The Visual Computer

Fast algorithms for clipping lines and line segments in E^2

Duc Huy Bui, Vaclav Skala

Department of Informatics and Computer Science,
University of West Bohemia, Univerzitni 22, Box 314,
306 14 Plzen, Czech Republic
bui@kiv.zcu.cz http://herakles.zcu.cz/~bui
skala@kiv.zcu.cz http://herakles.zcu.cz/~skala

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

Key words: Line clipping – Computer graphics – Algorithm complexity – Geometric algorithms – Algorithm complexity analysis

Correspondence to: D.H. Bui

1 Introduction

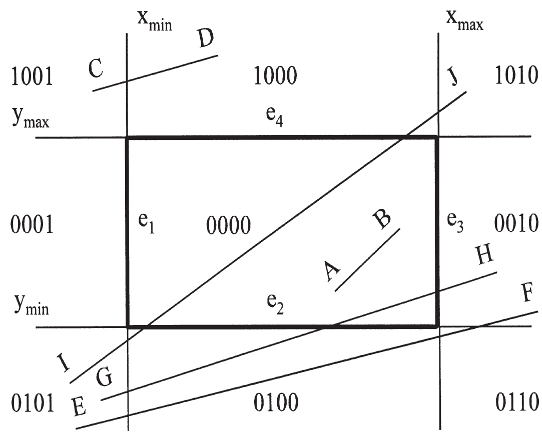
Many algorithms for clipping a line or a line segment by rectangular area have been published [Ska94], [Ska96], [Ska97]. Line clipping by rectangular window is often restricted to the use of the Cohen-Sutherland (CS) algorithm [Fol90] or its modifications that use a small clipping window or a more sophisticated coding technique [Sob87], etc. Solving the line clipping problem is a bottleneck in many packages and applications. It would be desirable to use a faster, though even more complex algorithm.

The CS algorithm, based on coding the end-points of the given line segment, is simple and robust. It detects all the cases when the line segment is completely inside a given rectangle and some cases when the line segment is outside (Fig. 1).

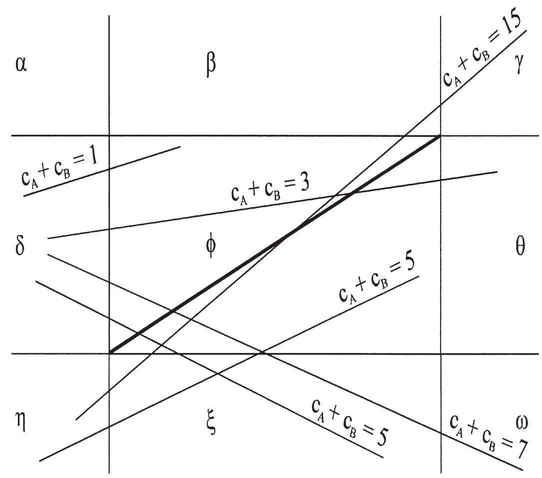
It is well known that segments *AB* and *CD* are handled in a very simple way. However, the line segments *EF* and *GH* are not recognized at all, and full intersection computations are necessary. In the worst case, (see line segment *IJ* in Fig. 1), all intersection points with each boundary line on which the rectangle edges lie are computed (Algorithm 1).

```
global var xmin, xmax, ymin, ymax: real; {clipping
window size definition}
 {operators land and lor are bitwise and and or
 are operators}
procedure CS_Clip (xA, yA, xB, yB: real);
var x, y: real;
 c, 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 the clipping rectangle}
 if (cA lor cB)=0 then begin DRAW_LINE
 (xA, yA, xB, yB); EXIT; end; {the line segment
 is inside of the clipping rectangle}
 repeat if cA≠0 then c=cA else c=cB;
  if (c land '0001')≠0 then {divide line at the
  left edge}
  begin y:=yA+(xmin−xA)*(yB−yA)/(xB−xA);
        x:=xmin;
  end
```



1



2

Fig. 1. A rectangle and line segments in various positions

Fig. 2. Line segments in characteristic situations

```
  else if (c land '0010')≠0 then
    begin y:=y_A+(x_max–x_A)*(y_B–y_A)/(xB–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};
```

Algorithm 1 – the Cohen-Sutherland algorithm

The Liang-Barsky (LB) algorithm is a well-known algorithm for line clipping [Fol90]. It is based on clipping a given line by each boundary line on which the rectangle edge lies. The given line is represented parametrically. At the beginning, the parameter t is limited by interval $(-\infty, +\infty)$, and then this interval is subsequently truncated by all the points intersecting each boundary line of the clipping rectangle (Algorithm 2). An additional trivial rejection test (function TEST) is used to avoid calculating all four parametric values for the lines, that do not intersect the clipping rectangle.

```
global var x_min, x_max, y_min, y_max: real;
    {clipping window size}
    {given values}

function TEST (p, q: real;
      var t_1, t_2: real):boolean;
var r:real;
begin TEST:=true;
  if p<0 then
  begin r:=q/p;
      if r>t_2 then TEST:=false
        else if r>t_1 then t_1:=r
  end else
    if p>0 then
    begin r:=q/p;
      if r<t_1 then TEST:=false
        else if r<t_2 then t_2:=r;
    end else if q<0 then TEST:=false
end {of TEST};

procedure LB_Clip (x_A, y_A, x_B, y_B: real);
var t_1, t_2, 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+(x*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};
```

Algorithm 2 – the Liang-Barsky algorithm

2 Proposed methods

The line segment suggested algorithm (LSSA) for line-segment clipping is based on the CS algorithm, but the arithmetic sum of end point codes and a new coding technique for the line segment direction are used in order to remove cases that the original CS algorithm is unable to recognize.

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

By testing the arithmetic sum of the end-point codes, we can distinguish the following situations:

- One end point of the line segment is inside the clipping rectangle, and the second one is in the side area (the cases $c_A+cB\in\{1, 2, 4, 8\}$). 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 have already been 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 it exists) has already been determined, and the second one is opposite or neighboring it.
- When $c_A+c_B\in\{5, 6, 9, 10\}$, there are two possible situations:
  1. The end points of the line segment are in the nearby side areas, i.e., (δ, β), (δ, ξ), (θ, β), (θ, ξ). Two clipping edges (if they exist) have already been determined.
  2. One end point of the line segment is inside of the clipping rectangle, and the second one is in the corner area. Only one intersection point can lie on the horizontal or vertical edge of the clipping rectangle.
- The end points of the line segment are in the opposite corner areas (the cases c_A+c_B=15). The comparison of the directions of the given line and the clipping rectangle diagonal decides which edges (horizontal or vertical) are used to compute the intersection points.

Recognizing all these cases avoids unnecessary calculation and causes considerable speed-up. A detailed description of the proposed algorithm is shown in Algorithm 3.

```
procedure LSSA_Clip (xA, yA, xB, yB: real);
var Δx, Δy, k, m, r: 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
 the clipping rectangle}
  begin DRAW_LINE(xA, yA, xB, yB); EXIT;
  end;
 Δx:=xB−xA; Δy:=yB−yA;
 case cA+cB of {see Fig. 3}
  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; r:=(xmin−xA)*k+yA;
     if r<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:=r; end;
      1: begin xB:=xB+(ymin–yB)/k; yB:=ymin;
         xA:=xmin; yA:=r; end;
      4: begin xA:=xA+(ymin–yak)/k; yak:=ymin;
         xB:=xmin; yB:=r; end;
      5: begin xA:=xmin; yA:=r;end; end;
  end;
 7: case cA of
    1: begin k:=(y/(x; 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 to case 1, cases 6, 9,
   10 are similar as case 5}
   {cases 11, 13, 14 are similar as case 7, case 12
   is similar to case 3}
  end {of case cA+cB};
  DRAW_LINE (xA, yA, xB, yB); {EXIT means
  leave the procedure}
end {of LSSA_Clip};
```

Algorithm 3 –
the LSSA algorithm for line-segment clipping

In many applications it is necessary to clip lines instead of line segments. It can be shown that the CS algorithm is faster than the LB algorithm 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 called the line suggested algorithm (LSA). The LSA algorithm is based on a new coding technique for line direction. The comparison of the directions of the given line and the diagonal of the clipping rectangle decides which edges (horizontal or vertical) are used to compute the intersection points between the line and the clipping window (Algorithm 4). This comparison is used to avoid calculating the intersection points that do not lie on the boundary edges of the clipping rectangle.

```
procedure LSA_Clip (xA, yA, xB, yB: real);
var Δx, Δy, k, m: real;
begin Δx:=xB–xA;
  if Δx=0 then begin
   if ΔxA<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 $m:=\Delta x/\Delta y$; $x_A=(y_{\min}-y_B)*m+x_B$;
 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 LSA_Clip}

Algorithm 4 – the LSB algorithm for line clipping

3 Theoretical considerations and experimental results

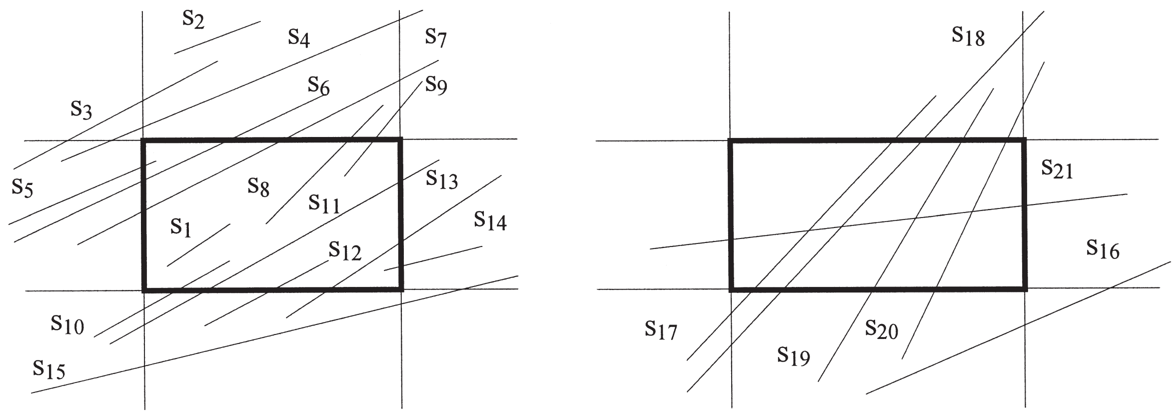
It is necessary to derive the expected theoretical properties of the proposed algorithms and prove their superiority to traditional algorithms. It should be mentioned here that algorithm efficiency can differ from computer to computer because of different instruction times. For a PC 586/133 MHz, we have the following timing: 6.7, 11.2, 2.3, 2.3, and 19.9 S, for 128.10^6 operations (:=,<, ±, *, /). For algorithm efficiency considerations for CS and the LSSA algorithms, we must consider several situations (Fig. 3). 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 LSSA algorithm complexities and compare them. Let us introduce a coefficient of efficiency ν_1 as

$$\nu_1 = \frac{T_{CS}}{T_{LSSA}}.$$

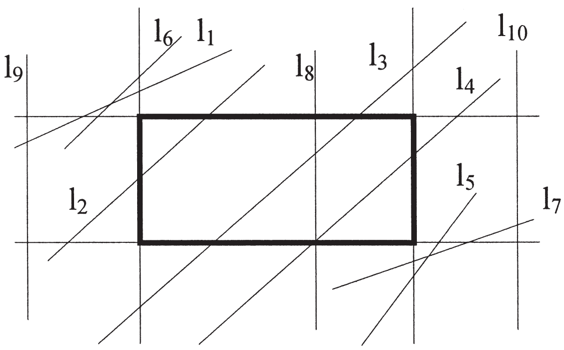
All significant cases are shown in Fig. 3.

Table 1. Comparison of the Cohen-Sutherland and the line segment suggested algorithms

Case	Theoretical considerations													Experimental results		
	CS						LSSA						ν_1	CS t[s]	LSSA t[s]	ν_1
	=	<	±	×	/	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**



3



4

Fig. 3. Line segments – significant cases

Fig. 4. Line segments – basic cases

Table 2. Comparison of the Liang Barsky algorithm and the line suggested algorithm

Case	Theoretical considerations													Experimental results		
	LB						LSA						ν_2	LB t[s]	LSA t[s]	ν_2
	=	<	±	×	/	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**

For experimental verification of the proposed algorithm, all cases were tested, and 64.10^6 different line segments were randomly generated for each case considered. Table 3 shows that the LSSA algorithm is theoretically and experimentally significantly faster in all considered nontrivial cases (different from s_1 and s_2). It can be seen that the LSSA algorithm is approximately 1.1 to 2.0 times faster.

Similarly, to consider the efficiency of the LB and new LSA algorithms, we must recognize the fundamental situations shown in the Fig. 3.
Let us introduce the coefficient of efficiency ν_2 as:

$$\nu_2 = \frac{T_{LB}}{T_{LSA}}.$$

The theoretical estimations and the experimental results are presented in Tabel 2. This table shows that the LSB algorithm is significantly faster than the LB algorithm in all cases – from 1.2 to 3.0 times faster, approximately.
A theoretical comparison with the Nicholl algorithm [Nic87] was also done. Although the Nicholl algorithm achieves an efficiency similar to that of the proposed algorithm for the line segment clipping, it unfortunately cannot be used for clipping a given line generally.

4 Conclusion

The new line-segment clipping algorithm (LSSA) and the line clipping algorithm (LSA) against a given rectangle in E^2 were developed, verified, and tested. The proposed LSSA and LSA algorithms are convenient for those applications in which 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 to the CS and LB algorithms, and experiments proved that the speed-up can be considered up to twice as fast for line segment clipping and three times as fast for line clipping in some cases.
Our 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 familiar algorithms. There is hope that these modifications of the CS algorithm can be implemented in hardware, too.

Acknowledgements. The authors would like to express their thanks to all who contributed to this work, especially to recent Ms and PhD students of the Computer Graphics courses at the University of West Bohemia in Pilsen, who stimulated this work. This work was supported by The Ministry of Education of the Czech Republic – projects PG 97193, PG 97185 and VS 97155.

References

[Fol90] Foley DJ, van Dam A, Feiner SK, Huges JF. Computer Graphics – Principles and Practice. Addison Wesley, publishing 2nd ed., 1990, USA

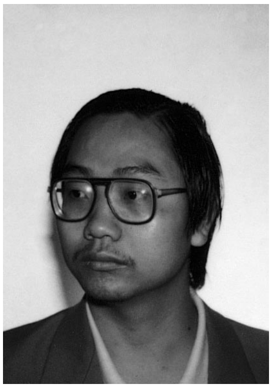
[Nic87] Nicholl TM, Lee DT, Nicholl RA (1987) An efficient new algorithm for 2-D line clipping: its development and analysis. SIGGRAPH Proceedings, 21:253–262

[Ska94] Skala V (1994) O(lg N) Line clipping algorithm in E^2. Comput Graph 18:517–524

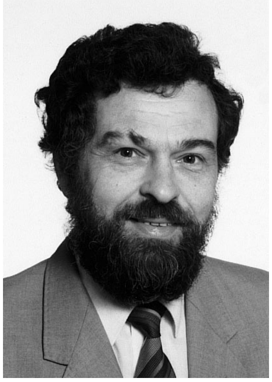
[Ska96] Skala V (1996) An efficient algorithm for line clipping by convex and non-convex polyhedra in E3. Comput Graph Forum 15:61–68

[Ska97] Skala V (1997) A fast algorithm for line clipping by convex polyhedron in E3. Comput Graph 21:209–214

[Sob87] Sobleow MS, Pospisil P, Yang YH (1987) A fast two dimensional line clipping algorithm via line encoding. Comput Graph 11:459–467



BUI DUC HUY is a PhD student of Computer Science at the Department of Computer Science and Engineering, University of West Bohemia in Pilsen, Czech Republic. He graduated from the University of West Bohemia in Pilsen in 1995. His current research interests are algorithms for computer graphics, medical and technical data visualisation and scientific computation, distributed and parallel processing.



VÁCLAV SKALA is a Professor of Computer Science at the Department of Computer Science and Engineering, University of West Bohemia in Pilsen, Czech Republic. He graduated at the Institute of Technology in Pilsen in 1975 and received his Ph.D in 1981. He is the Co-chair and organizer of the annual conferences WSCG – The International Conference in Central Europe on Computer Graphics and Visualisation held in Pilsen, Czech Republic, and a member of EUROGRAPHICS, ACM, and IEEE. His current research interests are algorithms for computer graphics, medical and technical data visualisation and scientific computation, distributed and parallel processing.