Volume 7, Issue 1, January 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparison of Various Line Clipping Algorithms: Review

Nisha
Department of Computer Science & University of Delhi,
Delhi, India

DOI: 10.23956/ijarcsse/V7I1/0149

Abstract: In computer graphics clipping is a method to selectively enable or disable rendering operations within a defined region of interest. A rendering algorithm only draws pixels in the intersection between the clip region and the scene model. Lines and surfaces outside the clipping window are removed. This paper is the survey of existing clipping techniques and it also represents comparison between different clipping techniques.

Keywords: Line Clipping Algorithms, LB, CSL, NLN etc.

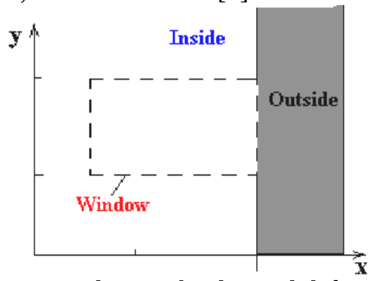
I. INTRODUCTION

In computer graphics, clipping is a technique by which we can identify the area of interest within a defined region. The region against which an object is to be clipped is called clipping window. Clipping algorithms can be applied in world coordinates or device coordinates. If clipping algorithms are applied in world coordinates then only clipped objects are mapped into the device coordinates. Otherwise firstly objects are mapped into the device coordinates and after that clipping algorithm can be applied. There are four primary clipping algorithms: Cohen–Sutherland algorithm, Liang Barsky, Cyrus Beck and Nicholl-Lee-Nicholl algorithm.

II. COHEN-SUTHERLAND LINE CLIPPING ALGORITHM

In computer graphics Cohen Sutherland is one line clipping algorithm. In this algorithm 2D space is divided into 9 regions against the clipping window and every region has one unique code. The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is **trivially accepted** and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is **trivially rejected** and needs to be neither clipped nor displayed.

To determine whether endpoints are inside or outside a window, the algorithm sets up a **half-space code** for each endpoint. Each edge of the window defines an infinite line that divides the whole space into two half-spaces, the **inside half-space** and the **outside half-space**, as shown below. [5]



As you proceed around the window, extending each edge and defining an inside half-space and an outside half-space, nine regions are created - the eight "outside" regions and the one "inside"region. Each of the nine regions associated with the window is assigned a 4-bit code to identify the region. Each bit in the code is set to either a **1**(true) or a **0**(false). If the region is to the **left** of the window, the **first** bit of the code is set to 1. If the region is to the **top** of the window, the **second** bit of the code is set to 1. If to the **right**, the **third** bit is set, and if to the **bottom**, the **fourth** bit is set. The 4 bits in the code then identify each of the nine regions as shown below.

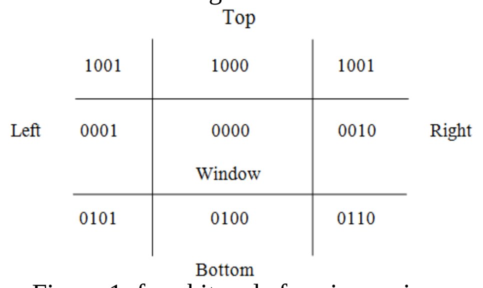


Figure 1: four bit code for nine regions

For any endpoint **(x , y)** of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set according to the following conditions:

- First bit set **1** : Point lies to **left** of window $x < x_{min}$
- Second bit set **1** : Point lies to **right** of window $x > x_{max}$
- Third bit set **1** : Point lies below(**bottom**) window $y < y_{min}$
- fourth bit set **1** : Point lies above(**top**) window $y > y_{max}$

The sequence for reading the codes' bits is **LRBT** (Left, Right, Bottom, Top).

Once the codes for each endpoint of a line are determined, the logical **AND** operation of the codes determines if the line is completely outside of the window. If the logical AND of the endpoint codes is **not zero**, the line can be trivally rejected. For example, if an endpoint had a code of 1001 while the other endpoint had a code of 1010, the logical AND would be 1000 which indicates the line segment lies outside of the window. On the other hand, if the endpoints had codes of 1001 and 0110, the logical AND would be 0000, and the line could not be trivally rejected.

The logical **OR** of the endpoint codes determines if the line is completely inside the window. If the logical OR is **zero**, the line can be trivally accepted. For example, if the endpoint codes are 0000 and 0000, the logical OR is 0000 - the line can be trivally accepted. If the endpoint codes are 0000 and 0110, the logical OR is 0110 and the line can not be trivally accepted.

Algorithm

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's endpoints are tested to see if the line can be trivally accepted or rejected. If the line cannot be trivally accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted.

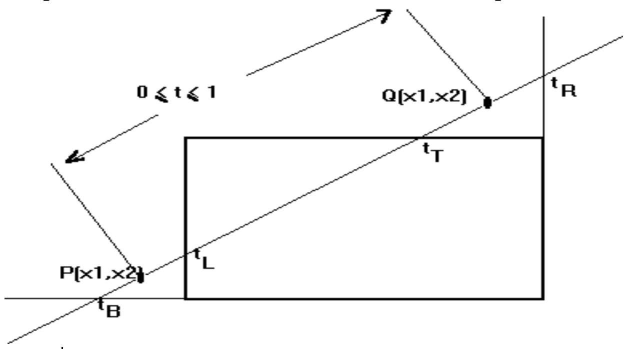
To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies. [5]

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.
   If both codes are **0000**,(bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.
   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.
3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left**-to-**Right**, **Bottom-**to-**Top**.
5. When a set bit (1) is found, compute the intersection **I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with **I** and repeat the algorithm.

III. LIANG- BARSKY LINE CLIPPING ALGORITHM

The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same. The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window. So we will study only the idea of **Liang-Barsky.**

Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations. This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.



Let $P(x_1,y_1)$, $Q(x_2,y_2)$ be the line which we want to study. The **parametric equation of the line segment** from gives x-values and y-values for every point in terms of a **parameter t** that ranges from 0 to 1. The equations are

$x = x_1 + (x_2 - x_1)*t = x_1 + dx*t$ and $y = y_1 + (y_2 - y_1)*t = y_1 + dy*t$

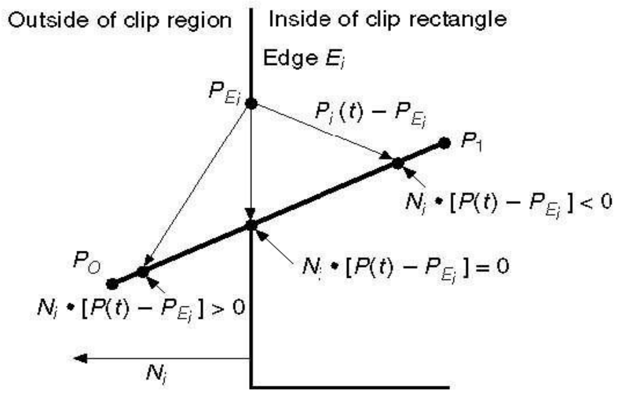
We can see that when t = 0, the point computed is P(x1,y1); and when t = 1, the point computed is Q(x2,y2). [2]

Algorithm

1. Set $t_{min} = 0$ and $t_{max} = 1$
2. Calculate the values of tL, tR, tT, and tB (t values).
   - if $t < t_{min}$ or $t > t_{max}$ ignore it and go to the next edge
   - otherwise classify the **t**value as entering or exiting value (using inner product to classify)
   - if **t** is entering value set $t_{min} = t$; if **t** is exiting value set $t_{max} = t$
3. If $t_{min} < t_{max}$ then **draw a line** from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax)
4. If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1 + dx*tmax, y1 + dy*tmax) are intersection between line and edge.

IV. CYRUS BECK LINE CLIPPING ALGORITHM

This algorithm is more efficient than Cohen-Sutherland algorithm. It employs parametric line representation and simple dot products.



Outside of clip region
Inside of clip rectangle
Edge E_i
P_{E_i}
$P_i(t) - P_{E_i}$
P_1
$N_i \bullet [P(t) - P_{E_i}] < 0$
$N_i \bullet [P(t) - P_{E_i}] = 0$
P_0
$N_i \bullet [P(t) - P_{E_i}] > 0$
N_i

Parametric equation of line is −

P_0P_1:$P(t) = P_0 + t(P_1 - P_0)$

Let N_i be the outward normal edge E_i. Now pick any arbitrary point P_{Ei} on edge E_i then the dot product $N_i.[P(t) – P_{Ei}]$ determines whether the point P(t) is “inside the clip edge” or “outside” the clip edge or “on” the clip edge. [3]

The point P(t) is inside if $N_i.[P(t) – P_{Ei}] < 0$

The point P(t) is outside if $N_i.[P(t) – P_{Ei}] > 0$

The point P(t) is on the edge if $N_i.[P(t) – P_{Ei}] = 0$ (Intersection point)

$N_i.[P(t) – P_{Ei}] = 0$

$N_i.[P_0 + t(P_1 - P_0) – P_{Ei}] = 0$ (Replacing P(t) with $P_0 + t(P_1 - P_0)$)

$N_i.[P_0 – P_{Ei}] + N_i.t[P_1 - P_0] = 0$

$N_i.[P_0 – P_{Ei}] + N_i{\cdot}tD = 0$ (substituting D for $[P_1 - P_0]$)

$N_i.[P_0 – P_{Ei}] = - N_i{\cdot}tD$

The equation for t becomes,

$$t=Ni.[Po−PEi]−Ni.D$$

It is valid for the following conditions −

- $N_i \neq 0$ (error cannot happen)
- $D \neq 0$ ($P_1 \neq P_0$)
- $N_i{\cdot}D \neq 0$ (P_0P_1 not parallel to E_i)

V. NICHOLL-LEE-NICHOLL LINE CLIPPING ALGORITHM

Assume that we want to clip a segment [P1,P2] against a window. The determination of the exact edges, if any, that one needs to intersect, reduces, using symmetry, to an analysis of the three possible positions of P1 shown in Figure 3.8. The cases are

(1) P1 is in the window (Figure 4.1(a)),
(2) P1is in a “corner region” (Figure 4.1(b)), or
(3) P1 is in an “edge region” (Figure 4.1(c)).

For each of these cases one determines the regions with the property that no matter where in the region the second point P2 is, the segment will have to be intersected with the same boundaries of the window. These regions are also indicated in Figure 4.1. As one can see, these regions are determined by drawing the rays from P1 through the four corners of the window

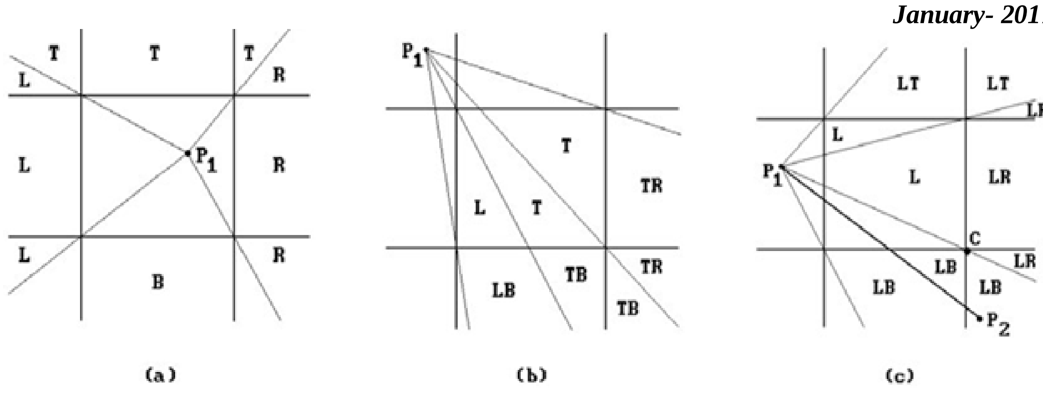


Figure 5.1 Nicholl-Lee-Nicholl line clipping

VI. CONCLUSION

In this paper I defined all the line clipping algorithms. Cohen Sutherland is the simplest line clipping algorithm but the Liang-Barsky algorithm is more efficient, since intersection calculations are reduced. Each update of parameters t1 and t2 requires only one division; and window intersections of the line are computed only once, when the final values of t1 and t2 have been computed. In contrast, the Cohen-Sutherland algorithm can repeatedly calculate intersection along a line path, even though the line may be completely outside the clip window. Both the Cohen-Sutherland and Liang-Barsky algorithms can be extended to three-dimensional clipping. NLN cannot extend to three-dimensional clipping [2].

REFERENCES

[1] Abhishek Pandey1, Swati Jain2, “Comparison of Various Line Clipping Algorithms for Improvement”, International Journal of Modern Engineering Research (IJMER) Vol.3, Issue.1, Jan-Feb. 2013.

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

[3] Rogers DF. “Procedural elements for computer graphics”. New York: McGraw-Hill, 1985. P.111–87. [4] Sobkow MS, Pospisil P, Yang YH. A fast two-dimensional line clipping algorithm via line encoding Computers and Graphics198.