Clipping

① window co-ordinate
② viewport

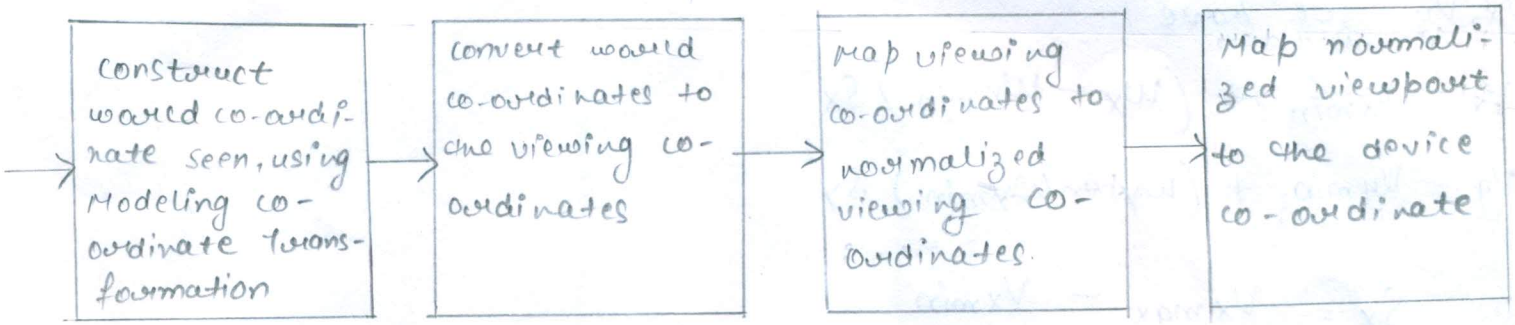
① window co-ordinate (1) A window co-ordinate area selected for display is called window.

(2) Area on a display device to which a window is mapped is called view port.

(3) A window define what is to be viewed and the viewport define where it is to be displayed.

(4) The general mapping of a part of world co-ordinate Seen to the device co-ordinates is referred to as transformation. Sometime Two Dimensional viewing transformation is also referred as window to viewport transformation for windowing transformation.

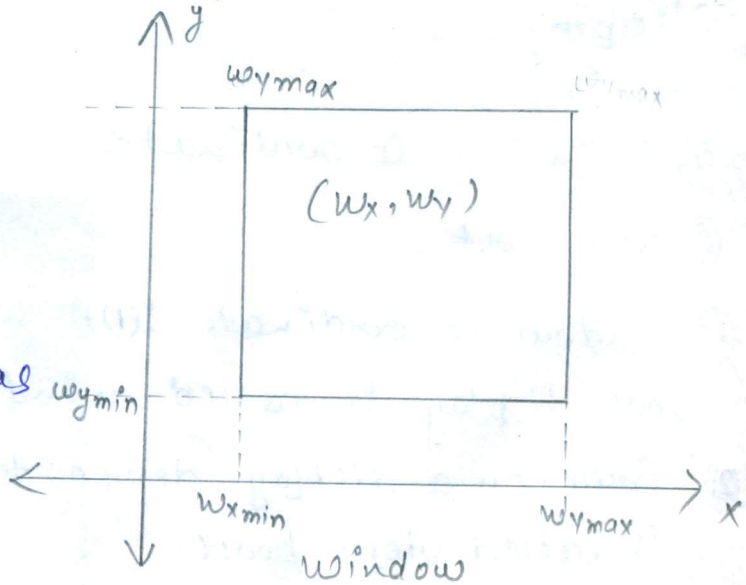
The viewing Pipeline



viewing transformation

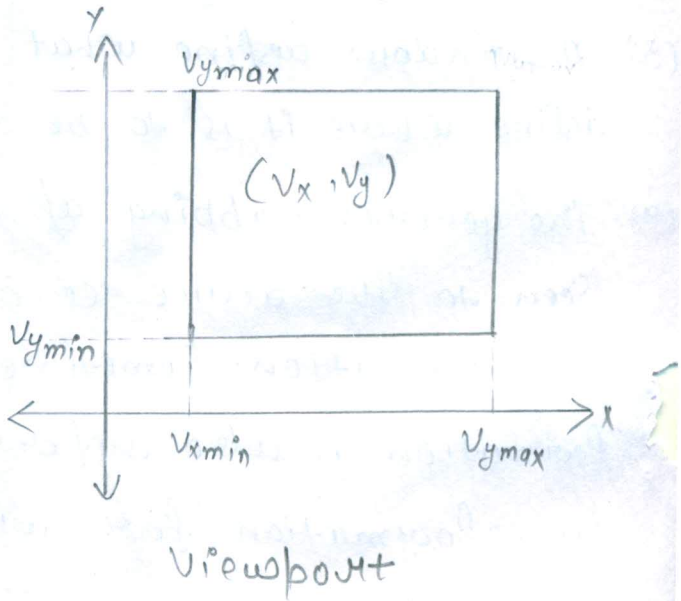
Window to viewport Co-ordinate Transformation :-

→ A point ~~at~~ at position (w_x, w_y) in the window is mapped into position (v_x, v_y) in the viewport to maintain same relative placement in the viewport as in window the required at



$$\frac{V_x - V_{xmin}}{V_{xmax} - V_{xmin}} = \frac{W_x - W_{xmin}}{W_{xmax} - W_{xmin}}$$

$$\frac{V_y - V_{ymin}}{V_{ymax} - V_{ymin}} = \frac{W_y - W_{ymin}}{W_{ymax} - W_{ymin}}$$



After Solving these expression for viewport position V_x, V_y we have :-

$$V_x = V_{xmin} + (W_x - W_{xmin}) S_x$$

$$V_y = V_{ymin} + (W_y - W_{ymin}) S_y$$

$$S_x = \frac{V_{xmax} - V_{xmin}}{W_{xmax} - W_{xmin}}$$

$$S_y = \frac{V_{ymax} - V_{ymin}}{W_{ymax} - W_{ymin}}$$

This conversion is perform with following sequence of transformation.

1.) Perform a scaling transformation using a fixed point position that scale the window Area to the size of a viewport.

2.) Translate scale window area to the position of viewport.

If $S_x = S_y$ Uniform Scaling

If $S_x \neq S_y$ Differential Scaling

Clipping operation

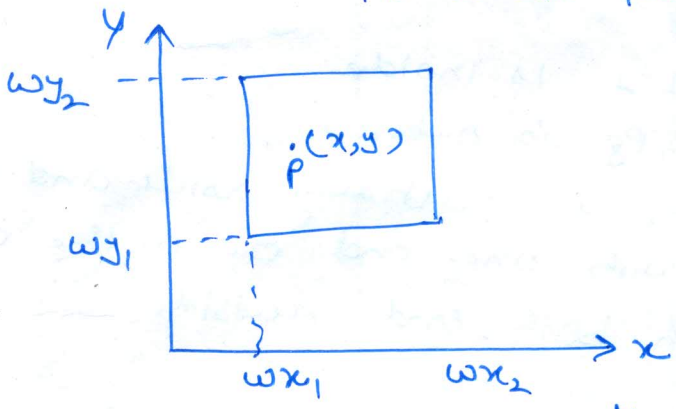
1. Point clipping
2. Line clipping
3. Polygon clipping
4. Curve clipping
5. Text clipping

Clipping can be perform on either window or viewport. window clipping is better than the viewport.

Clipping → is a procedure of determine which element of a picture lies inside the window or visible and what parts of the picture are discarded.

Types → 1. Point clipping
2. line ,,
3. Polygon ,,
4. Area ,,
5. curve ,,
6. Text ,,

Point clipping → (a) assume clip window is a rectangle in a std position.

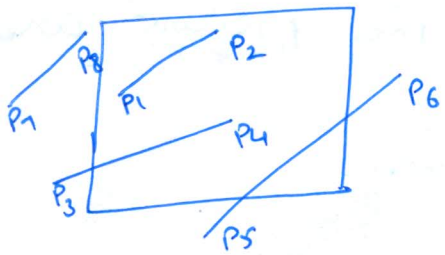


point P(x,y) is clipped if
x lies b/w $wx_1 \le x \le wx_2$ & y lies b/w $wy_1 \le y \le wy_2$

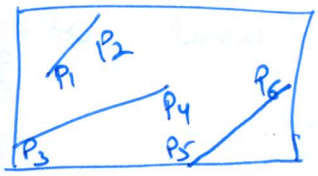
(1) point clipping can be applied to scenes involving explosions or sea foam that are modeled with particles distributed in some region of scene.

(2) It tells us whether the given point P(x,y) is within the given window or not.

Line clipping → In this, we will cut the portion of line which is outside of window and keep only the portion that is inside the window



Before clipping



After clipping

① firstly to determine wheather given line segment is completely outside or inside the clipping window

② if it is not identified then we must perform intersection calculation with one or more clipping boundaries.

for eg → ① line P_1P_2 is inside
② line P_7P_8 is outside
③ line P_3P_4 is partially inside and partially outside with one end outside the clip windo
④ P_5P_6 with both end outside.

So these lines cross one or more clipping boundaries & may require calculation of multiple intersections points. To minimize calculation, There are no. of algorithm which are used to identify the line segment & to reduce intersection calculations.

So for line P_5P_6 and P_3P_4

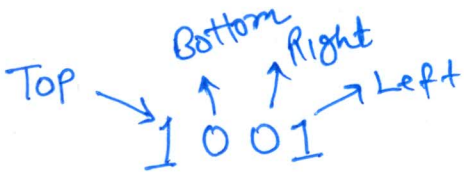
$$x = x_1 + u(x_2 - x_1)$$

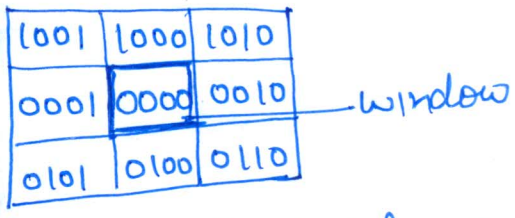
$$y = y_1 + u(y_2 - y_1) \qquad 0 \le u \le 1$$

if u does not lies range 0 to 1 → the line segment does not enter the interior of the window at that boundary

If lies in range 0 to 1 → line segment cross into clipping area

Cohen- Sutherland line clipping





① every line-end point is assigned a four digit binary code called region code that identifies the location of the point relative to the boundaries of clipping rectangles. each bit position in region code is used to indicate relative coordinate position of point w.r.t clip window

② After assigning region code for all end points, we can quickly determine which lines are completely ~~contained~~ inside the clip window & which are clearly outside.

(a) if line with both end points 0000 then line completely inside, it is accepted

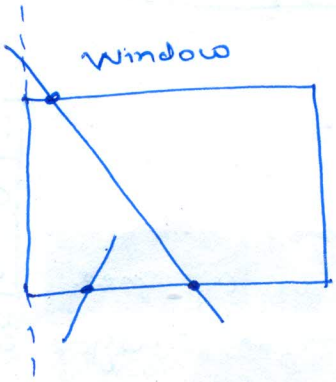
(b) if line have a 1 in the same bit position in the region codes for each end point are completely outside the clipping rectangle, It will be rejected

(c) A Method that can be used to test lines for total clipping is to perform logical and operation with both region codes. if result is not 0000, then line is completely outside the clipping region.

for 1001 ← 0101 same bit position

1001
0101
0001 → not 0000 then rejected

(d) some lines are not identified as completely outside or inside a clip window then these test are checked for intersection with window boundaries.



Intersection point with a clipping boundary can be calculated using the slope intercept form of line equation.

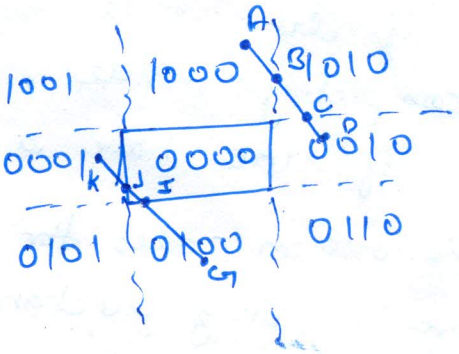
$$y = y_1 + m(x - x_1)$$

x value is set either to x_{min} or to x_{max}

Ily $\quad x = x_1 + \frac{y - y_1}{m}$

y — set to y_{max} or y_{min}

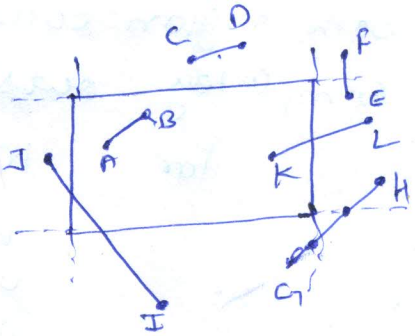
for eg



for line KJ — 0001
0001
0001 Non zero

JI — 0000
0000
0000 zero

IG — 0100
0100
0100 Non zero



(1) Visible → AB

(2) Invisible → CD, EF

(3) Clipping Candiate → KL, GH, IJ

Liang - Barsky line clipping →

$$x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y$$
→ parametric eqn of line segment $0 \le u \le 1$

$$x_{wmin} \le x_1 + u\Delta x \le x_{wmax}$$
$$y_{wmin} \le y_1 + u\Delta y \le y_{wmax}$$

each of these four inequalities can be expressed as

$$uP_k \le q_k \qquad k = 1, 2, 3, 4$$

$$p_1 = -\Delta x \qquad q_1 = x_1 - x_{wmin}$$
$$p_2 = \Delta x \qquad q_2 = x_{wmax} - x_1$$
$$p_3 = -\Delta y \qquad q_3 = y_1 - y_{wmin}$$
$$p_4 = \Delta y \qquad q_4 = y_{wmax} - y_1$$

(1) Any line ||el to clipping boundary has $p_k = 0$ for the value of k corresponding to that Boundary $k = 1, 2, 3, 4$ correspond to left, right, bottom & top boundary.

if $q_k < 0$ — outside the boundary

$q_k \ge 0$ — inside is inside the clipping boundary

(2) when $p_k < 0$ → line proceeds from outside to the inside of the infinite extension of this particular clipping boundary.

(3) when $p_k > 0$ → proceeds from the inside to the outside.

(4) for non-zero value of p_k, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extension of boundary k as

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

for each line, we can calculate values for Parameter u_1 & u_2 that define that part of line that lies within clip rectangle.

For u_1 → (outside to inside) ($p < 0$)

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

u_1 is taken as largest of set consisting of 0 and various values of r.

For u_2 → (inside to outside) ($p > 0$)

u_2 is taken as minimum of set consisting of 1 and calculated r values.

Now if $u_1 > u_2$ 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 u.

Algorithm → ① Line interaction parameter are initialized to values $u_1 = 0$ and $u_2 = 1$.

② For each line, p and q are calculated to determine wheather line can be rejected or wheather the interaction parameter are to be adjusted.

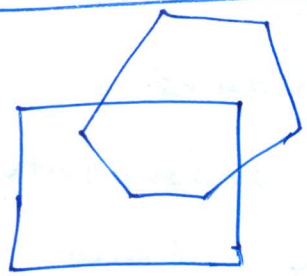
③ when $p < 0$, r is used to update u_1

when $p > 0$, r is used to update u_2

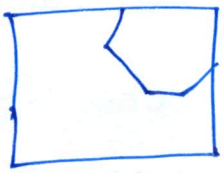
if $u_1 > u_2$ reject the line, otherwise, we update the appropriate u in a shortening of the line.

when $p = 0$ & $q < 0$, we can discard the line since it is ||el to and outside of this boundary.

(3) Polygon Clipping →



Before



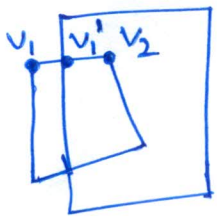
after

Sutherland – Hodgeman Polygon clipping

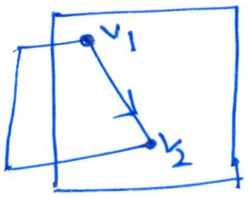
(1) This could be accomplished by processing all polygon vertices, against each clip rectangle boundary in turn. We could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices, then to right, bottom and top boundary.

There are four possible cases when processing vertices in sequence around the perimeter of polygon

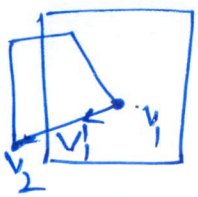
(a) if first vertex is outside the window boundary and second vertex is inside, both the intersection point of polygon edge and second vertex are added to output vertex list.



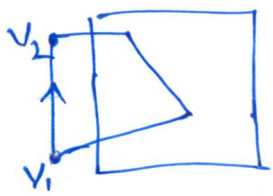
Out → In
Save V_1', V_2
(a)



In → In
Save V_2
(b)



In → out
Save V_1'
(c)



Out → out
Save None
(d)

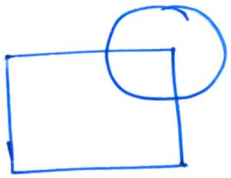
We requires only output list for storage of vertices.

Liang-Barsky Vs Cohen-Sutherland algo

① Liang-Barsky is more efficient than Sutherland, since intersection calculations are reduced.

② It requires only one division & window intersection of line, that are computed only once. whereas in Sutherland can repeatedly calculate intersection along line path, even though line may be completely outside the clip window. each intersection requires both division & multiplication.

Curve clipping →



Text clipping → There are no. of techniques which are used to clip character or string

(1) all or none string clipping → If all of string is inside a clip window, we keep it, otherwise the string is discarded.

(2) Alternative Method, we discard only those character that are not completely inside the window.

(3) Other Method for handling text clipping is to clip the component of individual characters. If an individual character overlaps a clip window boundary, we clip off the parts of character that are outside the window.

(a) (1) (2) (3)