Kapitel 5

2D-Clipping

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

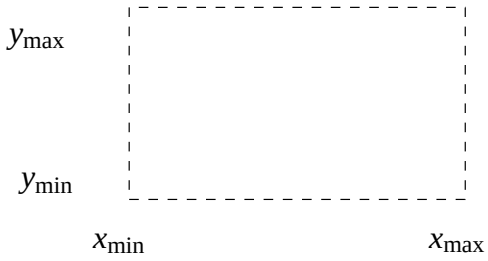

Abbildung 5.1: Clip-Fenster

5.1 Clipping von Linien

Zu einer Menge von Linien sind jeweils neue Anfangs- und Endpunkte zu bestimmen, welche komplett im Clip-Fenster liegen.

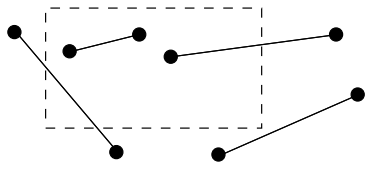

Abbildung 5.2: Ausgangslage beim Linien-Clipping

Idee von Cohen & Sutherland

Teile Ebene anhand des Clip-Fensters in 9 Bereiche ein, beschrieben durch 4-Bit-Bereichscode:

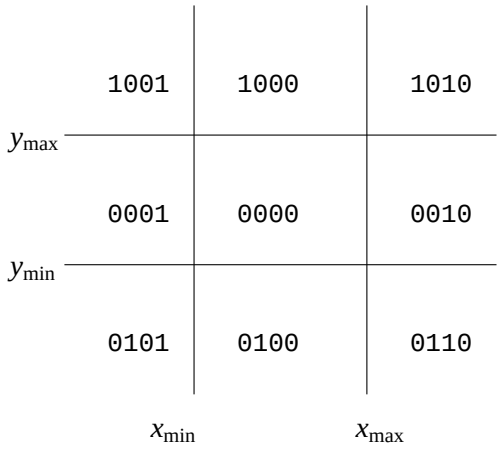

Abbildung 5.3: Bereichscodes nach Cohen & Sutherland

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.

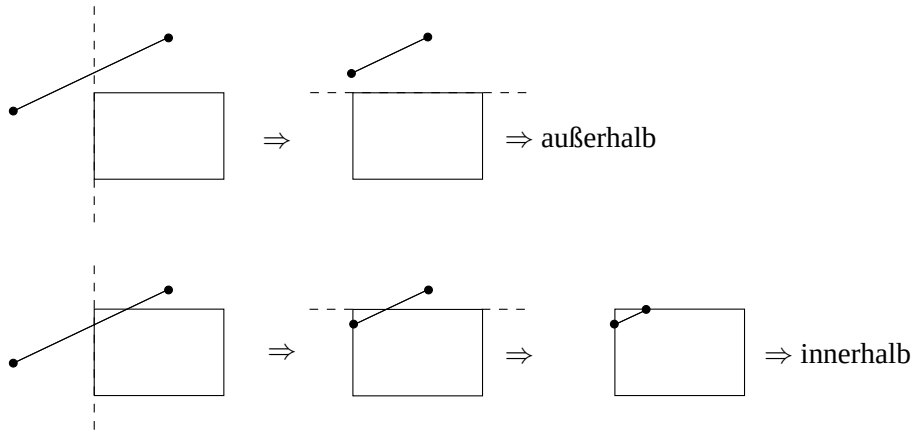

Abbildung 5.4: Möglichkeiten bei zwei außerhalb liegenden Punkten

```
/*****************************************************************************/
/*                                                                           */
/*                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:

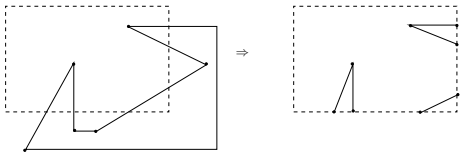

Abbildung 5.5: Zerfall eines Polygons bei reinem Linien-Clippen

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

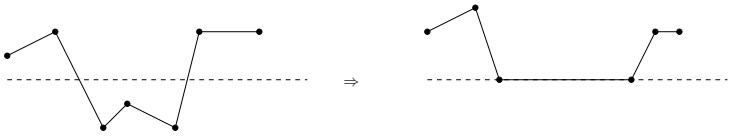

Abbildung 5.6: Verbinden der Ein- und Austrittspunkte

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

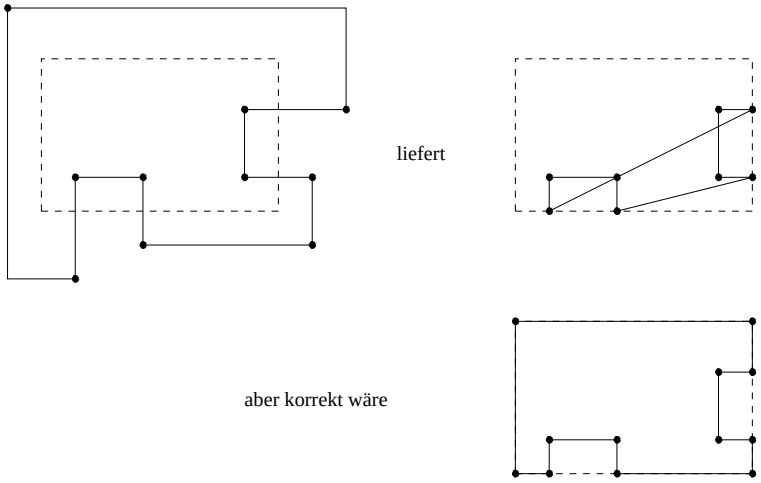

Abbildung 5.7: Spezialbehandlung an den Clipfensterecken

Der **Sutherland & Hodgman-Algorithmus** clippt an 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  */
/*                                                                           */
/*****************************************************************************/

public boolean onVisibleSide(              // Liefert true, wenn der Punkt p
                             Point p,      // bzgl. der waage- oder senkrechten
                             int value,    // Kante beim entspr.  y- oder x-Wert
                             int edge) {   // value auf der sichtb. Seite liegt.
                                           // edge symbolisiert die Lage der
                                           // Kante analog zum Regioncode.

    switch (edge) {
       case LEFT   : return p.x >= value;  // linke  Fenster-Kante
       case RIGHT  : return p.x <= value;  // rechte Fenster-Kante
       case TOP    : return p.y >= value;  // obere  Fenster-Kante
       case BOTTOM : return p.y <= value;  // untere Fenster-Kante
     }

     return false; // sonst: draussen
}


private Point intersection(                // liefert true, falls sich die Linie
                      Point p1, Point p2,  // p1-p2 mit waage- oder senkrechter
                      int value,           // Clipping-Kante bei y- oder x-Wert
                      int edge) {          // value schneidet. edge ist die
                                           // Clipping Kante relativ zum
                                           // Clipping-Rechteck

    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

    Point i = null;                        // liefert den Schnittpunkt i zurueck

    p1_vis=on_Visible_Side(p1,value,edge); // p1 sichtbar ?
    p2_vis=on_Visible_Side(p2,value,edge); // p2 sichtbar ?

    if ((p1_vis && p2_vis) || (!p1_vis && !p2_vis))
        return i;                          // kein Schnittpkt

    else {
        slope = (double)(p2.y-p1.y) / (double) (p2.x-p1.x);   // Steigung

        if (edge==TOP || edge == BOTTOM) {  // waagerechte Kante
            i.x = (int)((value-p1.y) / slope) + p1.x;
            i.y = value;
        }
        else {                             // senkrechte Kante
            i.x = value;
            i.y = (int)((value-p1.x) * slope) + p1.y;
        }
        return i;                         // Schnittpunkt liefern
    }
}
```

```
private boolean sutherland_hodgeman(Vector pv, int value, int edge) {
    int old; // durchlaeuft die alten Punkte
    Vector hilf = new Vector(pv.size(), 1); // Hilfsvektor
    Point s, i;

    for (old = 0; old < pv.size() - 1; old++) {
        s = (Point) pv.elementAt(old);
        if (s.onVisibleSide(value, edge)) {
            hilf.addElement(s);
        }

        i = l.intersects(s, pv.elemenatAt(old+1), value, edge);
        if (i != null) {
            hilf.addElement(i);
         }
     }
     if (hilf.size() > 0) { // Falls Punkte gemerkt
         hilf.addElement(hilf.elementAt(0)); // 1. nochmal ans Ende
     }
     pv = hilf; // neuen Vektor merken
     return hilf.size() != 0; // wenn draussen: Vektor leer
}
```

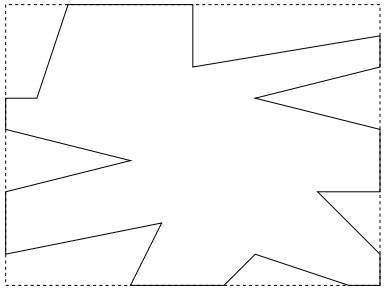

Abbildung 5.8: Vom Sutherland-Hodgman-Algorithmus geclipptes Polygon

5.3 Java-Applet zu 2D-Operationen

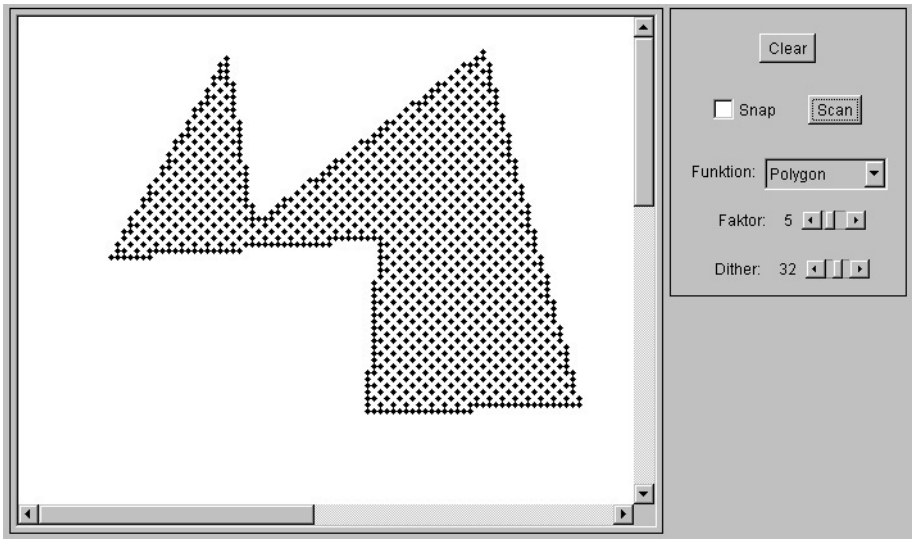

Abbildung 5.9: Screenshot vom 2D-Basic-Applet