*Comput. & Graphics* Vol. 17, No. 4, pp. 417–421, 1993
Printed in Great Britain.

0097-8493/93 $6.00 + .00


Technical Section

# AN EFFICIENT ALGORITHM FOR LINE CLIPPING BY CONVEX POLYGON

VÁCLAV SKALA
Department of Informatics and Computer Science, University of West Bohemia, Americká 42, Box 314, 306 14 Plzeň, Czech Republic

**Abstract**—A new line clipping algorithm against convex window based on a new approach for intersection detection is presented. Theoretical comparisons with Cyrus-Beck's algorithm are shown together with experimental results obtained by simulations. The main advantage of the presented algorithm is the substantial acceleration of the line clipping problem solution and that edges can be oriented clockwise or anti-clockwise.

## 1. INTRODUCTION

Many algorithms for clipping lines against rectangular, convex or non-convex windows have been published with many modifications derived from well-known Cohen-Sutherland, Liang-Barsky and Cyrus-Beck's algorithms[1–37]. Nevertheless, new developments or speed-up can be seen within every new paper.

The proposed algorithm is based on a simple rule: *Make tests first and then compute.*

A new criterion for testing whether the line intersects the given polygon or not is given, too.

The Cohen-Sutherland's algorithm (C-S) for clipping lines against a rectangular window is very well-known, not only because of its frequent usage, but for a very clever test whether the end-points are inside of the given rectangular area. The situation is a little bit more complicated when edges of the window are not collinear with axes.

### 1.1. *Algorithm* 1

```
procedure Clip 2D Cyrus Beck ( xA , xB );
begin { ! ! all vectors ni precomputed ! ! algorithm shortened }
  i := 1;  tmin := −∞;  tmax := ∞;  s := xB − xA;
  { for line segment  tmin := 0.0;  tmax := 1.0 }
  while i <= N do      { N is a number of edges }
  begin   { ni is a normal vector, e.g. ni = [ sy,−sx]^T }
          { and ni must point out of the convex window }
    ξ := s^T ni;  si := xi − xA;
    if ξ < > 0.0 then
    begin t := si^T ni / ξ;
          if ξ > 0.0 then tmax := min ( t , tmax )
                     else tmin := max ( t , tmin )
    end
    else Special case solution;
    i := i + 1
  end;
  if tmin > tmax then EXIT; { ! ! ⟨tmin,tmax⟩ = ∅ }
  { recompute end-points if changed }
  if tmax < 1.0 then xB := xA + s tmax;
  if tmin > 0.0 then xA := xA + s tmin;
  SHOW LINE( xA , xB );
end;
```

The known algorithms for clipping lines against general convex window do not make tests similar to C-S algorithm. The main reason seems to be the computational cost of such a test for convex window. If clipping algorithm is to be effective it is necessary to distinguish cases where lines pass through a given window from those where lines do not intersect the window. Cyrus-Beck's (C-B) algorithm solves this problem by direct computation of points of intersections, see Algorithm 1.

## 2. PROPOSED ALGORITHM

It is obvious that the line $p$ intersects the edge $\mathbf{x}_0\mathbf{x}_1$ of the given polygon *if and only if* the vector $\mathbf{s}$ lies between two vectors $\mathbf{s}_0$ and $\mathbf{s}_1$, see Fig. 1.

Let us define $\xi$ and $\eta$ as the $z$-coordinate of the cross products as follows

$$\xi = [\mathbf{s} \times \mathbf{s}_i]_z \quad \eta = [\mathbf{s} \times \mathbf{s}_{i+1}]_z \quad i = 0, \ldots, N-1$$

where $N$ is a number of edges.

It is possible to show that the line $p$ given by the end-point $\mathbf{x}_A$ and by the vector $\mathbf{s}$ intersects the edge $\mathbf{x}_i\mathbf{x}_{i+1}$ *if and only if* the expressions

$$(\xi > 0) xor (\eta < 0), \quad \text{resp.} \quad \xi * \eta < 0$$

have value *true*, see Table 1 and Fig. 2. The above mentioned expressions are *independent* on polygon *orientation*. It means that edges can be clockwise or anti-clockwise oriented.

Now, it is possible to specify a new algorithm that is based on the presumption that a line can intersect a convex polygon only in two points, see Algorithm 2, if special cases are not considered, see Fig. 2. Special cases should be handled carefully, see [38].

In comparison with Algorithm 1 the proposed algorithm *only detects* whether the given line intersects the given polygon and then computes intersection points, while C-B algorithm does *compute all* possible intersection points with all edges.

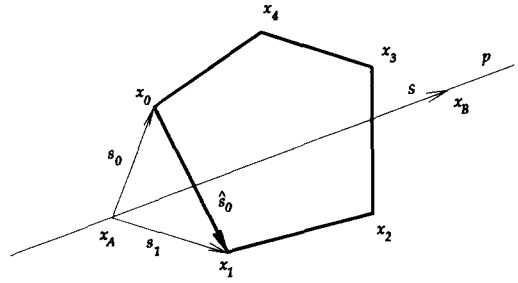


Fig. 1. Line segment clipping by convex polygon.

## 2.1. *Algorithm* 2

```
procedure Clip 2D ( x_A , x_B );
begin { !!!! all vectors ŝ_i are precomputed !!!! }
  k := 0; i := N − 1; j := 0;
  s := x_B − x_A;   ξ := [ s x s_i ]_z;
  while (j < N) and (k < 2) do
  begin η := [ s x s_j ]_z;
    if (ξ * η) < 0.0 then { intersection exists }
    begin
      index_k := i;   { save edge index having intersection }
      k := k + 1;   ξ := η;
    end
    else if ( ξ * η ) = 0.0 then Special cases { ξ=0 or η=0 }
                                 { cases i − m }
                            else Intersection does not exist;
                                 { cases e − h }
    i := j;   j := j + 1
  end;
  if k = 0 then { intersection does not exist } EXIT;
  { k = 2 intersections exist − edges saved in index_k }
  t_min := −∞; t_max := ∞; {for line segments t_min := 0; t_max := 1;}
  i := index_1;   t_1 := det[ x_i − x_A ¦ − ŝ_i ] / det[ s ¦ −ŝ_i ];
  { for effective implementation use identity x_i − x_A = s_i }
  i := index_2;   t_2 := det[ x_i − x_A ¦ −ŝ_i ] / det[ s ¦ −ŝ_i ];
  { recompute end-points if changed }
  if t_1 < t_2 then { swap t_1 <−> t_2 values ? }
  begin if t_2 < t_max then x_B := x_A + s t_2;
  if t_1 > t_min then x_A := x_A + s t_1
  end { t_2 < t_max; t_1 < t_min can be left out for lines }
  else begin if t_1 > t_min then x_B := x_A + s t_1;
  if t_2 < t_max then x_A := x_A + s t_2
  end;
  SHOW LINE( x_A , x_B );
  end
```

### 3. THEORETICAL ANALYSIS

Before making any comparisons it is necessary to point out that time needed for each operation (:=, <, ±, *, /) does differ from computer to computer, see Table 2.

Let us denote $N$ number of edges of the given convex polygon. Then it is possible to express the complexity of the C-B algorithm as (the worst case)

$$(8, 3, 6, 4, 0) + (5, 3, 7, 4, 1) * N$$

and time of computation, according to Table 2 (PC 486 used), can be estimated as

$$T_{C\text{-}B} = 590 + 621 * N$$

while for proposed algorithm the complexity is given as

$$(15, 3, 11, 14, 2) + (3, 1, 3, 3, 0) * N$$

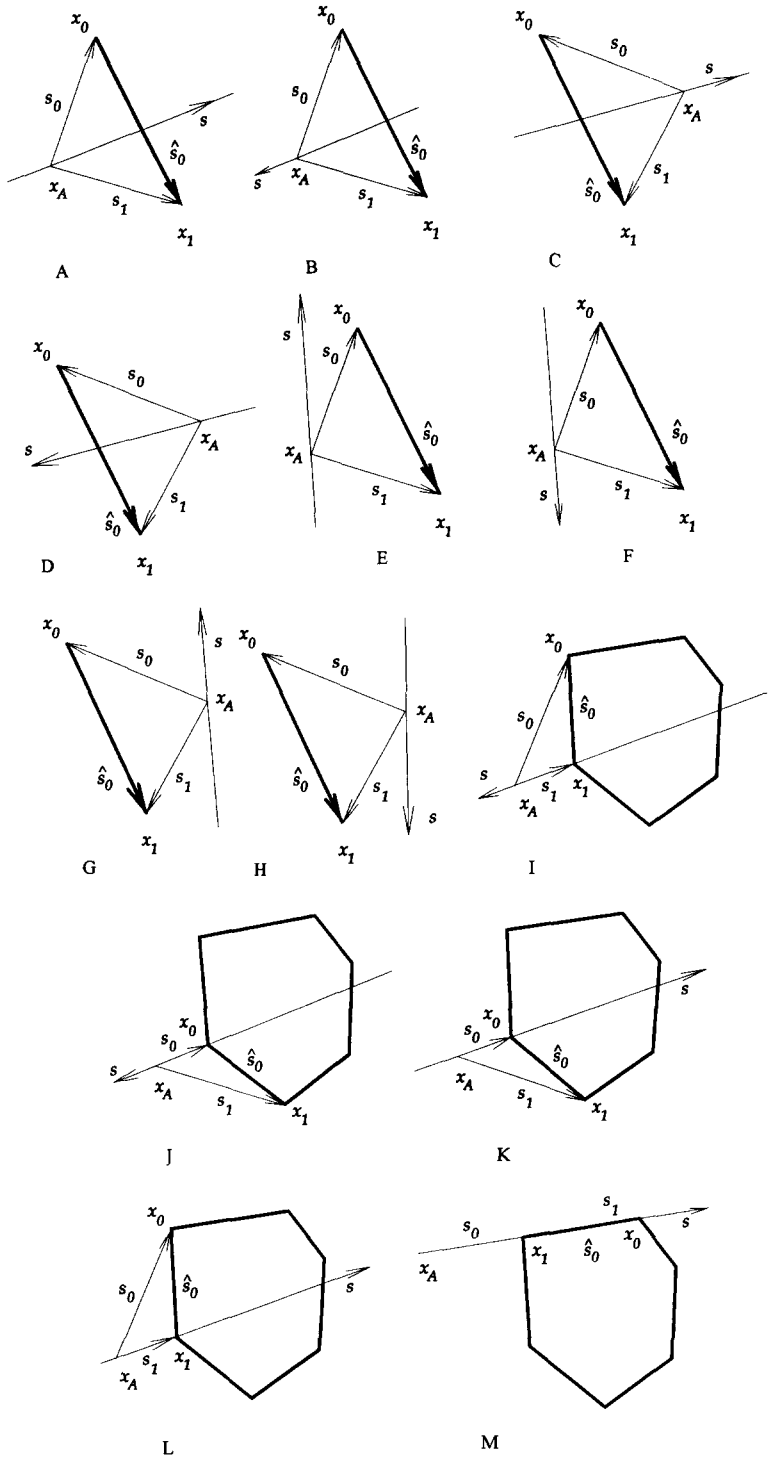


Fig. 2. Possible cases for line segment clipping.

and time of computation $T$ can be estimated

$$T = 1329 + 257 * N$$

It means that the new algorithm should be better than C-B algorithm for $N \geq 2.1$. Let us consider a coefficient $\nu$ as a speed up coefficient, which is defined as

$$\nu = \frac{T_{\text{C-B}}}{T}$$

then some theoretical results of speed up are given in Table 3.

From the theoretical point of view the proposed algorithm should be for $N = 4$, approximately 30% faster than C-S algorithm. It can be seen that the efficiency increases dramatically with $N$, especially for $4 \leq N \leq 20$. For $N > 20$ the algorithm needs *only half* of the time needed by C-B algorithm. For $N > 50$ the factor $\nu$ is nearly constant. Table 3 shows expected results for

Table 1. Possible cases.

| $\xi$ | $\nu$ | Does the line intersect edge? |
|---|---|---|
| >0 | >0 | no—cases *g, h* |
| >0 | =0 | yes—special case *l* |
| >0 | <0 | yes—cases *a, c* |
| =0 | >0 | yes—special case *j* |
| =0 | =0 | no—special case *m* |
| =0 | <0 | yes—special case *k* |
| <0 | >0 | yes—cases *b, d* |
| <0 | =0 | yes—special case *i* |
| <0 | <0 | no—cases *e, f* |

Table 2. Times for basic arithmetic operations.

| | PC 386DX/387 25 MHz | | | | |
|---|---|---|---|---|---|
| | := | < | ± | * | / |
| Int | 14 | 42 | 10 | 40 | 58 |
| Float | 204 | 260 | 80 | 82 | 154 |

| | PC 486 33 MHz 64KB cache—selected for evaluation | | | | |
|---|---|---|---|---|---|
| | := | < | ± | * | / |
| Int | 5 | 9 | 3 | 26 | 44 |
| Float | 33 | 50 | 16 | 20 | 114 |

Time is in 1/10 sec. for 5 000 000 operations.

Table 3. Theoretical results.

| | Expected results for PC 386DX/387 | | | | | |
|---|---|---|---|---|---|---|
| | Intersections exist | | | Intersections do not exist | | |
| $N$ | 4 | 10 | $\Rightarrow \infty$ | 4 | 10 | $\Rightarrow \infty$ |
| $\nu$ | 1.26 | 1.60 | 2.09 | 1.75 | 1.87 | 1.97 |

| | Expected results for PC 486 | | | | | |
|---|---|---|---|---|---|---|
| | Intersections exist | | | Intersections do not exist | | |
| $N$ | 4 | 10 | $\Rightarrow \infty$ | 4 | 10 | $\Rightarrow \infty$ |
| $\nu$ | 1.30 | 1.74 | 2.42 | 2.01 | 2.16 | 2.29 |

Table 4. Experimental results.

| Experimental results for PC 386DX/387 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $N$ | 5 | 8 | 10 | 15 | 20 | 30 | 70 | 190 |
| $\nu$ | 1.41 | 1.54 | 1.73 | 1.81 | 1.90 | 2.08 | 2.27 | 2.41 |

| Experimental results for PC 486 33 MHz | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $N$ | 5 | 8 | 10 | 15 | 20 | 30 | 70 | 190 |
| $\nu$ | 1.37 | 1.59 | 1.71 | 1.88 | 2.25 | 3.32 | 2.58 | 2.85 |

PC 386/387 DX, too. It can be seen that the efficiency of the proposed method should be higher for faster machines. It is necessary to point out that many simplifications have been made for theoretical analysis because time of operations for integers and time needed for instructions was not considered.

## 4. EXPERIMENTAL RESULTS

The C-B and proposed algorithms have been tested on data sets of end-points that have been randomly and uniformly generated over a space inside of a circle in order to eliminate an influence of rotation. Convex polygons were generated as regular $N$-sided convex polygons inscribed into a smaller circle. The results in Table 4 have been obtained if 80% of the lines intersect the window.

The coefficients $\nu$ are in a good correlation with theoretical results because the pessimistic estimations have been taken for the theoretical analysis. It is necessary to point out that different tests might be used instead of shown cross product, *e.g.*, test for detection on which side of the given line $p$ the point $x_i$ lies.

## 5. CONCLUSION

The new efficient algorithm for clipping lines against convex window has been developed. Edges of the given polygon can be arbitrarily oriented. The theoretical analysis showed the advances over well-known Cyrus-Beck's algorithm. Experimental results support the theoretical analysis in spite of the fact that many factors have been omitted, like computational cost with integers, *etc.*

All tests were implemented in C++ on a PC 386/387 25 MHz and PC 486 33 MHz. It can be expected that for workstations the efficiency $\nu$ will be higher than for PC 486.

There is a hope that the proposed algorithm can be modified for clipping lines against non-convex windows, similarly to[31].

*Acknowledgements*—The author would like to express his thanks to students of Computer Graphics and CAD Systems who stimulated this work, especially Mr. P. Blaha for careful tests implementation of the proposed algorithm.

## REFERENCES

1. K. Akeley and C. P. Korobkin, Efficient Graphics Processor for Clipping Polygons, US Patent No. 5 051 737 (1991).
2. R. D. Andreev, Algorithm for clipping arbitrary polygons. *Comp. Graph. Forum* **7**, 183–192 (1988).
3. A. Arokiasamy, Homogeneous coordinates and the principle of duality in two dimensional clipping. *Comp. & Graph.* **13**, 99–100 (1989).
4. J. F. Blinn and M. E. Newell, Clipping using homogeneous coordinates. *Comp. Graph. (SIGGRAPH'78)* **12**, 245–251 (1978).
5. J. F. Blinn, A trip down to graphics pipeline—Line clipping. *IEEE Comp. Graph. Appl.* **11**, 98–105 (1991).

6. E. A. Brewer and B. A. Barsky, Clipping after projection: An improved perspective pipeline. Manuscript submitted for publication.
7. A. Burkert and S. Noll, Fast algorithm for polygon clipping with 3D windows. *Eurographics'88 Proceedings*, 405–419 (1988).
8. F. Cheng and Y. Yen, A parallel line clipping algorithm and its implementation. *CGI'89 Conference Proceedings* (1989).
9. M. Cyrus and J. Beck, Generalized two and three dimensional clipping. *Comp. & Graph.* **3**, 23–28 (1979).
10. J. D. Day, A comparisons of line clipping algorithms. Queensland University of Technology, School of Computing Science, Report 2-91 (1991).
11. J. D. Day, A new two dimensional line clipping algorithms for small windows. Queensland University of Technology, School of Computing Science, Report 3-91 (1991).
12. J. D. Day, A new two dimensional line clipping algorithms for small windows. *Comp. Graph. Forum* **11**, 241–245 (1992).
13. M. Dorr, A new approach to parametric line clipping. *Comp. & Graph.* **14**, 449–464 (1990).
14. V. J. Duvanenko, W. E. Robins, and R. S. Gyurcsik, Improving line segment clipping. *Dr. Dobb's J. Software Tools* **15**, 36, 38, 40, 42, 44–45, 98, 100 (1990).
15. D. Y. Fong and J. Chu, A string pattern recognition approach to polygon clipping. *Pattern Recognition* **23**, 879–892 (1992).
16. I. Herman and J. Reviczky, Some remarks on the modelling clip problem. *Comp. Graph. Forum* **7**, 265–272 (1988).
17. S. Kaijian, J. A. Edwards, and D. C. Cooper, An efficient line clipping algorithm. *Comp. & Graph.* **14**, 297–301 (1990).
18. A. C. Kilgour, Unifying vector and polygon algorithm for scan conversion and clipping. TR CSC/87/R7, University of Glasgow (May 1987).
19. G. Krammer, A line clipping algorithm and its analysis. *Comp. Graph. Forum* (*EG'92 Conference Proceedings*) **11**, C253–266 (1992).
20. Y. D. Liang and B. A. Barsky, A new concept and method for line clipping. *ACM Trans. Graph.* **3**, 1–22 (1984).
21. Y. D. Liang and B. A. Barsky, An analysis and algorithms for polygon clipping. *CACM* **26**, 868–876 (1984).
22. Y. Liang and B. A. Barsky, The optimal tree algorithms for line clipping. Technical paper distributed at Eurographics'92 Conference, Cambridge (1992).
23. P. G. Maillot, A new, fast method for 2D polygon clipping: Analysis and software implementation. *ACM Trans. Graph.* **11**, 276–290 (1992).
24. T. M. Nicholl, D. T. Lee, and R. A. Nicholl, An efficient new algorithm for 2D line clipping: Its development and analysis. *ACM Comp. Graph.* **21**, 253–262 (1987).
25. H. P. Nielsen, An intersection test using dual figures. *CFG* (in press).
26. H. P. Nielsen, Line clipping using semi-homogeneous coordinates. *CFG* (in press).
27. R. M. O'Bara and S. Abi-Ezzi, An analysis of modeling clip. In *EG'89 Conference Proceedings*, 367–380 (1989).
28. A. Rappaport, An efficient algorithm for line and polygon clipping. *Visual Comp.* **7**, 19–28 (1991).
29. N. C. Sharma and S. Manohar, Line clipping revisited: Two efficient algorithms based on simple geometric observations. *Comp. & Graph.* **16**, 51–54 (1992).
30. V. Skala, Algorithms for 2D line clipping. In *CGI'89 Conference Proceedings*, 121–128 (1989).
31. V. Skala, Algorithms for 2D line clipping. In *EG'89 Conference Proceedings*, 355–367 (1989).
32. V. Skala, Algorithm for line clipping in E2 for convex window (in Czech). *Algorithms '93 Conference Proceedings* (in press).
33. M. Slater and A. B. Barsky, 2D line and polygon clipping based on space subdivision. Manuscript submitted for publication.
34. M. S. Sobkow, P. Pospil, and Y.-H. Yang, A fast two-dimensional line clipping algorithm via line encoding. *Comp. & Graph.* **11**, 459–467 (1987).
35. I. E. Sutherland and G. W. Hodgman, Reentrant polygon clipping, *CACM* **17**, 32–42 (1974).
36. D.-N. Ying, A new algorithm for polygon clipping and Boolean operations, University of Zhejiang, Hangzhou, China (1991).
37. L. Yong-Kui, A new algorithm for line clipping by convex polygon. Manuscript submitted for publication.
38. H. Edelsbrunner and E. P. Muche, Simulation of simplicity: A technique to cope with degeneration cases in geometric algorithms. *ACM Trans. Graph.* **9**, 66–104 (1990).