Clipping

When we have to display a large portion of the picture, then not only scaling & translation is necessary, the visible part of picture is also identified. This process is not easy. Certain parts of the image are inside, while others are partially inside. The lines or elements which are partially visible will be omitted.

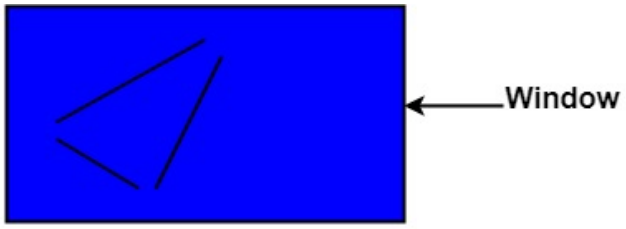
For deciding the visible and invisible portion, a particular process called clipping is used. Clipping determines each element into the visible and invisible portion. Visible portion is selected. An invisible portion is discarded.

Types of Lines:

Lines are of three types:

1. Visible: A line or lines entirely inside the window is considered visible
2. Invisible: A line entirely outside the window is considered invisible
3. Clipped: A line partially inside the window and partially outside is clipped. For clipping point of intersection of a line with the window is determined.

Case1:

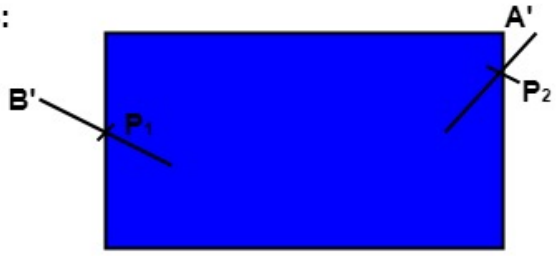


Visible Lines

Case2:

Invisible Lines

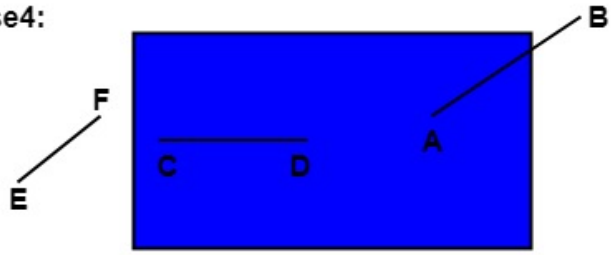
Case3:



In this figure P_1 and P_2 are point of intersection. The line P_2 to A' and P_1 to B' is discarded or clipped.

Clipped Lines

Case4:

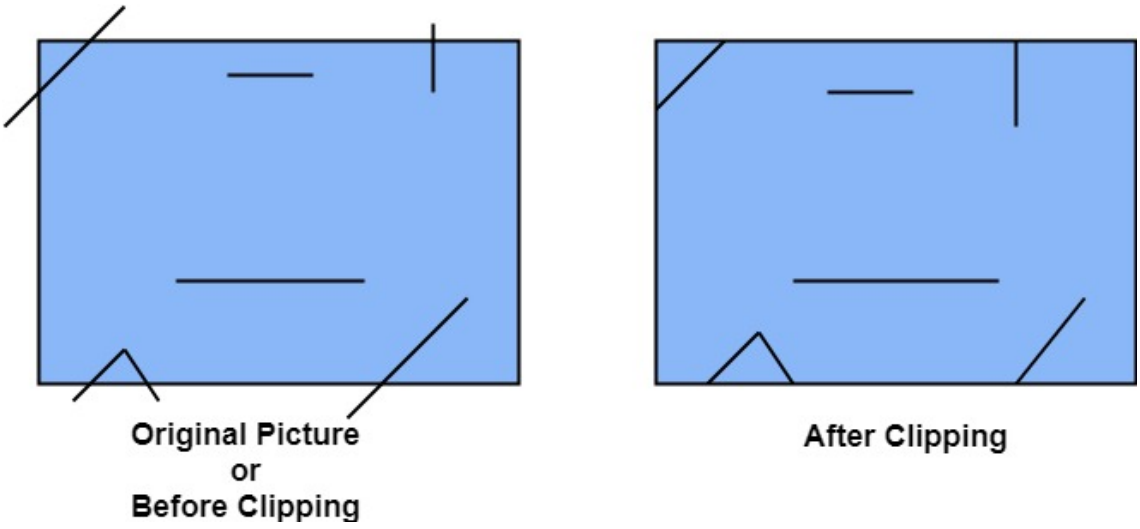


In this figure AB is clipped case.
CD is visible line.
EF is invisible line.

Mixed Lines

Clipping can be applied through hardware as well as software. In some computers, hardware devices automatically do work of clipping. In a system where hardware clipping is not available software clipping applied.

Following figure show before and after clipping :



The window against which object is clipped called a clip window. It can be curved or rectangle in shape.

Applications of clipping:

1. It will extract part we desire.
2. For identifying the visible and invisible area in the 3D object.
3. For creating objects using solid modeling.
4. For drawing operations.
5. Operations related to the pointing of an object.
6. For deleting, copying, moving part of an object.

Clipping can be applied to world co-ordinates. The contents inside the window will be mapped to device co-ordinates. Another alternative is a complete world co-ordinates picture is assigned to device co-ordinates, and then clipping of viewport boundaries is done.

Types of Clipping:

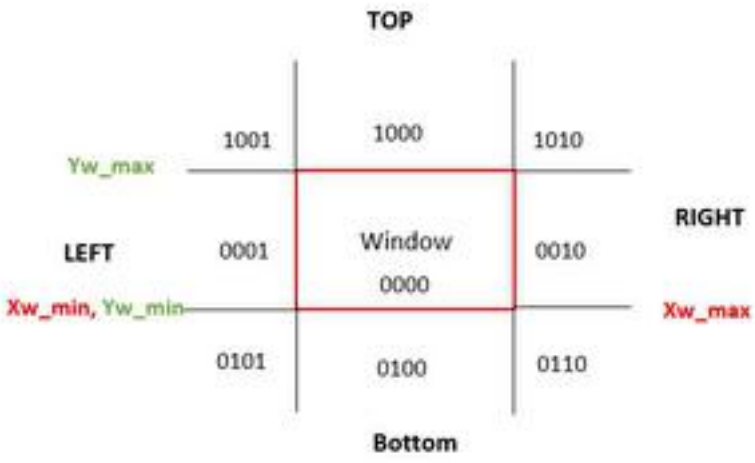
1. Point Clipping
2. Line Clipping
3. Area Clipping (Polygon)
4. Curve Clipping
5. Text Clipping
6. Exterior Clipping

Cohen–Sutherland Line Clipping Algorithm

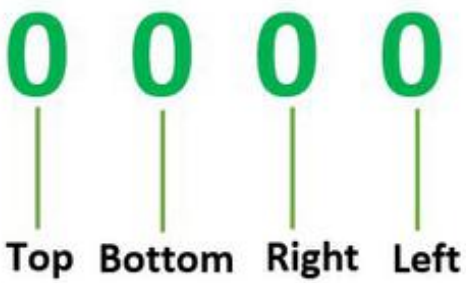
Description:- In this algorithm, we are given 9 regions on the screen. Out of which one region is of the window and the rest 8 regions are around it given by 4 digit binary. The division of the regions are based on (x_max, y_max) and (x_min, y_min).

The central part is the viewing region or window, all the lines which lie within this region are completely visible. A region code is always assigned to the endpoints of the given line.

To check whether the line is visible or not.



Formula to check binary digits:- TBRL which can be defined as top, bottom, right, and left accordingly.



Algorithm

Steps

1) Assign the region codes to both endpoints.

2) Perform OR operation on both of these endpoints.

3) if OR = 0000,

then it is completely visible (inside the window).

else

Perform AND operation on both these endpoints.

i) if AND ? 0000,

then the line is invisible and not inside the window. Also, it can't be considered for clipping.

ii) else

AND = 0000, the line is partially inside the window and considered for clipping.

4) After confirming that the line is partially inside the window, then we find the intersection with the boundary of the window. By using the following formula:- Slope:- m= (y2-y1)/(x2-x1)

a) If the line passes through top or the line intersects with the top boundary of the window.

$$x = x + (y\_wmax - y)/m$$

$$y = y\_wmax$$

b) If the line passes through the bottom or the line intersects with the bottom boundary of the window.

$$x = x + (y\_wmin - y)/m$$

$$y = y\_wmin$$

c) If the line passes through the left region or the line intersects with the left boundary of the window.

$$y = y + (x\_wmin - x)*m$$

$$x = x\_wmin$$

d) If the line passes through the right region or the line intersects with the right boundary of the window.

$$y = y + (x\_wmax - x)*m$$

$$x = x\_wmax$$

5) Now, overwrite the endpoints with a new one and update it.

6) Repeat the 4th step till your line doesn't get completely clipped

Liang Barsky Line Clipping Algorithm

Liang-Barsky algorithm is a line clipping algorithm. This algorithm is more efficient than Cohen–Sutherland line clipping algorithm and can be extended to 3-Dimensional clipping. This algorithm is considered to be the faster parametric line-clipping algorithm. The following concepts are used in this clipping:

The parametric equation of the line.
The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.

The parametric equation of a line can be given by,

$$X = x_1 + t(x_2-x_1)$$
$$Y = y_1 + t(y_2-y_1)$$

Where, t is between 0 and 1. Then, writing the point-clipping conditions in the parametric form:

$$xw_{min} <= x_1 + t(x_2-x_1) <= xw_{max}$$
$$yw_{min} <= y_1 + t(y_2-y_1) <= yw_{max}$$

The above 4 inequalities can be expressed as,

$$tp_k <= q_k$$

Where k = 1, 2, 3, 4 (correspond to the left, right, bottom, and top boundaries, respectively). The p and q are defined as,

p1 = -(x2-x1), q1 = x1 - xwmin (Left Boundary)
p2 = (x2-x1), q2 = xwmax - x1 (Right Boundary)
p3 = -(y2-y1), q3 = y1 - ywmin (Bottom Boundary)
p4 = (y2-y1), q4 = ywmax - y1 (Top Boundary)

When the line is parallel to a view window boundary, the p value for that boundary is zero. When p

k < 0, as t increase line goes from the outside to inside (entering). When p

k > 0, line goes from inside to outside (exiting). When p

k = 0 and q

k < 0 then line is trivially invisible because it is outside view window. When p

k = 0 and q

k > 0 then the line is inside the corresponding window boundary. Using the following conditions, the position of line can be determined:

Condition	Position of line
$p_k = 0$	parallel to the clipping boundaries
$p_k = 0$ and $q_k < 0$	completely outside the boundary
$p_k = 0$ and $q_k >= 0$	inside the parallel clipping boundary
$p_k < 0$	line proceeds from outside to inside
$p_k > 0$	line proceeds from inside to outside

Parameters t1 and t2 can be calculated that define the part of line that lies within the clip rectangle. When,

pk < 0, maximum(0, qk/pk) is taken.
pk > 0, minimum(1, qk/pk) is taken.
If t1> t2 , the line is completely outside the clip window and it can be rejected.
Otherwise, the endpoints of the clipped line are calculated from the two values of parameter t.

Algorithm

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 and right-bottom) of the window, say $(x_{wmin}, y_{wmax}, x_{wmax}, y_{wmin})$
3. Calculate the values of parameters p_i and q_i for i = 1, 2, 3, 4 such that

   $p_1 = -\Delta x$ $\quad q_1 = x_1 - x_{wmin}$

   $p_2 = \Delta x$ $\quad q_2 = x_{wmax} - x_1$

   $p_3 = -\Delta y$ $\quad q_3 = y_1 - y_{wmin}$

   $p_4 = \Delta y$ $\quad q_4 = y_{wmax} - y_1$
4. if $p_i = 0$, then

   { The line is parallel to i^{th} boundary.

   Now, if $q_i < 0$ then

   { line is completely outside the boundary, hence

   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 co-ordinates to draw line. Go to stop.**

   }

   }
5. **Initialise values for t_1 and t_2 as**

   $t_1 = 0$ and $t_2 = 1$
6. **Calculate values for q_i/p_i for i = 1, 2, 3, 4**
7. **Select values of q_i/p_i where $p_i < 0$ and assign maximum out of them as t_1.**
8. **Select values of q_i/p_i where $p_i > 0$ and assign minimum out of them as t_2.**
9. **If $(t_1 < t_2)$**

   { Calculate the endpoints of the clipped line as follows :

   $xx_1 = x_1 + t_1 \Delta x$

   $xx_2 = x_1 + t_2 \Delta x$

   $yy_1 = y_1 + t_1 \Delta y$

   $yy_2 = y_1 + t_2 \Delta y$

   Draw line (xx_1, yy_1, xx_2, yy_2)

   }
10. **Stop.**

END