AN EFFICIENT ALGORITHM FOR LINE CLIPPING IN COMPUTER GRAPHICS PROGRAMMING

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

[1]*Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya*
[2]*Post Graduate Institute of Science, Faculty of Science, University of Peradeniya*

Abstract: Most of the line clipping algorithm are based on Cohen-Sutherland and Liang-Barsky algorithms. These algorithms involve a lot of calculations. This paper proposes a new line clipping algorithm for 2D space which is more efficient than the existing algorithms. The possible extended algorithm for 3D space is also presented. The algorithm proposed for the 2D space is compared against traditional line clipping algorithms. The proposed algorithm was tested for a large number of random line segments and the results showed that it performs better than the Cohen-Sutherland and Liang-Barsky algorithms.

Keywords: Computer Graphics Programming, Line Clipping, 2D geometry, 3D geometry

1. INTRODUCTION

Line clipping is a basic and an important operation in computer graphics. There are many applications of line clipping. For example, line clipping is needed to extract a part of a given scene for viewing. Generally lines are clipped by using a region that includes the part of the given scene. It is known as the clipping window and it is a rectangle or a general polygon [2].

The traditional line clipping algorithms include Cohen-Sutherland line clipping algorithm [1], Liang-Barsky line clipping algorithm [2], Cyrus-Beck line clipping algorithm [3] and Nicholl-Lee-Nicholl line clipping algorithm [4]. The Cohen-Sutherland and the Liang-Barsky algorithms can be extended to three-dimensional clipping [1]. The Nicholl-Lee-Nicholl algorithm performs fewer comparisons and divisions. Therefore, it is faster than others [1]. The major disadvantage of this algorithm is that it can only be applied to two-dimensional clipping [1]. On the other hand, the Liang-Barsky and the Cohen-Sutherland methods are easily extended to three-dimensional scenes [1].

The Cohen-Sutherland line clipping algorithm is one of the earliest and most widely used line clipping methods [2]. In this algorithm, a rectangular clipping window along with a coding scheme is used to divide the space in to regions. Then, each end point of the line segment is assigned a region code according to the region which has been occupied by that point. Then the “AND” and “OR” operations are performed over the region codes of the end points to decide whether the line segment is inside the clipping window or outside the clipping window. This algorithm is very faster for simple situations such as line segment is completely inside or outside of the clipping window. When the line segment cannot be classified as completely inside or outside, the algorithm needs to be repeated several times to convert it in to a simple situation. So, if the line segment is intersecting more than two boundaries of the clipping window, lots of unnecessary computations are involved [1].

Cyrus and Beck have proposed another algorithm that deals with the parametric form of the line [3]. In order to clip a line segment which is neither vertical nor horizontal and lies entirely within the window, 12 additions, 16 subtractions, 20 multiplications and 4 divisions are required [2]. Besides, for the general case (the line segments crossing all the boundaries of the window), the algorithm first makes computations and finds the parameters of the intersection points. According to the signs of the denominators of the parameters, then it determines which parts of the line segment is outside of the window. This algorithm mainly suffers from the above mentioned limitations.

Nicholl-Lee-Nicholl line clipping algorithm makes four rays which pass an endpoint of the line segment and four vertices of the window, and creates three regions by the four rays. Then, the algorithm determines which region that the line segment lies in, and finds the intersections or rejects the line segment. Before finding the intersection points of the line segment and the window, the algorithm first determines the position of the first endpoint of the line segment for the nine possible regions relative to the clipping window. If the point is not in one of the three especial regions, the algorithm has to transform the point to one of the three especial regions. To find the region in which the other endpoint of the line segment lies, it compares the slope of the line segment to the slopes of the four rays. Because of that, for the algorithm, finding the intersection points are efficient, but finding the positions of the two endpoints of the line segment are more complicated than Cohen-Sutherland line clipping algorithm [2].

You-dong Liang, Brian A. Barsky and Mel Slater also introduced faster line clipping algorithm [1, 5]. This algorithm is based on a parametric representation of the line segment. It is somewhat complicated and inefficient. To clip a line segment which is neither vertical nor horizontal, it will perform 16 comparisons, 7 subtractions, and 4 divisions [2].

Vaclv Skala proposed a line clipping algorithm for convex polygon window [6]. The algorithm uses binary search to find the intersections in the clipping window. The complexity is O (lg N). For the rectangle window, the algorithm does not have an advantage in comparison with the Cyrus-Beck algorithm [2].

According to Hearn and Baker [1] any line clipping algorithm involves three steps:

1. Test a given line segment to check whether it lays completely inside the clipping window
2. If not check whether it lies completely outside the clipping window
3. Otherwise perform intersection calculations with one or more clipping boundaries

These three steps lead the algorithm to lot of calculations as explained above. In this paper, a new line clipping algorithm is proposed without using the traditional three step procedure. Instead the line segment is directly passed through the algorithm without any classification. It was tested for a large number of line segments and the results proved that the algorithm is more efficient than Cohen-Sutherland and Liang-Barksy algorithms.

2. METHODOLOGY

This section presents the proposed line clipping algorithm and analyzes its performance. For line clipping, a rectangular clipping window is considered. Following conventions have been used to label the rectangular window.

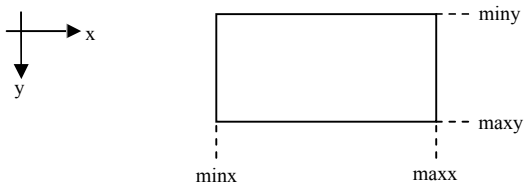


Fig.1. Clipping window

The general equation of a line, y = m * x + c, is used, where m is the gradient and c is the y-intercept. End points of the line segment are A = (x[0], y[0]) and B = (x[1], y[1]).

2.1 PSEUDO CODE OF THE PROPOSED ALGORITHM FOR 2D SPACE

All the symbols used in the following pseudo code are shown in Fig. 1. To increase the understandability of the pseudo code we have omitted following cases from it.

1) Line segment is just a point
2) Line segment is parallel to principle axes

Above two cases have been addressed at the implementation stage (See Appendix). Then the abstract pseudo code is as follows.

```
// Calculating m and c
For i = 0 to i = 1
        If x[i] < minx
                x[i] = minx;
                y[i] = m * minx + c;
        ElseIf x[i] > maxx
                x[i] = maxx;
                y[i] = m * maxx + c;
        EndIf
        If y[i] < miny
                x[i] = ( miny – c ) / m;
                y[i] = miny;
        ElseIf y[i] > maxy
                x[i] = ( maxy – c ) / m;
                y[i] = maxy;
        EndIf
EndFor
```

```
// Initial line is completely outside
If ( x[0] - x[1] < 1 ) AND ( x[1] - x[0] < 1)
        // Do nothing
Else
        // Save the line with end points ( x[0], y[0] ), ( x[1], y[1] )
EndIf
```

2.2 ANALYSIS OF THE ALGORITHM FOR 2D SPACE

This subsection analyzes the proposed algorithm for some of the most important test cases.

Case 1: Line is completely inside

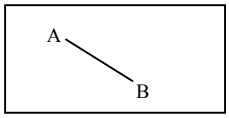


Fig.2.Line is completely inside

x[A] != x[B] → true
y[A] != y[B] → true
Consider point A,
x[A] < minx → false
x[A] > maxx → false
y[A] < miny → false
y[A] > maxy → false
Therefore, the initial position of A is not changed.
Consider point B,
x[B] < minx → false
x[B] > maxx → false
y[B] < miny → false
y[B] > maxy → false
Therefore the initial position of B is not changed.
(x[A] - x[B] < 1) && (x[B] - x[A]) → false
Therefore, the line with the end points A and B is drawn.

Case 2: Line is completely outside

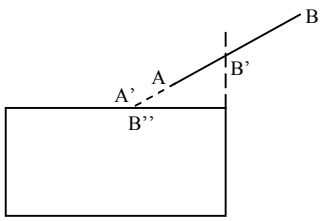


Fig.3. Line is completely outside

x[A] != x[B] → true
y[A] != y[B] → true
Consider point A,
x[A] < minx → false
x[A] > maxx → false
y[A] < miny → true
Therefore, A → A'
Consider point B,
x[B] < minx → false
x[B] > maxx → true
Therefore, B → B'
y[B'] < miny → true
Therefore, B' → B''
(x[A'] - x[B''] < 1) && (x[B''] - x[A']) → true
Therefore, the line is ignored.

Case 3: Line intersects the clipping window

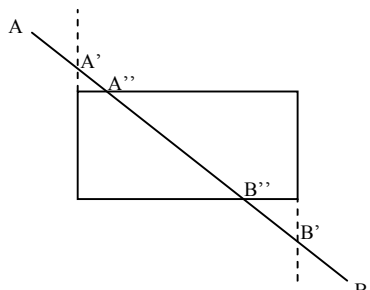


Fig.4. Line is intersecting the boundaries

x[A] != x[B] → true
y[A] != y[B] → true
Consider point A,
x[A] < minx → true
Therefore, A → A'
y[A'] < miny → true
Therefore, A' → A''
Consider point B,
x[B] < minx → false
x[B] > maxx → true
Therefore, B → B'
y[B'] < miny → false
y[B'] > maxy → true
Therefore, B' → B''
Therefore, the line with the end points A'' and B'' is drawn.

Case 4: Line partially inside the clipping window

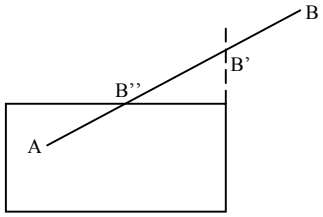
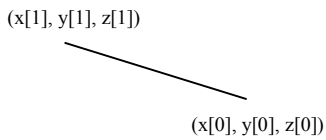


Fig.3. Line is partially inside the clipping window

x[A] != x[B] → true
y[A] != y[B] → true
Consider point A,
x[A] < minx → false
x[A] > maxx → false
y[A] < miny → false
y[A] > maxy → false
y[A] < miny → true
Therefore, the initial position of A is not changed.
Consider point B,
x[B] < minx → false
x[B] > maxx → true
Therefore, B → B'
x[B'] < miny → true
Therefore, B' → B''
Therefore, the line with the end points A and B'' is drawn.

2.3 PSEUDO CODE OF THE PROPOSED ALGORITHM FOR 3D SPACE

In this case an additional coordinate, Z coordinate, is involved. So the end points look like below.



$Equation\ of\ the\ line,$

$$\frac{x - x[0]}{l} = \frac{y - y[0]}{m} = \frac{z - z[0]}{n}; l, m, n\ are\ constants$$

$Substitute,\ (x[1], y[1], z[1])$

$$\frac{x[1] - x[0]}{l} = \frac{y[1] - y[0]}{m} = \frac{z[1] - z[0]}{n}$$

$Take,\quad \frac{x[1] - x[0]}{l} = \frac{y[1] - y[0]}{m}$

$$\frac{x[1]-x[0]}{y[1]-y[0]} = \frac{l}{m} = a$$

$$Take, \quad \frac{y[1]-y[0]}{m} = \frac{z[1]-z[0]}{n}$$

$$\frac{y[1]-y[0]}{z[1]-z[0]} = \frac{m}{n} = b$$

Point of intersection with x = p plane,

$$\left(p, \frac{p-x[0]}{a} + y[0], \frac{p-x[0]}{a*b} + z[0]\right)$$

Point of intersection with y = q plane,

$$\left(a*(q-y[0]) + x[0], q, \frac{q-y[0]}{b} + z[0]\right)$$

Point of intersection with z = r plane,

$$(a*b*(r-z[0]) + x[0], b*(r-z[0]) + y[0], r)$$

Pseudo code of the algorithm developed for the 2D space can be extended as follows. To increase the understandability of the pseudo code we have omitted following cases from it.

1) Line segment is just a point
2) Line segment is parallel to principle planes

Above two cases can be addressed at the implementation stage.

```
// Calculate a and b
For i = 0 to i = 1
        If x[i] < minx
                y[i] = ( minx - x[0] ) / a + y[0];
                z[i] = ( minx - x[0] ) / ( a * b ) + z[0];
                x[i] = minx;
        ElseIf x[i] > maxx
                y[i] = ( maxx - x[0] ) / a + y[0];
                z[i] = ( maxx - x[0] ) / ( a * b ) + z[0];
                x[i] = maxx;
        EndIf
        If y[i] < miny
                x[i] = a * ( miny - y[0] ) + x[0];
                z[i] = ( miny - y[0] ) / b + z[0];
                y[i] = miny;
        ElseIf y[i] > maxy
                x[i] = a * ( maxy - y[0] ) + x[0];
                z[i] = ( maxy - y[0] ) / b + z[0];
                y[i] = maxy;
        EndIf
        If z[i] < minz
```

```
            x[i] = a * b * ( minz - z[0] ) + x[0];
            y[i] = b * ( minz - z[0] ) + y[0];
            z[i] = minz;
        ElseIf z[i] > maxz
            x[i] = a * b * ( maxz - z[0] ) + x[0];
            y[i] = b * ( maxz - z[0] ) + y[0];
            z[i] = maxz;
        EndIf
EndFor

// Initial line is completely outside
If ( x[0] - x[1] < 1 ) AND ( x[1] - x[0] < 1 )
        // Do nothing
Else
        // Save the line with end points ( x[0], y[0], z[0] ), (x[1], y[1], z[1])
EndIf
```

3. RESULTS AND DISCUSSION

The proposed algorithm for the 2D space was tested for all the possible test cases. The test results indicated that it performs well in all possible situations. In order to validate the algorithm, it was compared against the Cohen-Sutherland and Liang-Barsky algorithms. The following hardware and software were used for testing.

Computer: Intel(R) Pentium(R) Dual CPU; E2180 @ 2.00 GHz; 2.00 GHz, 0.98 GB RAM IDE Details: Turbo C++; Version 3.0; Copyright(c) 1990, 1992 by Borland International, Inc.

Method: The clip window with values minx = miny = 100 and maxx = maxy = 300 was used for clipping. Random points were generated in the range 0-399 by using the randomize() function. These random points were considered as end points to generate random lines. Number of clock cycles taken by each algorithm to clip 10^8 random lines were counted using the clock() function [2]. The results are shown in Table 1.

Table.1. Number of clock cycles comparison of the proposed algorithm verses traditional algorithms

Step	Cohen-Sutherland	Liang-Barsky	Proposed algorithm
1	2596	2452	2296
2	2593	2452	2296
3	2594	2452	2296
4	2595	2452	2296
5	2596	2451	2296
6	2593	2451	2296
7	2593	2452	2296
8	2592	2451	2296
9	2593	2452	2296
10	2594	2451	2296

The results shown in Table 1 prove that the proposed algorithm is faster than both Cohen-Sutherland and Liang-Barsky algorithms. Average ratios can be calculated using following equation.

$$Average\ Ratio = \frac{\sum Clock\ cycles\ for\ traditional\ algorithm}{\sum Clock\ cycles\ for\ proposed\ algorithm}$$

Average Ratio (Cohen-Sutherland: Proposed) = 25939/22960=1.1297
Average Ration (Liang-Barsky: Proposed) = 24516/22960=1.0677

Therefore the proposed algorithm is 1.13 times faster than Cohen-Sutherland algorithm and 1.07 times faster than Liang-Barsky algorithm.

4. CONCLUSION

A new algorithm for line clipping in 2D space was introduced. Additionally possibility of expanding it for 3D space was discussed. Algorithm proposed for the 2D space was tested and compared against two well known algorithms. According to the test results, it is faster than the traditional algorithms. Therefore the proposed algorithm can be successfully used in applications where line clipping involved since its performance is better. Even the line segment is completely outside the proposed algorithm needs to calculate some of the intersection points. That is a disadvantage of the proposed method compared to the traditional two algorithms that were tested above. But in all the other cases the performance of the proposed algorithm is better.

REFERENCES

1. D. Hearn and M. P. Baker (1998), *Computer Graphics, C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River*, p. 224-248.
2. Wenjun Huang (2010), The Line Clipping Algorithm Basing on Affine Transformation, *Intelligent Information Management, 2,380-385*, Published Online June 2010 (http://www.SciRP.org/journal/iim)
3. M. Cyrus and J. Beck (1978), Generalized Two and Three Dimensional Clipping, *Computers and Graphics, Vol. 3, No. 1*, pp. 23-28.
4. T. M. Nicholl, D. T. Lee and R. A. Nicholl (1987), An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis, *Computers and Graphics, Vol. 21, No. 4*, pp. 253-262.
5. C. B. Chen and F. Lu (2006), Computer Graphics Basis, *Publishing House of Electronics Industry, Beijing*, pp.167-168.
6. V. Skala (1994), O (lg N) Line clipping Algorithm in E, *Computers and Graphics, Vol. 18, No. 4*, pp. 517-527.

APPENDIX

IMPLEMENTATION OF THE PROPOSED ALGORITHM FOR 2D SPACE

Implementation of the proposed algorithm for the 2D window has been developed using C++ programming language. Here all the test cases have been considered. The source code is as follows.

```
void clipMY(double x[],double y[],double minx,double miny,double maxx,double maxy)
{
// gradient and y-intercept of the line
double m,c;
int i;

// non vertical lines
if(x[0]!=x[1])
{
	// non vertical and non horizontal lines
	if(y[0]!=y[1])
	{
		// calculate the gradient
		m=(y[0]-y[1])/(x[0]-x[1]);
		// calculate the y-intercept
		c=(x[0]*y[1]-x[1]*y[0])/(x[0]-x[1]);

		for(i=0;i<2;i++)
		{
			if(x[i]<minx)
			{
				x[i]=minx;
				y[i]=m*minx+c;
			}
			else if(x[i]>maxx)
			{
				x[i]=maxx;
				y[i]=m*maxx+c;
			}
			if(y[i]<miny)
			{
				x[i]=(miny-c)/m;
				y[i]=miny;
			}
			else if(y[i]>maxy)
			{
```

```
                x[i]=(maxy-c)/m;
                y[i]=maxy;
            }
        }
        // initial line is completely outside
        if((x[0]-x[1]<1) && (x[1]-x[0]<1))
        {
            // do nothing
        }
        // draw the clipped line
        else
        {
            setcolor(15);
            line(x[0],y[0],x[1],y[1]);
        }
    }
    // horizontal lines
    else
    {
        // initial line is completely outside
        if((y[0]<=miny) || (y[0]>=maxy))
        {
            // do nothing
        }
        else
        {
            for(i=0;i<2;i++)
            {
                if(x[i]<minx)
                {
                    x[i]=minx;
                }
                else if(x[i]>maxx)
                {
                    x[i]=maxx;
                }
            }
            // initial line is completely outside
            if((x[0]-x[1]<1) && (x[1]-x[0]<1))
            {
                // do nothing
            }
            // draw the clipped line
            else
```

```
                        {
                                setcolor(15);
                                line(x[0],y[0],x[1],y[1]);
                        }
                }
        }

}
// vertical lines
else
{
        // initial line is just a point
        if(y[0]==y[1])
        {
                // initial point is outside
                if((y[0]<=miny) || (y[0]>=maxy))
                {
                        // do nothing
                }
                // initial point is outside
                else if((x[0]<=minx) || (x[0]>=maxx))
                {
                        // do nothing
                }
                // initial point is inside
                else
                {
                        putpixel(x[0],y[0],15);
                }
        }
        // initial line is completely outside
        else if((x[0]<=minx) || (x[0]>=maxx))
        {
                // do nothing
        }
        else
        {
                for(i=0;i<2;i++)
                {
                        if(y[i]<miny)
                        {
                                y[i]=miny;
                        }
                        else if(y[i]>maxy)
```

```
                {
                        y[i]=maxy;
                }
        }
        // initial line is completely outside
        if((y[0]-y[1]<1) && (y[1]-y[0]<1))
        {
                // do nothing
        }
        // draw the clipped line
        else
        {
                setcolor(15);
                line(x[0],y[0],x[1],y[1]);
        }
    }
}

}
```

Saluka Ranasinghe Kodituwakku is an associate professor at the Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya, Sri Lanka. His research interests include Database Systems, Distributed Computing, Role Based Access Control Systems, and Software Engineering. *Email:* salukak@pdn.ac.lk

Kasun Ranga Wijeweera is an undergraduate following a Computer Science Special degree at the Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya, Sri Lanka. His research interests include Computational Geometry, Computer Graphics, Image Processing, Computer Vision, and Artificial Intelligence. *Email:* krw19870829@gmail.com

M. A. Pathum Chamikara is working as a research assistant at the Post Graduate Institute of Science (PGIS), University of Peradeniya, Sri Lanka. He received his BSc (Special) degree in Computer Science from University of Peradeniya, Sri Lanka (2010). His research interests include Crime Analysis, GIS (Geographic Information Systems), Image Processing, Computer Vision, and Artificial Intelligence. *Email:* pathumchamikara@gmail.com

Note: The above paper has been published as,

S. R. Kodituwakku, K. R. Wijeweera, M. A. P. Chamikara, *An Efficient Algorithm for Line Clipping in Computer Graphics Programming*, Ceylon Journal of Science (Physical Sciences) 17 (2013) 1-7.