Annales Mathematicae et Informaticae
***52** (2020) pp. 199–215*
DOI: https://doi.org/10.33039/ami.2020.05.001
URL: https://ami.uni-eszterhazy.hu

Optimized line and line segment clipping in E2 and Geometric Algebra*

Vaclav Skala

University of West Bohemia, Pilsen, Czech Republic
www.VaclavSkala.eu

Submitted: November 28, 2019
Accepted: May 1, 2020
Published online: May 11, 2020

Abstract

Algorithms for line and line segment clipping are well known algorithms especially in the field of computer graphics. They are formulated for the Euclidean space representation. However, computer graphics uses the projective extension of the Euclidean space and homogeneous coordinates for representation geometric transformations with points in the E^2 or E^3 space. The projection operation from the E^3 to the E^2 space leads to the necessity to convert coordinates to the Euclidean space if the clipping operation is to be used.

In this contribution, an optimized simple algorithm for line and line segment clipping in the E^2 space, which works directly with homogeneous representation and not requiring the conversion to the Euclidean space, is described. It is based on Geometric Algebra (GA) formulation for projective representation.

The proposed algorithm is simple, efficient and easy to implement. The algorithm can be efficiently modified for the SSE4 instruction use or the GPU application, too.

Keywords: line clipping, line segment clipping, homogeneous coordinates, projective space, geometric algebra, principle of duality, GPU, SSE4 instruction.

MSC: 65D18, 68U05

*The research was partially supported by the Czech Science Foundation, Czech Republic, project GACR No. GA17-05534S

1. Introduction

The line and line segment clipping are fundamental and critical operations in the computer graphics pipeline as all the processed primitives have to be clipped out of the drawing area to decrease computational requirements and also respect the physical restrictions of the hardware. The clipping operations are mostly connected with the Window-Viewport and projection operations. There are many algorithms developed recently with many modifications, see Andreev [1], Day [4], Dörr [5], Duvalenko [8], Kaijian [12], Krammer [14], Liang [16], Sobkow [29].

However, those algorithms 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, i.e. projective representation. This results in the necessity to convert the results of the geometric transformations to the Euclidean space using division operation.

The conversion of a point $\mathbf{x} = [x, y : w]^T$ from homogeneous coordinates to the Euclidean representation $\mathbf{X} = (X, Y)$ is given as:

$$X = x/w, \quad Y = y/w, \quad w \neq 0,$$

where w is the homogeneous coordinate. It means, that a point $\mathbf{X} \in E^2$ is represented by a line in the projective space $x, y : w$ without the origin, which represents a point in the infinity, see Figure 1.

The extension to the E^3 case is straightforward, e.g. Foley [9].

$$X = x/w, \quad Y = y/w, \quad Z = z/w, \quad w \neq 0,$$

where $\mathbf{x} = [x, y, z : w]^T$. The use of the projective extension of the Euclidean space is convenient not only for geometric transformations, as it replaces addition by multiplication in the case of translation operation, but also it enables to represent a point in infinity. Also, it enables to express some geometric entities in more compact form, e.g. a line in the E^2 case as:

$$aX + bY + c = 0, \quad ax + by + cw = 0, \quad \mathbf{a}^T\mathbf{x} = 0, \tag{1.1}$$

where $\mathbf{a} = [a, b : c]^T$. It is necessary to note, that (a, b) represents the normal vector of a line, while c is related to the distance of a line from the origin of the Euclidean coordinate system. Similarly, a plane in the E^3 case is defined as:

$$aX + bY + cZ + d = 0, \quad ax + by + cz + dw = 0, \quad \mathbf{a}^T\mathbf{x} = 0, \tag{1.2}$$

where $\mathbf{a} = [a, b, c : d]^T$.

However, it is necessary to distinguish vectors, as "movable" entities, from "frames", which have the origin as the reference point. It is necessary to note, that metric is not defined in the projective space. In many cases, the principle of duality can be used to derive a solution of a dual problem and have only one programming sequence for both problems, i.e. the primary one and the dual. Unfortunately, the principle of duality is not usually part of the standard computer science curricula.

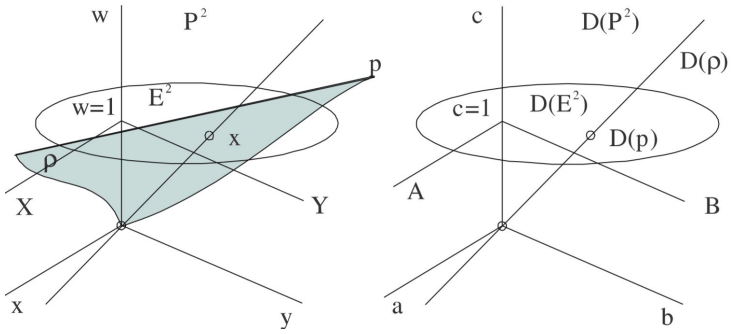

Figure 1: Projective space and its dual

2. Principle of duality

The principle of duality is one of the most important principles in mathematics. In our case of geometric problems described by linear equations, see (1.1) and (1.2), the principle of duality states that any theorem remains true when we interchange the words

- "point" and "line" in the E^2 case, resp. "point" and "plane" in the E^3 case,
- "lie on" and "pass through", "join" and "intersection" and so on.

Once the theorem has been established, the dual theorem is obtained as described by Johnson [11].

In other words, the principle of duality in the E^2 case says, that in all theorems it is possible to substitute the term "point" by the term "line" and term "line" by the term "point" and the given theorem remains valid. This helps a lot in the solution of some geometrical problems, similarly in the E^3 case. It means, that the intersection computation of two lines is dual to the computation of a line given by two points in the E^2 case.

Similarly, the intersection computation of three planes is dual to the computation of a plane given by three points in the E^3 case.

It is strange as the usual solution in the first case leads to formulation $\mathbf{Ax} = \mathbf{b}$, while in the second case, the parameters of a line are determined as $\mathbf{Ax} = \mathbf{0}$. However, if the projective representation is used, both cases are solved as $\mathbf{Ax} = \mathbf{0}$, Skala [23].

This is the direct impact of the fact, that the point must lie on a line in the E^2 case, resp. on a plane in the E^3 case, (2.1). Also, two lines in the E^2 case, respectively three planes in the E^3 case must not be collinear, i.e.:

$$\mathbf{a}^T\mathbf{x} = 0 \tag{2.1}$$

where $\mathbf{a} = [a, b : c]^T$ in E^2, resp. $\mathbf{a} = [a, b, c : d]^T$ in E^3. It can be seen that the meaning of the term $\mathbf{a}$ and $\mathbf{x}$ can be interchanged due to the principle of duality.

Let us consider the intersection of two lines in the E^2 case. Both lines must not be collinear, the conditions (2.1) for each line must be orthogonal to other, therefore the result of the outer product (cross product) must be zero. Similarly, for planes which must be non-collinear Lengyel [15], Skala [22–24].

Let us consider two lines $\mathbf{a}_1 = [a_1, b_1 : c_1]^T$ and $\mathbf{a}_2 = [a_2, b_2 : c_2]^T$ in the E^2 case (using the cross-product notation extended to the $x, y : w$ coordinate system). Then the intersection point $\mathbf{x} = [x, y : w]^T$ is given as:

$$\mathbf{a}_1^T\mathbf{x} = 0, \quad \mathbf{a}_2^T\mathbf{x} = 0, \quad (\mathbf{a}_1 \times \mathbf{a}_2)^T\mathbf{x} = 0.$$

Using the matrix notation:

$$\det \begin{bmatrix} x & y & w \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = 0.$$

It means that a point given as the intersection of two lines is given as:

$$\mathbf{x} = \mathbf{a}_1 \times \mathbf{a}_2 \quad \text{i.e.} \quad \mathbf{x} = \mathbf{a}_1 \wedge \mathbf{a}_2,$$

where $\mathbf{x} = [x, y : w]^T$ and $\wedge$ means the outer product.

As a direct consequence of the principle of duality a line $\mathbf{a} = [a, b : c]^T$ given by two points $\mathbf{x}_1 = [x_1, y_1 : w_1]^T$ and $\mathbf{x}_2 = [x_2, y_2 : w_2]^T$ is given as:

$$\mathbf{a} = \mathbf{x}_1 \times \mathbf{x}_2 \quad \text{i.e.} \quad \mathbf{a} = \mathbf{x}_1 \wedge \mathbf{x}_2. \tag{2.2}$$

It should be noted that the operator $\times$ is the equivalent specific symbol used in the E^3 case, while $\wedge$ is defined for the n-dimensional space, in general.

Extension to the E^3 dimensional case is quite simple due to multi-linearity. It means, that the intersection point of three planes $\mathbf{a}_i$, $i = 1, 2, 3$ is given as:

$$\det \begin{bmatrix} x & y & z & w \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{bmatrix} = 0.$$

It means that the point given as an intersection of three planes is given as:

$$\mathbf{x} = \mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \mathbf{a}_3,$$

where $\mathbf{x} = [x, y, z : w]^T$.

As a direct consequence of the principle of duality, a plane $\mathbf{a} = [a, b, c : d]^T$ given by three points $\mathbf{x}_i = [x_i, y_i, z_i : w_i]^T$, $i = 1, 2, 3$ is given as:

$$\mathbf{a} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_2.$$

3. Geometric algebra

Linear algebra is used for formulation and solution of many engineering problems, including a solution of geometrically oriented problems, e.g. in computer vision or computer graphics. Usually, vectors or matrices are used to represent one-dimensional or two-dimensional (data) structure and standard operations are defined. For vectors in the mathematical sense, basic mathematical operations are defined, e.g. addition, multiplication (dot-product, cross-product), etc. This "standard" vector algebra framework enables basic operation with geometric entities.

3.1. Geometric product

However, there is another framework called Geometric Algebra (GA), which comes from the William Kingdom Clifford formulation and which enables to define the product of union and intersection operations with points, lines, areas, volumes and hyper-volumes in general, see Vince [30], Kanatani [13]. The GA is an alternative formalism for describing geometrical entities and operations in n-dimensional space. It uses only one product (multiplication) called *geometric product* defined as:

$$\mathbf{ab} = \mathbf{a} \bullet \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$$

where $\mathbf{a} \bullet \mathbf{b}$ is the *dot* (scalar) product and $\mathbf{a} \wedge \mathbf{b}$ is the *outer product* (equivalent to the *cross-product* in the E^3 case).

It can be seen that the geometric product is "strange" as the result consists of a scalar value and a bivector (usually called as a vector, but having different properties and representation from a vector), which is the result of the outer product.

The geometric product can be easily computed as the non-commutative tensor product as $\mathbf{a} \otimes \mathbf{b}$, see Skala [28], as:

$$\mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{bmatrix}.$$

The diagonal elements represent the *dot product* part, while the upper and the lower triangular matrices represent the *outer product* part. It should be noted, that the geometric product computation using the non-commutative tensor product can be used for the n-dimensional space, too.

Nowadays, the geometric algebra is widely used across many fields, but mostly in connection with vector oriented operations in the Euclidean space. The applications of GA can be found in Physics (Hestenes [10]), Computer Science (Dorst [6]), Computer Graphics (Vince [30], Lengyel [15]), and in other engineering fields as well (Dorst [7]).

3.2. Geometric product and projective space

It should be noted, that it is possible to extend the GA for the projective extension of the Euclidean space as well. In the case of computer graphics, points are not

represented by vectors in the mathematical sense, as they are represented by a vector data structure, which represents a frame fixed to the origin of the coordinate system.

As mentioned in Chapter 2, computation of a line $\mathbf{p}$ by given two points and an intersection point $\mathbf{x}$ given as an intersection of two lines is given by the outer product as:

$$\mathbf{p} = \mathbf{x}_1 \wedge \mathbf{x}_2, \quad \mathbf{x} = \mathbf{p}_1 \wedge \mathbf{p}_2.$$

Using the determinant notation:

$$\det \begin{bmatrix} a & b & c \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0, \quad \det \begin{bmatrix} x & y & w \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = 0.$$

It can be seen, there is no need to convert homogeneous coordinates of the points to the Euclidean space, since the determinant is multi-linear.

Extension to the E^3 case is straightforward, i.e. a plane ρ given by three points and an intersection point $\mathbf{x}$ of three planes are given as:

$$\rho = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3, \quad \mathbf{x} = \rho_1 \wedge \rho_2 \wedge \rho_3.$$

Using the determinant notation, the intersection point of three planes, respectively the plane given by three points is are given as:

$$\det \begin{bmatrix} a & b & c & d \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_1 & y_3 & z_3 & w_3 \end{bmatrix} = 0, \quad \det \begin{bmatrix} x & y & z & w \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_1 & b_3 & c_3 & d_3 \end{bmatrix} = 0.$$

It can be seen, that there is no problem with singular cases, like collinear lines, respectively planes, as the intersection is in infinity. In this case, the homogeneous coordinate of the result is $w = 0$ or $w \mapsto 0$. This is to be evaluated after outer product computation.

4. Line clipping

The line clipping operation in E^2 space is a fundamental problem in Computer Graphics. It was already deeply analyzed and many algorithms have been developed. The Cohen-Sutherland (CS) [9] for a line segment clipping against the rectangular window, the Liang-Barsky (LB) [16] and Cyrus-Beck (CB) [3] (extensible to the E^3 case) algorithms for clipping a line against a convex polygon, the Nichol-Lee-Nichol (LNL) [17] (modified by Skala [27]) are the most used algorithms.

However, some more sophisticated algorithms or modification of the recent ones have been developed recently, e.g. line clipping against a rectangular window, see Bui [2], Skala [20, 21], line clipping by a convex polygon with $O(\lg N)$ complexity,

see Skala [26] (based on Rappaport [18]), or algorithm with $O_{\text{expected}}(1)$ complexity, see Skala [25], etc. In the case of E^3, the algorithms have computational complexity $O(N)$ as there is no ordering in E^3 case, however, the algorithm with $O_{\text{expected}}(\text{sqrt}(N))$ have been developed by Skala [19, 20, 25].

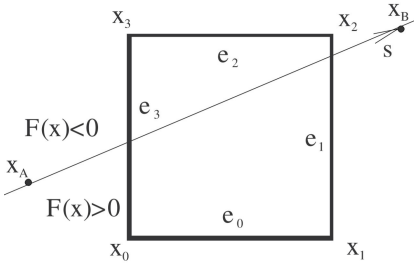

Figure 2: Clipping against the rectangular window in E^2

The line and line segment clipping algorithms against a rectangular window in E^2 are probably the most used algorithms and any improvements or speed up can have a high influence on the efficiency of the whole graphics pipeline.

Let us consider a typical example of a line clipping by a rectangular clipping window, see Figure 2, and a line p given in the implicit form using projective notation.

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

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.

In the following, a version of the line clipping algorithm for the general case will be described, which can be easily extended to a line clipping and line segment clipping by a convex polygon, and the optimization for the use in the Normalized Device Coordinate (NDC) system.

4.1. S-L-Clip algorithm

Let us consider an implicit function $F(\mathbf{x}) = \mathbf{a}^T\mathbf{x}$. The clipping operation should determine intersection points $\mathbf{x}_i = [x_i, y_i : w_i]^T$, $i = 1, 2$ of the given line with the window, if any. The line splits the plane into two parts, see Fig. 2. The corners of the window are split into two groups according to the sign of the $F(\mathbf{x})$ value. This results into Smart-Line-Clip (S-L-Clip) algorithm, see Algorithm 1.

It means that each corner can be classified by a bit value c_i as:

$$c_i = \begin{cases} 1 & \text{if } F(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad i = 0, 1, 2, 3,$$

where $\mathbf{a} = [a, b : c]^T$ are coefficients of the given line p, $\mathbf{x} = [x, y : w]^T$ means a point on this line.

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	None
13	1101	1	01	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

Table 1: All cases; N/A - Non-Applicable (impossible) cases

Table 1 shows the codes for all situations (some of those are not possible). The **TAB1** and **TAB2** contain indices of edges of the window intersected by the given line (values in the **MASK** will be used in the line segment algorithm).

It can be seen, that the S-L-Clip algorithm (see Algorithm 1) is quite simple and easily extensible for the convex polygon clipping case as well (Table 1 can be generated synthetically). It is significantly simpler than the Liang-Barsky algorithm [16]. It also supports SSE4 and GPU use directly and leads to simple implementations, as the cross-product and dot-product operations, are supported in hardware. It should be noted, that the algorithm is designed for a very general case, as the window corners and the points defining the line, are generally in the projective representation, i.e. $w \neq 1$. Therefore, the S-L-Clip algorithm has further potential for optimization, especially for the case, when the corner points of the window are given in the Euclidean coordinates, i.e. $w = 1$, and clipping is made in the Normalized Device Coordinate (NDC) system.

Algorithm 1 S-L-Clip - Line clipping algorithm by the rectangular window

```
1: procedure S-L-Clip(x_A, x_B);                    ▷ line is given by two points
2:     p := x_A ∧ x_B;                              ▷ computation of the line coefficients
3:     for i := 0 to 3 do
4:         if p^T x_i ≥ 0 then c_i := 1 else c_i := 0;        ▷ codes computation
5:     end for
6:     if c ≠ [0000]^T and c ≠ [1111]^T then        ▷ line intersects the window
7:         i := TAB1[c];   x_A := p ∧ e_i;          ▷ first intersection point
8:         j := TAB2[c];   x_B := p ∧ e_j;          ▷ second intersection point
9:         output(x_A, x_B);
10:    end if
11: end procedure
```

4.2. S-L-Clip-Opt - Optimization of the S-L-Clip

The S-L-Clip algorithm can be optimized for the use in the E^2 case, as the corners of the window and points defining the line are in the Euclidean coordinates, i.e. $w = 1$, and the edges of the window are vertical or horizontal only. Also, it is necessary to consider the computer graphics pipeline, where all primitives passing the clipping operations are transformed from the World Coordinates (WC) to the Normalized Device Coordinates(NDC) and then to the Device Coordinates (DC), where NDC coordinates are $\langle 0, 1\rangle \times \langle 0, 1\rangle$ or $\langle -1, 1\rangle \times \langle -1, 1\rangle$, which simplifies the outer-product (cross-product) computation significantly.

These computational transformations can be described as:

$$\mathbf{x}' = \mathbf{T}_{NDC \mapsto DC}\ \mathbf{CLIP}\ (\mathbf{T}_{WC \mapsto NDC}\ \mathbf{x}).$$

Let us consider a line coefficients determination first, using (2.2), and setting $w = 1$. Then the coefficients of the line are given by (4.1).

$$a = y_1 - y_2, \quad b = x_2 - x_1, \quad c = x_1 * y_2 - x_2 * y_1. \tag{4.1}$$

It leads to a significant reduction of a number of the floating point operations $(\pm, *)$ from $(6, 3)$ to $(3, 2)$. Also, the outer product is used for computation of the intersection points, i.e. $\mathbf{x}_A$ and $\mathbf{x}_B$, can be simplified.

As the edges of the window are vertical or horizontal only and clipping is done in the normalized space NDC, the codes of the corners and related intersection point computation can be simplified significantly. It means, that for each edge of the window the intersection computation with the line can be simplified as:

$\begin{bmatrix} x & y & w \\ a & b & c \\ 0 & 1 & 0 \end{bmatrix} = 0$	$\begin{bmatrix} x & y & w \\ a & b & c \\ 1 & 0 & -1 \end{bmatrix} = 0$	$\begin{bmatrix} x & y & w \\ a & b & c \\ 0 & 1 & -1 \end{bmatrix} = 0$	$\begin{bmatrix} x & y & w \\ a & b & c \\ 1 & 0 & 0 \end{bmatrix} = 0$
edge e_0	edge e_1	edge e_2	edge e_3

Table 2: Explicit evaluation of an intersection point for each edge

It means that the outer product sequence for the line intersection with an edge can be replaced by direct computing of all cases shown in Table 1. Rewriting those conditions at Table 2, the intersection for each edge is given as:

edge e_0:	$x = -c$,	$y := 0$,	$w := a$,
edge e_1:	$x = -b$,	$y := a + c$,	$w := -b$,
edge e_2:	$x = -b - c$,	$y := a$,	$w := a$,
edge e_3:	$x = 0$,	$y := c$,	$w := -b$.

It leads to another significant reduction of the number of the floating point operations $(\pm, *)$ from $(6, 3)$ to $(1, 0)$ in the most of cases. If the line does not intersect the window, there is no computation at all.

The optimized line clipping algorithm for the E^2 case is represented by the algorithm S-L-Clip-Opt, see Algorithm 2.

From Algorithm 2, it can be seen that also the evaluation of the window corners were simplified as instead of $4 * (2, 3)$ with floating point operations, only $(4, 0)$ operations are needed. This results in an additional speedup of the proposed optimization.

Algorithm 2 Optimized S-L-Clip-Opt line clipping algorithm in E^2

```
1: procedure S-L-Clip-Opt(x_A, x_B);                    ▷ line is given by two points
2:     a = y_1 − y_2;   b = x_2 − x_1;
3:     c = x_1 * y_2 − x_2 * y_1;                                  ▷ line coefficients
4:     c_0 := sign(c);   c_1 := sign(a + c);                ▷ corner's codes computation
5:     c_2 := sign(a + b + c);   c_3 := sign(b + c);                ▷ c = [c_3, c_2, c_1, c_0]^T
6:     if c ≠ [0000]^T and c ≠ [1111]^T then                ▷ line intersects the window
7:         i := TAB1[c];                                              ▷ x_A := [x_A, y_A : w_A]^T
8:         switch i do                                          ▷ equivalent of x_A := p ∧ e_i;
9:             case 0: x_A := −c;   y_A := 0;   w_A := a;
10:            case 1: x_A := −b;   y_A := a + c;   w_A := b;
11:            case 2: x_A := −b − c;   y_A := −a;   w_A := a;
12:            case 3: x_A := 0;   y_A := c;   w_A := −b;
13:            default: ERROR                                  ▷ actually the N/A case
14:        end switch
15:        j := TAB2[c];                                              ▷ x_B := [x_B, y_B : w_B]^T
16:        switch j do                                          ▷ equivalent of x_B := p ∧ e_j;
17:            case 0: x_B := −c;   y_B := 0;   w_B := a;
18:            case 1: x_B := −b;   y_B := a + c;   w_B := b;
19:            case 2: x_B := −b − c;   y_B := −a;   w_B := a;
20:            case 3: x_B := 0;   y_B := c;   w_B := −b;
21:            default: ERROR                                  ▷ actually the N/A case
22:        end switch
23:        output(x_A, x_B);                  ▷ output with the intersection points
24:    end if
25: end procedure
```

Now, the proposed optimized algorithm is to be modified for the line segment clipping case, which is used nearly exclusively in computer graphics.

5. Line segment clipping

In computer graphics, geometric elements like points, line segments, triangles, etc. are processed. Therefore, the proposed algorithm is to be modified for the

line segment clipping case, see Figure 3.

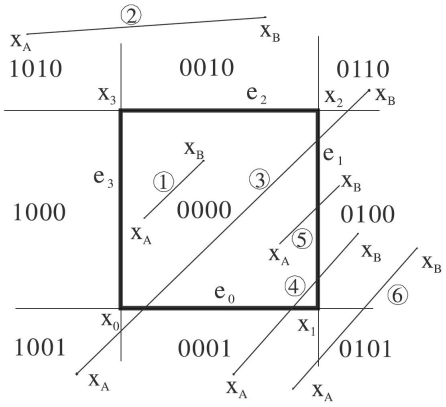

Figure 3: The codes of line segment end-points

It can be seen that there are some special line positions, which lead to direct acceptance or rejection of the whole line segment, while other cases have to be processed.

5.1. End-points coding

A line segment is defined by its end-points $\mathbf{x}_A$ and $\mathbf{x}_B$. The classification of the line segment end-points and the corners of the window mutual positions enables faster processing, see Algorithm 3. The end-point classification was used in the CS algorithms developed by Cohen-Sutherland [9]. Some additional coding for speedup were introduced in Bui [2]. It enables simple rejection of line segments not intersecting the window and direct acceptance of segments totally inside of the window. If $\mathbf{c}_A$ and $\mathbf{c}_B$ are codes of the end-points then the sequence catching those cases can be expressed as:

if $(\mathbf{c}_A$ **or** $\mathbf{c}_B) = [0000]$ **then** the line segment is totally inside;

if$(\mathbf{c}_A$ **and** $\mathbf{c}_B) \neq [0000]$ **then** the line segment is outside;

If the end-points of a line are given in the Euclidean space, i.e. $w = 1$, then the codes of the end-points are determined as in Algorithm 3. In the general case, i.e. when $w \neq 1$ and $w > 1$ the conditions must be modified using multiplication as $xw_{\min} < x_{\min}w$, etc. and therefore no division operation is needed.

It can be seen, that other cases, see Figure 3, cannot be directly distinguished by the CS algorithm coding and intersection points are to be computed, including the invalid ones. It is necessary to note, that the CS algorithm uses division operations in the floating point.

The S-LS-Clip algorithm (Algorithm 4) is derived from the S-L-Clip algorithm (Algorithm 1), which uses the gained information on positions of the line segment end-points.

Algorithm 3 End-point code computation

```
1: procedure CODE (c, x);                    ▷ code c for the position x = [x, y : 1]^T
2:     c := [0000]^T;                                              ▷ initial setting
3:     if x < x_min then c := [1000]^T          ▷ setting according to x coordinate
4:         if x > x_max then c := [0100]^T;
5:     if y < y_min then c := c lor [0001]^T  ▷ setting according to y coordinate
6:         if y > y_max then c := c lor [0010]^T;
7:                                  ▷ lor represents or operation on all bits
8: end procedure
```

Algorithm 4 Smart-line segment clipping algorithm by the rectangular window

```
1: procedure S-LS-CLIP(x_A, x_B);                    ▷ two line segment end-points
2:     CODE (c_A, x_A); CODE (c_B, x_B);     ▷ code for the end-points x_A and x_B
3:     if (c_A lor c_B) = [0000]^T then output (x_A, x_B); EXIT
4:                                                  ▷ the whole segment is inside
5:     if (c_A land c_B) ≠ [0000]^T then EXIT     ▷ the whole segment is outside
6:     p := x_A ∧ x_B;                          ▷ computation of the line coefficients
7:     for i := 0 to 3 do
8:         if p^T x_i ≥ 0 then c_i := 1 else c_i := 0;          ▷ codes computation
9:     end for
10:    if c = [0000]^T or c = [1111]^T then EXIT          ▷ line does not intersect
11:    if c_A ≠ 0 and c_B ≠ 0 then                        ▷ two intersection points
12:        x_A := p ∧ e_i; x_B := p ∧ e_j;
13:        output (x_A, x_B); EXIT
14:    i := TAB1[c];   j := TAB2[c];                ▷ only one end-point is inside
15:                                                    ▷ end-points handling
16:    if c_A = 0 then                                  ▷ point x_A is inside
17:        if (c_B land MASK[c]) ≠ 0 then
18:            x_B := p ∧ e_i;                          ▷ new position of x_B
19:        else
20:            x_B := p ∧ e_j;
21:    else                                             ▷ point x_B is inside
22:        if (c_A land MASK[c]) ≠ 0 then               ▷ new position of x_A
23:            x_A := p ∧ e_i;
24:        else
25:            x_A := p ∧ e_j;
26:    end if
27:    output (x_A, x_B);
28: end procedure
```

5.2. Optimized Line Segment Clipping S-LS-Clip-Opt

For clipping line segments, the line segment S-L-Clip algorithm (see Algorithm 2) is to be modified to take into account positions of the end-points of the given line segment using the **MASK** part of Table 1. The modification uses the end-points codes to determine the case, how the line segment intersects the window. However, computation of the line segment intersection points with the window is needed and the **MASK** determines the appropriate end-point, which is to be replaced by the computed intersection point. It can be seen, that the modification is quite simple, see Algorithm 4.

If the end-points of the line segments are given in homogeneous coordinates, i.e. $w \neq 1$ and $w > 0$, the algorithm Algorithm 3 needs a simple modification, i.e. the conditions are to be changed to $x < x_{\min} * w$, $y < y_{\min} * w$ and similarly for all other cases. It should be noted that in the NDC coordinate system, the conditions are even more simplified, see Algorithm 5.

Algorithm 5 End-point code computation for NDC coordinate system

```
1: procedure CODE (c, x);     ▷ code c for the position x = [x, y : w]^T   w > 0
2:     c := [0000]^T;                                          ▷ initial setting
3:     if  x < 0  then  c := [1000]^T              ▷ setting according to x coordinate
4:         if  x > w  then  c := [0100]^T;     ▷ as x_max = 1 w > 0 then w * 1 = w
5:     if  y < 0  then  c := c lor [0001]^T      ▷ setting according to y coordinate
6:         if  y > w  then  c := c lor [0010]^T;
7:                                  ▷ lor represents or operation on all bits
8: end procedure
```

The end-points classification was simplified for the NDC coordinate system above. The algorithm Smart Line Segment Clip (S-LS-Clip), see Algorithm 4, was designed for the general case, when the end-points of the given line and the corners of the window are given in the projective space, i.e. $w \neq 1$ and $w > 0$.

It means, that the S-LS-Clip algorithm can be further optimized for the case, when clipping is done in the NDC coordinate system. After clipping in the NDC coordinates, the *window-viewport* transformation is applied according to the output device resolution. The transformation can be made in homogeneous coordinates as it uses matrix multiplication, therefore the conversions of the line segment end-points are not needed.

In the NDC case, the CODE computation is to be modified, as the line segments end-points might be given in homogeneous coordinates (see Algorithm 5) and the algorithm for the line segment clipping can be simplified as well. Algorithm 6 is optimized line segment clipping algorithm for the case, when the end-points are given in the Euclidean space. If the end-points are given in homogeneous coordinates, the outer product for the line coefficients is to be used, however, it is one-clock instruction on GPU (cross-product).

Algorithm 6 Optimized S-LS-Clip-Opt line clipping algorithm in E^2

procedure S-L-CLIP-OPT($\mathbf{x}_A, \mathbf{x}_B$); ▷ line is given by two points
  CODE ($\mathbf{c}_A, \mathbf{x}_A$); CODE ($\mathbf{c}_B, \mathbf{x}_B$); ▷ **lor** represents **or** operation on all bits
  if ($\mathbf{c}_A$ **lor** $\mathbf{c}_B$) $= [0000]^T$ **then** ▷ code for the end-points $\mathbf{x}_A$ and $\mathbf{x}_B$
    output ($\mathbf{x}_A$, $\mathbf{x}_B$); **EXIT** ▷ the whole segment is inside
  if ($\mathbf{c}_A$ **land** $\mathbf{c}_B$) $\neq [0000]^T$ **then EXIT** ▷ the whole segment is outside
  $a = y_1 - y_2$; $b = x_2 - x_1$; ▷ line coefficients computation
  $c = x_1 * y_2 - x_2 * y_1$; ▷ if $w \neq 1$ use $[a, b : c]^T := \mathbf{x}_A \wedge \mathbf{x}_B$;
  $c_0 := sign(c)$; $c_1 := sign(a + c)$; ▷ corner's codes computation
  $c_2 := sign(a + b + c)$; $c_3 := sign(b + c)$; ▷ $\mathbf{c} = [c_3, c_2, c_1, c_0]^T$
  if $\mathbf{c} = [0000]^T$ **or** $\mathbf{c} = [1111]^T$ **then EXIT**; ▷ no intersection
  ▷ line segment intersects the window
  $i := TAB1[\mathbf{c}]$; ▷ $\mathbf{x}_A := [x_A, y_A, w_A]^T$
  switch i **do** ▷ equivalent of $\mathbf{x}_A := \mathbf{p} \wedge \mathbf{e}_i$;
    case 0: $x_A := -c$; $y_A := 0$; $w_A := a$;
    case 1: $x_A := -b$; $y_A := a + c$; $w_A := b$;
    case 2: $x_A := -b - c$; $y_A := -a$; $w_A := a$;
    case 3: $x_A := 0$; $y_A := c$; $w_A := -b$;
    default: ERROR ▷ actually the N/A case
  end switch
  $j := TAB2[\mathbf{c}]$; ▷ $\mathbf{x}_B := [x_B, y_B, w_B]^T$
  switch j **do** ▷ equivalent of $\mathbf{x}_B := \mathbf{p} \wedge \mathbf{e}_j$;
    case 0: $x_B := -c$; $y_B := 0$; $w_B := a$;
    case 1: $x_B := -b$; $y_B := a + c$; $w_B := b$;
    case 2: $x_B := -b - c$; $y_B := -a$; $w_B := a$;
    case 3: $x_B := 0$; $y_B := c$; $w_B := -b$;
    default: ERROR ▷ actually the N/A case
  end switch
  ▷ evaluation of the end-points $\mathbf{x}_i = [x_A, y_i : w_i]^T, i = 1, 2$
  if $\mathbf{c}_A = \mathbf{0}$ **then** ▷ point $\mathbf{x}_A$ is inside
    if ($\mathbf{c}_B$ **land** $MASK[\mathbf{c}]$) $\neq 0$ **then**
      $\mathbf{x}_B := \mathbf{x}_A$; ▷ new position of $\mathbf{x}_B$
    else
      $\mathbf{x}_B := \mathbf{x}_B$;
  else ▷ point $\mathbf{x}_B$ is inside
    if ($\mathbf{c}_A$ **land** $MASK[\mathbf{c}]$) $\neq 0$ **then** ▷ new position of $\mathbf{x}_A$
      $\mathbf{x}_A := \mathbf{x}_A$;
    else
      $\mathbf{x}_A := \mathbf{x}_B$;
  end if
  output ($\mathbf{x}_A, \mathbf{x}_B$);
end procedure

The algorithm S-L-Clip (see Algorithm 1) and S-LS-Clip (see Algorithm 4) can be easily modified for the line and line segment clipping by a convex polygon, as Table 1 can be generated synthetically for the given number of the convex polygon vertices and the cycle **for** and codes **c** computation must be modified accordingly.

6. Conclusion

Algorithms for the line clipping and line segment clipping by a rectangular window have been deeply studied for a long time and many algorithms and their modifications have been described.

This contribution describes a new line and line segment clipping algorithms with their optimization using principles of geometric algebra extended for the projective extension of the Euclidean space. The algorithms process line and line segments given by end-points in the homogeneous coordinates and use the outer product applied in the projective space. Also, a simple geometric product computation using tensor multiplication is presented.

It should be noted, that the S-L-Clip and S-LS-Clip algorithms can be easily modified for the line and line segment clipping by a convex polygon.

Acknowledgements The author would like to thank to colleagues and students at the University of West Bohemia (Czech Republic), Shandong University and Zhejiang University (China) for their critical comments and constructive suggestions, and to anonymous reviewers for their valuable comments and hints provided.

References

[1] R. Andreev, E. Sofianska: *New algorithm for two-dimensional line clipping*, Computers and Graphics 15.4 (1991), pp. 519–526, ISSN: 00978493, DOI: 10.1016/0097-8493(91)90051-I.

[2] D. Bui, V. Skala: *Fast algorithms for clipping lines and line segments in E2*, Visual Computer 14.1 (1998), pp. 31–37, DOI: 10.1007/s003710050121.

[3] M. Cyrus, J. Beck: *Generalized two- and three-dimensional clipping*, Computers and Graphics 3.1 (1978), pp. 23–28, DOI: 10.1016/0097-8493(78)90021-3.

[4] J. Day: *A New Two Dimensional Line Clipping Algorithm for Small Windows*, Computer Graphics Forum 11.4 (1992), pp. 241–245, ISSN: 01677055, DOI: 10.1111/1467-8659.1140241.

[5] M. Dörr: *A new approach to parametric line clipping*, Computers and Graphics 14.3-4 (1990), pp. 449–464, ISSN: 00978493, DOI: 10.1016/0097-8493(90)90067-8.

[6] L. Dorst, D. Fontijne, S. Mann: *Geometric Algebra for Computer Science (Revised Edition)*, 2009, ISBN: 9780123749420, DOI: 10.1016/B978-0-12-374942-0.X0000-0.

[7] L. J. Dorst L.: *Guide to Geometric Algebra in Practice*, London, Springer, 2011, isbn: 978-0-85729-810-2,
doi: 10.1007/978-0-85729-811-9.

[8] V. Duvanenko, W. Robbins, R. Gyurcsik: *Line-segment clipping revisited*, Dr. Dobb's Journal 21.1 (1996), pp. 107–110, issn: 1044789X.

[9] D. Foley, A. van Dam, S. Feiner, J. Hughes: *Computer graphics: principles and practice*, Boston, MA, USA: Addison-Wesley, 1990, isbn: 0-201-12110-7.

[10] D. Hestenes: *Tutorial on Geometric Calculus*, Advances in Applied Clifford Algebras 24.2 (2014), pp. 257–273, issn: 01887009,
doi: 10.1007/s00006-013-0418-0.

[11] M. Johnson: *Proof by Duality: or the Discovery of New Theorems*, Mathematics Today (1996).

[12] S. Kaijian, J. Edwards, D. Cooper: *An efficient line clipping algorithm*, Computers and Graphics 14.2 (1990), pp. 297–301, issn: 00978493,
doi: 10.1016/0097-8493(90)90041-U.

[13] K. Kanatani: *Understanding Geometric Algebra*, CRC Press, Japan, 2015, isbn: 9780429157127,
doi: 10.1016/B978-0-12-374942-0.X0000-0.

[14] G. Krammer: *A line clipping algorithm and its analysis*, Computer Graphics Forum 11.3 (1992), pp. 253–266, issn: 01677055,
doi: 10.1111/1467-8659.1130253.

[15] E. Lengyel: *Mathematics for 3D Game Programming and Computer Graphics*, Cengage Learning, USA, 2011, isbn: 978-1-4354-5886-4.

[16] Y.-D. Liang, B. Barsky: *A New Concept and Method for Line Clipping*, ACM Transactions on Graphics (TOG) 3.1 (1984), pp. 1–22,
doi: 10.1145/357332.357333.

[17] T. M. Nicholl, D. Lee, R. A. Nicholl: *Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis*, Computer Graphics (ACM) 21.4 (1987), pp. 253–262,
doi: 10.1145/37402.37432.

[18] A. Rappoport: *An efficient algorithm for line and polygon clipping*, The Visual Computer 7.1 (1991), pp. 19–28,
doi: 10.1007/BF01994114.

[19] V. Skala: *A fast algorithm for line clipping by convex polyhedron in E3*, Computers and Graphics (Pergamon) 21.2 (1997), pp. 209–214,
doi: 10.1016/S0097-8493(96)00084-2.

[20] V. Skala: *Algorithm for 2D line clipping*, New Advances in Computer Graphics, NATO ASI (1989), pp. 121–128,
doi: /10.1007/978-4-431-68093-2_7.

[21] V. Skala: *An efficient algorithm for line clipping by convex polygon*, Computers and Graphics 17.4 (1993), pp. 417–421,
doi: 10.1016/0097-8493(93)90030-D.

[22] V. Skala: *Barycentric coordinates computation in homogeneous coordinates*, Computers and Graphics (Pergamon) 32.1 (2008), pp. 120–127,
doi: 10.1016/j.cag.2007.09.007.

[23] V. Skala: *Intersection Computation in Projective Space Using Homogeneous Coordinates*, Int. Journal of Image and Graphics 8.4 (2008), pp. 615–628,
doi: 10.1142/S021946780800326X.

[24] V. Skala: *Length, Area and Volme Computation in Homogeneous Coordinates*, Int. Journal of Image and Graphics 6.4 (2006), pp. 625–639,
doi: 10.1142/S0219467806002422.

[25] V. Skala: *Line clipping in E2 with O(1) processing complexity*, Computers and Graphics (Pergamon) 20.4 (1996), pp. 523–530,
DOI: 10.1016/0097-8493(96)00024-6.

[26] V. Skala: *O(lg N) line clipping algorithm in E2*, Computers and Graphics 18.4 (1994), pp. 517–524,
DOI: 10.1016/0097-8493(94)90064-7.

[27] V. Skala, D. Bui: *Extension of the Nicholls-Lee-Nichols algorithm to three dimensions*, Visual Computer 17.4 (2001), pp. 236–242,
DOI: 10.1007/s003710000094.

[28] V. Skala, M. Smolik: *A New Formulation of Plücker Coordinates Using Projective Representation*, in: 5th Int. Conf. on Mathematics and Computers in Sciences and Industry (MCSI 2018), IEEE, 2018, pp. 52–56,
DOI: 10.1109/MCSI.2018.00020.

[29] M. Sobkow, P. Pospisil, Y.-H. Yang: *A fast two-dimensional line clipping algorithm via line encoding*, Computers and Graphics 11.4 (1987), pp. 459–467, ISSN: 00978493,
DOI: 10.1016/0097-8493(87)90061-6.

[30] J. Vince: *Geometric algebra for computer graphics*, 2008, pp. 1–252, ISBN: 9781846289965,
DOI: 10.1007/978-1-84628-997-2.