Robust Line-Convex Polygon Intersection Computation in E^2 using Projective Space Representation

Vaclav Skala
Dept. of Computer Science and Engineering, Faculty of Applied Sciences
University of West Bohemia, Pilsen, Czech Republic
www.VaclavSkala.eu

Abstract This paper describes modified robust algorithms for a line clipping by a convex polygon in E^2 and a convex polyhedron in E^3. The proposed algorithm is based on the Cyrus-Beck algorithm and uses homogeneous coordinates to increase the robustness of computation. The algorithm enables computation fully in the projective space using the homogeneous coordinates and the line can be given in the projective space, in general. If the result can remain in projective space, no division operation is needed. It supports the use of vector-vector operations, SSE/AVX instructions, and GPU.

Keywords: computer graphics, line convex polygon intersection, line convex polygon clipping, Cyrus-Beck algorithm, homogeneous coordinates, projective space, duality principle, vector-vector operations, GPU computing.

1. Introduction

Algorithms for a line and line segment intersection computation with the convex polygon in E^2 and convex polyhedron in E^3 are key parts of many geometrical packages, CAD and GIS systems, etc. An extensive survey of intersection and clipping algorithms can be found in [31]. Fundamental algorithms have been described in many textbooks, see Appendix.

Due to the apparent simplicity of intersection algorithms, they might fail due to the limited precision of computation of the Floating-Point Arithmetic (IEEE 754) used in today's computers. The Cyrus-Beck (CB) algorithm [5] is well known for solving the intersection problem of a line with a convex polygon in the E^2 case or a polyhedron in the E^3 case.

There are two basic principles in the E^2 case:

- the edges of a convex polygon define lines and intersection computation is based on direct intersection computation of the clipped line with an edge of the convex polygon,
- the given line defines a half-plane, which separates convex polygon vertices [20,24,28], and positions of the polygon vertices are tested.

The first approach is used by the CB algorithm [5], and the second one was used in [17].

In both cases, the convex polygon can be given as a set of oriented edges with all normal vectors pointing inside or outside the convex polygon and computational complexity is $O(N)$. It means, that the consecutive order of edges is not needed, resp. this property is not used within the CB algorithm.

It should be noted, that in the case of E^2 the convex polygon is given by an ordered sequence of vertices, i.e. clockwise or anti-clockwise, and such property leads to algorithms with $O(\lg N)$ [17]. In the E^3 case an ordering of facets is not possible, however in the triangular mesh case, where neighbors of a triangle are known, the algorithm with $O(\sqrt{N})$ was described [18, 19, 25].

1.1. Cyrus-Beck line clipping algorithm in Euclidean space

The CB algorithm [5] in Euclidean space for a line clipping against a convex polygon in E^2 or against a convex polyhedron in E^3 is well known. It is used in many computer graphics systems and related courses due to its simplicity and applicability for the E^3 case.

However, the CB algorithm has some assumptions:

- it was developed for Euclidean space, i.e. the polygon vertices or the points defining the line p need are given generally in the homogeneous coordinates, they have to be converted to the Euclidean space,
- consistent and known normal vectors orientation of edges, resp. facets, i.e. the normal vectors should all be pointing out or inside,
- generally, an unordered set of edges in the E^2 case, resp. facets in the E^3 case is given. In the E^2 case a polygon is usually given as an ordered set of edges with clockwise or anti-clockwise orientation,
- the given line, which is to be clipped, is given in the parametric form or by two points in the case of line segment clipping.

The CB 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

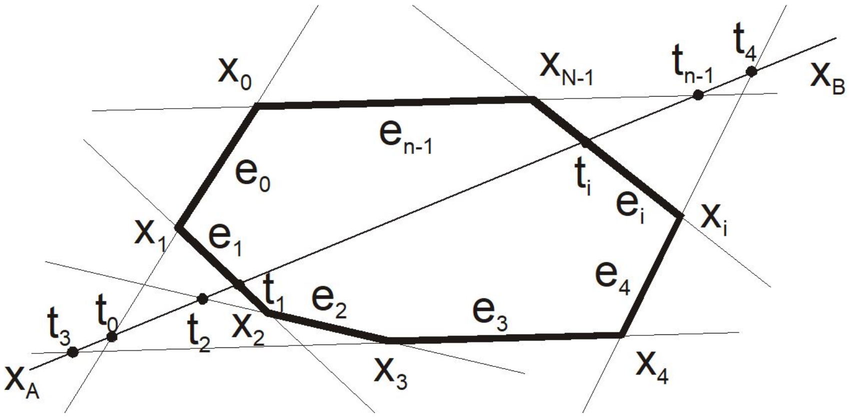

Fig. 1. Cyrus-Beck clipping algorithm against the convex polygon in E^2

implicit form. This leads to a solution of two linear equations (1) (the vector notation is used):

$$
\begin{aligned}
&p: \quad \mathbf{x}(t) = \mathbf{x}_A + \mathbf{s}\, t\,, \quad t \in (-\infty, +\infty)\,,\\
&e_i: \quad \mathbf{n}_i^T \mathbf{x} + c_i = 0\,, i = 0, \ldots, N-1\,,\\
&a_i x + b_i y + c_i = 0\,,
\end{aligned}
\tag{1}
$$

where $\mathbf{x}_A = [x_A, y_A]^T$, $\mathbf{s} = [s_X, s_Y]^T$ is the directional vector of the line p, $\mathbf{n}_i = [n_X, n_Y]^T$ is the "normal" vector[1] of the edge e_i, and c_i is related to the e_i distance from the origin.

Solving (1), the parameter t for the intersection point is obtained as:

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

Then the t_i is the parameter t value for the intersection of the line p and the line on which the edge e_i lies, see Fig. 1.

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

The CB algorithm is of $O(N)$ computational complexity with a fixed $O(N)$ pre-computational cost, as coefficients of lines on which the polygon edges lie need to be pre-computed, see Algorithm 1.

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 $t_i \to \pm\infty$. The fraction computation might cause an overflow or high imprecision of the computed parameter t value, see Fig. 1.

It is hard to detect and solve reliably such cases[2] and programmers usually use a sequence like:

$$\textbf{if } |\mathbf{n}_i^T \mathbf{s}| \;<\; \varepsilon \;\; \textbf{then} \text{ a singular case}\,, \tag{4}$$

which is an incorrect solution as the value ε is a programmer's choice and the value of $|\mathbf{n}_i^T \mathbf{s}|$ might be also close to the value ε (3).

The CB 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.

$$< t_{\min}, t_{\max} > := < t_{\min}, t_{\max} > \cap < 0, 1 > \;. \tag{5}$$

If the final interval of t is empty, i.e. $< t_{\min}, t_{\max} >= \varnothing$ (the case $t_{\min} > t_{\max}$), then the line segment does not have an intersection with the convex polygon.

The known modifications of the CB algorithm use a separation function for more reliable detection of "close to singular" cases were described in [16]:

[1]The "normal" vector is a bivector having different properties from a vector.

[2]However, many textbooks do not point out such dangerous construction as far as the robustness and computational stability are concerned and consider it as a singular case.

Algorithm 1 Cyrus-Beck Line Clipping Algorithm

```
for $i := 0$ to N-1 do                          ▷ computation for the given convex polygon
    Compute $(a_i, b_i : c_i)$                    ▷ $[a_i, b_i : c_i]^T = [\mathbf{n}_i^T : c_i]^T$ for all polygon edges
end for

procedure CB-Clip($\mathbf{x}_A, \mathbf{x}_B$);      ▷ line is given by two points $\mathbf{x}_A, \mathbf{x}_B \in E^2$
    $t_{\min} := -\infty$; $t_{\max} := \infty$;        ▷ set initial conditions for the parameter $t$
    $\mathbf{s} := \mathbf{x}_B - \mathbf{x}_A$;                   ▷ directional vector of the line
    for $i := 0$ to $N - 1$ do                      ▷ for each edge
        $q := \mathbf{n}_i^T \mathbf{s}$;
        if abs($q$) $< \varepsilon$ then
            NOP;                                ▷ Singular or close to singular case
        else
            $t = -(\mathbf{n}_i^T \mathbf{x}_A + c_i)/\mathbf{n}_i^T \mathbf{s}$;
            if $q < 0$ then $t_{\min} := \max\{t, t_{\min}\}$;
            else $t_{\max} := \min\{t, t_{\max}\}$;
            end if
        end if
    end for                                     ▷ all convex polygon edges processed
    if $t_{\min} < t_{\max}$ then                     ▷ intersection of a line and the polygon exists
        { $\mathbf{x}_B := \mathbf{x}_A + \mathbf{s}\ t_{\max}$;   $\mathbf{x}_A := \mathbf{x}_A + \mathbf{s}\ t_{\min}$; }
                                                ▷ if $t_{\min} > t_{\max}$ – NO intersection case
    end if
end procedure
```

- a separation implicit function $F_i(\mathbf{x})$ defined as
  $F_i(\mathbf{x}) = \mathbf{n}_i^T \mathbf{x} + c_i = a_i x + b_i y + c_i$ for the i^{th} edge [20],

- the parametric form of the given line
  $\mathbf{x}(t) = \mathbf{x}_A + (\mathbf{x}_B - \mathbf{x}_A)\ t$
  for intersection computation with found edges intersected (3).

It can be seen that the CB algorithm is of $O(N)$ complexity and the division operation, which is the most consuming time operation in the floating point representation, is used N times. Also, the division operations cause imprecision and lead to robustness issues.[3]

[3]There is a possibility to postpone division operations if the homogeneous coordinates are used, but comparison operations must be modified appropriately [28, 30].

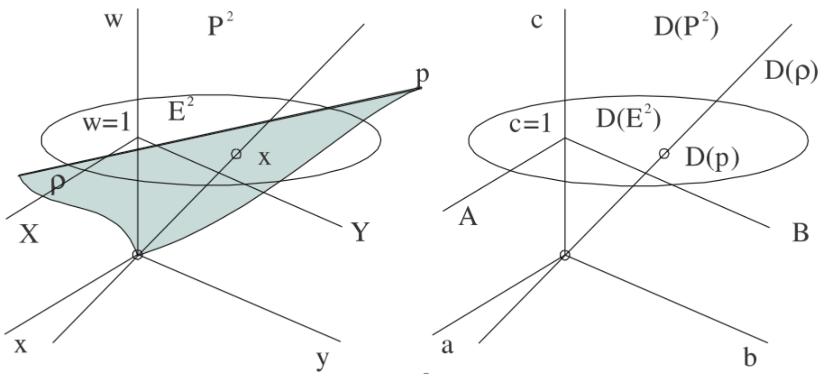

Fig. 2. Projective extension and dual space

2. Projective space

The projective extension of Euclidean space is not a part of standard computer science courses. However, homogeneous coordinates are used in computer graphics and computer vision algorithms, as they enable to represent geometric transformations like translation and rotation by matrix multiplication and also offer to represent a point in infinity.

The mutual conversion between the Euclidean space and projective space in the case of the E^2 space:

$$X = \frac{x}{w}\,, \qquad Y = \frac{y}{w}\,, \qquad w \neq 0\,, \tag{6}$$

where $\mathbf{X} = (X, Y)$, resp. $\mathbf{x} = [x, y : w]^T$ are coordinates in the Euclidean space E^2, resp. in the projective space P^2. The extension to the E^3 is straightforward.

The geometrical interpretation of the Euclidean and the projective spaces is presented in Fig. 2. It should be noted, that a distance of a point $\mathbf{X} = (X, Y)$, i.e. $\mathbf{x} = [x, y : w]^T$, from a line p in the E^2 is defined as:

$$\text{dist} = \frac{aX + bY + c}{\sqrt{a^2 + b^2}} = \frac{ax + by + cw}{w\sqrt{a^2 + b^2}}\,, \quad \mathbf{p} = [a, b : c]^T\,, \tag{7}$$

where $\mathbf{n} = (a, b)$ is the normal vector (actually it is a bivector) of the line p and c is related to the orthogonal distance of the line p from the origin. In the E^3 case:

$$\text{dist} = \frac{aX + bY + cZ + d}{\sqrt{a^2 + b^2 + c^2}} = \frac{ax + by + cz + dw}{w\sqrt{a^2 + b^2 + c^2}}\,, \quad \mathbf{p} = [a, b, c : d]^T\,, \tag{8}$$

where $\mathbf{n} = (a, b, c)$ is the normal vector (actually it is a bivector) of a plane ρ and d is related to the orthogonal distance of a plane ρ from the origin.

2.1. Principle of duality

A line p given by two points $\mathbf{x}_A = [x_A, y_A : w_A]^T$,
$\mathbf{x}_B = [x_B, y_B : w_B]^T$ is given using the outer product as:[4]

$$\mathbf{p} = \mathbf{x}_A \wedge \mathbf{x}_B = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_A & y_A & w_A \\ x_B & y_B & w_B \end{vmatrix} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ X_A & Y_A & 1 \\ X_B & Y_B & 1 \end{vmatrix} , \tag{9}$$
$$[y_A w_B - y_B w_A, -(x_A w_B - x_B w_A) : x_A y_B - x_B y_A]^T = [a, b : c]^T ,$$

where $w_A > 0$, $w_B > 0$, $\mathbf{p} = [a, b : c]^T$ are coefficients of the line p and $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ are the orthonormal basis vectors of the projective space [10].[5]

The projective extension of the Euclidean space enables to use the principle of duality for the intersection of two lines p_1 and p_2 in E^2 using the outer product:

$$\mathbf{x} = \mathbf{p}_1 \wedge \mathbf{p}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} = \tag{10}$$
$$[b_1 c_2 - b_2 c_1, -(a_1 c_2 - a_2 c_1) : a_1 b_2 - a_2 b_1]^T = [x, y : w]^T .$$

It is because lines and points are dual primitives in the P^2 projective extension [3, 9, 21, 22, 23, 26, 27, 32, 33].

The outer-product $\mathbf{x}_A \wedge \mathbf{x}_B$ is formally equivalent to the cross-product $\mathbf{x}_A \times \mathbf{x}_B$ in the P^2 projective extension case and the non-normalized normal vector of the line p is $\mathbf{n} = [a, b : 0]^T$.

In the E^3 case, the dual primitives are points and planes, i.e.

$$\mathbf{x} = \rho_1 \wedge \rho_2 \wedge \rho_3 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{vmatrix} = [x, y, z : w]^T ,$$
$$\rho = \mathbf{x}_A \wedge \mathbf{x}_B \wedge \mathbf{x}_C = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ x_A & y_A & z_A & w_A \\ x_B & y_B & z_B & w_B \\ x_C & y_C & z_C & w_C \end{vmatrix} = [a, b, c : d]^T . \tag{11}$$

It should be noted, that the non-normalized directional vector $\mathbf{s}$ of the line p in E^2 is orthogonal to the normal bivector of the line and it is given as:

$$\mathbf{s} = (X_B - X_A, Y_B - Y_A) = (s_X, s_Y) = (-b, a) , \quad \mathbf{n} = (a, b) . \tag{12}$$

[4] In this case, the outer product is formally equivalent to the cross product

[5] There is a direct connection with the *geometric product* which is defined as $\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$, i.e. $\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \times \mathbf{b}$ [10].

The line p splits the E^2 plane into two half-planes:

$$F_p(\mathbf{x}) = 0\,, \quad F_p(\mathbf{x}) = \mathbf{p} \cdot \mathbf{x} = \mathbf{p}^T\mathbf{x} = ax + by + cw\,, \tag{13}$$

where $w > 0$. If $w \to \pm\infty$ then the point $\mathbf{x}$ is closed or in infinity. It should be noted that the dot-product (scalar product) is a single instruction on GPU.

3. Proposed clipping algorithm in projective space

The CB algorithm computes the parameter t value using division operation N times. However, only two values are needed, if any. It means, that $N - 2$ computations of the parameter t are unnecessary. Also, reliable detection of "singular or close to singular" cases is difficult and time-consuming.

Let us consider the case, when the convex polygon vertices and points defining the line p are given in projective space, i.e. in homogeneous coordinates with $w \neq 0$. In this case, a conversion of $\mathbf{x}_A, \mathbf{x}_B$ and $\mathbf{x}_i$ to Euclidean space using a division operation is needed, if the CB algorithm is to be used. It means, that $2N + 4$ division operations would be needed for the conversion to Euclidean space.

Let us consider the case, when the polygon vertices are generally given in the projective space (6), i.e. using the homogeneous coordinates, as:

$$\mathbf{x}_i = [x_i, y_i : w_i]^T\,, \quad i = 0, \ldots, N-1 \quad \text{and} \quad w \neq 0\,, \tag{14}$$

where w_i are is homogeneous coordinate and points $\mathbf{x}_A$ and $\mathbf{x}_B$ define the line p, resp. line segment to be clipped.

$$\mathbf{x}_A = [x_A, y_A : w_A]^T\,, \quad \mathbf{x}_B = [x_B, y_B : w_B]^T\,. \tag{15}$$

Then the given line p, given by the points $\mathbf{x}_i$ and $\mathbf{x}_{i\oplus 1}$, intersects the edge e_i **if and only if**:

$$F_p(\mathbf{x}_i) > 0 \ \mathbf{xor}\ F_p(\mathbf{x}_{i\oplus 1}) > 0\,, \quad i = 0, \ldots, N-1\,, \tag{16}$$

i.e. the points $\mathbf{x}_i$ and $\mathbf{x}_{i\oplus 1}$ are on the opposite sides of the line p[6].

In the case of the line convex polygon intersection, two intersected edges are detected, if any. It means, that $N - 2$ division operations are saved and no division operation is needed to find out if the convex polygon edge is intersected by the line p.

In the case when the line p intersects the convex polygon edges e_k and e_l the intersection points can be determined as:

- direct intersection computation using the homogeneous coordinates of points (10),
- using the parametric form of the line p (1), but modified for the projective space.

In both cases, all computations support *vector-vector* operations, and therefore they are convenient for GPU or SSE instruction use.

[6]The operator $\oplus$ means addition modulo N, i.e. $a \oplus b = (a + b)$ **mod** N.

3.1. Direct intersection coordinates computation

The direct computation of the intersection points is quite simple, as the intersected edges e_k and e_l have been determined in the previous step (16). Using the outer-product (10) (in this case the cross-product) the points of intersections are given as, if any:

$$\begin{gathered} \mathbf{x}_A = \mathbf{p} \wedge \mathbf{e}_k \equiv \mathbf{p} \times \mathbf{e}_k \,, \quad \mathbf{x}_B = \mathbf{p} \wedge \mathbf{e}_l \equiv \mathbf{p} \times \mathbf{e}_l \,, \\ \mathbf{e}_k = \mathbf{x}_k \wedge \mathbf{x}_{k\oplus 1} = [a_k, b_k : c_k]^T \,, \quad \mathbf{e}_l = \mathbf{x}_l \wedge \mathbf{x}_{l\oplus 1} = [a_l, b_l : c_l]^T \,, \end{gathered} \tag{17}$$

where $\mathbf{x}_A = [x_A, y_A : w_A]^T$, $\mathbf{x}_B = [x_B, y_B : w_B]^T$ and $\oplus$ means mod N operation.[7]

It should be noted that the dot product and cross products are single instructions on GPU [32].

In the case of a line segment clipping, some additional logical conditions are to be used to keep the line segment orientation and respect to situations, when an end-point of the line segment is already inside the given convex polygon (5) [28, 29, 30].

3.2. Intersection using parametric form

Using the parametric form is similar to the CB algorithm. It is simple, but the projective representation has to be respected.

There are two possibilities:

- linear interpolation conversion from Euclidean space to the projective space, which leads to the linear parameterization of the parameter t;
- linear parameterization directly in the projective space, which leads to the non-linear monotonic parameterization of the parameter.

The linear parameterization directly in the projective space has significant advantages is this case, as it is simple and robust. The line π given by two points $\mathbf{x}_A = [x_A, y_A : w_A]^T$, $\mathbf{x}_B = [x_B, y_B : w_B]^T$ and $\mathbf{p} = [a, b : c]^T$ are coefficients of the line p, see Fig. 3.

In the projective space P^2, the line π is given by two points $\mathbf{x}_A = [x_A, y_A : w_A]^T$, $\mathbf{x}_B = [x_B, y_B : w_B]^T$. It forms with the origin 0_P a plane $\rho :\ ax + by + cw = 0$, where the point $[0, 0 : 0]^T$ represents a point in infinity.

$$\begin{gathered} \mathbf{x}(\tau) = \mathbf{x}_A + (\mathbf{x}_B - \mathbf{x}_A)\,\tau \,, \quad \tau \in (-\infty, +\infty) \,, \\ x(\tau) = x_A + (x_B - x_A)\,\tau \,, \quad y(\tau) = y_A + (y_B - y_A)\,\tau \,, \quad w(\tau) = w_A + (w_B - w_A)\,\tau \,. \end{gathered} \tag{18}$$

The interpolation (18) is linear but when results are converted to Euclidean space, the parameterization is non-linear but **monotonic**.

The line p in E^2 is given by two points $\mathbf{X}_A = (X_A, Y_A)$ and $\mathbf{X}_B = (X_B, Y_B)$ as:

$$\mathbf{X}(t) = \mathbf{X}_A + (\mathbf{X}_B - \mathbf{X}_A)\,t \,, \quad t \in (-\infty, +\infty) \,. \tag{19}$$

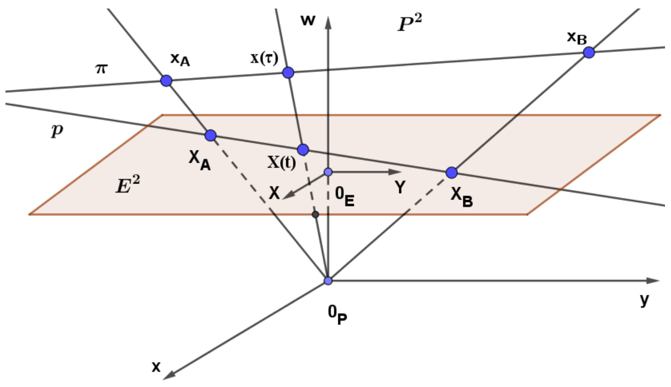

Fig. 3. Parameterization of a line in E^2 and P^2

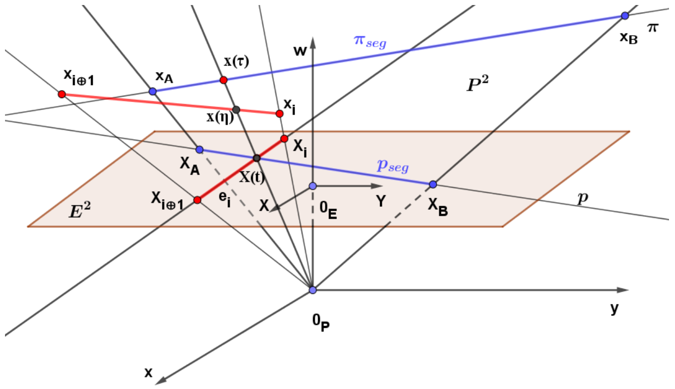

Fig. 4. Intersection of the line π and an edge e_i

It should be noted that $t \neq \tau$, except for $t = \tau = 0$ and $t = \tau = 1$.

In the case of the projective interpolation, see Fig. 4, the system of equations (20) is to be solved:

$$\begin{gathered} e_i : \mathbf{p}^T \mathbf{x} = 0\,, \quad ax + by + cw = 0\,, \\ \pi : \mathbf{x}(\tau) = \mathbf{x}_A + (\mathbf{x}_B - \mathbf{x}_A)\,\tau\,, \quad \tau \in (-\infty, +\infty)\,. \end{gathered} \tag{20}$$

[7]It should be noted, that instead of using **mod** operation, one "virtual" vertex is added so that $\mathbf{x}_N \equiv \mathbf{x}_0$. This enables us to avoid **mod** operation, which is computationally expensive.

It means, that also homogeneous coordinate w is parametrized (21).

$$\begin{aligned} x(t) &= x_A + (x_B - x_A)\,\tau = x_A + s_x\,\tau\,, \\ y(t) &= y_A + (y_B - y_A)\,\tau = y_A + s_y\,\tau\,, \\ w(t) &= w_a + (w_B - w_A)\,\tau = w_a + s_w\,\tau\,. \end{aligned} \tag{21}$$

This leads to:

$$\begin{aligned} \mathbf{p}^T\mathbf{x}(\tau) = 0\,, \quad & \mathbf{p}^T\mathbf{x}_A + \mathbf{p}^T(\mathbf{x}_B - \mathbf{x}_A)\,\tau\,, \\ \tau = -\frac{\mathbf{p}^T\mathbf{x}_A}{\mathbf{p}^T\mathbf{s}} & \triangleq [\mathbf{p}^T\mathbf{x}_A : \mathbf{p}^T\mathbf{s}]^T = [t : t_w]^T\,, \end{aligned} \tag{22}$$

where $\mathbf{s} = \mathbf{x}_B - \mathbf{x}_A = [s_x, sy : s_w]^T$ and τ value can be expressed as a projective scalar value in the projective form as $\tau = [t : t_w]^T$.

Now, the τ values of intersections for the intersected edges are determined. It should be noted, that the τ interpolation is monotonic, but in Euclidean space (X, Y) the interpolation is linear with non-linear parameterization.

It can be seen that division operations are not needed if the result can remain in the projective notation, i.e. no conversion to Euclidean space is required.

The proposed algorithm, with two possible modifications, described above is simple, easy to implement, and convenient for vector-vector implementation. It is based on the projective extension of Euclidean space.

Contrary to the CB algorithm, the algorithm does not require computation of the edges' implicit form, as it uses a separation function.

The first modification is based on the outer product application's implicit representation. The second one is based on the parametric form of the clipped line p, which is more convenient for line segment clipping cases.
It can be seen that in the projective case

- $2N$ division operations are eliminated, if the polygon vertices are given in the projective space, i.e. $w \neq 1$, as the transformations of x_i, y_i to Euclidean space are not needed,
- there is no need to compute edges' coefficients, i.e. a_i, b_i, c_i ,
- $N-2$ division operations are saved during the run-time and if the intersection points can remain in the projective representation, *no division operation is needed* at all.

It leads to significant improvement in robustness and with additional speed-up as vector-vector operations can be used.

4. Conclusion

This contribution presents a new fully projective algorithm, based on the Cyrus-Beck algorithm, for a line and line segment clipping by a convex polygon using the *vector-vector* operations and supporting GPU implementation, resp. SSE/AVX instructions.

The presented approach eliminates $2N$ division operations in preprocessing of the polygon edges if the polygon vertices are given in the projective space and $N-2$ division operations in the run-time. It also increases the numerical robustness especially in cases, when the given line is parallel or close to parallel to an edge of the convex polygon.

If the computed results can remain in the projective space, i.e. the conversion to the Euclidean space is not needed, *no division operation is required* by the proposed algorithm. The proposed algorithm can be extended to the E^3 case, i.e. line-convex polyhedron intersection if intersections' points are computed using the parametric form of the given line; however, instead of the separation line two orthogonal planes, which define the clipped line p have to be used, similarly as in [19]. For a deeper study of intersection algorithms, a reader is advised to read "A brief survey of intersection and clipping algorithms" [31].

In future work, the proposed algorithm is to be analyzed from the E^3 case perspective, i.e. line and line segment clipping by a convex polyhedron using the Plücker coordinates [11, 27, 34].

Acknowledgment

The author thanks recent students and colleagues at the University of West Bohemia, Pilsen, Zhejiang University, Hangzhou and Shandong University, Jinan in China for their recent suggestions and constructive comments.

Thanks also belong to the anonymous reviewers, as their comments and hints helped to improve this paper and to several authors of recently published relevant papers [31] for sharing their views and hints provided.

Appendix

The following relevant books are recommended to a reader:

- Salomon, D.: The Computer Graphics Manual [13],
- Salomon, D.: Computer Graphics and Geometric Modeling [12],
- Agoston, M. K.: Computer Graphics and Geometric Modelling: Mathematics [2],
- Agoston, M. K.: Computer Graphics and Geometric Modelling: Implementation & Algorithms [1],
- Lengyel, E.: Mathematics for 3D Game Programming and Computer Graphics [10],
- Foley, J. D., van Dam, A., Feiner, S., Hughes, J. F.: Computer graphics – Principles and Practice [7],

- Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S. K., Akeley, K.: Computer Graphics – Principles and Practice [8],
- Ferguson, R. S.: Practical Algorithms for 3D Computer Graphics [6],
- Shirley, P., Marschner, S.: Fundamentals of Computer Graphics [15],
- Theoharis, T., Platis, N., Papaioannou, G., Patrikalakis, N.: Graphics and Visualization: Principles & Algorithms [35],
- Comninos, P.: Mathematical and Computer Programming Techniques for Computer Graphics [4],
- Schneider, P. J., Eberly, D. H.: Geometric Tools for Computer Graphics [14].

References

[1] M. K. Agoston. *Computer Graphics and Geometric Modelling: Implementation & Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2004. doi:10.1007/b138805.

[2] M. K. Agoston. *Computer Graphics and Geometric Modelling: Mathematics*. Springer-Verlag, Berlin, Heidelberg, 2005. doi:10.1007/b138899.

[3] A. Arokiasamy. Homogeneous coordinates and the principle of duality in two dimensional clipping. *Computers and Graphics*, 13(1):99–100, 1989. doi:10.1016/0097-8493(89)90045-9.

[4] P. Comninos. *Mathematical and Computer Programming Techniques for Computer Graphics*. Springer-Verlag, Berlin, Heidelberg, 2005. doi:10.1007/978-1-84628-292-8.

[5] M. Cyrus and J. Beck. Generalized two- and three-dimensional clipping. *Computers and Graphics*, 3(1):23–28, 1978. doi:10.1016/0097-8493(78)90021-3.

[6] R. S. Ferguson. *Practical Algorithms for 3D Computer Graphics*. A. K. Peters, Ltd., USA, 2nd edn., 2013. doi:10.1201/b16333.

[7] J. D. Foley, A. van Dam, S. Feiner, and J. F. Hughes. *Computer Graphics – Principles and Practice*. Addison-Wesley, 2nd edn., 1990.

[8] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, et al. *Computer Graphics – Principles and Practice*. Addison-Wesley, 3rd edn., 2014.

[9] M. Johnson. Proof by duality: or the discovery of "new" theorems. *Mathematics Today*, December:138–153, 1996.

[10] E. Lengyel. *Mathematics for 3D Game Programming and Computer Graphics*. Course Technology Press, Boston, MA, USA, 3rd edn., 2011.

[11] N. Platis and T. Theoharis. Fast ray-tetrahedron intersection using Plücker coordinates. *Journal of Graphics Tools*, 8(4):37–48, 2003. doi:10.1080/10867651.2003.10487593.

[12] D. Salomon. *Computer Graphics and Geometric Modeling*. Springer-Verlag, Berlin, Heidelberg, 1st edn., 1999.

[13] D. Salomon. *The Computer Graphics Manual*. Springer, 2011. doi:10.1007/978-0-85729-886-7.

[14] P. J. Schneider and D. H. Eberly. *Geometric Tools for Computer Graphics*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco, 2003. doi:10.1016/B978-1-55860-594-7.50025-4.

[15] P. Shirley and S. Marschner. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., USA, 3rd edn., 2009. doi:10.1201/9781439865521.

[16] V. Skala. An efficient algorithm for line clipping by convex polygon. *Computers and Graphics*, 17(4):417–421, 1993. doi:10.1016/0097-8493(93)90030-D.

[17] V. Skala. O(lg N) line clipping algorithm in E2. *Computers and Graphics*, 18(4):517–524, 1994. doi:10.1016/0097-8493(94)90064-7.

[18] V. Skala. An efficient algorithm for line clipping by convex and non-convex polyhedra in E3. *Computer Graphics Forum*, 15(1):61–68, 1996. doi:10.1111/1467-8659.1510061.

[19] V. Skala. A fast algorithm for line clipping by convex polyhedron in E3. *Computers and Graphics (Pergamon)*, 21(2):209–214, 1997. doi:10.1016/s0097-8493(96)00084-2.

[20] V. Skala. A new approach to line and line segment clipping in homogeneous coordinates. *Visual Computer*, 21(11):905–914, 2005. doi:10.1007/s00371-005-0305-3.

[21] V. Skala. Duality and intersection computation in projective space with GPU support. In: *Latest Trends on Applied Mathematics, Simulation, Modelling – Proc. 4th International Conference on Applied Mathematics, Simulation, Modelling (ASM'10)*, pp. 66–71. Corfu, Greece, 2010. `http://hdl.handle.net/11025/11797`.

[22] V. Skala. Duality, barycentric coordinates and intersection computation in projective space with GPU support. *WSEAS Transactions on Mathematics*, 9(6):407–416, 2010. `http://afrodita.zcu.cz/~skala/PUBL/PUBL_2010/2010_NAUN-journal.pdf`.

[23] V. Skala. Geometry, duality and robust computation in engineering. *WSEAS Transactions on Computers*, 11(9):275–293, 2012.

[24] V. Skala. S-clip E2: A new concept of clipping algorithms. *SIGGRAPH Asia Posters, SA*, pp. 1–2, 2012. doi:10.1145/2407156.2407200.

[25] V. Skala. Algorithms for line and plane intersection with a convex polyhedron with O(sqrt(N)) expected complexity in E3. In: *SIGGRAPH Asia 2014 Posters*, SA '14. Association for Computing Machinery, New York, NY, USA, 2014. doi:10.1145/2668975.2668976.

[26] V. Skala. Geometric transformations and duality for virtual reality and haptic systems. *Communications in Computer and Information Science*, 434 PART I:642–647, 2014. doi:10.1007/978-3-319-07857-1_113.

[27] V. Skala. Projective geometry, duality and plücker coordinates for geometric computations with determinants on GPUs. *ICNAAM 2017*, 1863, 2017. doi:10.1063/1.4992684.

[28] V. Skala. Optimized line and line segment clipping in E2 and geometric algebra. *Ann. Math. Inf.*, 52:199–215, 2020. doi:10.33039/ami.2020.05.001.

[29] V. Skala. A new coding scheme for line segment clipping in E2. *Lecture Notes in Computer Science*, LNCS-accepted for publication ICCSA 2021:16–29, 2021. doi:10.1007/978-3-030-86976-2_2.

[30] V. Skala. A novel line convex polygon clipping algorithm in E2 with parallel processing modification. *Lecture Notes in Computer Science*, LNCS 12953 ICCSA 2021:3–15, 2021. doi:10.1007/978-3-030-86976-2_1.

[31] V. Skala. A brief survey of clipping and intersection algorithms with a list of references (including triangle-triangle intersections). *Informatica (Lithuania)*, 34(1):169–198, 2023. doi:10.15388/23-INFOR508.

[32] V. Skala, S. A. A. Karim, and E. A. Kadir. Scientific computing and computer graphics with GPU: Application of projective geometry and principle of duality. *International Journal of Mathematics and Computer Science*, 15(3):769–777, 2020. `http://ijmcs.future-in-tech.net/15.3/R-Skala-AbdulKarim.pdf`.

[33] V. Skala and M. Kuchař. The hash function and the principle of duality. In: *Proc. Computer Graphics International Conference (CGI'01)*, pp. 167–174. Hong Kong, China, 2001. doi:10.1109/CGI.2001.934671.

[34] V. Skala and M. Smolik. A new formulation of Plücker coordinates using projective representation. In: *Proc. 2018 5th International Conference on Mathematics and Computers in Sciences and Industry (MCSI 2018)*, pp. 52–56. Corfu, Greece, 2018. doi:10.1109/MCSI.2018.00020.

[35] T. Theoharis, N. Platis, G. Papaioannou, and N. Patrikalakis. *Graphics and Visualization: Principles & Algorithms (1st ed.)*. A K Peters/CRC Press, 2008. doi:10.1201/b10676.

Vaclav Skala is a professor at the Dept. of Computer Science and Engineering, University of West Bohemia (UWB), Pilsen [Plzen]. He has been with Brunel University in London, U.K., Gavle University, Sweden, Moscow Power Engineering Institute, Russia, etc. His current research is targeted at fundamental algorithms for computer graphics, geometric algebra, meshless (meshfree) methods for scalar and vector fields interpolation and approximation, and applied mathematics. He is the Editor-in-Chief of the Journal of WSCG and Computer Science Research Notes [CSRN].
Vaclav Skala has published over 160+ research-indexed papers with more than 800+(WoS/Publons), 1300+(Scopus), and 2600+(Scholar Google) citations.