

A Novel Line Convex Polygon Clipping Algorithm in E2 with Parallel Processing Modification

Vaclav Skala(✉)

Department of Computer Science and Engineering,
University of West Bohemia, 301 00 Pilsen, Czech Republic
skala@kiv.zcu.cz
http://www.VaclavSkala.eu

Abstract. This paper presents a new approach to line clipping by a convex polygon problem solution. The algorithm is based on a separation function, which separates the polygon vertices to the left or right hand side of the given line. It leads to numerically robust algorithms in comparison to the well-known Cyrus-Beck's algorithm and its modifications.

The proposed algorithm has $O(N)$ complexity, but supports parallel processing and simple implementation in hardware.

The presented approach has also impact to the algorithm design methodology and importance of a detailed analysis in algorithm development, if the algorithm robustness and efficiency is required.

Keywords: Line clipping · Line segment clipping · Cyrus-Beck algorithm · Convex polygon clipping · Homogeneous coordinates · Projective representation · Duality

1 Introduction

There are many algorithms for a line clipping or a line segment clipping by a convex polygon with many modifications. Probably the mostly published algorithms are devoted to a line or line segment clipping by a rectangular window in E^2, which was motivated by computer graphics output devices and Window-Viewport operations, the Cohen-Sutherand's (CS) [8] and Liang-Barsky (LB) [12] algorithms are the most known for line segment and line segment clipping in E^2 with several modification and improvements, e.g. Nicholl-Lee-Nicholl [13], Bui [2], Skala [16,33], Andreev [1], Day [5], Dörr [7], Duvalenko [6], Kaijian [10], Krammer [11], Liang [12], Sobkow [35] and Zhang [36]. Line clipping in E^2 using homogeneous coordinates was introduced by Nielsen [14] and optimized line segment clipping for the normalized window was published in Skala [26,31]. A new classification scheme for the line segment end-points is introduced in Skala [32].

Supported by the University of West Bohemia - Institutional research support No. 1311.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12953, pp. 3–15, 2021.
https://doi.org/10.1007/978-3-030-86976-2_1

However, clipping by a convex polygon is a little bit more complicated problem as it depends on number of vertices of the given convex polygon. Probably the Cyrus-Beck (CB) [4] algorithm is the most known for line segment and line segment clipping in the E^2 case having applicability also in the E^3 case for clipping by a convex polyhedron. The Cyrus-Beck algorithm has $O(N)$ computational complexity. The Cyrus-Beck algorithm was modified for non-convex polygons with self-intersecting edges and quadratic curves in Skala [17,18]. A line convex clipping algorithm based on space subdivision was introduced by Slater [34]. A line clipping algorithm based on shearing transformation was published by Huang [9], algorithm for a polygon clipping was published in Rappoport [15].

The algorithm for a line and line segment clipping by a convex polygon with $O(lgN)$ complexity was described in Skala [20]. Complexity decrease is possible due to "ordering" of vertex indexes of the convex polygon in the E^2 case. Unfortunately, this is not extensible for the E^3 case, i.e. line clipping by a convex polyhedron, as in the E^3 case no ordering of vertices is available. The algorithm with $O_{exp}(\sqrt{N})$ was introduced by Skala [23] for the case when the polyhedron is represented by a triangular mesh using information on the neighbours of triangles. A line intersection algorithm with a non-convex polyhedron in E^3 was introduced in Skala [21,25].

In the E^2 case, if the convex polygon is constant and many lines or line segments are to be clipped, it is possible to pre-compute the convex polygon using dual space representation and the point-in-convex polygon location strategy Skala [24]. Then, the line segment clipping algorithm is $O_{exp}(1)$ run-time complexity, Skala [22]. The algorithm was extended for the E^3 case in Skala [23].

In the following, a new approach to the line clipping by a convex polygon in E^2 is described in comparison to the Cyrus-Beck's algorithm.

2 Cyrus-Beck's Algorithm

The Cyrus-Beck's (CB) algorithm is well known and is used in many computer graphics courses due to its simplicity and applicability for the E^3 case.

The Cyrus-Beck's algorithm is based on direct intersection computation of the given line p in the parametric form and a line on which the polygon edge e_i lies, see Fig. 1, in the implicit form, i.e. on a solution of two linear equations (vector notation is used):

$$\begin{aligned} p: &\quad \mathbf{x}(t) = \mathbf{x}_A + \mathbf{s}\, t \\ e_i: &\quad \mathbf{n}_i^T \mathbf{x} + c_i = 0 \end{aligned} \tag{1}$$

Solving those equations, the parameter t for the intersection point is obtained as:

$$e_i: \quad \mathbf{n}_i^T \mathbf{x}_A + \mathbf{n}_i^T \mathbf{s}\, t + c_i = 0 \tag{2}$$

and therefore

$$t = -\frac{\mathbf{n}_i^T \mathbf{x}_A + c_i}{\mathbf{n}_i^T \mathbf{s}} \tag{3}$$

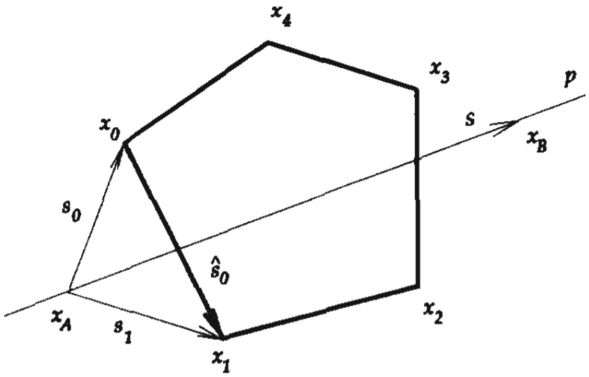


Fig. 1. Clipping against the convex polygon in E^2

It can be seen, that there is an instability of the algorithm as if the line p is parallel or nearly parallel to the edge e_i, the expression $\mathbf{n}_i^T\mathbf{s} \to 0$ and $t \to \pm\infty$. The fraction computation might cause an overflow or high imprecision of the computed parameter t value, see Fig. 2.

It is hard to detect and solve reliably such cases and programmers usually use a sequence like

$$\textbf{if } |\mathbf{n}_i^T\mathbf{s})| \ < \ \textit{eps} \textbf{ then } \textit{a singular case}$$

which is incorrect solution as the value *eps* is a programmer choice. Unfortunately, text books do not point this in spite of this dangerous construction as far as the robustness and computational stability is concerned.

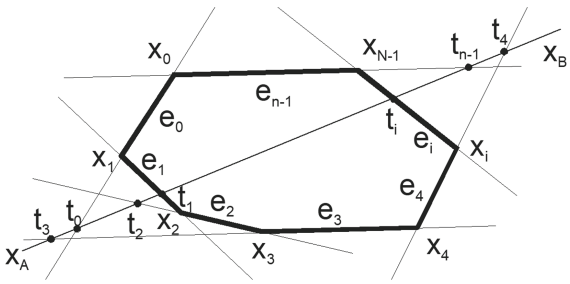


Fig. 2. Cyrus-Beck clipping algorithm against the convex polygon in E^2

The modification of the Cyrus-Beck's algorithm using the cross product for more reliable detection of the "close to singular" cases was described by Skala [19]. However, it is computationally more expensive and not solving the "close to singular" cases in total.

Algorithm 1. Cyrus-Beck's Line Clipping Algorithm

```
1: for i := 0 to N-1 do
2:     Compute n_i and c_i for all polygon edges
3:                                  ▷ pre-computation for the given convex polygon
4: procedure C-B-Clip(x_A, x_B);                    ▷ line is given by two points
5:     t_min := −∞; t_max := ∞;          ▷ set initial conditions for the parameter t
6:     s := x_B − x_A;                      ▷ computation of the line coefficients
7:     for i := 0 to N − 1 do                                    ▷ for each edge
8:         q := n_i^T s;                                        ▷ pre-computation
9:         if abs(q) < eps then NOP;          ▷ Singular case-usual solution
10:        else
11:            t = −(n_i^T x_A + c_i)/n_i^T s;
12:            if q < 0 then t_min := max(t, t_min);
13:            else t_max := min(t, t_max);
14:            end if
15:        end if
16:    end for                                  ▷ all convex polygon edges processed
17:    if t_min < tmax then          ▷ intersection of a line and the polygon exists
18:        { x_B := x_A + s t;   x_A := x_A + s t; }
19:    end if
20: end procedure
```

The Cyrus-Beck's algorithm for a line clipping is described by the Algorithm 1. It can be easily modified for a line segment clipping just restricting the range of the parameter t to $<0, 1>$, i.e.

$$<t_{min}, t_{max}> := <t_{min}, t_{max}> \cap <0, 1>$$

It can be seen, that the algorithm complexity is of $O(N)$ and the division operation, which is the most consuming time operation in the floating point representation, is used N times. However, only 2 values of the parameter t are valid, i.e. $N-2$ computations of the parameter t are lost. Also reliable detection of the "close to singular" cases is difficult and time consuming.

In following, the S-Convex-Clip algorithms based on implicit formulation using projective representation will be presented. It is based on the classification of the window corners against the given line in the implicit form with high numerical stability.

3 Proposed Algorithm

The majority of line clipping algorithms in the E^2 and E^3 cases have been developed for the Euclidean space representation in spite of the fact, that geometric transformations, i.e. projection, translation, rotation, scaling and Window-Viewport etc., use homogeneous coordinates, e.g. projective representation.

This results into necessity to convert the results of the geometric transformations to the Euclidean space using division operation as follows:

$$\mathbf{X} = (X, Y) \quad \mathbf{x} = [x, y : w]^T \qquad X = \frac{x}{w} \quad Y = \frac{y}{w} \quad w \neq 0 \tag{4}$$

where (X, Y) are the point coordinates in the Euclidean space E^2, while $[x, y : w]^T$ are in the homogeneous coordinates Skala [28–30]; similarly in the E^3 case. It should be noted, that ":" is used in the notation to point out, that the w is the homogeneous coordinate and has no physical unit in the contrary of the x, resp. y which has a physical unit, e.g. meters [m].

If a point is given in the Euclidean space, its homogeneous coordinates are given as $\mathbf{x} = [X, Y : 1]^T$, i.e. $w = 1$. The homogeneous coordinates also enable to represent a point close or in infinity, i.e. when $w \to 0$, and postpone the division operations. It leads to better numerical robustness and computational speed-up especially if GPU or SSE instructions are used.

4 S-Convex-Clip

Let us consider a typical example of a line clipping by the convex clipping window, see Fig. 1, and a line p given in the implicit form using the projective notation:

$$p: \quad ax + by + cw = 0 \quad , i.e. \quad \mathbf{a}^T\mathbf{x} = 0 \tag{5}$$

where $\mathbf{a} = [a, b : c]^T$ are coefficients of the given line p, $\mathbf{x} = [x, y : w]^T$ is a point on this line using projective notation (w is the homogeneous coordinate). It can be seen, that if the Eq. 5 is divided by $w \neq 0$, then:

$$a\frac{x}{w} + b\frac{y}{w} + c\frac{w}{w} = 0 \quad , \; i.e. \quad aX + bY + c = 0 \tag{6}$$

The advantage of the projective notation is, that a line p passing two points $\mathbf{x}_A$, $\mathbf{x}_B$ or an intersection point $\mathbf{x}$ of two lines p_1, p_2 can be computed due to the principle of duality as Coxeter [3] and Skala [30]:

$$\mathbf{p} = \mathbf{x}_A \wedge \mathbf{x}_B \quad , \quad \mathbf{x} = \mathbf{p_1} \wedge \mathbf{p_2} \tag{7}$$

where $\mathbf{a} \wedge \mathbf{b}$ is the outer product application on the vectors $\mathbf{a}$, $\mathbf{b}$ using homogeneous coordinates (application the cross-product $\mathbf{a} \times \mathbf{b}$ is used)

The line p is given by two points as:

$$\mathbf{p} = \mathbf{x}_A \times \mathbf{x}_B = [a, b : c]^T = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_A & y_A & w_A \\ x_B & y_B & w_B \end{bmatrix} \tag{8}$$

where $\mathbf{i} = [1, 0 : 0]^T$, $\mathbf{j} = [0, 1 : 0]^T$, $\mathbf{k} = [0, 0 : 1]^T$. Now, an intersection point of two given lines is given as:

$$\mathbf{x} = \mathbf{p}_1 \times \mathbf{p}_2 = [x, y : w]^T = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \tag{9}$$

where $\mathbf{i} = [1, 0 : 0]^T$, $\mathbf{j} = [0, 1 : 0]^T$, $\mathbf{k} = [0, 0 : 1]^T$.

Let us consider an implicit function $F(\mathbf{x}) = \mathbf{a}^T\mathbf{x}$. The line p is then defined as $F(\mathbf{x}) = 0$. The clipping operation should determine intersection points $\mathbf{x}_i = [x_i, y_i : w_i]^T$, $i = A, B$, of the given line p with the convex polygon edges, if any. The line p splits the E^2 plane into two parts, see Fig. 1. The corners $\mathbf{x}_i$, $i = 0, ..., N-1$, of the convex polygon are split into two groups according to the sign value of the function $F(\mathbf{x}_i)$, $i = 0, \ldots, N-1$. It means that the ith corner is classified by a bit value c_i as:

$$c_i = \begin{cases} 1 & if\ F(\mathbf{x}_i) \geq 0 \\ 0 & otherwise \end{cases} \qquad i = 0, \ldots, N-13 \tag{10}$$

and it is actually an application of the dot product as $\mathbf{a}^T\mathbf{x} \equiv \mathbf{a} \bullet \mathbf{x}$.

This leads to the $O(N)$ computational complexity of the S-Convex-Clip algorithm without the division operation use at all, see Algorithm 2.

Algorithm 2. S-Convex-Clip - Line clipping algorithm by the convex polygon

```
1: procedure S-Convex-Clip(x_A, x_B);                    ▷ line is given by two points
2:                                                       ▷ x_k = [x_k, y_k : w_k]^T;  i = A, B
3:     p := x_A ∧ x_B;     ▷ computation of the line coefficients - use the cross product
4:     for i := 0 to N − 1 do                            ▷ to be done in parallel par for
5:         if p^T x_i ≥ 0 then c_i := 1 else c_i := 0;                ▷ codes computation
6:     end for
7:           ▷ the bit vector c contains code of all polygon vertices against the line p
8:     if  c ≠ [0...0]^T and c ≠ [1...1]^T then                  ▷ line intersects the window
9:         i := TAB1[c];   x_A := p ∧ e_i;                          ▷ first intersection point
10:        j := TAB2[c];   x_B := p ∧ e_j;                          ▷ second intersection point
11:        output(x_A, x_B)          ▷ operator ∧ means the cross-product application
12:    else
13:        NOP                                          ▷ line does not intersect the window
14:    end if
15: end procedure
```

As the indexes of the intersected are known at the lines 9 and 10 of the S-Convex-Clip algorithm, the relevant parameter t can be determined similarly as in the original Cyrus-Beck's algorithm or the coordinates of the intersection points computed directly using the outer product as shown in the algorithm.

It can be seen, that the S-Convex-Clip algorithm, see Algorithm 2, is quite simple. Computational complexity $O(N)$ is needed to determine the code vector $\mathbf{c}$ using *dot product* and only two intersection computations are needed, if an intersection exists. It means, that the algorithm requires:

- *dot product* operations: N (line 5)
- comparison operations in the floating point: N (line 5)
- *cross product* operations: 2 (lines 3 and 9,10)

The S-Convex-Clip algorithm is significantly computationally simpler than the Cyrus-Beck's algorithm and the causes of instability of the Cyrus-Beck's algorithm were removed.

It should be noted, that in the GPU and SSE instructions use, the algorithm gets much faster as the cross product and dot products takes one clock on GPU. Also the points of intersections remain in the projective notation, i.e. $\mathbf{x}_A = [x_A, y_A : w_A]^T$ and $\mathbf{x}_B = [x_B, y_B : w_B]^T$, which can be used for further processing without direct need to converting them to the Euclidean space. In this case, no division operations are needed at all.

Table 1. All cases for $N = 4$; N/A - Non-Applicable (impossible) cases

c	**c**	TAB1	TAB2	MASK
0	0000	None	None	None
1	0001	0	3	0100
2	0010	0	1	0100
3	0011	1	3	0010
4	0100	1	2	0010
5	0101	N/A	N/A	N/A
6	0110	0	2	0100
7	0111	2	3	1000

c	**c**	TAB1	TAB2	MASK
15	1111	None	None	None
14	1110	3	0	0100
13	1101	1	0	0100
12	1100	3	1	0010
11	1011	2	1	0010
10	1010	N/A	N/A	N/A
9	1001	2	0	0100
8	1000	3	2	1000

The values in TAB, illustrative table for $N = 4$ is presented by Table 1, can be generated synthetically for general N. As the table is symmetrical in some sense, only 1/2 of the cases are needed to be generated; the other cases can be determined using the bit-wise negation, i.e. **not c**, and swapping columns TAB1 and TAB2.

It should be noticed, that there is no need to generate the whole Table 1 for the given N, especially if N is higher, as the intersected edges of the convex polygon can be easily detected from the bit vector $\mathbf{c}$. It can be seen, that if there is a sequence "$\ldots 0, 1 \ldots$" or "$\ldots 1, 0 \ldots$" the relevant convex polygon edges are intersected. The Table 1 generation is to be used in the case of several lines and constant N of the convex polygon clipping, while the second possibility, i.e. finding "$\ldots 0, 1 \ldots$" and "$\ldots 1, 0 \ldots$", is to be used in cases, when N is changing.

As the clipping polygon is convex, the only two edges might be intersected. It means, that only one sequence $\ldots 0,1\ldots$ and $\ldots 1,0\ldots$ can occur, which simplifies detection of the edges intersected, if it is made "on the fly", not by using pre-generated table. Identification of those sequences is simple, just using binary shift and binary mask operations.

The proposed S-Convex-Clip algorithm can be easily modified also for the line segment clipping case similarly as in line segment against rectangular window Skala [27]. In this case, the MASK column of the Table 1 is used and this can be again generated for the given N or determined on the fly case by case.

It can be seen, that the proposed S-Convex-Clip algorithm is computationally robust, limiting unnecessary computations in the floating point representation. In addition, it does not use the division operation, if the resulting coordinates of the end-points of intersections are not needed to be converted to the Euclidean space.

5 Conclusion

This contribution describes shortly a new robust line clipping algorithm against a convex polygon with $O(N)$ computational complexity. It eliminates instability of "close to singular" cases, which causes instability in the Cyrus-Beck's algorithm. It also significantly reduces the floating point operations, especially division operations.

As the proposed algorithm uses projective notation, there is no need to convert points and polygon vertices from the homogeneous coordinates to the Euclidean space, which requires unnecessary division operations as well.

The algorithm is convenient for implementations using GPU or/and SSE instructions as it supports parallel processing and additional speed up as the cross product and dot product are implemented in hardware. Experiments proved over 10–15% speedup against the original Cyrus-Becks algorithm for small N and grows with the convex polygon vertices N substantially due to saving unnecessary intersection computation in the floating point representation.

Acknowledgment. The author would like to thank to colleagues at the University of West Bohemia for fruitful discussions and to anonymous reviewers for their comments and hints, which helped to improve the manuscript significantly. A special thanks belong to recent students of computer Science and the University of West Bohemia at Pilsen, who made many tests of line clipping algorithms developed and modified at the site.

Appendix A

The TABLE generation for a general N is a little bit tricky, but it is simple from the algorithm point of view. Let us consider $N = 6$ as an example. Then all the cases which can occur, except of lines that do no not intersect the convex polygon, are geometrically presented in the Table 5. It can be seen, that they are invariant to rotation, from the geometrical point of view. Analyzing all those cases, a simple pattern of similarity can be detected, see Table 2, Table 3 and Table 4. The cases S_0 and S_6 represent the cases, when a line does not intersect the convex polygon. The TABLE Tab. 1 for a general N-sided convex polygon is has $2^N - 1$ entries and many of those are the non applicable cases. However, the number of the "applicable" cases, which can occur in the line clipping, is $N*(N-1)+2$, only. In the case $N = 6$, we have 30 possible different intersections and 2 cases for the "not intersecting" cases, as we have to respect line segment orientation.

Table 2. Cases I & II; N means Non-Applicable N/A case

case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	0	0	0	0	0	1	5	0
S_2	0	0	0	0	1	1	5	1
S_3	0	0	0	1	1	1	5	2
S_4	0	0	1	1	1	1	5	3
S_5	0	1	1	1	1	1	5	4
S_6	1	1	1	1	1	1	N	N

case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	0	0	0	0	1	0	0	0
S_2	0	0	0	1	1	0	0	1
S_3	0	0	1	1	1	0	0	2
S_4	0	1	1	1	1	0	0	3
S_5	1	1	1	1	1	0	0	4
S_6	1	1	1	1	1	1	N	N

Table 3. Cases III & IV; N means Non-Applicable N/A case

case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	0	0	0	1	0	0	1	0
S_2	0	0	1	1	0	0	1	1
S_3	0	1	1	1	0	0	1	2
S_4	1	1	1	1	0	0	1	3
S_5	1	1	1	1	0	1	1	4
S_6	1	1	1	1	1	1	N	N

case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	0	0	1	0	0	0	2	0
S_2	0	1	1	0	0	0	2	1
S_3	1	1	1	0	0	0	2	2
S_4	1	1	1	0	0	1	2	3
S_5	1	1	1	0	1	1	2	4
S_6	1	1	1	1	1	1	N	N

Table 4. Cases V & VI; N means Non-Applicable N/A case

case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	0	1	0	0	0	0	3	0
S_2	1	1	0	0	0	0	3	1
S_3	1	1	0	0	0	1	3	2
S_4	1	1	0	0	1	1	3	3
S_5	1	1	0	1	1	1	3	4
S_6	1	1	1	1	1	1	N	N

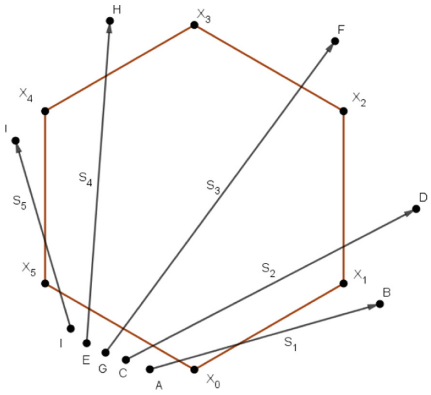
case	C_5	C_4	C_3	C_2	C_1	C_0	i_0	i_1
S_0	0	0	0	0	0	0	N	N
S_1	1	0	0	0	0	0	4	0
S_2	1	0	0	0	0	1	4	1
S_3	1	0	0	0	1	1	4	2
S_4	1	0	0	1	1	1	4	3
S_5	1	0	1	1	1	1	4	4
S_6	1	1	1	1	1	1	N	N

Algorithm 3. S-Convex-Clip-Table-Generator

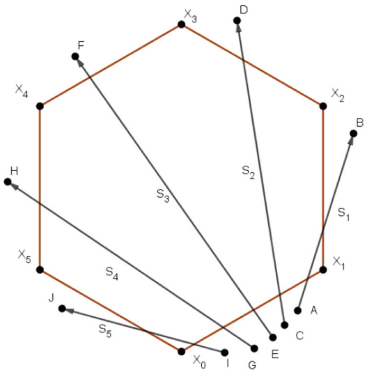
```
1: # This is a sequence for generating the TABLE #
2: # This sequence can be further optimized #
3: M := 2^N − 1;                          ▷ M is a bit vector of "1" of the length N
4: for i := 0 to M do                                  ▷ initialization of the TABLE
5:     TAB1[i] := -1; TAB2[i] := -1;          ▷ "-1" means the "N' or "N/A" cases
6: end for
7: TAB1[0] := -1; TAB2[0] := -1;          ▷ settings for the "non-intersecting" cases
8: TAB1[M] := -1; TAB1[M] := -1;
        ▷ bit-vectors are 2^N bit long independently of the unsigned integer length
9: C_ones := M;              ▷ 2^N long bit mask                ▷ C_ones = [111...111]
10: C_zeros := 0;                                               ▷ C_zeros = [000...000]
11:
12: C_A := C_zeros + 1;                                        ▷ setting the bit C_0 to "1"
13: k := N-1;   ▷ setting index if the index of the last edge - avoiding mod operation
14: for ii := 0 to N − 1 do                          ▷ for the each case I, II, ..., V, VI do
15:                 ▷ C_A contains bit vector setting for the S_1 for all the cases I,..,VI
16:     Generate_Sequences(C_A, ii, k);
17:     C_A := (C_A shl 1);       ▷ shift left without carry - setting for all the S_1 cases
18:     k := ii;
19: end for
20:
21: procedure GENERATE_SEQUENCES(C_A, ii, k);          ▷ generation if the ii-table
22:     C_temp := C_A;
23:     for i := 1 to N − 1 do    ▷ setting the i-th row of the TABLE for the S_i case
24:         index := C_temp;            ▷ code C_temp converted to the unsigned integer
25:         TAB1[index] := k; TAB2[index] := i;
26:         carry := C_temp[N − 1];        ▷ set carry to the most left bit of the C_temp
27:                                  ▷ simulates the "circular shift" on N bits
28:         C_temp := (C_temp shl 1) + carry;
                                ▷ shift left with carry transfer to the C_temp[0] bit
29:     end for
30: end procedure
```

It should be noted, that the generated codes respect the line, resp. line segment orientation as well Skala [27,31].

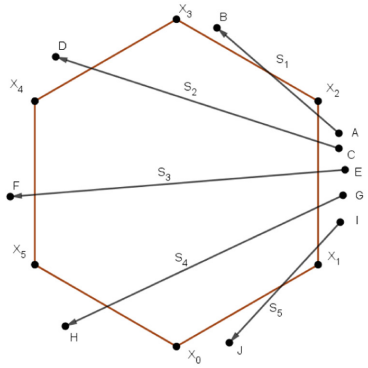
Table 5. All possible cases for $N = 6$ except of lines passing out the convex polygon



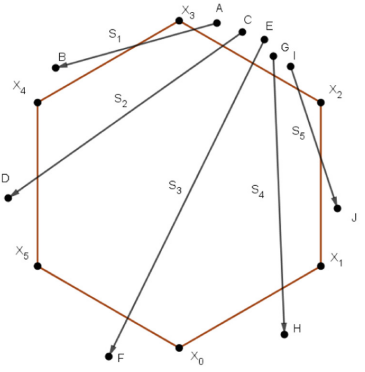
Cases I



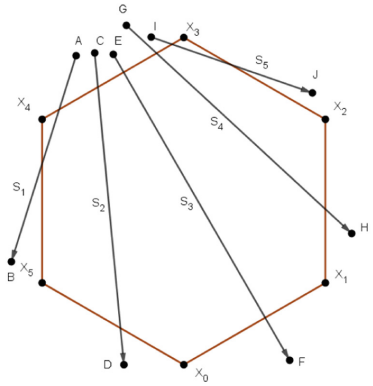
Cases II



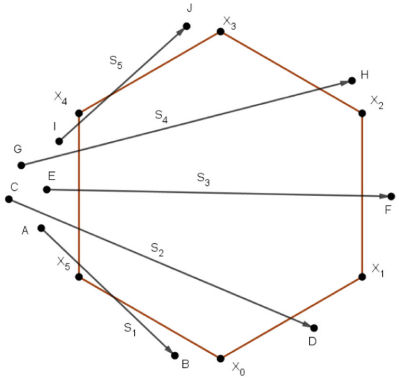
Cases III



Cases IV



Cases V



Cases VI

References

1. Andreev, R., Sofianska, E.: New algorithm for two-dimensional line clipping. Comput. Graph. **15**(4), 519–526 (1991)
2. Bui, D., Skala, V.: Fast algorithms for clipping lines and line segments in E2. Vis. Comput. **14**(1), 31–37 (1998)
3. Coxeter, H.: Introduction to geometry. Math. Gaz. **48**(365), 343 (1964)
4. Cyrus, M., Beck, J.: Generalized two- and three-dimensional clipping. Comput. Graph. **3**(1), 23–28 (1978)
5. Day, J.: A new two dimensional line clipping algorithm for small windows. Comput. Graph. Forum **11**(4), 241–245 (1992)
6. Duvanenko, V., Robbins, W., Gyurcsik, R.: Line-segment clipping revisited. Dr. Dobb's J. **21**(1), 107–110 (1996)
7. Dörr, M.: A new approach to parametric line clipping. Comput. Graph. **14**(3–4), 449–464 (1990)
8. Foley, D., van Dam, A., Feiner, S., Hughes, J.: Computer Graphics: Principles and Practice. Addison-Wesley, Reading (1990)
9. Huang, Y., Liu, Y.: An algorithm for line clipping against a polygon based on shearing transformation. Comput. Graph. Forum **21**(4), 683–688 (2002)
10. Kaijian, S., Edwards, J., Cooper, D.: An efficient line clipping algorithm. Comput. Graph. **14**(2), 297–301 (1990)
11. Krammer, G.: A line clipping algorithm and its analysis. Comput. Graph. Forum **11**(3), 253–266 (1992)
12. Liang, Y.-D., Barsky, B.: A new concept and method for line clipping. ACM Trans. Graph. (TOG) **3**(1), 1–22 (1984)
13. Nicholl, T.M., Lee, D., Nicholl, R.A.: Efficient new algorithm for 2D line clipping: its development and analysis. Comput. Graph. (ACM) **21**(4), 253–262 (1987)
14. Nielsen, H.: Line clipping using semi-homogeneous coordinates. Comput. Graph. Forum **14**(1), 3–16 (1995)
15. Rappoport, A.: An efficient algorithm for line and polygon clipping. Vis. Comput. **7**(1), 19–28 (1991)
16. Skala, V.: Algorithm for 2D line clipping, pp. 121–128. New Advances in Computer Graphics, NATO ASI (1989)
17. Skala, V.: Algorithms for 2D line clipping. In: EG 1989 proceedings (1989)
18. Skala, V.: Algorithms for clipping quadratic arcs. In: Chua, T.-S., Kunii, T.L. (eds.) CG International '90, pp. 255–268. Springer, Tokyo (1990). https://doi.org/10.1007/978-4-431-68123-6_16
19. Skala, V.: An efficient algorithm for line clipping by convex polygon. Comput. Graph. **17**(4), 417–421 (1993)
20. Skala, V.: O(lg N) line clipping algorithm in E2. Comput. Graph. **18**(4), 517–524 (1994)
21. Skala, V.: An efficient algorithm for line clipping by convex and non-convex polyhedra in E3. Comput. Graph. Forum **15**(1), 61–68 (1996)
22. Skala, V.: Line clipping in E2 with O(1) processing complexity. Comput. Graph. (Pergamon) **20**(4), 523–530 (1996)
23. Skala, V.: Line clipping in e3 with expected complexity O(1). Mach. Graph. Vis. **5**(4), 551–562 (1996)
24. Skala, V.: Trading time for space: an O(1) average time algorithm for point-in-polygon location problem. Theoretical fiction or practical usage? Mach. Graph. Vis. **5**(3), 483–494 (1996)

25. Skala, V.: A fast algorithm for line clipping by convex polyhedron in E3. Comput. Graph. (Pergamon) **21**(2), 209–214 (1997)
26. Skala, V.: A new line clipping algorithm with hardware acceleration. In: Proceedings of Computer Graphics International Conference - CGI, pp. 270–273 (2004)
27. Skala, V.: A new approach to line and line segment clipping in homogeneous coordinates. Vis. Comput. **21**(11), 905–914 (2005)
28. Skala, V.: Length, area and volume computation in homogeneous coordinates. Int. J. Image Graph. **6**(4), 625–639 (2006)
29. Skala, V.: Barycentric coordinates computation in homogeneous coordinates. Comput. Graph. (Pergamon) **32**(1), 120–127 (2008)
30. Skala, V.: Intersection computation in projective space using homogeneous coordinates. Int. J. Image Graph. **8**(4), 615–628 (2008)
31. Skala, V.: Optimized line and line segment clipping in E2 and geometric algebra. Ann. Math. Inform. **52**, 199–215 (2020)
32. Skala, V.: A new coding scheme for line segment clipping in E2. In: Gervasi, O., et al. (eds.) ICCSA 2021. LNCS, vol. 12953, pp. 16–29. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86976-2_2
33. Skala, V., Bui, D.: Extension of the Nicholls-Lee-Nichols algorithm to three dimensions. Vis. Comput. **17**(4), 236–242 (2001)
34. Slater, M., Barsky, B.: 2D line and polygon clipping based on space subdivision. Vis. Comput. **10**(7), 407–422 (1994)
35. Sobkow, M., Pospisil, P., Yang, Y.-H.: A fast two-dimensional line clipping algorithm via line encoding. Comput. Graph. **11**(4), 459–467 (1987)
36. Zhang, M., Sabharwal, C.: An efficient implementation of parametric line and polygon clipping algorithm. In: ACM Symposium on Applied Computing, pp. 796–800. ACM (2002)