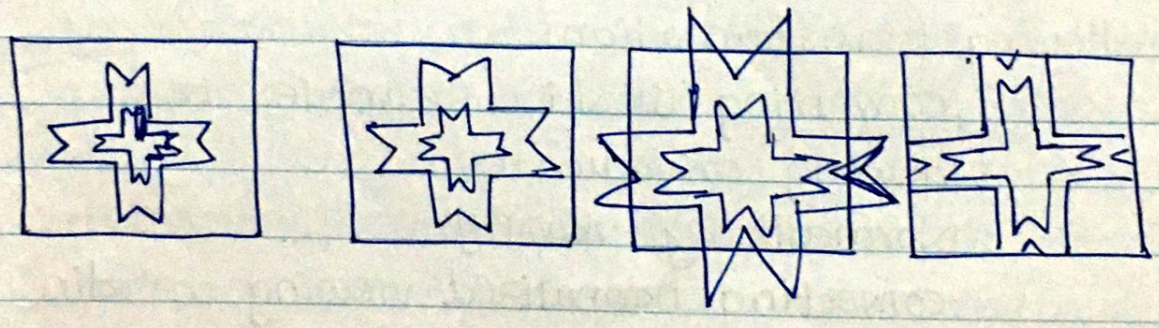
000555

Chapter 4: Windowing & Clipping

The process of selecting & viewing the picture with different views is called windowing & process which divides each element of the picture into its visible & invisible portion, allowing the invisible portion to be discarded is called clipping.



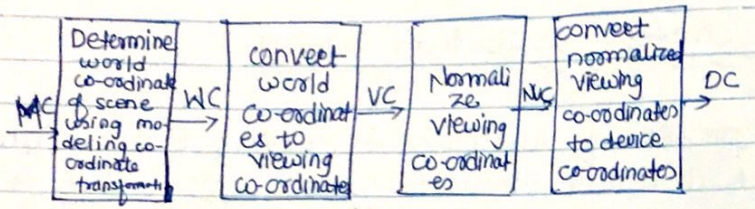
* Window to Viewport Transformation

The picture is stored in the computer memory using any convenient Cartesian co-ordinate system, referred to as World co-ordinate system (WCS)

When picture is displayed on the display device it is measured, in Physical Device co-ordinate System (PDCS) corresponding to display device

Therefore, displaying an image of picture involves mapping the co-ordinates of the points

& lines that form the picture into the appropriate physical device co-ordinate where the image is to be displayed. This mapping of co-ordinates is achieved with the use of co-ordinate transformation known as viewing transformation.



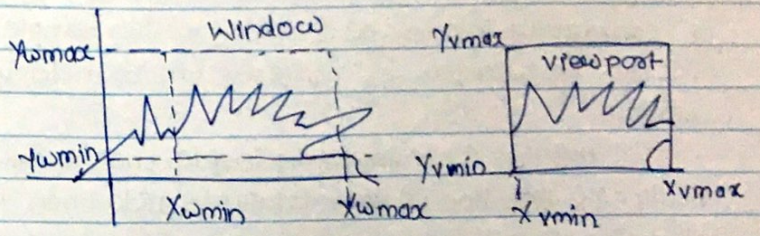
viewing transformation pipeline

- The viewing transformation which maps picture co-ordinates in WCS to display co-ordinates in PDCS is performed by the following transformations:
  - converting world co-ordinates to viewing co-ordinates
  - Normalizing viewing
  - converting normalized viewing co-ordinates to device co-ordinates.

World co-ordinate system is infinite in extent & device display area is infinite. To perform a viewing transformation we select a finite world co-ordinate area for display called window.

An area on device to which a window is mapped is called a viewport.

The window defines what is to be viewed; the viewport defines where it is to displayed.



The steps involved in viewing transformations:

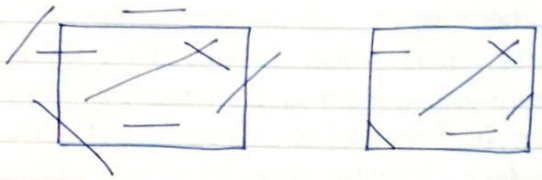
1. construct the scence in world co-ordinates using the output primitives & attributes.
2. obtain particular orientation for the window by setting a 2D viewing co-ordinate system in the world co-ordinate plane & define a window in the viewing co-ordinate system
3. use viewing co-ordinates reference frame to provide a method for setting up arbitrary orientations for rectangular windows
4. once the viewing reference frame is established, transform descriptions in window co-ordinates to viewing co-ordinates.
5. Define view port in normalized co-ordinates & map the viewing co-ordinate descriptions of the scence to normalized co-ordinates
6. clip all parts of the picture which lie outside the viewports

Line Clipping

Here we have to examine each & every line which we are going to display.

We have to determine whether or not a line is completely inside the window, or lies completely outside the window, or crosses the boundary of window.

If the line is completely inside then display is fully. If the line is ~~com~~ totally outside then discard that line. But if the line is crossing to the window boundary, the we have to find the intersection point of the line with the edge of window & draw the portion which lies inside



before clipping After clipping

Line Clipping Algorithm

A. Cohen-Sutherland Algo.
B. Mid-point Subvision Algo.
C. Cyrus-Beck-Line clipping Algo.
D. Liang Barsky Line clipping Algo.

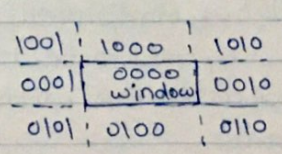
1) Cohen Sutherland Clipping Algo

- Oldest algo
- most popular algo developed by Dan cohen & Ivan Sutherland.
- This algo immediately removes the lines which are ~~to~~ lying totally outside the window.
- This algo divides the plane in nine (9) parts & assign outcode or binary number to each part.

The four bit codes are called region codes or outcodes.

These codes identify the locations of the point relative to the boundaries of the clipping rectangle.

Each bit position in the region code is used to indicate one of the four relative co-ordinate positions of the point with respect to clipping window: To the left, right, top or bottom. (sequence LRBT)



The rightmost bit is the first bit & the bits are set to 1 based on the following scheme

set Bit 1 : If the end point is the left of the window.
set Bit 2 : If the end point is the right of window
set Bit 3 : If the end point is below the window
set Bit 4 : If the end point is above the window

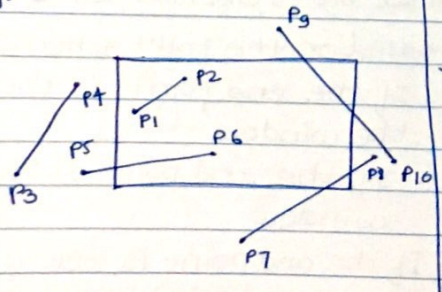
Otherwise, the bit is set to zero

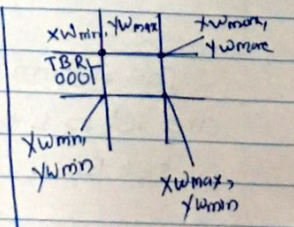
- Once we have established region codes for all the line endpoints, we can determine which lines are completely inside the clipping window & which are clearly outside
- Any ~~point~~ lines that are completely inside the window boundaries have a region code of 0000 for both endpoints
- Any lines that ~~are~~ have a 1 in the same bit position in the region codes for each end point are completely outsides the clipping rectangle & we trivially reject these lines.

A method used to test lines for total clipping is equivalent to logical AND operator.

- If the result of the logical AND operations with two ends point code is not 000 0, the line is complete outside the clipping region.
- The lines ~~that~~ cannot be identified as completely inside or completely outside clipping window by these test are checked for intersection with the window boundaries

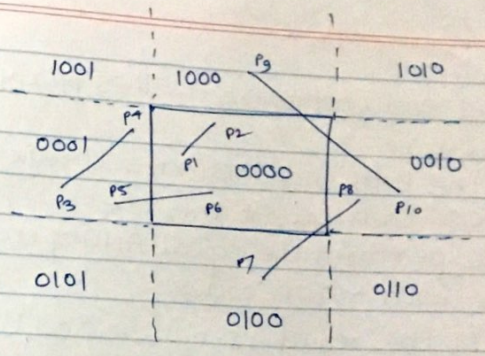
eg:-





$x_{min} \leq x \leq x_{max}$

$y_{min} \leq y \leq y_{max}$



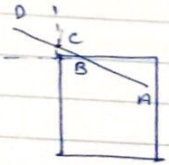
	Line	Endpoint codes		Logical ANDing	Result
①	P_1P_2	0000	0000	0000	completely visible
②	P_3P_4	0001	0001	0001	completely invisible
③	P_5P_6	0001	0000	0000	partially visible
④	P_7P_8	0100	0010	0000	partially visible
⑤	P_9P_{10}	1000	0010	0000	Partially visible

		AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

1. Read two end points of lines $p_1(x_1, y_1)$ & $p_2(x_2, y_2)$
2. If the both endpoints have a region code 0000 then accept this line
3. else perform the logical AND operation for both region codes.
   3.1. If the result is not 0000, then reject the line
   3.2 else you need clipping
   3.2.1 Choose endpoint of the line that is outside the window
   3.2.2 Find the intersection point at the window boundary
   3.2.3. Replace end point with the intersection point & update the region code
   3.2.4. Repeat step 2 until we find clipped line either trivially accepted or rejected

Step 4: Repeat step 1 for other lines.

eg.



consider AD
A = 0000
D = 1001

AND 0000 ∴ line cannot be trivially rejected
OR 1001 ∴ is not zero

line cannot be trivially accepted

Choose D as outside point, we first use the top edge to clip AD at B. The algo recomputes B' outcode as 0000 with the next iteration of algo, AB is tested & is trivially accepted & display

S.N.J.B.'s
SHRI H. H. J. B. POLYTECHNIC, CHANDWAD
CLASS TEST I / II (201 - 201)

Course & Semester :	Roll No. :	Date : / /201
Name of Subject :		Marks obtained :
Sign. of Supervisor :	Sign. of Sub. Examiner :	

Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

Supplied By : SNJB

000538

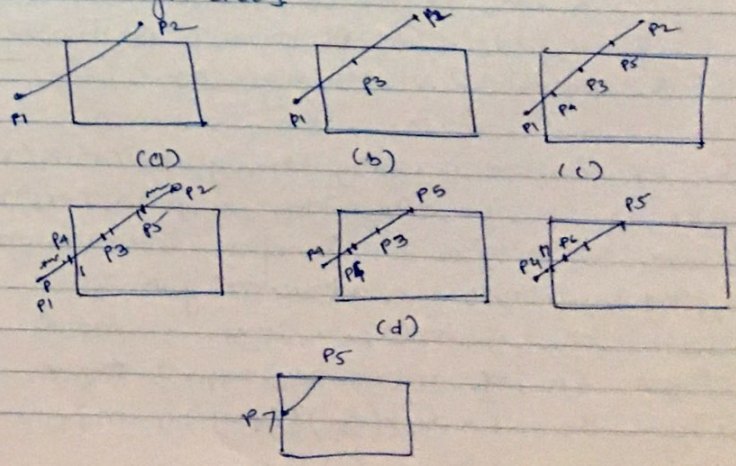
Mid point Subvision Algo

This algo is mainly used to compute visible areas of line that are present in the view port are of the sector or the image.

initially the line is tested for visibility. If line is completely visible it is drawn & if it is completely invisible it is rejected. If the line is partially visible then it is subdivided into two equals parts.

The visibility tests are then applied to each ~~parts~~ halfs.

This subvision process is repeat until we get completely visible & completely invisible line segments



line P_1P_2 partially invisible. It is subdivided into two equal parts P_1P_3 & P_3P_2 both are tested for visibility & found to be partially visible.

Both line segments are then subdivided in two equal parts to get midpoints P4 & P5.

It is observed that line P_1P_4 & P_5P_2 are completely invisible so rejected

Hence, line segment P_3P_5 is completely visible & hence drawn.

The remaing line segment P_4P_3 still partially visible.

It is then subdivided to get midpoint P_6

It is observed that P_6P_3 is completely visible whereas P_4P_6 is partially visible

Thus P_6P_3 line segment is drawn & P_4P_6 line segment is further divided into equal parts to get midpoints P_7. Now it is observed that line segment P_4P_7 is completely invisible & line segment P_7P_6 is completely visible & no there is no further partially visible segment

The eqn for line passing through points $P_1(x_1,y_1)$ & $P_2(x_2,y_2)$ is $y = m(x-x_1)+y_1$ or $y = m(x-x_2)+y_2$ where, $m = y_2-y_1/x_2-x_1$. Therefore, intersection with clipping boundaries of window are give as

① Left : x_L, $y = m(x_L-x_1)+y_1$; $m \neq \infty$

② Right : x_R, $y = m(x_R-x_1)+y_1$; $m \neq \infty$

③ Top : y_T, $x = x_1+(\frac{1}{m})(y_T-y_1)$, $m \neq 0$

④ Bottom: y_B, $x = x_1+(\frac{1}{m})(y_B-y_1)$; $m \neq 0$

Algorithm

1. Read two endpoints of line say $P_1(x_1,y_1)$ & $P_2(x_2,y_2)$
2. Read two corner (left, top & right bottom) of the window, say Wx_1, Wy_1 & Wx_2, Wy_2
3. Assign region code for two endspoints using foll^n step:

   Initialize code with bit 0000

   set bit 1 - if $(x < Wx_1)$

   set bit 2 - if $(x > Wx_2)$

   set bit 3 - if $(y < Wy_1)$

   set bit 4 - if $(y > Wy_2)$
4. check for visibility line

   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line & go to step 6

   b) If the region codes for the endpoints are not zero & the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line & go to step 6

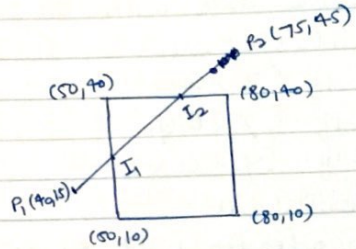
   c) If the region code for two endpoints do not satisfy the condition in 4a) & 4b) the line is partially visible
5. Divide the partially visible line segment in equal parts & repeat step 3 through 5 for both subdivided line segment until you get completely visible & invisible line segment
6. Stop

AND OR

eg:- $P_1(40,15)$ - $P_2(75,45)$ & $P_3(70,20)$ - $P_4(100,10)$ against window A(50,10), B(80,10), C(80,40), D(50,40)

→ Line 1: $P_1(40,50)$, $P_2(75,40)$
$P_3(70,20)$, $P_4(100,10)$

$X_L = 50$, $Y_B = 10$
$X_R = 80$, $Y_T = 40$



Point	End code	Anding	
P_1	0001	0000	} partially visible
P_2	1000	0000	

X_L, $y = m(x_L - x_1) + y_1$

$= \frac{(y_2-y_1)}{(x_2-x_1)}(x_L-x_1)+y_1 = \frac{30}{35}(50-40)+15$

$= \frac{6}{7}(10)+15$

$= 23.57$

Y_T, $x = x_1 + \left(\frac{1}{m}\right)(Y_T - y_1)$

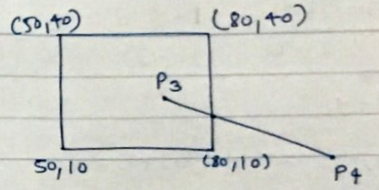
$= 40 + \left(\frac{7}{6}\right)(40-15)$

$= 69.16$

so, $I_1(50, 23.57)$, $I_2 = (69.19, 40)$

Line 2 : $P_3(70,20)$ $P_4(100,10)$

Point	Endpoint	Anding	
P_3	0000	0000	Partially visible
P_4	0010	0000	



X_R, $y = m(x_R - x_1) + y_1$

$= \frac{(y_2-y_1)}{(x_2-x_1)}(x_R-x_1)+y_1$

$= \left(\frac{10-20}{100-70}\right)(80-70)+20$

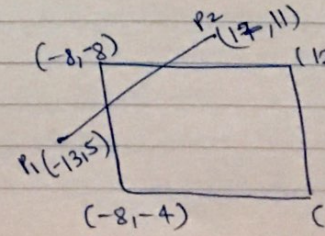
$= \left(\frac{-10}{30}\right)(10)+20$

$= \frac{-10}{3}+20 = 16.66$

$I = (80, 16.66)$.

Q. use outcode based line clipping method to clip a line starting from (-13,5) & ending at (17,11) against the window having its lower corner at (-8,-4) & upper right corner at (12,8).

→



$X_L = -8$
$Y_B = -8$
$X_R = 12$
$Y_T = 8$

$P_1(-13,5)$ & $P_2(17,11)$

Point	End code	Anding	Position
P1	0001	0000	partially visible
P_2	1010	0000	partially visible

Slope of line $= m = \frac{y_2-y_1}{x_2-x_1} = \frac{17-5}{17-(-13)} = \frac{6}{30} = \frac{1}{5}$

$x_L, y = m(x_L - x) + y_1$

$= \frac{1}{5}(-8+13)+5 = \frac{1}{5}(5)+5 = 6$

$I_1 = (-8,6)$

$y_T, x = x_1 + \left(\frac{1}{m}\right)(y_T - y)$

$= -13 + (5)(8-5)$

$= -13 + 5*3$

$= -13 + 15$

$= 2$

$I_2(2,8)$.

Cyrusbeck

It is generalized line a clipping algorithm. It was designed to be more efficient than the cohen_sutherland algo, which uses repetitive clipping.

Cyrus_Beck is used with convex polygon clipping window. unlike sutherland_cohen - which can be used only on rectangular clipping area.

Cyrus Beck generally clips only once or twice unlike the cohen sutherland algo where the lines are clipped about four times.

Clipping the line twice has the idea that the line will be clipped for the 1st time when it enteres the boxe & 2nd time when it exists.

Functionality

Parametric eqn of the line segment from P_1 to P_2 is

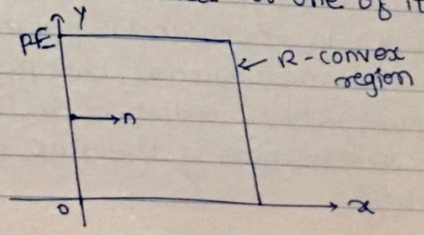
$$P(t) = P_1 + (P_2 - P_1)t \quad ; \quad 0 \le t \le 1$$

where t is parameter

$t=0$ at P_1

$t=1$ at P_2

consider a convex clipping region R, f is boundary point of the convex region R, & n is inner normal for one of its boundaries



$x(t) = x_0 + t(x_1 - x_0)$

$y(t) = y_0 + t(y_1 - y_0)$

Now to find intersection point with the clipping window we calculate value of dot product

Let P_E be point on the clipping plane E

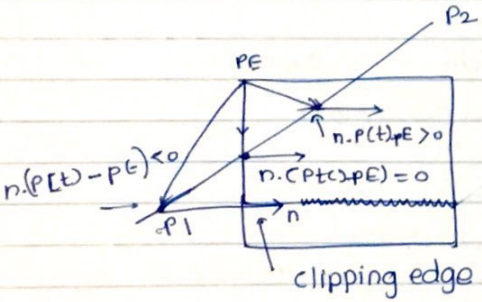
Calculate $n.(p(t) - pE)$.

(+ve) if > 0 vector pointed toward interior

if $= 0$ vector pointed parallel to the plane containing P

if < 0 (-ve) vector pointed away from interior

- Hence, n stands for normal of the current clipping plane (pointed away from interior)
- By this we select the point of intersection of line & clipping window where (dot product = 0) & hence clip the line.



if point PE lies in the boundary plane or edge for which n is inner normal, the point t on the line P(t) which satisfies $n.(P(t) - PE) = 0$ condn is intersection of the line with boundary edge

S.N.J.B.'s

SHRI H. H. J. B. POLYTECHNIC, CHANDWAD

CLASS TEST 1 / II (201 - 201)

Course & Semester :		Roll No.:	Date : / /201
Name of Subject :			Marks obtained :
Sign. of Supervisor :		Sign. of Sub. Examiner :	
Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

Supplied By : SNJB

000572

Algorithm

1. Read two end points of the line, say P_1 & P_2
2. Read the vertex co-ordinates of the clipping window
3. Calculate $P_2 - P_1$.
4. Assign boundary point (PE) with particular edge
5. Find inner normal vector for corresponding edge (n)
6. Calculate
   $(P_2 - P_1).n$
   & $P_1 - PE$
7. if $(P_2 - P_1)n > 0$

$$t_L = \frac{-(P_1 - PE).n}{(P_2 - P_1)n}$$

else

$$t_U = -\frac{(P_1 - PE)n}{(P_2 - P_1)n}$$

end if

8. Repeat step 4 through 7 for each edge of the clipping window
9. Find maximum lower limit & minimum upper limit
10. If maximum lower limit & minimum upper limit do not satisfy condition $0 \le t \le 1$ then ignore the line
11. Calculate the intersection points by substituting values of maximum lower limit & minimum upper

upper limit in the parametric eqn of the line P_1P_2.

(12) Draw the line segment $P(t_L)$ to $P(t_U)$

(13) stop.

Liang Barsky Algo

It uses parametric eqn of line & inequalities describing the range of the clipping box to determine the intersections between the line & clipping box. With this intersections it knows which portion of the line should be drawn.

These parametric eqn are given as

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y \qquad 0 \le t \le 1$$

where,

$\Delta x = x_2 - x_1$ & $\Delta y = y_2 - y_1$

The point clipping conditions ~~for~~ is the parametric form can be given as

$$x_{wmin} \le x_1 + t\Delta x \le x_{wmax} \;\&$$

$$y_{wmin} \le y_1 + t\Delta y \le y_{wmax}$$

Liang-Barsky express these four inequalities with two parameters p & q as follows:

$$tp_i \le q_i \qquad i = 1,2,3,4$$

where,

parameters p & q are defined as

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

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

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

$p_4 = \Delta y$ $\qquad q_4 = y_{wmax} - y_1$

following observations can be easily made from above definitions of parameter p & q

If $p_1 = 0$: Line is parallel to left clipping boundary.

If $p_2 = 0$: right

If $p_3 = 0$: bottom

If $p_4 = 0$: top

If $p_i = 0$: Line is ||el to one of the clipping boundaries corresponding to value of i.

If $q_i < 0$: Line is completely outside the boundary & can be eliminated.

If $q_i \geq 0$: Line is inside the clipping boundary.

If $p_i < 0$: Line proceeds from outside to inside of the clipping boundary.

If $p_i > 0$: Line proceeds from inside to outside of the clipping boundary.

Therefore, for nonzero value of p_i, the line crosses the clipping boundary & we have to find parameter t. The parameter t for any clipping boundary i can be given as

$$t = \frac{q_i}{p_i} \qquad i = 1, 2, 3, 4.$$

This algo calculates two values of parameter t: t_1 & t_2 that define that part of the line that lies within the clip rectangle. The value of t_1 is determined by checking the rectangle edges for which the line proceeds from outside to inside ($p < 0$). The value of t_1 is taken as largest value among various values of intersection with all edges.

On the other hand, the value of t_2 is determined by checking the rectangle edges for

for which the line proceeds from the inside to outside ($p>0$). The minimum of calculated value is taken as value for t_2.

Now if $t_1 > t_2$, the line is completely outside the clipping window & it can be rejected. Otherwise the values of t_1 & t_2 are substituted in the parametric eqn to get end points of the clipped line.

Algorithm

1. Read two endpoints of the line say $P_1(x_1, y_1)$ & $P_2(x_2, y_2)$
2. Read two corners (left-top & right-bottom) of the window, say ($x_{wmin}, y_{wmax}, x_{wmax}, y_{wmin}$)
3. calculate the values of parameters pi & qi for i=1,2,3,4 such that

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

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

$q_1 = -\Delta y$ $\quad q_3 = y_1 - y_{wmin}$

$q_2 = \Delta x$ $\quad q_4 = y_{wmax} - y_1$

4. if pi=0 then
   { The line is ||el to ith boundary
   Now if $q_i < 0$ then
   { line is completely outside the boundary, hence, discard the line segment & goto.
   stop
   }
   else
   { check whether the line is horizontal or vertical & according check the line endpoint with corresponding boundaries. If line endpoints lie within the bounded area then use them to draw line otherwise use boundary co-ordinates to draw line. Go to stop.
   }
   }
5. Initialize values for t_1 & t_2 as $t_1 = 0$ & $t_2 = 1$
6. Calculate values for q_i/p_i for i=1,2,3,4.
7. Select values of q_i/p_i where $p_i < 0$ & assign max out of them as t_1.
8. Select values of q_i/p_i where $p_i > 0$ & assign min out of them as t_2.
9. If ($t_1 < t_2$)
   { calculate the endpoints of the clipped line as follows:
   $xx_1 = x_1 + t\Delta x$
   $xx_2 = x_1 + t_2\Delta x$
   $yy_1 = y_1 + t_1\Delta y$
   $yy_2 = y_1 + t_2\Delta y$
   Drawline (xx_1, yy_1, xx_2, yy_2)
   }
10. Stop.

eg:- ① Apply the Liang Barsky algo to the line with co-ordinates (30,60) & (60,25) against the window (X_{min}, Y_{min}) = (10,10) & (X_{max}, Y_{max}) = (50,50)

→ Here,

$x_1 = 30$ $\quad x_{wmin} = 10$

$y_1 = 60$ $\quad y_{wmin} = 10$

$x_2 = 60$ $\quad x_{wmax} = 50$

$y_2 = 25$ $\quad y_{wmax} = 50$

$p_1 = -\Delta x = -(60-30) = -30$

$p_2 = \Delta x = (60-30) = 30$

$p_3 = -\Delta y = -(25-60) = 35$

$p_4 = \Delta y = (25-60) = -35$

$q_1 = x_1 - x_{wmin} = 30-10 = 20$

$q_2 = x_{wmax} - x_1 = 50-30 = 20$

$q_3 = y_1 - y_{wmin} = 60-10 = 50$

$q_4 = y_{wmax} - y_1 = 50-60 = -10$

$\frac{q_1}{p_1} = \frac{20}{(-30)} = -0.667$

$\frac{q_2}{p_2} = \frac{20}{30} = 0.667$

$\frac{q_3}{p_3} = \frac{50}{35} = 1.4285$

$\frac{q_4}{p_4} = \frac{(-10)}{(-35)} = 0.2857$

$t_1 = \max(-0.667, 0.2857)$

$= 0.2857$

$t_2 = \min(0.667, 1.4285)$

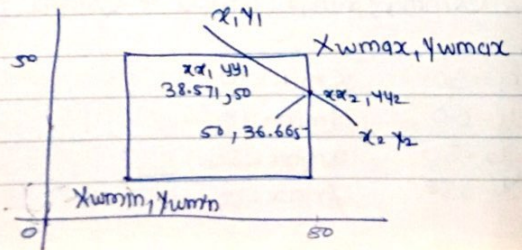
$= 0.667$

$xx_1 = x_1 + t_1 \Delta x = 30 + 0.2857 \times 30 = 38.571$

$yy_1 = y_1 + t_1 \Delta y = 60 + 0.2857 \times (-35) = 50$

$xx_2 = x_1 + t_2 \Delta x = 30 + 0.667 \times 30 = 50$

$yy_2 = y_1 + t_2 \Delta y = 60 + 0.667 \times (-35) = 36.665$



S.N.J.B.'s

SHRI H. H. J. B. POLYTECHNIC, CHANDWAD

CLASS TEST I / II (201 - 201)

Course & Semester :	Roll No.:	Date : / /201
Name of Subject :		Marks obtained :
Sign. of Supervisor :	Sign. of Sub. Examiner :	

Q. No. 1	Q. No. 2	Q. No. 3	Total Marks

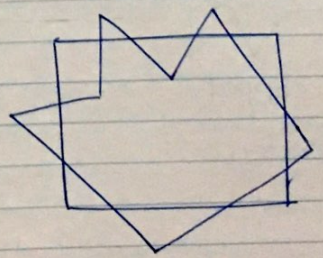
Supplied By : SNJB

000573

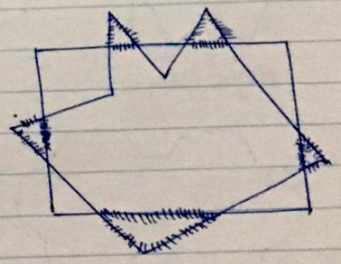
Polygon clipping

Polygon is nothing but the collection of lines Therefore, we might think that the line clipping algo can be used directly for polygon clipping. However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines. Thus, we need to modify the line clipping algo to clip polygons.

We consider polygon as closed solid area. Hence after clipping it should remain closed. To achieve this we requiee an algorithm that will generate additional line segment which makes the polygon as close area.



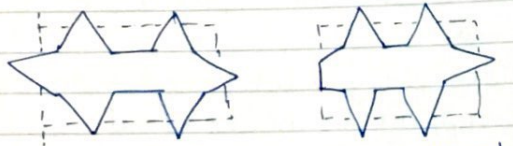
Before clipping



After clipping

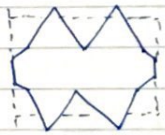
Sutherland - Hodgeman Polygon Clipping

A polygon can be clipped by processing its boundary as a whole against each window. edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce new sequence of vertices. The new set of vertices could then be successively passed to the right boundary clip, top boundary clipper & bottom boundary clipper, At each step a new the next window boundary clipper.

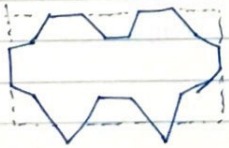


original polygon

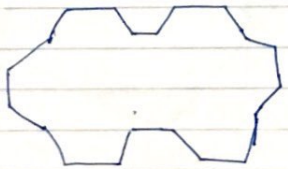
Left clipped



Right clipped



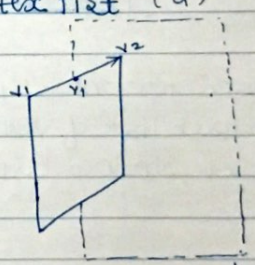
Top clipped



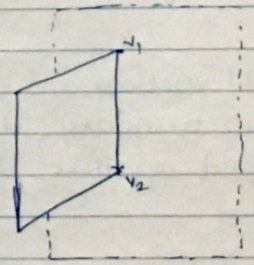
Bottom clipped

- The output of the algo is list of polygon vertices all of which are on the visible side of clipping plane. such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This result in 4 possible relationships between the edge & the clipping boundary or plane.

1. If the first of the edge is outside the window boundary & the second vertex of the edge is inside then the intersection point of the polygon edge with window boundary & the second vertex are added to output vertex list (a)



v_1 - outside
v_2 - inside
∴ save v_i & v_2
(a)

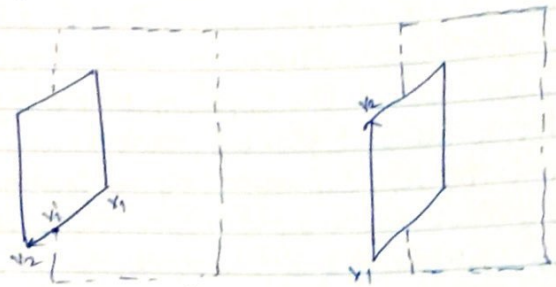


v_1 - inside
v_2 - inside
∴ save v_2
(b)

2. If both the vertices of edge are inside the window boundary, only the second vertex is added to output vertex list.

3. If the first vertex of edge is inside the window boundary, only the second vertex is ~~added~~ outside, only the edge intersection with the window boundary is added to the Output vertex list

4. If both vertices of the edge are outside the window boundary, nothing is added to the output list



V_1 - inside
V_2 - outside
∴ Save V_1'
(c)

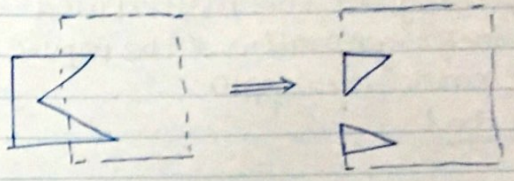
V_1 - outside
V_2 - outside
∴ save nothing.
(d)

once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary

Algorithm

1. Read co-ordinates of all vertices of the polygon
2. Read co-ordinates of clipping window
3. Consider left edge of the window.
4. Compare the vertices of each edge of the polygon, individually with the clipping plane
5. Save the resulting intersections & vertices in the new list of vertices according to four possible relationships between the edge & clipping boundary.
6. Repeat the step 4 & 5 for remaining edges of clipping window. each time the resultant list of vertices is successively passed to process the next edge of the clipping.
7. stop.

In concave polygon, then extra edges appears in final clipped.



The problem of extra edges occurs since we are storing all the vertices in single vertex list & to form a close Close Figure, we are forming an edge between last & first vertex

To overcome this problem, one way is to split the concave polygon. It means we will divide single concave polygon into two or more convex polygon & process each convex polygon separately. But dividing concave polygon is not simple method so we will go for more general clipping algo such as Weiler Atterton polygon clipping algo.

Text clipping

There are no of text clipping techniques that can be used to provide text clipping in a graphics package.

We have to choose the text clipping technique depending on the method used for the character generation & the requirements of the particular appln

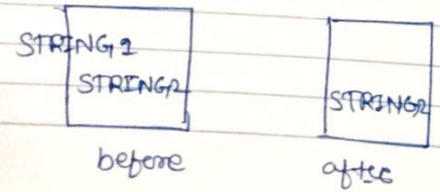
Three Method

1. All or none string clipping
2. All or none character clipping
3. Text clipping

1) All or none string clipping :-

If all the string is inside a clip window, we keep it. Otherwise, the string is discarded. This procedure is implemented by considering a boundary rectangle around the text pattern.

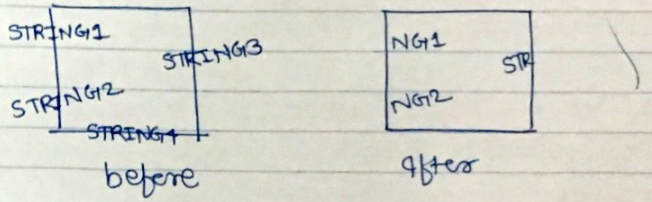
The boundary position of rectangle are then compared to the window boundaries, & the string is rejected if there is any overlap. This method gives fastest text clipping



before after

2) All or none character clipping

In this method, we discard only those characters that are not completely inside the window. In this case, the boundary limits of individual characters are compared to window & characters which either overlap or outside a window boundary are clipped.



before after

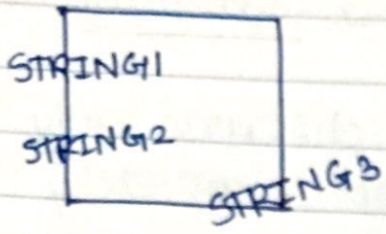
3) Text clipping :-

In this method, either a part of character that is not inside the window boundary or a character outside a window boundary are clipped.

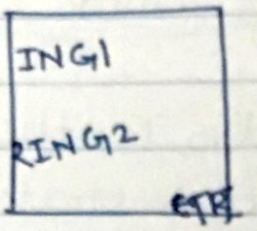
Here, we treat characters in much the same way that we treated lines.

The individual lines which forms the character are processed by line clipping algo to clip the outside part of the overlapping character with a clip window boundary.

When characters are defined by bit map, they are clipped by comparing relative positions of individual pixels in character grid patterns to the clipping boundaries.



before



after