The known algorithms for clipping lines against a general convex window 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 windows. If a 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 (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. Cyrus-Beck's (CB), Efficient Cyrus-Beck's (ECB) and Rappaport's algorithms have been compared with the new proposed O(lg N) algorithm.

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).

The Rappaport's algorithm [RAP91a] is the only one algorithm with O(lg N) complexity that could be used for line segments clipping against convex polygon. The algorithm, see alg. 1, is based on known fact that an answer whether a point is inside of the convex polygon can be given in O(lg N) steps, where N is a number of vertices of the given polygon [FRE85a].

procedure RAPPAPORT ( x_A, x_B );
( x_A, x_B are end-points of the clipped line segment )
begin
if CLASSIFY ( x_A ) = IN then
begin
(s,s1) := SECTOR ( x_A, x_B );
if x_B is to the left of s-s1 edge of the polygon
( s1 is the next vertex to vertex s )
then OUTPUT ( x_B ) { the line segment is totally inside } 
else
begin
compute the intersection point of the line segment with
the edge s-s1 ( x );
OUTPUT( x );
end
end
else
O(lg N) Line Clipping Algorithm in E^2
begin
(left_sup, right_sup) := SUPPORT_VERTICES (x_A);
if x_B is left of left_sup or right of right_sup
then DO NOTHING
else
begin { find an intersected edge from the front chain }
    (s, s1) := FRONT_SECTOR (left_sup, right_sup);
    if x_B is to the right of s-s1
    then DO NOTHING
    else
        begin { compute the intersection point of the line segment
                with the edge s-s1 (x);
            OUTPUT (x);
            (s, s1) := BACK_SECTOR (right_sup, left_sup);
            if x_B is to the left of s-s1
            then OUTPUT (x_B)
            else
                begin { find an intersected edge from the back chain }
                    compute the intersection point of the line segment
                    with the edge s-s1 (x);
                    OUTPUT (x);
                end
        end
end { RAFFAPORT };

Algorithm 1

There are used the following functions in alg. 1:

CLASSIFY (x) gives an answer if the point x is inside of the
given convex polygon in O(lg N) steps and has complexity

\{ (\text{:=}, \text{<}, \text{\&}, \text{/}) counting FFP operations only \}

(0, 2, 4, 4, 0) + lg N = (0, 1, 2, 2, 0),

SECTOR (x_A, x_B) finds an edge with vertices (s, s1) that is
intersected by the given line segment x_A x_B in O(lg N)
steps and has complexity

lg N = (7, 2, 9, 5, 0),

O(lg N) Line Clipping Algorithm in \(\mathbb{E}^2\)

SUPPORT_VERTICES (x_A) finds the (left_sup, right_sup) indexes
of end-points of the back and front chains that are
formed by edges of the given polygon in O(lg N) steps
and has complexity

(0, 2, 10, 4, 0) + lg N = (0, 2, 10, 4, 0),

FRONT_SECTOR (left_sup, right_sup) finds from front chain of
gives with vertices (s, s1) that is intersected by the
given line segment x_A x_B in O(lg N) steps and has complexity

lg N = (0, 1, 2, 2, 0),

BACK_SECTOR (left_sup, right_sup) finds from back chain of edges
with vertices (s, s1) that is intersected by the given
line segment x_A x_B in O(lg N) steps,

lg N = (0, 1, 2, 2, 0),

It can be seen that all steps are of O(lg N) complexity and
therefore the whole algorithm is of O(lg N) complexity, too.
Unfortunately some steps are quite complex and the overall
complexity for the worst case can be estimated as

(4, 2, 12, 22, 2) + lg N = (0, 4, 14, 8, 0)

Detailed description of the Rappaport's algorithm can be found in
[RAAP91a].

2. Proposed algorithm

\[ F(x) > 0 \]

\[ F(x) < 0 \]

\[ x \]

\[ p \]

\[ x \]

\[ d \]

\[ p \]

\[ O(lg N) \text{ Line Clipping Algorithm in } \mathbb{E}^2 \]

176

177
Let us suppose that we have a given convex clipping polygon anti-clockwise oriented and a line \( p \) is determined by two end-points:

\[
x_A = [x_A, y_A]^T, \quad x_B = [x_B, y_B]^T
\]

The convex window is represented by \( n + 1 \) points:

\[
x_i = [x_i, y_i]^T, \quad i = 0, \ldots, n
\]

where points \( x_0 \) and \( x_n \) are identical (column notation is used), \( x_i \) and \( y_i \) are coordinates of the vertex \( x_i \).

The notation \( \overline{x_i \ldots x_k} \) is used for a polyline from \( x_i \) to \( x_k \), i.e. it is a chain of line segments from \( x_i \) to \( x_k \).

Let us define the separation function \( F(x) \) in the form:

\[
F(x) = Ax + By + C
\]

where \( F(x) = 0 \) is an equation for the given line \( p \) and assume that the line has the orientation shown in fig.1.

\( x \) is defined as \( x = [x, y]^T \).

It can be seen (fig.2) that the oriented distance \( d \) of the point \( x \) 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 \( F(x) \) is actually proportional to the distance \( d \) for the given line \( p \).

First of all, let us assume that (see fig.1):

\[
i = 0, \quad j = n, \quad k = (i + j) / 2
\]

and:

\[
x_0 = x_B, \quad x_i = x_0, \quad x_j = x_n, \quad x_k = x_2
\]

Let us concentrate on a special case shown in fig.1. If the points \( x_i \) and \( x_k \) are on the opposite sides of the line \( p \), i.e.

\[
F(x_i) \cdot F(x_k) < 0
\]

then there must be just one intersection point on the chains \( \overline{x_i \ldots x_k} \) and \( \overline{x_B \ldots x_i} \), for each chain, because the given polygon is convex.

Because \( F(x_i) \cdot F(x_k) < 0 \) for the chain \( \overline{x_i \ldots x_k} \) there must exist an index \( l \) so that:

\[
F(x_l) \cdot F(x_{(l+1)}) < 0
\]

\( i \leq l < k \)

i.e. an edge \( x_l x_{l+1} \) must be intersected.

Similarly for the chain \( \overline{x_k \ldots x_j} \). It is obvious that in this case the intersection point can be found in \( O(\log M) \) steps using binary search over vertices, where \( M \) is a number of line segments in the given chain.

Unfortunately, other possible situations are more complex to solve, see fig.3. It is possible to distinguish four fundamental cases supposing the previously shown orientation of the separation function \( F(x) \). In case a) the chain \( \overline{x_i \ldots x_j} \) can be removed, while in case b) the chain \( \overline{x_i \ldots x_j} \) can be removed. In the first, resp. second, case index \( j \), resp. index \( i \), must be changed to \( k \). In both cases a new value of \( k \) must be computed as:

\[
k = (i + j) / 2
\]

Both mentioned cases can be distinguished by a criterion:

\[
F(x_{(i+1)}) < F(x_i)
\]

because if \( F(x_{(i+1)}) < F(x_i) \) then the chain \( \overline{x_i \ldots x_k} \) can intersect the line \( p \), see fig.3. This condition actually expresses that we are getting closer to the line \( p \), i.e. the oriented distance \( d \) is smaller.

In both cases we assumed that the line \( p \) has the shown orientation, i.e. \( F(x_i) > 0 \) and:

\[
F(x_i) \leq F(x_k)
\]

Possible situations as a variation of cases a) and b) in fig.3, when this condition is not true, are shown as cases c) and d).

A little bit more complex situation is shown by cases c) and d) where \( F(x_j) > F(x_k) \). In case c) the chain \( \overline{x_i \ldots x_j} \) can be removed, while in case d) the chain \( \overline{x_i \ldots x_k} \) can be removed. In the first, resp. second, case index \( j \), resp. index \( i \), must be changed to \( k \). In both cases a new value of \( k \) must be again determined as:

\[
k = (i + j) / 2
\]

Both last mentioned cases can be distinguished by using criterion:

\[
F(x_{(k+1)}) > F(x_k)
\]

Actually we must distinguish whether we are getting closer to the given line \( p \) or not. If the line \( p \) has an opposite orientation then similar situations must be solved, see alg.2.

This procedure is repeated until:

\[
F(x_i) \cdot F(x_k) < 0
\]

\( O(\log N) \) Line Clipping Algorithm in \( \mathbb{E}^2 \)
Now it can be seen that all parts of the proposed algorithm are of complexity $O(\lg N)$, where $N$ is a number of edges in the given chain because we used for all steps the binary search over vertices of the clipping convex polygon. The whole proposed $O(\lg N)$ algorithm is described by alg.2. 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.

```
procedure CLIP_2D-lg ( x_A, x_B );
    { Note: initialization of the clipping window x_B := x_0 }

    function macro F ( x ); real;
    { should be 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 := j - i; always }
        begin k := (i + j) div 2; { shift to the right }
            if (F(x_k) - F(x_i)) < 0 then j := k
                else i := k;
        end ( while );
    SOLVE := INTERSECTION (p, x_A, x_j);
    { gives the value t of an intersection point }
    { of the line p with the given line segment x_k x_j }
    end ( SOLVE );

    begin { determine the A, B, C values for the function 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(x_k) - F(x_i)) < 0 then
```

*Figure 3*

If this condition becomes true we will obtain two chains $x_i x_k$ and $x_k x_j$ that intersect the line $p$ and binary search over vertices can be used again as we get a similar situation shown in fig.1.

$O(\lg N)$ Line Clipping Algorithm in $E^2$
begin { see fig.1 } 
\( t_1 := \text{SOLVE}(i,k); \) \{ find an intersection on \( \overline{x_i x_k} \) chain \} 
\( t_2 := \text{SOLVE}(k,j); \) \{ find an intersection on \( \overline{x_k x_j} \) chain \} 
\{ for the line segment clipping include the next 5 lines \} 
\{ if \( t_1 > t_2 \) then begin \( t := t_1; \) \( t_1 := t_2; \) \( t_2 := t_1 \) end; \} 
\{ compute \( \langle t_1, t_2 \rangle \) as \( \langle t_1, t_2 \rangle \cap \langle 0,1 \rangle \) \} 
\{ if \( t_1 = 0 \) \} 
\{ if \( t_1 \leq t_2 \) then \}
\( \text{SHOW_LINE}(x(t_1), x(t_2)); \) 
\} \{ exit procedure CLIP_2D lg \};
end { if; }
\{ for the polygon orientation shown in fig.3 \}
\if F(x_j) > 0 \then 
begin { for the orientation of line p shown in fig.3 \}
\if F(x_j) < F(x_k) \then \{ cases a and b \}
begin { DELETE CHAIN(i,j) removes the chain \( \overline{x_i x_j} \) }
\if F(x_{i+1}) < F(x_i) \then 
\begin{align*}
\text{begin } j := k; \{ \text{DELETE CHAIN(k,j); case a } \} \text{ end}
\end{align*}
\text{begin } i := k; \{ \text{DELETE CHAIN(i,k); case b } \} \text{ end}
\end{align*}
\else \{ cases c and d \}
begin
\if F(x_{k+1}) > F(x_k) \then 
\begin{align*}
\text{begin } j := k; \{ \text{DELETE CHAIN(k,j); case c } \} \text{ end}
\end{align*}
\text{begin } i := k; \{ \text{DELETE CHAIN(i,k); case d } \} \text{ end}
\end{align*}
\else \{ for an opposite orientation of the line p \}
\if F(x_j) > F(x_k) \then 
begin 
\if F(x_{i+1}) > F(x_i) \then 
\begin{align*}
\text{begin } j := k; \{ \text{DELETE CHAIN(k,j); } \text{ end}
\end{align*}
\text{begin } i := k; \{ \text{DELETE CHAIN(i,k); } \text{ end}
\end{align*}
\else 
begin
\begin{align*}
\text{begin } j := k; \{ \text{DELETE CHAIN(k,j); } \text{ end}
\end{align*}
\text{begin } i := k; \{ \text{DELETE CHAIN(i,k); } \text{ end}
\end{align*}
\end{align*}
\end{end}
\end{align*}

Algorithm 2

3. Theoretical analysis and experimental results

Before making any experiments it is convenient to point out that time needed for operations \( :=, <, \leq, *, / \) differ significantly from computer to computer.

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
float & := & < & \leq & * & / \\
\hline
time & 33 & 50 & 16 & 20 & 114 \\
\hline
\end{tabular}
\end{center}

Times for \( 5.10^6 \) floating point operations in 1/10 sec. for PC 486/33MHz

Table 1

Let us introduce coefficients of the effectiveness \( \nu \) as
\[ \nu_1 = \frac{T_{CB}}{T}, \quad \nu_2 = \frac{T_{CB}}{T_0}, \quad \nu_3 = \frac{T_R}{T} \]
where \( T_{CB}, T_0, T_R, T \) are execution times needed by Cyrus-Beck's, ECB, Rappaport's and proposed \( O(lg N) \) algorithms.

Description of CB and ECB algorithms can be found in [SKA93b] together with their theoretical and experimental comparisons.

Generally it is possible to express the complexity of the CB algorithm
\[ (8, 3, 6, 4, 0) + (5, 3, 7, 4, 1) \times N \]
and time of computation as \( T_{CB} \) (for PC 486, see tab.1) can be estimated
\[ O(lg N) \text{ Line Clipping Algorithm in } \mathbb{E}^2 \]
The complexity of the ECB algorithm (in the worst case) as
\[
T_{CB} = 590 + 621 \times N
\]
and time of computation \( T_0 \) can be estimated as
\[
T_0 = 1329 + 257 \times N
\]
Description of CB and ECB algorithms and their theoretical and experimental comparisons can be found in [SKA93b]. Their complexities are \( O(N) \).
Complexity of the Rappaport's algorithm can be expressed as
\[
(4, 12, 22, 2) + (0, 4, 14, 8, 0) \times \lfloor \log(N + 1) \rfloor
\]
and time of computation \( T_R \) can be estimated as
\[
T_R = 1092 + 584 \times \lfloor \log(N + 1) \rfloor
\]
while for the suggested algorithm \( O(\log N) \) the complexity is given as
\[
(14, 4, 11, 15, 2) + (2, 4, 6, 6, 0) \times \lfloor \log(N + 1) \rfloor
\]
and time of computation \( T \) can be estimated as
\[
T = 1267 + 376 \times \lfloor \log(N + 1) \rfloor
\]
The Rappaport's and proposed algorithms are of \( O(\log N) \) complexity. Theoretical speed up is given in tab.2 (the worst cases and operations in floating point were considered only).

<table>
<thead>
<tr>
<th>( N )</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu_1 )</td>
<td>1.28</td>
<td>1.54</td>
<td>1.80</td>
<td>2.06</td>
<td>2.01</td>
<td>2.23</td>
<td>2.45</td>
<td>4.13</td>
<td>6.11</td>
<td>8.98</td>
<td>10.08</td>
</tr>
<tr>
<td>( \nu_2 )</td>
<td>0.98</td>
<td>1.09</td>
<td>1.20</td>
<td>1.31</td>
<td>1.22</td>
<td>1.31</td>
<td>1.41</td>
<td>2.06</td>
<td>2.87</td>
<td>4.02</td>
<td>6.93</td>
</tr>
<tr>
<td>( \nu_3 )</td>
<td>1.19</td>
<td>1.19</td>
<td>1.19</td>
<td>1.19</td>
<td>1.24</td>
<td>1.24</td>
<td>1.24</td>
<td>1.27</td>
<td>1.27</td>
<td>1.30</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Theoretical estimations (worst case)

Table 2

\( O(\log N) \) Line Clipping Algorithm in \( E^2 \)
The proposed algorithm has been tested against Cyrus-Beck's, ECB and Rappaport's 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.

There are practically no significant differences as far as the percentage is intersecting lines is concerned. See tab. 3.

It can be seen that, see tab. 3, that the proposed algorithm is significantly faster than CB algorithm. A comparison of ECB and proposed algorithms shows that for \(N < 7\) the ECB algorithm is faster than the proposed one. "Waves" for \(\nu_2\) are caused by the influence of binary division of an index interval and relation between data and convex polygon position. The waves can be seen in tab. 2 with theoretical estimations, too. The significant difference for \(N = 100\) is caused by considering the worst cases only in theoretical estimations.

The proposed \(O(\log N)\) algorithm is approx. two times faster than Rappaport's algorithm and it is much more simple to implement.

It is necessary to point out that careful implementation of conditions like to \(F(x_1) \leq F(x_k)\) might further improve the efficiency of the proposed algorithm, because of comparison operation is the longest operation after division, see tab. 1.

4. Conclusion

The new efficient algorithm of \(O(\log N)\) complexity for clipping lines against convex window in \(E^2\) has been developed. Edges of the given convex polygon can be arbitrarily oriented. It also proved the applicability of Computational Geometry results [CHA87a] even for small \(N\). Similarly as the Rappaport's algorithm the proposed algorithm can be easily modified for polygon clipping. The suggested algorithm also proved the duality principle with the problem point-in-polygon, see [FRE85a], [NIE92a], [NIE92b]. It also proved applicability of principles of Computational Geometry results [CHA87a] even for small \(N\). Similarly as Rappaport's algorithm the proposed algorithm can be modified for polygon clipping, where the clipped polygon might be non-convex. Superiority of the proposed algorithm over CB, ECB and Rappaport's algorithms was proved by theoretical estimations and experimental results.

All tests were implemented in Borland C++ on PC 486/33 MHz 256KB Cache. It is expected that for workstations the efficiency \(\nu\) will be higher than for PC 486 as the comparison operation is the longest operation used in the algorithm, see tab. 2, and the timing ratio of operations on workstations is better.

5. Acknowledgments

The author would like to express his thanks to students of Computer Graphics courses at the University of West Bohemia in Plzen and Charles's University in Prague who stimulated this work, especially to Mr. P.Blaha for careful test implementations and verification of the proposed algorithm, dr.A.Ferko and dr.F.Jezek for reading a manuscript, critical comments and suggestion they made and to anonymous referees who made critical and constructive recommendations that improved this paper very much.

6. References


\(O(\log N)\) Line Clipping Algorithm in \(E^2\)
O(\lg N) Line Clipping Algorithm in $\mathbb{E}^2$
O(lg N) Line Clipping Algorithm in $E^2$