**RUHUNA JOURNAL OF SCIENCE**
Vol 10(2): 161-173, December 2019
eISSN: 2536-8400
DOI: http://doi.org/10.4038/rjs.v10i2.81



Short Paper

# A novel and efficient approach for line segment clipping against a convex polygon

K. R. Wijeweera[1, 4*], S. R. Kodituwakku[2, 4], M. A. P. Chamikara[2, 3]

[1]*Department of Computer Science, Faculty of Science, University of Ruhuna, Sri Lanka*
[2] *Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya, Sri Lanka*
[3]*School of Computer Science and Information Technology, Royal Melbourne Institute of Technology, Australia*
[4]*Postgraduate Institute of Science, University of Peradeniya, Sri Lanka*

*Correspondence: *krw19870829@gmail.com*; https://orcid.org/0000-0002-8933-1687

Received: 07th August 2018, Revised: 20th November 2019, Accepted: 03rd December 2019

**Abstract:** This paper proposes a new line clipping algorithm against a convex polygon with $O(N)$ time complexity. The line segment is pruned against each extended edge of the polygon as the first step of the proposed algorithm. Then, the pruning process gives accurate outcomes for completely inside and partially inside line segments only. The algorithm was developed based on the observation that the endpoints of completely outside line segments coincide after the pruning process. Theoretical and experimental comparisons of the current algorithm against existing ones reveal that it is faster than the Cyrus Beck algorithm but is slower than ECB, Rappaport, and Skala algorithms.

**Keywords:** Computer Graphics Programming, Line Clipping Algorithms, Computational Geometry, Convex Analysis, Time Complexity.

## 1 Introduction

Generally, any approach that extracts parts of a picture that are either inside or outside of a specified region of space is called a clipping algorithm. The clip window is defined as the region against which an object is clipped. There are many applications of clipping algorithms: extracting part of a defined scene for viewing; identifying visible surfaces in three-dimensional views; antialiasing line segments or object boundaries; creating objects using solid modeling procedures; displaying a multi-window environment; drawing and painting operations that allow parts of a picture to be selected for copying, moving,

erasing, or duplicating. The clip window can be a general polygon or it can even have curved boundaries depending on the application. The clipping algorithms are basically used for clipping the following primitive types: point clipping, line clipping (straight line segments), area clipping (polygons), curve clipping, and text clipping. The graphics packages usually include line and polygon clipping methods as standard components. In addition many graphics packages facilitate curved objects, specially spline curves and conics, such as circles and ellipses. An alternative way to handle curved objects is to approximate them with straight line segments and apply the line clipping or polygon clipping methods (Hearn and Baker 1998).

There are several algorithms in literature to clip line segments against a convex polygon (Skala 1994). These algorithms have been derived from well-known Cohen Sutherland (Cohen 1969), Liang Barsky (Liang and Barsky 1983, 1984), and Cyrus Beck (Cyrus and Beck 1978) algorithms. Let N be the number of vertices of the convex polygon. All of the existing algorithms have O (N) time complexity except the algorithms proposed by Rappaport and Skala. The Rappaport algorithm and Skala algorithm have O (log N) time complexity (Rappaport 1991). The speed of these algorithms depends on more or less clever implementation of tests and intersection computation. The ECB line clipping algorithm was invented by observing the convexity feature of the clipping polygon and the possibility of binary search usage over polygon vertices (Skala 1993). Skala further improved the ECB line clipping algorithm using the known order of vertices of the polygon to an O (log N) algorithm (Skala 1994).

The Cohen Sutherland algorithm is a well-known algorithm for clipping line segments against a rectangular window (Cohen 1969). The original paper describes the algorithm when the edges of the rectangular window are parallel to the principle axes. The extended edges of the rectangular window partition the plane into nine regions. Each region is assigned a region code of four bits as shown in Figure 1.

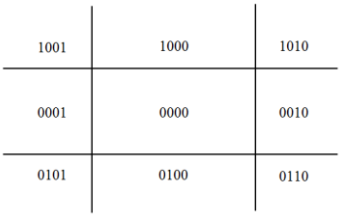


**Fig. 1. The nine region codes**

The region codes of each endpoint of the line segment are computed first. It can be decided whether the line segment is completely inside by performing operator OR between the two region codes. If not, the line segment is tested for being completely outside by performing operator AND between the two region

codes. If the line segment is neither completely inside nor completely outside, then only intersections are calculated with the extended edges of the clipping window. This process is iterated until the final outcome is reached. The geometric transformations are used when the edges of the rectangular window are not parallel to the principle axes. The system is rotated so that the edges become parallel to the principle axes. Then the original algorithm can be used to clip the line segment. Again, the system is re-rotated to get the actual endpoints of the clipped line segment. These geometric transformations involve a higher computational cost due to the use of trigonometric functions. The number of regions generated by the extended edges depends on the angles between the edges and the number of vertices when the clipping window is a polygon. Therefore, the known algorithms for line clipping against a polygon do not use the tests similar to the Cohen Sutherland algorithm (Skala 1994).

It is necessary to distinguish cases where line segments intersect a given window from those where line segments do not intersect the window in order to develop an effective line clipping algorithm. The Cyrus Beck algorithm performs direct computation of intersection points to solve this problem. The ECB algorithm was developed by using the separation theorem. However, the ECB algorithm does not use the known order of vertices of the given polygon and achieves O (N) time complexity (Skala 1994). The Rappaport algorithm is based on the known fact that whether a given point inside the convex polygon can be found in O (log N) time (Preparata and Shamos 1985). The other O (log N) algorithm proposed by Skala can be applied in situations where the edges of the convex polygon are arbitrarily oriented.

This paper proposes a novel line segment clipping algorithm extending a concept proposed earlier (Kodituwakku *et al.* 2012, 2013) to a convex polygon. The proposed algorithm takes O (N) time and does not use the property of known order of vertices of the polygon.

## 2 Material and Methods

This section presents the proposed line clipping algorithm. A convex polygon is considered as the clipping window. The vertices of the convex polygon have been labeled as shown in Figure 2.

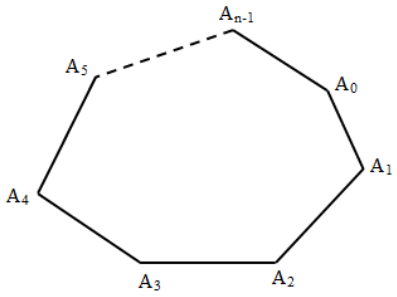


**Fig 2. A general convex polygon clipping window**

The convex polygon has n vertices: $A_0$, $A_1$…, $A_{n-1}$. They form the polygon by connecting each vertex and its subsequent vertex with straight line segments as shown in Figure 1. Finally, $A_0$ and $A_{n-1}$ are also connected to form the closed polygon. Note that the interior angle at each vertex is less than π and non-consecutive edges do not intersect. Here, $A_0 \equiv$ (x[0], y[0]), $A_1 \equiv$ (x[1], y[1])…, $A_{n-1} \equiv$ (x[n - 1], y[n - 1]).

## 2.1 Mathematical background of the proposed algorithm

The general equation of a straight line can be expressed as y = m * x + c. End points of the line segment to be clipped are A = (x1[0], y1[0]) and B = (x1[1], y1[1]), and m1 and c1 are considered as gradient and y-intercept of the line segment respectively.

Then the mean (xc, yc) of the vertices of the polygon can be calculated as given below.

$$xc = \frac{\sum_{i=0}^{n-1} x[i]}{n};\ yc = \frac{\sum_{i=0}^{n-1} y[i]}{n}$$

Let m[i] and c[i] be the gradient and the y-intercept of the line segment $A_iA_{i+1}$ respectively; where i = 0…, (n – 1). Then m[i] can be expressed as m[i] = (y[i+1] - y[i])/(x[i+1] - x[i]); where i = 0…, (n – 1). Note that the index arithmetic is *modulo* n.

By substituting m[i] in y[i] = m[i] * x[i] + c[i], c[i] can be computed as c[i] = ((x[i+1] * y[i]) - (x[i] * y[i+1]))/(x[i+1] - x[i]); where i = 0…, (n - 1).

Similarly, m1 and c1 (gradient and y-intercept respectively) of the line segment to be clipped are also calculated.

Let val[i] = m[i] * xc – yc + c[i]; and val1[i][j] = m[i] * x1[j] - y1[j] + c[i] where i = 0…, (n – 1); j = 0, 1.

In this method, the following two theorems are used (Green 1991).

**Theorem 1:** In a convex polygon, the mean of all vertices is always inside the polygon.
**Theorem 2:** Let (p, q) and (r, s) be two points and a * x + b * y + c = 0 be an equation of a straight line. If the value of (a * p + b * q + c) * (a * r + b * s + c) is negative, then the two points are in the opposite sides of the straight line.

Therefore, val[i] * val1[i][j] < 0 implies that the point (x1[j], y1[j]) is in the opposite side of the centroid with respect to $i^{th}$ polygon boundary where i = 1, 2…, (n - 1); j = 0, 1 and only in this case the intersection point of the line segment with the $i^{th}$ boundary has to be calculated.

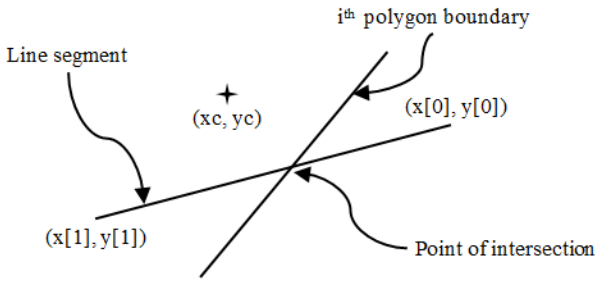


**Fig 3. Intersection calculation**

As shown in Figure 3, suppose (x[0], y[0]) is outside the polygon. Then the equation of the $i^{th}$ boundary of the polygon is y = m[i] * x + c[i]. The equation of the line segment is y = m1 * x + c1. By solving these two equations, we can get the intersection point (x_intersection, y_intersection). Then the resultant line segment is the line segment joining (x[1], y[1]) and (x_intersection, y_intersection).

By applying intersection calculation, the line segment can be pruned with respect to each extended edge of the polygon so that it is will eventually end up as the final version. If the above procedure is performed to a completely outside line segment, it would become a single point theoretically. However, the resultant two endpoints are approximately equal due to the precision error occurs in calculation of intersection points. Therefore, completely outside line segments can be easily removed by using this fact (Kodituwakku *et al.* 2012, Kodituwakku *et al.* 2013).

## 2.2 Pseudo code of the proposed algorithm

All the symbols used in the following pseudo code have been considered in the previous sections. To increase the understandability of the pseudo code, the cases given below have been ignored from it.

Case 1: Line segment is just a point.
Case 2: Line segment is parallel to the principle axes.
Case 3: Polygon boundaries are parallel to the principle axes.
Case 4: Line segment is parallel to some of the boundaries of the polygon.

Above four cases have been addressed at the implementation stage. Then the abstract pseudo code is as follows.

```
L1: BEGIN
L2:
L3: // Calculate val[i] and val1[i][j]
L4: For j = 0 to j = 1
L5:         For all the i boundaries of the polygon
L6:                 If ( val[i] * val1[i][j] < 0 ) Then
L7:                         //Calculate (x_intersection, y_intersection)
L8:                         x[j] = x_intersection;
L9:                         y[j] = y_intersection;
L10:                EndIf
L11:        EndFor
L12: EndFor
L13:
L14: // Initial line is completely outside
L15: If (x[0] - x[1] < 1) AND (x[1] - x[0] < 1) Then
L16: // Do nothing
L17: Else
L18: /*Save the line with end points   (x[0], y[0]), (x[1], y[1])*/
L19: EndIf
L20:
L21: END
```

When the line segment is completely outside, applying the FOR loops makes it a single point theoretically. Since computers truncate latter part of decimals, testing the exact equality is not possible. Therefore, approximate equality should be tested (Line L15). If the distance between x-coordinates are less than one-pixel length, we can consider those two points are equal. Therefore, if the points (x[0], y[0]) and (x[1], y[1]) are equal,

$$Distance(x[0], x[1]) < 1 \rightarrow |x[0] - x[1]| < 1 \rightarrow (x[0] - x[1]) < 1 \; AND \; (x[1] - x[0]) < 1$$

## 3 Results & Discussion

The proposed algorithm is developed so that it can be used for any polygon with any number of vertices. In order to simplify the analysis, a pentagon is

used as the clipping window. How a line segment is clipped against the pentagon window for all possible cases is explained below.

**Case 1:** Line segment that is completely inside as shown in Figure 4.

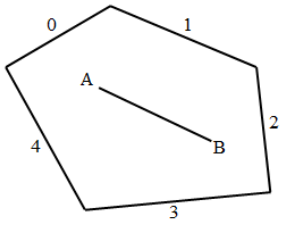


**Fig 4. Line segment is completely inside.**

Consider point A:
val[i] * val1[i][A] < 0 → false; i = 0, 1, 2, 3, 4. (Line L6)
Therefore, the initial position of A is not changed.

Consider point B:
val[i] * val1[i][B] < 0 → false; i = 0, 1, 2, 3, 4. (Line L6)
Therefore, the initial position of B is not changed.
(x[A] - x[B] < 1) AND (x[B] - x[A] < 1) → false. (Line L15)
Therefore, the line segment with the end points A and B is drawn. (Line L18)

**Case 2:** Line segment which is completely outside as shown in Figure 5.

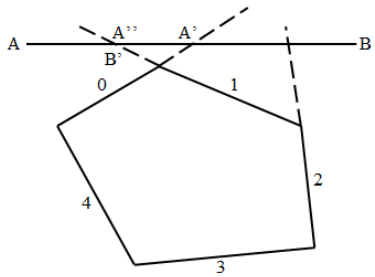


**Fig 5. Line segment is completely outside**

Consider point A:
val[0] * val1[0][A] < 0 → true. (Line L6)
Therefore, A → A'. (Line L8 & Line L9)
val[1] * val1[1][A'] < 0 → true. (Line L6)
Therefore, A' → A''. (Line L8 & Line L9)
val[i] * val1[i][A''] < 0 → false; i = 2, 3, 4. (Line L6)
Consider point B:
val[0] * val1[0][B] < 0 → false. (Line L6)

val[1] * val1[1][B] < 0 → true. (Line L6)
Therefore, B → B'. (Line L8 & Line L9)
val[i] * val1[i][B'] < 0 → false; i = 2, 3, 4. (Line L6)
(x[A''] - x[B'] < 1) AND (x[B'] - x[A''] < 1) → true. (Line L15)
Therefore, the line segment is ignored. (Line L16)

**Case 3:** Line segment which intersects the clipping window as shown in Figure 6.

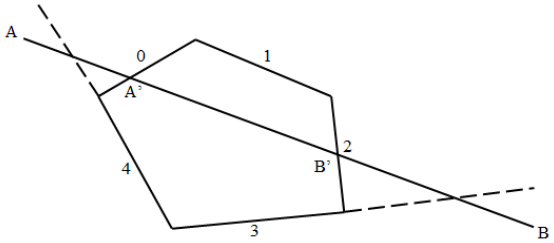


**Fig 6. Line segment is intersecting the boundaries**

Consider point A:
val[0] * val1[0][A] < 0 → true. (Line L6)
Therefore, A → A'. (Line L8 and Line L9)
val[i] * val1[i][A'] < 0 → false; i = 1, 2, 3, 4. (Line L6)
Consider point B:
val[i] * val1[i][B] < 0 → false; i = 0, 1. (Line L6)
val[2] * val1[2][B] < 0 → true. (Line L6)
Therefore, B → B'. (Line L8 and Line L9)
val[i] * val1[i][B'] < 0 → false; i = 3, 4. (Line L6)
(x[A'] - x[B'] < 1) AND (x[B'] - x[A'] < 1) → false. (Line 15)
Therefore, the line with the end points A' and B' is drawn. (Line 18)

**Case 4:** Line segment which is partially inside the clipping window as shown in Figure 7.

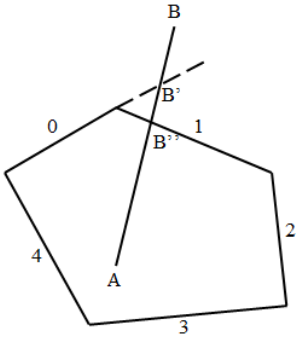


**Fig 7. Line segment is partially inside**

Consider point A:
val[i] * val1[0][i] < 0 → false; i = 0, 1, 2, 3, 4, 5, 6. (Line L6)
Therefore, the initial position of A is not changed.
Consider point B:
val[0] * val1[0][B] < 0 → true. (Line L6)
Therefore, B → B'. (Line L8 and Line L9)
val[1] * val1[1][B'] < 0 → true. (Line L6)
Therefore, B' → B''. (Line L8 and Line L9)
val[i] * val1[i][B''] < 0 → false; i = 2, 3, 4. (Line L6)
(x[A] - x[B''] < 1) AND (x[B''] - x[A] < 1) → false. (Line 15)
Therefore, the line with the end points A and B'' is drawn. (Line 18)

The proposed algorithm was compared against algorithms: Cyrus Beck, ECB, Rappaport, and Skala in order to validate the performance. The algorithms: Cyrus Beck and ECB have O (N) time complexity while the algorithms: Rappaport and Skala have O (log N) time complexity (Skala 1994). The time complexity of a line clipping algorithm is determined as the time taken to clip a given line segment against a polygon with N number of vertices. The line segment is tested against each edge of the polygon considering each end point of the line segment. Therefore, the time complexity of the proposed algorithm is O (N) + O (N) = O (N). Theoretical analysis proves that the proposed algorithm has the same speed as the algorithms: Cyrus Beck and ECB while it is slower than the algorithms: Rappaport and Skala.

Theoretical time complexity compares the algorithms for large values of N. However, polygonal clip windows with small number of vertices are also used in practice (Hearn and Baker 1998). An experimental analysis was performed in order to compare the behavior of the proposed algorithm with the existing algorithms for convex polygons with small number of vertices. All the algorithms were implemented in C programming language with following hardware and software resources.

**Computer:** Intel(R) Pentium(R) Dual CPU; E2180 @ 2.00 GHz; 2.00 GHz, 0.98 GB of RAM;

**IDE:** Turbo C++; Version 3.0; Copyright(c) 1990, 1992 by Borland International, Inc;

A pentagon window with vertices (200, 50), (400, 100), (300, 400), (150, 350) and (50, 250) was used (vertices are in boundary traversal order). Random points were generated by using the *randomize ()* and *random (double r)* functions. Those random points were generated in the range of 0-550. They were used as the end points of line segments. Then the number of clock cycles consumed to clip $10^8$ of such random line segments by each algorithm was counted. This procedure was repeated for 10 rounds where a new set of random line segments was used for each round. The results obtained are shown in Table 1. For an example, in the first round CB, ECB, Rapport, Skala, and Proposed

algorithms consumed 4020, 3350, 3095, 2610, and 3762 clock cycles respectively.

**Table 1:** The number of clock cycles to clip against pentagon window

| Round | CB | ECB | Rappaport | Skala | Proposed |
|---|---|---|---|---|---|
| 1 | 4020 | 3350 | 3095 | 2610 | 3762 |
| 2 | 4021 | 3351 | 3094 | 2611 | 3761 |
| 3 | 4020 | 3351 | 3096 | 2610 | 3763 |
| 4 | 4022 | 3352 | 3095 | 2612 | 3761 |
| 5 | 4020 | 3351 | 3095 | 2611 | 3760 |
| 6 | 4020 | 3350 | 3094 | 2610 | 3762 |
| 7 | 4021 | 3351 | 3094 | 2611 | 3761 |
| 8 | 4021 | 3352 | 3095 | 2611 | 3762 |
| 9 | 4021 | 3350 | 3096 | 2612 | 3763 |
| 10 | 4020 | 3351 | 3094 | 2610 | 3762 |

Let $T$ be the average number of clock cycles consumed by a given algorithm from the ten rounds given in the Table 1. Let's define coefficients of effectiveness [2] $v$ as $v_1 = T_{CB}/T_{Pro}$; $v_2 = T_{ECB}/T_{Pro}$; $v_3 = T_{Rap}/T_{Pro}$; $v_4 = T_{Ska}/T_{Pro}$.

Consider $v = T_{Alg1}/T_{Alg2}$ for the two algorithms *Alg1* and *Alg2*. Thus, $v \lesseqgtr 1$ gives a measure of effectiveness (in the sense of running time) of Algorithm 1 against Algorithm 2.

According to the values in the Table 1, $v_1 = 1.06$, $v_2 = 0.89$, $v_3 = 0.82$, $v_4 = 0.69$. Therefore, the proposed algorithm is 6% faster than Cyrus Beck algorithm, 11%, 18% and 31% slower than ECB, Rapport and Skala algorithms respectively for a pentagon window. Further, the proposed algorithm was compared against the existing algorithms using the set of polygons shown in Table 2. Note that *Pn* denotes a polygon with *n* number of vertices and the vertices are shown in boundary traversal order.

**Table 2:** The set of polygons.

| Polygon | Vertices |
|---|---|
| P3 | (100, 100), (300, 150), (500, 400) |
| P4 | (200, 100), (300, 200), (200, 400), (100, 200) |
| P5 | (200, 50), (400, 100), (300, 400), (150, 350), (50, 250) |
| P6 | (100, 50), (150, 50), (400, 300), (350, 430), (90, 350), (50, 250) |
| P7 | (100, 400), (250, 450), (450, 400), (500, 350), (400, 100), (200, 50), (50, 300) |
| P8 | (100, 400), (250, 400), (400, 300), (500, 150), (200, 50), (150, 50), (60, 100), (50, 300) |
| P9 | (150, 440), (300, 450), (400, 420), (500, 370), (350, 100), (250, 10), (100, 10), (10, 120), (50, 400) |

Next, a number of different polygons as shown in Table 2 were considered. The same strategy was performed without changing other conditions. Performance measures are listed in Table 3 and a comparison is shown in Figure 8.

**Table 3:** The number of clock cycles to clip against polygon windows

| Polygon | CB | ECB | Rappaport | Skala | Proposed | v1 | v2 | v3 | v4 |
|---|---|---|---|---|---|---|---|---|---|
| P3 | 3405 | 3825 | 2884 | 2530 | 3364 | 1.01 | 1.13 | 0.85 | 0.75 |
| P4 | 3661 | 3735 | 3403 | 2860 | 3595 | 1.01 | 1.03 | 0.94 | 0.79 |
| P5 | 4020 | 3351 | 3095 | 2610 | 3762 | 1.06 | 0.89 | 0.82 | 0.69 |
| P6 | 4317 | 3597 | 2853 | 2398 | 3964 | 1.08 | 0.9 | 0.71 | 0.6 |
| P7 | 4704 | 3590 | 2716 | 2283 | 4116 | 1.14 | 0.87 | 0.65 | 0.55 |
| P8 | 4919 | 4031 | 3034 | 2447 | 4288 | 1.14 | 0.94 | 0.7 | 0.57 |
| P9 | 5424 | 4140 | 3015 | 2432 | 4485 | 1.2 | 0.92 | 0.67 | 0.54 |

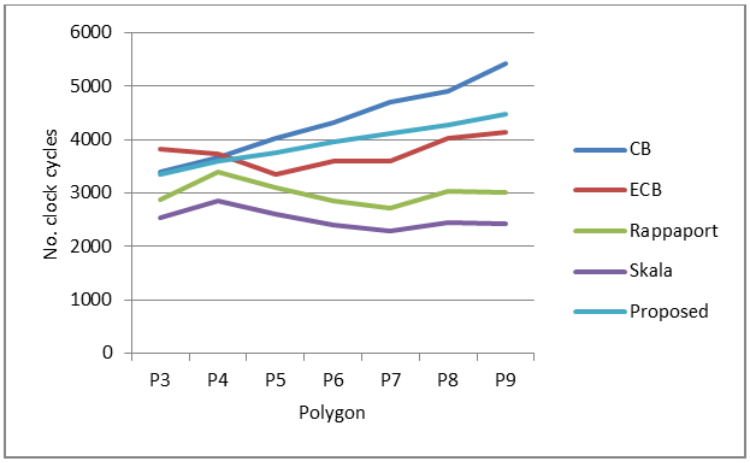


**Fig 8. The number of clock cycles to clip against polygon windows**

According to the results in Table 3 along with Figure 8, the proposed algorithm is faster than Cyrus Beck algorithm. Further, the proposed algorithm is faster than ECB algorithm when the number of vertices of the polygon is less than 5. The proposed algorithm is slower than the other existing algorithms.

## 4 Conclusions

A new algorithm of O (N) time complexity for clipping line segments against a convex polygon was proposed. The proposed algorithm was compared against the existing algorithms both theoretically and experimentally. The proposed algorithm is faster than Cyrus Beck algorithm and it is slower than ECB, Rappaport, and Skala algorithms.

The existing four algorithms can handle convex polygons only and they fail to handle arbitrary polygons (Skala 1993). The notions used in the algorithms

restrict them to convex polygons. The proposed algorithm computes the centroid of the polygon and considers that the centroid is inside the polygon. The centroid of an arbitrary polygon may be outside the polygon. Therefore, the proposed algorithm is also restricted to convex polygons. If a polygon contains a reflex vertex then it is not a convex polygon. The reflex vertices of a given polygon can be easily found (Wijeweera and Kodituwakku 2016). Thus, convex polygons can be identified. There are algorithms to partition an arbitrary polygon into a set of convex polygons (Wijeweera and Kodituwakku 2017). If a line segment needs to be clipped against an arbitrary polygon then the polygon could be partitioned into a set of convex polygons as the first step. The line segment could be clipped against each convex polygon and the set of derived clipped line segments could be merged to generate the final outcome. Thus, the proposed approach could be used to clip line segments again an arbitrary polygon as well.

The time complexities of the proposed algorithm and the Cyrus Beck algorithm are equal. However, the proposed algorithm consumes smaller number of clock cycles than the Cyrus Beck algorithm according to the experimental results. The Cyrus Beck algorithm computes all the intersection points to select the actual intersection point. In contrast, the proposed algorithm can early exit and avoid unnecessary intersection calculations (See Figure 5 and Figure 6). Therefore, the proposed algorithm is faster than the Cyrus Beck algorithm in practice.

**Acknowledgements**

Two anonymous reviewers are acknowledged for providing critical comments on the initial and revised versions of the manuscript.

**References**

Cohen D. 1969. Incremental methods for computer graphics. PhD Thesis, University of Harvard, Massachusetts.

Cyrus M, Beck J. 1978. Generalized two and three-dimensional clipping. *Computers & Graphics* 3(1): 23-28.

Green SL. 1991. Advanced level pure mathematics. North Point, Hong Kong: University Tutorial Press.

Hearn D, Baker MP. 1998. Computer graphics: c version, 2nd Edition. Prentice Hall, Inc. Upper Saddle River. 224-237.

Kodituwakku SR, Wijeweera KR, Chamikara MAP. 2012. An efficient line clipping algorithm for 3D space. *International Journal of Advanced Research in Computer Science and Software Engineering* 2 (5): 96-101.

Kodituwakku SR, Wijeweera KR, Chamikara MAP. 2013. An efficient algorithm for line clipping in computer graphics programming. *Ceylon Journal of Science (Physical Sciences)* 17: 1-7.

Liang YD, Barsky BA. 1983. An analysis and algorithms for polygon clipping. *CACM* 26 (11): 868-876.

Liang YD, Barsky BA. 1984. A new concept and method for line clipping. *ACM Transactions on Graphics* 3 (1): 1-22.

Preparata PF, Shamos MI 1985. Computational geometry: an introduction. Springer-Verlag, New York.

Rappaport A. 1991. An efficient algorithm for line and polygon clipping. *The Visual Computer* 7 (1): 19-28.

Skala V. 1993. An efficient algorithm for line clipping by convex polygon. *Computers & Graphics* 17 (4): 417-421.

Skala V. 1994. O (lg N) line clipping algorithm in $E^2$. *Computers & Graphics* 18 (4): 517-524.

Wijeweera KR, Kodituwakku SR. 2016. Accurate, simple, and efficient triangulation of a polygon by ear removal with lowest memory consumption. *Ceylon Journal of Science* 45 (3): 65-76.

Wijeweera KR, Kodituwakku SR. 2017. Convex partitioning of a polygon into smaller number of pieces with lowest memory consumption. *Ceylon Journal of Science* 46 (1): 55-66.

## Appendix

The implementation of the proposed algorithm in C# programming language is available as an appendix to the paper.