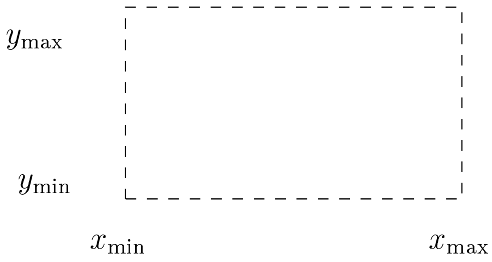
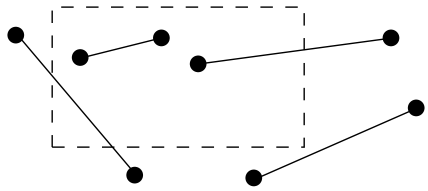
Kapitel 5

Clipping

Ziel: Nur den Teil eines Objekts darstellen, der innerhalb eines Fensters sichtbar ist.

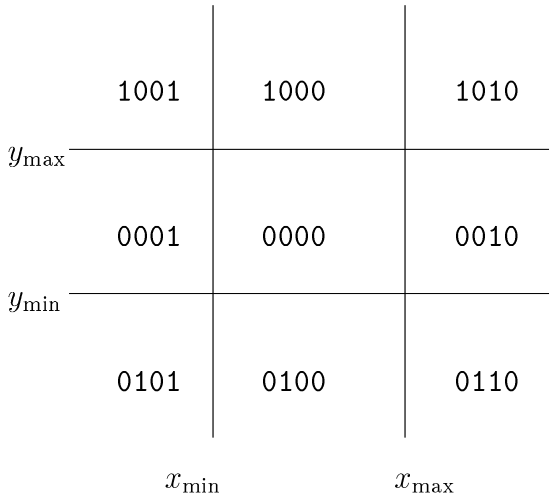


5.1 Clipping von Linien



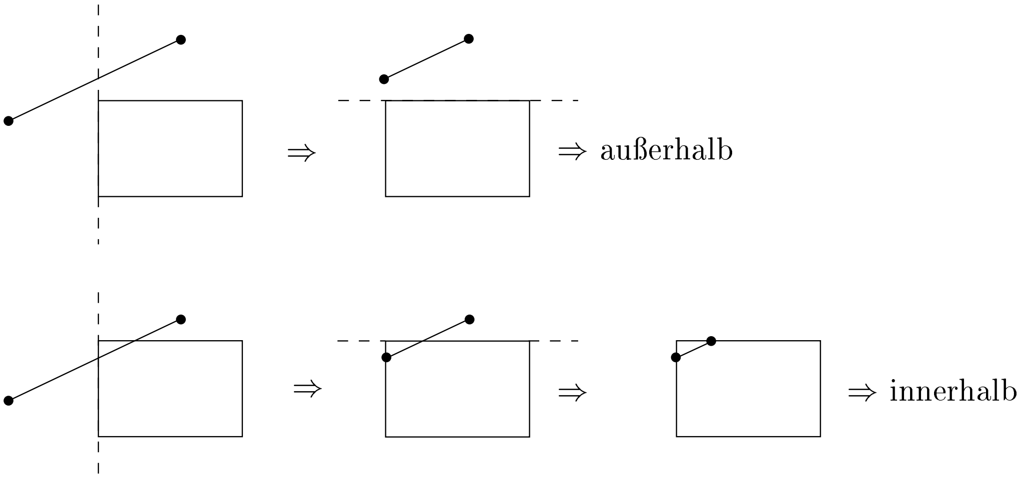
Idee von Cohen & Sutherland

Teile Ebene anhand des Clip-Fensters in 9 Bereiche ein, wähle 4-Bit-Bereichscode für jeden Punkt:



Bit 0:	links vom Fenster
Bit 1:	rechts vom Fenster
Bit 2:	unter dem Fenster
Bit 3:	über dem Fenster

Sei $\overline{P_1P_2}$ eine Linie. Dann gilt bei einer bitweisen Verknüpfung:
code (P_1) AND *code* $(P_2) \neq 0 \Rightarrow \overline{P_1P_2}$ komplett außerhalb (beide Punkte auf derselben Seite)
code (P_1) OR *code* $(P_2) = 0 \Rightarrow \overline{P_1P_2}$ komplett innerhalb (beide Punkte im Clip-Fenster)
In den anderen Fällen wird $\overline{P_1P_2}$ mit einer Fensterkante geschnitten und der Test mit der verkürzten Linie erneut ausgeführt.

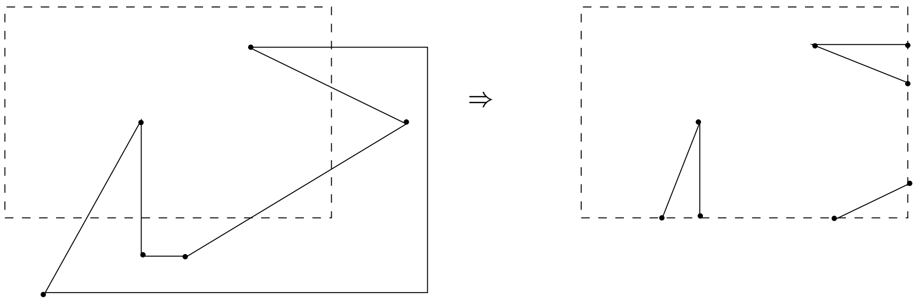


```
/**************************************************************************************/
/*                                                                                    */
/*                Clippen von Linien an einem Fenster nach Cohen-Sutherland           */
/*                                                                                    */
/**************************************************************************************/

private static final byte EMPTY  = 0;
private static final byte LEFT   = 1;
private static final byte RIGHT  = 2;           // 4-Bit-Bereichscodes
private static final byte BOTTOM = 4;
private static final byte TOP    = 8;

private int xmin, xmax, ymin, ymax;             // Clip-Fensterraender

private byte region_code(                       // liefert den region-code
                          Point P )             // fuer den Punkt P
{
        byte c;

        c = EMPTY;
        if (P.x < xmin) c = LEFT; else
        if (P.x > xmax) c = RIGHT;
        if (P.y < ymin) c |= BOTTOM; else
        if (P.y > ymax) c |= TOP;
        return(c);
}


private void set_clip_window(         // setzt die Variablen xmin, ymin, xmax, ymax
                Point P,              // anhand des Ursprungs P des Clip-Fensters
                Point delta )         // und anhand seiner Breite/Hoehe delta
        {
                xmin = P.x; xmax = P.x + delta.x;
                ymin = P.y; ymax = P.y + delta.y;
        }


private boolean cohen_sutherland(     // liefert true,
                Point p1, Point p2,   // falls die Gerade p1-p2 sichtbar ist
                Point Q1, Point Q2)   // liefert ggf. sichtbaren Teil Q1-Q2 zurueck
{
        boolean finite_slope;         // true falls Gerade p1-p2 nicht senkrecht laeuft
        double slope = 0.0;           // Steigung der Geraden p1-p2
        byte C, C1, C2;               // 4-Bit-Bereichs-Code
        Point Q = new Point();        // zu berechnender Schnittpunkt mit Gerade

        Point P1 = new Point(p1);     // lokale
        Point P2 = new Point(p2);     // Variablen
```

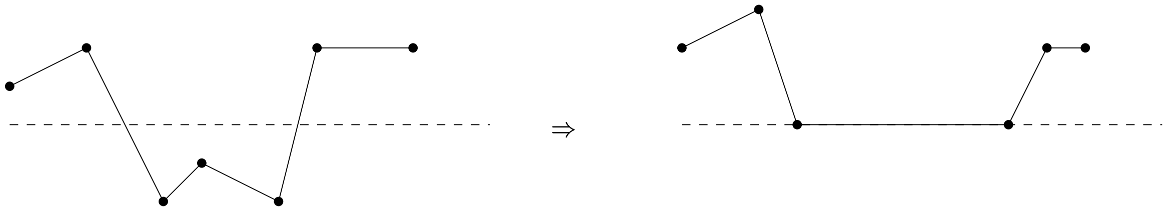
```
        finite_slope = (P1.x != P2.x);
        if (finite_slope) slope = (double)(P2.y-P1.y)/(double)(P2.x-P1.x);
        C1 = region_code(P1);
        C2 = region_code(P2);

        while ((C1 != EMPTY) || (C2 != EMPTY))  { // mind. ein Endpunkt noch ausserhalb

            if ((C1&C2) != EMPTY) return(false);  // beide auf derselben Seite ausserhalb
            else
            {
                if (C1 == EMPTY) C = C2; else C = C1; // C ist ausserhalb. Berechne
                                                      // einen Schnittpunkt mit den
                                                      // verlaengerten Fensterkanten

                if ((C & LEFT) != EMPTY) {         // schneide mit linker Fenster-Kante
                        Q.x = xmin;
                        Q.y = (int)((Q.x-P1.x)*slope + P1.y);
                        } else

                if ((C & RIGHT) != EMPTY){        // schneide mit rechter Fenster-Kante
                        Q.x = xmax;
                        Q.y = (int)((Q.x-P1.x)*slope + P1.y);
                        } else

                if ((C & BOTTOM) != EMPTY) {      // schneide mit unterer Fenster-Kante
                        Q.y = ymin;
                        if (finite_slope)
                                Q.x = (int)((Q.y-P1.y)/slope + P1.x); else
                                Q.x = P1.x;
                        }else

                if ((C & TOP) != EMPTY) {         // schneide mit oberer Fenster-Kante
                        Q.y = ymax;
                        if (finite_slope)
                                Q.x = (int)((Q.y-P1.y)/slope + P1.x); else
                                Q.x = P1.x;
                        }

                if (C==C1) {
                        P1.x = Q.x; P1.y = Q.y; C1 = region_code (P1);
                } else {
                        P2.x = Q.x; P2.y = Q.y; C2 = region_code (P2);
                }
            }

        }
        // uebergib Anfang und Ende des sichtbaren Teils
        Q1.x=P1.x; Q1.y=P1.y;
        Q2.x=P2.x; Q2.y=P2.y;

        return(true);
}
```

5.2 Clipping von Polygonen

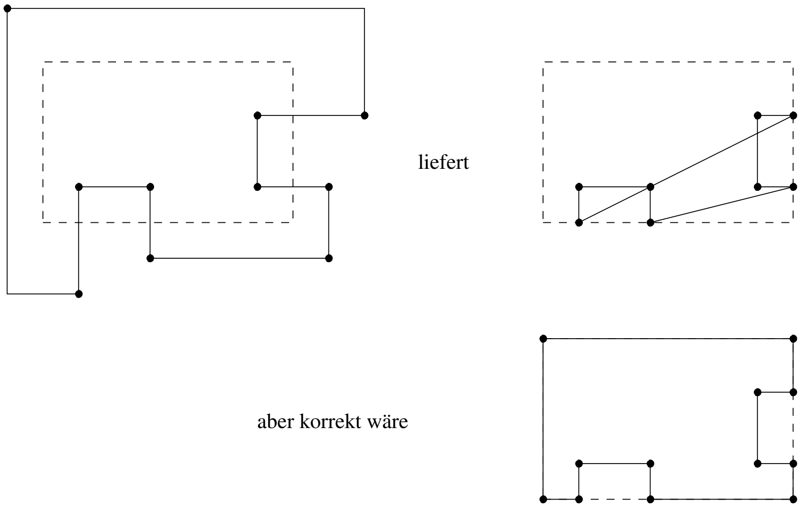
Für das Clippen von Polygonen reicht es nicht, jede beteiligte Kante zu clippen.



Vielmehr müssen zusätzlich die Ein- und Austrittspunkte verbunden werden, um nach dem Clipping wieder ein Polygon zu erhalten:



Obacht: Bzgl. der Ecken des Clip-Windows ist eine Spezialbehandlung erforderlich:

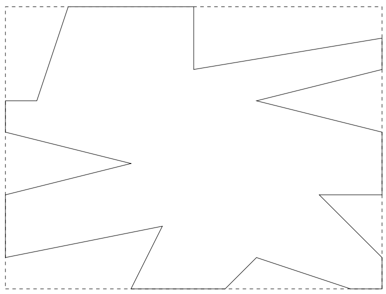


Der **Sutherland & Hodgman-Algorithmus** clippt daher an den vier Fensterkanten nacheinander:

```
Fuer jede Clipping-Gerade E tue:
  fuer jeden Polygonpunkt P tue:
    falls P sichtbar: uebernimm ihn
    falls Kante von P zu seinem Nachfolger E schneidet:
          uebernimm den Schnittpunkt
```

```
/*****************************************************************************/
/*                                                                           */
/*          Clippen von Polygonen an einem Fenster nach Sutherland-Hodgmann  */
/*                                                                           */
/*****************************************************************************/

private boolean On_Visible_Side(                  /* liefert true, falls         */
                        Point P,                  /* Punkt p auf der sichtbaren  */
                        int wert,                 /* Seite der Kante wert liegt  */
                        int fall)                 /* gemaess Fall fall           */
{
        switch (fall) {

                case 0: return (P.x >= wert);     /* linke  Fenster-Kante        */
                case 1: return (P.y >= wert);     /* untere Fenster-Kante        */
                case 2: return (P.x <= wert);     /* rechte Fenster-Kante        */
                case 3: return (P.y <= wert);     /* obere  Fenster-Kante        */
        }

        return false;
}


private boolean Intersection(                /* liefert true,                      */
                        Point P1, Point P2,  /* falls sich die Linie P1-P2         */
                        int wert,            /* mit Clipping-Kante wert schneidet  */
                        int fall,            /* gemaess Fall fall                  */
                        Point I)             /* liefert den Schnittpunkt I zurueck */
{
        boolean P1_vis, P2_vis;              /* true, falls P1 bzw P2 sichtbar     */
        double slope;                        /* Steigung der Geraden P1-P2         */
                                             /* y = (x-P1.x)*slope + P1.y          */

        P1_vis = On_Visible_Side(P1,wert,fall);      /* P1 sichtbar ?              */
        P2_vis = On_Visible_Side(P2,wert,fall);      /* P2 sichtbar ?              */

        if ((P1_vis && P2_vis) || (!P1_vis && !P2_vis)) return (false);/*kein Schnittpkt*/
        else
        {
                slope = (double)(P2.y-P1.y) / (double) (P2.x-P1.x);   /* Steigung      */

                if (fall%2==1) {
                        I.x = (int)((wert-P1.y) / slope) + P1.x;
                        I.y = wert;
                } else
                {
                        I.x = wert;
                        I.y = (int)((wert-P1.x) * slope) + P1.y;
                }
                return(true);
        }
}
```

```
private int sutherland_hodgman(                  /* fuehrt ein Clipping durch          */
                        int num_points,          /* fuer num_points Punkte             */
                        Point[] points,          /* der Polygonliste Points            */
                        int wert,                /* bezueglich Clipping-Kante wert     */
                        int fall)                /* gemaess Fall fall                  */
/* liefert neue Anzahl Punkte */
{
        int   i;
        int   old_ptr;
        int   new_ptr;
        Point S;
        Point I;
        Point[] hpoints = new Point[MAX_POINTS];

        if (num_points > 2) {
                old_ptr = 0;
                new_ptr = -1;
                points[num_points] = points[0];

                while (old_ptr < num_points) {

                        S = points[old_ptr];
                        if (On_Visible_Side(S,wert,fall)) {
                                new_ptr++;
                                hpoints[new_ptr] = S;
                        }
                        old_ptr++;
                        I = new Point();
                        if (Intersection(S,points[old_ptr],wert,fall,I)) {
                                new_ptr++;
                                hpoints[new_ptr] = I;
                        }
                }

                num_points = new_ptr+1;
                for (i=0; i<num_points; i++) points[i]=hpoints[i];
        }
        return num_points;
}
```



Vom Sutherland-Hodgman-Algorithmus geclipptes Polygon