

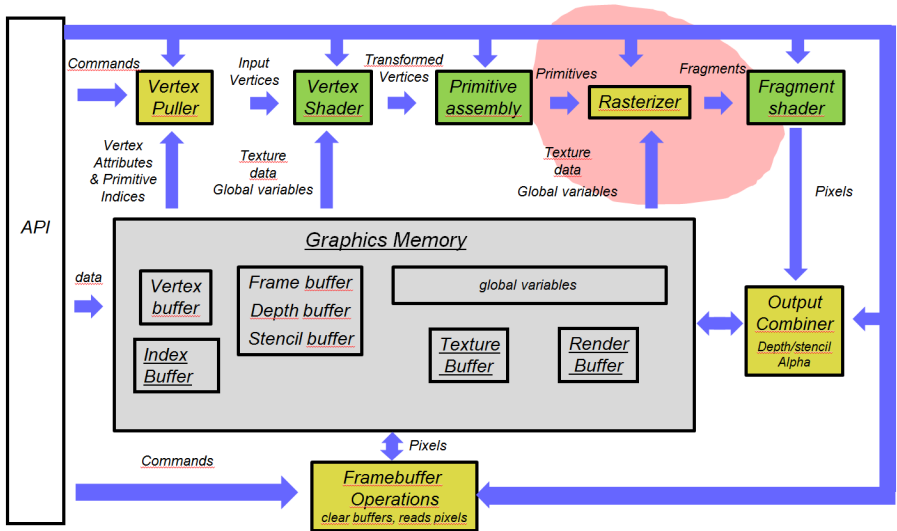
Einführung in die Computergrafik

Kapitel 8: Clipping

Prof. Dr. Matthias Hullin

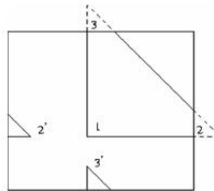
Institut für Informatik
Abteilung 2: Visual Computing
Universität Bonn

16. Mai 2020



Was ist Clipping und warum muss Clipping betrieben werden?

- ▶ Üblicherweise wird ein **rechteckiges Fenster (engl. Window)** definiert, das den Bildausschnitt begrenzt
- ▶ Um ein fehlerfreies Bild zu erhalten, muss die außerhalb liegende Bildinformation vor der Ausgabe abgeschnitten werden (**Clipping = Abschneiden**)
- ▶ Wird das Fenster (Window) auf die gesamte zur Verfügung stehende Fläche abgebildet, so erzeugen Bildelemente außerhalb des Fensters einen **Überlauf (Wraparound)** der Koordinatenadressierung



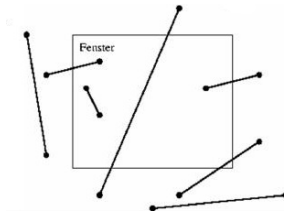
Wraparound

- ▶ Sehr oft will man den Bildschirm in verschiedene **Darstellungsflächen** (engl. **Viewports**) aufteilen
- ▶ Ohne Clipping würden die Inhalte der einzelnen Viewports sich gegenseitig beeinflussen, also falsche Bilder erzeugen, etc.
→ Clipping in den meisten Anwendungen unumgänglich
- ▶ Auf dem Bildschirm wird als Koordinatensystem das Bildschirmkoordinatensystem verwendet und das Ausgabefenster in diesem angegeben und Viewport genannt

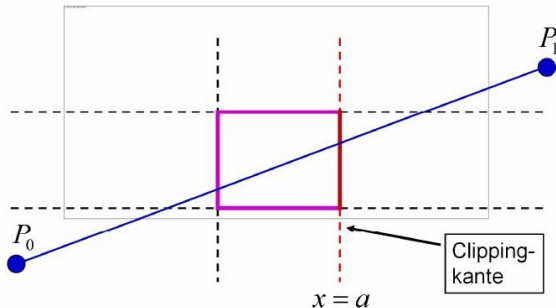


Abbildung: Screenshot mit 4 Viewports aus Mario Kart 8.

- ▶ Beim Clipping von Liniensegmenten an einem rechteckigen Fenster (allgemeiner konvexen Objekt) erkennt man, dass bei Unterteilung einer Geraden in sichtbare und unsichtbare Teile **nur ein sichtbarer Teil** entstehen kann.
- ▶ Linien, bei denen **beide** Endpunkte oberhalb, unterhalb, rechts oder links des Fensters liegen, sind völlig unsichtbar
→ können direkt aussortiert werden
- ▶ Dieses Aussortieren bewirkt eine erhebliche Beschleunigung des Clippings
→ Der Cohen-Sutherland-Algorithmus nutzt diese Eigenschaft



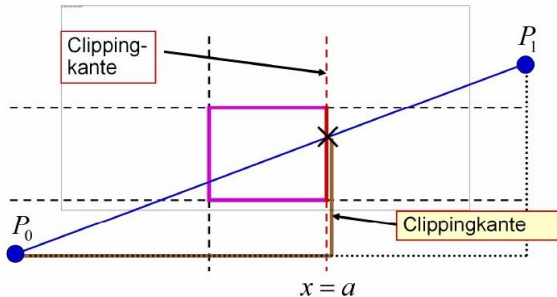
Gegeben: Linie von P_0 nach P_1 und Clippingkante $x = a$



Die Linie kann in Parameterform geschrieben werden:

$$P(t) = P_0 + t \cdot (P_1 - P_0) \quad 0 \leq t \leq 1$$

Um den Schnittpunkt zu berechnen, verwenden wir den Strahlensatz.



Da P_1 bei $t = 1$ liegt, liegt also der Schnittpunkt bei

$$t' = \frac{a - x_0}{x_1 - x_0}$$

Sonderfälle:

- kein Schnittpunkt, wenn $t' < 0$ oder $t' > 1$ ist
- vertikale und horizontale Linie hat keinen Schnittpunkt

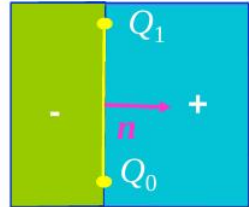
Wir wollen obigen Ansatz verallgemeinern:

Jede Clippingkante wird als **implizite Gerade** dargestellt:

$$Q_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \quad Q_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$n = \begin{pmatrix} \Delta y \\ -\Delta x \end{pmatrix} = \begin{pmatrix} y_1 - y_0 \\ x_0 - x_1 \end{pmatrix}$$

$$E(P) = \langle n|P \rangle - \langle n|Q_0 \rangle$$

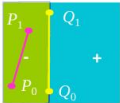
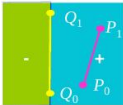
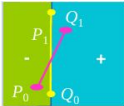


Diese Formel gibt an, auf welcher Seite sich der Punkt P befindet.

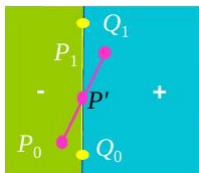
Ist $E(P) = 0$, so liegt P auf der Geraden.

Konvention: Normale zeigt ins Innere

Nun gibt es drei Fälle zu unterscheiden:

Fall	Liang-Barsky	
P_0, P_1 liegen außen	$E(P_0) \leq 0, E(P_1) \leq 0$	
P_0, P_1 liegen innen	$E(P_0) \geq 0, E(P_1) \geq 0$	
P_0, P_1 liegen auf versch. Seiten	$E(P_0) < 0, E(P_1) > 0$ bzw. $E(P_0) > 0, E(P_1) < 0$	

Im dritten Fall müssen wir also den Schnittpunkt P' berechnen:



Setzen wir also die parametrische Liniengleichung in die implizite Gleichung ein und nutzen die Linearität der Skalarprodukts aus, dann folgt:

$$\begin{aligned} E(P_0 + t \cdot (P_1 - P_0)) &= 0 \\ \Leftrightarrow \langle n | P_0 + t \cdot (P_1 - P_0) \rangle - \langle n | Q_0 \rangle &= 0 \\ \Leftrightarrow t &= \frac{\langle n | Q_0 \rangle - \langle n | P_0 \rangle}{\langle n | P_1 - P_0 \rangle} \end{aligned}$$

Somit erhalten wir den Schnittpunkt

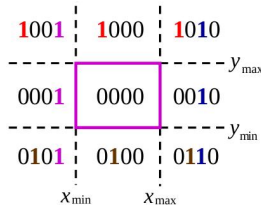
$$P' = P_0 + \frac{\langle n | Q_0 \rangle - \langle n | P_0 \rangle}{\langle n | P_1 - P_0 \rangle} \cdot (P_1 - P_0)$$

Der Liang-Barsky-Algorithmus verallgemeinert auf 3D:

- ▶ Halbräume sind nun durch Ebenen definiert.
- ▶ Ebene wird auch in impliziter Form durch Punkt und Normale beschrieben.
- ▶ Parametrische Form für das Liniensegment bleibt unverändert.

Die Idee des Cohen-Sutherland-Algorithmus ist das schnelle Identifizieren von Trivialfällen:

1. Jedem Liniensegmentendpunkt wird entsprechend seiner Lage in einer der 9 Regionen, die durch die Fensterbegrenzungen gebildet werden, ein 4-bit-Code zugeordnet (**Region Outcodes**).



Bedeutung:

- ▶ Bit 1 : über dem Fenster
- ▶ Bit 2 : unter dem Fenster
- ▶ Bit 3 : rechts vom Fenster
- ▶ Bit 4 : links vom Fenster

2. Prüfe Vorzeichen

- ▶ Bit 1 $\leftarrow 1$ if $y > y_{max}$, 0 otherwise
- ▶ Bit 2 $\leftarrow 1$ if $y < y_{min}$, 0 otherwise
- ▶ Bit 3 $\leftarrow 1$ if $x > x_{max}$, 0 otherwise
- ▶ Bit 4 $\leftarrow 1$ if $x < x_{min}$, 0 otherwise

3. Klassifikation der Endpunkte anhand deren Outcodes. Ein Liniensegment liegt

- ▶ ... völlig innerhalb des Fensters, wenn der Code für beide Endpunkte Null ist.
- ▶ ... außerhalb des Fensters, wenn der Durchschnitt (logisches UND) der Codes beider Endpunkte verschieden von Null ist.

d.h.

- ▶ $C_0 \wedge C_1 \neq 0000 \Rightarrow$ komplett außerhalb
- ▶ $C_0 \vee C_1 = 0000 \Rightarrow$ komplett innerhalb
- ▶ $C_0 \wedge C_1 = 0000 \Rightarrow$ **eventuell innerhalb**

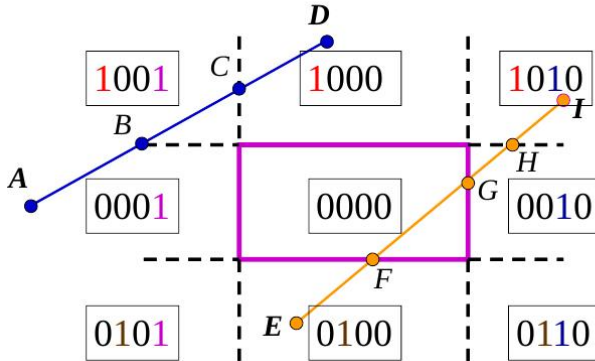
4. Clippe an Kante i , für die $(C_0 \vee C_1)_i \neq 0$

5. Ermittle neue Outcodes (eventuell wiederholt)

Falls $C_0 \wedge C_1 = 0000$, so wird in der zweiten Stufe des Algorithmus der Schnittpunkt des Liniensegments mit einer geeigneten Fensterbegrenzung berechnet.

- ▶ Jede Schnittpunktberechnung zerlegt das Liniensegment in zwei Teile, die wieder nach Stufe 1 behandelt werden. Dabei kann jeweils ein außerhalb des Fensters liegender Teil beseitigt werden.
- ▶ Wird der verbleibende Teil weder als völlig innerhalb noch als völlig außerhalb des Fensters erkannt, so wird Stufe 2 mit einer anderen Fensterbegrenzung durchgeführt.
- ▶ Die tatsächlich erforderlichen Schnittpunktberechnungen ergeben sich durch Vergleich der Outcodes der Endpunkte. Bei ungleichem Wert in einer Bitstelle wird mit der entsprechenden Fensterbegrenzung geschnitten.

Beispiel:



Initiale Outcodes:

$$OC(D) = 1000, OC(A) = 0001$$

$$OC(E) = 0100, OC(I) = 1010$$

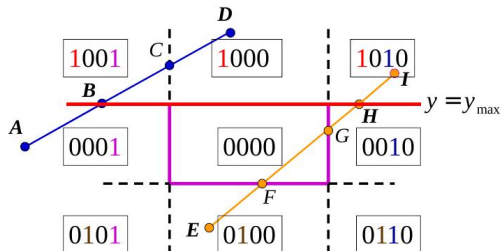
$$\blacktriangleright 1000 \wedge 0001 = 0000$$

$$\blacktriangleright 0100 \wedge 1010 = 0000$$

$$\blacktriangleright 1000 \vee 0001 = \textcolor{red}{1}001$$

$$\blacktriangleright 0100 \vee 1010 = \textcolor{red}{1}110$$

Clipping an Kante $y = y_{\max}$:



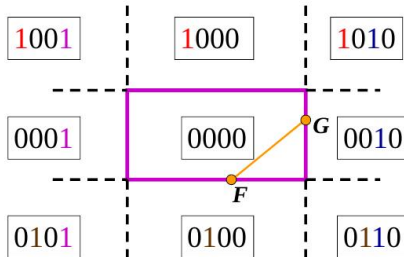
Neue Outcodes:

$$OC(F) = 0000, OC(G) = 0000$$

$$\blacktriangleright 0000 \wedge 0000 = 0000$$

$$\blacktriangleright 0000 \vee 0000 = 0000$$

Ergebnis:

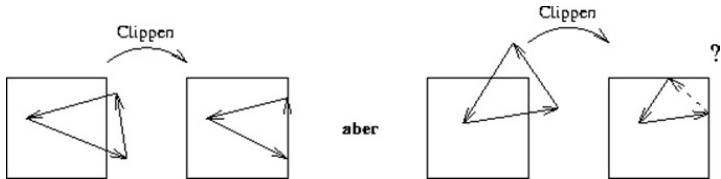


Bewertung: Wann ist der Algorithmus gut?

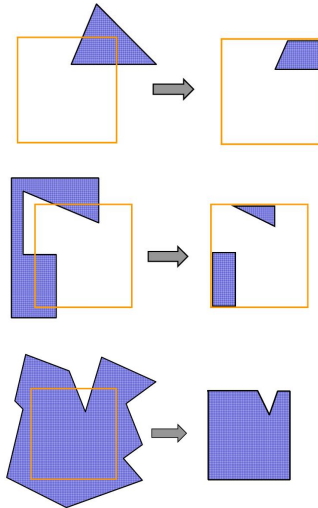
- ▶ Wenn er in den meisten Fällen trivial akzeptiert oder verwirft
- ▶ Wenn das Fenster groß im Verhältnis zu den Linien ist
- ▶ Wenn das Fenster klein im Verhältnis zu den Linien ist

Er funktioniert also in den Extremfällen sehr gut, wenn die meisten Entscheidungen trivial sind.

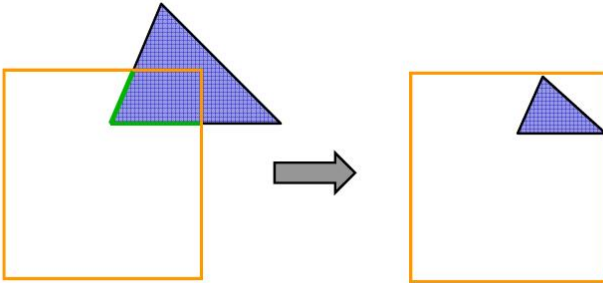
Ein Clipping-Algorithmus für geschlossene Polygone muß als Ergebnis des Clippingvorgangs wieder geschlossene Polygone liefern. Dies ist nur durch richtige Einbeziehung von Teilen der Fensterbegrenzung in das *geclippte* Polygon möglich.



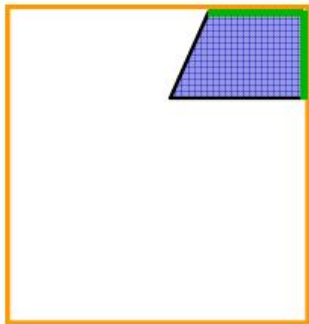
Probleme entstehen, wenn das zu *clippende* Polygon Ecken des Fensters umschließt. Verschiedene Methoden wurden vorgeschlagen, um diese Sonderfälle behandeln zu können. Die einfachste Methode ist die Umkehrung der Schleifenschachtelung: Das gesamte Polygon wird zunächst an einer Fenstergrenze geclippt, anschließend wird an der nächsten Fenstergrenze geclippt, etc.. Dies ist die wesentliche Idee des Sutherland-Hodgman-Algorithmus.



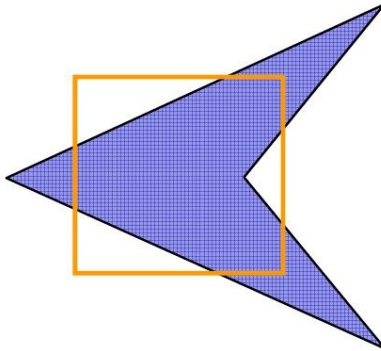
Wichtig: Polygon Clipping \neq Clipping von Liniensegmenten

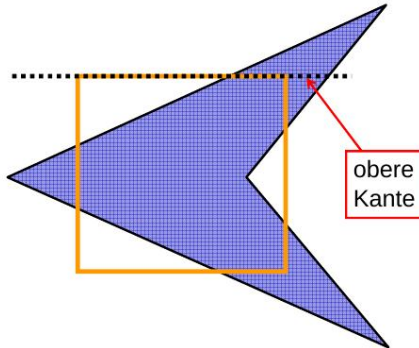


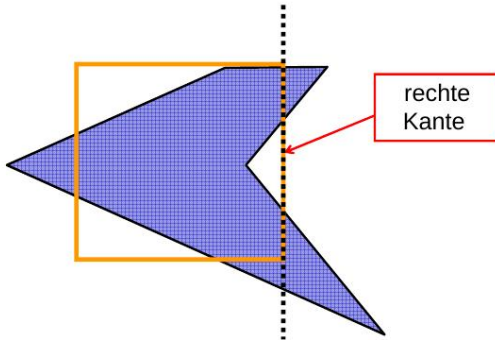
Ein Algorithmus muss also als Ergebnis wieder ein geschlossenes Polygon liefern.

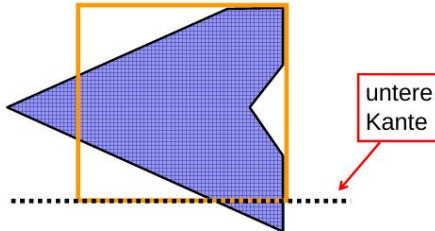


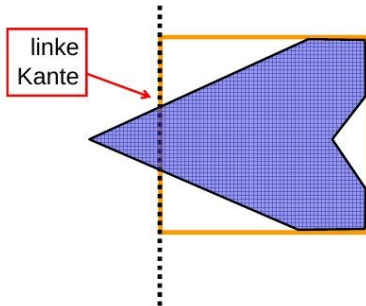
Ein einfacher Algorithmus dafür ist der [Sutherland-Hodgeman-Algorithmus](#) .

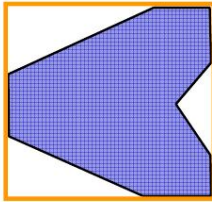






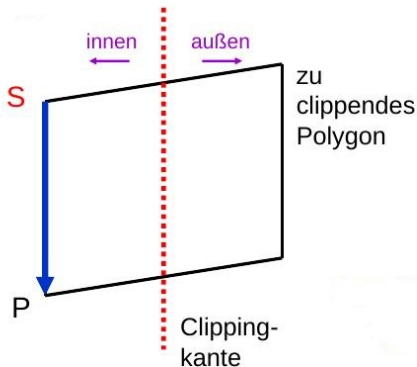






Idee: Wir definieren einen beliebigen Eckpunkt des Polygons als Startpunkt und laufen dann die Kanten des Polygons entlang.

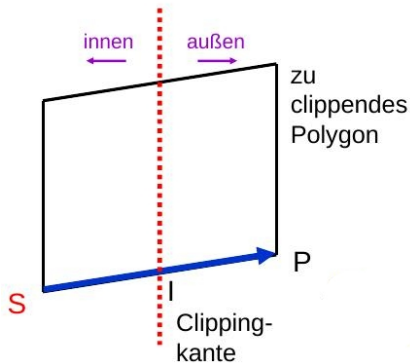
1. Fall: S und P innen



Eingabe: $(S) \rightarrow P$

Ausgabe: $\rightarrow P$

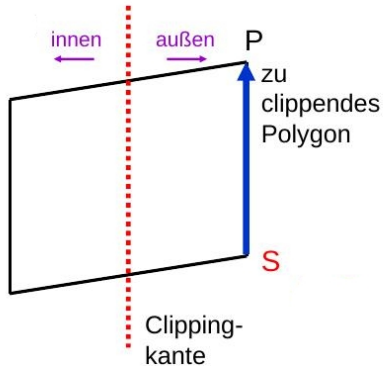
2. Fall: S innen und P außen



Eingabe: $(S) \rightarrow P$

Ausgabe: $\rightarrow I$

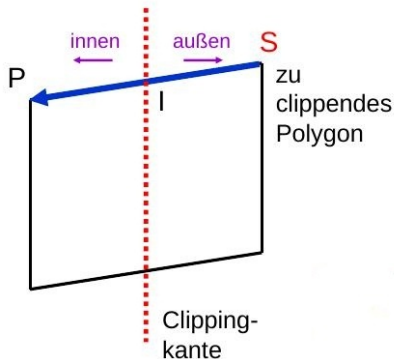
3. Fall: S und P außen



Eingabe: $(S) \rightarrow P$

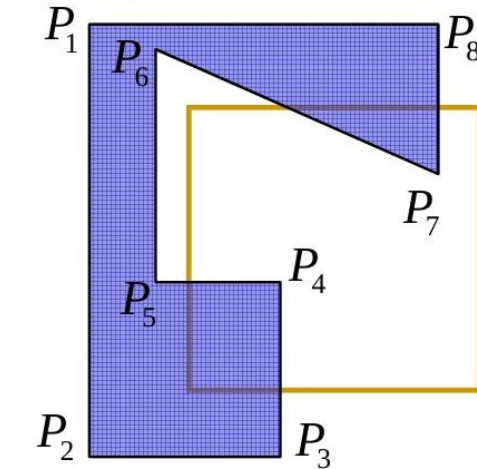
Ausgabe: keine

4. Fall: S außen und P innen

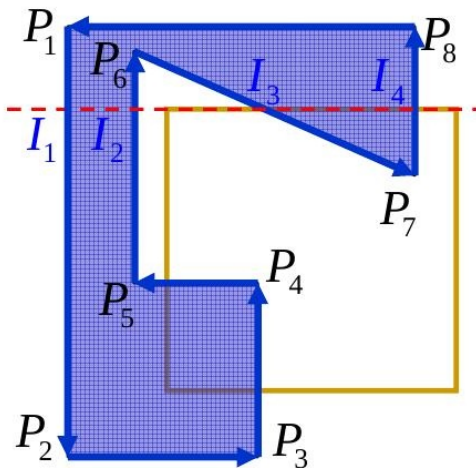


Eingabe: $(S) \rightarrow P$

Ausgabe: $\rightarrow I \rightarrow P$

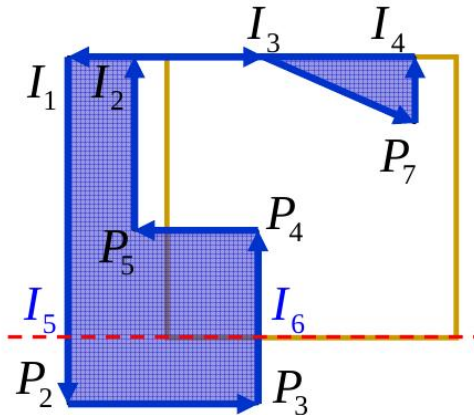


Eingabe: P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8



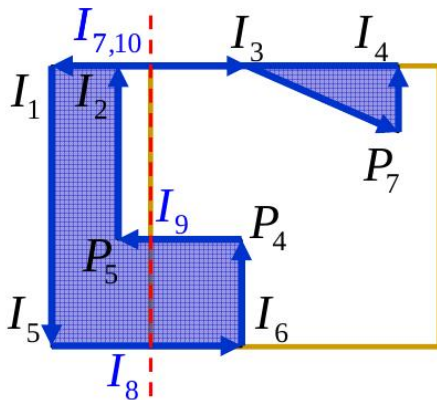
Eingabe: $P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8$

Ausgabe: $I_1 P_2 P_3 P_4 P_5 I_2 I_3 P_7 I_4$



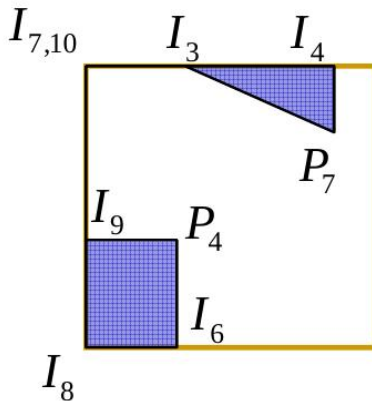
Eingabe: I_1 P_2 P_3 P_4 P_5 I_2 I_3 P_7 I_4

Ausgabe: I_1 I_5 I_6 P_4 P_5 I_2 I_3 P_7 I_4



Eingabe: $I_1 I_5 I_6 P_4 P_5 I_2 I_3 P_7 I_4$

Ausgabe: $I_7 I_8 I_6 P_4 I_9 I_{10} I_3 P_7 I_4$

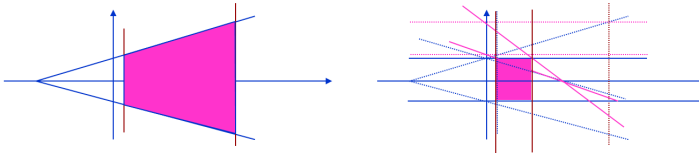


Ausgabe und Ergebnis: I_7 I_8 I_6 P_4 I_9 I_{10} I_3 P_7 I_4

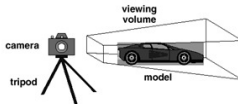
Ist das Sichtvolumen ein Einheitswürfel, so kann ein auf 3D erweiterter Cohen-Sutherland-Algorithmus zum Clipping verwendet werden. Ist das Sichtvolumen eine Pyramide, so kann der Liang-Barsky-Algorithmus angewendet werden.

Dieses Clipping hat folgende Nachteile:

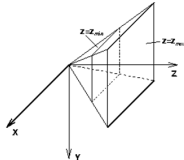
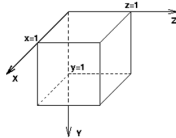
- Für parallele und perspektivische Projektionen wird an unterschiedliche Sichtvolumen geclippt. Man benötigt also verschiedene Algorithmen.
- Da das Clipping in affinen Koordinaten durchgeführt wird, dürfen bis zum Clipping nur affine Transformationen verwendet werden. Die Perspektive muß deshalb in einem separaten Schritt danach berechnet werden.



Der interessierende Teil der Bildinformation durch das Sichtvolumen (view volume) begrenzt. Im Fall der Parallelprojektion ist das Sichtvolumen ein in Projektionsrichtung unendlich ausgedehnter Quader, im Fall der perspektivischen Projektion eine Pyramide. Beide Volumina werden aus praktischen Gesichtspunkten durch eine vordere und hintere Ebene begrenzt. Die oben beschriebenen Verfahren für den ebenen Fall können sinngemäß auf 3D erweitert werden. Um die Berechnungen beim Clipping zu vereinfachen, wird an normierten Sichtvolumen geclippt.



$$\begin{aligned}x &= 0, x = 1 \\y &= 0, y = 1 \\z &= 0, z = 1\end{aligned}$$



$$\begin{aligned}x &= z, x = -z \\y &= z, y = -z \\z &= z_{\min}, z = z_{\max}\end{aligned}$$

Dieses Clipping hat folgende Nachteile:

- ▶ Für parallele und perspektivische Projektionen wird an unterschiedlichen Sichtvolumen geclippt. Man benötigt also verschiedene Algorithmen.
- ▶ Da das Clipping in affinen Koordinaten durchgeführt wird, dürfen bis zum Clipping nur affine Transformationen verwendet werden. Die Perspektive muß deshalb in einem separaten Schritt danach berechnet werden.

Das Ziel ist jedoch eine Darstellung, die auch die Perspektive einschließt.