Fast Modifications of Cohen-Sutherland algorithm for Line Segment and Line Clipping in E^2

Duc Huy Bui, Václav Skala
Departments of Informatics and Computer Science[1]
University of West Bohemia
Univerzitní 22, Box 314
306 14 Plzeň
Czech Republic

bui@kiv.zcu.cz
http://herakles.zcu.cz/~bui

skala@kiv.zcu.cz
http://yoyo.zcu.cz/~skala

Abstract

New modifications of the Cohen-Sutherland algorithm for the line and line segment clipping in E^2 are presented. The suggested algorithms are based on a new coding technique based on coding of the line direction together with coding of end-points of the clipped line segment. It allows to solve all cases more effectively. The proposed algorithms are convenient for line or line segment clipping by the rectangle. Theoretical considerations and experimental results are also presented.

Keywords: Line Clipping, Computer Graphics, Algorithm Complexity, Geometric Algorithms, Algorithm Complexity Analysis.

1. Introduction

Many algorithms for line or line segment clipping by rectangular area have been published so far, see [Ska94] for references. Line clipping problem solution is a bottleneck of many packages and applications. It would be desirable to use a faster algorithm even though it is more complex. The line clipping by rectangular window is often restricted to the usage of Cohen-Sutherland algorithm (CS) [Fol90] or its modifications that are based on some presumptions like small clipping window or more sophisticated coding technique [Sob87] etc.

The CS algorithm based on coding of the end-points of the given line segment is a very well known algorithm. It is simple and robust. It enables to detect all the cases when the line segment is completely inside of the given rectangle and some cases when the line segment is outside, see fig.1.1.

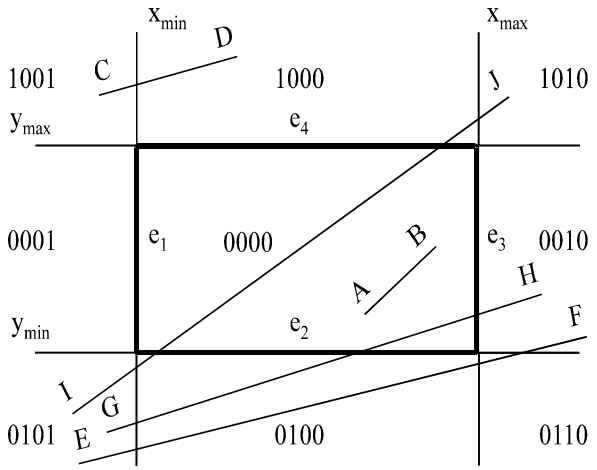

Figure 1.1

It is well known that the segments AB and CD are handled in a very simple way. However the line segments EF and GH are not distinguished at all and full intersection computations must be done. In the worst case, see line segment IJ in fig.1.1, all intersection points with each boundary line on which the rectangle edges lie are computed, see alg.1.1.

[1]Supported by the grant UWB-156/1995-6 and the grant MSMT-PG97185

```
global var x_min, x_max, y_min, y_max: real;  { clipping window size definition }
        { operators land, resp. lor are bitwise and, resp. or operators }

procedure CS_Clip ( x_A, y_A, x_B, y_B: real);
var     x, y: real;
        c, c_A, c_B: integer;
procedure CODE (x, y: real; var c: integer); { implemented as a macro }
begin   c := 0;
        if x < x_min then c := 1 else if x > x_max then c := 2;
        if y < y_min then c := c + 4 else if y > y_max then c := c + 8;
end { of CODE };

begin   CODE (x_A, y_A, c_A); CODE (x_B, y_B, c_B);
        if (c_A land c_B) ≠ 0 then EXIT;  { the line segment is outside the clipping rectangle }
        if (c_A lor c_B) = 0 then begin DRAW_LINE (x_A, y_A, x_B, y_B); EXIT;  end;
                                        { the line segment is inside of the clipping rectangle }
        repeat if  c_A ≠ 0 then c = c_A else c = c_B;
                if (c land '0001') ≠ 0 then { divide line at the left edge }
                begin  y := y_A + (x_min - x_A)*(y_B - y_A) / (x_B - x_A);
                        x := x_min;
                end
                else    if (c land '0010') ≠ 0 then
                        begin  y := y_A + (x_max - x_A)*(y_B - y_A) / (x_B - x_A);
                                x := x_max;
                        end
                        else    if (c land '0100') ≠ 0 then
                                begin  x := x_A + (y_min - y_A)*(x_B - x_A) / (y_B - y_A);
                                        y := y_min;
                                end
                                else    if (c land '1000') ≠ 0 then
                                        begin  x := x_A + (y_max - y_A)*(x_B - x_A) / (y_B - y_A);
                                                y := y_max;
                                        end;
                if  c = c_A then begin x_A := x; y_A := y; CODE (x_A, y_A, c_A); end
                                else begin x_B := x; y_B := y; CODE (x_B, y_B, c_B); end;
                if (c_A land c_B) ≠ 0 then EXIT;
        until (c_A lor c_B) = 0;
        DRAW_LINE (x_A, y_A, x_B, y_B);
end { of CS_Clip };
```

Cohen-Sutherland algorithm
Algorithm 1.1

The well known algorithm for the line clipping is the Liang-Barsky algorithm (LB) [Fol90]. It is based on clipping of the given line by each boundary line on which the rectangle edge lies. The given line is parametricaly represented. At the beginning the parameter t is limited by interval $(-\infty,+\infty)$ and then this interval is subsequently truncated by all the intersection points with each boundary line of the clipping rectangle, see alg.1.2. It can be seen that an additional trivial rejection test (function TEST) is used to avoid calculation of all four parametric values for lines, which do not intersect the clipping rectangle.

```
global var xmin, xmax, ymin, ymax: real;
        { clipping window size }
        { given values }

function TEST ( p, q : real;
                var t1, t2 : real):boolean;
var     r : real;
begin   TEST := true;
   if p < 0 then
   begin r := q / p;
         if r > t2 then TEST := false
                   else if r > t1 then t1:= r
   end else
        if p > 0 then
        begin r := q / p;
              if r < t1 then TEST := false
                        else if r < t2 then t2:= r;
        end else if q < 0 then TEST := false
end { of TEST };

procedure LB_Clip ( xA, yA, xB, yB: real);
var     t1, t2, Δx, Δy : real;
begin
   t1:= –∞; t2:= +∞;  Δx := xB - xA;
   if TEST(–Δx, xA - xmin, t1, t2) then
      if TEST(Δx, xmax - xA, t1, t2) then
         begin
            Δy := yB - yA;
             if TEST(–Δy, yA - ymin, t1, t2) then
                if TEST(Δy, ymax - yA, t1, t2) then
                   begin xB := xA + Δx * t2;
                         yB := yA + Δy * t2;
                         xA := xA + Δx * t1;
                         yA := yA + Δy * t1;
                    DRAW_LINE(xA, yA, xB, yB)
                   end
         end
end { of LB_Clip };
```

Liang-Barsky algorithm
Algorithm 1.2

2. Proposed methods

The proposed algorithm (LSSB) for the line segment clipping is based on the CS algorithm but the arithmetic sum of end points' codes and a new coding technique for the line segment direction are used in order to remove cases, which the original CS algorithm is unable to distinguish.

Let us assume characteristic situations from fig.2.1 and denote c_A, c_B as CS codes of line segment's end points; α,γ,η,ω areas as corner areas and β,θ,ξ,δ as side areas.

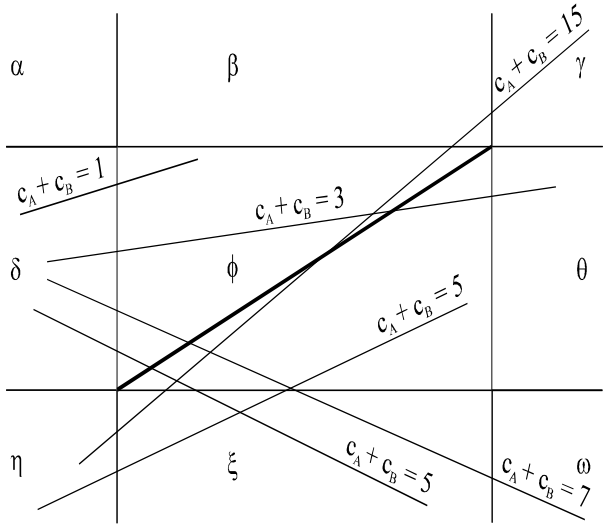

Figure 2.1

By testing arithmetic sum of end points' codes we can distinguish the following situations:

- One end point of the line segment is inside of the clipping rectangle and the second one is in the side area (the cases $c_A + c_B \in \{1,2,4,8\}$). In these cases one intersection point is computed.
- The end points of the line segment are in the opposite side areas (the cases $c_A + c_B \in \{3,12\}$). In these cases two intersection points are computed (the clipping edges are already determined).
- One end point of the line segment is in the corner area of the clipping rectangle and the second one is in the side area (the cases $c_A + c_B \in \{7,11,13,14\}$). In these cases one clipping edge (if exists) is already determined and the second one is opposite or neighboring to it.
- The cases when $c_A + c_B \in \{5,6,9,10\}$. There are two possible situations:
  a) The end points of the line segment are in the near-by side areas, i.e (δ,β), (δ,ξ), (θ,β), (θ,ξ). Two clipping edges (if exist) are already determined.
  b) One end point of the line segment is inside of the clipping rectangle and the second one is in the corner area. The only one intersection point can lie on the

horizontal or vertical edge of clipping rectangle.

- The end points of the line segment are in the opposite corner areas (the cases $c_A + c_B = 15$). The comparison between directions of the given line and the clipping rectangle's diagonal decides which edges (horizontal or vertical) are used to compute the intersection points.

Distinguishing all those cases enables avoidance of unnecessary calculation and causes considerable speed-up. Detailed description of the proposed LSSB algorithm is shown in alg. 2.1.

```
procedure LSSB_Clip ( xA, yA,xB,yB: real);
var     Δx, Δy, k, m, z : real; cA , cB : integer;
        procedure  CODE (x,y: real; var c: integer); { implemented as a macro }
        begin  c := 0; if x < xmin then c := 1 else if x > xmax then c := 2;
                  if y < ymin then c := c + 4 else if y > ymax then c:= c + 8;
        end { of CODE };
begin
   CODE (xA, yA, cA); CODE (xB, yB, cB);
   if (cA land cB) <> 0 then EXIT; { the line segment is outside }
   if (cA lor cB) = 0 then { the line segment is inside of the clipping rectangle }
       begin DRAW_LINE(xA, yA, xB, yB); EXIT; end;
   Δx := xB - xA; Δy := yB - yA;
   case cA + cB of { see figure 3.k }
     1: if cA = 1 then begin xA := xmin; yA := ( xmin - xB )* Δy / Δx + yB; end
                        else begin xB := xmin; yB := ( xmin - xA )* Δy / Δx + yA; end;
     3: begin k := Δy / Δx; yA := ( xmin - xA )* k + yA; xA := xmin;
              yB := ( xmax - xB )* k + yB; xB := xmax; end;
     5: begin  k := Δy / Δx; z := ( xmin - xA )* k + yA;
          if z < ymin then case cA of
                                  0: begin  xB := xB + ( ymin - yB )/k; yB := ymin; end;
                                  5: begin  xA := xA + ( ymin - yA )/k; yA := ymin; end;
                               else  EXIT;{ the line segment is outside } end
                      else case cA of
                                  0: begin xB := xmin; yB := z; end;
                                  1: begin xB :=  xB + ( ymin - yB )/k; yB := ymin; xA := xmin; yA := z; end;
                                  4: begin xA :=  xA + ( ymin - yak )/k; yak := yin; xB := xmin; yB := z; end;
                                  5: begin xA := xmin; yA := z;end; end;
        end;
     7: case  cA  of
          1: begin k := Δy / Δx; yA := ( xmin - xB )* k + yB;
               if yA < ymin then EXIT;{ the line segment is outside }
               xA := xmin; yB := ( xmax - xmin )* k + yA;
               if yB < ymin then begin xB := ( ymin - yB )/k + xmax; yB := ymin; end else xB := xmax;
             end; { similarly for cases cA =  2, 5, 6 }
        end;
    15: case cA of
          5: if Δy*(xmax - xmin ) < Δx*(ymax - ymin)  then
             begin k := Δy / Δx; yA := ( xmin - xB )* k + yB;
                if yA > ymax then EXIT; { the line segment is outside }
                yB := ( xmax - xmin )* k + yA;
                 if yB < ymin then EXIT; { the line segment is outside }
```

```
            if yA < ymin then begin xA := ( ymin - yA )/k + xmin; yA := ymin; xB := xmax; end
            else begin xA := xmin;
                  if yB > ymax then begin xB := ( ymax - yB )/k + xmax; yB := ymax; end
                              else xB := xmax;
                 end;
          end else
          begin m := Δx /Δy; xA := ( ymin - yB )* m + xB;
            if xA > xmax then EXIT; { the line segment is outside }
            xB := ( ymax - ymin )* m + xA;
            if xB < xmin then EXIT; { the line segment is outside }
            if xA < xmin then begin yA := ( xmin - xA )/m + ymin; xA := xmin; yB := ymax; end
            else begin yA := ymin;
                  if xB > xmax then begin yB := ( xmax - xB )/m + ymax; xB := xmax; end
                              else yB := ymax;
                 end;
          end
         { similarly for cases cA =  6, 9, 10 }
       end;
    { cases 2, 4, 8 are similar  as case 1, cases 6, 9, 10  are similar as case 5 }
    { cases 11, 13, 14  are similar as case 7, case 12 is similar case 3 }
  end { of case cA + cB };
  DRAW_LINE (xA, yA, xB, yB);{ EXIT means leave the procedure }
end {of LSSB_Clip };
```

LSSB algorithm for line segment clipping
Algorithm 2.1

In many applications it is necessary to clip lines instead of line segments. It can be shown that CS algorithm is faster than the Liang-Barsky algorithm (LB) for line segment clipping,but for line clipping the LB algorithm is more convenient and faster. Therefore, a similar consideration was made for the LB algorithm that resulted in a new algorithm for line clipping denoted as LSB algorithm. The proposed LSB algorithm for line clipping is based on a new coding technique for line direction. The comparison between directions of the given line and clipping rectangle's diagonal decides which edges (horizontal or vertical) are used to compute the intersection points between the line and the clipping window, see alg.2.2. This comparison is used to avoid the calculation of the intersection points that do not lie on the boundary edges of the clipping rectangle.

```
procedure LSB_Clip ( xA, yA,xB,yB: real);
var     Δx, Δy, k, m: real;
begin  Δx := xB - xA;
   if  Δx = 0 then begin
        if (xA < xmin) or (xA > xmax) then EXIT; { the line is outside of the clipping rectangle}
        yA := ymin; yB := ymax; DRAW_LINE(xA, yA, xB, yB); end;
   Δy := yB - yA;
   if Δx > 0 then
     if Δy > 0 then
       if (Δy*(xmax - xmin ) < Δx*(ymax - ymin) ) then
       begin k := Δy / Δx; yA := ( xmin - xB )* k + yB;
         if yA > ymax then EXIT; { the line is outside of the clipping rectangle}
         yB := ( xmax - xmin )* k + yA;
```

```
    if y_B < y_min then EXIT; { the line is outside of the clipping rectangle}
    if y_A < y_min then begin x_A := ( y_min - y_A )/k + x_min; y_A := y_min; x_B := x_max; end
   else begin x_A := x_min; if y_B > y_max then begin x_B := ( y_max - y_B )/k + x_max; y_B := y_max; end
                                          else  x_B := x_max; end;
  end else
  begin m := Δx /Δy; x_A := ( y_min - y_B )* m + x_B;
    if x_A > x_max then EXIT; { the line is outside of the clipping rectangle}
     x_B := ( y_max - y_min )* m + x_A;
    if x_B < x_min then EXIT; { the line is outside of the clipping rectangle}
     if ( x_A < x_min ) then begin y_A := ( x_min - x_A )/m + y_min; x_A := x_min; y_B := y_max; end
    else begin y_A := y_min; if x_B > x_max then begin y_B := ( x_max - x_B )/m + y_max; x_B := x_max; end
                                          else  y_B := y_max; end;
   end; { similarly for the other cases }
   DRAW_LINE(x_A, y_A, x_B, y_B);
end; { of LSB_Clip }
```

LSB algorithm for line clipping
Algorithm 2.2

3. Theoretical considerations and experimental results

It is necessary to derive the expected theoretical properties of the proposed algorithms and prove their superiority over traditional algorithms. It should be mentioned here that algorithm efficiency can differ from computer to computer because of different instruction timing. For the used PC 586/133 MHz we have for 128.10^6 operations (:=, <, ±, *, /) the following timing (6.7, 11.2, 2.3, 2.3, 19.9). For algorithm efficiency considerations between CS and new LSSB algorithms we must consider several situations, see fig.3.1. It can be seen that the line segments s_1 and s_2 are handled exactly as in the original CS algorithm. For the other cases it is necessary to derive CS and LSSB algorithms complexities and compare them. Let us introduce the coefficient of efficiency ν_1 as

$$\nu_1 = \frac{T_{CS}}{T_{LSSB}}$$

All significant cases are shown in fig.3.1.

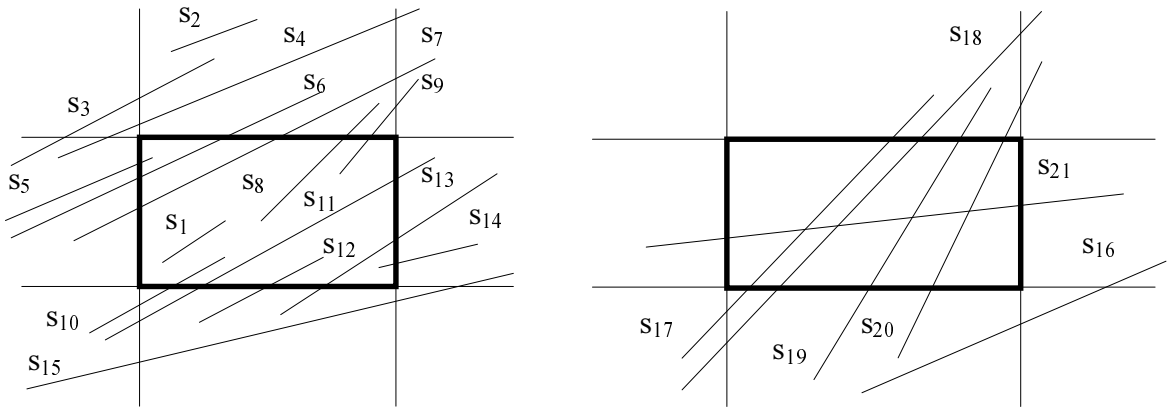

Figure 3.1.

case	Theoretical considerations												Experimental results		
	C S						LSSB						ν_1	CS	LSSB	ν_1
	=	<	±	x	/	t[s]	=	<	±	x	/	t[s]		t[s]	t[s]	
s_1	2	10	0	0	0	125,40	2	10	0	0	0	125,40	**1,00**	150,28	150,28	**1,00**
s_2	4	9	2	0	0	132,20	4	9	2	0	0	132,20	**1,00**	151,70	151,70	**1,00**
s_3	11	17	6	1	1	300,00	8	12	6	1	1	223,90	**1,34**	238,41	210,71	**1,13**
s_4	12	17	6	1	1	306,70	9	12	6	1	1	230,60	**1,33**	238,90	213,85	**1,12**
s_5	9	17	4	1	1	282,00	7	11	5	1	1	203,70	**1,38**	236,98	206,10	**1,15**
s_6	19	27	9	2	2	494,60	12	12	8	1	2	275,10	**1,80**	355,44	245,39	**1,45**
s_7	21	33	14	3	3	608,80	13	13	10	2	2	299,90	**2,03**	462,58	272,58	**1,70**
s_8	9	22	5	1	1	340,30	7	12	6	1	1	217,20	**1,57**	250,88	222,53	**1,13**
s_9	17	32	10	2	2	539,50	10	12	8	1	2	261,70	**2,06**	361,15	258,90	**1,39**
s_{10}	16	27	10	2	2	476,80	10	10	8	1	2	239,30	**1,99**	327,53	237,91	**1,38**
s_{11}	24	37	14	3	3	673,70	13	12	8	2	2	284,10	**2,37**	440,28	267,75	**1,64**
s_{12}	9	20	5	1	1	317,90	7	11	6	1	1	206,00	**1,54**	240,33	211,87	**1,13**
s_{13}	16	30	9	2	2	508,10	12	12	8	1	2	275,10	**1,85**	356,70	257,20	**1,39**
s_{14}	9	20	4	1	1	315,60	7	12	5	1	1	214,90	**1,47**	248,96	221,54	**1,12**
s_{15}	19	26	10	2	2	485,70	9	11	6	1	1	219,40	**2,21**	327,47	210,00	**1,56**
s_{16}	11	19	5	1	1	320,10	8	12	6	1	1	223,90	**1,43**	240,33	221,54	**1,08**
s_{17}	24	39	15	3	3	698,40	12	12	9	2	1	259,90	**2,69**	442,86	231,87	**1,91**
s_{18}	31	49	20	4	4	890,90	13	15	11	4	1	309,40	**2,88**	554,78	266,81	**2,08**
s_{19}	16	32	10	2	2	532,80	11	10	9	2	1	230,80	**2,31**	358,79	223,46	**1,61**
s_{20}	24	42	15	3	3	732,00	12	13	9	2	1	271,10	**2,70**	465,93	245,39	**1,90**
s_{21}	15	28	8	2	2	476,70	11	10	7	2	1	226,20	**2,11**	353,46	222,53	**1,59**

Table 3.1

For experimental verification of the proposed algorithm all cases were tested and 64.10^6 different line segments were randomly generated for each considered case. Table 3.1 shows that the LSSB algorithm is theoretically and experimentally significantly faster in all considered non-trivial cases(different from s_1 and s_2). It can be seen that the speed-up varies from 1.1 to 2.0 approximately.

Similarly, for algorithm efficiency considerations between LB and new LSB algorithms we must distinguish fundamental situations shown in the fig.3.2.

Let us introduce the coefficient of efficiency ν_2 as :

$$\nu_2 = \frac{T_{LB}}{T_{LSB}}$$

The theoretical estimation and experimental results are presented in tab.3.2. This table shows, that the LSB algorithm is significantly faster than LB algorithm in all cases. It can be seen that the speed up varies from 1.2 to 3.0 approximately

A theoretical comparison with the Nicholl algorithm [Nic87] was also done. Although the Nicholl algorithm achieves the similar efficiency as the proposed algorithm for the line segment clipping, unfortunately it cannot be used for clipping of the given line generally.

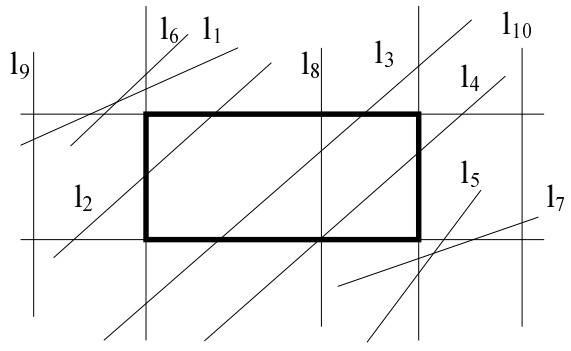

Figure 3.2

case	Theoretical considerations												Experimental results		
	LB						LSB						LB	LSB	ν_2
	=	<	±	x	/	t[s]	=	<	±	x	/	t[s]	t[s]	t[s]	
l_1	10	13	6	0	4	305,60	5	6	8	4	1	148,10	**2,06**	303,46	191,65	**1,58**
l_2	15	14	10	4	4	368,70	8	7	10	4	2	203,80	**1,81**	349,18	236,98	**1,47**
l_3	16	14	10	4	4	375,40	7	8	8	4	1	183,90	**2,04**	351,10	216,26	**1,62**
l_4	15	14	10	4	4	368,70	8	8	10	4	2	215,00	**1,71**	349,18	248,08	**1,41**
l_5	9	9	5	0	3	232,00	4	5	6	3	1	123,30	**1,88**	241,32	167,58	**1,44**
l_6	10	13	6	0	4	305,60	4	5	6	3	1	123,30	**2,48**	303,41	167,14	**1,82**
l_7	9	9	5	0	3	232,00	5	6	8	4	1	148,10	**1,57**	241,32	191,21	**1,26**
l_8	12	13	10	4	2	297,80	3	3	1	0	0	56,00	**5,32**	297,64	98,74	**3,01**
l_9	3	3	2	0	0	58,30	1	2	1	0	0	31,40	**1,86**	101,65	85,71	**1,19**
l_{10}	3	6	3	0	0	94,20	1	3	1	0	0	42,60	**2,21**	134,40	96,81	**1,39**

Table 3.2.

4. Conclusion

The new line segment clipping algorithm (LSSB) and line clipping algorithm (LSB) against a given rectangle in E^2 were developed, verified and tested. The proposed LSSB and LSB algorithms are convenient for those applications, where many lines or line segments must be clipped. The proposed approach gives similar algorithms for line and line segment clipping. The proposed algorithms claim superiority over the CS and LB algorithm and experiments proved, that the speed up can be considered up to 2 times for line segment clipping and 3 times for line clipping in some cases.

The presented modifications of the well-known algorithms proved, that the approach "test first and compute after all tests" can bring a significant speed-up even with the known algorithms. There is a hope, that the presented modifications of the CS algorithm can be implemented in hardware, too.

5. Acknowledgments

The authors would like to express their thanks to all who contributed to this work, especially to recent Ms. and Ph.D. students of Computer Graphics courses at the University of West Bohemia in Pilsen, who stimulated this work.

6. References

[Fol90] Foley,D.J., van Dam,A., Feiner,S.K., Huges,J,F.: Computer Graphics - Principles and Practice, Addison Wesley, 2nd ed., 1990.

[Nic87] Nicholl,T.M., Lee D.T., Nicholl R.A.: An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis, Computers & Graphics, Vol.21, No.4, Pergamon Press, pp.253-262, 1987.

[Ska94] Skala,V.: O(lg N) Line Clipping Algorithm in E^2, Computers & Graphics, Vol.18, No.4, Pergamon Press, pp.517-524, 1994.

[Sob87] Sobleow,M.S.,Pospisil,P.,Yang,Y.H.: A Fast Two Dimensional Line Clipping Algorithm via Line Encoding, Computers & Graphics, Vol.11, No.4, pp.459-467, 1987.