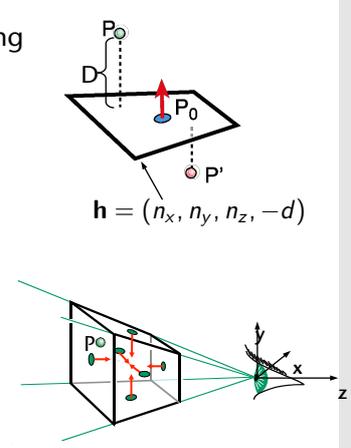


Clipping in Bezug auf das View Frustum

- Erinnerung: homogene Ebenengleichung

$$\mathbf{h} \cdot \mathbf{p} = 0$$
- Clipping eines Punktes:
 - Teste gegen alle 6 Ebenen
 - Annahme: Normalenvektoren sind nach **innen** gerichtet
 - Verwerfe Punkt P wenn für eine Ebene

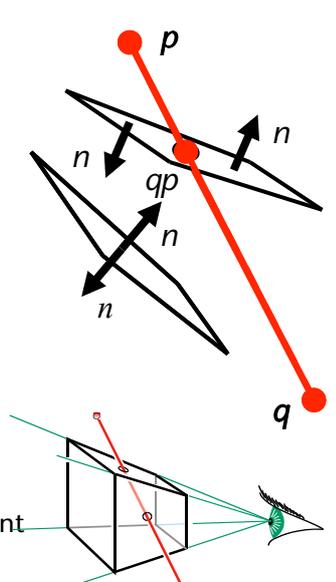
$$\mathbf{h} \cdot \mathbf{p} < 0$$
- Klappt für beliebige konvexe Polyeder



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 13

Clipping eines Linien-Segmentes

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



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 15

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?

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 17

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: $out(P) \wedge out(Q) \neq 0 \rightarrow$ "trivial reject"

Bitweises UND!!
- Bsp.:

P	H1	H2	Q	
1010	0010	0110		
1000	0000	0100		H3
1001	0001	0101		H4

Outcode von P : 1010

Outcode von Q : 0110

AND : 0010

\rightarrow "trivial reject", da $\neq 0$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 18

▪ Weiteres Beispiel:

Outcode von P : 1000
 Outcode von Q : 0010
 AND : 0000
 → "potentially visible"

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 19

▪ Weitere Aussage auf Basis der Outcodes:
 $out(P) \vee out(Q) = 0 \rightarrow$ "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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 20

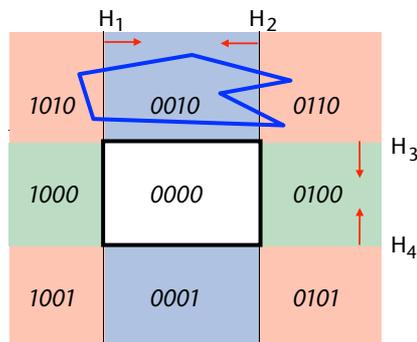
Der Cohen-Sutherland-Algorithmus

- 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

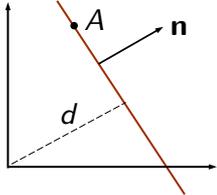
→ Clipped

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 homogenen 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

$$(P - A) \cdot \mathbf{n} = P \cdot \mathbf{n} - \underbrace{A \cdot \mathbf{n}}_d \equiv \hat{\mathbf{p}} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \\ -d \end{pmatrix} = 0$$
- Der Vektor $\hat{\mathbf{n}} = (n_x, n_y, n_z, -d)$ heißt auch **homogene Darstellung der Ebene, oder homogene Normale**



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 24

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

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 25

Outcodes in homogenen Koordinaten

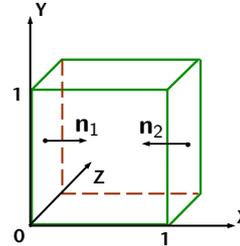
- Betrachte das Clip-Window $(0,0,0) - (1,1,1)$
 - Die homogene Darstellung der beiden Ebenen $x=0$ und $x=1$ ist

$$\mathbf{n}_1 = (1 \ 0 \ 0 \ 0) \quad \mathbf{n}_2 = (-1 \ 0 \ 0 \ 1)$$
- 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. $d_1 = x \quad d_2 = w - x$
 - Erstelle Tabelle aller BDs \rightarrow
- Outcodes sind auch trivial zu bestimmen:

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

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



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	$w - z$	$z=1$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 26

Trivial accept / reject

- Sei Linie PQ gegeben; in homogenen Koordinaten: \mathbf{p}, \mathbf{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:

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

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

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

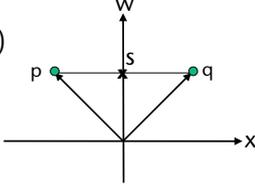
$$t = \frac{d_i^p}{d_i^p - d_i^q}$$
 - Bemerkung: $t \in \{0, 1\} \Leftrightarrow \text{sign}(d_i^p) \neq \text{sign}(d_i^q)$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 27

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



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 28

Visualisierung der Clipping-Region im 4D:



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 29

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 30

- 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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 31

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 32

Der Algorithmus

- Berechne das Maximum und Minimum

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

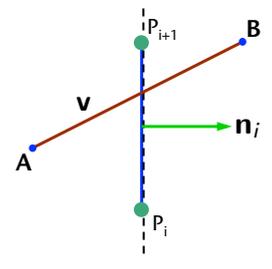
G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 33

Sonderfälle

- Wann kann der Nenner Null werden?

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

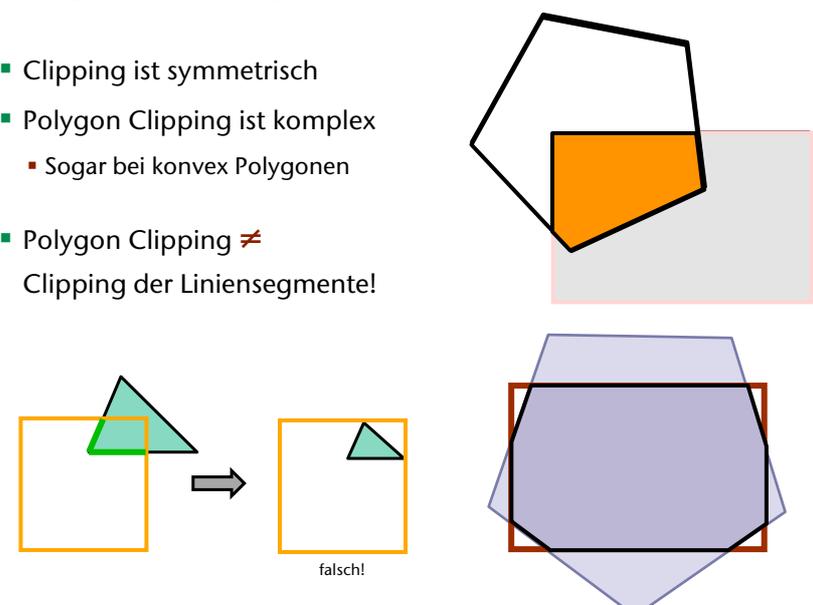
- $\mathbf{v} = 0$:
 - Start- und Endpunkt der Linie sind identisch (muß vorher abgefangen werden)
- $\mathbf{n}_i = 0$:
 - Nur, falls 2 Punkte des Clip-Windows identisch sind, vorher abfangen
- $\mathbf{n}_i \cdot \mathbf{v} = 0$:
 - Zu zeichnende Linie ist parallel zu einer Kante des Clip-Objekts →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



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 34

Polygon Clipping in 2D

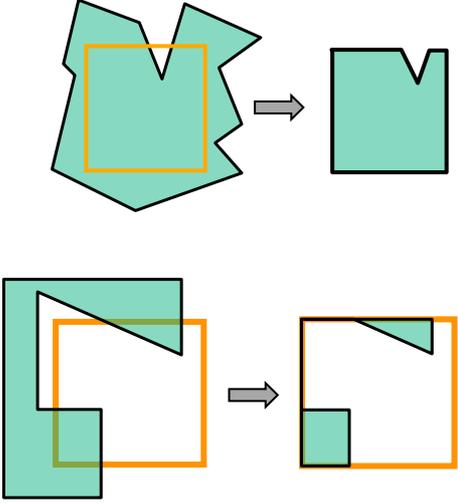
- Clipping ist symmetrisch
- Polygon Clipping ist komplex
 - Sogar bei konvex Polygonen
- Polygon Clipping \neq Clipping der Liniensegmente!



falsch!

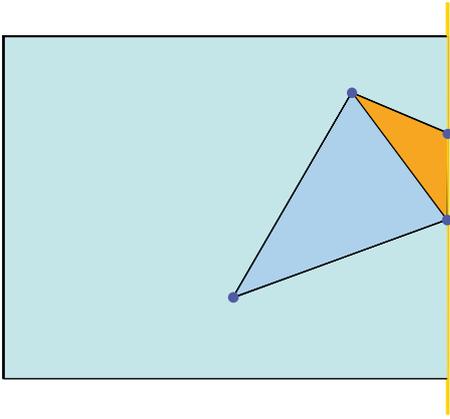
G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 35

- Polygon-Clipping kann, insbesondere bei nicht-konvexen Polygonen, unangenehm werden
- Es können sogar mehrere Polygone entstehen

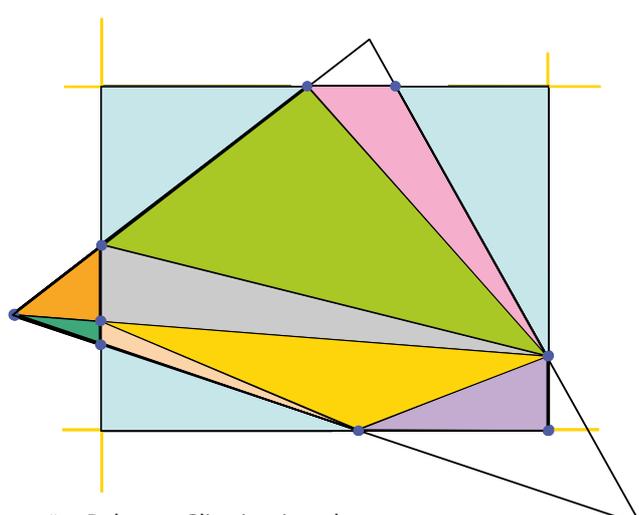


G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 36

Naiver Clipping-Algorithmus



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 37

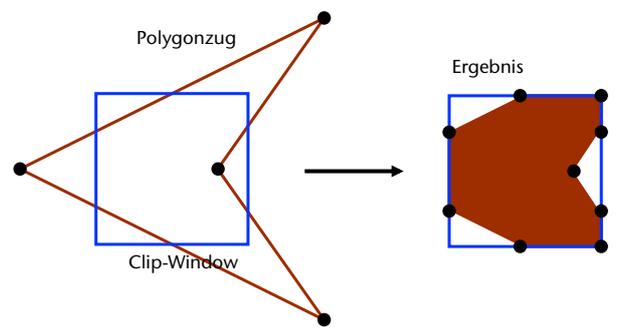


Fazit: das naive Polygon-Clipping ist schon für Dreiecke zu ineffizient (erzeugt viel zu viel Output)

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 38

Sutherland-Hodgman

- Clipping eines Polygonzugs gegen ein konvexes Clip-Polygon (z.B. Viewport); der Polygonzug darf konkav sein



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 39

Vorgehen

- Eingabe =
 - Liste der Eckpunkte in der richtigen Reihenfolge (gegen den Uhrzeigersinn)
 - Menge von Clip-Kanten, die ein konvexes Clip-Window definieren
- Betrachte eine (beliebige) Clip-Kante:
 - Nach dem Schnitt mit dieser Clip-Kante wird eine neue Liste von Eckpunkten erzeugt
 - Dieses Ergebnis ist wieder ein geschlossener Polygonzug
 - Alle Punkte des neuen Polygons befinden sich auf der "Innenseite" (der "richtigen" Seite) dieser Clip-Kante (Schleifeninvariante)
- Das wird mit allen Clip-Kanten wiederholt
 - (Im Prinzip ist die Reihenfolge der Clip-Kanten egal)

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 40

Der Polygonzug wird der Reihe nach an den Clip-Kanten geschnitten

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 41

4 Fälle

- Annahme: der Punkt A wurde bereits behandelt

Beide Punkte drinnen: → Output B

Linie „zeigt“ nach außen: → Output S

Linie „zeigt“ nach innen: → Output S, B

Beide Punkte draußen: → kein Output

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 42

Beispiel

Input: A B C D

Output: S₁ A B C S₂

Man beginnt mit der Kante [letzter Punkt – erster Punkt], hier also D-A.
(D wird am Ende der Schleife betrachtet)

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 43

Input: $S_1 A B C S_2$

Output: $S_3 S_1 A S_4 S_5 C S_6$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 44

- Kleines Problem des Sutherland-Hodgeman-Algos: falls das ursprüngliche Polygon in mehrere Teile zerfällt beim Clipping, dann entsteht eine unschöne Polygonkante am Rand des Windows
- Beispiel:

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 45

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 47

Etwas komplexeres Beispiel

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

Ausgabe:
 $l_1 P_2 P_3 P_4 P_5 l_2 l_3 P_7 l_4$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 48

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

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 49

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

Ausgabe:
 $I_1 I_5 I_6 P_4 I_9 I_{10} I_3 P_7 I_4$

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 50

Diagram illustrating the Weiler-Atherton clipping algorithm. A window (orange outline) is defined by vertices $I_{7,10}$, I_3 , I_4 , I_8 , and I_6 . A polygon (green fill) is defined by vertices P_4 , P_7 , and I_9 . The output sequence of vertices is $I_7 I_8 I_6 P_4 I_9 I_{10} I_3 P_7 I_4$.

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

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 51

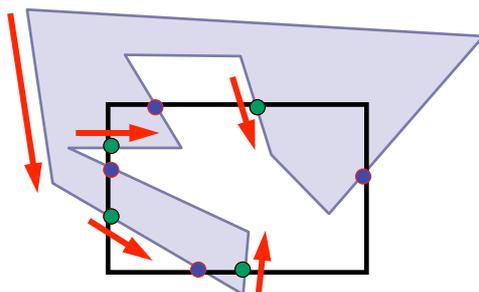
Der Algorithmus von Weiler-Atherton

- Strategie: "Wandere" auf dem Polygonzug oder dem Window-Rand
- Konvention (wie immer): Polygone sind CCW orientiert

Diagram illustrating the Weiler-Atherton clipping algorithm. A polygon (blue fill) is shown being clipped by a window (black outline). A red arrow indicates the direction of traversal along the polygon boundary, and a black arrow indicates the direction of traversal along the window boundary.

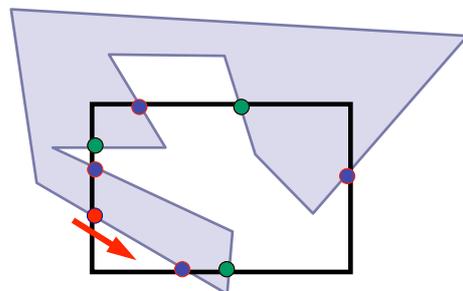
G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 56

- Berechne alle Schnittpunkte
- Markiere die Punkte, an denen das Polygon in das Clipping-Window eintritt (hier grün)



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 57

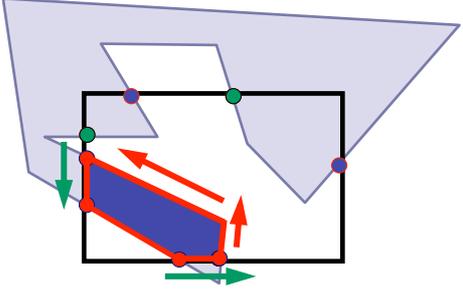
- Solange noch ein unbearbeiteter Eintrittsschnittpunkt vorhanden ist wird das Polygon weiter umlaufen
- Die Ausgabe ist (wie bei Sutherland-Hodgman) ein oder mehrere Listen von Punkten (Eckpunkte des Polygons, Schnittpunkte, und/oder Eckpunkte des Windows)
 - Unterschied: die Eingabe wird nur einmal abgearbeitet



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 58

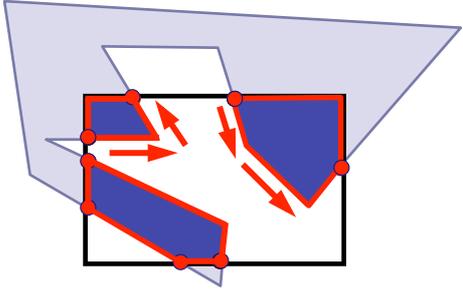
Umlauf-Regeln

- Trifft man beim Umlauf auf einen Schnittpunkt, dann:
 - Füge Schnittpunkt zur Ausgabe hinzu
 - Falls Schnittpunkt = "Out-to-in": folge dem Polygonzug (ccw)
 - Falls Schnittpunkt = "In-to-out": folge dem Window-Rand (ccw)



G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 59

- Solange noch ein unbearbeiteter Eintrittsschnittpunkt vorhanden ist wird das Polygon weiter umlaufen



- Der Weiler-Atherton-Algorithmus erzeugt echt separate Polygonzüge für jedes sichtbare Fragment

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 60

Implementierung des Algorithmus

- Eingabe (hier ist die Konvention *clockwise ordering*):

The diagram illustrates the input for a clipping algorithm. On the left, a gray polygon is shown with vertices labeled 0 through 9 in a clockwise direction. A black clip window is overlaid on it, with vertices labeled a, b, c, and d. The clip window is a rectangle with dashed lines extending from its corners. The polygon's vertices are also labeled with letters i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, aa, ab, ac, ad, ae, af, ag, ah, ai, aj, ak, al, am, an, ao, ap, aq, ar, as, at, au, av, aw, ax, ay, az, ba, bb, bc, bd, be, bf, bg, bh, bi, bj, bk, bl, bm, bn, bo, bp, bq, br, bs, bt, bu, bv, bw, bx, by, bz, ca, cb, cc, cd, ce, cf, cg, ch, ci, cj, ck, cl, cm, cn, co, cp, cq, cr, cs, ct, cu, cv, cw, cx, cy, cz, da, db, dc, dd, de, df, dg, dh, di, dj, dk, dl, dm, dn, do, dp, dq, dr, ds, dt, du, dv, dw, dx, dy, dz, ea, eb, ec, ed, ee, ef, eg, eh, ei, ej, ek, el, em, en, eo, ep, eq, er, es, et, eu, ev, ew, ex, ey, ez, fa, fb, fc, fd, fe, ff, fg, fh, fi, fj, fk, fl, fm, fn, fo, fp, fq, fr, fs, ft, fu, fv, fw, fx, fy, fz, ga, gb, gc, gd, ge, gf, gg, gh, gi, gj, gk, gl, gm, gn, go, gp, gq, gr, gs, gt, gu, gv, gw, gx, gy, gz, ha, hb, hc, hd, he, hf, hg, hh, hi, hj, hk, hl, hm, hn, ho, hp, hq, hr, hs, ht, hu, hv, hw, hx, hy, hz, ia, ib, ic, id, ie, if, ig, ih, ii, ij, ik, il, im, in, io, ip, iq, ir, is, it, iu, iv, iw, ix, iy, iz, ja, jb, jc, jd, je, jf, jg, jh, ji, jj, jk, jl, jm, jn, jo, jp, jq, jr, js, jt, ju, jv, jw, jx, jy, jz, ka, kb, kc, kd, ke, kf, kg, kh, ki, kj, kk, kl, km, kn, ko, kp, kq, kr, ks, kt, ku, kv, kw, kx, ky, kz, la, lb, lc, ld, le, lf, lg, lh, li, lj, lk, ll, lm, ln, lo, lp, lq, lr, ls, lt, lu, lv, lw, lx, ly, lz, ma, mb, mc, md, me, mf, mg, mh, mi, mj, mk, ml, mm, mn, mo, mp, mq, mr, ms, mt, mu, mv, mw, mx, my, mz, na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm, nn, no, np, nq, nr, ns, nt, nu, nv, nw, nx, ny, nz, oa, ob, oc, od, oe, of, og, oh, oi, oj, ok, ol, om, on, oo, op, oq, or, os, ot, ou, ov, ow, ox, oy, oz, pa, pb, pc, pd, pe, pf, pg, ph, pi, pj, pk, pl, pm, pn, po, pp, pq, pr, ps, pt, pu, pv, pw, px, py, pz, qa, qb, qc, qd, qe, qf, qg, qh, qi, qj, qk, ql, qm, qn, qo, qp, qq, qr, qs, qt, qu, qv, qw, qx, qy, qz, ra, rb, rc, rd, re, rf, rg, rh, ri, rj, rk, rl, rm, rn, ro, rp, rq, rr, rs, rt, ru, rv, rw, rx, ry, rz, sa, sb, sc, sd, se, sf, sg, sh, si, sj, sk, sl, sm, sn, so, sp, sq, sr, ss, st, su, sv, sw, sx, sy, sz, ta, tb, tc, td, te, tf, tg, th, ti, tj, tk, tl, tm, tn, to, tp, tq, tr, ts, tt, tu, tv, tw, tx, ty, tz, ua, ub, uc, ud, ue, uf, ug, uh, ui, uj, uk, ul, um, un, uo, up, uq, ur, us, ut, uu, uv, uw, ux, uy, uz, va, vb, vc, vd, ve, vf, vg, vh, vi, vj, vk, vl, vm, vn, vo, vp, vq, vr, vs, vt, vu, vv, vw, vx, vy, vz, wa, wb, wc, wd, we, wf, wg, wh, wi, wj, wk, wl, wm, wn, wo, wp, wq, wr, ws, wt, wu, wv, ww, wx, wy, wz, xa, xb, xc, xd, xe, xf, xg, xh, xi, xj, xk, xl, xm, xn, xo, xp, xq, xr, xs, xt, xu, xv, xw, xx, xy, xz, ya, yb, yc, yd, ye, yf, yg, yh, yi, yj, yk, yl, ym, yn, yo, yp, yq, yr, ys, yt, yu, yv, yw, yx, yy, yz, za, zb, zc, zd, ze, zf, zg, zh, zi, zj, zk, zl, zm, zn, zo, zp, zq, zr, zs, zt, zu, zv, zw, zx, zy, zz.

Zyklische Liste der Vertices des Polygons

Zyklische Liste der Vertices des Clip-Windows

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 61

- Finde die Schnittpunkte und füge sie in **beide** Listen ein

Add Vertex i:

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 62

Find the intersection points and add them to **both** lists

Add Vertex i:

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 63

Find the intersection points and add them to **both** lists

Add Vertex k:

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 64

Find the intersection points and add them to both lists

Add Vertex j:

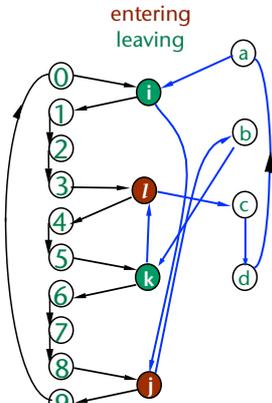
G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 65

Classify each intersection point as „entering“ or „leaving“

entering
leaving

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 66

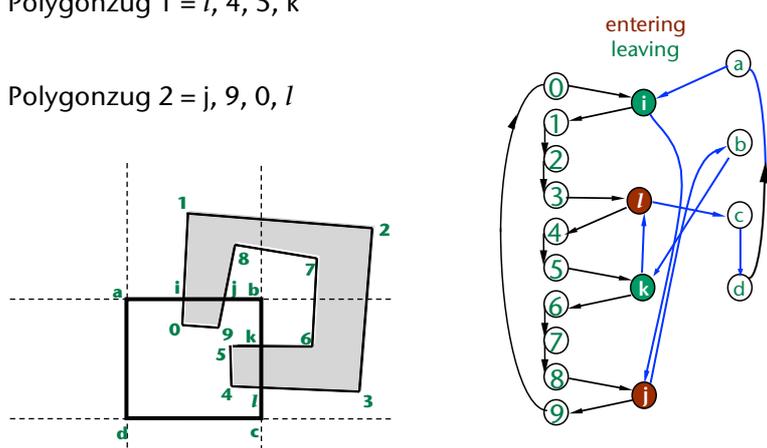
- Erzeuge das geclippte Polygon:
 - Starte bei einem "entering"-Vertex
 - Falls man auf einen "leaving"-Vertex trifft, dann wechsele auf die Liste des Clip-Polygons (blaue Zeiger)
 - Falls man auf einen "entering"-Vertex trifft, dann wechsele auf die Liste des Polygons (schwarze Zeiger)
 - Eine Polygonzug ist beendet, wenn der Startpunkt wieder erreicht ist
 - Wiederhole, solange noch unbesuchte "entering"-Vertices vorhanden sind



Clipping 67

Beispiel

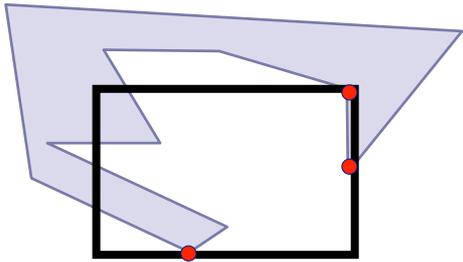
- Polygonzug 1 = *l*, 4, 5, *k*
- Polygonzug 2 = *j*, 9, 0, *l*



Clipping 68

Robustheit, Präzision, Entartungen

- Die üblichen (leidigen) Fragen:
 - Was passiert wenn ein Vertex (beinahe) auf dem Rand des anderen Polygonzuges liegt?



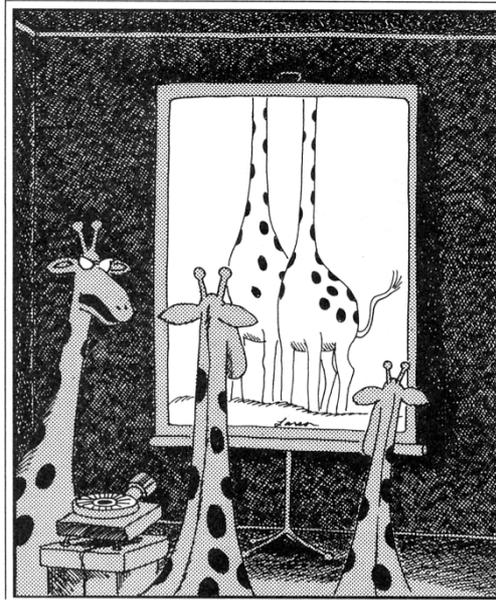
G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 69

Absch(l)ießende Bemerkungen

- Ältere Hardware machte volles Clipping
- Moderne Hardware vermeidet Clipping weitgehend:
 - Nur bzgl. der Ebene $z=z_0$
- Im Allgemeinen ist es nützlich, Clipping zu kennen, da es viele ähnliche geometrische Probleme und Algorithmen gibt, z.B.:
 - Zur Bestimmung, welche Objekte innerhalb eines "Picking-Frustums" liegen
 - Schnittpunkte zwischen Objekten
 - Berechnung analytischer Schatten

Modeling Transformations
Illumination (Shading)
Viewing Transformation (Perspective / Orthographic)
Clipping
Projection (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

G. Zachmann Computer-Graphik 1 – WS 10/11 Clipping 70



“Oh, lovely — just the hundredth time you’ve managed to cut everyone’s head off.”