International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.3, Issue.1, Jan-Feb. 2013 pp-69-74 ISSN: 2249-6645

Comparison of Various Line Clipping Algorithm for Improvement

Abhishek Pandey[1], Swati Jain[2]
[1, 2](Department: C.S.E., College Name: Takshshila Institute of Engineering and Technology, Country Name: India)

ABSTRACT: *Demonstration of various Line clipping algorithms on the basis of their working principles. One way for improving the efficiency of a line clipping algorithm is to reduce the repetition of algorithm. In this region codes are used to identify the position of line. One algorithm reduces intersection calculations. An efficient clipping algorithm is presented here to achieve this goal. One is based on testing xy plane to reduce intersection calculation. Our algorithm with reducing the confluence point can avoid the repetition of algorithm.*

Keywords: *Line Clipping Algorithms, LB, CSL, NLN, Computer graphics.*

I. INTRODUCTION

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 algorithm can be applied in world coordinates so that only the contents of t6he window interior are mapped to the device coordinates. There are five primitive types clipping, such as point, line, polygon or are, curve and text clipping.
Classical line clipping algorithms includes Cohen–Sutherland algorithm, Midpoint Subdivision algorithm, Liang Bearsky and Nicholl-Lee-Nicholl algorithm.

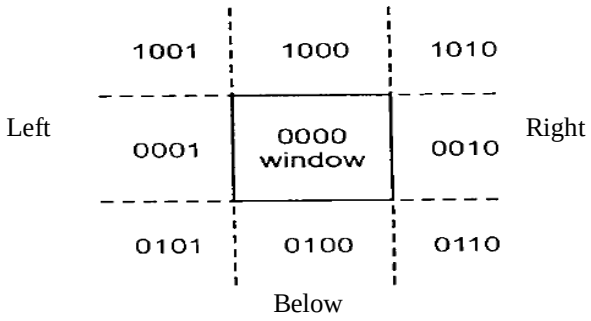
Until recently, most works are concentrated on accelerating the intersection calculation so as to improve the clipping efficiency [Sha92a, Day92a, Wan98a, Wan91a].

II. COHEN SUTHERLAND CLIPPING OVERVIEW

In computer graphics CSL algorithm is a line clipping algorithm. The algorithm divides the 2D space into 9 parts, using the infinite extension of four linear boundaries of the window. Assign a bit pattern to each region as shown in figure.

CSL algorithm reputedly calculates intersections along a line path, even thought the line may be completely outside the clip window and each calculation requires both a division and a multiplication.

Figure 1.1 Four bits code for nine region.

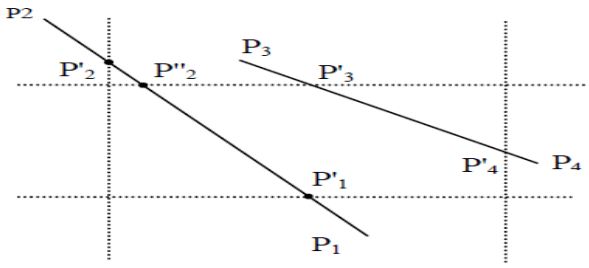


The numbers in the figure above are called out codes. An out codes are computed for each of the two points in the line. The bits in the out codes represent: TOP, BOTTOM, RIGHT, and LEFT. Each bit in the code is set to 1. If the region is to the right of the window, the second bit of the code is set to 1. If the Bottom, third bit is set and if the Top, the fourth bit is set. The four bits in the code the identify each of the nine region.

Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries. As shown in fig.2, such lines may or may not cross into the window interior. We begin the clipping process for a line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the line is checked against to other boundaries, and we continue until either the line is totally discarded or a section is found outside the window. We set up our algorithm to checked line endpoints against clipping boundaries in the order Left, Right, Bottom, and Top.

To illustrate the specific steps in clipping lines using the Cohen – Sutherland algorithm, we show how the lines in Fig. 2 could be processed. Starting with the Bottom endpoint of the line from p1 to p2, we checked p1 against the Left, Right, Bottom, and above boundaries in turn and find that this point is below the clipping rectangle. We then find the intersection point p'1 with the Bottom boundary and discard the line section from p1 to p'1. The line now has been reduced to the section from p'1 to p2. Since p2 is outside the clip window, we check this endpoint against the boundaries and find that it is to the Left and above of the window. 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 [Don04b].

Figure 2.1 Lines 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(x1, y1) and (x2, y2), the y coordinate of the intersection point with a vertical boundary can be obtained with the calculation

$$y = y1 + m(x – x1) \quad (1)$$

Where the x value is set either to xwmin or to xwmax, and the slop of the line is calculated as

$$m = (y2 – y1) / (x2 – x1).$$

Similarly, if we are looking for the intersection with a horizontal boundary, the x coordinate can be calculated as

$$x = x1 + (y – y1) / m \quad (2)$$

With y set either to ywmin or to ywmax.

The following procedure demonstrates the Cohen-Sutherland line-clipping algorithm. Codes for each endpoints are stored as bytes and processed using bit manipulations.

```
#define ROUND (a) ((int) (a+0.5))
/* Bits masks encode a point's position relative to the clip EGs. A points status is encoded by OR'ing together appropriate bit masks.*/
#define Left_EG 0x1
#define Right_EG 0x2
#define Bottom_EG 0x4
#define Top_EG 0x8
/* points encoded as 0000 are completely inside the clip rectangle; All others are outside at least one EG.if OR'ing two codes is FALSE (no bits are set in either code), the line can be Accepted. If the AND operation between two codes is true, the line defined by those endpoints is completely outside the clip region and can be Rejected.*/
*/
#define INSIDE (a) (! a)
#define REJECT (a,b) (a&b)
#define ACCEPT (a,b) (!(a|b))
#define FALSE 0
#define TRUE 1
Unsigned char encode (wcpt2 pt,dcpt winmin, dcpt winmax)
{
        if (pt.x < winmin.x)
                Code = code | Left_EG;
        if (pt.x > WinMax.x)
                Code = code | Right_EG;
        if (pt.y < WinMin.y)
                Code = code | Bottom_EG;
        if (pt.y > WinMax.y)
                Code = code | Top_EG;
                Code = code | Top_EG;
        Return (code);
}
void SwapPoints (wcpt2 *p1, wcpt2 *p2)
{
        Wcpt2 tmp;
        tmp = *p1; *p1 = *p2; *p2 = tmp;
}
void SwapCodes (unsigned char *c1, unsigned
{
        unsigned char tmp;
        tmp = *c1; *c1 = *c2; *c2 = tmp;
}
```

```
void ClipLine (dcpt WinMin, dcpt WinMax, wcpt2 p1, wcpt2 p2)
{
        Unsigned char code1, code2;
         int done= FALSE, draw = FALSE;
         Float m;
        while (!done)
        {
           code1 = encode (p1, WinMin, WinMax);
           code2 = encode (py, WinMin, WinMax);
           if (ACCEPT (code1, code2))
          {
                Done = TRUE;
                Draw = TRUE;
           }
           Else if (REJECT (code1, code2))
           Done = FALSE;
           Else {
           /*ensure that p1 is outside window*/
           If (INSIDE (code1)) {
           SwapPts(&p1, &p2);
           SwapCodes(&code1, &code2); }
          /* use slope (m) to find line_clip EG Intersection */
          If (p2.x!= p1.x)
                m = (p2.y - p1.y) / (p2.x - p1.x);
          If (code1 & Left_EG)
          {
                p1.y += (WinMin.x - p1.x) * m;
                p1.x = WinMin.x;
          }
          elseif (code1 & Right_EG)
        {
        p1.y += (WinMax.x - p1) * m;
        p1.x = WinMax.x;
  }
  elseif (code1 & Bottom_EG)
  {
     if (p2.x != p1.x)
        p1.x += (WinMin.y – p1.y) / m;
        p1.y = WinMin.y ;
  }
  elseif (code1 & Top_EG)
  {
        if (p2.x != p1.x)
                p1.x += (WinMax.y – p1.y)/m;
                p1.y = WinMax.y ;
         }
     }
}
if (draw)
lineDDA (ROUND(p1.x), ROUND(p1.y),
ROUND (p2.x), ROUND(p2.y));
}
```

III. LIANG- BERSKY CLIPPING ALGORITHM

Liang-bersky uses a parametric line representation to set up a more efficient line clipping procedure that reduces intersection calculations. In general, the LB algorithm is more efficient tan the CSL algorithm since intersection calculation is reduced. Each updates of parameters u1 and u2 requires only one division and window intersection of the line are computed only once when the final values of u1 and u2 have been computed.

We can write parametric equation of a line segment is in the form

$$x = x_1 + u\Delta x$$

$$y = y_1 + u\Delta y$$

where $0 \le u \le 1$ and $\Delta x = x_2 - x_1 \quad \Delta y = y_2 - y_1$,

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

$x\omega_{min} \leq x_1 + \Delta x.u \leq x\omega_{max}$
$y\omega_{min} \leq y_1 + \Delta y.u \leq y\omega_{max}$
Rewrite these four inequalities as
$p_k u \leq q_k \quad k = 1,2,3.$
Where parameter p and q are defined as
$p_1 = -\Delta x \quad q_1 = x_1 - x\omega_{min}$ (Left)
$p_2 = \Delta x \quad q_2 = x\omega_{max} - x_1$ (Right)
$p_3 = -\Delta y \quad q_3 = y_1 - y\omega_{min}$ (Bottom)
$p_4 = \Delta y \quad q_4 = y\omega_{max} - y_1$ (Top)
Observe the following facts:
If $p_k = 0$, the line is parallel to the corresponding boundary and (a) if $q_k < 0$, the line is completely outside the boundary and can be eliminated.(b) if $q_k > 0$, the line is inside the boundary and need further consideration.
If $p_k < 0$, the extended line proceeds from the outside to inside of the corresponding boundary line.
If $p_k > 0$, the extended line proceeds from the inside to outside of the corresponding boundary line.
If $p_k \neq 0$, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of boundary k as

$$u = \frac{q_k}{p_k}$$

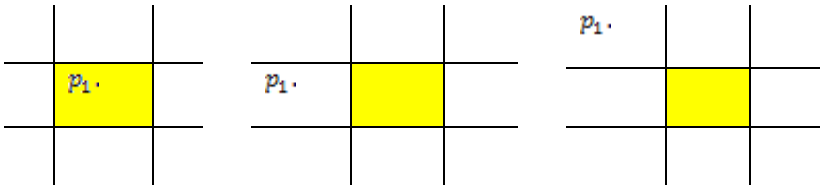
The LB algorithm for finding the visible portion of the line, if any, can be stated as a 4 step process:
(a) If $p_k = 0$ and $q_k < 0$, for any k, eliminates the line and stop otherwise proceed to next step.
(b) For all k such that $p_k < 0$ calculate $r_k = q_k/p_k$. Let u_1 be the maximum of the set containing 0 and the various values of r.
(c) For all k such that $p_k > 0$ calculate $r_k = q_k/p_k$. Let u_2 be the minimum of the set containing 1 and the calculated r value.
(d) If $u_1 > u_2$, eliminate the; line since it is completely outside the clipping window otherwise, use u_1 and u_2 to calculate the endpoints of the clipped line.

IV. NICHOLL LEE NICHOLL CLIPPING ALGORITHM

NLN method uses more regions testing in the xy plane to reduce intersection calculations. The extra intersection calculation eliminated in the NLN algorithm by carrying out more region testing before intersection position is calculated.

It uses symmetry to categorize endpoints into one of 3 regions (in the clip window, edge & corner). Then according to end point category it cheeks several clip cases. For a line with endpoint p_1 and p_2 , we first determine the position of point p_1 for the nine possible region relative to the clipping rectangle. Only the three region shown below need be considered.

Figure 4.1 NLN clipping window.



(a) $p_1$in the clip window (b) $p_1$in an edge region (c) $p_1$in corner region

If $p_1$lies in any one of the other six region, we can move it to one of the three regions using a symmetry transformation. For example, the region directly above the clip window can be transformed to the region left of the clip window using a reflection about the line y=-x or we could use a 90° counter clockwise rotation. Next, we find out the position of p_2 relative to p_1. To do this, we create some new regions in the plane, depending on the location of p_1. Boundaries of the new region are line segments that start at the position of $p_1$and pass through the window corners.

V. IMPROVED CSL CLIPPING ALGORITHM

As before said our algorithm is based on Cohen–Sutherland algorithm whereas we try to be efficient. In our new algorithm, Firstly, we apply Encoding & Code checking technique of Cohen–Sutherland algorithm to accept the completely inside lines and reject lines that lie completely on the outside of each boundary of the clip window. Then, if the endpoint isn't in the corner (A||B|C||D) of clipping window, the process of clipping will be same as Cohen–Sutherland algorithm. But if the endpoint is in the corner fig1 (A||B|C||D), by comparing the distances between the endpoint to the points that gain by the cross of line to the two clipping boundaries, confluence-points, and choose a boundary that have bigger distance for

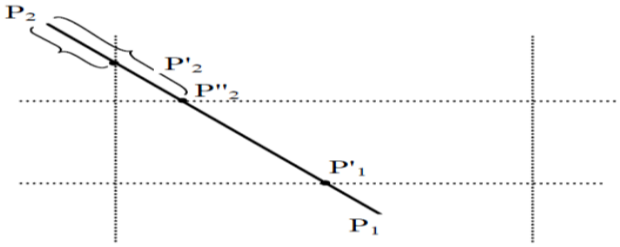
discarding, we can avoid repeating the algorithm. Against clipping boundaries in the order Left & above, Right & above, Left & Bottom, Right & Bottom, Left, Right, Bottom, Top. When the endpoint is out of two clipping boundaries, in the corner fig1 (A‖B‖C‖D), we calculate the confluence-points of line to the clipping boundaries, in fig.3. The confluence-points are p'2 and p"2. We consider two segments, from endpoint to the confluence-points, and then we calculate the length of segments. The length of segment can be calculated as:

$$d = \sqrt{(x - x_1)^2 + (y - y_1)^2} \qquad (3)$$

Then we compare the two lengths that came by calculating the length of segments and determine a bigger segment. Finally we discard the line than to a clip boundary that its confluence-points makes bigger segment. And finally, we check that the line endpoints be in the clipping window by OR'ing the new region
Codes.

To illustrate the specific steps in clipping lines using proposed algorithm, we show how the lines in Fig. 3 could be processed. Starting with the Bottom endpoint of the line from p1 to p2, we checked p1 against the Left, Right, Bottom, and above boundaries in turn and find that this point just is below the clipping rectangle. We then find the intersection point p'1 with the Bottom boundary and discard the line section from p1 to p'1. The line now has been reduced to the section from p'1 to p2. Since p2 is outside the clip window, we check this endpoint against the boundaries and find that it is outside of the Left and above of clipping rectangle. We calculate a confluence-points of line to the clipping boundaries and consider two segments; from p2 to the p'2 and from p2 to p''2. Then we compare the segments and determine a bigger segment. Finally we discard the line than to a clip boundary that makes bigger segment. So the final intersection calculation yields p"2 and the line from p'1 to p"2 is saved.

Figure 3.1 Applying proposed algorithm to clip line.



The following procedure demonstrates the changes that we apply on cohen-sutherland line-clipping algorithm and makes our new algorithm.

If the line is not rejected and not completely accepted follow steps else exit. If codes of (endpoint & LEFT_EG & TOP_EG) is not zero then calculate the distances between endpoint to LEFT_EG and endpoint to TOP_EG, then choose a boundary that have a bigger distance for discarding.

If codes of (endpoint & LEFT_EG BOTTOM_EG) is not zero then calculate the distances between endpoint to LEFT_EG and endpoint to BOTTOM_EG, then choose a boundary that have a bigger distance for discarding.

If codes of (endpoint & LEFT_EG & TOP_EG) is not zero then calculate the distances between endpoint to LEFT_EG and endpoint to TOP_EG, then choose a boundary that have a bigger distance for discarding.

If codes of (endpoint & RIGHT_EG & TOP_EG) is not zero then calculate the distances between endpoint to RIGHT_EG and endpoint to TOP_EG, then choose a boundary that have a bigger distance for discarding.

If codes of (endpoint & RIGHT_EG & BOTTOM_EG) is not zero then calculate the distances between endpoint to RIGHT_EG and endpoint to BOTTOM_EG, then choose a boundary that have a bigger distance for discarding.

Else the endpoint is not in the corner of clipping boundary and uses a common cohen-sutherland line clipping algorithm.

VI. IMPLEMENTATION

We implemented the new algorithm on PC and compared its performance with that of Cohen-Sutherland algorithm. Our machine is based on Pentium IV 2.00GHz, and the compiler is Turbo C running on Windows XP.
We apply the algorithm on five lines and the results are presented in Tables I. we assumed the clipping rectangle was from (2, 3) to (8, 7).

Table 1: results of implementation

S. No.	Line equation	Number of Confluence-points	
		CSL algorithm	Current algorithm
1	Y = 2x – 6 0≤x≤9	3	2
2	Y = -5/4x + 49/4 1≤x≤12	4	2
3	Y = -3x + 20 3≤x≤5	1	1
4	Y = -1/2x + 5 0≤x≤10	3	2
5	Y = 6 1≤x≤14	2	2

VII. RESULT

Certainly in best condition which the endpoints are completely inside or outside the clipping window, proposed algorithm and Cohen-Sutherland algorithm are done the same. When endpoints are outside the clipping window and not in the corner fig1 (A||B|C||D), these two algorithm line clip are similarly, because each endpoints are just out of one boundary of clipping rectangle. Besides even if one of the endpoints be in the corner of clipping window, as you see in table.1, they work different. When at least one endpoint be at the corner fig1 (A||B|C||D) our algorithm can done better. In this case the average of confluence-points for Cohen-Sutherland algorithm may be three and two for our algorithm. Furthermore, if both of endpoints be at the corner, the average of confluence-points may be four for Cohen-Sutherland algorithm and two for our algorithm. As you see, in any condition our algorithm at most calculate just two confluence-points to line clipping. On the other hand Cohen-Sutherland algorithm at most calculate four confluence- points. And as we said before whatever the number of confluence-endpoints be lesser, the efficiency will be higher.

VIII. CONCLUSION

In this article we try to introduce a new algorithm that is based on Cohen -Sutherland algorithm. However when the line endpoints be at the corner, our algorithm may perform better. As a whole the presented algorithm just by calculating at most two confluence-points for the line in every condition can discard faster. We can extend this algorithm for the clipping windows which even if they are not square. Moreover for increasing the accuracy of calculation we can use fuzzy.

IX. FUTURE WORK

Both CSL and LB can be extended to 3D clipping.

X. ACKNOWLEDGEMENT

This work received support from the Department of Computer Science & Engineering, Takshshila Institute of Engineering and Technology.

BIOGRAPHIES

Abhishek Pandey B.E.[I.T.], M.E.[C.S.E] is an assistant professor in the Department of Computer Science, Takshshila Institute of Technology, Jabalpur[M.P.], India. His research interests include computer graphics, data structure and software engineering.

Swati Jain is an assistant professor in the Department of Computer Science, Takshshila Institute of Technology, Jabalpur [M.P.], India. His research interests include computer graphics, Fuzzy Logics, Soft Computing and software engineering.

REFERENCES

[1] Donald Hearn, and M. Pauline Baker, "Computer Graphics, C Version", 3 edition, pp. 226-230, December 2004.
[2] Goudong Lu*, Xuanhui Wu, Qunsheng Peng, "An efficient line clipping algorithm based o adaptive line rejection", Computers & Graphics 26 (2002) 409–415.
[3] Sharma NC, Manohar S. Line clipping revisited: two efficient algorithm basedon simple geometric observations. Computers and Graphics 1992; 16(1): 51–4. Day JD. "A new two dimensional ling clipping algorithm for small windows", Computer Graphics Forum 1992; 11(4): 241–5.
[4] Wang Haohong, Wu Ruixun, Cai Shijie. "A new efficient clipping algorithm basedon geometric transformation", Journal of Software 1998; 9(10): 728–33 (in Chinese).
[5] 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).
[6] Sproull RF, Sutherland IE. "A clipping divider". In: Proceedings of the Fall Joint Computer Conference.Washington: Thompson Books, 1968. p. 765–75.
[7] Rogers DF. "Procedural elements for computer graphics". New York: McGraw-Hill, 1985. P.111–87.
[8] Sobkow MS, Pospisil P, Yang YH. A fast two-dimensional line clipping algorithm via line encoding Computers and Graphics198.