Visual Comput (2005) 21: 905–914
DOI 10.1007/s00371-005-0305-3

ORIGINAL ARTICLE

Vaclav Skala

A new approach to line and line segment clipping in homogeneous coordinates

Published online: 14 September 2005
© Springer-Verlag 2005

V. Skala
University of West Bohemia, Department of Computer Science and Engineering, Plzeň, Czech Republic
skala@kiv.zcu.cz

Abstract The clipping operation is still the bottleneck of the graphics pipeline in spite of the latest developments in graphical hardware and a significant increase in performance. Algorithms for line and line segment clipping have been studied for a long time and many research papers have been published so far. This paper presents a new robust approach to line and line segment clipping using a rectangular window. A simple extension for the case of convex polygon clipping is presented as well. The presented approach does not require a division operation and uses homogeneous coordinates for input and output point representation. The proposed algorithms can take advantage of operations supported by vector–vector hardware.
The main contribution of this paper is a new approach to intersection computations applied to line and line segment clipping. This approach leads to algorithms that are simpler, robust, and easy to implement.

Keywords Clipping · Homogeneous coordinates · Projective space · Duality · Computer graphics

Notation

N – number of edges of the given polygon
$\boldsymbol{X} = (X, Y)$ – a point in Euclidean coordinates
X – coordinate in Euclidean coordinates
$\boldsymbol{x} = [x, y, w]^{\mathrm{T}}$ – point in projective space represented by homogeneous coordinates
x – coordinate in homogeneous coordinates
$\boldsymbol{x}_A, \boldsymbol{x}_B$ – end-points of the line segment or points that define a line p
$\boldsymbol{x}_i$ – ith vertex of the given polygon
p – line to be clipped or a line segment on which the clipped line segment lies
p_i – line on which the ith edge of the given polygon lies
$\boldsymbol{p} = [a, b, c]^{\mathrm{T}}$ – vector of a line p defined as $ax + by + c = 0$
$\boldsymbol{e}_i$ – vector of a line on which the ith edge of the given polygon lies; $\boldsymbol{e}_i = [a_i, b_i, c_i]^{\mathrm{T}}$
$\boldsymbol{s}_i$ – directional vector of the ith edge of the given polygon
$\boldsymbol{n}_i$ – normal vector of the ith edge of the given polygon – a vector perpendicular to the ith edge
$\boldsymbol{s}$ – directional vector of the given line p, i.e., $\boldsymbol{s} = \boldsymbol{x}_B - \boldsymbol{x}_A$
ρ – a plane
$D(p)$ – dual representation of p
$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$ – dot product of two vectors $\boldsymbol{a}, \boldsymbol{b}$
$\boldsymbol{a} \times \boldsymbol{b}$ – cross product of two vectors $\boldsymbol{a}, \boldsymbol{b}$
$\boldsymbol{i} = [1, 0, 0]^{\mathrm{T}}$, $\boldsymbol{j} = [0, 1, 0]^{\mathrm{T}}$, $\boldsymbol{k} = [0, 0, 1]^{\mathrm{T}}$
P – polygon, vertices indexed as $0, \ldots, N-1$
land – bitwise **and** logical operation
lor – bitwise **or** logical operation
operations used with indices: $\boldsymbol{x}_{i+1}$ means $\boldsymbol{x}_{(i+1) \mod N}$

1 Introduction

In spite of the latest graphical hardware developments and a significant increase in performance, clipping is still the bottleneck of the graphics pipeline.

There are many algorithms devoted to line and line segment clipping in E^2 and E^3. Generally, algorithms have been developed and modified for line or line segment or polygon clipping against a rectangular window, a convex polygon, or a polygonal clipping area (see [4, 5, 8, 10–13, 15–19, 21]). Some modifications have also been developed for a self-intersecting clipping polygon or for areas with linear or quadric edges [14]. A comparison of several algorithms can be found in [2].

In addition, some algorithms were specially developed or modified for special cases such as small windows or for specific characteristics of the input data set. Some algorithms and their efficiencies are very sensitive to input data, the i.e. geometrical distribution of lines or line segments.

Algorithms used in E^2 are usually based on a Euclidean space representation. Nevertheless, the positive projective space representation using homogeneous coordinates is more convenient in some cases. In many applications, the clipping window or polygon is constant for many clipped lines. In this case, pre-processing might be considered to speed up the algorithm as well.

This paper presents a new and robust approach for line and line segment clipping against a rectangular window with a simple modification for line clipping by a convex polygon. The proposed algorithms are compared with the well-known Cyrus–Beck [4] and Cohen–Sutherland [5] algorithms, as these are commonly used as reference algorithms that enable us to make a mutual-comparison of several algorithms. Theoretical evaluation of the proposed algorithm is presented, and experimental verification and algorithm comparison are presented as well.

2 Projective geometry and duality

Homogeneous coordinates are widely used in computer graphics applications, usually connected with geometric transformations such as rotation, scaling, translation, and projection. In many cases, homogeneous coordinates are seen just as a mathematical tool that enables a simple description of geometric transformations. Nevertheless, there are many "invisible" impacts of algorithm design that can lead to new, fast, and robust algorithms that can also be supported by GPU (Graphics Processing Unit) hardware. Figure 1 presents a geometrical interpretation of the Euclidean and projective spaces.

The point $\boldsymbol{x}$ is defined as a point in E^2 with coordinates $\boldsymbol{X} = (X, Y)$ or as a point with homogeneous coordinates $[x, y, w]^{\mathrm{T}}$, where $w = 1$, usually. The point $\boldsymbol{x}$ is actually a line without an origin in the projective space P^2, and $X = x/w$ and $Y = y/w$. It can be seen that a line $p \in E^2$ is actually a plane ρ without the origin in the projective space P^2, i.e., the Euclidean line p is defined as

$$ax + by + cw = 0, \quad w \neq 0$$

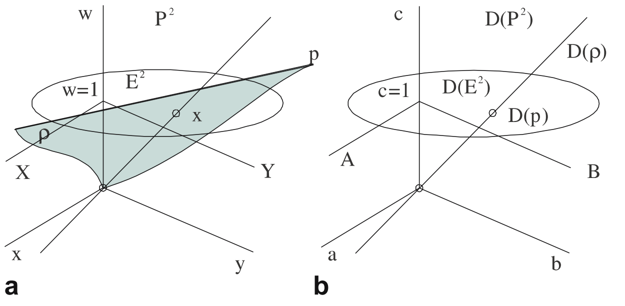

Fig. 1a,b. Euclidean, projective, and dual space representations

Any $\xi \neq 0$ can multiply the equation without any effect to the geometry. In dual representation, the plane ρ can be represented as a line $D(\rho) \in D(P^2)$ or as a point $D(p) \in D(E^2)$, when a projection is made, e.g., for $c = 1$. A complete theory on projective spaces can be found in [6, 20].

On the other hand, there is a principle of duality that is useful when deriving certain formulas. The principle states that any theorem remains true when we interchange the words "point" and "line," "lie on" and "pass through," "join" and "intersect," and so on. Once the theorem has been established, the dual theorem is obtained as described above (see [3, 9]).

In other words, the principle of duality says that in all theorems, it is possible to substitute the term "point" by the term "line" and the term "line" by the term "point," and the given theorem stays valid. This helps tremendously in the solution of some geometrical problems.

Definition 1. *The cross product of two vectors* $\boldsymbol{x}_1 = [x_1, y_1, w_1]^T$ *and* $\boldsymbol{x}_2 = [x_2, y_2, w_2]^T$ *is defined as*

$$\boldsymbol{x}_1 \times \boldsymbol{x}_2 = \det \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ \boldsymbol{x}_1 & \boldsymbol{y}_1 & \boldsymbol{w}_1 \\ \boldsymbol{x}_2 & \boldsymbol{y}_2 & \boldsymbol{w}_2 \end{bmatrix},$$

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

Please note that homogeneous coordinates are used.

Theorem 1. *Let two points* $\boldsymbol{x}_1$ *and* $\boldsymbol{x}_2$ *be given in the projective space. Then, the coefficients of the line p, which is defined by those two points, are the cross product of their homogeneous coordinates,* $\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2$.

Proof. Let the line $p \in \mathrm{E}^2$ be defined as

$$ax + by + cw = 0.$$

Then

$$a = \det \begin{bmatrix} y_1 & w_1 \\ y_2 & w_2 \end{bmatrix} \quad b = -\det \begin{bmatrix} x_1 & w_1 \\ x_2 & w_2 \end{bmatrix}$$

$$c = \det \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}.$$

For $w = 1$, we get the standard cross-product formula, and the cross product defines the line p, i.e.,

$$\boldsymbol{x}_1 \times \boldsymbol{x}_2 = \boldsymbol{p},$$

where $\boldsymbol{p} = [a, b, c]^{\mathrm{T}}$.

Theorem 2. *Let two lines p_1 and p_2 be given in projective space. Then the homogeneous coordinates of the point $\boldsymbol{x}$ at the intersection of those two lines are given by the cross product of their coordinates $\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2$, i.e.,*

$$\boldsymbol{p}_1 \times \boldsymbol{p}_2 = \det \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix},$$

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

Proof. Immediate from Theorem 1 and the duality principle.

These two theorems are very important as they enable us to handle some problems defined in homogeneous coordinates efficiently and make computations quite robust and effective.

3 The Cyrus–Beck algorithm

The Cyrus–Beck (CB) algorithm [4] is the most well-known algorithm for line and line segment clipping by a convex polygon. There are several algorithms for line clipping against the given convex polygon that claimed some advantages over the original CB algorithm (see [1, 12, 13, 16]). Nevertheless, the CB algorithm is very stable, and its performance is nearly independent of such factors as the geometrical distribution of clipped primitives. The algorithm assumes that the clipping polygon is convex, and its vertices $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N-1}$ are ordered in a defined sense, usually counterclockwise.

Let $\boldsymbol{x}_A, \boldsymbol{x}_B$ define the line to be clipped. For each edge $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}$, one computes the parameter t_i along the line of its intersection with the line of the edge $\boldsymbol{x}_i\boldsymbol{x}_{i+1}$ and whether this edge belongs to the leading or tailing part of the polygon relative to the direction $\boldsymbol{x}_B - \boldsymbol{x}_A$ (see Fig. 2).

The maximum $t_{\max}$ of the tailing part and minimum $t_{\min}$ of the leading part delimit the visible part of the segment. If the interval is empty, the segment is invisible. Since the CB algorithm computes the intersection for each edge, its complexity is $O(N)$, where N is the number of edges.

The parameter t of the point where the given line p intersects the line p_i defined by the points $\boldsymbol{x}_i\boldsymbol{x}_{i+1}$ is obtained by solving the following equations:

$$p : \boldsymbol{x}(t) = \boldsymbol{x}_A + \boldsymbol{s}t \quad p_i : \boldsymbol{n}_i^T \boldsymbol{x} + c_i = 0.$$

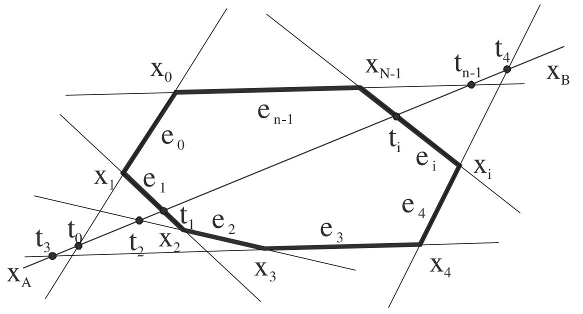

Fig. 2. Intersections computed by the CB algorithm

We get

$$\boldsymbol{n}_i^T \boldsymbol{x}_A + \boldsymbol{n}_i^T \boldsymbol{s}t + c_i = 0.$$

Solving this equation, we get

$$t_i = -\frac{\boldsymbol{n}_i^T \boldsymbol{x}_A + c_i}{\boldsymbol{n}_i^{\mathrm{T}} . \boldsymbol{s}}$$

The principle of the CB algorithm is described by Algorithm 1 (see [5] for details).

Algorithm 1: Principle of the Cyrus–Beck algorithm

```
procedure CB;
{input: $\boldsymbol{x}_A, \boldsymbol{x}_B$}
begin
    $\boldsymbol{s} := \boldsymbol{x}_B - \boldsymbol{x}_A$;
    $t_{\min} := -\infty$; $t_{\max} := \infty$;
    {$t_{\min} := 0$; $t_{\max} := 1$; in the case of a line segment}
    for $i := 0$ to $N - 1$ do
    begin
        $q := \boldsymbol{n}_i^{\mathrm{T}} \boldsymbol{s}$;
        if $q < 0$ then
        begin  {the $i$th edge can be seen from}
               {infinity in the negative direction of $-s$}
            $t := -(\boldsymbol{n}_i^{\mathrm{T}} \boldsymbol{x}_A + c_i)/q$; $t_{\min} := \max(t, t_{\min})$;
        end else
            if $q > 0$ then
            begin  {the $i$th edge can be seen from}
                   {infinity in the direction of $\boldsymbol{s}$}
                $t := -(\boldsymbol{n}_i^{\mathrm{T}} \boldsymbol{x}_A + c_i)/q$; $t_{\max} := \min(t, t_{\max})$;
            end else solve a special case {$\boldsymbol{n}_i^{\mathrm{T}} \boldsymbol{s} = 0$}
    end;
    if $t_{\min} < t_{\max}$ then
    begin $\boldsymbol{x}_A := \boldsymbol{x}_A + \boldsymbol{s}t_{\min}$;
        $\boldsymbol{x}_B := \boldsymbol{x}_A + \boldsymbol{s}t_{\max}$;
        output ($\boldsymbol{x}_A, \boldsymbol{x}_B$)
    end
end {CB}
```

The CB algorithm runs well, but there are some critical situations that must be handled very carefully – namely, the cases when the clipped line is collinear or nearly collinear with an edge, i.e., the case when $\boldsymbol{n}_i^{\mathrm{T}}\boldsymbol{s} = 0$ or is close to zero.

It is obvious that the CB algorithm computes N values of the parameter t for all intersection points, but only two are actually needed if an intersection exists. It means the CB algorithm is of $O(N)$ complexity. There are algorithms of $O(\lg N)$ complexity, that are convenient for higher N – see [13, 16].

4 The line-clipping algorithm

A new approach to line clipping by a convex polygon in E^2 is presented. It is based on a function of separation. The main advantages of the proposed approach are robustness, stability, speed, and native use of homogeneous coordinates.

Let us assume a convex polygon P (see Fig. 3) and a line p given as $F(x) = ax + by + c = 0$.

The line p subdivides the space into two half-spaces as shown in Fig. 3, i.e., $F(x) < 0$ and $F(x) \geq 0$.

It can be seen that the function $F(x)$ can be evaluated for each vertex of the given convex clipping polygon, and for the ith vertex, the value c_i is obtained as follows:

$$c_i = \begin{cases} 1 & \text{if } F(x_i) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad i = 0, \ldots, N-1.$$

This means that each vertex is classified whether it is on the "left" (if we look from infinity in the direction of the $\boldsymbol{x}_B - \boldsymbol{x}_A$ vector) or on the "right" side of the clipped line, p.

For the purposes of explaining the algorithm, let us assume a rectangular window, i.e., a polygon with 4 orthogonal edges (see Fig. 4). This is actually a very frequent occurrence for line segment clipping.

Now, we will concentrate on the algorithm for line clipping. The line segment clipping algorithm will be explained later on.

In the case of a rectangular window, the vector $\boldsymbol{c}$ consists of bits:

$$\boldsymbol{c} = [c_3, c_2, c_1, c_0]^{\mathrm{T}}$$

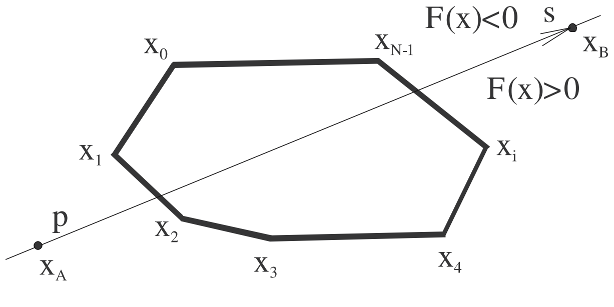

Fig. 3. Classification of polygon vertices

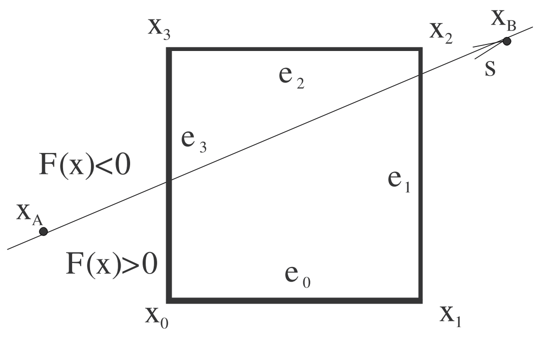

Fig. 4. Clipping against a rectangular window

Table 1. Table TAB with all possible cases. N/A denotes non-applicable cases

$\boldsymbol{c}$	c_3	c_2	c_1	c_0	***TAB*1**	***TAB*2**	MASK
0	0	0	0	0	None	None	None
1	0	0	0	1	0	3	0100
2	0	0	1	0	0	1	0100
3	0	0	1	1	1	3	0010
4	0	1	0	0	1	2	0010
5	0	1	0	1	N/A	N/A	N/A
6	0	1	1	0	0	2	0100
7	0	1	1	1	2	3	1000
8	1	0	0	0	2	3	1000
9	1	0	0	1	0	2	0100
10	1	0	1	0	N/A	N/A	N/A
11	1	0	1	1	1	2	0010
12	1	1	0	0	1	3	0010
13	1	1	0	1	0	1	0100
14	1	1	1	0	0	3	0100
15	1	1	1	1	None	None	None

and "codes" the position of a line p according to the given polygon. It is necessary to point out that this coding is different from the coding scheme of the Cohen–Sutherland algorithm (we do not code the position of points $\boldsymbol{x}_A, \boldsymbol{x}_B$).

Let us construct a table TAB of all the possible values of the vector $\boldsymbol{c}$ (see Table 1).

If all the combinations are interpreted geometrically, it can be seen that for each line of the table, the indices of the intersected edges can be pre-computed. This means that the function $F(x)$ is evaluated for each vertex of the clipping polygon, and the indices of the intersected edges are stored in vectors ***TAB*1** and ***TAB*2** (see Table 1). Some combinations are not applicable, e.g., $[0, 1, 0, 1]^{\mathrm{T}}$, and these combinations are indicated as N/A.

It can be seen that the vectors ***TAB*1** and ***TAB*2** are symmetric, i.e., the orientation of the clipping polygon P is not needed.

The vectors ***TAB*1** and ***TAB*2** do not depend on the position of the polygon vertices; they depend only on the number of vertices N of the given convex clipping polygon, i.e., the given table TAB is unchanging for all orthogonal clipping windows.

The extension to the general case when the given line p is clipped by the convex polygon P is straightforward, i.e., the vector $\boldsymbol{c}$ has N bits. In this case, vectors $\boldsymbol{TAB}$**1** and $\boldsymbol{TAB}$**2** can be similarly generated. Of course, it is necessary to determine which edges intersect for each possible combination of the vector $\boldsymbol{c}$.

In fact, the indices of intersected edges can be determined easily. If the two following bits in the vector $\boldsymbol{c}$ are different, i.e., $c_i \neq c_{i+1}$, the edge $\boldsymbol{x}_i\boldsymbol{x}_{i+1}$ will be intersected. Of course, there will be non-applicable cases, denoted as N/A, because in the case of a convex clipping polygon, the edges are actually split into

- one segment consisting of edges on the "right" side of the line p,
- one segment consisting of edges on the "left" side of the line p,
- edges intersected by the line p, if any.

It can be seen that the table construction can be done only once for all lines clipped if the clipping polygon P is not changed. If necessary, the vectors $\boldsymbol{TAB}$**1** and $\boldsymbol{TAB}$**2** can be interpreted by **if** statements "on the fly," or alternatively, indices can also be computed by a simple algorithm.

The Algorithm 2 describes the proposed algorithm for line clipping by the given convex polygon. It can be seen that the proposed algorithm is very simple.

Algorithm 2: The proposed algorithm for line-clipping by a rectangular window

```
procedure C_L;
{C_L – clipping lines}
{$\boldsymbol{x}_A$ , $\boldsymbol{x}_B$ – in homogeneous coordinates}
{input: $\boldsymbol{x}_A$ , $\boldsymbol{x}_B$}
begin {$\boldsymbol{x}_A = [x_A, y_A, w_A]^T$}
{1}     $\boldsymbol{p} := \boldsymbol{x}_A \times \boldsymbol{x}_B$; {$ax+by+c=0$; $\boldsymbol{p}=[a,b,c]^T$}
{2}     for $k := 0$ to $N-1$ do {$\boldsymbol{x}_k = [x_k, y_k, w_k]^T$}
{3}         if $\boldsymbol{p}^T\boldsymbol{x}_k \geq 0$ then $c_k := 1$ else $c_k := 0$;
{4}     if $\boldsymbol{c} \neq [0\ldots0]^T$ and $\boldsymbol{c} \neq [1..1]^T$ then
{5}     begin  $i := \boldsymbol{TAB}1[\boldsymbol{c}]$; $j := \boldsymbol{TAB}2[\boldsymbol{c}]$;
{6}            $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_i$ ; $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_j$ ;
{7}            output ($\boldsymbol{x}_A, \boldsymbol{x}_B$ )
{8}     end
end {C_L}
```

As all operations in Algorithm 2 are performed in homogeneous coordinates, the algorithm can be used in cases when input and output points are in homogeneous coordinates as well.

Note that

- the dot-product processor can compute $\boldsymbol{p}^T\boldsymbol{x}_k$ – step{3}
- the cross-product processor can perform steps {1, 6}
- the loop – steps {2,3} – can be made in parallel.

In the case where the clipping polygon is a rectangular window, the proposed algorithm can be implemented efficiently as the dot product and cross product computations can be simplified.

5 Theoretical comparison

The properties of the CB and the proposed C_L algorithms were analyzed and compared. A number of floating point operations were used for rough estimation of the properties of both algorithms and for theoretical comparison. The number of operations presented in Table 2 were found after optimization of the appropriate sequences of the CB and the proposed C_L algorithms for a rectangular window. The following simple modifications of both algorithms were evaluated:

1. input (x, y), output (x, y), denoted as CB_1 and C_L_1
2. input (x, y, w), output (x, y), denoted as CB_2 and C_L_2
3. input (x, y, w), output (x, y, w), denoted as CB_3 and C_L_3

It is obvious that if the algorithms are used in today's computer systems, this estimation is influenced by caching, multithreading, etc., and this analysis can only be used for very basic performance estimations.

Table 2. Number of floating-point operations needed

	(	±	*	/	>	:=	)		(	±	*	/	>	:=	)
CB_1	(	6	4	0	1	8	)	+ N *	(	4	5	1,5	2,5	4	)
CB_2	(	6	4	4	1	8	)	+ N *	(	4	5	1,5	2,5	4	)
CB_3	(	6	4	4	1	8	)	+ N *	(	4	5	1,5	2,5	4	)
C_L_1	(	9	17	2	0	9	)	+ N *	(	3	6	0	0	3	)
C_L_2	(	9	18	0	0	9	)	+ N *	(	3	6	0	0	3	)
C_L_3	(	9	13	0	0	9	)	+ N *	(	3	6	0	0	3	)

Let us define the increase in speed as

$$v_i = \frac{T_{\text{CB}_i}}{T_{\text{C\_L}_i}},$$

where T_{CB} and $T_{\text{C\_L}}$ is time spent by CB and by the new proposed C_L algorithm, respectively.

The theoretical estimation of the potential increase in speed is based on experimentally obtained timing of floating-point instructions (where the ratio, not the absolute value, is substantial); other operations were not considered.

Table 3. Relative timing of floating operations for a 750 MHz CPU

Timing	±	*	/	>	:=
Clocks	31	31	78	149	16

Table 4. Estimation of the increase in speed of the algorithm modifications for clipping on an N-sided convex polygon

N	3	4	5	6	10	20
ν_1	1,5	1,6	1,7	1,8	2,0	2,3
ν_2	1,7	1,8	1,9	2,0	2,2	2,3
ν_3	1,9	2,0	2,1	2,1	2,3	2,4

The theoretical increase in speed for the abovementioned modifications, (C_L$_1$, C_L$_2$, C_L_3) is as follows:

In the first two cases, the output of the proposed algorithm is in Euclidean coordinates, while in the third case, the output is in homogeneous coordinates, as expected in the graphical pipeline of GPU processors.

It can be seen that the proposed C_L algorithm:

- should be faster than the CB algorithm and the increase in speed will effectively grow with the number of polygon edges N,
- no division by homogeneous coordinates after vertex transformation is needed,
- if the resulting end-points are in homogeneous coordinates, no division is needed,
- is robust as no sensitive tests are needed, e.g., $a = b$ in floating-point representation,
- it is very simple and easy to implement,
- if dot product and cross product operations are supported in the hardware, the proposed algorithm will benefit from it.

In the case of hardware implementation, the timing of all of the operations takes just one cycle, and operations such as $a * b + c$ take one cycle as well. Unfortunately, the current GPUs do not allow us to prepare an experimental implementation, as the clipping part of the graphics pipeline is fixed in the hardware.

6 Experimental results

Both the CB and the proposed C_L algorithms were implemented in Pascal (Delphi) and C++ and compared. The experiments proved that the increase in speed for all cases

$$\boldsymbol{\nu} \geq 1,9.$$

In addition, the increase in speed grows slightly with value N, where N is the number of edges of the given convex clipping polygon.

Several factors, e.g., caching, additional instructions, cycles, etc., cause the difference between theoretical estimation and experimental results.

The main advantages of the proposed C_L algorithm are as follows:

- the robustness of the algorithm,
- the simplicity of computation in homogeneous coordinates,
- simple implementation in software and hardware,
- vector–vector and parallel processing can be used to speed up the proposed algorithm significantly (the **for** loop can be performed in parallel).

7 Line segment clipping

In the previous parts, a new algorithm for line clipping by a convex polygon has been described, compared with the Cyrus–Beck algorithm, and experimentally evaluated. In order to explain the proposed algorithm, a rectangular window was used.

In many applications, line segment clipping is required, especially for a rectangular clipping window. The Cohen–Sutherland algorithm is used almost exclusively for those purposes. Therefore, the question is how the proposed algorithm can be modified for the line segment case. Of course, the coding of the end-points used in the Cohen–Sutherland algorithm is very effective, and the proposed algorithm for the line segment clipping will use the coding of the end-points as well.

The Cohen–Sutherland algorithm

The Cohen–Sutherland (CS) [5] algorithm is used very often. The CS algorithm relies on the end-point classification (see Fig. 5). This classification enables us to detect cases when the given line segment is inside or totally above, below, to the left or to the right of the clipping window.

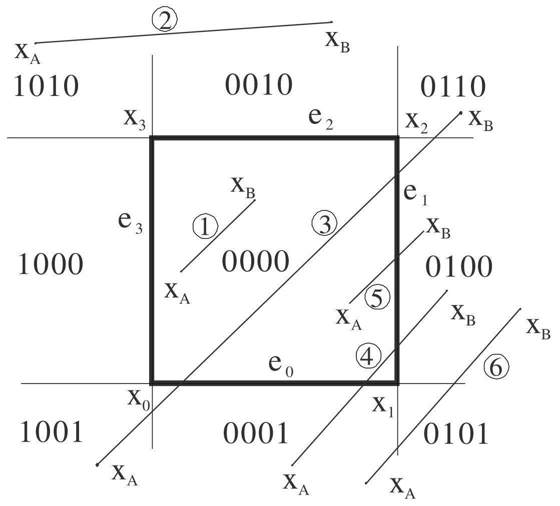

Fig. 5. A line segment classification

Algorithm 3 describes the principle of the CS algorithm – see [5] for the complete algorithm. It can be seen that the algorithm contains a loop and a quite complicated decision scheme for deciding which edge is intersected by the given line segment.

Algorithm 3: Principle of the Cohen–Sutherland algorithm

```
{global variables $x_{\min}$ , $y_{\min}$ , $x_{\max}$, $y_{max}$}
function CODE ($\boldsymbol{x}$);
begin
    $\boldsymbol{c} := [0000]$;
    if $x < x_{\min}$ then $\boldsymbol{c} := [1000]$
            else if $x > x_{\max}$ then $\boldsymbol{c} := [0100]$;
    if $y < y_{\min}$ then $\boldsymbol{c} := \boldsymbol{c}$ lor $[1001]$
            else if $y > y_{\max}$ then $\boldsymbol{c} := \boldsymbol{c}$ lor $[0010]$;
    CODE := $\boldsymbol{c}$
end {CODE};
procedure CS;
{input: $\boldsymbol{x}_A$ , $\boldsymbol{x}_B$}
{The EXIT statement ends the procedure}
begin
    $\boldsymbol{c}_A$ := CODE ($\boldsymbol{x}_A$); $\boldsymbol{c}_B$ := CODE ($\boldsymbol{x}_B$);
    if ($\boldsymbol{c}_A$ lor $\boldsymbol{c}_B$) = 0 then {case 1}
        begin output ($\boldsymbol{x}_A$; $\boldsymbol{x}_B$); EXIT {CS}
    end;
    if ($\boldsymbol{c}_A$ land $\boldsymbol{c}_B$) $\neq$ 0 then {case 2} EXIT; {CS}
    repeat
        if $\boldsymbol{c}_A = \mathbf{0}$ then $\boldsymbol{c} := \boldsymbol{c}_B$ else $\boldsymbol{c} := \boldsymbol{c}_A$;
        if ($\boldsymbol{c}$ land [0001]) then
        begin
          $y := y_A + (y_{\max} - y_A) * (y_B - y_A)/(x_B - x_A)$;
          $x := y_{\min}$;
        end else . . .  {similarly for others cases}
        if $\boldsymbol{c} = \boldsymbol{c}_A$ then
        begin $x_A := x$; $y_A := y$; $\boldsymbol{c}_A$ := CODE($\boldsymbol{x}_A$) end
        else
        begin $x_B := x$; $y_B := y$; $\boldsymbol{c}_B$ := CODE($\boldsymbol{x}_B$) end
        if ($\boldsymbol{c}_A$ land $\boldsymbol{c}_B$) $\neq$ 0 then {case 2} EXIT;
    until ($\boldsymbol{c}_A$ lor $\boldsymbol{c}_B$) = 0 {zero vector};
    output ($\boldsymbol{x}_A$, $\boldsymbol{x}_B$)
end {CS}
```

It can be seen that up to four divisions are needed for intersection computations in the worst case if the CS algorithm is used and additional four divisions for input points given in homogeneous coordinates.

Modification of the C_L algorithm for line segment clipping

Let us introduce a modification of the C_L algorithm for line segment clipping. We have to distinguish cases when both end-points are inside or outside of the clipping window from the case when only one end-point is inside the window. In the latter case, we have to determine which edge of two known edges (a line on which the line segment lies intersect two edges of the given rectangular window) is intersected by the given line segment. One end-point can lie inside of the clipping rectangle, i.e., an intersection point of the line p with e_0 or e_1 must be computed – see case 5 in Fig. 5.

The solution is very simple. Let us consider case 5 in Fig. 5. The same coding used within CS algorithm is applied for the each end-point, i.e., for the situation shown in Fig. 5. In case of the end-point $\boldsymbol{x}_A$, the vector $\boldsymbol{c}_A$ is constructed according to the CS coding scheme as

$$\boldsymbol{c}_A = [\text{LEFT} \mid \text{RIGHT} \mid \text{TOP} \mid \text{BOTTOM}]$$

and similarly for $\boldsymbol{x}_B$.

If the C_L algorithm is used directly, we know the edges, i.e., edges e_0 and e_1, intersected by a line p on which the given line segment lies. The indices of those edges are stored in the vectors ***TAB*1** and ***TAB*2**.

This means that a simple test, whether the "outside" point is on the left of the edge e_1 or below the edge e_0, must be used. If $\boldsymbol{c}_B = [*1**]$, then the edge e_1 is intersected. By contrast, if $\boldsymbol{c}_B \neq [*1**]$, then the edge e_0 is intersected by the given line segment. This actually results in the following condition:

```
if $\boldsymbol{c}_B$ land MASK[$\boldsymbol{c}$] $\neq$ 0
   then intersection $p\&e_1$
   else intersection $p\&e_3$
```

where $\boldsymbol{c}$ is the polygon vertex (not the end-point) classification code, and MASK[$\boldsymbol{c}$] = [0100] for this case. The table TAB (see Table 1) defines these mask values.

Algorithm 4 describes the proposed clipping line segment (C_LS) algorithm for the line segment clipping by a rectangular window.

Algorithm 4: The proposed algorithm for line segment clipping

```
function CODE ($\boldsymbol{x}$);
begin
    $\boldsymbol{c} := [0000]$;
    if $x < x_{\min}$ then $\boldsymbol{c} := [1000]$
            else if $x > x_{\max}$ then $\boldsymbol{c} := [0100]$;
    if $y < y_{\min}$ then $\boldsymbol{c} := \boldsymbol{c}$ lor $[1001]$
            else if $y > y_{\max}$ then $\boldsymbol{c} := \boldsymbol{c}$ lor $[0010]$;
    CODE := $\boldsymbol{c}$
end {CODE};
procedure C_LS; {input: $\boldsymbol{x}_A$, $\boldsymbol{x}_B$}
{C_LS – clipping line segment}
{$\boldsymbol{x}_A$, $\boldsymbol{x}_B$ – can be in homogeneous coordinates}
{The EXIT statement ends the procedure}
{land / lor – bitwise operations and / or}
begin {end-points classifications}
    $\boldsymbol{c}_A$ := CODE($\boldsymbol{x}_A$); $\boldsymbol{c}_B$ := CODE($\boldsymbol{x}_B$);
    {detection of trivial cases}
    {case 1 – line segment inside}
    if $(\boldsymbol{c}_A$ lor $\boldsymbol{c}_B) = \mathbf{0}$ then
        begin output $(\boldsymbol{x}_A; \boldsymbol{x}_B)$; EXIT {C_LS}
    end;
    {case 2 – line segment outside – just EXIT}
    if $(\boldsymbol{c}_A$ land $\boldsymbol{c}_B) \neq \mathbf{0}$ then EXIT; {C_LS}
    {all trivial cases solved}
    {compute coefficients of the line $p$}
    $\boldsymbol{p} := \boldsymbol{x}_A \times \boldsymbol{x}_B$; $\{ax+by+c=0;\ \boldsymbol{p}=[a,b,c]^T\}$
    for $k := 0$ to 3 do $\{\boldsymbol{x}_k = [x_k, y_k, 1]^T\}$
        if $\boldsymbol{p}^T\boldsymbol{x}_k \geq 0$ then $c_k := 1$ else $c_k := 0$;
    $\{\boldsymbol{c} = [c_3, c_2, c_1, c_0]^T\}$
    if $\boldsymbol{c} = [0000]^T$ or $\boldsymbol{c} = [1111]^T$ then EXIT;
    {the line segment lies outside of the window}
    {it distinguishes cases 4 and 6 in Fig. 5}
    {there MUST be one intersection at least}
    $i := \boldsymbol{TAB1}[\boldsymbol{c}]$; $j := \boldsymbol{TAB2}[\boldsymbol{c}]$;
    if $\boldsymbol{c}_A \neq \mathbf{0}$ and $\boldsymbol{c}_B \neq \mathbf{0}$
    then
        {there are two intersections}
        begin $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_i$; $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_j$ end
        {vector $\boldsymbol{e}_i$ is pre-defined for the $i$th edge}
    else {there is only one intersection point}
        if $\boldsymbol{c}_A = \mathbf{0}$ then {$\boldsymbol{x}_B$ is outside}
            begin
                if $\boldsymbol{c}_B$ land MASK$[\boldsymbol{c}] \neq \mathbf{0}$
                then $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_i$
                else $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_j$
            end
        else if $\boldsymbol{c}_B = \mathbf{0}$ then {$\boldsymbol{x}_A$ is outside}
            begin
                if $\boldsymbol{c}_A$ land MASK$[\boldsymbol{c}] \neq \mathbf{0}$
                    then $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_i$
                    else $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_j$
            end;
    output $(\boldsymbol{x}_A, \boldsymbol{x}_B)$
end {C_LS}
```

It can be seen that the proposed C_LS algorithm

- uses the vector dot product and cross product operations in homogeneous coordinates,
- has a loop that can be easily replaced by evaluation in parallel,
- has a very simple decision scheme convenient for hardware implementation,
- does not use the division operation.

The main advantage is that it only uses homogeneous coordinates, which make hardware implementation simpler.

Experimental results

Both the original CS and the proposed C_LS algorithms were implemented and tested. The C_LS is slightly faster, and the experimentally measured increase in speed ν was approximately

$$\nu = \frac{T_{CS}}{T_{C\_LS}} \cong 1,2.$$

Nevertheless, its use and implementation is targeted toward hardware-supported implementation. Also, all tests were performed only for cases when the homogeneous coordinate of the end-points was $w = 1$. If the end-points of the line segment are given in homogeneous coordinates, the algorithm is faster, as it saves up to four divisions per line segment.

It is expected that the proposed C_LS algorithm will perform significantly better in hardware implementations and speed up the clipping and computational operations in the graphical pipeline.

Future possible extensions of the algorithm

One can ask if the proposed algorithm can be modified for a case when the clipping polygon is not convex. In this case, we have to answer the following questions:

- What actually does the convexity mean to the algorithm?
- How is the table TAB constructed and interpreted?

It is actually a simple and straightforward extension of the presented principle. Let us imagine a simple situation in Fig. 6 with a non-convex, non-self-intersecting clipping polygon P.

It is obvious that the polygon P is intersected by the given line p several times. The direct impact to the table TAB is that it has no N/A cases anymore, and all combinations have to be interpreted. In our case, the vector $\boldsymbol{c}$ is defined as

$$\boldsymbol{c} = [c_8, c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0]^T;$$

i.e., for the case in Fig. 6,

$$\boldsymbol{c} = [1, 0, 1, 0, 1, 1, 1, 0, 0]^T,$$

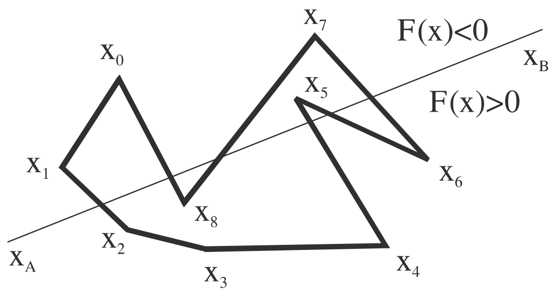

Fig. 6. Clipping by a non-convex polygon

and the same algorithm for finding indices of intersected edges can be applied as in the convex polygon case. Of course, the table TAB must be implemented as a "jagged" array (a vector of vectors) in this case. This is due to the fact that we can have more pairs of edges that are intersected by the given line p. Special handling of the intersection points will be needed as they are not ordered in one direction on the line p, and a similar approach to [14] can be applied.

8 Conclusion

This paper describes a new approach to the line clipping problem and the line segment clipping problem in E^2 using projective space and homogeneous coordinates. The proposed algorithms are robust and easy to implement. Both algorithms use a separation function in order to obtain higher stability, and all computations are performed in homogeneous coordinates.

The first proposed algorithm C_L for the line clipping by a convex polygon is faster than the original Cyrus–Beck algorithm and does not need a predefined clipping polygon orientation, for example. If the clipped line is given by end-points in homogeneous coordinates, the C_L algorithm is especially convenient.

The second proposed algorithm C_LS for the line segment clipping by a rectangular window is a direct modification of the C_L algorithm. It is faster than the Cohen–Sutherland algorithm and significantly faster especially if the line segment is given by two end-points in homogeneous coordinates, and output points are in homogeneous coordinates as well.

Both proposed algorithms will benefit in an increase in speed if vector–vector operations or/and parallel processing can be used. The cross product processor will increase the performance significantly as well. Both proposed algorithms increase performance if the clipping window is $\langle 0, 1\rangle \times \langle 0, 1\rangle$, as many operations might be optimized for this case.

Acknowledgement The author would like to express his thanks to students and colleagues at the University of West Bohemia for recommendations, constructive discussions, and hints that helped to finish the work. Many thanks belong to the anonymous reviewers for their valuable comments and suggestions that improved this paper significantly, to Ivana Kolingerova for her critical comments and recommendations, to Eva Hamousova for proof reading, to Jana Hajkova, Petr Cizek for verifications, and to Ivo Hanak for hints in the evaluation of hardware features.

This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200005, Microsoft Research Ltd. (U.K.) project No. 2004 – 360.

References

1. Bui DH, Skala V (1998) Fast algorithms for clipping lines and line segments in E^2. Vis Comput 14(1):31–37
2. Bui DH (1999) Algorithms for line clipping and their complexity. Dissertation, University of West Bohemia
3. Coxeter HSM (1961) Introduction to geometry. Wiley, New York
4. Cyrus M, Beck J (1978) Generalized two- and three-dimensional clipping. Comput Graph 2(1):23–28
5. Foley DJ, van Dam A, Feiner SK, Hughes JF (1990) Computer graphics – principles and practice. Addison-Wesley, Boston
6. Hartley R, Zisserman A (2000) Multiview geometry in computer vision. Cambridge University Press, Cambridge
7. Hill FS (2001) Computer graphics using OpenGL. Prentice-Hall, New York
8. Huang YQ, Liu YK (2002) An algorithm for line clipping against a polygon based on shearing transformations. Comput Graph Forum 21(4):683–688
9. Johnson M (1996) Proof by duality: or the discovery of "new" theorems. Math Today, pp. 171–174, November/December
10. Liang Y, Barsky B (1984) A new concept and method for line clipping. ACM Trans Graph 3(1):1–22
11. Maillot PG (1992) A new, fast method for 2D polygon clipping: analysis and software implementation. ACM Trans Graph 11:276–290
12. Nicholl TM, Lee DT, Nicholl RA (1987) An efficient new algorithm for 2-D line clipping: its development and analysis. In: SIGGRAPH Proceedings 21(4):253–262
13. Rappaport A (1991) An efficient algorithm for line and polygon clipping. Vis Comput 7(1):19–28
14. Skala V (1989) Algorithm for 2D line clipping. In: Proceedings of Computer Graphics International '89, pp 121–128
15. Skala V (1993) An efficient algorithm for line clipping by convex polygon. Comput Graph 17(4):417–421
16. Skala V (1994) O(lg N) line clipping algorithm in E^2. Comput Graph 18(4):517–524
17. Skala V (1996) An efficient algorithm for line clipping by convex polygon and non-convex polyhedrons in E^3. Comput Graph Forum 15(1):61–68
18. Skala V (1996) Line clipping in E^2 with suboptimal compexity $O(1)$. Comput Graph 20(4):523–530
19. Skala V, Bui DH (2001) Extension of the Nicholls–Lee–Nichols algorithm to three dimensions. Vis Comput 17:236–242
20. Stolfi J (2001) Oriented projective geometry. Academic Press, Burlington
21. Zhang M, Sabharwal CL (2002) An efficient implementation of parametric line and polygon clipping algorithm. In: Proceedings of the ACM Symposium on Applied Computing. ACM Press, New York, pp 796–800

VACLAV SKALA is a full professor at the University of West Bohemia in Plzeň. He is responsible for the Centre of Computer Graphics and Visualization (http://herakles.zcu.cz) and he is the Head of Computer Graphics Group at University of West Bohemia in Plzeň.