# COMPARISON OF VARIOUS LINE CLIPPIN ALGORITHMS

Htwe Htwe Aung

*Lecturer, Faculty of Computer Science, University of Computer Studies, Pathein, Myanmar*

## ABSTRACT

*Line clipping process often requires repeating clipping algorithm. One method for improving the efficiency of a line clipping algorithm is to reduce the repetition of algorithm. An efficient clipping algorithm is presented here to acquire this goal. The paper exhibit various line clipping algorithms on the basis of their working principles. In this paper a comparison is made for different line clipping algorithms used in computation. One algorithm is region codes are used to identify the position of line. One algorithm reduces intersection calculations. Other one is based on testing XY plane to reduce intersection calculation. Which is the best suited line clipping algorithm can only be decided by comparing the available algorithms in different aspects.*

**Keyword:** *Line Clipping Algorithms, Cohen-Sutherland, Liang-Barsky, Nicholl-Lee-Nicholl, Comparative.*

## 1. INTRODUCTION

Clipping is one of the fundamental problems of the computational geometry with applications in various areas of computer graphics, visualization and computer-aided design system. Any procedure, which identifies those portions of picture that are either inside or outside of the specified region of the space is referred to as a clipping algorithm. The region against which an object is to clip is called a clip window. Clipping, which is a basic operation to several aspects of computer graphics, include two elements: the clipping window which could be rectangle, circle, convex window, concave window or open window, and the object to be clipped which could be line, polygon, circle, character or irregular curves. Different combination of the two elements and different clipping strategies lead to various algorithm. Among all algorithms, line clipping against the rectangle window receives special attentions [5]. The traditional line clipping algorithms include Cohen-Sutherland line clipping algorithm [6], Liang-Barsky line clipping algorithm [7], Cyrus-Beck line clipping algorithm [2] and Nicholl-Lee-Nicholl line clipping algorithm [9]. The major disadvantage of this algorithm is that it can only be applied to two-dimensional clipping [7]. Until recently, most works are concentrated on accelerating the intersection calculation so as to improve the clipping efficiency [11][13][14].

## 2. LINE CLIPPING USING RECTANGULAR WINDOW

A line clipping procedure involves several parts. First, test a given line segment to determine whether it lays completely inside the clipping window. If it does not, try to determine whether it lies completely outside the window. Finally, if cannot identify a line as completely inside or completely outside, then, perform intersection calculations with one or more clipping boundaries and process lines through the "inside-outside" tests by checking the line endpoints [7].

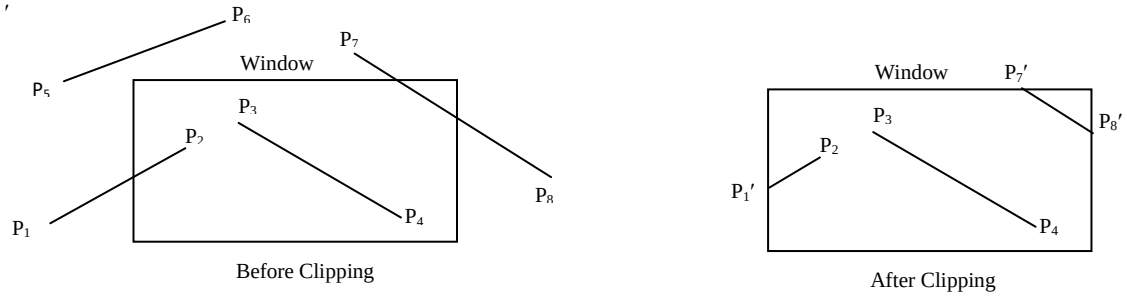


**Fig -1**. Line clipping against a rectangular clip window

A line with both endpoints inside all clipping boundaries, such as the line from $P_3$ to $P_4$, is saved. A line with both endpoints outside any one of the clip boundaries (line $P_5P_6$) is outside the window. All other lines cross one or

more clipping boundaries, and may require calculation of multiple intersection points. To minimize calculations, try to devise clipping algorithms that can efficiently identify outside lines and reduce intersection calculations [7][12].

For a line segment with endpoints $(x_1, y_1)$ and $(x_2. y_2)$ and one or both endpoints outside the clipping rectangle, the parametric representation

$$x = x_1 + u(x_2 - x_1)$$
$$y = y_1 + u(y_2 - y_1), \qquad 0 \leq u \leq 1$$

could be used to determine values of parameter u for intersections with the clipping boundary coordinates. If the value of u for an intersection with a rectangle boundary edge is outside the range 0 to 1, the line does not enter the interior of the window that boundary. If the value of u is within the range from 0 to 1, the line segment does indeed cross into the clipping area. This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed. Line segments that are parallel to window edges can be handled as specia1 cases. Clipping line segments with these parametric tests require a good deal of computation, and faster approaches to clipping are possible. A number of efficient line clippers have been developed and some algorithms are designed explicitly for two-dimensional pictures and some are easily adapted to three-dimensional applications [1][7].

### 2.1 Cohen-Sutherland Line Clipping

This is one of the oldest and most popular line clipping procedures. Generally, the method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must he calculated. Every line endpoint in a picture is assigned a four-digit binary code, called a region code, which identifies the location of the point relative to the boundaries of the clipping rectangle. Regions are set up in reference to the boundaries [7].

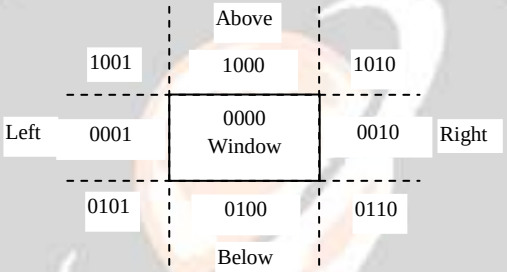


**Fig -1:** Four binary region codes with respect to the clipping rectangle.

Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom. By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as bit 1: left, bit 2: right, bit 3: below, bit 4: above [8].

A value of 1 in any bit position indicates that the point is in that relative position; otherwise, the bit position is set to 0. If a point is within the clipping rectangle, the region code is 0000. A point that is below and to the left of the rectangle has a region code of 0101. Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries. Bit 1 is set to 1 if $x < xw_{min}$. The other three-bit values can be determined using similar comparisons. For languages in which bit manipulation is possible, region-code bit values can be determined with the following two steps:

(1) Calculate differences between endpoint coordinates and clipping boundaries.

(2) Use the resultant sign bit of each difference calculation to set the corresponding value in the region code. Bit 1 is the sign bit of $x - xw_{min}$; bit 2 is the sign bit of $xw_{max} - x$; bit 3 is the sign bit of $y - yw_{min}$ and bit 4 is the sign bit of $yw_{max} - y$ [8].

The steps of clipping lines using the Cohen-Sutherland algorithm are following:

- Starting with the endpoint of the line from $p_1$ to $p_2$, checked $p_1$ against the left, right, below, and above boundaries in turn and find that this point is below the clipping rectangle.
- Then find the intersection point $p_1'$ with the below boundary and discard the line section from $p_1$ to $p_1'$.
- The line has been reduced to the section from $p_1'$ to $p_2$.
- Since $p_2$ is outside the clip window, check this endpoint against the boundaries and find that it is to the left and above of the window and therefore, intersection point $p_2'$ is calculated.
- But this point is above the window, so the final intersection calculation yields $p_2''$ and the line from $p_1'$ to $p_2''$ is saved [4].

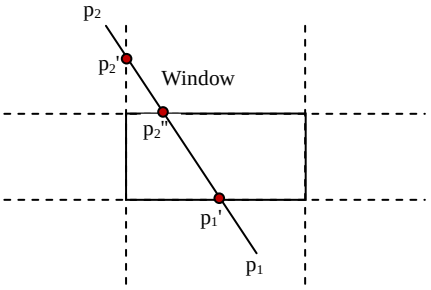


**Fig-2:** Line extending from one coordinate region to another.

Intersection points with a clipping boundary can be calculated using the slop-intercept form of the line equation. For a line with endpoint coordinates($x_1$, $y_1$) and ($x_2$, $y_2$), the y coordinate of the intersection point with a vertical boundary can be obtained with the calculation

$$y = y_1 + m(x - x_1) \quad (1)$$

Where the x value is set either to $xw_{min}$ or to $xw_{max}$, and the slope of the line is calculated as

$$m = (y_2 - y_1) / (x_2 - x_1).$$

For the intersection with a horizontal boundary, the x coordinate can be calculated as

$$x = x_1 + (y - y_1) / m \quad (2)$$

with y set either to $yw_{min}$ or to $yw_{max}$ [1][7].

The pseudo code of Cohen-Sutherland line clipping algorithm is as follows:

1. Assign a region code for two endpoints of given line.
2. If both endpoints have a region code 0000 then given line is completely inside.
3. Else, perform the logical AND operation for both region codes.
   i. If the result is not 0000, then given line is completely outside.
   ii. Else line is partially inside.
   - Choose an endpoint of the line that is outside the given rectangle.
   - Find the intersection point of the rectangular boundary (based on region code).
   - Replace endpoint with the intersection point and update the region code.
   - Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
4. Repeat step 1 for other lines

### 2.2. Liang-Barsky Line Clipping

Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment in the form

$$x = x_1 + r * \Delta x$$
$$y = y_1 + r * \Delta y, \quad 0 \leq r \leq 1$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

For a point (x,y) inside the clipping window,

$$xw_{min} \leq x_1 + r * \Delta x \leq xw_{max}$$
$$yw_{min} \leq y_1 + r * \Delta y \leq yw_{max}$$

These four inequalities can be expressed as

$$r * p_j \leq q_j, \quad j = 1, 2, 3, 4$$

Where parameter p and q are defined as

$p_1 = -\Delta x$, $q_1 = x_1 - xw_{min}$ (left)
$p_2 = \Delta x$, $q_2 = xw_{max} - x_1$ (right)
$p_3 = -\Delta y$, $q_3 = y_1 - yw_{min}$ (below)
$p_4 = \Delta y$, $q_4 = yw_{max} - y_1$ (above)

From the above definitions of parameters, the following observations can be easily made.

- If $p_j = 0$, the line is parallel to one of the clipping boundaries corresponding correspinding the value of j.
- If $q_j < 0$, the line is completely outside the boundary and can be eliminated.
- If $q_j > 0$ the line is inside the boundary and need further consideration.
- If $p_j < 0$, the line proceeds from the outside to inside of the corresponding boundary line.
- If $p_j > 0$, the line proceeds from the inside to outside of the corresponding boundary line.
- If $p_j \neq 0$, calculate the value of r that corresponds to the point where the infinitely extended line intersects the extension of boundary j as

$$r = q_j / p_j$$

The Liang-Barsky algorithm for finding the visible portion of the line, if any, can be stated as a four step process:

(i) If $p_j = 0$ and $q_j < 0$, for any j, eliminates the line and stop otherwise proceed to next step.

(ii) For all j such that $p_j < 0$ calculate $u_j = q_j / p_j$. Let $r_1$ be the maximum of the set containing 0 and the various values of r.

(iii) For all j such that $p_j > 0$ calculate $u_j = q_j / p_j$. Let $r_2$ be the minimum of the set containing 1 and the calculated u value.

(iv) If $r_1 > r_2$, eliminate the line because it is completely outside the clipping window otherwise, use and to calculate the endpoints of the clipped line [1][7].

The algorithm of Liang-Barsky line clipping is as follows:

1. Read two endpoints of the line say $p_1 (x_1, y_1)$ and $p_2 (x_2, y_2)$.
2. Read two corners left-top ( $xw_{min}$, $yw_{max}$) and right-bottom ($xw_{max}$, $yw_{min}$) of the window.
3. Calculate the values of parameters $p_j$ and $q_j$ for

$p_1 = -\Delta x$, $q_1 = x_1 - xw_{min}$
$p_2 = \Delta x$, $q_2 = xw_{max} - x_1$
$p_3 = -\Delta y$, $q_3 = y_1 - yw_{min}$
$p_4 = \Delta y$, $q_4 = yw_{max} - y_1$

4. If $p_j = 0$, then
   { The line is parallel to $j^{th}$ boundary.
   If $q_j < 0$ then
   { line is completely outside the boundary, discard the line segment and goto stop.
   }
   else
   { Check whether the line is horizontal or vertical and accordingly check the line endpoint with corresponding boundaries. If line endpoint(s) lie within the bounded area then use them to draw line otherwise use boundary coordinates to draw line. Go to stop.
   }
5. Initialise values for $r_1$ and $r_2$ as
   $r_1 = 0$ and $r_2 = 1$
6. Calculate values for $q_j / p_j$ for j= 1, 2, 3, 4.
7. Select values of $q_j / p_j$ where $p_j < 0$ and and assign maximum out of them as $r_1$.
8. Select values of $q_j / p_j$ where $p_j > 0$ and and assign manimum out of them as $r_2$.
9. If ($r_1 < r_2$)
   { Calculate the endpoints of the clipped line as follows:
   $xx_1 = x_1 + r_1 \Delta x$
   $xx_2 = x_1 + r_2 \Delta x$

   $yy_1 = y_1 + r_1 \Delta y$
   $yy_2 = y_1 + r_2 \Delta y$
   Draw line ($xx_1$, $xx_2$, $yy_1$, $yy_2$ )
   }
10. Stop.

**2.3. Nicholl-Lee-Nicholl Line Clipping**

Nicholl-Lee-Nicholl (or NLN) algorithm avoids multiple clipping of an individual line segment by creating more regions around the clip window. NLN 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 (in the clip window, edge & corner), the algorithm has to transform the point to one of the three especial regions [12].

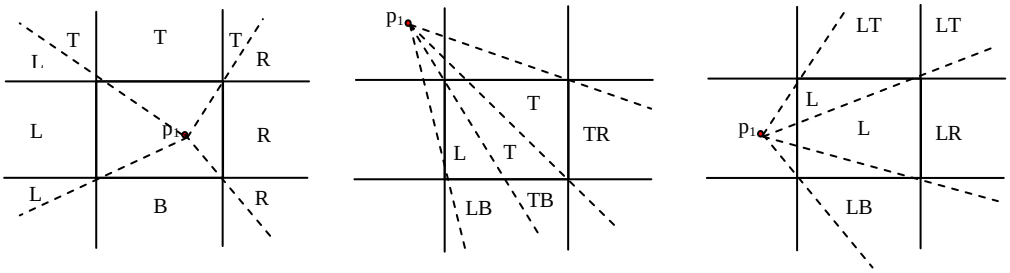


**Fig-3:** Four clipping regions used in NLN algorithm.

The Algorithm of edge region case of the Nicholl-Lee-Nicholl line-clipping algorithm is as below:

```
procedure LeftEdgeRegionCase (ref real x1, y1, x2, y2; ref boolean visible)
begin
    real dx, dy;
    if x2 < xmin
      then visible = false;
      else if y2 < ymin
          then LeftBottom (xmin, ymin, xmax, ymax, x1, y1, x2, y2, visible)
          else if y2 > ymax
              then
                begin                          {Use symmetry to reduce to LeftBottom case}
                  y1 = – y1;   y2 = – y2;      {reflect about x-axis}
                  LeftBottom (xmin, -ymax, xmax, -ymin, x1, y1, x2, y2, visible);
                  y1 = – y1;   y2 = – y2;      {reflect back}
                end
              else
                begin
                  dx = x2 – x1;          dy = y2 – y1;
                  if x2 > xmax then
                    begin
                      y2 = y1 + dy * (xmax – x1) / dx;
                      x2 = xmax;
                    end;
                  y1 = y1 + dy * (xmin – x1) / dx;
                  x1 = xmin;
                  visible = true;
                end
end
procedure LeftBottom (real xmin, ymin, xmax, ymax; ref real x1, y1, x2, y2; ref boolean visible)
begin
    real dx, dy, a, b, c;
    dx = x2 – x1;     dy = y2 – y1;
    a = (xmin – x1)* dy;          b = (ymin – y1)* dx;
    if b > a
      then visible = false {(x2, y2) is below ray from (x1, y1) to bottom left corner}
      else
        begin
          visible = ture;
          if x2 < xmax
            then
              begin
                x2 = x1 + b / dy;
                y2 = ymin;
              end
            else
              begin
                c = (xmax – x1) * dy;
                if b > c
                  then {(x2, y2) is between rays from (x1, y1) to bottom left and right corner}
                    begin
```

```
                    x2 = x1 + b / dy;
                    y2  = ymin;
                  end
                else
                  begin
                    y2 = y1 + c / dx;
                    x2 = xmax;
                  end
            end;
        end;
    y1 = y1 + a / dx;
    x1 = xmin;
end;
```

## 3. COMPARATIVE STUDY AND DISCUSSION

Liang-Barsky algorithm is intersection calculations can be reduced, so more efficient than the Cohen-Sutherland algorithm. Each update of parameters $r_1$ and $r_2$ requires only one division and window intersections of the line are computed only once, when the final values of $r_1$ and $r_2$ have been computed[15]. In contrast, the Cohen-Sutherland algorithm can repeatedly calculate intersections along a line path, even though the line may be completely outside the clip window. And, each intersection calculation requires both a division and a multiplication [7].

Although Nicholl-Lee-Nicholl algorithm achieves lesser comparisons and divisions which make it faster than others it is difficult to expand for three-dimensional clipping [10]. The major disadvantage of this algorithm is that it can only be applied to two-dimensional clipping scenes but Liahg-Barsky and Cohen-Sutherland methods are easily extended to three-dimensional scenes. In the Cohen-Sutherland method, for example, multiple intersections may be calculated along the path of a single line before an intersection on the clipping rectangle is located or the line is completely rejected. These extra intersection calculations are eliminated in the NLN algorithm by carrying out more regions testing before intersection positions are calculated. Compared to both the Cohen-Sutherland and the Liang-Barsky algorithms, the Nicholl-Lee-Nicholl algorithm performs fewer comparisons and divisions [7].

## 4. CONCLUSIONS

Clipping is indispensable technique in computer graphics, and as such it has been studied very extensively in the past. Nowadays, the problem is very often considered as solved in many aspects, that this is indeed true. The fundamental problem in algorithm design is to use all known data properties as much as possible in order to get an algorithm with better efficiency. If this factor is used in algorithm design, it is possible not only to improve algorithms property but sometimes also to vary the algorithms complexity. This argument is proved very clearly in this paper by analyzing three algorithms for line clipping. Generally, to improve a better algorithm for the given problem, need to consider the following issues:

- the trade-off between run-time memory and preprocessing (time or memory) complexities,
- what kind of preprocessing complexity is need to faster solution of the given problem,
- the use of pre-processing or parallel and distributed.

The presented algorithms proved that applying these approaches can bring a significant speed-up even with the known algorithms. In today, need to outmatch with large amounts of available data not only to visualize, but also to make it as realistic and as fast as possible. Therefore, any improvement in solving the line clipping algorithm is always welcome.

## 5. REFERENCES

[1] Abhishek Pandey, Swati Jain, Takshshila Institute of Engineering and Technology, International Journal of Modern Engineering Research (2013) pp-69-74.

[2] Cyrus M. and Beck J. (1978) Generalized Two- and Three-Dimensional Clipping, Computers and Graphics 3 (1): 23-28.

[3] Day JD. A new two-dimensional ling clipping algorithm for small windows, Computer Graphics Forum 1992; 11(4): 241-5.

[4] Donald Hearn, and M. Pauline Baker, Computer Graphics, C Version, 3 edition, pp. 226-230, December 2004.

[5] Goudong Lu*, Xuanhui Wu, Qunsheng Peng, An efficient line clipping algorithm-based o adaptive line rejection, Computers & Graphics 26 (2002) 409-415.

[6] Hearn, D. and Baker, M. P. (1998) Computer Graphics, C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, p. 224-248.

[7] Huang, W. (2010) The Line Clipping Algorithm Basing on Affine Transformation, Intelligent Information Management, 2,380-2385, (http://www. SciRP.org /journal/iim).

[8] Mohammad Saber Iraji, H. Motameni, Ayda Mazandarani, An Efficient Line Clipping Algorithm based on Cohen-Sutherland Line Clipping Algorithm, American Journal of Scientific Research (2011), pp.65-71.

[9] Nicholl, T. M. Lee, D. T. and Nicholl, R. A. (1987) An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, Computers and Graphics 21(4): 253-262.

[10] R. Kodituwakku, K. R. Wijeweera, M. A. P. Chamikara, An Efficient Line Clipping Algorithm for 3D Space, International Journal of Advanced Research in Computer Science and Software Engineering Volume 2, Issue 5, May 2012.

[11] Sharma NC, Manohar S. Line clipping revisited: two efficient algorithms based on simple geometric observations. Computers and Graphics 1992; 16(1): 51–4.

[12] S. R. Kodituwakku1, 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), pp.1-7.

[13] Wang Haohong, Wu Ruixun, Cai Shijie. A new efficient clipping algorithm basedon geometric transformation, Journal of Software 1998; 9(10): 728–33 (in Chinese).

[14] Wang Jun, Liang Youdong, Peng Qunsheng. A 2-D lineclipping with the least arithmetic operations, Chinese Journal of Computers 1991 ;( 7): 495–504(in chinese).

[15] Vaclav Skala, Pavel Lederbush, A comparison of a new O(1) and the Cyrus-Beck line clipping algorithms in E, COMPUGRAPHICS 96 Int.Conf., Paris, 1996.