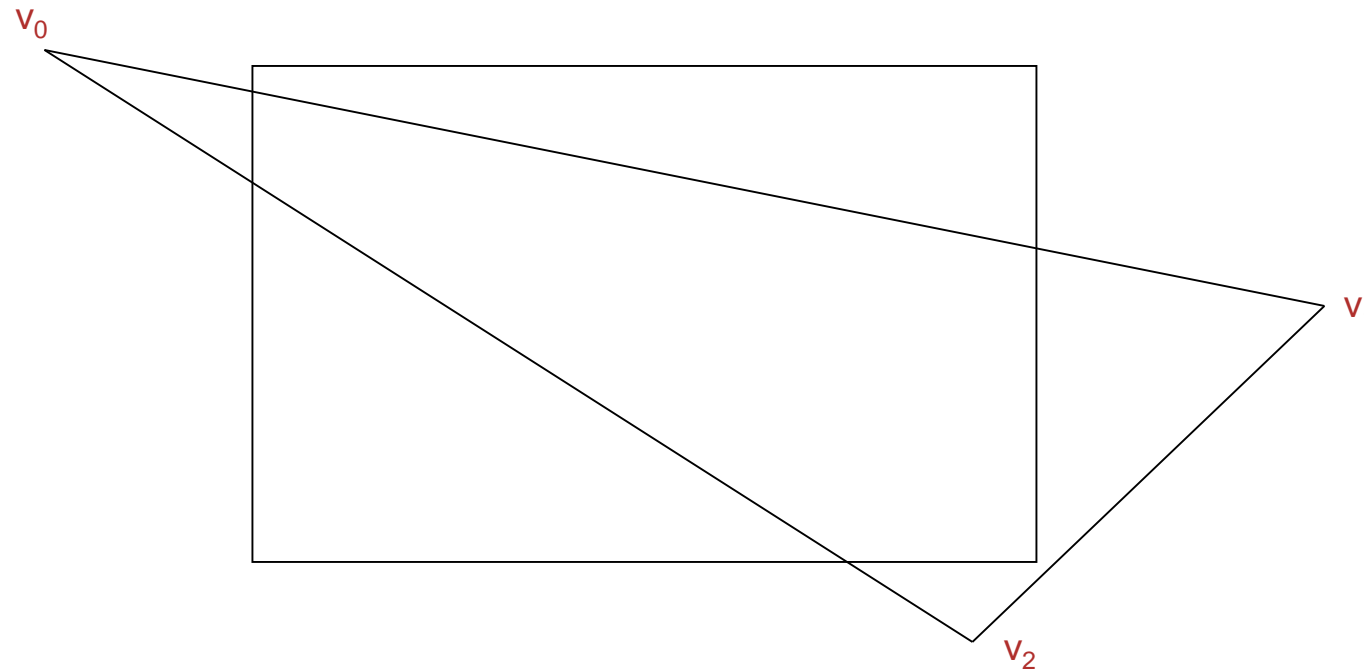
- Problem:
Clippen jeder Polygonseite gegen alle 4 Fensterseiten
- Vollständiges Clippen des Polygons gegen eine Fensterseite nach der anderen führt zum Ziel
- Die Zwischenergebnisse müssen gespeichert werden



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

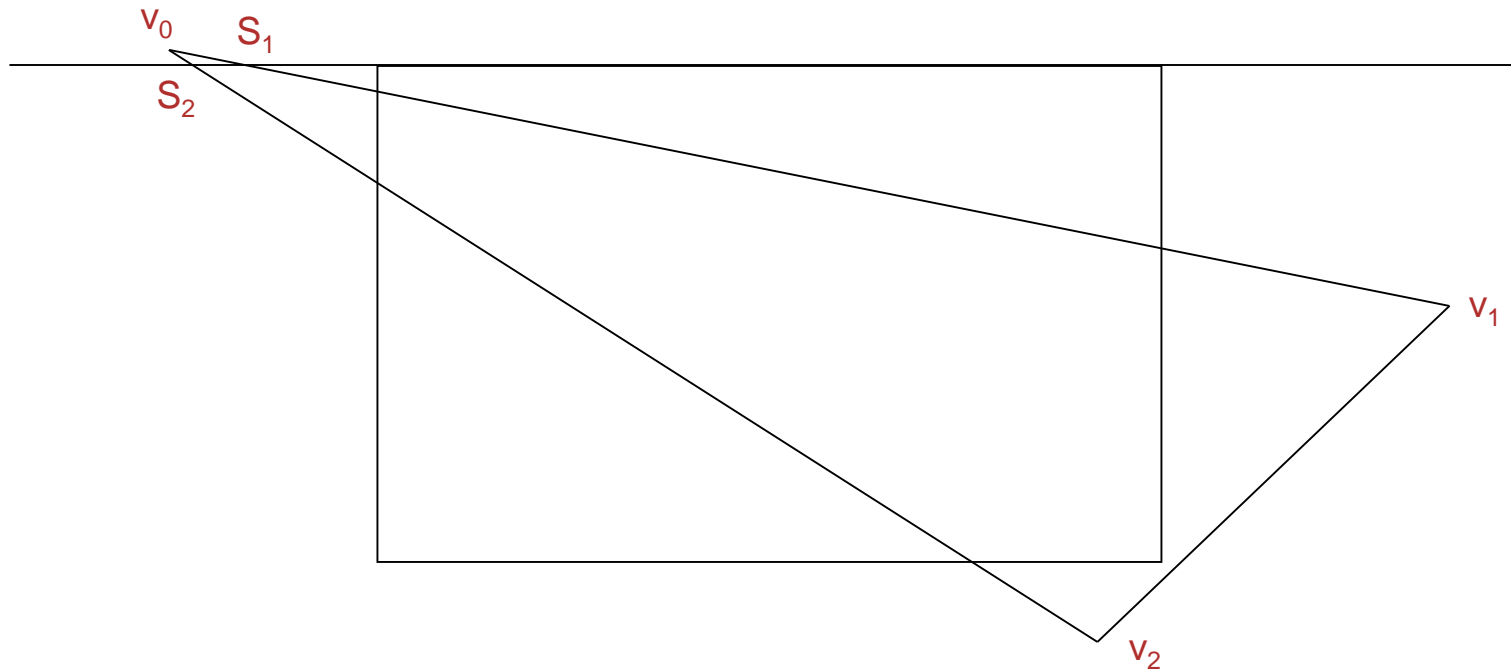
- Ausgangssituation: aktuelles Polygon $\{v_0, v_1, v_2\}$



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

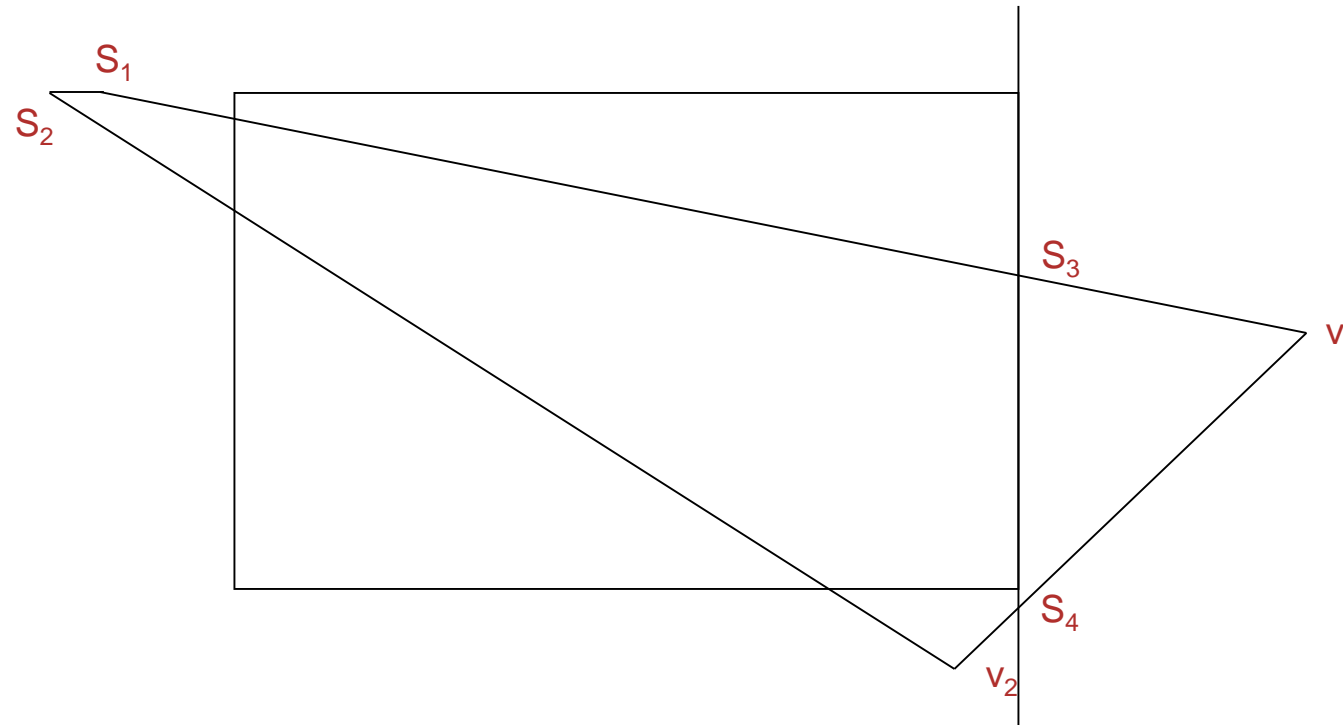
- Clip oben: aktuelles Polygon $\{s_1, v_1, v_2, s_2\}$



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

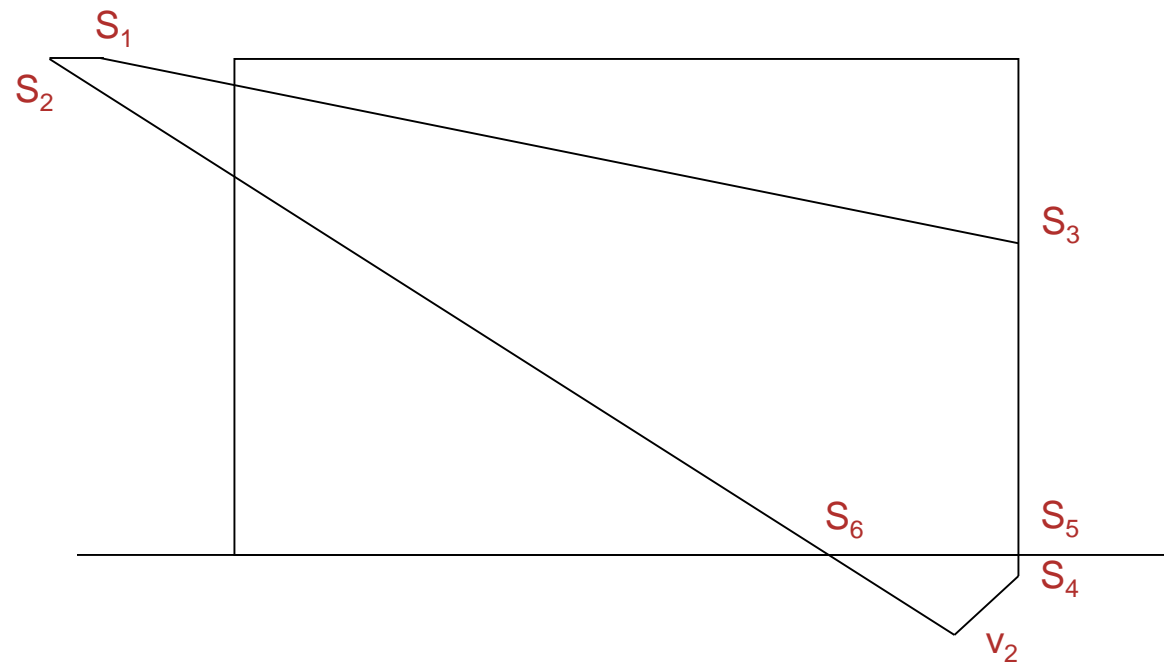
- Clip rechts: aktuelles Polygon $\{s_1, s_3, s_4, v_2, s_2\}$



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

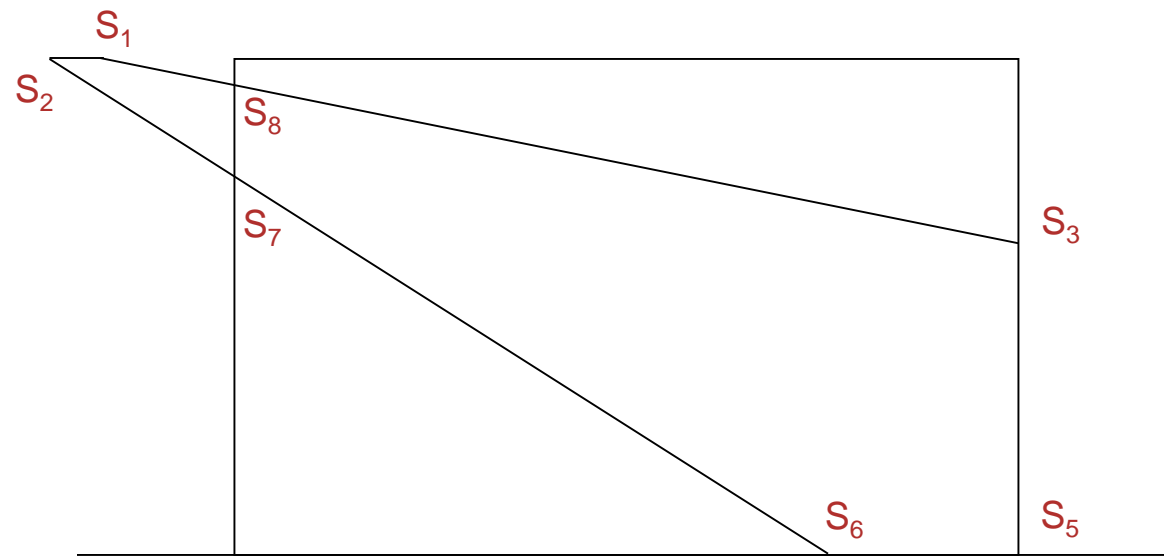
- Clip unten: aktuelles Polygon $\{s_1, s_3, s_5, s_6, s_2\}$



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

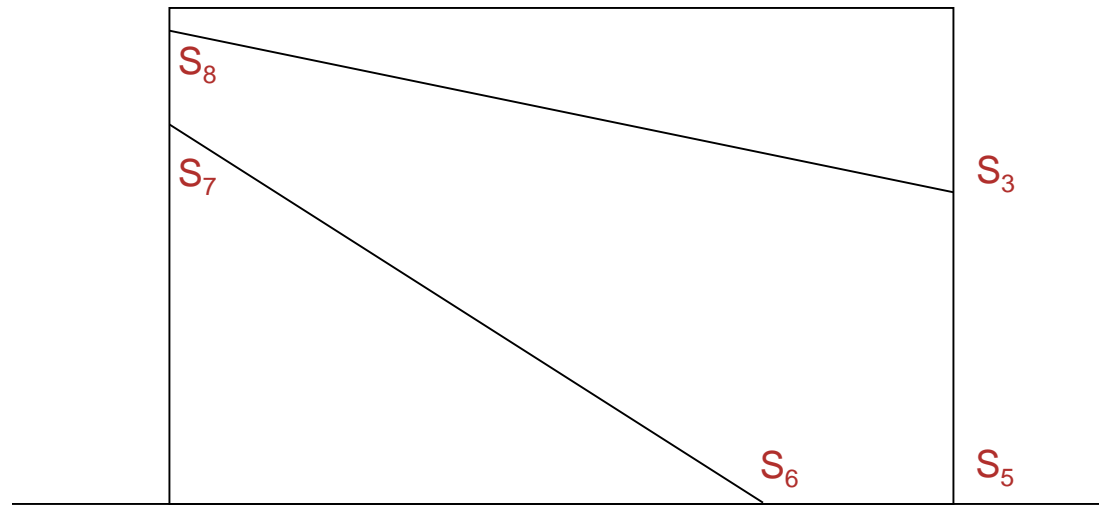
- Clip links: aktuelles Polygon $\{s_8, s_3, s_5, s_6, s_7\}$



7.2 Clipping

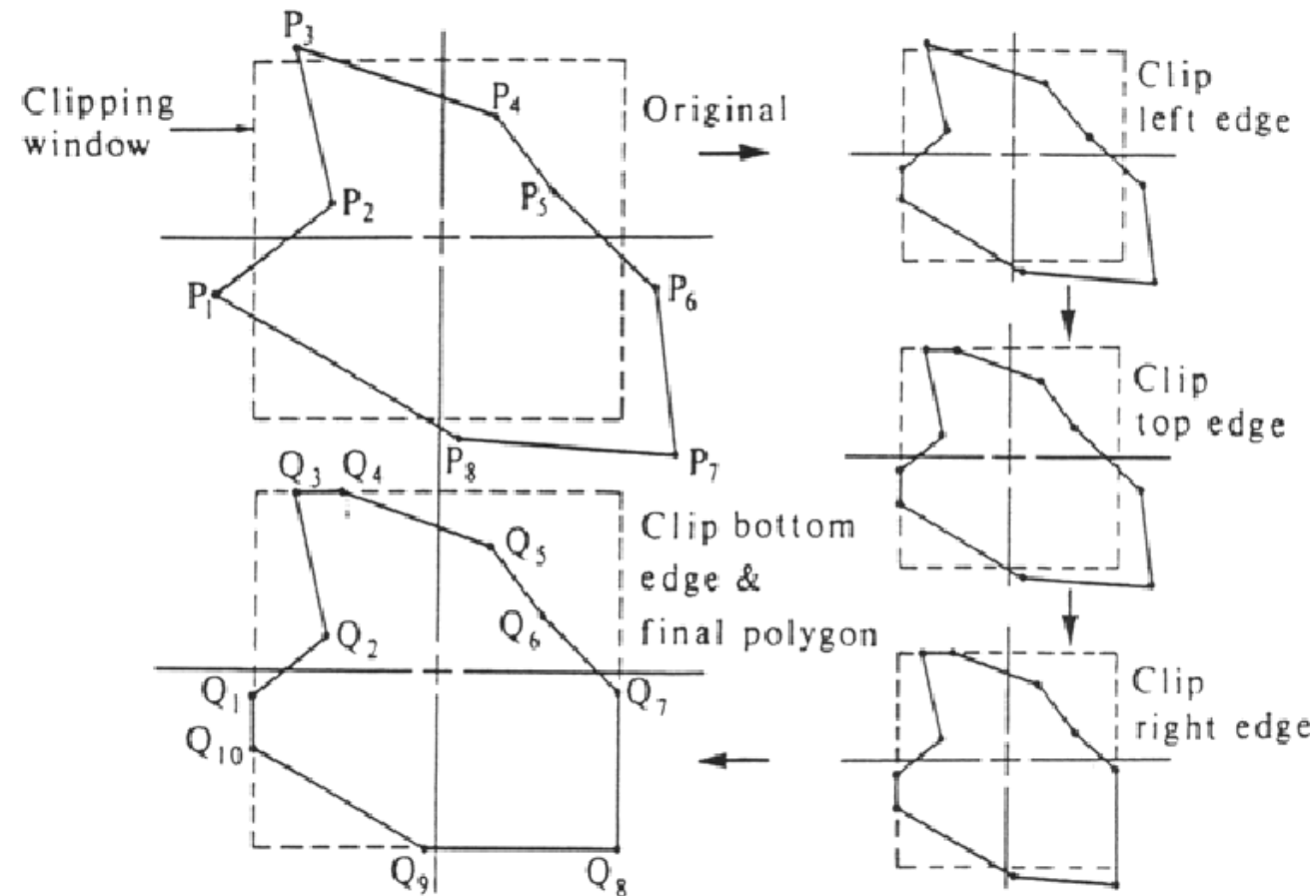
Sutherland-Hodgman Polygon-Clipping Algorithmus

- Ergebnis-Polygon $\{s_8, s_3, s_5, s_6, s_7\}$



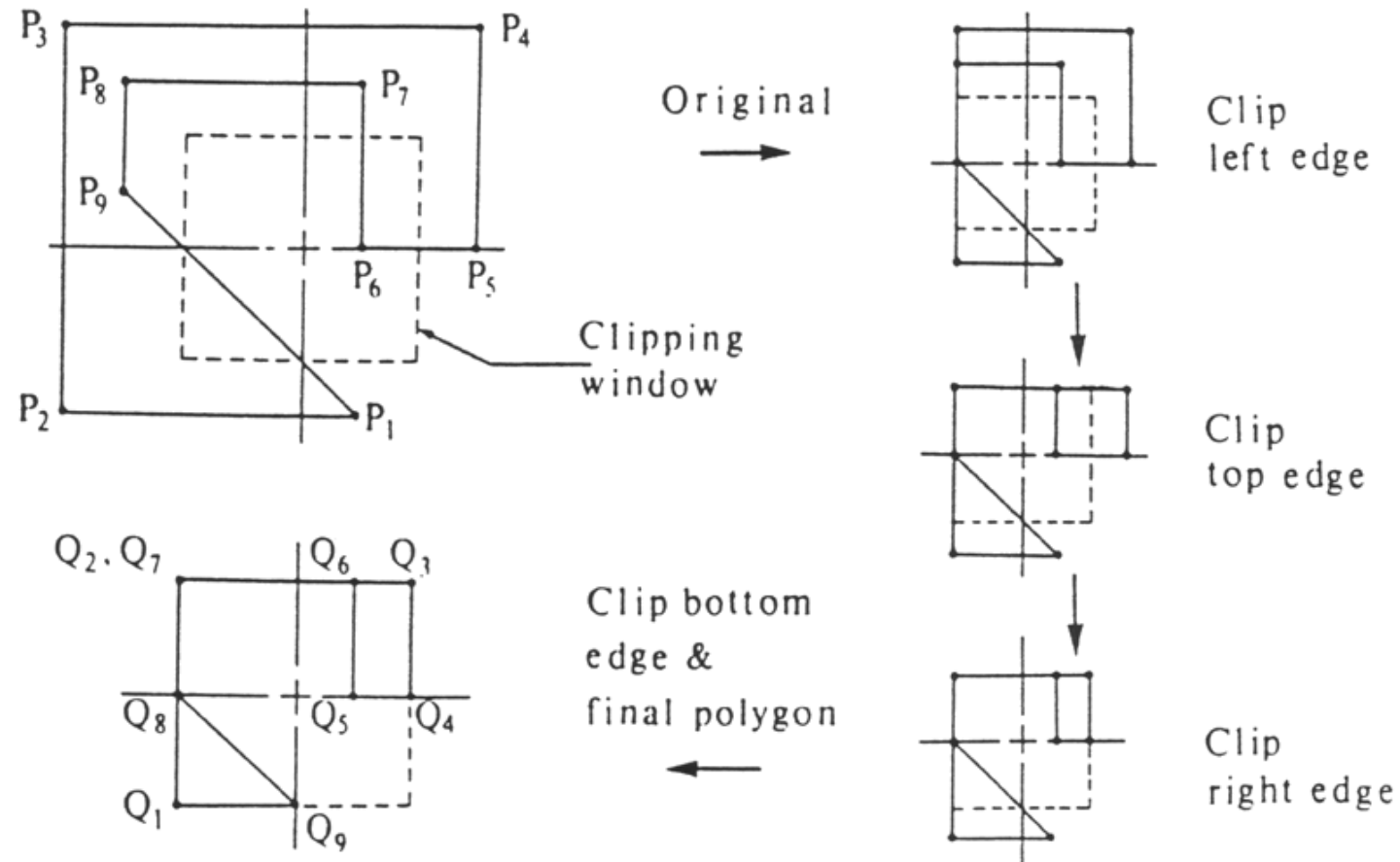
7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus



7.2 Clipping

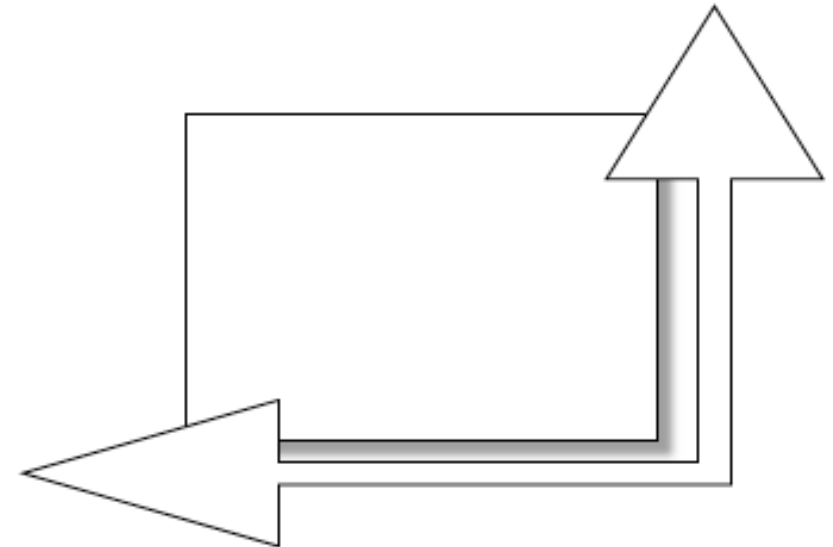
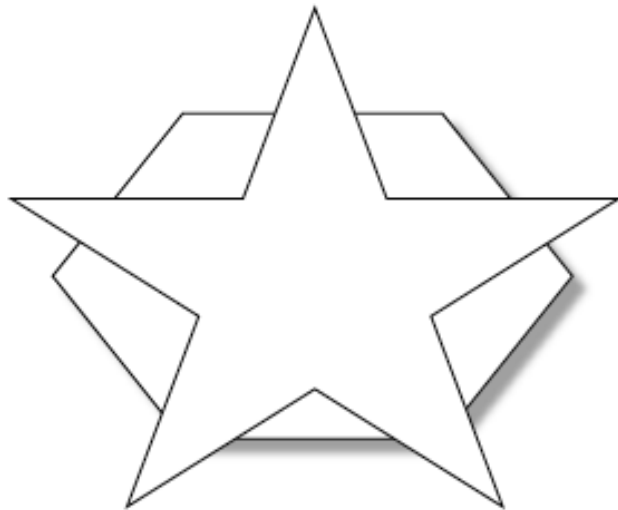
Sutherland-Hodgman Polygon-Clipping Algorithmus



7.2 Clipping

Sutherland-Hodgman Polygon-Clipping Algorithmus

- Fenster konvexes Polygon und
 - Kandidat einfaches Polygon
- ⇒ Ergebnis ist immer ein geschlossener Kantenzug



7.2 Clipping

Andere Polygon-Clipping-Verfahren

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

7.2 Clipping

Clipping in 3D

- 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
→ w-Clipping