EUROGRAPHICS '89
W. Hansmann, F.R.A. Hopgood and W. Strasser (Editors)
Elsevier Science Publishers B.V. (North-Holland)

Algorithms for 2D Line Clipping

Vaclav Skala

Department of Computer Science and Informatics
Institute of Technology
Plzen, Czechoslovakia

New algorithms for 2D line clipping against convex, non-convex windows and windows that consist of linear edges and arcs are being presented. Algorithms were derived from the Cohen-Sutherland 's and Liang-Barsky s algorithms. The general algorithm with linear edges and arcs can be used especially for engineering drafting systems. Algorithms are easy to modify in order to deal with holes too. The presented algorithms have been verified in TURBO-PASCAL. Because of the unifying approach to the clipping problem solution all algorithms are simple, easy to understand and implement.

1. INTRODUCTION

Clipping is a very important part of all graphics packages. Generally it is the evaluation of a line intersection against a window boundary. There are many efficient algorithms such as the Cohen-Sutherland 's [6], Liang-Barsky 's [5], Cyrus-Beck 's [1] ones. All these algorithms have some presumptions, e.g. the windows must be orthogonal or convex with oriented edges, etc.

In the following paragraph new algorithms will be described for convex-polygon and non-convex polygon clipping without any necessity orient edges in any order. An algorithm for non-convex area clipping, where boundaries are formed by line segments or arcs, is described, too. Particular care was devoted to handle all special situations properly. All algorithms are based on the only basic idea that is gradually widened for more general cases.

As far as the author is concerned none of these algorithms have been published in any accessible literature.

2. CONVEX POLYGON CLIPPING

The below shown convex polygon clipping algorithm is based on the principle of Liang-Barsky's algorithm and is simpler than the Cyrus-Beck's algorithm and does not need an anticlockwise orientation of the polygon edges as Liang-Barsky s algorithm does. Provided a convex polygon is given by its vertices in the clockwise or in the anticlockwise order arbitralily and no pair of edges lies on the same line (it is not a principle restriction). Let us consider some situations that might occur if a line segment with end points P_r and P_s ought to be clipped, see fig.2.2.

All intersections of the line w(q) with edges of the convex polygon are obtained by solving the following linear equations:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in \langle 0 , 1 \rangle$$

$$x(p) = x_i + (x_{i+1} - x_i) \cdot p \quad p \in (0 , 1 \rangle \quad i=0,1,\dots,n-1$$

where + means addition modulo n and point P_k has coordinates x_k.

The interval for parameter p is not closed in order to get rid of all ambiguities in case that the line segment is "passing" (line $w_0(q)$) or "touching" (line $w_1(q)$) a polygon vertex. The below given algorithm is based on the fact that a line segment can intersect a convex polygon only in two points. The algorithm finds the q values for all intersection points of a line on which the respective line segment lies with the edges of the convex polygon and then the proper part of the line segment that is inside the convex polygon can be found. The algorithm is shown in fig.2.1. It is necessary, of course, to solve special cases when a line segment touches or passes a vertex or when it lies on a polygon edge.

The algorithm shown in fig.2.1. is faster than the Cyrus-Beck s algorithm [1] (it doesn't need any inner normal computation) and for a rectangle polygon it is equivalent to the Liang-Barsky s algorithm [4] in case that computation of all intersections is simplified for polygon edges parallel to the axes. The algorithm can be easily generalized or modified for a case when two edges of the given polygon lie on the same line, see [8].

```
VAR i,k: INTEGER; (* end points of the polygon edges *)
    j: INTEGER; (* counter *)
    t: ARRAY [1..2] OF REAL;
BEGIN
    j:=0;i:=0;k:=n-1; (* set end points for the first edge *)
    REPEAT
        IF an intesection point exists for the edge x_k x_i
           and the line w(q) so that p ∈ <0,1)
        THEN
           BEGIN j:=j+1;
                 t[j]:=q (* save the q value *)
           END
        ELSE
           IF the edge x_k x_i lies on the line w(q)
           THEN
              BEGIN t[1]:= a value q which corresponds to the
                               vertex x_k;
                    t[2]:= a value q which corresponds  to  the
                               vertex x_i;
                    j:=2
              END;
        k:=i; i:=i+1; (* take the next polygon edge *)
    UNTIL ( j = 2 ) OR ( i > n );
    IF j <> 0
    THEN
       BEGIN
          IF j = 1
          THEN t[2]:=t[1] (* the line w(q) "touches" vertex *)
          ELSE
             IF t[1]>  t[2] THEN t[1] SWAP t[2];
          t[1]:= max ( 0.0 , t[1] ); (* maximal value *)
          t[2]:= min ( 1.0 , t[2] ); (* minimal value *)
          LINE ( x (t[1]) , x (t[2]) )
       END
END;
```

Figure 2.1.

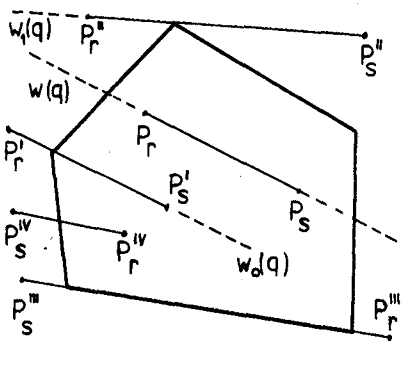


Figure 2.2.

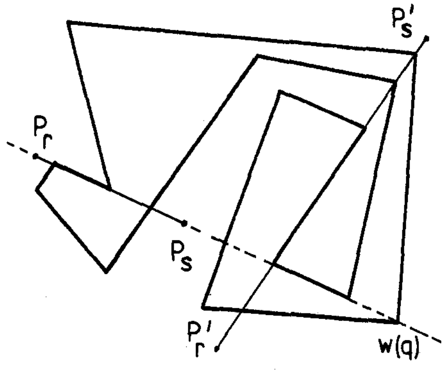


Figure 3.1.

3. NON-CONVEX POLYGON CLIPPING

An algorithm for non-convex polygon clipping is based on the parametric equations that express linear segments. In this case the algorithm must be more complex, because the line w(q) can intersect the polygon in many points. Let us assume that a non-convex polygon is given by its vertices in the clockwise or anticlockwise order. Further that two successive edges do not lie on the same line, that all vertices have different coordinates, that not a single vertex lies on any edge of the given polygon and that two edges might have only a vertex as a common point.

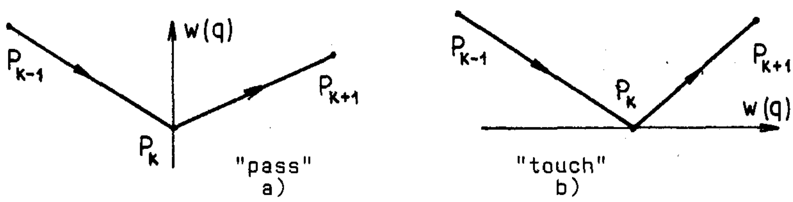
Let us consider again some situations that might occur if a line segment with end points P_r and P_s ought to be clipped, see fig.3.1. The given line segment that ought to be clipped can be expressed by:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in \langle 0 , 1 \rangle$$

and the edges of the non-convex polygon can be expressed by:

$$x(p) = x_i + (x_{i+1} - x_i) \cdot p \qquad p \in \langle 0 , 1) \quad i=0,1,\dots,n-1$$

$[s_1 \times s_2] \cdot [s_3 \times s_2] > 0$ $\qquad$ $[s_1 \times s_2] \cdot [s_3 \times s_2] < 0$



where + means addition modulo n, - means subtraction modulo n

$$s_1 = x_K - x_{K-1} \; , \; s_2 = x_r - x_s \; , \; s_3 = x_{K+1} - x_K$$

Figure 3.2.

Now it is necessary to find all intersection points of the line w(q) with the non-convex polygon. Then the parametric equation:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \quad (-\infty , +\infty)$$

expresses the line w(q). Coordinates of all intersection points will be determined by the value of the parameter q. But it is necessary to take into consideration the following special cases when the line w(q) passes or touches a vertex of the polygon. There are only two possibilities. In fig.3.2.a. one intersection point is generated only, while in fig.3.2.b. double intersection point is generated. In both cases these points are processed as an ordinary intersection point. Quite different situation arises when the line w(q) lies on an edge of the polygon. In that cases it is impossible to decide immediately and therefore a special attribute associated to the value q must be generated. The attribute depends on the sign of the z coordinate of the cross product result of the vectors s_1 and s_2 resp. S3 and s_2 . The following situations should be considered, see fig.3.3. :

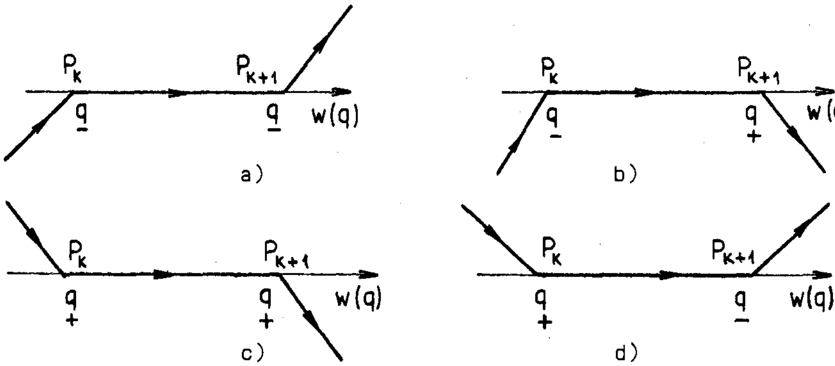


Figure 3.3.

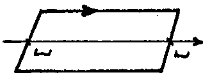
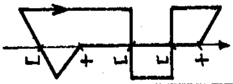
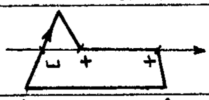
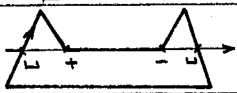
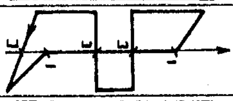
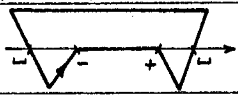
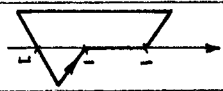
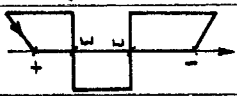
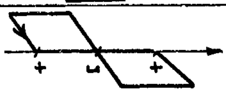
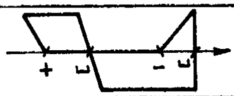
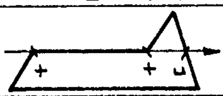
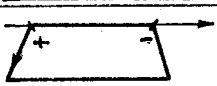
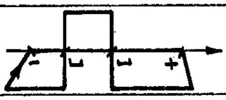
The values of q parameters that correspond to the points P_K and P_{K+1} must be generated in these cases. But it is necessary to distinguish between the shown cases. It can be made by attributes associated with q values. Therefore the intersection point will be determined not only by value q but also by the type of the intersection as follows:

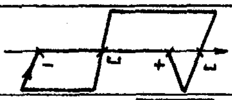
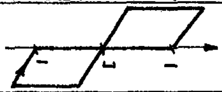
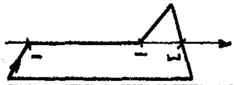
␣ intersection with edge, intersection of the "pass" or "touch" type

\+ or - according to the sign of the z coordinate of the cross product $[s_1 \times s_2]$ resp. $[s_3 \times s_2]$

When all intersection points are found together with their types, the given set of q values is sorted together with their attributes. The set of q values will be processed according to table 3.1. Results that determine these parts of the line w(q) which are inside the given polygon are couples of q values. Now it is necessary to determine what parts of the line segment are inside the given polygon, e.g. to determine those parts of the line w(q) that are inside and that are part of the line segment $P_r P_s$. According to the process of getting the parts of the line w(q) it is necessary to make intersection of all couples of the q values with the interval $\langle 0, 1 \rangle$, see fig.3.5.

Table 3.1. Possible situations for reduction

attributes			situation	action
q_i	q_{i+1}	q_{i+2}		
␣	␣	*		save (q_i , q_{i+1}); i:=i+2
␣	+	␣		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to +
␣	+	+		save (q_i , q_{i+2}); i:=i+3;
␣	+	-		save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
␣	-	␣		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to -
␣	-	+		save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
␣	-	-		save (q_i , q_{i+2}); i:=i+3;
+	␣	␣		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to +
+	␣	+		% save (q_i , q_{i+2}); i:=i+3;
+	␣	-		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
+	+	*		save (q_i , q_{i+1}); i:=i+1; change attribute of q to ␣
+	-	*		save (q_i , q_{i+1}); i:=i+2;
-	␣	␣		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to -

-	␣	+		% save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
-	␣	-		% save (q_i , q_{i+2}); i:=i+3;
-	+	*		save (q_i , q_{i+1}); i:=i+2;
-	-	*		save (q_i , q_{i+1}); i:=i+1; change attribute of q to ␣

% cases when at least two edges intersect or touch each other
* means all cases, e.g. ␣ + -

The coordinates of the resulted points determined by their q values can be obtained from the equation for the line w(q):

$$x(q) = x_r + (x_s - x_r) . q$$

The whale algorithm can be described as follows in fig.3.4.

```
k:=n-1; i:=0;
s_1:= x_k - x_{k-1} ; s_2:= x_s - x_r  ;  (* s, x   are vectors *)
WHILE i < n DO
BEGIN
   s_3:= x_i - x_k ; (* operation with vectors *)
   COMPUTE_VALUE(q);
   IF the vertex x_k lies on the line w(q)
     THEN
         BEGIN
             IF [s_3 x s_2]_z = 0
             THEN (* the edge x_k x_i lies on the line w(q) *)
                 generate(q with the attribute sign [s_1 x s_2]_z  )
             ELSE
                 IF [s_1 x s_2 ]_z = 0
                 THEN (* the edge x_{k-1} x_k lies on the line w(q) *)
                     generate( q with the attribute sign [s_3 x s_2]_z )
                 ELSE
                     IF [s_1 x s_2]_z .[s_3 x s_2]_z < 0
                     THEN (* "touch" *)
                         generate(q,q with attributes ␣ )
                     ELSE (* "pass" *)
                         generate(q with attribute ␣ )
         END
     ELSE
         IF an intersection of the line w(q) with the edge <x_k,x_i )
             exists
         THEN generate(q with attribute ␣ );
   s_1:= s_3; k:=i; i:=i+1;
END;
SORT ( values q );
REDUCE ( set of q values );
SELECT ( subintervals as <q_j , q_{j+1}> ∩ <0 , 1> for all j );
COMPUTE ( the end points );
```

Figure 3.4.

```
i:=1;
WHILE i <= No of intersection -1 DO
BEGIN
   IF max ( 0.0 , q_i ) <= min ( 1.0 , q_{i+1} )
      THEN save ( max ( 0.0 , q_i ) , min ( 1.0 , q_{i+1} ) );
   i:=i+2
END;
```

Figure 3.5.

The presented algorithm in fig.3.4. enables to clip a given line segment against a non-convex polygon. The algorithm is based on a similar idea as the previous one and because the polygon is non-convex some additional operations must be employed.

4. NON–CONVEX AREA CLIPPING

So far presented algorithms have solved the line clipping by convex or non-convex polygon, e.g. by areas that consist of linear edges. But plenty of applications require clipping over areas that are formed by linear edges and arcs, see fig.4.1. Provided a non-convex area is given by its vertices in the clockwise or anticlockwise order and if the edge is not linear then information whether the right or left part of the circle is to be taken from the actual vertex, see fig.4.1. It is also assumed that all vertices have different coordinates, that no vertex lies on an edge or arc and that two edges or arcs might have only a vertex as a common point.

Contrary to the previous problem the line w(q) can intersect the arc edge in two points. It partially increases the complexity of the given problem.

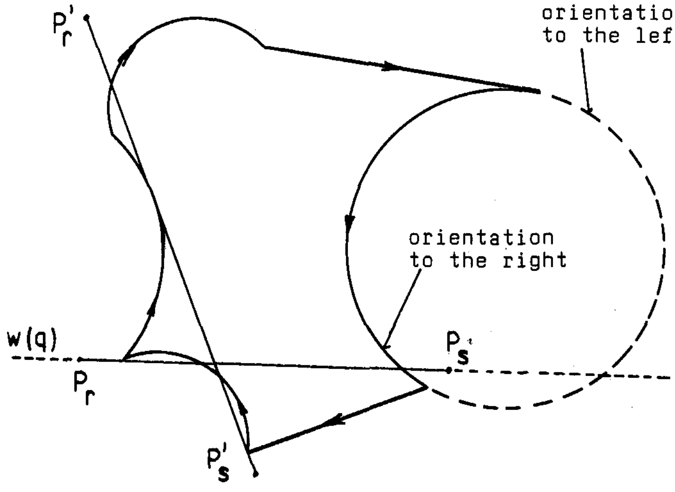


Figure 4.1.

The line w(q) is described by the parametric equation:

$$x (q) = x_r + (x_s - x_r) . q \qquad q \in (-\infty ,+\infty)$$

The procedure for finding all intersection points is similar to the previously stated algorithm, but now in case of arc edge it is necessary to solve the following equations:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty, +\infty)$$
$$(x - x_u)^2 + (y - y_u)^2 - r^2 = 0$$

where (x_u, y_u) is the centre of the given arc
2r is the diameter of this arc.

solving these equations with regard to variable q a quadratic equation

$$a q^2 + b q + c = 0$$

will be obtained, where:

$$a = (x_s - x_r)^2 + (y_s - y_r)^2$$
$$b = 2 [(x_r - x_u) \cdot (x_s - x_r) + (y_r - y_u) \cdot (y_s - y_r)]$$
$$c = (x_r - x_u)^2 + (y_r - y_u)^2 - r^2$$

In the case that the line w(q) intersects or touches the given circle two solutions are obtained, not necessarily different, as:

$$q_{1,2} = (-b \pm \sqrt{b^2 - 4ac}) / (2a)$$

Now it is necessary to determine which part of the circle forms the boundary of the given area. Because the border is oriented it can be discerned whether the arc is on the right or on the left from the connection of $x_K x_{K+1}$ points. If the line w.(q) is considered then it must be decided which intersection point ought to be taken. It is obvious that only the point which lies on the proper arc can be considered. It means that:

- if the left arc is considered then the point $x(q_i)$ will be taken into consideration if and only if

$$[s_1 \times s_2]_z > 0 \qquad i=1,2$$

- if the right arc is considered then the point $x(q_i)$ will be taken into consideration if and only if

$$[s_1 \times s_2]_z < 0 \qquad i=1,2$$

assuming that $x_K \neq x(q_i)$, $s_1 = x_{K+1} - x_K$ and $s_2 = x(q_i) - x_K$.

Of course some special situations must be solved again, e.g. when the line passes or touches the vertex x_K. In those cases the tangent vectors s_1, s_2, s_3 are determined as:

-for the arc $s_1 = [y_K - y_u , x_u - x_K]$ where (x_u, y_u) is the centre
for linear edge $s_1 = [x_K - x_{K-1} , y_K - y_{K-1}]$

-for the arc $s_3 = [y_K - y_w , x_w - x_K]$ where (x_w, y_w) is the centre
for linear edge $s_3 = [x_{K+1} - x_K , y_{K+1} - y_K]$

-for the line w(q) $s_2 = [x_s - x_r , y_s - y_r]$

The possible situations are shown in table 4.1. and in fig.4.2.

Table 4.1.

$[s_1 \times s_2]_z$	$[s_3 \times s_2]_z$	situation on fig.4.2.	type "touch"/"pass"
< 0	< 0	a	pass
< 0	> 0	b	touch
> 0	> 0	+ a	pass
> 0	< 0	+ b	touch
< 0	= 0	c	if $-s_3 . s_2 > 0$ then pass else touch
> 0	= 0	d	if $-s_3 . s_2 > 0$ then touch else pass
= 0	< 0	e	if $-s_1 . s_2 > 0$ then touch else pass
= 0	> 0	f	if $-s_1 . s_2 > 0$ then pass else touch
= 0	= 0	g	if $-s_1 . s_2 > 0$ xor $-s_3 . s_2 > 0$ then pass else touch

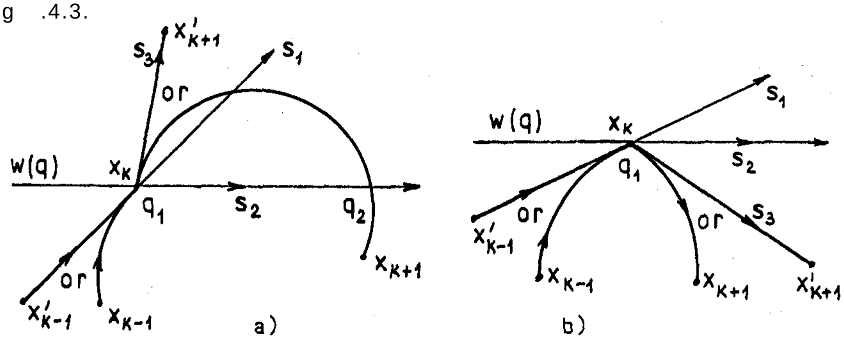
\+ for the opposite orientation of the line w(q)

If the arc is oriented to the right then the sign of the tangent vector s must be changed in some situations. It means that we can define variables a and b by the following sequences:

```
a:=[s1 x s2]z ;
IF xK-1 xK is the arc
   THEN IF a = 0
           THEN a:=-s1.s2;
           ELSE IF orientation of the arc is to the right
                   THEN a:=-a;

b:=[s3 x s2]z ;
IF xK xK+1 is the arc
   THEN IF b = 0
           THEN b:=-s3.s2;
           ELSE IF orientation of the arc is to the right
                   THEN b:=-b;
```

Now only the first four lines of table 4.1. are needed. The whole algorithm for clipping lines by non-convex areas is shown in fig .4.3.



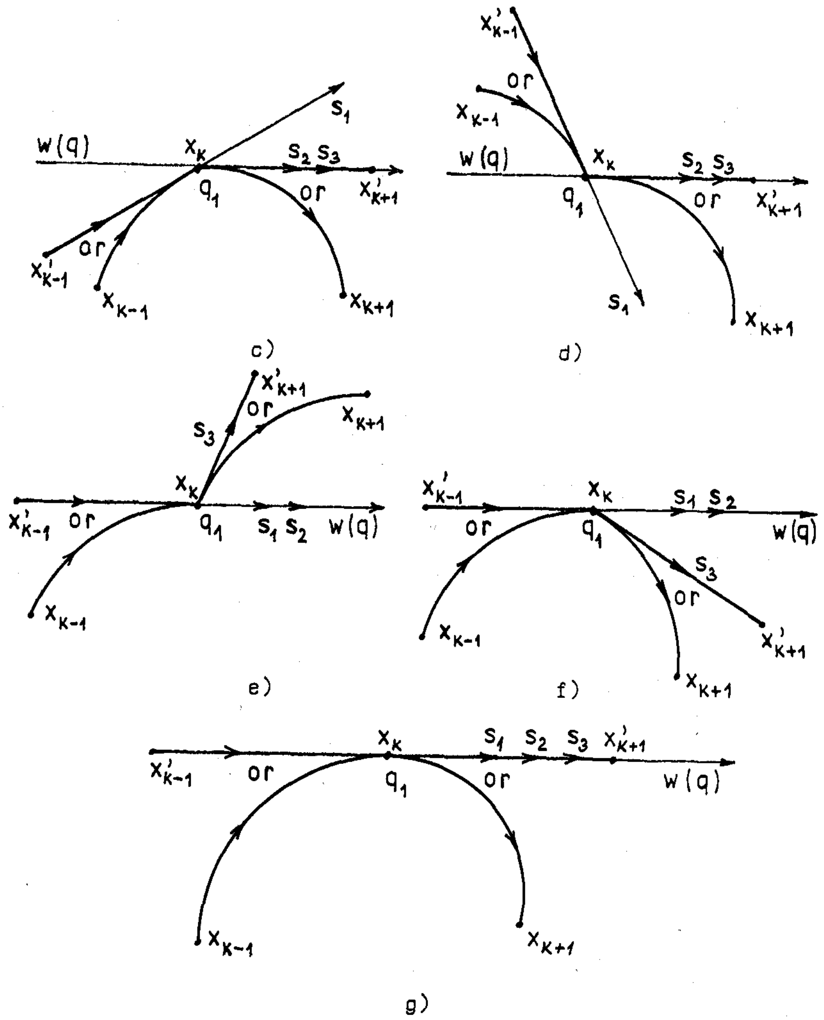


Figure 4.2.

```
PROCEDURE COMPUTE_TANGENT ( xA , xB , r , t );
BEGIN
  IF xA xB is linear
     THEN BEGIN s := xB- xA; r := [ s x s2]z ;
          END
     ELSE BEGIN s := [ yK - yW , xW - xK ]; r := [ s x s2]z ;
                  (* (xW,yW) is the centre of the given arc *)
               IF r = 0 THEN IF t THEN r:= s.s2 ELSE r:=-s.s2
                        ELSE IF the arc is oriented to the right
                               THEN r := -r
          END
END (* COMPUTE_TANGENT *);
```

```
k:=n-1; i:=0; s2:= xs- xr; (* operation with vectors *)
WHILE i < n DO
BEGIN
  IF xk lies on the line w(q) THEN
  BEGIN COMPUTE_TANGENT(xk,xi ,b,true);
        COMPUTE_TANGENT(xk-1 ,xk,a,false);
        IF xkxi is linear THEN
        BEGIN COMPUTE_VALUE( q );
              IF a.b > 0
              THEN GENERATE( q with attribute ⌴ )
              ELSE IF a.b < 0
                   THEN GENERATE( q, q with attribute ⌴ )
                   ELSE IF a = 0
                        THEN GENERATE( q with attribute sign b )
                        ELSE GENERATE( q with attribute sign a )
        END
        ELSE (* xkxi is the arc *)
        BEGIN COMPUTE_VALUES( q1, q2);
           IF a.b > 0 THEN GENERATE( q1, q2* with attribute ⌴ )
           ELSE IF a.b < 0
                THEN GENERATE( q1, q1, q2* with attribute ⌴ )
                ELSE IF a = 0
                     THEN GENERATE( q1with attribute sign b )
                     ELSE GENERATE( q1with attribute sign a )
        END
  END
  ELSE IF xkxi linear THEN
       BEGIN COMPUTE_VALUE (q);
             IF an intersection point is inside of <xkxi )
             THEN GENERATE ( q with attribute ⌴ )
       END
       ELSE BEGIN COMPUTE_VALUES( q1, q2);
                  IF an intersection point exists
                  THEN GENERATE( q1*, q2* with attribute ⌴ )
                  (* * means if the intersection point lies *)
                  (*   on the required side of the xkxi arc *)
            END
  k := i; i := i + 1;
END (* of while *);
SORT ( values q );
REDUCE ( set of q values according to table 3.1. );
SELECT ( subintervals as <qj , qj+1)∩<0 , 1> for all j );
COMPUTE ( the end points );
```

Figure 4.3.

It is obvious that the presented algorithm for clipping line by non-convex area can be easily modified for a case when the area is formed by linear segments and quadratic arcs. In this case it is necessary to define conveniently the quadratic arcs. A similar approach to the circle case can be chosen for this general case too. Generally all quadratic curves are described by the function f(x,y) together with their tangent vectors as:

$$f(x, y) = 0 \qquad\qquad s = [\, f_y \, , \, - f_x \,]$$

where: $$f(x,y) = ax^2 + by^2 + 2cxy + 2dx + 2ey + g = 0$$

If the given area consists of some holes it is necessary to apply the presented algorithm for all the given holes themselves and merge the obtained q values together.

5.CONCLUSION

The presented algorithms are based on the principle of the Liang-Barsky s algorithm. It is shown how the algorithms become more complicated if the requirements are more general. In general they do not need oriented half-planes of the clipping window. The second algorithm solves the situation when the clipping polygon is non-convex. The increase of complexity is expressed in the need to distinquish between different cases and to sort the final set of intersection points. The last presented algorithm solves the problem when the clipping area is formed by line segments and arcs. This problem has not been solved in the accessible literature as far as it is known to the author. The algorithms are fast and all special cases are properly handled.

ACKNOWLEDGMENT

The author would like to express his thanks to Prof.L.M.V. Pitteway, Dr.J.P.A.Race (Brunel Univ.,U.K.), Dr.J.E.Bresenham (Winthrop College, U.S.A.) and Dr.R.A.Earnshaw (Univ. of Leeds, U.K.) for their helpful discussions during his stay in U.K., to Miss I. Kolingerova (Inst. of Technology, Plzen) for her interest, comments and the final implementation on IBM-PC, who enabled to find unspecified special cases and errors, and to the students of Computer Graphics course that stimulated this work,

LITERATURE

[1] Cyrus, M., Beck, J., Generalized Two- and Three Dimensional Clipping, Computers & Graphics, Vol.3, 1979, No.1.,pp.23-28.

[2] Foley,J.D. ,van Dam A,, Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1982, Reading, Mass.

[3] Kilgour,A.C., Unifying Vector and Polygon Algorithms for Scan Conversion and Clipping, Report CSC 87/R7, Computer Science Dept., Univ. of Glasgow, 1987.

[4] Liang,Y .D., Barsky,B.A., An Analysis and Algorithms for Polygon Clipping, CACM 26, No.11 (November), 1984, pp.868-876.

[5] Liang,Y.D., Barsky,B.A., A New Concept and Method for Line Clipping, ACM Transaction on Graphics, Vol.3., No.1.,1984, pp.1-22.

[6] Newman,W.M., Sproull, R.F., Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, 1979, New York.

[7] Nicholl ,T .M., Lee,D.T., Nicholl, R.A., An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, ACM Computer Graphics, Vol.21, No.4., 1987, July, pp.253-262.

[8] Skala,V., A Unifying Approach to the Line Clipping Problem Solution TR-209-5-89 Computer Science Dept., Inst. of Technology Plzen, 1989.