ALGORITHMS COMPLEXITY AND LINE CLIPPING PROBLEM SOLUTIONS

Vaclav Skala[1]

Department of Informatics and Computer Science
University of West Bohemia, Univerzitní 22, Box 314
306 14 Plzen
Czech Republic
Skala@kiv.zcu.cz http://herakles.zcu.cz
Affiliated with: Multimedia Technology Research Centre, University of Bath, BATH, U.K.

ABSTRACT

Algorithm complexity is very often used for comparison of different algorithms in order to assess theirs properties. Nevertheless there are some other factors like actual speed for the expected applications, memory needed and others that might influence the final behaviour of the proposed algorithm. The aim of this contribution is to demonstrate some thoughts and connection between algorithm complexity, speed and discuss the influence of possible pre-processing to the final algorithm complexity. It will be also shown how some precise formulations could lead to better and faster algorithms to decrease algorithm complexity. The influence of pre-processing will be analysed according to the algorithm complexity change.

Keywords: algorithm complexity, computer graphics, line clipping, optimal algorithm, geometric algorithms.

INTRODUCTION

The fundamental problem in algorithm design is to use all known data properties as much as possible in order to get an algorithm with better efficiency. Sometimes it is quite difficult as some features or dependencies are not known in their explicit forms. If those factors are used in algorithm design it is possible not only to improve algorithms property but sometimes also to change the algorithms complexity. Line clipping algorithms have been studied heavily in the past because they are the bottleneck of all fundamental graphics packages and pipelines. There have been several attempts more or less successful to develop faster algorithms almost with a complexity $O(N)$. It can be shown that the optimal algorithm (without pre-processing) is of $O(lg\,N)$ complexity for the E^2 case. Some new algorithms and modifications have been developed, see [Skala94c].

[1] This work was supported by The Ministry of Education of the Czech Republic - projects PG 97193, PG 97185 and VS 97155

Clipping problem solutions in E^2, resp. in E^3 can be classified as clipping by

- orthogonal window,
- convex polygon, resp. polyhedron,
- non-convex polygon, resp. polyhedron

and as

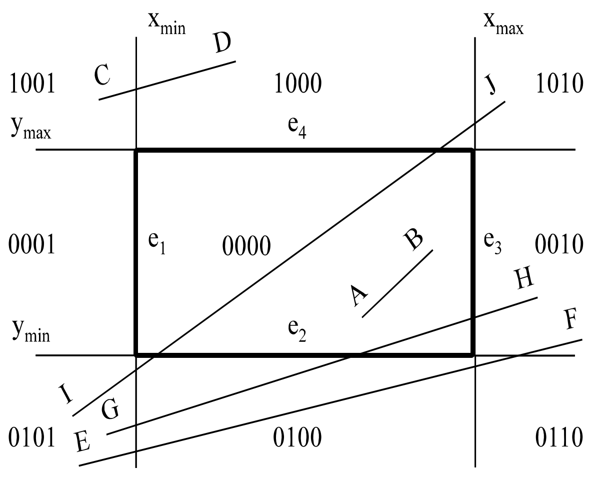
- line segment clipping
- line clipping

Clipping problem solution seems to be quite a simple problem as its formulation is to find a part of the given line or line segment that is inside of the given polygon or polyhedron.

CLIPPING BY A RECTANGULAR WINDOW

Very famous and popular algorithm for clipping by rectangular window in E^2 is the Cohen-Sutherland (CS) algorithm for a line segment clipping and Liang-Barsky (LB) algorithm for a line clipping.

The CS algorithm rely on the coding scheme of the end-points that enables to detect some trivial cases, like line segment is totally outside or inside of the clipping window, see Fig.1.



Coding scheme

Figure 1

The CS algorithm is very simple and stable. In spite of that there are some cases that are not handled properly and as cases I, H and F, see Fig.1, are not distinguished and some computation is wasted. Several attempts have been made to speed up the CS algorithm. It is obvious that everybody can easily distinguish those cases and find directly which edges are intersected.

Let us assume characteristic situations from Fig.2 and denote c_A, c_B as CS codes of line segment's end points; $\alpha,\gamma,\eta,\omega$ areas as corner areas and β,θ,ξ,δ as side areas.

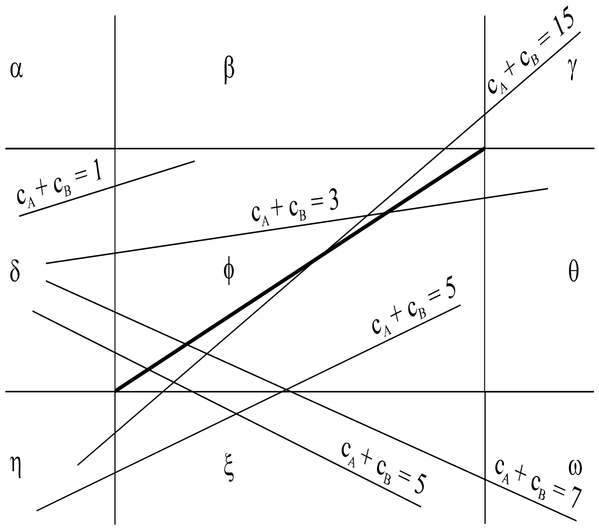


Figure 2

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 neighbouring 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 and led to the **LSSB** algorithm. Experimental results proved that expected speed up is in interval <**1.00, 2.08**>, see [Bui-Skala97] for details.

Line clipping is a similar problem to the line segment clipping problem solution discussed in the previous section. In spite of simplicity the CS algorithm cannot be used for clipping lines directly as there are no end points as we deal with lines. The well known algorithm for the line clipping is the Liang-Barsky algorithm (LB). It is based on clipping of the given line by each boundary line on which the rectangle edge lies. The given line which passes the points (x_A, y_A) and (x_B, y_B) is parametrically 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. Development of the LSSB algorithm led to the similar thoughts and as the result the LSB algorithm was derived and experimental verification showed that the expected speed up is in <**1.2 , 3.0**>, see [Bui-Skala97] for details.

From the above presented algorithms it can be seen that the algorithms complexity have not been changed but the speed up is substantial and in both cases the precise coding and deeper understanding of the problem has lead to the faster algorithms.

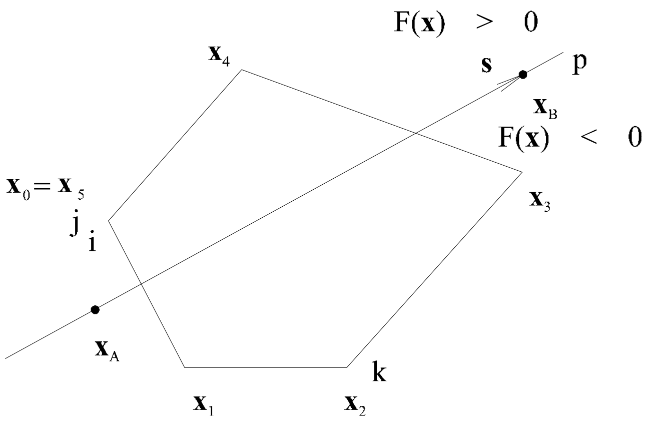
CLIPPING BY CONVEX POLYGON

Many algorithms for the line clipping by convex polygon has been proposed, nevertheless the Cyrus-Beck (CB) algorithm is the most used, simple to implement and stable. The CB algorithm relies on a brute force as it computes intersections all edges (or lines on which the edges lie) with the given line or a line on which the given line segments lies. It leads to ineffective algorithm, as ***N-2*** computed intersection points are lost because the only two can be valid.

In algorithm design there is always the very old rule "***Make tests first and then compute***". There were several attempts to find an intersection detection method, the one used the cross product of two vectors and brought significant speed up of the CB algorithm known as an ECB algorithm, for details see [Skala93]. Nevertheless the detection function can be replaced by the separation function with better speed up. Experiments proved the speed up can be expected in <**1.37 , 2.85**>.

The ECB algorithm does not use the known order of vertices of the given clipping polygon for a principal speed up of the algorithm, though it has the complexity *O(N)*. This understanding led to understanding that such knowledge could be used to decrease the algorithm complexity.

It can be shown that the test whether a line intersects the convex polygon is the dual problem to the test whether a point is inside of the convex polygon, which has optimal complexity of ***O(lg N)***. This has led to the question if there is a line clipping algorithm with *O(lg N)* complexity. Let us assume the situation from Fig.3.



Principle of the *O(lg N)* algorithm

Figure 3

It can be seen that if we select an index *k* as

$$k = (1 + N)/2$$

as in Fig.3 we need only *lg N* steps to find which edges are intersected on each chain of the edge segments. Although some other cases are a little bit more complicated it can be shown that the whole algorithm is of *O(lg N)* complexity, see Algorithm 1. It is necessary to point out that for effective implementation values *F(x$_i$)* should be stored in separate variables as they are used several times.

Algorithm 1

```
procedure CLIP 2D lg ( $\mathbf{x}_A$, $\mathbf{x}_B$);
{initialisation for a clipping window $\mathbf{x}_n$ := $\mathbf{x}_0$ }

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

function Solve ( i , j ): real;
{finds two nearest vertices on the opposite}
{sides of the given line p}
begin  while ( j - i ) ≥ 2 do { j ≥ i always }
         begin  k := ( i + j ) div 2; {shift right }
                if $(F(\mathbf{x}_i) * F(\mathbf{x}_k)) < 0$ then
                      j := k else i := k;
         end { while };
         Solve := Intersection ( p , $\mathbf{x}_i$ , $\mathbf{x}_j$ );
         {gives the value t for an point of the}
               {line p with the given segment $\mathbf{x}_i\,\mathbf{x}_j$ }
end { Solve };

begin {determine A, B, C values for the F(x) }
    $A := y_1 - y_2$;  $B := x_2 - x_1$;
    $C := x_1 * y_2 - x_2 * y_1$; i := 0; j := n;
    { for lines $t_{\min} := -\infty$;      $t_{\max} := \infty$;}
    { for line segments $t_{\min} := 0$; $t_{\max} := 1$; }
    while ( j - i ) ≥ 2 do
    begin k:= (i + j) div 2;{shift to the right}
     if $(F(\mathbf{x}_i) * F(\mathbf{x}_k)) < 0$ then
     begin $t_1$ := Solve ( i , k ) ;
        {finds an intersection on $\overline{\mathbf{x}_i\mathbf{x}_k}$ chain}
        $t_2$ := Solve ( k , j );
        {finds an intersection on $\overline{\mathbf{x}_k\mathbf{x}_j}$ chain}
   /*  if $t_1 > t_2$ then
        begin  t:= $t_2$; $t_2$:= $t_1$; $t_1$:= t  end;
          {compute $< t_1, t_2 > \cap < 0 , 1 >$ }
        $t_1$:=max($t_{\min}$, $t_1$); $t_2$:=min($t_{\max}$, $t_2$);
        if $< t_1, t_2 >= \emptyset$ then EXIT
        { exit procedure CLIP 2D lg };
        if $t_1 \le t_2$ then LINE ( $\mathbf{x}(t_1)$, $\mathbf{x}(t_2)$ );
   */{for segment clipping include those lines}
   end { if };

   if $F(\mathbf{x}_i) > 0$ then
   begin
        if $F(\mathbf{x}_i) < F(\mathbf{x}_k)$ then
        begin Delete_chain( i, j );
                if $F(\mathbf{x}_{i+1}) < F(\mathbf{x}_i)$ then
                begin j := k;
                  Delete_chain( k, j ); end
                else
                begin i := k;
                  Delete_chain ( i, k); end
        end else
        begin if $F(\mathbf{x}_{k+1}) > F(\mathbf{x}_k)$ then
                begin j := k;
                  Delete_chain ( k , j ); end
                else
                begin i := k;
                  Delete_chain ( i , k ); end
                end
        end
     else
     begin
```

```
      {similarly for opposite line orientation}
    end
  end { while }
end { CLIP-2D-lg }
```

This approach enabled to speed up clipping line significantly and it is even faster than CB algorithm for $N > 3$ (for $N = 100$ the speed up is over 10).

Nevertheless an algorithm for the test whether a point is inside of the convex polygon with *O(1)* expected complexity was developed [Skala94b] and it led to a question whether a line clipping algorithm with *O(1)* expected run-time complexity exists and what would be the complexity of pre-processing. Such algorithm was developed recently. The algorithm is based on the dual space representation and on non-orthogonal space subdivision, [Skala96b]. It was theoretically and experimentally proved that the algorithm has *O(1)* expected run-time complexity with pre-processing complexity $O(N^2)$, see [Skala-Lederbuch96].

Similar approach has been applied to E^3 problems and several experimental attempts have also been done for the case of line and line segment clipping in E^3. These thoughts and experiments led to new algorithms for

- line clipping by convex polyhedron with *O(N)* complexity [Skala96a], based on an idea that the given line *p* can be defined as an intersection of two nonparallel planes ρ_1 and ρ_2 . If the planes ρ_1 and ρ_2 intersect the triangle than the detailed computation is made. Expected speed up is up to **3,17** and the algorithm can be easily modified for non-convex polyhedron, too.
- line clipping by convex polyhedron with complexity $O(N^{1/2})$, based on the knowledge of all neighbours for each facet of the given polyhedron [Skala97a]. The expected speed up for polyhedron with 500 facets is about **4,5-4,9** against Cyrus-Beck algorithm.
- line clipping in E^3 with expected complexity *O(1)* with similar results to the E^2 case, [Skala-Lederbuch-Sup96d]. The algorithm is based on two orthogonal projections to E^2 co-ordinate system and on pre-processing of the given polyhedron. The experiments proved that the algorithm solves the problem very close to the constant time independently to the number of facets of the given polyhedron. It is obvious that such algorithm is convenient for cases when the clipping polyhedron is constant and many lines are clipped.

It is generally known that pre-processing will decrease the run-time complexity and computational geometry study this phenomena heavily. Nevertheless this approached proved that many fundamental algorithms do not use all known information known on the solved problem, that can lead to significant decrease of the run-time complexity.

QUESTIONS TO BE ANSWERED

Let us suppose we have an algorithm for the given problem. There are several questions to be answered:

- Is the algorithm the optimal one?
- What is the trade off between memory, run-time and pre-processing complexities?
- What is the lowest possible run-time complexity if the most sophisticated pre-processing is used?
- What kind of pre-processing complexity (time and memory) we can expect if we need faster solution of the given problem?
- Is the usage of pre-processing and parallel & distributed processing a "dual problem" in some sense if we need to speed up the solution?

Generally speaking - if we need to solve a problem in a limited time we have two choices

- to use parallel and distributed processing, but there is a limit for the total speed up,
- to use pre-processing if the major part of the problem is "constant" for many computed cases.

Those approaches seem to be hot topics nowadays especially in the field of the large data sets exploration parallel and distributed processing.

ACKNOWLEDEMENTS

I would like to express my thanks to all students of Computer Graphics courses and colleagues at the University of West Bohemia in Plzen that have stimulated this thoughts and development of new algorithms, for their suggestions and critical comments.

REFERENCES

Skala,V.1993 *An Efficient Algorithm for Line Clipping by Convex polygon*, Computers&Graphics, Vol.17, No.4, pp.417-421.

Kolingerova, I.1994 *3D-Line Clipping Algorithms - A Comparative Study*, Visual Computer, Vol.11, No.2, pp.96-104.

Skala,V. 1994 *O(lgN) Line Clipping Algorithm in E^2*, WSCG´94 International Conference, conference proceedings, pp.174-191.

Skala,V. 1994 *Point-in-Polygon with O(1) Complexity*, TR 68/94, Univ. of West Bohemia, Plzen.

Skala,V.1994 *O(lg N) Line Clipping Algorithm in E^2*, Computers & Graphics, Pergamon Press,Vol.18, No.4.

Skala,V. 1994 *An Algorithm for Line Clipping by Convex Polyhedron in E^3 with $O(N^{1/2})$ Complexity*, Preprint No.67, Univ.of West Bohemia, Plzen.

Skala,V., Kolingerova,I., Blaha,P. 1995 *A Comparison of 2D Line Clipping Algorithms*, Machine Graphics&Vision, Vol.3,No.4, pp.625-633.

Skala,V. 1996 *An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedra in E^3*, Computer Graphics Forum, Vol.15,No.1,pp.61-68.

Skala,V. 1996 *Line Clipping in E^2 with O(1) Processing Complexity*, Computers & Graphics, Vol.20, No.4, pp.523-530.

Skala,V., Lederbuch,P. ,Sup,B.1996 *A Comparison of O(1) and Cyrus-Beck Line Clipping Algorithm in E^2 and E^3*, SCCG96 Conference proceedings, Comenius Univ. Bratislava, Slovak Republic, pp.27-44.

Skala,V. 1996 *Line Clipping in E^3 with Expected Complexity O(1)*, Machine Graphics and Vision, Poland Academy of Sciences, Vol.5, No.4, pp.551-562.

Skala,V., Lederbuch,P.1996 *A Comparison of a New O(1) and the Cyrus-Beck Line Clipping Algorithms in E^2*, in COMPUGRAPHICS'96 Int.Conf., Paris.

Skala,V. 1997 *A Fast Algorithm for Line Clipping by Convex polyhedron in E^3*, Computers & Graphics, No.2,Vol.21, pp.209-214.

Bui, D.H., Skala,V.1997 *Fast Algorithms for Line Segment and Line Clipping in E^2*, submitted for publication in The Visual Computer.