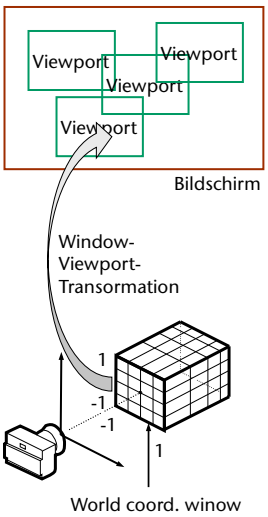
Computergraphik I
Clipping

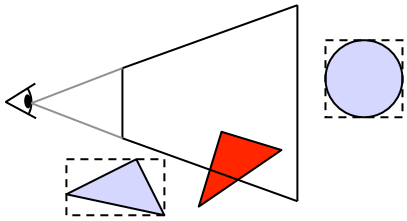
G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de

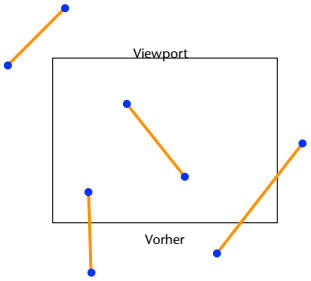
Ausschnittsbildung (Windowing und Clipping)

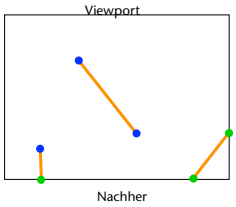
- Viewport = Ausgabefenster auf dem Bildschirm
  - Wird in Bildschirmkoordinaten spezifiziert
- World coordinate window = Fenster in die Szene (meist einfach "Window")
  - Wird in Weltkoordinaten spezifiziert
- Transformation zwischen Weltkoordinatensystem und Bildschirmkoordinatensystem → Window-Viewport-Transformation
- Ohne Clipping würden die einzelnen Viewports sich gegenseitig überschreiben



Unterschiede: Culling und Clipping



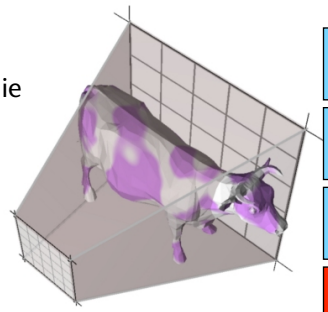




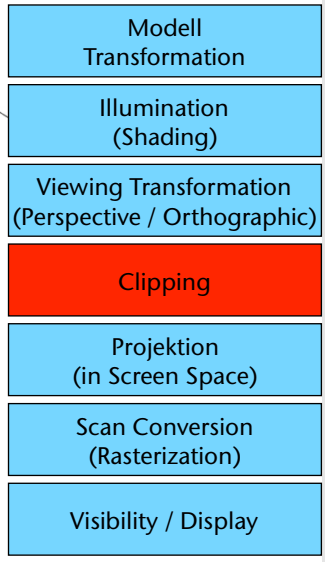
- Culling = Ausschluss ganzer Objekte (oft über *bounding volumes)*
  - Resultat = Ja / Nein (Entscheidungsproblem)
- Clipping = teilweise sichtbare Objekte (Linien / Polygone) müssen gegen Window / Viewport geclippt werden
  - Resultat = maximales Teil-Objekt, das vollständig im Window liegt (Konstruktionsproblem)

Station in der Graphik-Pipeline

- Abschneiden der Objekte, die außerhalb des sichtbaren Bereiches liegen (*view frustum*)

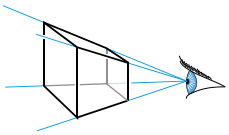


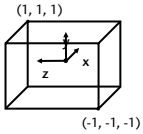
- Frühere Graphik-Hardware führte vollständiges Clipping durch — moderne Hardware "kürzt ab"
- Trotzdem sinnvoll, Clipping-Algos kennenzulernen, da oft wiederkehrendes Problem

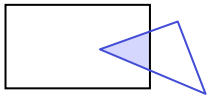


Wann sollte geclippt weden?

- Vor der Kamera-Transformation im 3D Raum
  - Benötigt 6 allgemeine Ebenengleichungen
- In homogenen Koordinaten nach der Kamera-Transformation im 4D, bevor durch die Perspektive geteilt wird (*Clip space*)
  - ergibt ungewöhnliche w-Werte
  - ist tatsächlich am einfachsten zu implementieren
- Im perspektivisch transformierten 3D-Screen-Raum
  - Problem: Objekte in der Kameraebene
- Nach der Projektion am Viewport in 2D
- Während der Rasterisierung für jedes Pixel

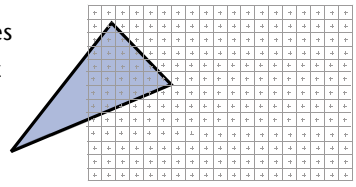






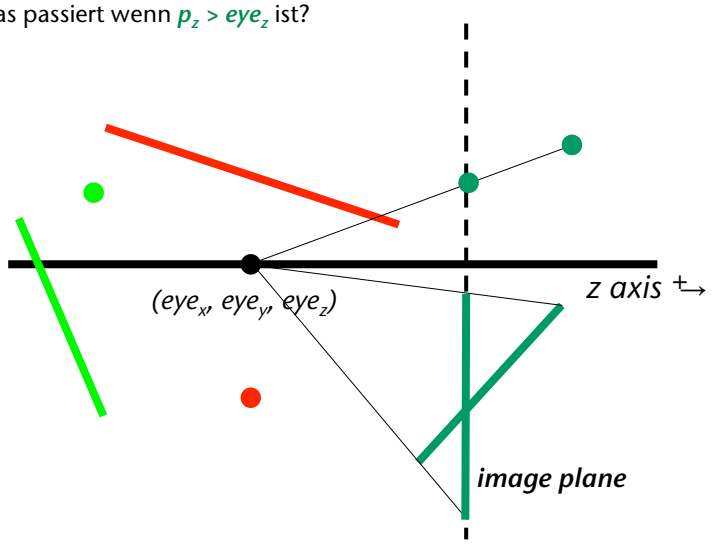
Naives Clipping

- Idee:
  - Führe Clipping während des Rasterisierens aus
  - Teste vor dem tatsächlichen Setzen eines Pixels, ob es innerhalb des Viewports ist
- Vorteil: funktioniert für beliebige Clipping-Windows (auch mit "Löchern")
- Nachteil: evtl. werden sehr viele Pixel ausgerechnet, die dann letztlich doch nicht gezeichnet werden (im worst-case alle)

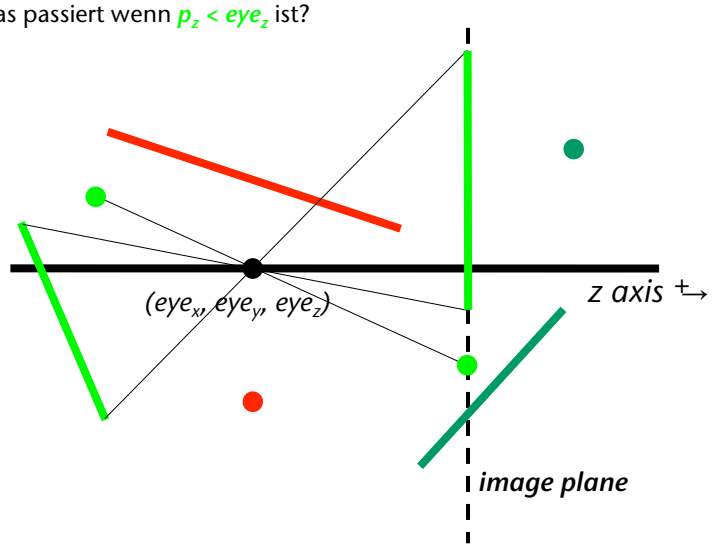


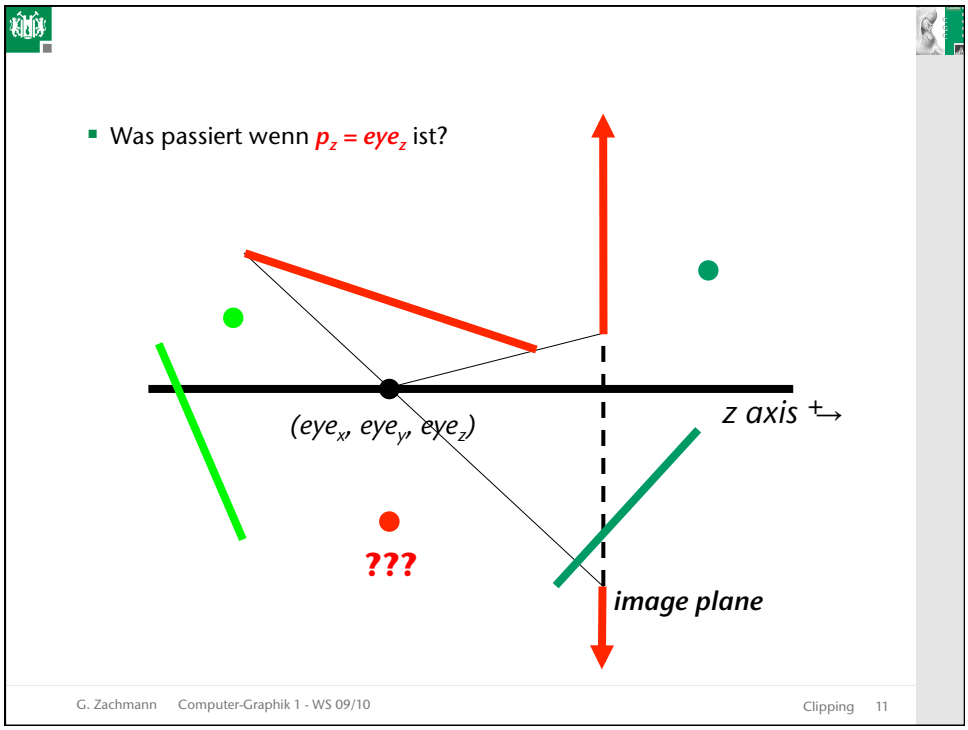
Motivation für Clipping vor der Projektion

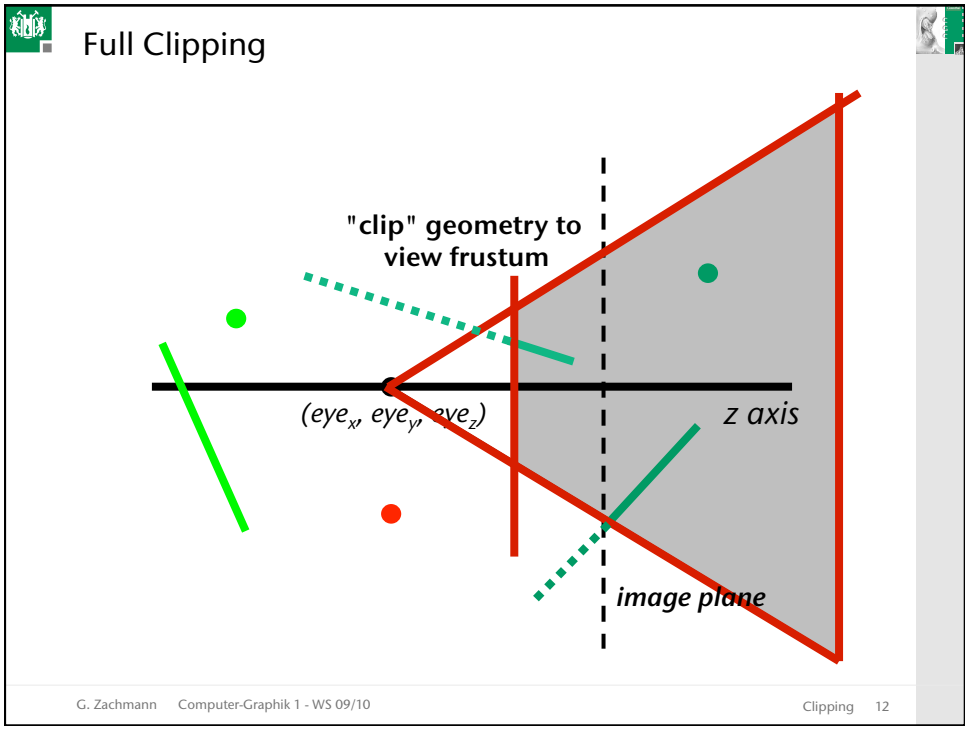
- Was passiert wenn $p_z > eye_z$ ist?



- Was passiert wenn $p_z < eye_z$ ist?



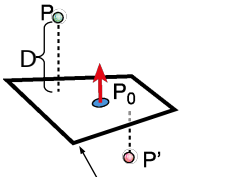

Was passiert wenn $p_z = eye_z$ ist?
(eye_x, eye_y, eye_z)
z axis
???
image plane
G. Zachmann Computer-Graphik 1 - WS 09/10 Clipping 11


Full Clipping
"clip" geometry to view frustum
(eye_x, eye_y, eye_z)
z axis
image plane
G. Zachmann Computer-Graphik 1 - WS 09/10 Clipping 12

Clipping in Bezug auf das View Frustum

- Erinnerung: homogene Ebenengleichung

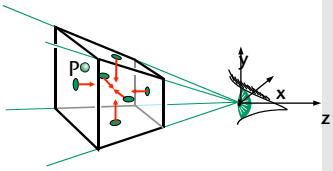
$$\mathbf{h} \cdot \mathbf{p} = 0$$



$\mathbf{h} = (n_x, n_y, n_z, -d)$

- Clipping eines Punktes:
  - Teste gegen alle 6 Ebenen
  - Annahme: Normalenvektoren sind nach innen gerichtet
  - Verwerfe Punkt P wenn für eine Ebene

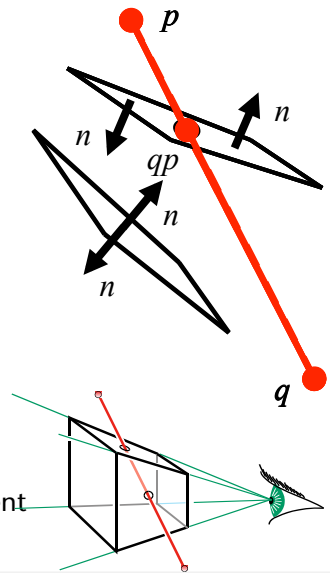
$$\mathbf{h} \cdot \mathbf{p} < 0$$



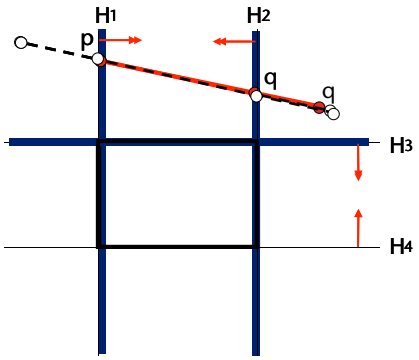
- Klappt für beliebige konvexe Polyeder

Clipping eines Linien-Segmentes

- 4 Fälle:
  1. Wenn $\mathbf{h} \cdot p > 0 \ \wedge \ \mathbf{h} \cdot q < 0$
     - Ersetze q
  2. Wenn $\mathbf{h} \cdot p < 0 \ \wedge \ \mathbf{h} \cdot q > 0$
     - Ersetze p
  3. Wenn $\mathbf{h} \cdot p > 0 \ \wedge \ \mathbf{h} \cdot q > 0$
     - "pass through"
  4. Wenn $\mathbf{h} \cdot p < 0 \ \wedge \ \mathbf{h} \cdot q < 0$
     - Komplett verwerfen ("reject")
- Für das ganze Frustum: alle Ebenen auf diese Weise durchlaufen
- Das Ergebnis ist ein einzelnes Segment (warum?)



Ist dieses Clipping effizient?



- Was ist das Problem?
  - Die Berechnung der Schnittpunkte und aller dazugehörigen interpolierten Werte sind — in diesem Fall — unnötig
- Kann man dies früher erkennen?

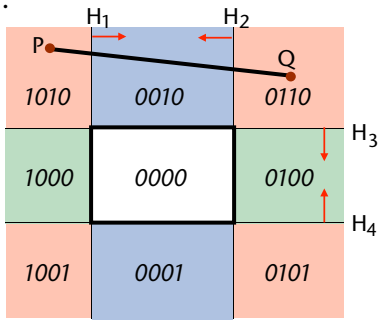
Erhöhung der Effizienz: Outcodes

- Berechne die "Sidedness" jedes Vertex bzgl. jeder Ebene
  - 0 = "richtige" Seite (Vorderseite); 1 = "falsche" Seite (Rückseite)
  - Ergibt pro Vertex einen 6-Bit langen *Outcode* (4 Bit im 2D)
- Bedingung:

$$\text{out}(P) \wedge \text{out}(Q) \neq 0 \rightarrow \text{"trivial reject"}$$

Bitweises UND!!

- Bsp.:

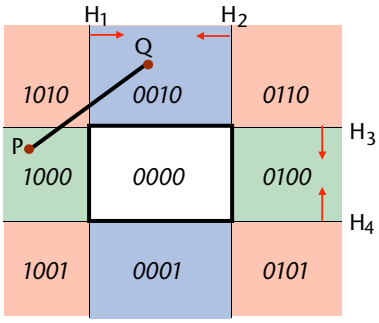


Outcode von P : 1010
Outcode von Q : 0110

AND : 0010

→ "trivial reject", da ≠ 0

- Weiteres Beispiel:



Outcode von P : 1000
Outcode von Q : 0010

AND : 0000

→ "potentially visible"

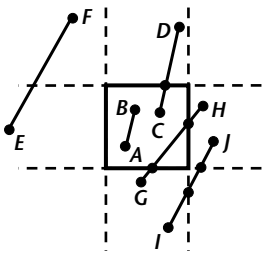
- In diesem Fällen macht der Test also keine Aussage!
- Dies ist der sog. Cohen-Sutherland-Algorithmus

- Weitere Aussage auf Basis der Outcodes:

$$\text{out}(P) \vee \text{out}(Q) = 0 \rightarrow \text{"trivial accept"}$$

Bitweises ODER!!

- Beispiele:



Linie	out(A)	out(B)	AND	OR
AB	0000	0000	0000	0000
CD	0000	1000	0000	1000
EF	0001	1001	0001	1001
GH	0100	0010	0000	0110
IJ	0100	0010	0000	0110

Der Cohen-Sutherland-Algorithmus

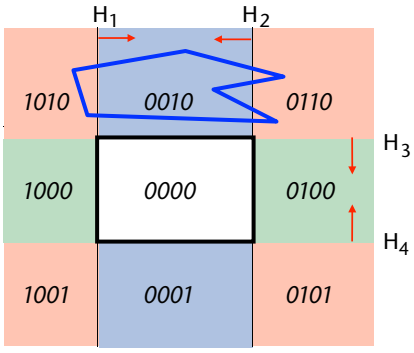
- Genereller Trick, der hier angewandt wird:
  - Erst einfache Tests durchführen, ob Clipping nötig ist
    - Die Tests liefern evtl. keine definitive Antwort, aber dafür sind sie sehr schnell
  - Dann erst im "nötigen" Fall die (teuren) mathematischen Operationen durchführen
- Der Code für die Outcodes:

```
unsigned int outcode( int x, int y )
{
  unsigned int c = 0;

  if ( y > ymax ) c = c | 8; // 1000
  if ( y < ymin ) c = c | 4; // 0100
  if ( x > xmax ) c = c | 2; // 0010
  if ( x < xmin ) c = c | 1; // 0001

  return c;
}
```


- Das Gute an dem Test: er funktioniert für beliebige Primitive in beliebigen Dimensionen mit beliebigen, konvexen Clip-Windows!
- Beispiel:



Outcode of p	: 1010
Outcode of q	: 1010
Outcode of r	: 0110
Outcode of s	: 0010
Outcode of t	: 0110
Outcode of u	: 0010
AND	: 0010

→ Clipped

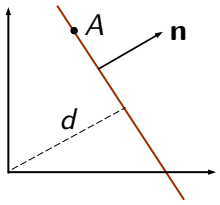
Clipping in homogenen Koordinaten

- Erinnerung:
  - Ein Punkt $P = (x, y, z) \in \mathbb{R}^3 \equiv$ Vektor $\hat{\mathbf{p}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4$ in homogene Koord.
  - Alle Vielfache des homogenen Vektors entsprechen demselben Punkt:

$$s \cdot \hat{\mathbf{p}} \equiv P$$

- Sei eine Ebene gegeben durch $\mathbf{n}$, A
- Dann liegt P in der Ebene $\Leftrightarrow$

$$(P - A) \cdot \mathbf{n} = P \cdot \mathbf{n} - \underbrace{A \cdot \mathbf{n}}_{d} \equiv \hat{\mathbf{p}} \cdot \begin{pmatrix} \mathbf{n}_x \\ \mathbf{n}_y \\ \mathbf{n}_z \\ -d \end{pmatrix} = 0$$



- Der Vektor $\hat{\mathbf{n}} = (\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z, -d)$ heißt auch homogene Darstellung der Ebene, oder homogene Normale

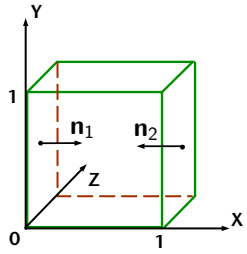
- Fazit: Punkt/Vektor im 3D ≡ Vektor im 4D
  Ebene im 3D ≡ Vektor im 4D
- Bemerkung: alle Vektoren $t \cdot \hat{\mathbf{n}}$ beschreiben die gleiche Ebene

$$\hat{\mathbf{p}} \cdot \hat{\mathbf{n}} = 0 = s\hat{\mathbf{p}} \cdot t\hat{\mathbf{n}}$$

Outcodes in homogenen Koordinaten

- Betrachte das Clip-Window (0,0,0) — (1,1,1)
  - Die homogene Darstellung der beiden Ebenen x=0 und x =1 ist

$$\mathbf{n}_1 = (1\ 0\ 0\ 0) \quad \mathbf{n}_2 = (-1\ 0\ 0\ 1)$$



- Definiere die *boundary distance (BD)*

$$d_i = \mathbf{p} \cdot \mathbf{n}_i\ , \quad \mathbf{p} = (x, y, z, w)$$

  - Berechnung ist trivial; z.B. $d_1 = x \qquad d_2 = w - x$
  - Erstelle Tabelle aller BDs →

BD	homog. Wert	Ebene
d_1	x	x=0
d_2	$w - x$	x=1
d_3	y	y=0
d_4	$w - y$	y=1
d_5	z	z=0
d_6	$w - z$	z=1

  - Outcodes sind auch trivial zu bestimmen:

$$_i(\mathbf{p}) = (d_i)$$

$$= \begin{cases} 1 & , P \text{ außerhalb Ebene } i \\ 0 & , P \text{ innerhalb Ebene } i \end{cases}$$

Trivial accept / reject

- Sei Linie PQ gegeben; in homogenen Koordinaten: **p**, **q**
- Wie bisher auch:
  - $\text{out}(\mathbf{p}) \wedge \text{out}(\mathbf{q}) \neq 000000 \Rightarrow$ trivial reject
  - $\text{out}(\mathbf{p}) \vee \text{out}(\mathbf{q}) = 000000 \Rightarrow$ trivial accept
- Ansonsten: es muß mindestens eine Bitposition i geben, wo $\text{out}_i(\mathbf{p}) = 0$ und $\text{out}_i(\mathbf{q})$ oder umgekehrt.
- Schneide X(t) mit dieser Clip-Window-Ebene:

$$X(t) = P_0 + t(Q - P) \ \leftrightarrow\ \hat{x}(t) = \mathbf{p} + t(\mathbf{q} - \mathbf{p})$$

$$\hat{x}(t) \cdot \mathbf{n}_i = [\mathbf{p} + t(\mathbf{q} - \mathbf{p})] \cdot \mathbf{n}_i = \mathbf{p}\mathbf{n}_i + t(\mathbf{q}\mathbf{n}_i - \mathbf{p}\mathbf{n}_i)$$

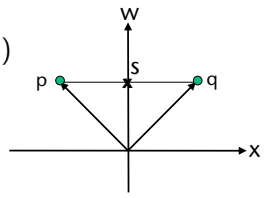
$$= d_i^p + t(d_i^q - d_i^p) \stackrel{!}{=} 0$$

$$t = \frac{d_i^p}{d_i^p - d_i^q}$$

- Bemerkung: $t \in \{0, 1\} \Leftrightarrow \text{sign}(d_i^p) \neq \text{sign}(d_i^q)$

Beispiel

- $P = (-1\ 0\ 0)$ $Q = (1\ 0\ 0)$ Ebene 1: $x = 0$
- $\mathbf{p} = (-1\ 0\ 0\ 1)$ $\mathbf{q} = (1\ 0\ 0\ 1)$ $\mathbf{n}_1 = (1\ 0\ 0\ 0)$
- Parameter des Schnittpunktes **s**:

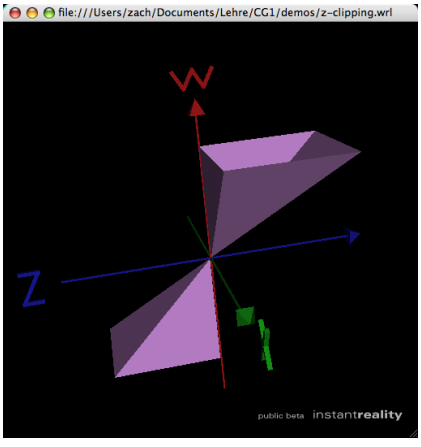


$$t = \frac{d_1^p}{d_1^p - d_1^q} = \frac{-1}{-1-(+1)} = \frac{1}{2}$$

- Schnittpunkt: $\mathbf{s} = \mathbf{p} + \frac{1}{2}(\mathbf{q} - \mathbf{p}) = (0\ 0\ 0\ 1)$
- Frage: was ist mit $\mathbf{p}' = (-2\ 0\ 0\ 2)$? (Ist derselbe Punkt P in 3D!)
  - Parameter $t' = \frac{2}{3}$
  - Ist das ein anderer Punkt?
  - Nein, denn Schnittpunkt

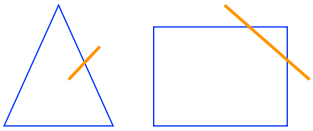
$$\mathbf{s}' = \mathbf{p}' + \frac{2}{3}(\mathbf{q} - \mathbf{p}') = \begin{pmatrix} -2 \\ 0 \\ 0 \\ 2 \end{pmatrix} + \frac{2}{3}\begin{pmatrix} 3 \\ 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1\frac{1}{3} \end{pmatrix} \equiv \mathbf{s}$$

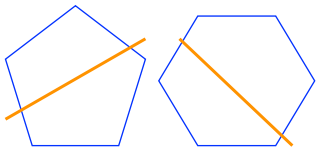
- Visualisierung der Clipping-Region im 4D:

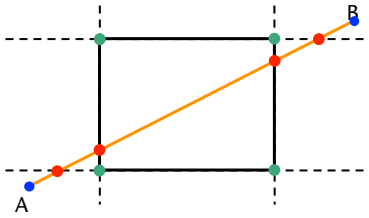


Der Cyrus-Beck-Algorithmus [1978]

- Clippen von Linien an beliebigen, konvexen Clip-Windows
- Verwendet die Parameterdarstellung der Linien
- Im Folgenden wird wieder das Rechteck als Beispiel verwendet; der Algorithmus ist aber für beliebige (konvexe) Clip-Windows anwendbar
- Bei einem n-Eck kann es bis zu n Schnittpunkte geben
- Nur 2 davon sind echte Schnittpunkte mit dem Clip-Window

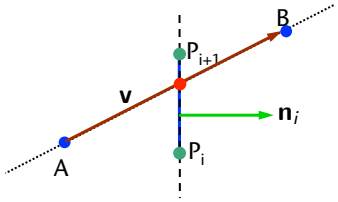






- Berechne für jeden Clipping-Rand *i* das zugehörige *t*:

$$t = -\frac{(A - P_i)\cdot \mathbf{n}_i}{\mathbf{v}\cdot \mathbf{n}_i}$$



- Die Werte $t<0$ und $t>1$ werden ignoriert
- Jeweils genau ein *t* markiert den Eintrittspunkt und den Austrittspunkt der Linie

Entering and Leaving t's

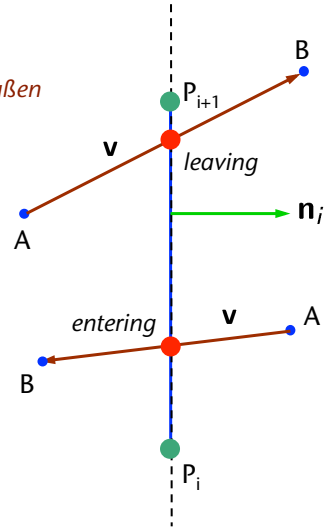
- Idee: klassifiziere alle t's
  - Konvention: Normalen zeigen nach *außen*
  - "Leaving", falls:

$$\mathbf{n}_i \cdot \mathbf{v} > 0 \Rightarrow t_i^l$$

  - "Entering", falls:

$$\mathbf{n}_i \cdot \mathbf{v} < 0 \Rightarrow t_i^e$$

  - Sonst: Sonderfall, der anderweitig abgefangen wird



Der Algorithmus

- Berechne das Maximum und Minimum

$$t_e = \max\{t_i^e, 0\} \qquad t_l = \min\{t_i^l, 1\}$$

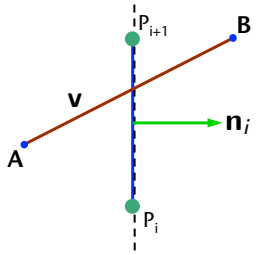
- Falls $t_e > t_l$ → Linie ist komplett außerhalb des Clip-Windows
- Sonst: t_e und t_l definieren die Enden der geclippten Linie

- Alternative Betrachtungsweise:
  - Starte mit Intervall [0,1]
  - Schneide Linie der Reihe nach gegen jeden Clip-Rand
  - Falls "entering" → schneide aktuelles Intervall unten ab (falls überhaupt)
  - Falls "leaving" → schneide aktuelles Intervall oben ab
  - Stop, falls Intervall leer wird

Sonderfälle

- Wann kann der Nenner Null werden?

$$t = -\frac{(A - P_i)\cdot \mathbf{n}_i}{\mathbf{v}\cdot \mathbf{n}_i}$$



- $\mathbf{v} = 0$:
  - Start- und Endpunkt der Linie sind identisch (muß vorher abgefangen werden)
- $\mathbf{n}_i = 0$:
  - Nur, falls 2 Punkte des Clip-Windows identisch sind, vorher abfangen
- $\mathbf{n}_i \cdot \mathbf{v} = 0$:
  - Zu zeichnende Linie ist parallel zu einer Kante des Clip-Objekts → kein t ausrechnen, nächsten Clip-Rand betrachten