A Brief Survey of Clipping and Intersection Algorithms with a List of References (including Triangle-Triangle Intersections)✩

Vaclav SKALA

University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Pilsen, CZ 301 00, Czech Republic

e-mail: skala@kiv.zcu.cz

Received: June 2022; accepted: January 2023

Abstract. This contribution presents a brief survey of clipping and intersection algorithms in $E^2$ and $E^3$ with a nearly complete list of relevant references. Some algorithms use the projective extension of the Euclidean space and vector-vector operations, which support GPU and SSE use.

This survey is intended to help researchers, students, and practitioners dealing with intersection and clipping algorithms.

Key words: intersection algorithms, line clipping, line segment clipping, polygon clipping, triangle-triangle intersection, homogeneous coordinates, projective space, duality, computer graphics, geometry, convex polygon, convex polyhedron.

1. Introduction

Intersection algorithms are key algorithms in many areas, e.g. in geometry intersection algorithms of two lines in $E^2$ or three planes in $E^3$, CAD/CAM systems, etc. Many of those algorithms are part of standard courses and based on formulations in the Euclidean geometry, e.g. Schneider and Eberly (2003). However, there is a problem with results in infinity or close to infinity. Some of those can be solved using the projective extension of the Euclidean space and the principle of duality (Johnson, 1996; Skala, 2010). The projective extension of the Euclidean space enables representation of points in infinity and the application of the principle of duality to solve dual problems by the same algorithm (Coxeter and Beck, 1992; Johnson, 1996; Skala, 2008b). Such approach leads to formulations using vector-vector operations, which is convenient for GPU and SSE instructions.

Algorithms for intersection computation of different geometric entities in $E^2$ and $E^3$ are studied for a long time from various aspects. Their robustness and precision of numerical calculations is severely influenced by the limited numerical accuracy available on today’s computer system. It is well known that $(1/3) \times 3 \neq 1$ in “the computer world”.

✩ The research was supported by the University of West Bohemia – Institutional research support.
Even a simple summation \( S = \sum_{i=1}^{n} a_i \) is not easy in the case of large-range data (Skala, 2013b).

It should be noted that, not only in geometry oriented algorithms, a special care has to be devoted to the cases where differences between mathematics with infinite precision and mathematics with a limited precision might cause problems leading to the unexpected and incorrect results, sometimes also leading to disasters.

Unfortunately, programmers and computer scientists are mostly targeted at “the technology of implementation”. They have a limited understanding of numerical aspects of today’s numerical data representation, limited more or less to the IEEE-754 floating-point representation (Wikipedia, 2021b). Despite the technological progress, the binary128 and binary256 precision are not supported in hardware. It appears that there is no possibility to represent rational, irrational and transcendental numbers used in mathematics, where unlimited accuracy is expected, e.g. what is the difference between the value of \( \pi \) and \((\text{long real } \pi)(\text{long real } \pi)\) if the IEEE-754 representation is used?

Line, half-line (ray), line segment and triangle-triangle intersection algorithms are considered fundamental in nearly all algorithms dealing with geometrical aspects (Skala, 2022).

2. Projective Space and Principle of Duality

The majority of intersection 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 into the necessity to convert the results of the geometric transformations to the Euclidean space using division operation.

2.1. Projective Extension of the Euclidean Space

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

\[
X = x/w, \quad Y = y/w \quad & \quad w \neq 0, \tag{1}
\]

where \( w \) is the homogeneous coordinate.\(^1\)

It means that a point \( X \in E^2 \) is represented by a line in the projective space \( [x, y : w]^T \) without the origin, which represents a point in infinity, see Fig. 1.

The extension to the \( E^3 \) case is straightforward (Foley et al., 1990).

\[
X = x/w, \quad Y = y/w, \quad Z = z/w \quad & \quad w \neq 0, \tag{2}
\]

where \( x = [x, y, z : w]^T \).

\(^1\)In mathematics, a different notation \( x = [x_0 : x_1, \ldots, x_n]^T \) is used; where \( x_0 \) represents the homogeneous coordinate \( w \).
Clipping and Intersection Algorithms: Survey

171

Fig. 1. Projective space and its dual.

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 it enables to represent a point in infinity. Also, it enables to express some geometric entities in a more compact form, e.g. a line in the $E^2$ case as:

$$aX + bY + c = 0, \quad ax + by + cw = 0, \quad a^T x = 0,$$

where $a = [a, b : c]^T$, and $x = [x, y : w]^T$.

It is necessary to note that $(a, b)$ represents the normal vector\(^2\) 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:

$$aX + bY + cZ + d = 0, \quad ax + by + cz + dw = 0, \quad a^T x = 0,$$

where $a = [a, b, c : d]^T$ and $x = [x, y, z : w]^T$. 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. Figure 1 presents the duality in $E^2$ – the line $p$ is represented as a point $D(p)$ in the dual space (Stolfi, 1991). Unfortunately, the principle of duality is not usually part of the standard computer science curricula.

2.2. Principle of Duality

The principle of duality is one of essential principles in mathematics. In our case of geometric problems described by linear equations, see Eq. (3) and Eq. (4), 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,

\(^2\)Actually, it is a bivector (Vince, 2008).
• “lie on” and “pass through”, “join” and “intersection” and so on.

Once the theorem has been established, the dual theorem is obtained as described (John-son, 1996).

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.

\[
\begin{bmatrix}
  a_1 & b_1 \\
  a_2 & b_2 
\end{bmatrix}
\begin{bmatrix}
  X \\
  Y 
\end{bmatrix} = \begin{bmatrix}
  -c_1 \\
  -c_2 
\end{bmatrix}, \quad \text{i.e. } Ax = b, \quad \begin{bmatrix}
  X_1 & Y_1 & 1 \\
  X_2 & Y_2 & 1 
\end{bmatrix}
\begin{bmatrix}
  a \\
  b \\
  c 
\end{bmatrix} = \begin{bmatrix}
  0 \\
  0 
\end{bmatrix}, \quad \text{i.e. } Ax = 0.
\]

(5)

It is strange as the usual solution in the first case leads to formulation $Ax = b$, while in the second case, the parameters of a line are determined as $Ax = 0$. However, if the projective representation is used, both cases are solved as $Ax = 0$ (Skala, 2008b). Similarly, the intersection computation of three planes is dual to the computation of a plane given by three points in the $E^3$ case.

Generally, a system of linear equations $Ax = 0$ can be solved as:

\[
x = a_1 \land a_2 \land \cdots \land a_n,
\]

(6)

where $a_i$ are rows of the matrix $A$, $\land$ is the outer product, i.e. extended cross product, and $x = [x_1, \ldots, x_n : w]^T$ is the solution in the homogeneous coordinates. It means that a line given by two points $x_A, x_B$, resp. an intersection of two lines $p_1, p_2$ is given in $E^2$ as $p = x_A \land x_B$, resp. $x = p_1 \land p_2$, due to the principle of duality.

It should be noted that a line in $E^2$ can be expressed as:

\[
aX + bY + c = 0 \quad \text{in the implicit form or} \quad
X(t) = X_A + S_x t, \quad Y(t) = Y_A + S_y t \quad \text{in the parametric or}
\]

\[
Y = kX + q, \quad \text{resp. } X = mY + p \quad \text{in the explicit form.}
\]

(7)

In the case of $E^3$ a line cannot be expressed in the implicit form, but as an intersection of two planes or in the parametric form as:

\[
a_1X + b_1Y + c_1 = 0 \quad \& \quad a_2X + b_2Y + c_2 = 0 \quad \text{in the implicit form or} \quad
X(t) = X_A + S_x t, \quad Y(t) = Y_A + S_y t, \quad Z(t) = Z_A + S_z t
\]

(8)

There is a special parametric form of the line in $E^3$, which uses the Plücker coordinates. It has a specific property as the point $(X_A, Y_A, Z_A)$ is the closest point to the origin of
the coordinate system (Blinn, 1977; Mahovsky and Wyvill, 2004; Platis and Theoharis, 2003; Wikipedia, 2020).

In computer graphics, some intersection algorithms are called clipping algorithms and serve to determine a part of one geometric entity inside the second one.

In the following, a brief classification of intersection algorithms in 2D and 3D will be presented with short characteristics; a short overview can be found in Wikipedia (2021a).

There are many variants of fundamental algorithms that differ in some aspects; mainly, the timing factor is the primary motivation. However, the claimed speed up mostly depends on the hardware properties (memory caching, processor used, etc.), programmer’s skill and actual language and compiler used.

3. Intersection Algorithms in 2D

Algorithms for intersections of different 2D geometric entities have been studied for a long time from various aspects, primarily due to the computation speed, robustness and limited numerical precision of the floating-point representation. The majority of 2D algorithms deal with an intersection of a line or a half-line (ray) or a line segment with 2D geometric entity, e.g. a rectangle, convex polygon (Cyrus and Beck, 1978; Rappoport, 1991), non-convex polygon (Weiler and Atherton, 1977), quadric and cubic curves, parametric curves (Skala, 2021a) and areas with quadratic arcs (Skala, 2015, 1989, 1990a), etc.

There are two main strategies, which are “dual” in some sense:

- a position of the window, resp. polygon edges against the intersected line, resp. line segment, etc.,
- a position of the vertices of the window, resp. polygon against the intersected line, resp. line segment, etc.

3.1. Intersection with a Rectangular Area

Intersection algorithms with a rectangular area (window) are well known as the line clipping or as the line segment clipping algorithms. The first algorithm was developed and used for the flight simulator project led by Cohen (1969) in 1967. Efficient coding of the line segment position coding leading to significant computational reduction was introduced in Sproull and Sutherland (1968) and patented in 1972 (Sutherland, 1972). The Cohen-Sutherland algorithm is described in Newman and Sproull (1979), Comninos (2006), Matthes and Drakopoulos (2019a, 2019b), etc. The Cohen-Sutherland algorithm generates a bit-code LRTB, i.e. [Left, Right, Top, Bottom], for each end-point of the line segment, see Fig. 2. The coding is redundant. However, it enables simple identification of the cases, when the line segment is totally inside or outside as follows:

- if (\(c_A \lor c_B\)) = [0000] then the line segment is totally inside,
- if (\(c_A \land c_B\)) \(\neq\) [0000] then the line segment is totally outside,

where \(\land, \lor\) mean bit-wise \(\text{and}, \text{or}\) operations.
Table 1

Numerical summation codes \( C_{AB} = C_A + C_B \), IN – inside area, C – corner area, S – side area, n/a – non-applicable cases or outside case.

<table>
<thead>
<tr>
<th>( C_{AB} )</th>
<th>( C_B )</th>
<th>IN</th>
<th>C</th>
<th>S</th>
<th>C</th>
<th>S</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>0001</td>
<td>0010</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

The ultimately deep classification of all the possible cases using arithmetic operation with the codes was described in Skala (2021b), see Table 1 and Fig. 3. The \( C_{AB} \) value is the index to the array of functions representing each case.

Distinguishing all the cases leads to more efficient coding and efficient implementation (Skala, 2021b); specific cases are presented in Table 2.

The Cohen-Sutherland algorithm can also be extended to the 3D case, i.e. intersection of a line segment with a cube or right-angled parallelepiped.

The Cohen-Sutherland algorithm was improved by Nicholl et al. (1987). It uses the window corners position classification in relation to the line segment position, see Fig. 4. The Nicholl-Lee-Nicholl algorithm was improved by Bui and Skala (1998) using some additional classification of possible cases and extended to the \( E^3 \) case in Skala and Bui (2001).

The algorithms (Liang and Barsky, 1983) and (Dörr, 1990) are based on the direct intersection computation of a line with the polygon edges in the parametric form. Analy-
Fig. 3. Two specific situations – SS-SnCS: side-side and side-neighbour corner-side.

Table 2
Possible cases: n/a – non-applicable or solved by the C-S coding, C – corner area, S – side area, IN – inside area, End-points: IC – inside-corner, IS – inside-side; Cases: SS – side-side, SnCS – side-near corner-side, SdC – side-distant corner-side, CoC – corner-opposite corner, id – case re-indexing.

<table>
<thead>
<tr>
<th>id</th>
<th>Case</th>
<th>( C_B )</th>
<th>( C_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IN</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>S</td>
<td>1001</td>
<td>1001</td>
</tr>
</tbody>
</table>

Fig. 4. Nicholl-Lee-Nicholl algorithm – window corners position evaluation.
sis of the Nicholl-Lee-Nicholl and Liang-Barsky algorithms was given in Devai (2005). Simple and robust line and line segment clipping algorithms in $E^2$ was described in Skala (2004, 2005, 2012, 2020). They are based on the projective representation and homogeneous coordinates using a separation of the convex polygon vertices by the given line, see Fig. 5. The sign of the function values $F(x)$, which represents the given line, for each window corner gives a 4-bit code identifying the edges intersected by the given line. The algorithm can be extended for the convex polygon case.

3.2. S-L-Clip Algorithm

Let us consider an implicit function $F(x) = a^T x$, where $a = [a, b : c]^T$ are coefficients of the given line $p$, $x = [x, y : w]^T$ means a point on this line. Then the equation $F(x) = 0$ represents the given line $p$ in $E^2$ using the projective extension of the Euclidean space.

The clipping operation should determine the intersection points $x_i = [x_i, y_i : w_i]^T$, $i = 1, 2$ of the given line with the window, if any. The line splits the plane into two parts, see Fig. 5. The corners of the window are split into two groups according to the sign of the function $F(x)$ 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:

$$c_i = \begin{cases} 1, & F(x_i) \geq 0, \\ 0, & \text{otherwise}, \end{cases}$$

where $a = [a, b : c]^T$ are coefficients of the given line $p$, $x = [x, y : w]^T$ means a point on this line. Table 3 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 is used in the line segment algorithm).

It can be seen, that the S-L-Clip Algorithm 1 is quite simple and easily extensible for the convex polygon clipping case as well. Table 3 can be generated synthetically. It is significantly more straightforward than the algorithm (Liang and Barsky, 1984). 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
Clipping and Intersection Algorithms: Survey

Table 3
All cases; N/A – non-applicable (impossible) cases.

<table>
<thead>
<tr>
<th>c</th>
<th>c</th>
<th>TAB1</th>
<th>TAB2</th>
<th>MASK</th>
<th>c</th>
<th>c</th>
<th>TAB1</th>
<th>TAB2</th>
<th>MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>15</td>
<td>1111</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0</td>
<td>3</td>
<td>0100</td>
<td>14</td>
<td>1110</td>
<td>3</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0</td>
<td>1</td>
<td>0100</td>
<td>13</td>
<td>1101</td>
<td>1</td>
<td>01</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>1</td>
<td>3</td>
<td>0010</td>
<td>12</td>
<td>1100</td>
<td>3</td>
<td>1</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>1</td>
<td>2</td>
<td>0010</td>
<td>11</td>
<td>1011</td>
<td>2</td>
<td>1</td>
<td>0010</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>10</td>
<td>1010</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0</td>
<td>2</td>
<td>0100</td>
<td>9</td>
<td>1001</td>
<td>2</td>
<td>0</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>2</td>
<td>3</td>
<td>1000</td>
<td>8</td>
<td>1000</td>
<td>3</td>
<td>2</td>
<td>1000</td>
</tr>
</tbody>
</table>

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: \> computation of the line coefficients
3: \> codes computation
4: \> first intersection point
5: \> second intersection point

1: \> defining the line, are generally in the projective representation, i.e. $w \neq 0$. 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 (Skala, 2020).

The modification of the S-L-Clip algorithm for a line segment clipping is simple and described in Skala (2004). The advantage of it is that the end-points and the window corners might be given generally in the projective space, i.e. $w \neq 0$. The cross-product is used for the intersection computation using SSE4 or GPU acceleration.


3.3. Intersection with Polygons

Generic solutions for polygon clipping were developed by Weiler and Atherton (1977), Rappoport (1991), Vatti (1992), Wu et al. (2004), Xie et al. (2010), Zhang and Sabharwal (2002), Zhang et al. (2022). Boolean operations with polygons were introduced by Rivero and Feito (2000), Martinez et al. (2009).

Algorithms for a line clipping $E^2$ by a polygon depend on the polygon property, i.e. if the polygon is convex or non-convex. In the case of convex polygons, the convexity property and ordering of vertices enables to decrease complexity from $O(N)$ to $O(\lg N)$ (Skala, 1994). It should be noted that a similar complexity decrease is not possible in the $E^3$ case as there is no ordering.

In the non-convex polygon cases, when the polygon can be self-intersecting, etc., problems with robustness of computation can be expected. Also, in some cases a three-value logic is to be used in order to solve specific cases properly, e.g. a line passes a vertex, a line touches a vertex, a line lies on an edge, etc. (McCoid and Gander, 2022; Skala, 1989, 1990a).

3.4. Convex Polygons

The Cyrus-Beck’s algorithm (Cyrus and Beck, 1978) is probably the famous algorithm for line-convex polygon clipping. It is based on a computation of the parameter $t$ of the given line in the parametric form with edges of the given convex polygon, Fig. 6. The algorithm is of $O(N)$ computational complexity and can be extended for the $E^3$ case.

![Fig. 6. Cyrus-Beck line clipping algorithm.](image-url)
The Cyrus-Beck’s algorithm is based on direct intersection computation of the given line $p$ in the parametric form and a line on which the polygon edge $e_i$ lies, see Fig. 6, in the implicit form, i.e. on a solution of two linear equations (vector notation is used):

\begin{align*}
  p : \quad & x(t) = x_A + s t, \\
  e_i : \quad & n_i^T x + c_i = 0, \quad i = 0, \ldots, N - 1,
\end{align*}

where $x_A = [x_A, y_A]^T$, $s = [s_x, s_y]^T$ is the directional vector of the line $p$, $n_i = [n_x, n_y]^T$ is the normal vector of the edge $e_i$.

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

\begin{equation}
  n_i^T x_A + n_i^T s t + c_i = 0.
\end{equation}

Then $t_i$ is the parameter $t$ value for the intersection of the line $p$ and the line on which the edge $e_i$ lies, see Fig. 6.

\begin{equation}
  t_i = -\frac{n_i^T x_A + c_i}{n_i^T s}.
\end{equation}

It can be seen that the algorithm is not robust as if the line $p$ is parallel or nearly parallel to the edge $e_i$, the expression $n_i^T s \to 0$ and $t_i \to \pm \infty$. The fraction computation might cause an overflow or high imprecision of the computed parameter $t$ value, see Fig. 6.

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

\begin{verbatim}
if |n_i^T s| < eps then a singular case
\end{verbatim}

which is an incorrect solution as the value $eps$ is the programmer’s choice and the value of $n_i^T s$ might also be close to the value of $n_i^T x_A + c_i$, see Eq. (12).

However, textbooks do not point out such dangerous construction as far as robustness and computational stability are concerned.

The modification of the Cyrus-Beck’s algorithm using the cross product for more reliable detection of the “close to singular” cases was described by Skala (1993). Probably the most reliable modification of the Cyrus-Beck’s algorithm is to use:

- a separation implicit function $F(x) = 0$ representing the given line $p$ defined as $F(x) = n^T x + c$ for intersection detection as in Skala (2005),
- the parametric form of the given line for intersection computation with the found edges intersected, see Eq. (12).

The Cyrus-Beck’s algorithm for a line clipping is described by Algorithm 2. It can be easily modified for the line segment clipping just restricting the range of the parameter $t$ to $\langle 0, 1 \rangle$, i.e.

\begin{equation}
  \langle t_{\min}, t_{\max} \rangle := \langle t_{\min}, t_{\max} \rangle \cap \langle 0, 1 \rangle.
\end{equation}
Algorithm 2  Cyrus-Beck’s line clipping algorithm

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

It can be seen that the algorithm complexity is $O(N)$ and the division operation, which is \( \text{the most consuming time operation in the floating-point representation, is used } N \text{ times.}^3 \)

However, only 2 values ($t_{\min}$, $t_{\max}$) of the parameter $t$ are valid, i.e. $N − 2$ computations of the parameter $t$ are lost. Also, reliable detection of the “singular or close to singular” cases is difficult and time-consuming, especially in the $E^3$ case.

Some improvements and modifications were described by Skala (1993). As the edges of the convex polygon are ordered, the algorithm with the $O(\lg N)$ complexity was derived by Skala (1994). An algorithm based on space subdivision was described in Slater and Barsky (1994).

Another approach based on the implicit form of the given line and convex polygon vertices classification, the S-Clip algorithm, was developed in Skala (2021c) and modified by Konashkova (2014, 2015). Another algorithm based on the S-Clip algorithm was described in Skala (2021c). An algorithm for a line segment clipping based on the line segment end-points evaluation with the $O(N)$ complexity was described by Matthes and Drakopoulos (2022).

---

3There is a possibility to postpone division operations if the homogeneous coordinates are used, but comparison operations must be modified appropriately (Skala, 2020, 2021c).
The Liang-Barsky algorithm (Liang and Barsky, 1984, 1983) based on direct intersection computation of a line with the convex polygon edges in the parametric form has the $O(N)$ computational complexity, too.

The algorithm with the run-time $O(1)$ complexity using pre-computation was developed by Skala (1996b, 1996d). The algorithm was motivated by the scan-line raster conversion used recently for solving visibility in rendering. The memory requirements depend on the geometrical properties of the given convex polygon. A comparison of the $O(1)$ algorithm with the Cyrus-Beck algorithm is presented in Skala and Lederbuch (1996), Skala et al. (1996).

Other related algorithms or modifications of existing ones were published by: Li (2005), Nishita and Johan (1999), Raja (2019), Sun et al. (2006), Vatti (1992), Wang et al. (2005), Wijeweera et al. (2019), Sharma and Kaur (2016), Sharma and Manohar (1992) use the affine transformation.

### 3.5. Non-Convex Polygons

Probably, the first algorithm dealing with the non-convex polygon clipping was published in the Reentrant polygon clipping algorithm paper (Sutherland and Hodgman, 1974), followed by the Weiler-Atherton algorithm for polygon-polygon clipping (Weiler, 1980; Weiler and Atherton, 1977; Rappoport, 1991).

Intersections with arbitrary non-convex polygons were described in Greiner and Hormann (1998) and solutions of “the singular” (degenerated) cases were described in Foster et al. (2019). The algorithm (Skala, 1989) uses a three-value logic.

A robust solution of triangle-triangle intersection in $E^2$ is described in McCroid and Gander (2022). Other algorithms or modifications are described in Dimri (2015), Evangeline and Anitha (2014), Lu and Wu (2002), Lu et al. (2002a), Tang and He (2009). The affine transformations are used in Huang (2013), Huang and Wangyong (2009), Huang and Liu (2002).

Algorithms that also handle arcs and use a three-value logic to handle singular cases properly, including self-intersecting non-convex polygons, were described in Skala (2015, 1989, 1990a), Wang and Chong (2016), Tran (1986).

### Non-Polygonal Window

The algorithm for circular arc was described in Van Wyk (1984), Gupta et al. (2016), for overlapping areas by Li et al. (2012) and for circular window in Lu et al. (2002b), Kumar et al. (2018), Wu and Li (2022), Wu et al. (2006), Skala (1989), see Fig. 7. The above-mentioned algorithms lead to algorithms for set operations with polygons, i.e. union, intersection etc. of polygons described, e.g. Kui Liu et al. (2007), Martinez et al. (2009).

### 3.6. Clipping Using Homogeneous Coordinates

Homogeneous coordinates are used in computer graphics not only for geometric transformations. Sproull and Sutherland (1968) used the homogeneous coordinates in the Clipping divider in 1968. Arokiasamy (1989) used them with duality in 1989, Blinn (1991), Blinn

In the following, algorithms related to the intersection in 3D will be briefly mentioned in a short introductory overview.

4. Intersection Algorithms in 3D

Intersection algorithms in 3D are widely used in many applications. An overview of the clipping algorithms is given in the Bui’s PhD (Bui, 1999). The intersection of a line segment with a polygon in 3D was studied in Segura and Feito (1998) and the intersection of polygonal models was analysed by Melero et al. (2019). Algorithms for 3D clipping were overviewed in Skala (1990b) and reliable intersection tests with geometrical objects were published by Held (1998). Boolean operations with polygonal and polyhedral meshes were described by Landier (2017).

Line-Viewing Pyramid

Special attention was recently given to a line clipping by a pyramid in 3D due to the perspective pyramid clipping. The problem was analysed recently by Cohen (1969), Sproull and Sutherland (1968), Blinn (1991), Blinn and Newell (1978), Skala and Bui (2000, 2001).

Convex Polyhedron Case

The Cyrus-Beck’s algorithm (Cyrus and Beck, 1978) is probably the famous algorithm for the line-convex polyhedron clipping in $E^3$. It computes a parameter $t$ of a line in the parametric form and plane of the given face of the convex polyhedron. The algorithm is of the $O(N)$ computational complexity given by the fact that in the $E^3$ space there is “no order” of the polyhedron facets defined. Rogers and Rybak (1985) published a more general clipping algorithm in 3D in 1995.
The algorithm with the $O_{\exp}(\sqrt{N})$ complexity was described in Skala (1997, 2014). It assumes a triangular mesh, i.e. there is info on the neighbour triangles available. The algorithm is based on two planes representing the given line in $E^3$ and testing of the neighbours in the triangular mesh of the given polyhedron. The algorithm was modified by Konashkova (2015). An interesting approach using the vertex connection table was published in Konashkova (2014).

Using pre-computation, the algorithm in $E^3$ with a run-time $O(1)$ complexity was developed by Skala (1996c). Comparison was presented in Skala et al. (1996).

**Ray-Convex Polyhedron**

The Moeller-Trumbore algorithm for a ray-triangle intersection was published in Möller and Trumbore (1997). Since then many modifications and approaches have been published, e.g. Xiao et al. (2020) using GPUs, Skala (2010, 2008a) uses the computation of the barycentric coordinates in the homogeneous coordinates, Rajan et al. (2020) uses dual-precision fixed-point arithmetic for low-power ray-triangle intersections. Platis and Theoharis (2003) published an algorithm for a ray-tetrahedron intersection using the Plücker coordinates. The intersection with the AABBBox is described in Eisemann et al. (2007), Kodituwakku and Wijeweera (2012), Maonica et al. (2017) and Mahovsky and Wyvill (2004). Other algorithms are available in Sharma and Manohar (1993), Skala (1996a), Williams et al. (2005), Llanas and Sainz (2012). The 3D line segment-triangle intersection algorithm is described in Jokanovic (2019), Amanatides and Choi (1995), Lagae and Dutré (2005) (in 2D only) and a ray/convex polyhedron intersection was described in Zheng and Millham (1991). Intersection of a line or a ray with a triangle using the SSE4 instructions was developed and described in Havel and Herout (2010). An extensive list of relevant publications can be found via Wikipedia (2021c).

**Intersection with Complex Objects**

The intersection computation with implicitly defined objects was published by Petrie and Mills (2020), intersection with a torus was published by Cychosz (1991) and alternative formulations were given in Skala (2013a). Reshetov (2022) published an efficient algorithm for a ray/ribbon intersections computation, ray tracing of 3D Bézier curves given by Reshetov (2017) and a ray/bilinear patch intersection (Reshetov, 2019). The intersection with general quadrics using the homogeneous coordinates was described in Skala (2015) and clipping by a spherical window was published by Deng et al. (2006).

However, as polygonal models are mostly formed by triangular surfaces, a special attention is also targeted to triangle-triangle intersections.

**Triangle-Triangle Intersection in 3D**

The computation of the intersection of triangles is probably the most important, as nearly all Computer Aided Design (CAD) systems depend on efficient, robust and reliable computation. Figures 8 and 9 present the non-trivial cases, when triangles are split into a set of triangles, which potentially leads to an explosion of small triangles and numerical and robustness problems.

In the CAD systems, two different data sets are usually used:
set of triangles – there is no connection between triangles; typical example is the STL format for the 3D print,
triangular mesh – there is information on the neighbours of the given triangles and triangles sharing the given vertex; a typical example is the winged edge or the half-edge data structures, etc.

An efficient triangle-triangle intersection algorithm was developed by Möller (1997). It is based on the mutual triangle intersection with the plane of the other. Other methods or approaches were described by Chang and Kim (2009), Danaei et al. (2017), Devillers and Guigue (2002), Elsheikh and Elsheikh (2014), Guigue and Devillers (2003), Held (1998), Sabharwal and Leopold (2016), Sabharwal et al. (2013), Sabharwal and Leopold (2015), Shen et al. (2003), Tropp et al. (2006), Roy and Dasari (1998), Wei (2014), Ye et al. (2015). A deep analysis of possible situations is given in Lo and Wang (2004). Robust and reliable solution of the triangle-triangle intersection was developed by Mccoid and Gander (2022).

Clipping triangular strips using homogeneous coordinates was described by Maillot (1991) in GEM II (Arvo, 1991). Parallel exact algorithm for the intersection of large 3D triangular meshes was described in de Magalhães et al. (2020) and a comparison of triangle-triangle tests on GPU was described in Xiao et al. (2020). Triangular mesh repair was described by McLaurin et al. (2013).
5. Conclusion

This contribution briefly summarizes known clipping algorithms with some extent to the intersection in 3D and ray-tracing related algorithms. The list of published papers related to clipping algorithms should be complete to the author’s knowledge and extensive search via Web of Science, Scopus, Research Gate and WEB search with the related topics. The relevant DOIs were included, if found. If other source was found, the relevant URL was included.

There is hope that this summary will help researchers, students and software developers to find relevant papers easily.

However, developers are urged to consider a limited precision of the floating-point representation (Wikipedia, 2021b) and handle numerical robustness issues properly for near singular cases in the actual implementations.

Surprisingly, during this summary preparation it was found that there are still some problems to be addressed and explored more deeply, like a robust and efficient intersection of triangular meshes as application of triangle-triangle intersection algorithms tend to lead to inconsistencies, inefficiency and unreliability, in general.

A short list of relevant books and research journals is given in Appendix A.

Appendix A

There are many books published related to intersection algorithms, clipping and computer graphics, which give more context and deeper understanding, e.g.:

• Theoharis, T., Platis, N., Papaioannou, G., Patrikalakis, N.: Graphics and Visualization: Principles & Algorithms – Theoharis et al. (2008),
• Comninos, P.: Mathematical and Computer Programming Techniques for Computer Graphics – Comninos (2005),
• Ammeraal, L., Zhang, K.: Computer graphics for Java programmers – Ammeraal and Zhang (2017),

There are also computer graphics books using OpenGL interface, e.g.:

More advanced books using Geometric Algebra and Conformal Geometric Algebra approaches are recommended for a deeper study, e.g.:
• Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry – Dorst et al. (2009),
• Calvet, R.G.: Treatise of Plane Geometry through Geometric Algebra – Calvet (2007),
• Pharr, M., Jakob, W., Humphreys, G.: Physically Based Rendering: From Theory to Implementation – Pharr et al. (2016),

It is also recommended to study “the historical” books, e.g.:
Clipping and Intersection Algorithms: Survey

- Eberly, D.H.: Game Physics – Eberly (2003),

Many algorithms with codes are presented in GEMS books:

- Graphics Gems, Ed. Glassner, A. – Glassner (1990),

and in the leading computer graphics journals:

- ACM Transactions on Graphics (TOG),
- Computer Graphics Forum (CGF),
- Computers & Graphics (C&G),
- IEEE Trans. on Visualization and Computer Graphics (TVCG),
- The Visual Computer (TVC),
- Computer Animation and Virtual Worlds (CAVW),
- Journal of Graphics Tools (JGT),
- Graphical Models.

The books mentioned above present a wide variety of intersection algorithm principles and applications.

Acknowledgements

The author would like to thank colleagues and students at the University of West Bohemia in Pilsen, VSB-Technical University and Ostrava University in Ostrava for their comments and recommendations, anonymous reviewers for hints and constructive suggestions.

The author is also grateful to several authors of recently published relevant papers for sharing their views and hints provided.

References

Clipping and Intersection Algorithms: Survey


192

V. Skala


Sabharwal, C.L., Leopold, J.L. (2016). A generic design for implementing intersection between triangles in Clipping Algorithm


V. Skala is a full professor of computer science at the University of West Bohemia, Pilsen, Czech Republic. He received his Ing. (equivalent of MSc) degree in 1975 from the Institute of Technology in Pilsen, CSc. (equivalent of PhD) degree from the Czech Technical University in Prague, in 1981. In 1996, he became a full professor in computer science. He is a Fellow of the Eurographics Association, member of several editorial boards of international research journals and the editor-in-chief of the Journal of WSCG and Computer Science Research Notes. He is the organizer of the WSCG conferences on computer graphics, visualization and computer vision (www.wscg.eu) held annually since 1992.

His current research interests are computer graphics and visualization, applied mathematics, especially geometrical algebra, algorithms, and data structures.