SCCG'99 Int.Conf. proceedings, Budmerice, Slovak Republic, pp.221-228, 1999

New Fast Line Clipping Algorithm in E^2 with O(lgN) Complexity

Duc Huy Bui[1], Václav Skala[2]
Department of Computer Science and Engineering[3]
University of West Bohemia[4]

[4] Univerzitni 22, Box 314, 306 14 Plzeň, Czech Republic

Abstract

New faster line clipping algorithm in E^2 against a convex polygon with *O(lgN)* complexity is presented. The main advantage of the presented algorithm is the principal acceleration of the line clipping problem solution. A comparison of the proposed algorithm with others shows a significant improvement in run-time. Experimental results for selected known algorithms are also shown.

Keywords: Line Clipping, Convex Polygon, Computer Graphics, Algorithm Complexity.

1 Introduction

Many algorithms for clipping lines against convex or non-convex polygons in E^2 with many modifications derived from well known Liang-Barsky's [LIA83a], [LIA84a] and Cyrus-Beck's [CYR79a] algorithms have been published, see [SKA89a], [SKA89b] and [FOL90a]. All of them have the same complexity *O(N)*, with an exception of Rappaport's algorithm [RAP91a] for convex polygon clipping, that has *O(lgN)* complexity. Their speed is determined by more or less clever implementation of tests and intersection computation. The convexity feature of the clipping polygon and the possibility of binary search usage over polygon vertices, because of known vertices order, have been used for principal speed up of the ECB line clipping algorithm [SKA93a] that resulted into new line clipping algorithm with complexity *O(lgN)*, see [SKA94a]. It has been expected that an algorithm for line clipping against convex polygon with complexity *O(lgN)* exists, see [CHA87a] and the algorithm for a line segment clipping with *O(lgN)* complexity was published in [RAP91a]. The known algorithms for clipping lines against a general convex polygon do not make tests similar to Cohen-Sutherland's clipping algorithm. The main reason seems to be the computational cost of such tests for convex polygons. If a clipping algorithm is to be effective, it is necessary to distinguish the cases where lines pass through a given polygon from those where lines do not intersect the polygon. Cyrus-Beck's (CB) algorithm solves this problem by direct computation of points of intersections, the ECB algorithm uses the separation theorem for Cyrus-Beck's algorithm to achieve a speed up of approx. 1.2 - 2.5 times. The ECB algorithm does not use the known order of vertices of the given clipping polygon and it has the complexity *O(N)*. The former *O(lgN)* algorithm [SKA94a] shows that a line can be clipped in *O(lgN)* steps, but the algorithm also takes *O(lgN)* steps to reject the line when it does not intersect the polygon. A new criterion can be used to eliminate the unnecessary computation for the cases when lines do not intersect the polygon and it leads to a faster *O(lgN)* algorithm.

2 Algorithm's principle

Let us suppose that we have a given convex clipping polygon anti-clockwise oriented and the line p is determined by two end-points

$$\mathbf{x}_A = [x_A, y_A]^T \quad , \quad \mathbf{x}_B = [x_B, y_B]^T$$

The convex polygon is represented by $N+1$ points

$$\mathbf{x}_i = [x_i, y_i]^T \quad , i = 0, \dots ,N$$

where: points $\mathbf{x}_0$ and $\mathbf{x}_N$ are identical (column notation is used), x_i and y_i are co-ordinates of the vertex $\mathbf{x}_i$.

The notation $\overline{\mathbf{x}_i\mathbf{x}_k}$ is used for a polyline from $\mathbf{x}_i$ to $\mathbf{x}_k$, i.e. it is a chain of line segments from $\mathbf{x}_i$ to $\mathbf{x}_k$.

Let us define the separation function $F(\mathbf{x})$ in the form

$$F(\mathbf{x}) = Ax + By + C$$

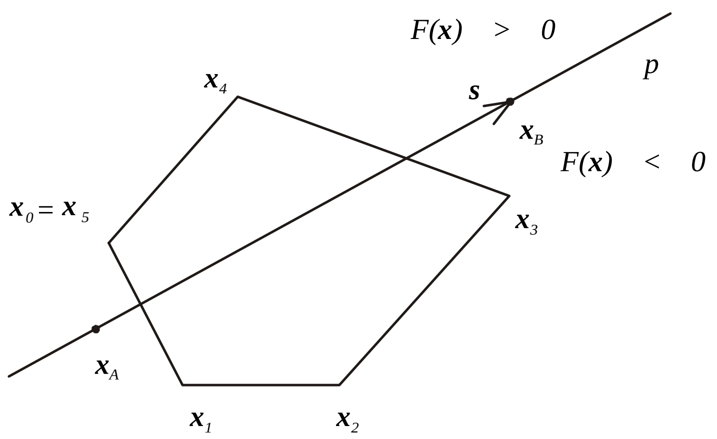
where $F(\mathbf{x}) = 0$ is an equation for the given line p and assume that the line has the orientation shown in Figure 1, $\mathbf{x}$ is defined as $\mathbf{x} = [x, y]^T$.

---

[1] {bui|skala} @kiv.zcu.cz
http://iason.zcu.cz/{~bui|~skala}

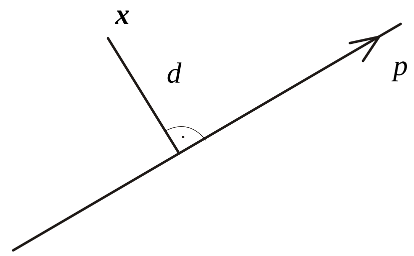
[2] Affiliated with the Multimedia Technology Research Centre, University of Bath, U.K.

[3] This work was supported by The Ministry of Education of the Czech Republic: project VS 97155 and project GA AV A2030801.



Separation function $F(\mathbf{x})$ of the given line p

Figure 1



Oriented distance d of the point $\mathbf{x}$ from the line p

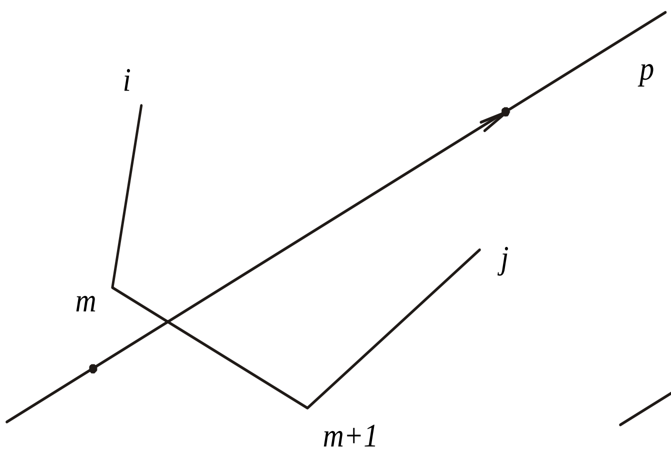
Figure 2

It can be seen in Figure 2, the oriented distance d of the point $\mathbf{x}$ from the line p can be determined as

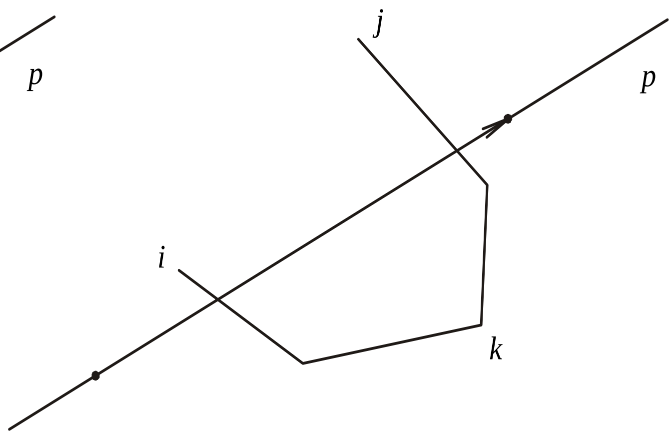
$$d = \frac{Ax + By + C}{\sqrt{A^2 + B^2}}$$

It means that the value of the function $F(\mathbf{x})$ is actually proportional to the distance d of the point $\mathbf{x}$ from the given line p. First of all, let us consider the chain $\overline{\mathbf{x}_i\mathbf{x}_j}$, where $0 \le i < j < N$. There are two following possible cases:

- In the first case, the points $\mathbf{x}_i$ and $\mathbf{x}_j$ are on the opposite sides of the line p, i.e. $F(\mathbf{x}_i) * F(\mathbf{x}_j) < 0$, there must be just one intersection point with the line p on the chains $\overline{\mathbf{x}_i\mathbf{x}_j}$ (because the given polygon is convex), i.e. there must exist an index m so that $F(\mathbf{x}_m) * F(\mathbf{x}_{m+1}) < 0 \quad i \le m < j$, see Figure 3. It is obvious that in this case the intersection point can be found in $O(lgM)$ steps using binary search over vertices, where M is a number of line segments in the chain $\overline{\mathbf{x}_i\mathbf{x}_j}$.
- Unfortunately, in the second case when the points $\mathbf{x}_i$ and $\mathbf{x}_j$ are on the same side of the line p, the situation is more complex to solve. Let us concentrate on the point $\mathbf{x}_k$, where $k = (i + j)$ ***div*** 2. The condition $F(\mathbf{x}_i) * F(\mathbf{x}_k) < 0$ shows that the point $\mathbf{x}_k$ is on one side of the line p, whereas $\mathbf{x}_i$ and $\mathbf{x}_j$ are on the opposite side. This also derives that there must be just one intersection point on the chains $\overline{\mathbf{x}_i\mathbf{x}_k}$ and $\overline{\mathbf{x}_k\mathbf{x}_j}$ for each chain, because the given polygon is convex. The intersection point on each chain can be again found in $O(lgM)$ steps using binary search over vertices, where M is a number of line segments in the given chain, see Figure 4. The worse case will happen when all of three points $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{x}_k$ lie on the same side of the line p. It is possible to distinguish all three fundamental sub-cases supposing the previously shown orientation of the separation function $F(\mathbf{x})$.



$\mathbf{x}_i$ and $\mathbf{x}_j$ are on the opposite sides of the line p



$\mathbf{x}_i$ and $\mathbf{x}_j$ are on the same side of the line p

Figure 3

Figure 4

a) The point $\mathbf{x}_i$ is the closest point to the line p, i.e. $F(\mathbf{x}_i) = \mathbf{min}\,\{F(\mathbf{x}_i), F(\mathbf{x}_j), F(\mathbf{x}_k)\}$. In this case, if $F(\mathbf{x}_{i+1}) < F(\mathbf{x}_i)$ then the chain $\overline{\mathbf{x}_i\mathbf{x}_k}$ can intersect the line p, see Figure 5. This condition actually expresses that we are getting closer to the line p, i.e. the oriented distance d is smaller, therefore, the chain $\overline{\mathbf{x}_k\mathbf{x}_j}$ can be removed by the assignment $j=k$. When this condition is not true, the whole chain $\overline{\mathbf{x}_i\mathbf{x}_j}$ is on one side of the line p, i.e. there is no intersection point on this chain and the chain is rejected, see Figure 6.

b) Similarly for the case when the point $\mathbf{x}_j$ is the closest point to the line p, i.e. $F(\mathbf{x}_j) = \mathbf{min}\,\{F(\mathbf{x}_i), F(\mathbf{x}_j), F(\mathbf{x}_k)\}$, see Figures 7-8, the intersection points can lie only the chain $\overline{\mathbf{x}_k\mathbf{x}_j}$ or do not exist at all. The condition $F(\mathbf{x}_{j-1}) < F(\mathbf{x}_j)$ decides that the chain $\overline{\mathbf{x}_i\mathbf{x}_k}$ can be removed and the index i must be changed to k, see Figure 7.

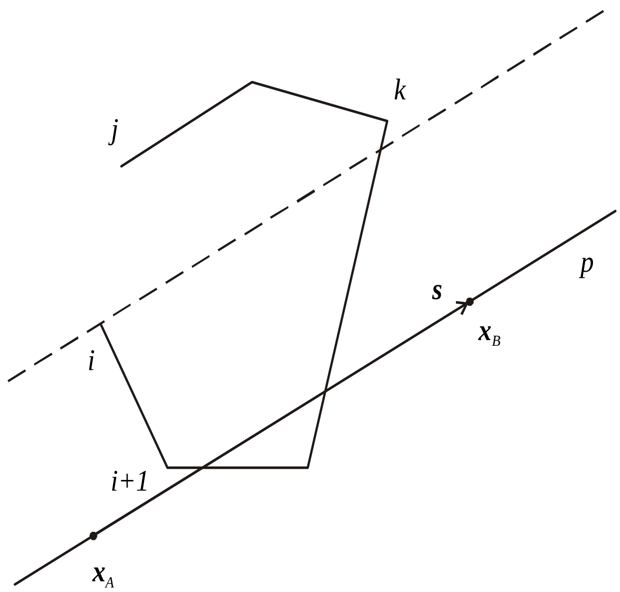
c) A little bit more complex situation is shown by Figure 9-10, where the point $\mathbf{x}_k$ is the closest point to the line p, i.e. $F(\mathbf{x}_k) = \mathbf{min}\,\{F(\mathbf{x}_i), F(\mathbf{x}_j), F(\mathbf{x}_k)\}$. In Figure 9 the chain $\overline{\mathbf{x}_k\mathbf{x}_j}$ can be removed, similarly in Figure 10 the chain $\overline{\mathbf{x}_i\mathbf{x}_k}$ can be removed, too. In the first, resp. second, case index j, resp. index i, must be changed to k. Theses cases can be distinguished by using criterion $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$. Actually we must distinguish whether we are getting closer to the given line p or not.

It is very easy to derive the similar conditions for those cases when the line p has an opposite orientation.

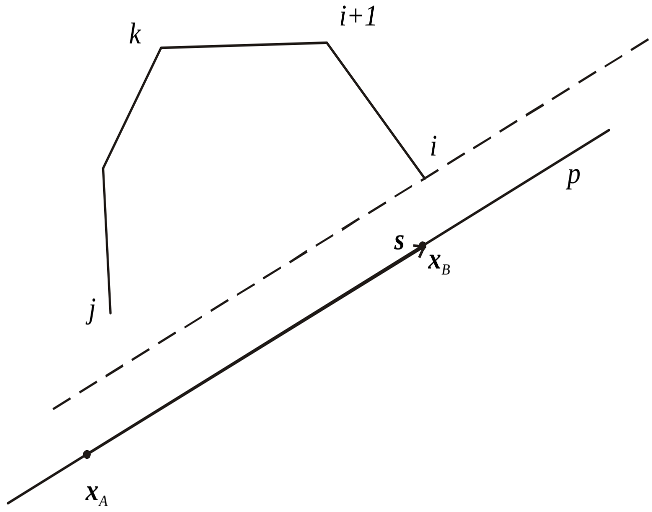
The above mentioned analysis leads to a new algorithm. The proposed algorithm contains the following basic steps:

- The algorithm starts with $i = 0;\ j = n - 1$
- If the points $\mathbf{x}_0$ and $\mathbf{x}_{n-1}$ are on the opposite sides of the line p, i.e. $F(\mathbf{x}_0) * F(\mathbf{x}_{n-1}) < 0$ then one intersection point is on the edge $\overline{\mathbf{x}_{n-1}\mathbf{x}_0}$ and the second one is on the chains $\overline{\mathbf{x}_0\mathbf{x}_{n-1}}$
- If the points $\mathbf{x}_0$ and $\mathbf{x}_{n-1}$ are on the same side of the line p, the algorithm continues with the process to subsequently shorten the chain $\overline{\mathbf{x}_i\mathbf{x}_j}$. This process is repeated until the whole chain is rejected or $F(\mathbf{x}_i) * F(\mathbf{x}_k) < 0$. If this condition becomes true we will obtain two chains $\overline{\mathbf{x}_i\mathbf{x}_k}$ and $\overline{\mathbf{x}_k\mathbf{x}_j}$, that intersects the line p and binary search over vertices can be used again as we get a similar situation shown in Figure 3.

Now it can be seen that all parts of the proposed algorithm are of complexity $O(lgM)$, where M is a number of edges in the given chain because we have used the binary search over vertices of the clipping convex polygon for all steps. Therefore the algorithm has $O(lgN)$ complexity and it is described by Algorithm 1.



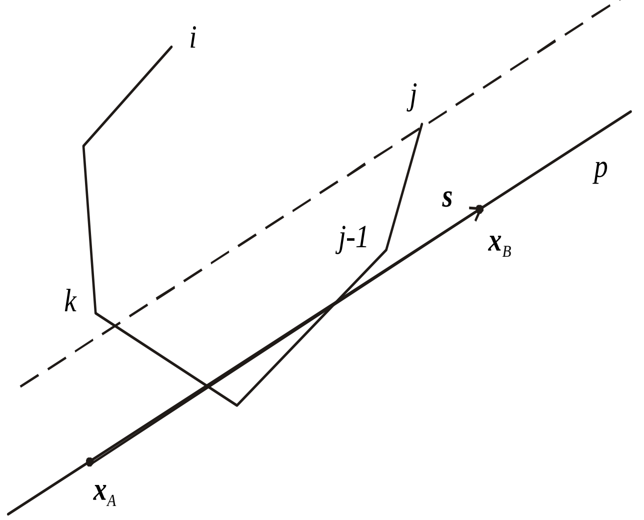
$F(\mathbf{x}_{i+1}) < F(\mathbf{x}_i) \Rightarrow \overline{\mathbf{x}_k\mathbf{x}_j}$ is removed



$F(\mathbf{x}_{i+1}) > F(\mathbf{x}_i) \Rightarrow \overline{\mathbf{x}_i\mathbf{x}_j}$ is rejected

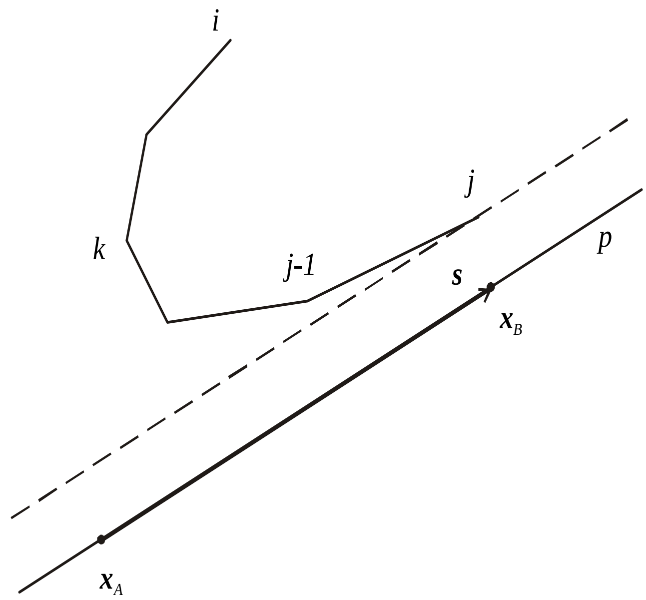
Figure 5

Figure 6



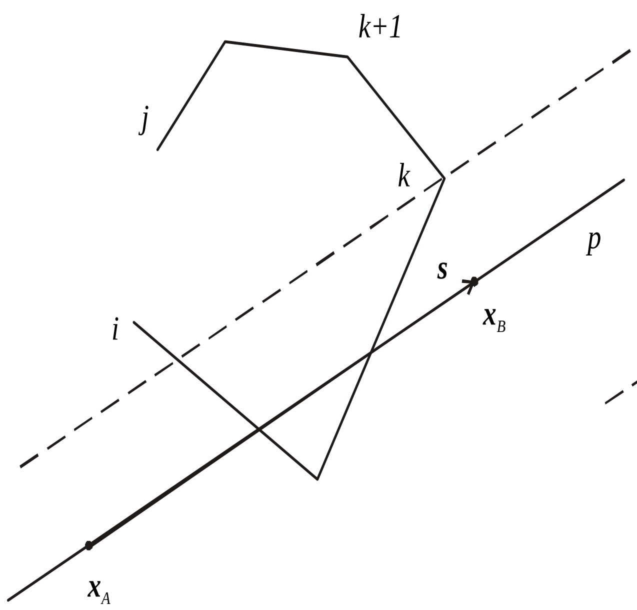
$F(\mathbf{x}_{j-1}) < F(\mathbf{x}_j) \Rightarrow \overline{\mathbf{x}_i\mathbf{x}_k}$ is removed

Figure 7



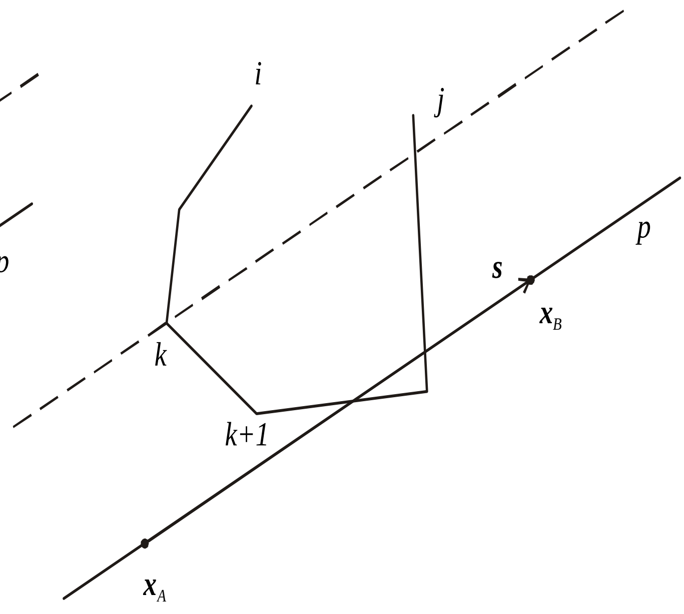
$F(\mathbf{x}_{j-1}) > F(\mathbf{x}_j) \Rightarrow \overline{\mathbf{x}_i\mathbf{x}_j}$ is rejected

Figure 8



$F(\mathbf{x}_{k+1}) > F(\mathbf{x}_k) \Rightarrow \overline{\mathbf{x}_k\mathbf{x}_j}$ is removed

Figure 9



$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k) \Rightarrow \overline{\mathbf{x}_i\mathbf{x}_k}$ is removed

Figure 10

```
procedure CLIP_2D_log (x_A, x_B);
{N+1 points x_i = [x_i, y_i]^T , i = 0, ... ,N represent the convex polygon}
{the line p or line segment is determined by two points x_A = [x_A, y_A]^T , x_B = [x_B, y_B]^T}

function macro F(x): real;                    { should be implemented as an in-line function }
begin
      F := A * x + B * y + C;
end { F };

function INTERSECTION (p, x_i , x_j ): real;  { should be implemented as an in-line function }
begin
      INTERSECTION := ((x_j – x_i) * (y_i – y_A) – (y_j – y_i) * (x_i – x_A)) / ((x_j – x_i) * (y_B – y_A) – (y_j – y_i) * (x_B – x_A));
end { INTERSECTION };

function SOLVE ( i , j, i_GT_0 ): real; { finds two nearest vertices on the opposite sides of the given line p }
{ i_GT_0 is a boolean parameter indicating whether F(x_i) > 0 }
begin if i_GT_0 then  while ( j - i ) ≥ 2 do { j ≥ i always }
                      begin  k := ( i + j ) div 2; { shift to the right }
                             if F(x_k) < 0 then j := k else i := k
                      end { while }
      else            while ( j - i ) ≥ 2 do { j ≥ i always }
                      begin  k := ( i + j ) div 2; { shift to the right }
                             if F(x_k) < 0 then i := k else j := k
                      end { while };
      { compute the value t of an intersection point of the line p with the polygon edge x_i x_j }
      SOLVE := INTERSECTION (p, x_i , x_j );
end { SOLVE };

begin { determine the A, B, C values for the separation function F(x) }
      A := y_A – y_B; B := x_B – x_A; C := x_A * y_B – x_B * y_A;
      i := 0;           j := N-1;
      { for lines             t_min := –∞; t_max := +∞;}
      { for line segments     t_min := 0; t_max := 1;}
      Fc := F(x_i) ; { proportional distance of the closer point }
      { for the polygon orientation shown in Figure 1 }
      if Fc > 0 then
      begin { for the orientation of line p shown in Figure 3 }
            if F(x_N-1) < 0 then
            begin { see Figure 3 }
                  t_1 := SOLVE ( 0 , N-1, TRUE ) ; { find an intersection on x_0 x_n-1 chain }
                  t_2 := INTERSECTION (p, x_N-1 , x_0); {find an intersection on x_n-1 x_0 edge}
                  { for the line segment clipping include the next 5 lines}
                  { if t_1 > t_2 then begin t := t_2 ; t_2 := t_1 ; t_1 := t end;}
                  {compute <t_1, t_2> as <t_1, t_2> ∩ <0, 1> }
                  { t_1 := max (t_min , t_1 );      t_2 := min (t_max , t_2 );   }
                  { if <t_1, t_2> ≠ ∅ then draw line segment                     }
                  { if t_1 ≤ t_2 then SHOW-LINE(x(t_1) , x(t_2));               }
            EXIT { exit procedure CLIP_2D_log }
            end { if };
```

```
          if F(x_{N-1}) < Fc  then
          begin    Fc := F(x_{N-1});
                   i_closer_j := FALSE       { vertex x_j  is closer than vertex x_i }
          end else i_closer_j := TRUE;       { vertex x_i  is closer than vertex x_j }
          while ( j - i ) ≥ 2 do
          begin    k := ( i + j ) div 2; { shift to the right }
                   if F(x_k) < 0   then
                   begin   { see Figure 4}
                           t1 := SOLVE ( i , k, TRUE ) ; { find an intersection on x_i x_k chain }
                           t2 := SOLVE ( k , j, FALSE ); {find an intersection on x_k x_j chain }
                           { for the line segment clipping include the next 5 lines}
                           { if t1 > t2 then begin t := t2 ; t2 := t1 ; t1 := t end;}
                           {compute <t1, t2> as <t1, t2> ∩<0, 1> }
                           { t1 := max (t_min , t1 );      t2 := min (t_max , t2 );    }
                           { if <t1, t2> ≠ ∅ then draw line segment            }
                           { if t1 ≤ t2 then  SHOW-LINE(x(t1) , x(t2));        }
                           EXIT { exit procedure CLIP_2D_log }
                   end { if };
                   if F(x_k) > Fc  then { Figures 5-8}
                   begin { DELETE CHAIN ( i , j ) removes the chain x_i x_j }
                           if i_closer_j then { Figures 5-6}
                                   if F(x_{i+1}) < F(x_i) then
                                           j := k { DELETE CHAIN ( k , j ); {Figure 5}
                                   else EXIT { exit procedure CLIP_2D_log }{Figure 6}
                           else    if F(x_{j-1}) < F(x_j) then
                                           i := k { DELETE CHAIN ( i , k ); {Figure 7}
                                   else EXIT { exit procedure CLIP_2D_log}  {Figure 8}
                   end
                   else {Figures 9-10}
                   begin   if F(x_{k+1}) > F(x_k) then
                           begin   j := k; { DELETE CHAIN ( k , j ); }          {Figure 9}
                                   i_closer_j := FALSE       { vertex x_j  is closer than vertex x_i }
                           end else
                           begin   i := k; { DELETE CHAIN ( i , k );}           {Figure 10}
                                   i_closer_j := TRUE        { vertex x_i  is closer than vertex x_j }
                           end;
                           Fc := F(x_k) ;
                   end
          end { while }
     end else
     begin    { for an opposite orientation of the line situations are solved similarly } end
end { CLIP_2D_log }
```

Algorithm 1: *O(lgN)* algorithm

3 Experimental results

The new proposed *O(lgN)* algorithm was verified experimentally on Pentium Pro, 200MHz, 128MB RAM, 512KB CACHE. The proposed algorithm has been tested against the Cyrus-Beck's (**CB**) and the former *O(lgN)* (**FL**) algorithms on data sets of line segments (10^5) with end-points that have been randomly and uniformly generated inside a circle in order to eliminate an influence of rotation. Convex polygons were generated as *N*-sided convex polygons inscribed into a smaller circle.

To compare these algorithms, let us introduce coefficients of the effectivity ν as

$$\nu_1 = \frac{T_{CB}}{T} \quad , \quad \nu_2 = \frac{T_{FL}}{T}$$

where: T_{CB} , T_{FL} , T are execution times needed by the **CB**, **FL** and the proposed *O(lgN)* algorithms.

Description of the **CB** and **FL** algorithms can be found in [SKA94a] together with their theoretical and experimental comparisons. The Table 1 and Table 2 present the obtained results. In these tables, the first row shows the number of polygon edges and the first column the percentage ***q*** of intersecting lines.

It can be seen that, see Table 1, that the proposed algorithm is significantly faster then **CB** algorithm, specially for the high *N*. This is expectable because the proposed algorithm runs with *O(lgN)* complexity, whereas the complexity of **CB** algorithm is *O(N)*.

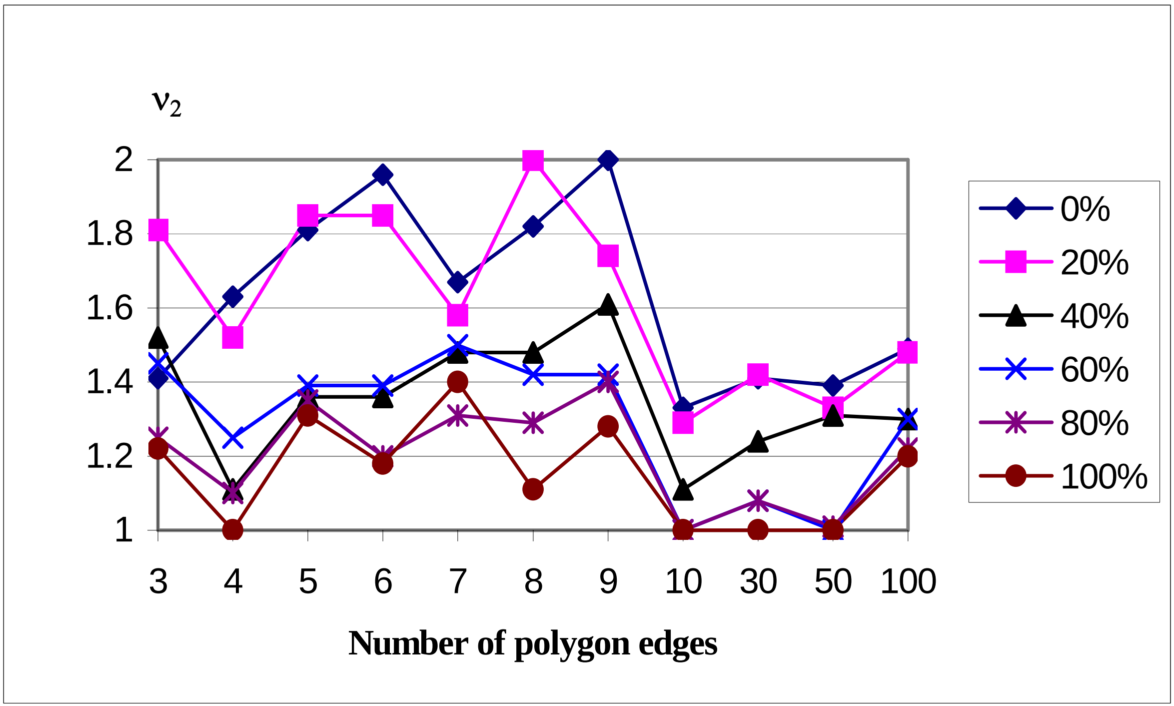
Table 2 shows that the proposed *O(lgN)* algorithm relatively improves the former one significantly, especially for the cases when the given line does not intersect the clipping polygon.

ν_1	*N*										
q	3	4	5	6	7	8	9	10	30	50	100
0%	2.85	3.85	4.67	5.68	5.48	6.15	7.00	8.33	20.85	30.84	36.01
10%	2.93	3.18	3.97	4.82	5.48	5.37	6.97	8.33	18.48	27.67	33.42
20%	3.04	3.18	4.00	4.85	4.76	6.33	6.08	7.24	19.02	27.57	33.83
30%	2.33	3.15	4.00	4.18	4.76	4.61	5.23	7.03	16.26	24.58	31.51
40%	2.33	2.39	3.00	3.50	4.11	4.64	5.25	6.23	16.59	24.56	29.76
50%	2.33	2.36	3.47	3.61	4.14	4.08	4.60	5.61	14.87	24.67	31.45
60%	2.16	2.39	3.00	3.50	4.11	4.16	4.60	5.00	13.33	20.47	29.71
70%	1.97	2.36	2.69	3.08	3.62	4.16	4.82	5.00	12.30	20.47	27.96
80%	1.75	2.08	2.69	2.80	3.29	3.69	4.18	5.00	12.32	19.03	27.96
90%	1.91	1.91	2.40	2.57	3.02	3.80	3.83	4.98	11.28	18.76	26.56
100%	1.54	1.73	2.40	2.57	3.29	3.80	3.85	5.00	10.56	17.48	25.12

Table 1: Comparison between the **CB** algorithm and the proposed algorithm

ν_2	*N*										
q	3	4	5	6	7	8	9	10	30	50	100
0%	1.41	1.63	1.81	1.96	1.67	1.82	2.00	1.33	1.41	1.39	1.49
10%	1.57	1.48	1.67	1.67	1.85	1.58	2.00	1.30	1.25	1.35	1.40
20%	1.81	1.52	1.85	1.85	1.58	2.00	1.74	1.29	1.42	1.33	1.48
30%	1.48	1.48	1.82	1.58	1.74	1.48	1.61	1.26	1.22	1.31	1.31
40%	1.52	1.11	1.36	1.36	1.48	1.48	1.61	1.11	1.24	1.31	1.30
50%	1.67	1.25	1.74	1.50	1.50	1.42	1.42	1.12	1.11	1.29	1.44
60%	1.45	1.25	1.39	1.39	1.50	1.42	1.42	1.00	1.08	1.00	1.30
70%	1.54	1.36	1.35	1.32	1.42	1.44	1.45	1.00	1.00	1.00	1.27
80%	1.25	1.10	1.35	1.20	1.31	1.29	1.40	1.00	1.08	1.01	1.22
90%	1.40	1.09	1.29	1.20	1.20	1.11	1.28	1.00	1.00	1.07	1.22
100%	1.22	1.00	1.31	1.18	1.40	1.11	1.28	1.00	1.00	1.00	1.20

Table 2: Comparison between the **FL** algorithm and the proposed algorithm



Graph 1: Comparison between the **FL** algorithm and the proposed algorithm

4 Conclusion

The new efficient algorithm of *O(lgN)* complexity for clipping lines against convex polygon in E^2 has been developed, verified and tested. Edges of the given convex polygon can be arbitrarily oriented. The algorithm also proved the applicability of computational geometry principles even for small *N* although it deals mostly with the cases for large *N*. The proposed algorithm can be easily modified for polygon clipping by convex polygon with complexity *O(M*lgN)*. The superiority of the suggested algorithm over the **CB** and **FL** algorithms was proved experimentally.

Some related reports are available in the on-line form at the URL:

http://herakles.zcu.cz/publication.htm

5 Acknowledgements

The authors would like to express their thanks to all who contributed to this work, especially to recent MSc. and PhD. students at the University of West Bohemia in Plzen, who stimulated this work, and colleagues for critical comments that improved the algorithm a lot.

6 References

[CHA87a] Chazelle,B., Dobkin,D.P.: Intersection of Convex Objects in Two and Three Dimensions, JACM, Vol.34, No.1., pp.1-27, 1987.

[CYR79a] Cyrus,M.,Beck,J.: Generalized Two and Three Dimensional Clipping, Computers&Graphics, Vol.3, No.1, pp.23-28, 1978.

[FOL90a] Foley,D.J., van Dam,A., Feiner,S.K., Hughes,J,F.: Computer Graphics - Principles and Practice, Addison Wesley, 2nd ed., 1990.

[LIA83a] Liang,Y.D., Barsky,B.A.: An Analysis and Algorithms for Polygon Clipping, CACM, Vol.26, No.11, pp.868-876, 1983.

[LIA84a] 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.

[RAP91a] Rappaport,A.: An Efficient Algorithm for Line and Polygon Clipping, The Visual Computer, Vol.7, No.1, pp.19-28, 1991.

[SKA89a] Skala,V.: Algorithms for 2D Line Clipping, in CGI'89 Conference Proceedings, pp.121-128, 1989.

[SKA89b] Skala,V.: Algorithms for 2D Line Clipping, in EG'89 Conference Proceedings, pp.355-367, 1989.

[SKA93a] Skala,V.: An Efficient Algorithm for Line Clipping by Convex Polygon, Computers & Graphics, Vol.17, No.4, pp.417-421, 1993.

[SKA94a] Skala,V.: O(lg N) Line Clipping Algorithm in E^2, Computers & Graphics, Vol.18, No.4, Pergamon Press, pp.517-524, 1994.