Laboratory work 8

1 Objectives

This laboratory presents the topic of clipping algorithms for graphical primitives. Two algorithms for line clipping are discussed, Cyrus-Beck and Cohen-Sutherland.

2 Line clipping algorithms

Clipping algorithms are used to eliminate out-of-range values, meaning parts of segments or polygons which are outside the display area. In most cases the display area is defined as a rectangle and is called the clipping window. Relative to this clipping window the primitive (point, line or polygon) can be in the following relationship:

- Entirely inside the clipping window – no need to clip the primitive, continue to rasterize it.
- Entirely outside the clipping window – no need to clip the primitive, discard it.
- Intersects the clipping window – compute the intersection points and update the primitive, continue to rasterize it.

3 Cyrus-Beck algorithm

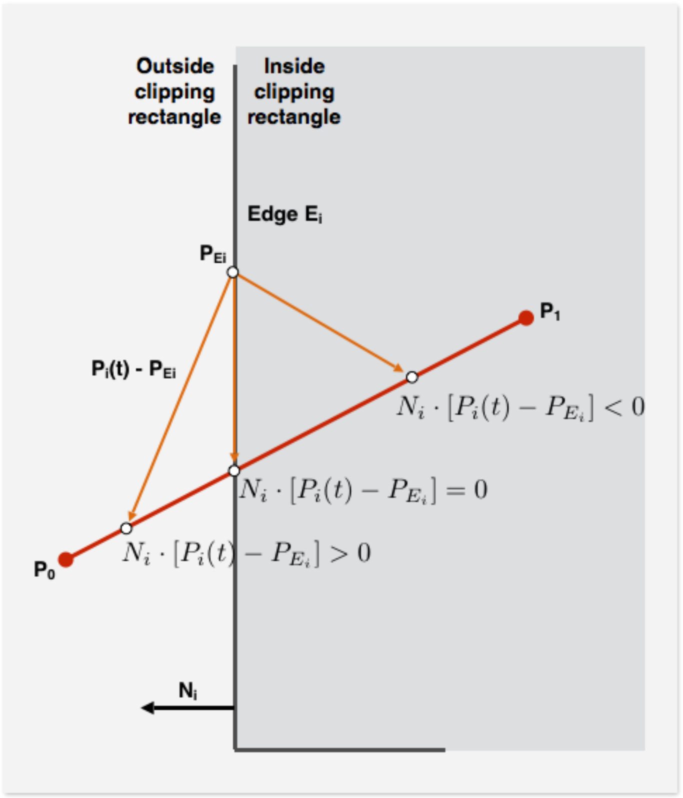
The Cyrus-Beck algorithm is used to clip a line segment against a convex polygon. The line segment is defined parametrically as:

$P(t) = P_0 + (P_1 - P_0)t$, where $t \in [0,1]$

For each edge E_i we compute the normal vector in such a way it points outward (see the figure on the right). From each edge we pick a point P_{Ei}. By computing the dot product $N_i \cdot (P(t) - P_{Ei})$ we can find the relative position of every point on the line segment (defined for a particular value of the **t** parameter):

- if the value is negative the point lies in the inside halfplane
- if the value is positive the point lies in the outside halfplane
- if the value is zero the point is on the edge.

We are interested in the value of **t** for which $N_i \cdot (P(t) - P_{Ei}) = 0$.



Outside clipping rectangle
Inside clipping rectangle
Edge E_i
P_{Ei}
P_1
$P_i(t) - P_{Ei}$
$N_i \cdot [P_i(t) - P_{E_i}] < 0$
$N_i \cdot [P_i(t) - P_{E_i}] = 0$
$N_i \cdot [P_i(t) - P_{E_i}] > 0$
P_0
N_i

First we substitute P(t) and we regroup some terms:

$$N_i \cdot (P_0 + (P_1 - P_0)t - P_{Ei}) = 0$$

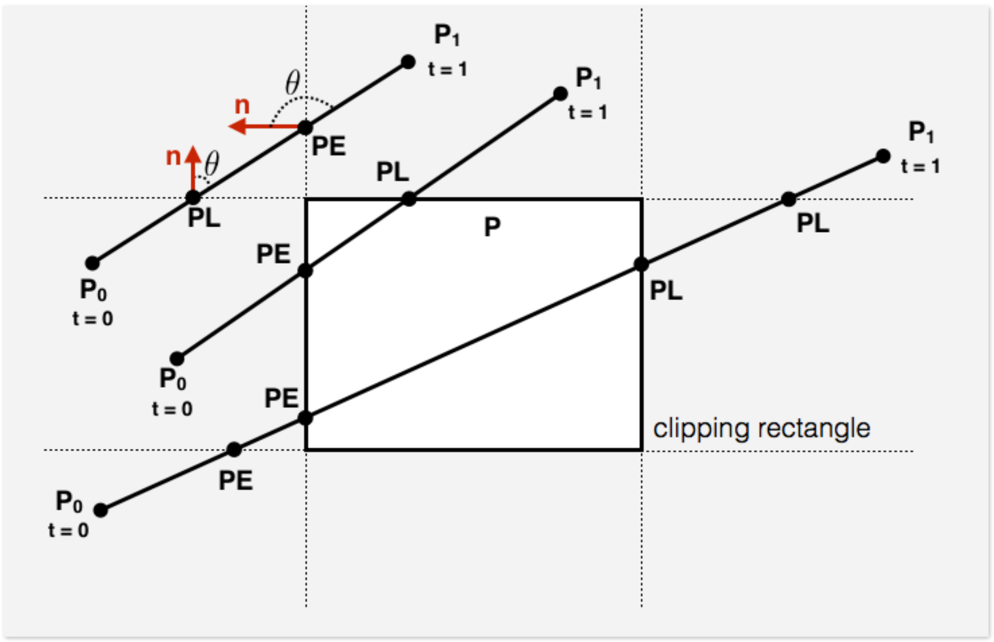
$$N_i \cdot (P_0 - P_{Ei}) + N_i \cdot (P_1 - P_0)t \ = 0$$

Let $D = P_1 - P_0$ be the vector from point P_0 to point P_1. Then we can compute **t** as:

$$t = \frac{N_i \cdot (P_0 - P_{Ei})}{-N_i \cdot D}$$

We need to compute the value of **t** (for the intersection point) for each edge of the clipping window. The values of **t** outside the interval [0, 1] are discarded. Each intersection is characterized by computing the angle between P_0P_1 and N_i as:

- "potentially entering" (PE) – the angle > 90°
- "potentially leaving" (PL) – the angle < 90°



3.1 Pseudocode

```
precalculate Ni and select a PEi for each edge;
if (P1 = P0)
  line degenerates to a point, so clip as a point;
else
  tE = 0; tL = 1;
  for (each candidate compute intersection with a clipping edge){
    if (Ni * D != 0){
      calculate t;
      use sign of Ni * D to categorize as PE (potentially entering) or PL   (potentially leaving);
      if (PE)
        tE = max(tE, t);
      if (PL)
        tL = min(tL, t);
    }
  }
  if (tE > tL)
    return -1
  else
        return P(tE ) and  P(tL ) as true clip intersections
```

4 Cohen-Sutherland clipping algorithm

4.1 Determine if a point P(x,y) is visible

Considering P1(x_{min}, y_{min}), and P2(x_{max}, Y_{max}) the defining points of the visible area rectangle, the point P(x,y) is visible only if the following conditions are met:

$$x_{min} \le x \le x_{max}$$

$$y_{min} \le y \le y_{max}$$

4.2 Determine if a segment is visible

In order to determine if a line segment is visible, we need slightly more complex algorithms. One idea would be to test the visibility of each point of the segment, before displaying it on the screen. But this method will require a lot of time and very many computations. The method can be easily improved by testing first the heads of the segment. If both these points are in the visible area, the entire segment will be visible. This case is called “simple acceptance”. On the same logic, if both points are outside and on the same side of the visible area, no part of the segment will be visible. This case is called “simple rejection”. In all the other cases we must use other algorithms to establish which part of the segment (if any) is visible.

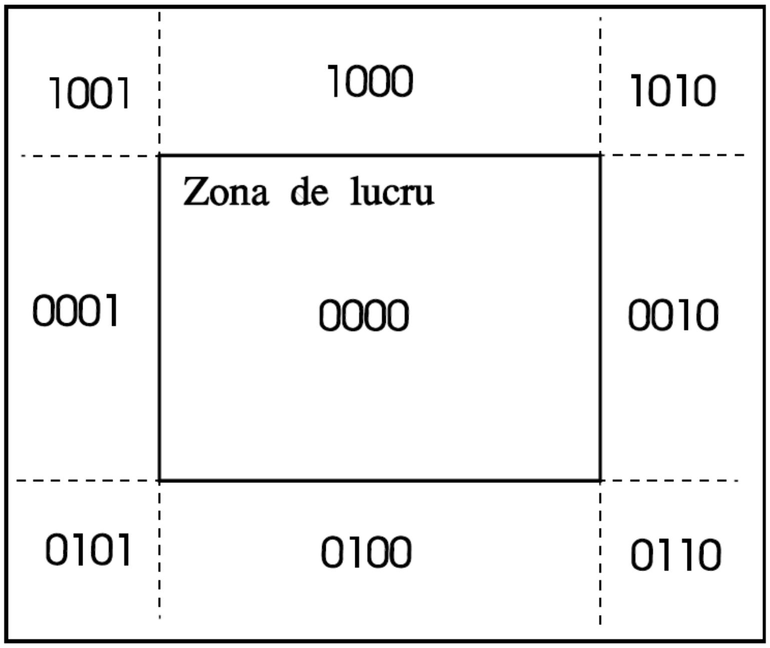
If a line segment cannot be included either in “simple acceptance” or “simple rejection” cases, then we have to compute its intersection points with the following lines:

$$y = y_{max},\ x = x_{max},\ y = y_{min},\ x = x_{min}$$

and to eliminate the segments that are placed outside the visible area. As a result, we will obtain a new line segment. The algorithm is repeated until the resulted segment can be included in one of the “simple acceptance” or “simple rejection” cases.

The Cohen-Sutherland algorithm uses a four digits code to describe each head of the segment. The code has the following structure:

- the first digit is 1 if the point is above the visible area; otherwise is 0
- second digit is 1 if the point is under the visible area; otherwise is 0
- third digit is 1 if the point is on the right of the visible area; otherwise is 0
- fourth digit is 1 if the point is on the left side of the visible area; otherwise is 0



4.3 Pseudocode

```
repeat until FINISHED = TRUE
{
   COD1 = computeCScode($x_1$, $y_1$)    //compute the 4 digits code for P($x_1$, $y_1$)
   COD2 = computeCScode($x_2$, $y_2$)    // compute the 4 digits code for P($x_2$, $y_2$)
   RESPINS = SimpleRejection(COD1, COD2)    //test for simple rejection case
   if RESPINS = TRUE
      FINISHED = TRUE
   else
   {
      DISPLAY = SimpleAcceptance(COD1, COD2)    //test for simple acceptance case
      if DISPLAY = TRUE
         FINISHED = true
      else
      {
         if(P(x1, y1) is inside the display area)
            invert($x_1$,$y_1$,$x_2$,$y_2$,COD1,COD2)    //if P($x_1$, $y_1$) is inside the display area, invert P($x_1$, $y_1$) and P($x_2$, $y_2$)
                                                         together with their 4 digits CS codes

         if(COD1[1] = 1) and ($y_2$ <> $y_1$)    //eliminate the segment above the display area
         {
            $x_1 = x_1+(x_2-x_1)*(Y_{max}-y_1)/(y_2-y_1)$
            $y_1 = Y_{max}$
         }
         elseif(COD1[2] = 1) and ($y_2$ <> $y_1$)    //eliminate the segment under the display area
         {
            $x_1 = x_1+(x_2-x_1)*(Y_{min}-y_1)/(y_2-y_1)$
            y_1 = Y_min
        }
        elseif(COD1[3] = 1) and (x_2 <> x_1)     //eliminate the segment on the right of the display area
        {
            y_1 = y_1+(y_2-y_1)*(X_max-x_1)/(x_2-x_1)
            x_1 = X_max
        }
        elseif(COD1[4] = 1) and (x_2 <> x_1)     //eliminate the segment on the left of the display area
        {
            y_1 = y_1+(y_2-y_1)*(X_min-x_1)/(x_2-x_1)
            x_1 = X_min
        }
      }
    }
}
```

5 Assignment

- Implement the Cyrus-Beck algorithm
- Implement the Cohen-Sutherland algorithm
- Implement (using SDL) an interactive demonstration of the algorithms. Define using the mouse the clipping window (a rectangle or a convex polygon) and a line segment.