

# COMPUTER GRAPHICS & IMAGE PROCESSING

## MODULE 3

NAMITHA RAMACHANDRAN

## **Module - 3 (Clipping and Projections)**

Window to viewport transformation. Cohen Sutherland Line clipping algorithm. Sutherland Hodgeman Polygon clipping algorithm. Three dimensional viewing pipeline. Projections- Parallel and Perspective projections. Visible surface detection algorithms- Depth buffer algorithm, Scan line algorithm.

## **Window to viewport transformation.**

**Window to Viewport Transformation** is the process of transforming 2D world-coordinate objects to device coordinates. Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

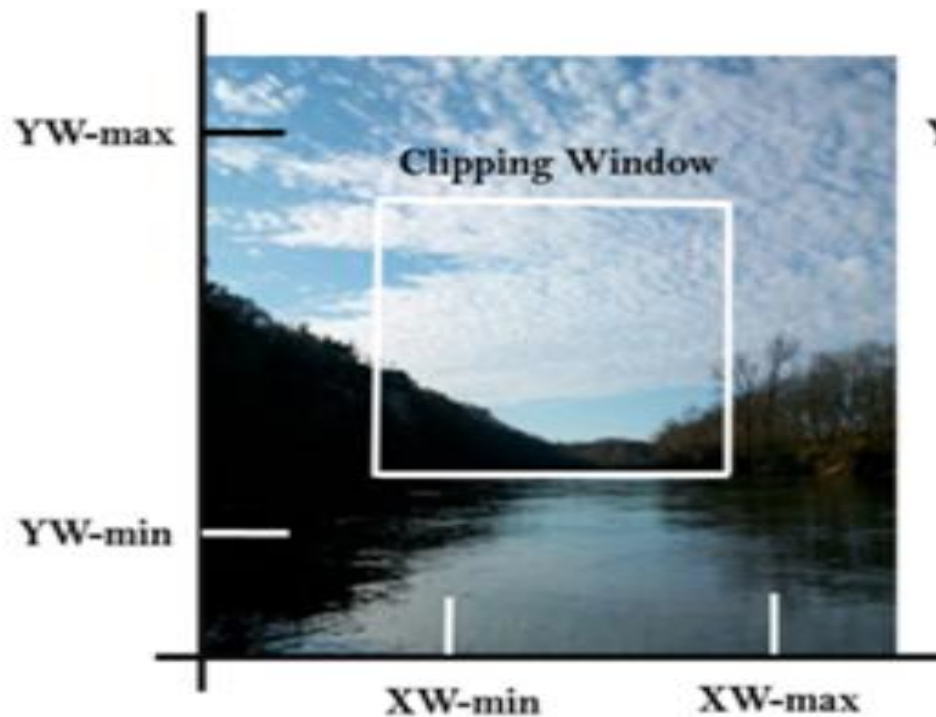
**A world-coordinate area selected for display is called a window.**

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

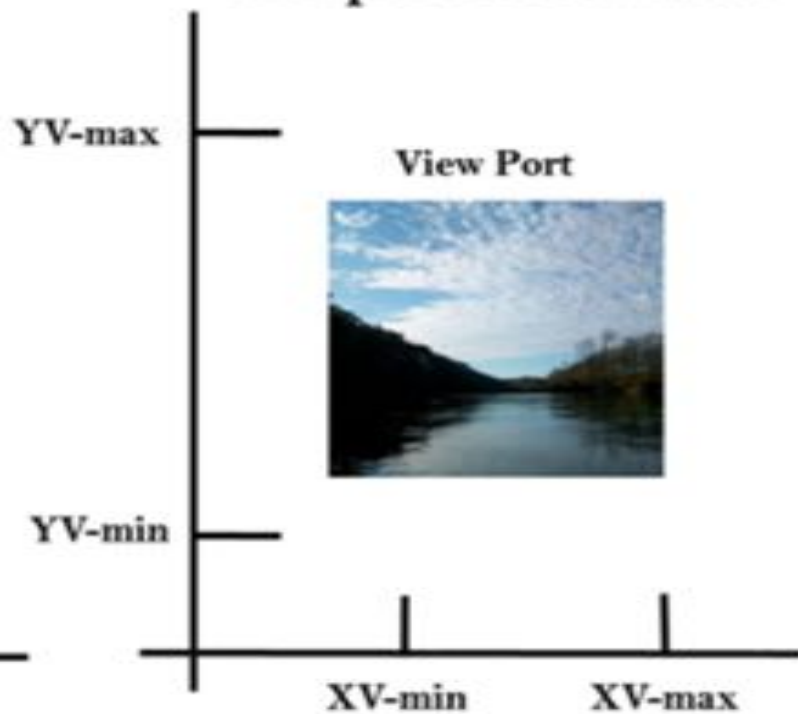
The window defines what is to be viewed;

The viewport defines where it is to be displayed.

## World Coordinates



## Viewport Coordinates



**Fig: Window to viewport mapping**

**World coordinate** – It is the Cartesian coordinate w.r.t which we define the diagram, like  $X_{wmin}$ ,  $X_{wmax}$ ,  $Y_{wmin}$ ,  $Y_{wmax}$

**Device Coordinate** –It is the screen coordinate where the objects are to be displayed, like  $X_{vmin}$ ,  $X_{vmax}$ ,  $Y_{vmin}$ ,  $Y_{vmax}$

**Window** –It is the area on world coordinate selected for display.

**ViewPort** –It is the area on the device coordinate where graphics is to be displayed.

It may be possible that the size of the Viewport is much smaller or greater than the Window. In these cases, we have to increase or decrease the size of the Window according to the Viewport and for this, we need some mathematical calculations.

$(x_w, y_w)$ : A point on Window

$(x_v, y_v)$ : Corresponding point on Viewport

we have to calculate the point  $(x_v, y_v)$

WINDOW COORDINATE □ NORMALISED COORDINATE □ DEVICE COORDINATES  
(within (0,1))

**Window coordinate to normalized coordinate translation .**

**Scaling in the normalised coordinate system in order to make it match with viewport(device coordinate system) .**

**Translate to Device coordinate (view port ) system.**

In order to maintain the same relative placement of the point in the viewport as in the window, we require

Normalized Point on Window

$$\left( \frac{X_W - X_{Wmin}}{X_{Wmax} - X_{Wmin}}, \frac{Y_W - Y_{Wmin}}{Y_{Wmax} - Y_{Wmin}} \right)$$

Normalized Point on Viewport

$$\left( \frac{X_V - X_{Vmin}}{X_{Vmax} - X_{Vmin}}, \frac{Y_V - Y_{Vmin}}{Y_{Vmax} - Y_{Vmin}} \right)$$

Now the relative position of the object in Window and Viewport are same



$$\frac{X_w - X_{wmin}}{X_{wmax} - X_{wmin}} = \frac{X_v - X_{vmin}}{X_{vmax} - X_{vmin}}$$

$$\frac{Y_w - Y_{wmin}}{Y_{wmax} - Y_{wmin}} = \frac{Y_v - Y_{vmin}}{Y_{vmax} - Y_{vmin}}$$

$$X_v = X_{vmin} + (X_w - X_{wmin}) S_x$$

where  $s_x$  is the scaling factor of x coordinate and  $s_y$  is the scaling factor of y coordinate

$$S_x = \frac{X_{vmax} - X_{vmin}}{X_{wmax} - X_{wmin}}$$

$$Y_v = Y_{vmin} + (Y_w - Y_{wmin}) S_y$$

$$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}$$

- for window,  $X_{wmin} = 20$ ,  $X_{wmax} = 80$ ,  $Y_{wmin} = 40$ ,  $Y_{wmax} = 80$ .
- for viewport,  $X_{vmin} = 30$ ,  $X_{vmax} = 60$ ,  $Y_{vmin} = 40$ ,  $Y_{vmax} = 60$ .
- Now a point  $( X_w, Y_w )$  be  $( 30, 80 )$  on the window. We have to calculate that point on the viewport i.e  $( X_v, Y_v )$ .

$$S_x = ( 60 - 30 ) / ( 80 - 20 ) = 30 / 60$$

$$S_y = ( 60 - 40 ) / ( 80 - 40 ) = 20 / 40$$

- So, now calculate the point on the viewport (  $X_v, Y_v$  ).

$$X_v = 30 + ( 30 - 20 ) * ( 30 / 60 ) = 35$$

$$Y_v = 40 + ( 80 - 40 ) * ( 20 / 40 ) = 60$$

- So, the point on window (  $X_w, Y_w$  ) = ( 30, 80 ) will be

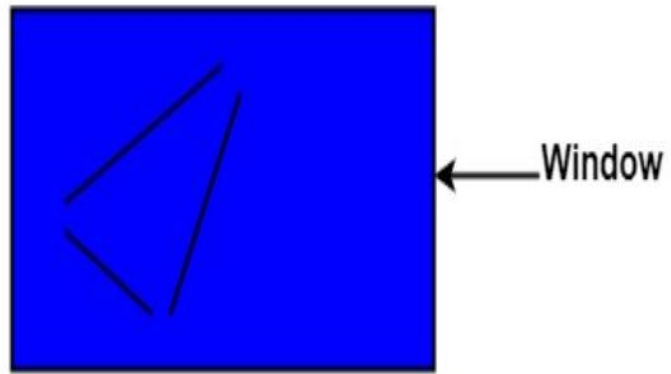
(  $X_v, Y_v$  ) = ( **35, 60** ) on viewport.

# 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.

Case1:



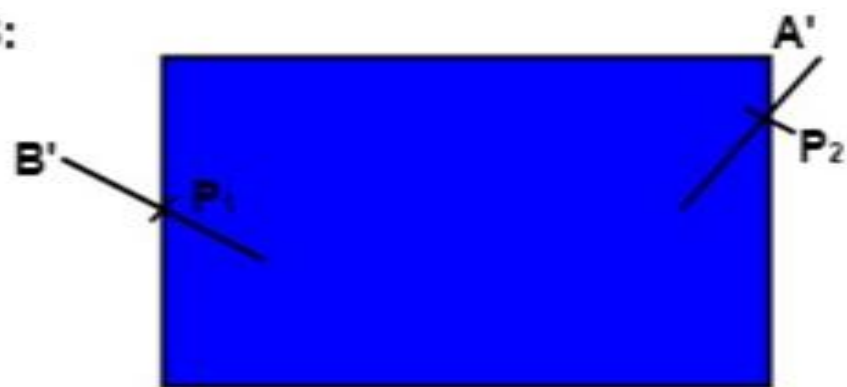
Visible Lines

Case2:



Invisible Lines

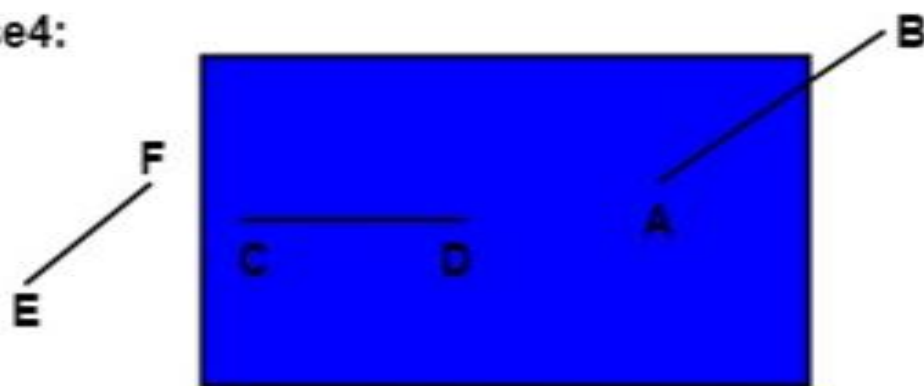
Case3:



Clipped Lines

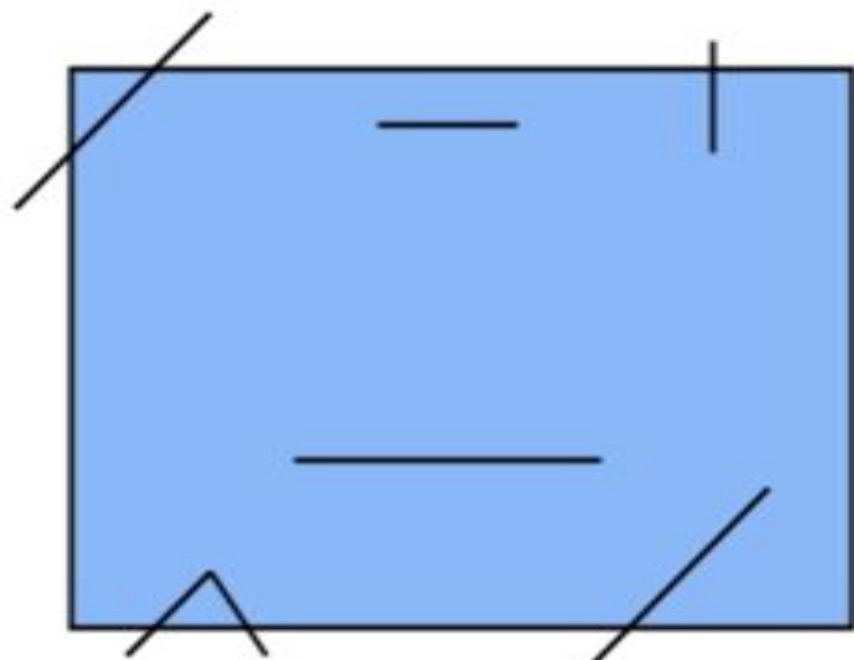
In this figure  $P_1$  and  $P_2$  are point of intersection. The line  $P_2$  to  $A'$  and  $P_1$  to  $B'$  is discarded or clipped.

Case4:

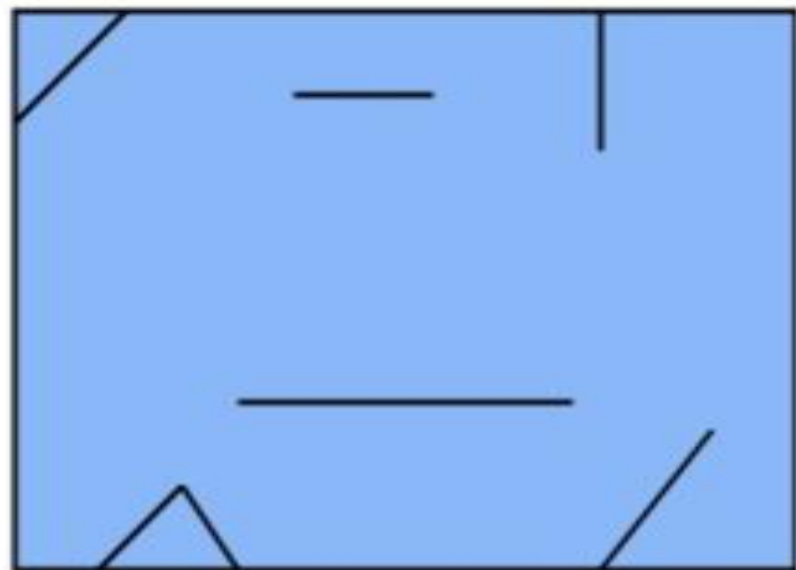


Mixed Lines

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



**Original Picture  
or  
Before Clipping**



**After Clipping**



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

Applications of clipping:

It will extract part we desire.

For identifying the visible and invisible area in the 3D or 2D object.

For creating objects using solid modeling.

For drawing operations.

For deleting, copying, moving part of an object.

Types of Clipping:

Point Clipping, Line Clipping, Area Clipping (Polygon), Curve Clipping

Text Clipping, Exterior Clipping

# Cohen Sutherland Line Clipping

Most popular line clipping algorithm ,speed up the processing of line segments by initial tests that reduce the number of intersections that must be calculated .

All lines come under any one of the following categories

**Visible:** If a line lies within the window, i.e., both endpoints of the line lie within the window. A line is visible and will be displayed as it is.

**Not Visible:** If a line lies outside the window, it will be invisible and rejected.

**Clipping Case:** If the line is neither visible case nor invisible case. It is considered to be the clipped case. Reject the portion of line that are outside the clipping window and accept portion that are inside the clipping window .

In case of clipping ,we need to find the intersecting point of line with window boundary

$$m = (y-y_1) / (x_{wmin}-x_1)$$

$$m(x_{wmin}-x_1) = (y-y_1)$$

$$m(x_{wmin}-x_1) + y_1 = y$$

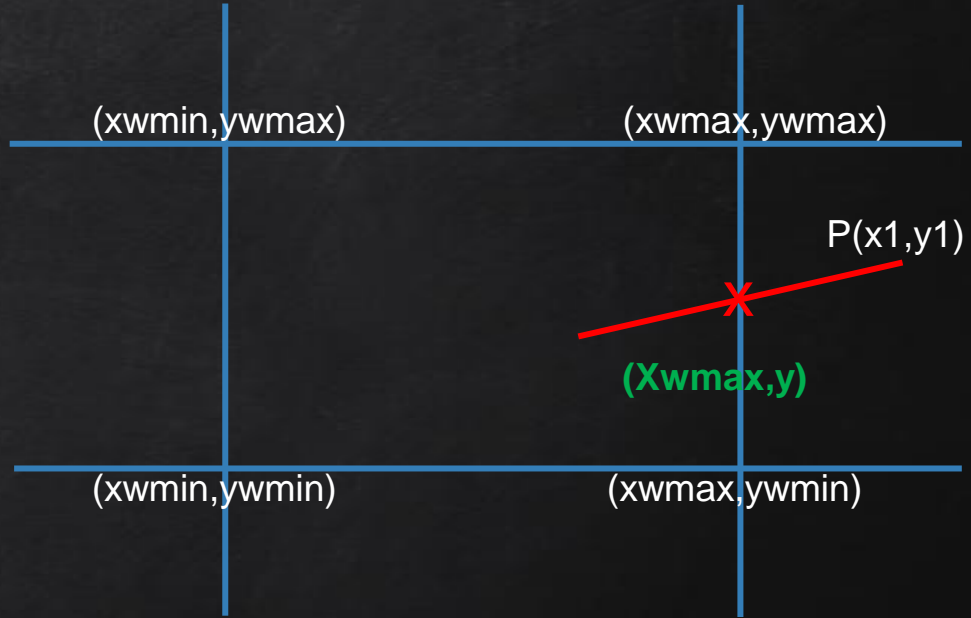


In case of clipping ,we need to find the intersecting point of line with window boundary

$$m = (y-y_1) / (x_{wmax}-x_1)$$

$$m(x_{wmax}-x_1) = (y-y_1)$$

$$m(x_{wmax}-x_1) + y_1 = y$$

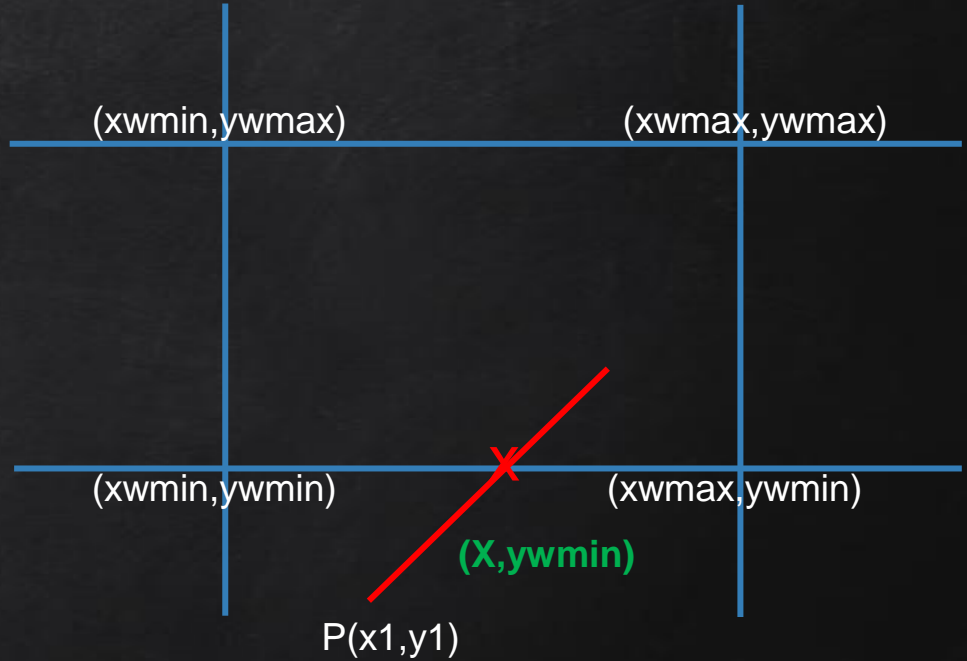


In case of clipping ,we need to find the intersecting point of line with window boundary

$$m = (ywmin-y1) / (x-x1)$$

$$m(x-x1) = (ywmin-y1)$$

$$X = \frac{(ywmin-y1)+x1}{m}$$

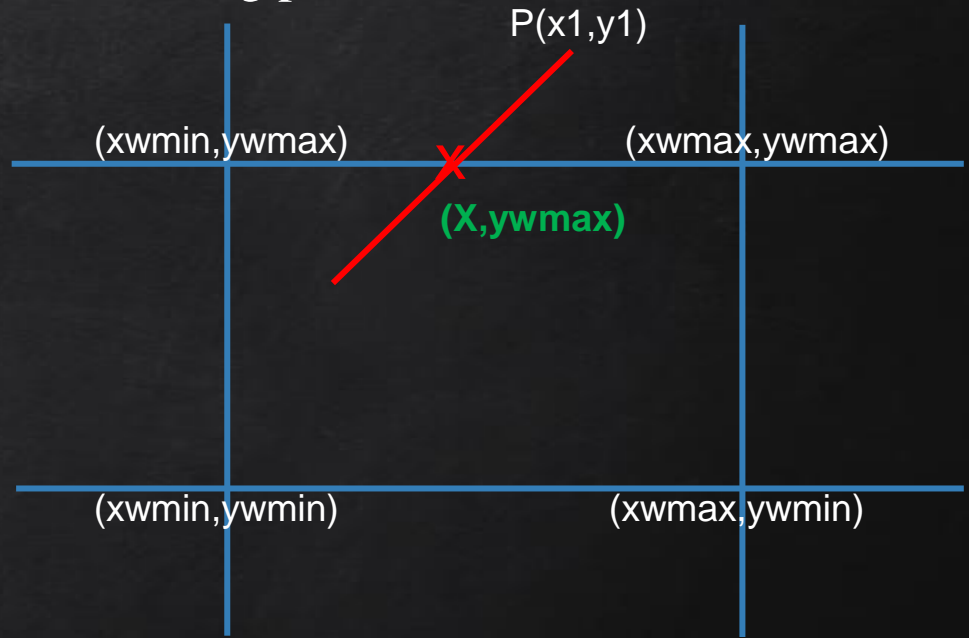


In case of clipping ,we need to find the intersecting point of line with window boundary

$$m = (ywmax - y1) / (x - x1)$$

$$m(x - x1) = (ywmax - y1)$$

$$X = \frac{(ywmax - y1) + x1}{m}$$



Every line endpoint in a picture is assigned a four-digit binary code, called a **region code** that identifies the location of the point relative to the boundaries of the clipping window. Regions are set up in reference to the boundaries of the clipping window.



bit4	bit3	bit2	bit1
above	below	right	left

If x is less than x<sub>min</sub>  
then bit number 1 is set.

If x is greater than x<sub>max</sub>  
then bit number 2 is set.

If y is less than y<sub>min</sub>  
then bit number 3 is set.

If y is greater than y<sub>max</sub>  
then bit number 4 is set

**(a) If bit 1 is "1" line intersects with left boundary of rectangle window**

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

where  $x = x_{wmin}$

where  $x_{wmin}$  is the minimum value of X co-ordinate of window,  $(x_1, y_1)$

**line endpoint coordinate .**

**(b) If bit 2 is "1" line intersecting with right boundary**

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

where  $x = x_{wmax}$

where  $x$  more is maximum value of  $x$  co-ordinate of the window,

**$(x_1, y_1)$  line endpoint coordinate .**



(c) If bit 3 is "1" line intersects with bottom boundary

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

where  $y = y_{wmin}$

$y_{wmin}$  is the minimum value of Y co-ordinate of the window, **(x1,y1) line endpoint coordinate .**

(d) If bit 4 is "1" line intersects with the top boundary

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

where  $y = y_{wmax}$

$y_{wmax}$  is the maximum value of Y co-ordinate of the window, **(x1,y1) line endpoint coordinate .**

Step 1 : Assign a region code for two endpoints of given line.

Step 2 : If both endpoints have a region code 0000 then given line is completely inside.

Step 3 : Else, perform the logical AND operation for both region codes.

Step 3.1 : If the result is not 0000, then given line is completely outside.

Step 3.2 : Else line is partially inside.

Step 3.2.1 : Choose an endpoint of the line that is outside the given rectangle

Step 3.2.2 : Find the intersection point of the rectangular boundary .

Step 3.2.3 : Replace endpoint with the intersection point and update the region code.

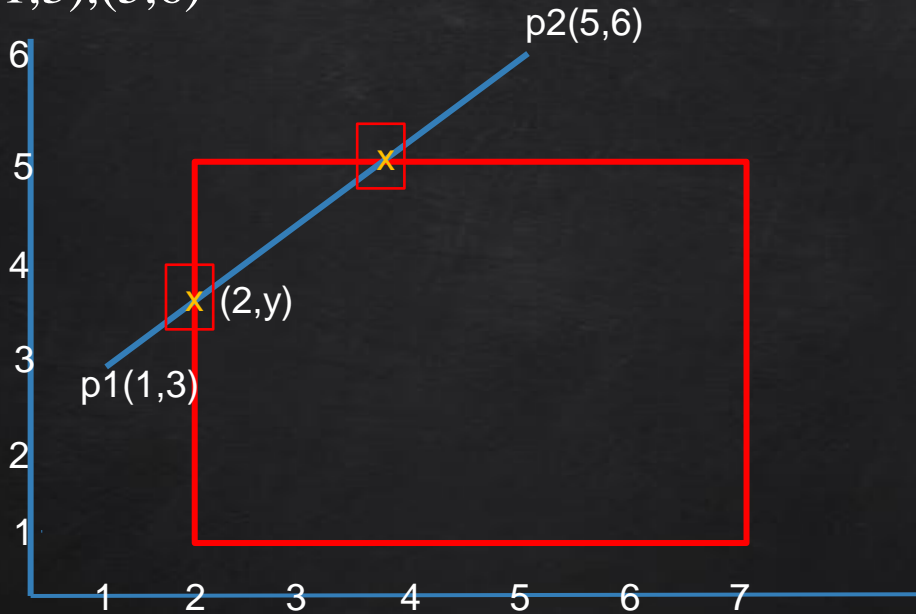
Step 3.2.4 : Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 : Repeat step 1 for other lines .

# Problem solving

Clipping window lower left(2,1) , upper right(7,5)

Line points (1,3),(5,6)



P1 region code:0001

P2 region code:1000

P1 AND P2

0001

1000

0000

P1 intersecting Left edge

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

$$x = x_{wmin}$$

$$m = (6-3)/(5-1) \\ = 3/4 = 0.75$$

$$x = 2$$

$$y = 3 + 0.75(2-1) \\ = 3.75$$

**(x,y)=(2,3.75) --new end point p1**

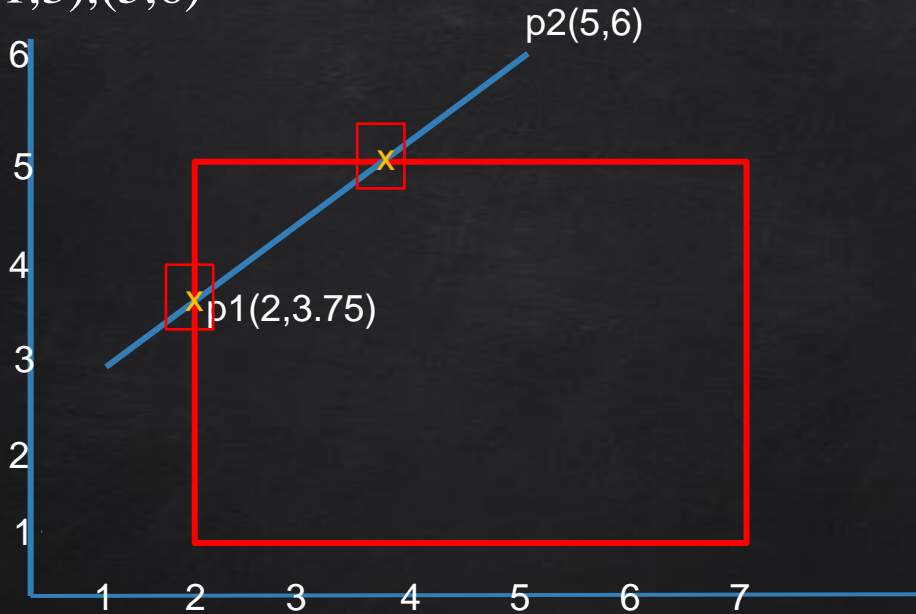
**P1 New region code 0000  
,completely inside**

NAMITHA RAMACHANDRAN

# Problem solving

Clipping window lower left(2,1) , upper right(7,5)

Line points (1,3),(5,6)



P1 region code:0000

P2 region code:1000

P1 AND P2

0000

1000

0000

P2 intersecting above edge

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

where  $y = y_{wmax}$

$$m = (6-3)/(5-1) \\ = 3/4 = 0.75$$

$y = 5$

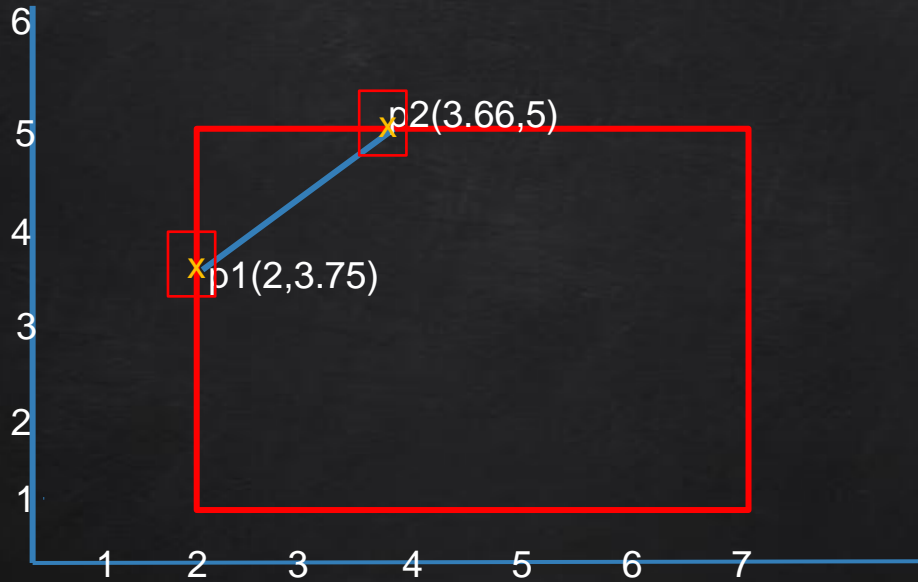
$$x = 5 + (5-6) / 0.75 \\ = 5 - (1.33) = 3.66$$

**(x,y)=(3.66,5) --new end point p2**

**P2 New region code 0000**

NAMITHA RAMACHANDRAN

# Problem solving

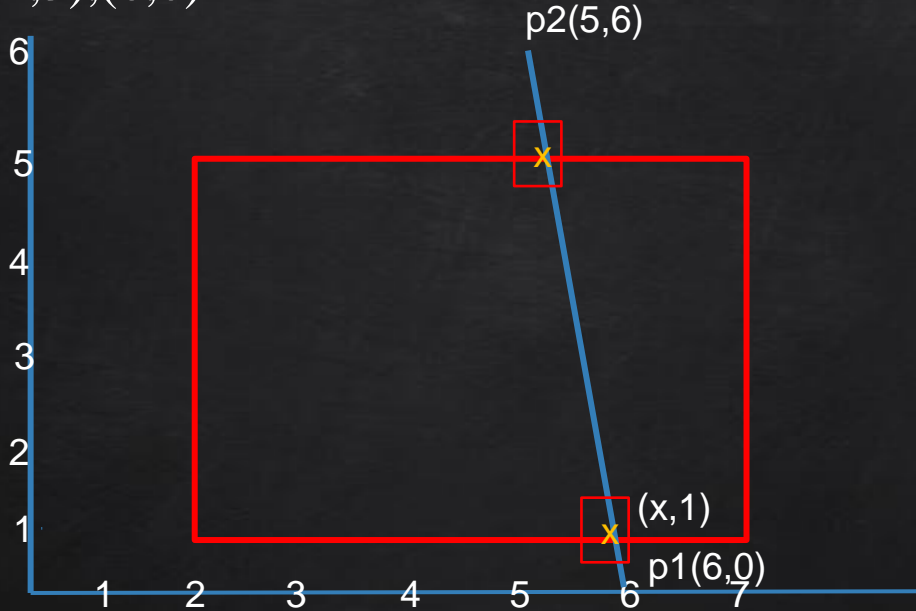


**P1 and p2 region codes 0000 ,  
Completely accept now and  
stop .**

# Problem solving

Clipping window lower left(2,1) , upper right(7,5)

Line points (1,3),(6,0)



P1 region code:0100

P2 region code:1000

P1 AND P2

0100

1000

0000

P1 intersecting below edge

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

where  $y = y_{wmin}$

$$m = (6-0)/(5-6) \\ = 6/-1 = -6$$

$y=1$

$$6+(1-0)/-.6 = 6-0.1667 \\ = 5.83$$

**(x,y)=(5.83,1) --new end point p1**

**P1 New region code 0000 ,**

NAMITHA RAMACHANDRAN

# Problem solving

Clipping window lower left(2,1) , upper right(7,5)

Line points (1,3),(6,0)



P1 region code:0100

P2 region code:1000

P1 AND P2

0000

1000

0000

P2 intersecting above edge

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

where  $y = y_{wmax}$

$$m = (6-0)/(5-6) \\ = 6/-1 = -6$$

Y=5

$$5 + (5-6)/-6 = -1/-6$$

$$5 + 0.167$$

$$= 5.167$$

**(x,y)=(5.167,5) --new end point**

**p2**

**P2 New region code 0000** , NAMITHA RAMACHANDRAN

# Problem solving

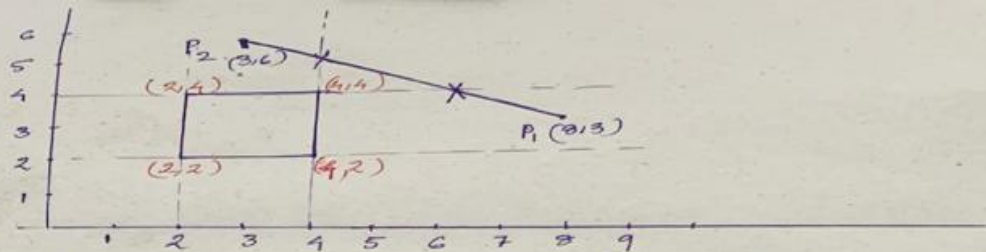
Clipping window lower left(2,1) , upper right(7,5)  
Line points (1,3),(6,0)



**P2 New region code 0000 ,**

**P1 and p2 region codes 0000 ,  
Completely accept now and  
stop .**





$$m = \frac{6-3}{3-0} = \frac{3}{-3} = -\frac{3}{3} = -0.6$$

$P_1 : 0010$   
 $P_2 : 1000$

$P_1$  AND  $P_2$   $\begin{array}{r} 0010 \\ 1000 \\ \hline 0000 \end{array}$  Find intersection point with above edge.

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

$$y = y_{\max}$$

$$y = 4$$

$$x = 0 + (4 - 3) / -0.6$$

$$= 0 + \frac{1}{-0.6} = -1.67$$

$$= 6.33$$

$$(x, y) \rightarrow (6.33, 4)$$

updated  $P_1$   $(6.33, 4)$  this point  $>$  Right Boundary

$P_1$  region code: 0010

$P_1$  AND  $P_2$   $\begin{array}{r} 0010 \\ 1000 \\ \hline 0000 \end{array}$  Find intersection point with Right boundary.

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

$$x = x_{\max}$$

$$x = 4$$

$$y = 4 + -0.6(4 - 3)$$

$$= 4 - 0.6(1)$$

$$= 4 - 0.6 = 3.4$$

$$(x, y) \rightarrow (4, 3.4)$$

updated  $P_1$   $(4, 3.4)$ ; Region code: 1000

$P_1 = 1000$

$P_2 = 1000$

same region code, not yet satisfied Complete acceptance condition. No further intersection with any boundary, so stop and completely reject.

# Sutherland Hodgeman Polygon clipping algorithm.

It is performed by processing the boundary of polygon against each window corner or edge. First of all entire polygon is clipped against one edge, then resulting polygon is considered, then the polygon is considered against the second edge, so on for all four edges.

If the first vertex is an outside the window, the second vertex is inside the window. The point of intersection of window boundary and polygon side (edge) is added to the output list, second vertex is also added to the output list. { outside to inside }.

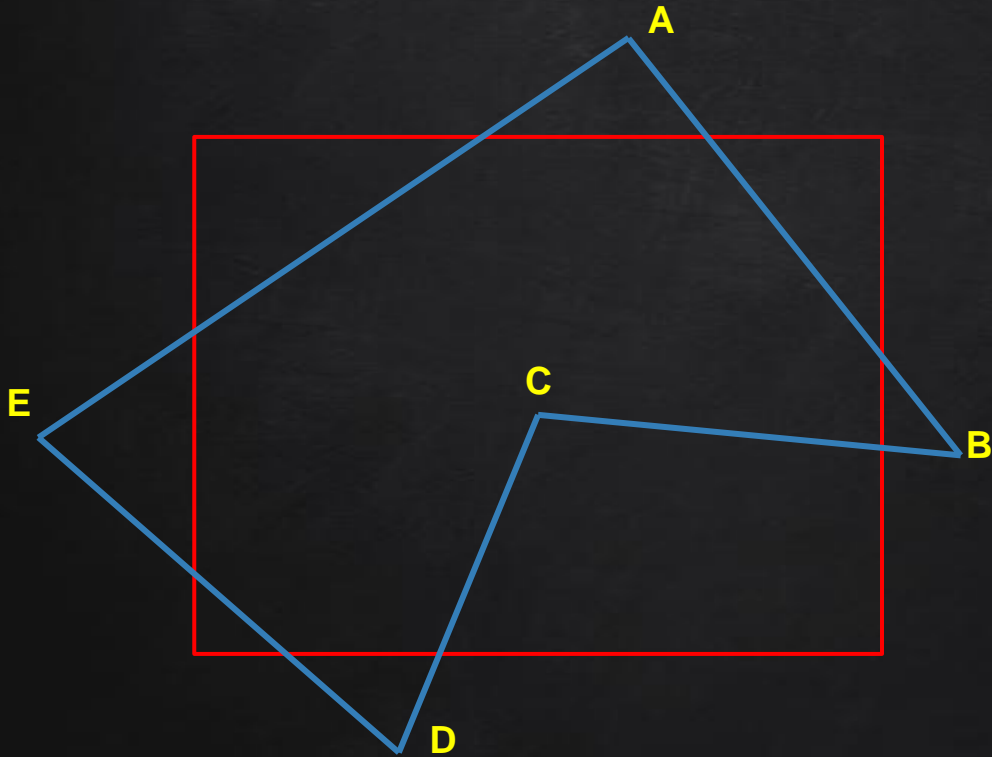
If both vertexes are inside window boundary. Then only second vertex is added to the output list .{ inside to inside }.

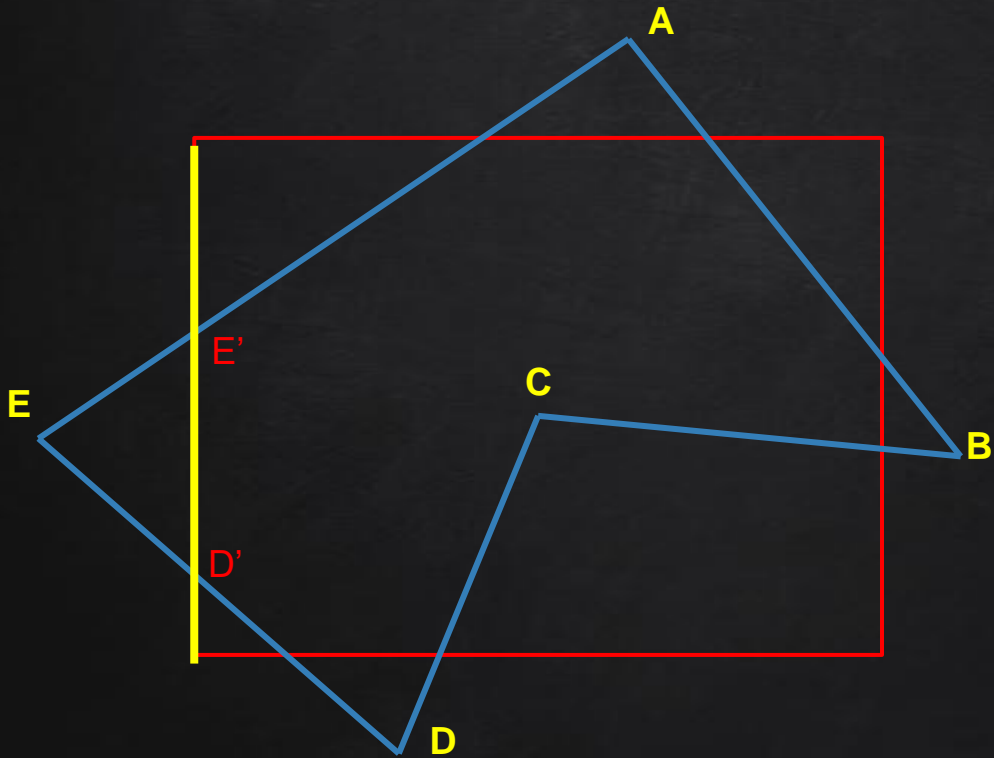
If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.{ inside to outside }.

If both vertices are the outside window, then nothing is added to output list.

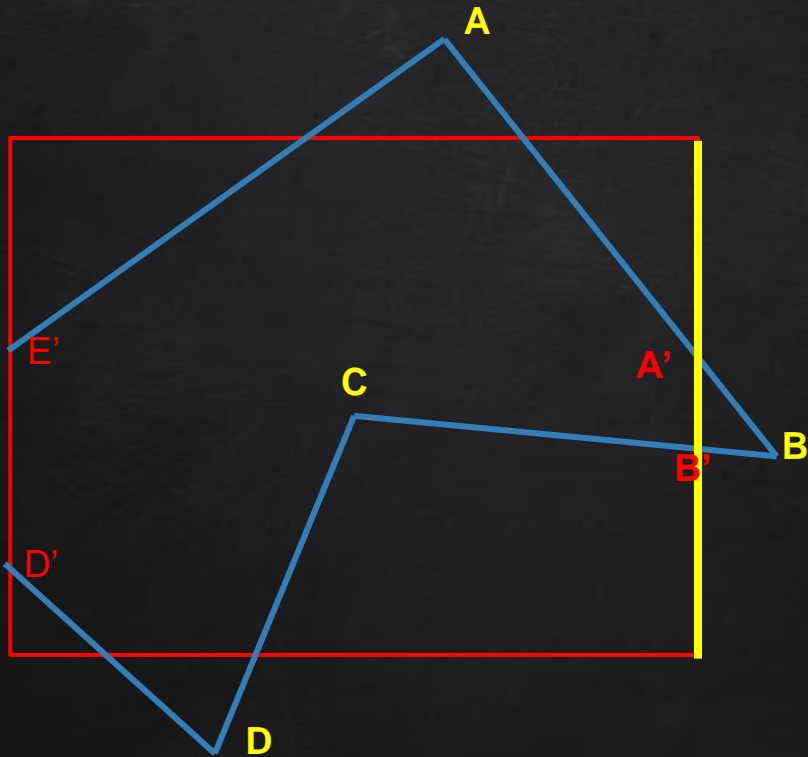
## Disadvantage of Cohen Hodgmen Algorithm:

This method requires a considerable amount of memory.

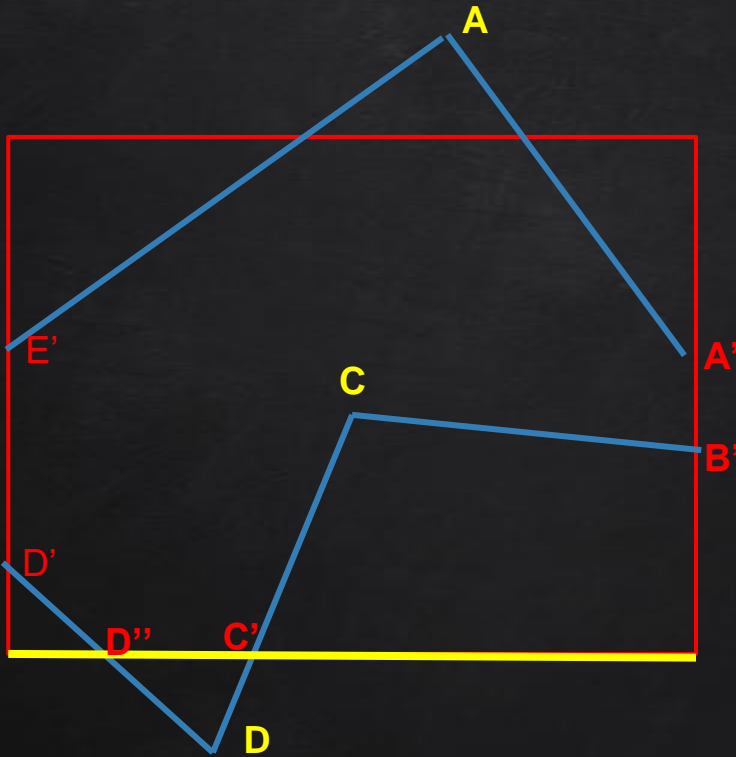




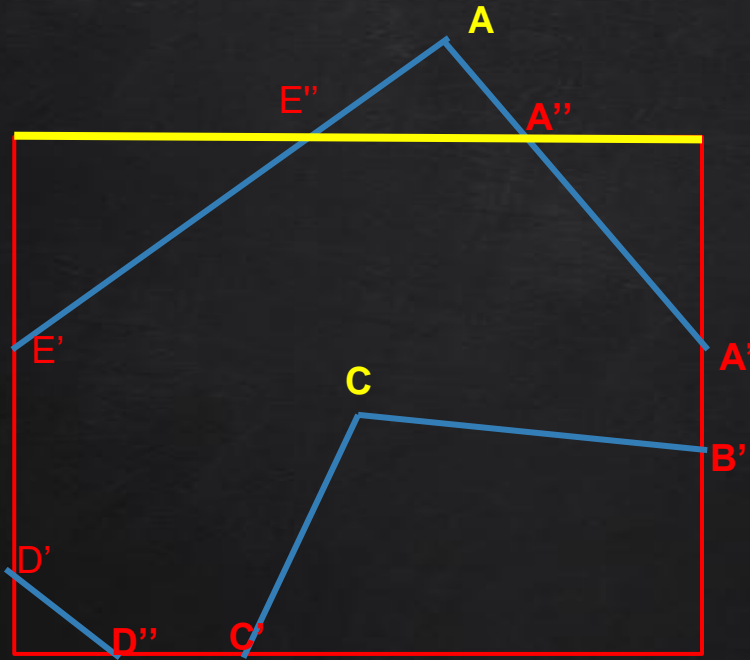
VERTEX	CASE	OUTPUT
AB	in <input type="checkbox"/>	B
BC	in <input type="checkbox"/>	C
CD	in <input type="checkbox"/>	D
DE	in <input type="checkbox"/> out	D'
EA	in <input type="checkbox"/> out	E'A



VERTEX	CASE	OUTPUT
AB	in $\square$ out	A'
BC	out $\square$ in	B'C
CD	in $\square$	D
DD'	in $\square$ in	D'
D'E'	in $\square$ in	E'
E'A	in $\square$ in	A

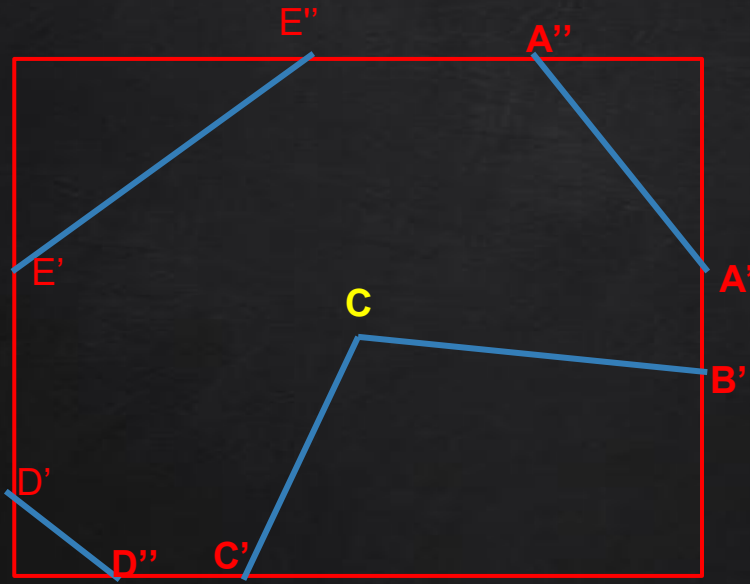


VERTEX	CASE	OUTPUT
AA'	in □ in	A'
A'B'	in □ in	B'
B'C	in □ in	C
CD	in □ out	C'
DD'	out □ in	D'' D'
D'E'	in □ in	E'
E'A	in □ in	A



VERTEX	CASE	OUTPUT
AA'	out $\square$ in	A'' A'
A'B'	in $\square$ in	B'
B'C	in $\square$	C
CC'	in $\square$ in	C'
C'D''	in $\square$ in	D''
D''D'	in $\square$ in	D'
D'E'	in $\square$ in	E'
E'A	in $\square$ out	E''

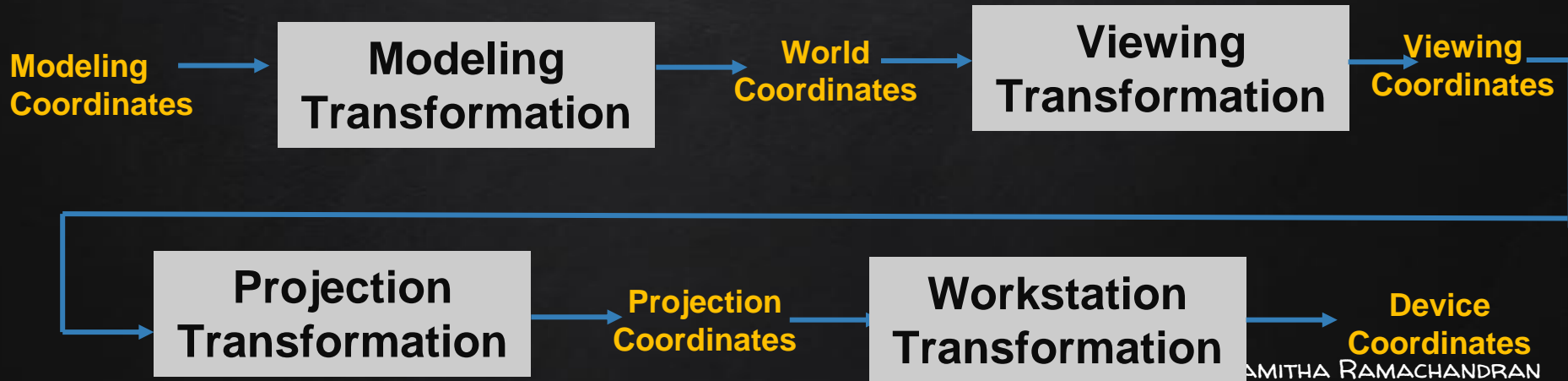




VERTEX	CASE	OUTPUT
AA'	out □ in	A'' A'
A'B'	in □ in	B'
B'C	in □ in	C
CC'	in □ in	C'
C'D''	in □ in	D''
D''D'	in □ in	D'
D'E'	in □ in	E'
E'A	in □ out	E''

# Three-dimensional viewing pipeline

The steps for computer generation of a **view** of a **three dimensional** scene are somewhat analogous to the processes involved in taking a **photograph**.



Construct the shape of individual objects in a scene within **modeling coordinate**,

Deciding the dimensions of how much of the scene to capture is **modeling transformation**,

and place the objects into appropriate positions within the scene in order to fit in the frame. **world coordinate**.

Setting the position and orientation of the camera is **viewing transformation**.

Appropriate scaling up or down is done so that proper view become available in camera window coordinate ( **viewing coordinate** ).

Convert the viewing coordinate description of the scene to coordinate positions on the projection plane by adjusting focusing of camera and direction of light is **projection transformation**,

so that it forms a image in the camera frame or film **projection**

**pcordinates**

Window to view port transformation, while click the camera button image formed in camera screen ( **device coordinate** ).

# PROJECTIONS

Once the world –coordinate descriptions of the objects in the scene are converted to viewing coordinates , we can project the 3D objects onto the 2D view plane.

**Two basic Projection methods: Parallel Projection, Perspective Projection.**

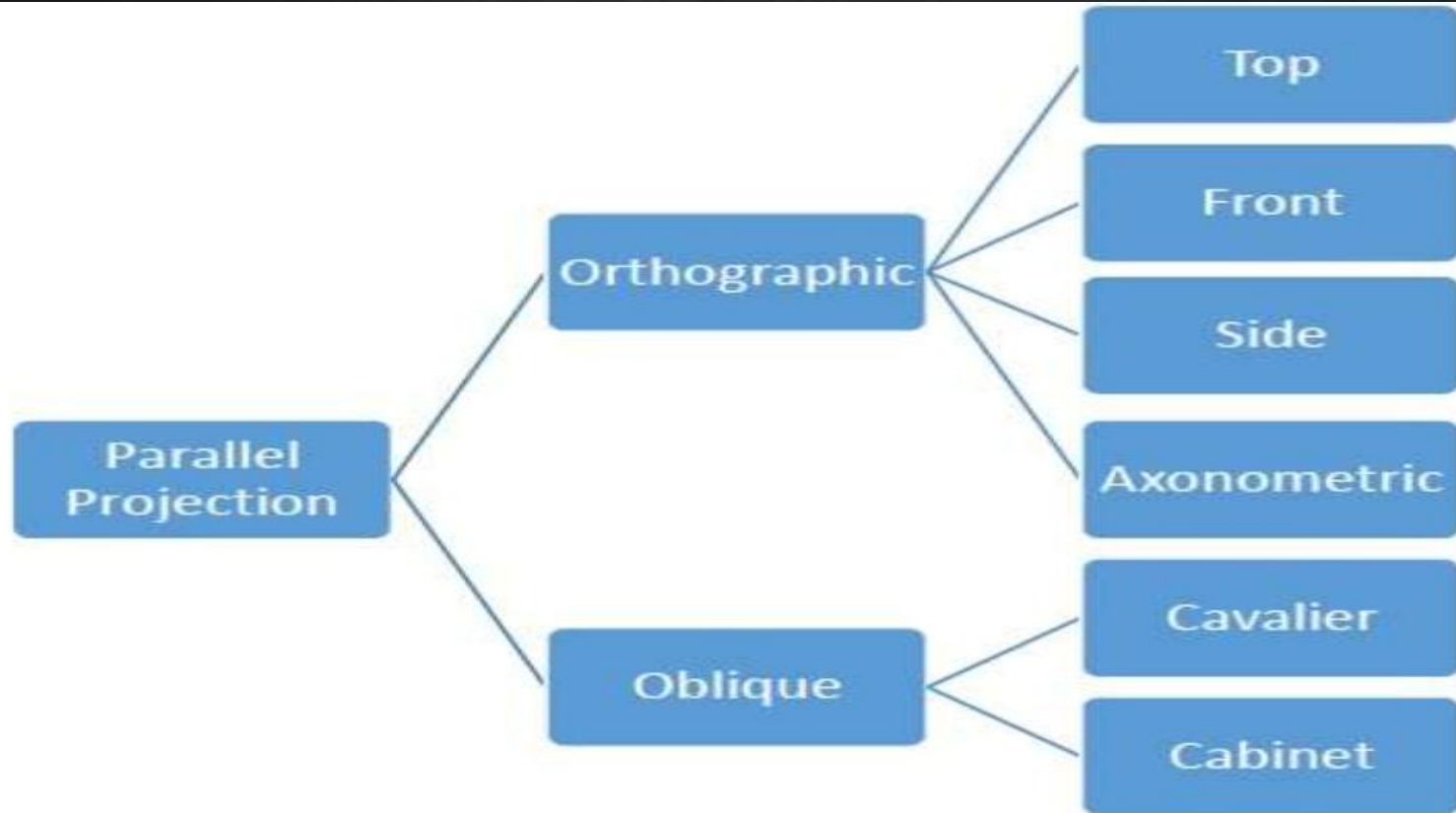
**Parallel Projection:** coordinate positions are transformed to the view plane along parallel lines .

**Perspective Projection:** coordinate positions are transferred to the view plane along lines that converge to a point called the projection reference point ( or center of projection ).

Projected view of an object is determined by calculating the intersection of the projection lines with the view plane.

**Parallel projection** preserves relative proportions of objects , and accurate views of the various sides of an object are obtained with a parallel projection ,but does not give a realistic representation of the 3D object.

**Perspective projection** produces a realistic view but does not preserve relative proportions.



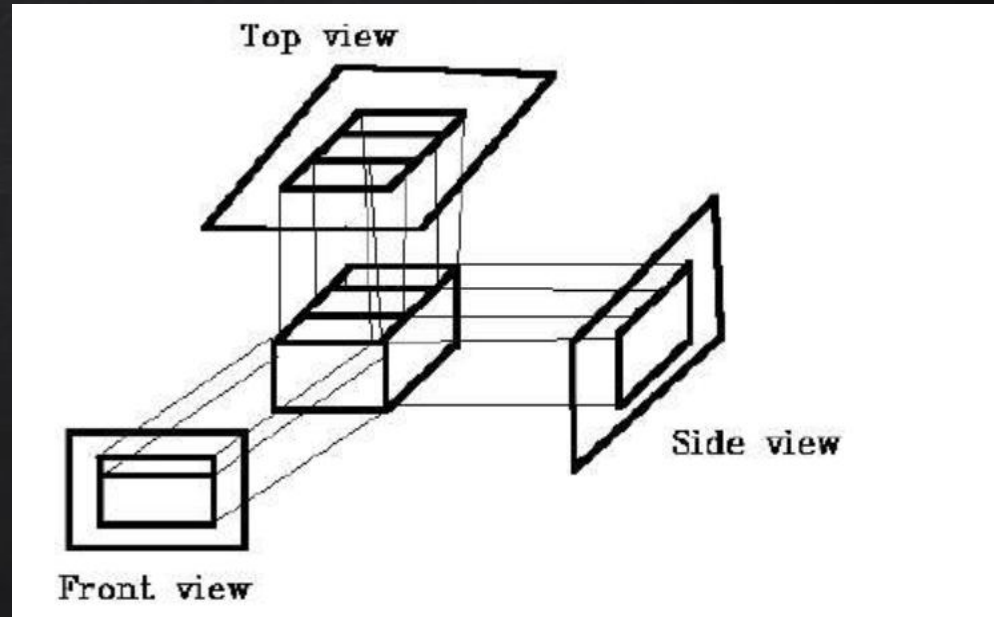
# Orthographic Projection

In orthographic projection the direction of projection is normal to the projection plane. There are three types of orthographic projections –

**Front Projection**

**Top Projection(plan view)**

**Side Projection**



Transformation equations for an orthographic parallel projections are straight forward .

If a view plane is placed at position  $zvp$  along  $z$  axis then any point  $(x,y,z)$  in viewing coordinates is transformed to projection coordinates as

$$xp=x , \quad yp=y$$



Find the orthographic projection of a unit cube onto the  $x=0$ ,  $y=0$  and  $z=0$  plane.

## X=0 plane projection /YZ plane projection

$A(0,0,0)$

$A'(0,0,0)$

$B(0,0,1)$

$B'(0,0,1)$

$C(1,0,1)$

$C'(0,0,1)$

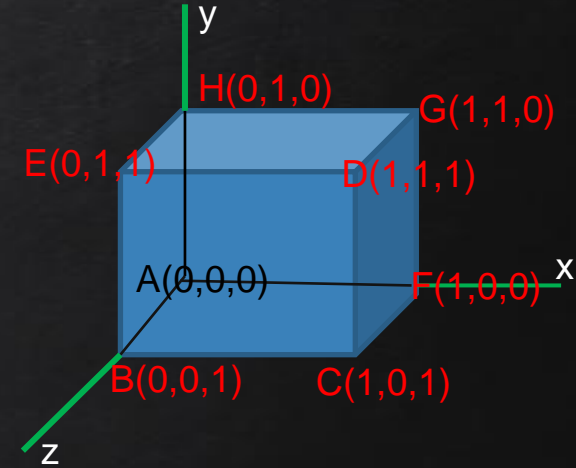
$D(1,1,1)$

$D'(0,1,1)$

$E(0,1,1)$    $E'(0,1,1)$

$F(1,0,0)$    $F'(0,0,0)$

$G(1,1,0)$



Find the orthographic projection of a unit cube onto the  $x=0$ ,  $y=0$  and  $z=0$  plane.

## Y=0 plane projection / XZ plane projection

$A(0,0,0)$

$A'(0,0,0)$

$B(0,0,1)$

$B'(0,0,1)$

$C(1,0,1)$

$C'(1,0,1)$

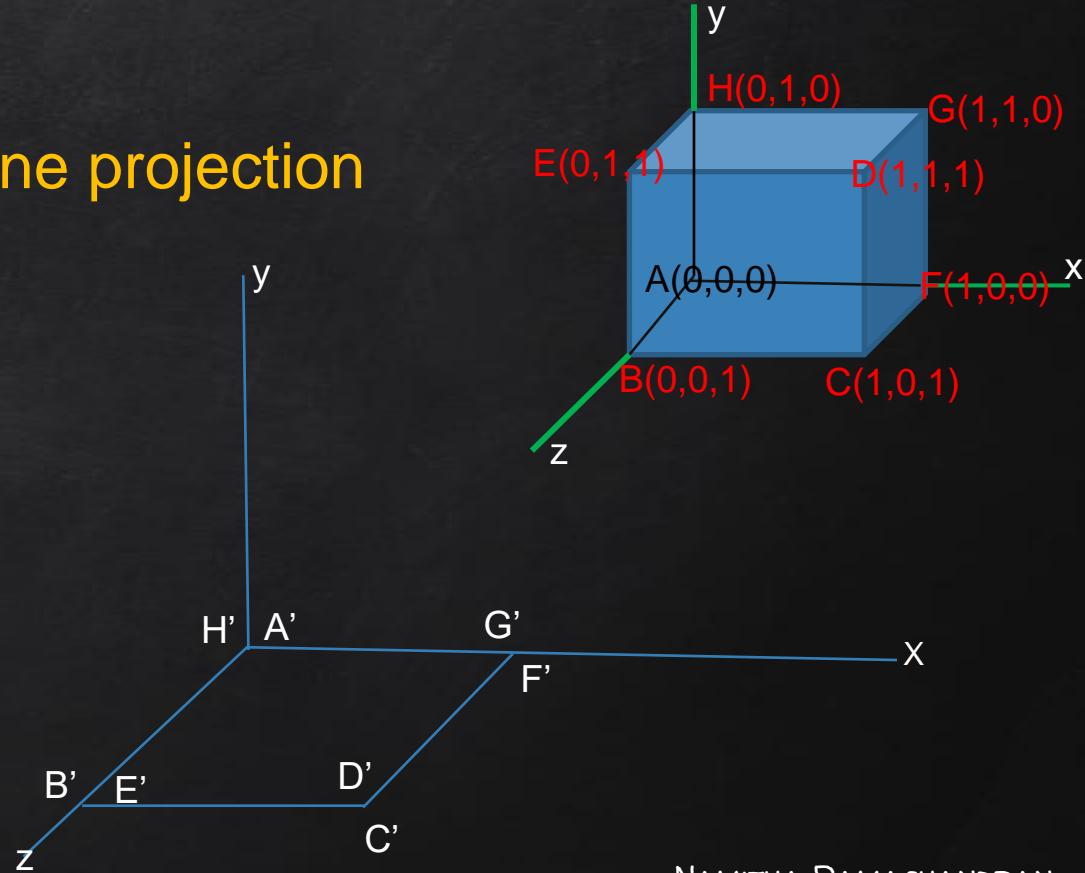
$D(1,1,1)$

$D'(1,0,1)$

$E(0,1,1)$    $E'(0,0,1)$

$F(1,0,0)$    $F'(1,0,0)$

$G(1,1,0)$



Find the orthographic projection of a unit cube onto the  $x=0$ ,  $y=0$  and  $z=0$  plane.

## Z=0 plane projection /XY plane projection

$A(0,0,0)$

$A'(0,0,0)$

$B(0,0,1)$

$B'(0,0,0)$

$C(1,0,1)$

$C'(1,0,0)$

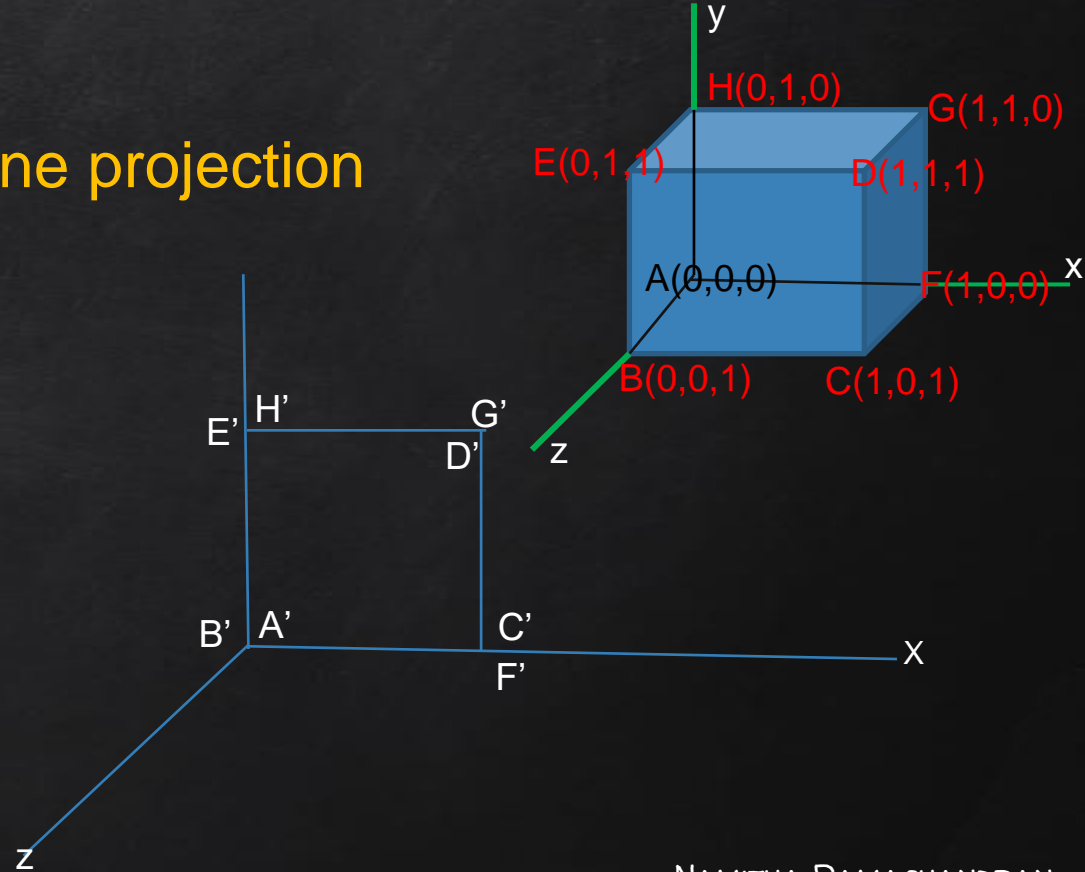
$D(1,1,1)$

$D'(1,1,0)$

$E(0,1,1)$    $E'(0,1,0)$

$F(1,0,0)$    $F'(1,0,0)$

$G(1,1,0)$

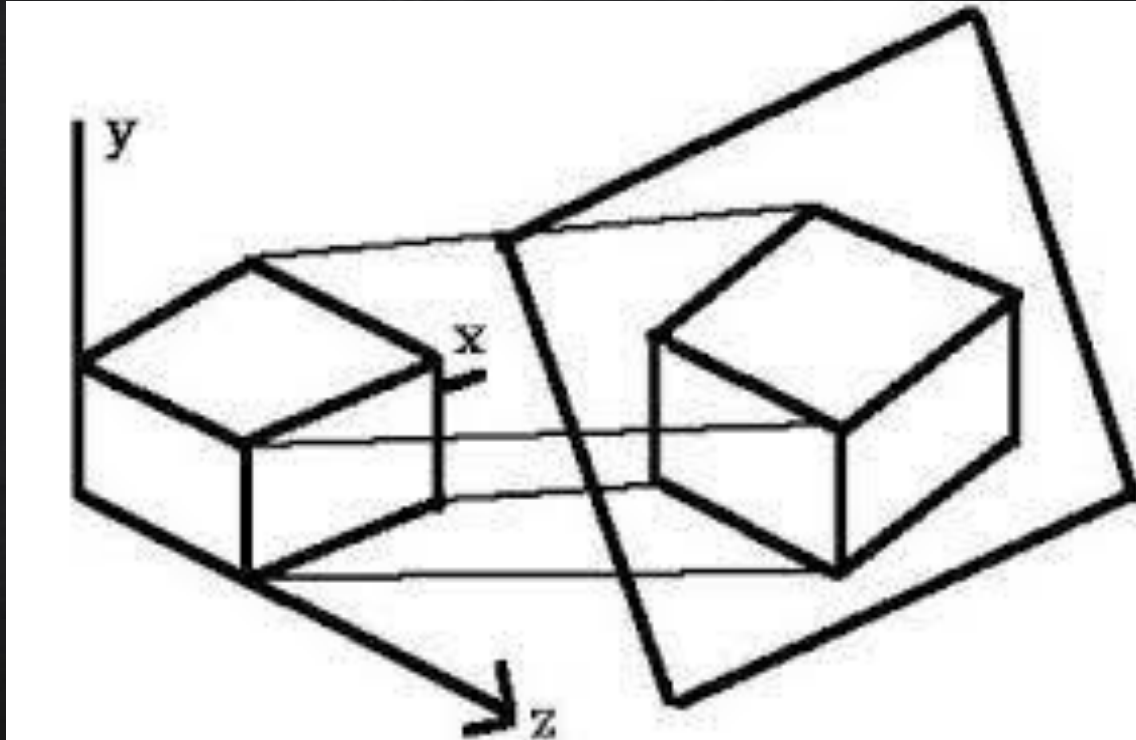


## Isometric Projections

Orthographic projections that show more than one side of an object are called **axonometric orthographic projections**.

The most common axonometric projection is an **isometric projection** where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance.

In this projection parallelism of lines are preserved but angles are not preserved.



## Oblique Projection

Projecting the plane along parallel lines that are not perpendicular to the projection plane.

There are two types of oblique projection, these are:

Cavalier Projection

Cabinet Projection

The cavalier projection preserves the lengths of lines that are perpendicular to the projection plane.

Projection line makes an angle 45 degree with projection plane .

$$x' = x + m$$

$$\cos \theta = m/L$$

$$m = L \cos \theta$$

$$x' = x + L \cos \theta$$

$$y' = y + n$$

$$\sin \theta = n/L$$

$$n = L \sin \theta$$

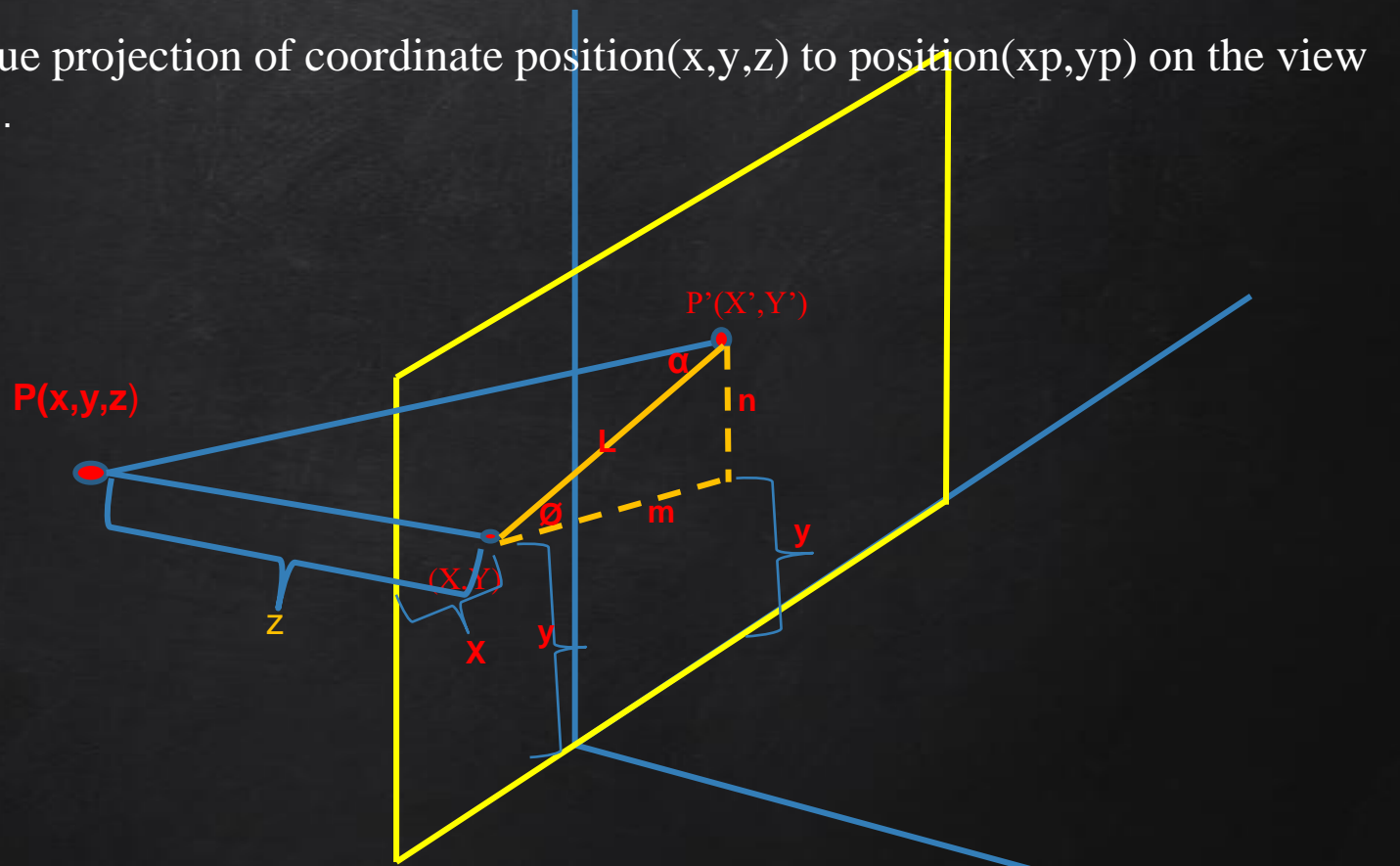
$$y' = y + L \sin \theta$$

$$\tan \alpha = z/L$$

$$L = z / \tan \alpha$$

$$L = z \cot \alpha$$

Oblique projection of coordinate position  $(x, y, z)$  to position  $(x_p, y_p)$  on the view plane .



$$x' = x + z \cot \alpha \cos \theta$$

$$y' = y + z \cot \alpha \sin \theta$$

$$x' = x + Z \cot \alpha \cos \theta$$

$$y' = y + Z \cot \alpha \sin \theta$$

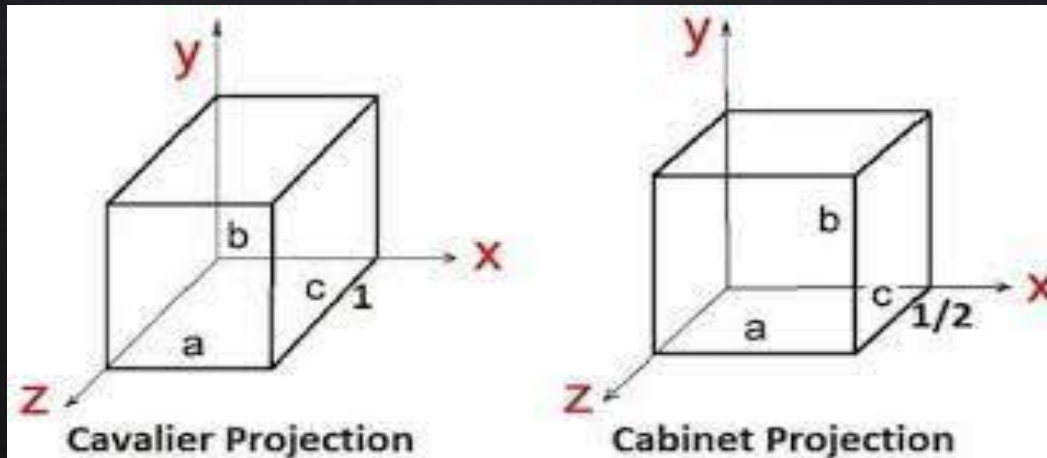
In Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cot \alpha \cos \theta & 0 \\ 0 & 1 & \cot \alpha \sin \theta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



**Cabinet projection: Line perpendicular to the viewing surface are projected at one –half of** Projection line makes an angle 63.4 degree with projection plane .

Cabinet projections appear more realistic than cavalier projections because of reduction in the length of perpendiculars .



# Perspective Projection

The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection or projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

**One point** perspective projection is simple to draw.

**Two point** perspective projection gives a better impression of depth.

**Three point** perspective projection is most difficult to draw.

Parallel lines that are parallel to the view plane will be projected as parallel lines.

Any set of parallel lines of objects that are not parallel to the projection plane are projected into converging lines .

The point at which set of projected parallel lines appears to converge is called **vanishing point** .

A different set of projected parallel lines will have a separate vanishing point.

Vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as the **principal vanishing point**.

▲ RWQ

▲ R'W'Q

▲ WSQ

▲ W'S'Q

$$\frac{RW}{R'W'} = \frac{WQ}{W'Q}$$

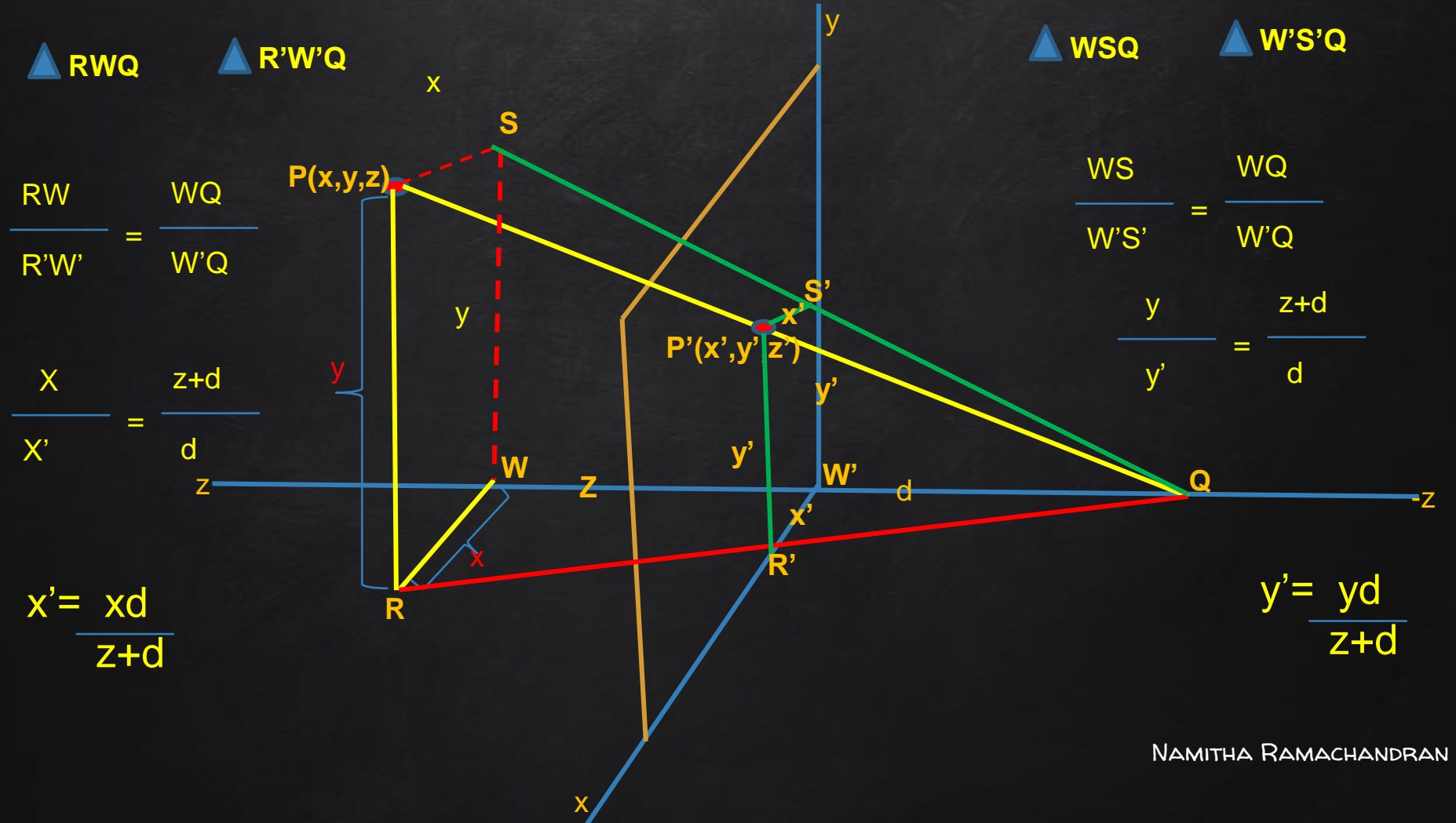
$$\frac{X}{X'} = \frac{z+d}{d}$$

$$x' = \frac{xd}{z+d}$$

$$\frac{WS}{W'S'} = \frac{WQ}{W'Q}$$

$$\frac{y}{y'} = \frac{z+d}{d}$$

$$y' = \frac{yd}{z+d}$$

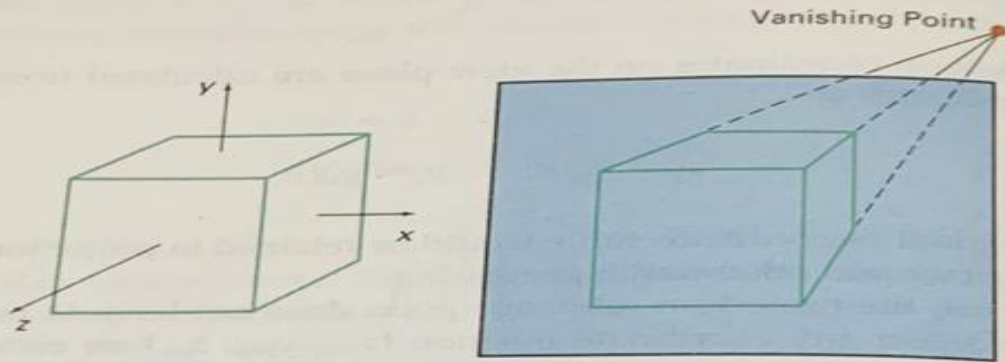


$$x' = x + Z \cot \alpha \cos \theta$$

$$y' = y + Z \cot \alpha \sin \theta$$

In Homogeneous coordinates

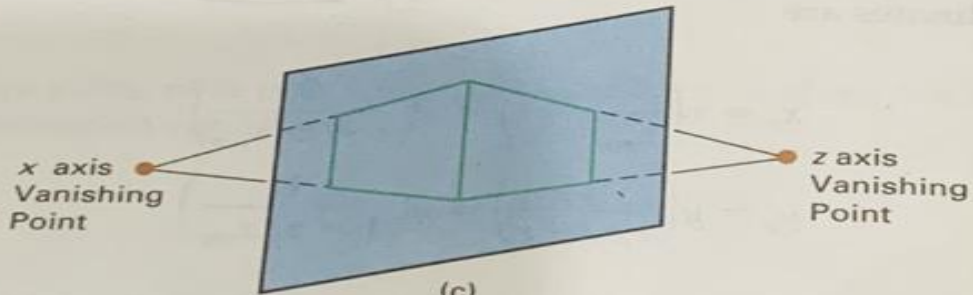
$$\begin{bmatrix} x' \\ y' \\ z' \\ h \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



(a)  
Coordinate  
Description

(b)  
One-Point  
Perspective  
Projection

One principal vanishing point  
then one point projection .



(c)  
Two-Point  
Perspective  
Projection

Two principal vanishing points  
then two point projection .

## *What is visible Surface detection?*

When a picture that contains the non-transparent objects and surfaces are viewed, the objects that are behind the objects that are closer cannot be viewed. To obtain a realistic screen image, these hidden surfaces need to be removed. This process of identification and removal of these surfaces is known as Hidden-surface removal or visible surface detection .

The hidden surface problems can be solved by two methods – **Object-Space method and Image-space method.**

The Object-space method is implemented in **physical coordinate system**

and image-space method is implemented in **screen coordinate system.**

Object space method compares objects and parts of objects to each other within the scene definition to determine which surfaces as a whole we should label visible .

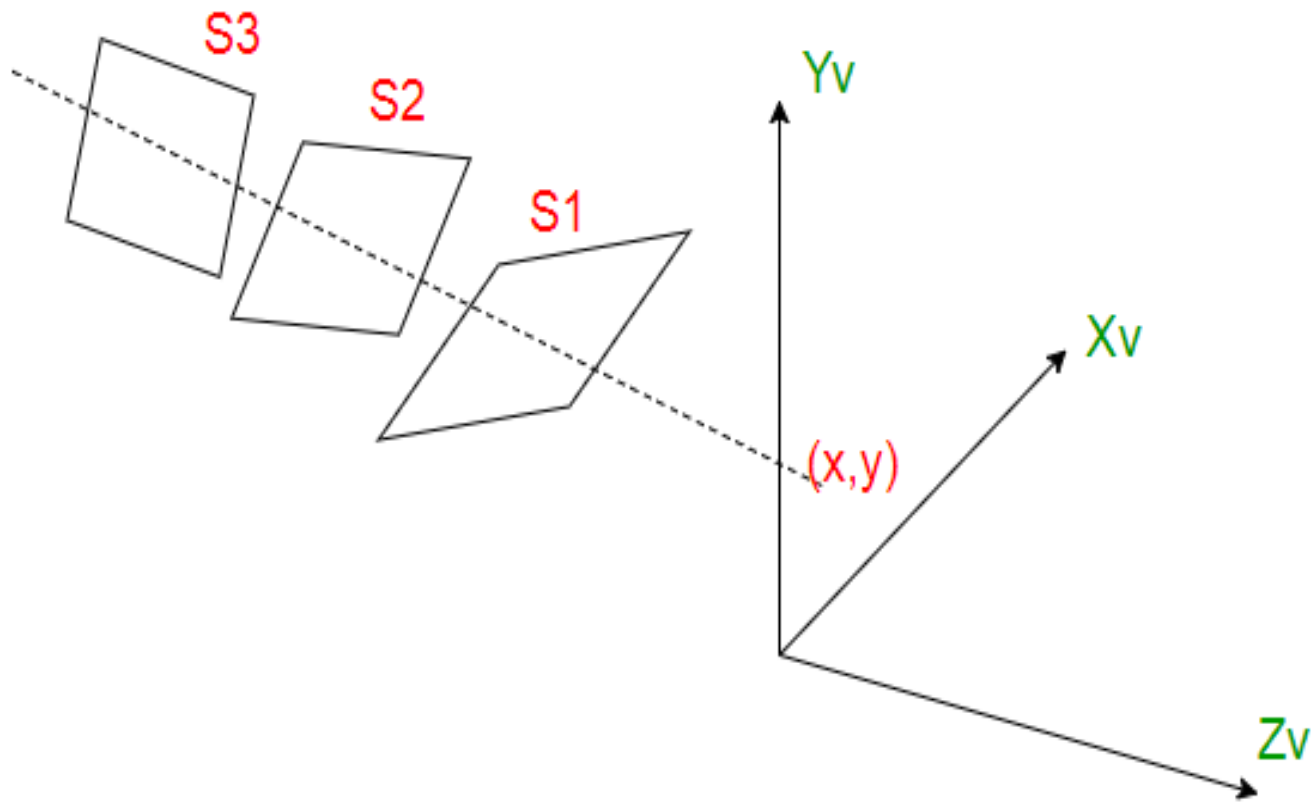
In image space method visibility is decided point by point at each pixel position on the projection plane .

Most visible surface detection algorithm us **image space methods** .



# Depth Buffer Algorithm.

It is also called a **Z Buffer Algorithm**. Depth buffer algorithm is simplest **image space algorithm**. For each pixel on the display screen, we keep a record of the depth of an object within the pixel that lies closest to the observer. In addition to depth, we also record the intensity that should be displayed to show the object.



## Algorithm

1. Initialise the depth buffer and refresh buffer so that for all buffer positions  $(x,y)$ ,  
$$\text{depth}(x,y)=0 \quad \text{refresh}(x,y)=I_{\text{background}}$$
2. For each position on each polygon surface , compare depth values to previously stored values in the depth buffer to determine visibility.
  - . Calculate the depth  $z$  for each  $(x,y)$  position on the polygon.
  - . If  $z > \text{depth}(x,y)$  , then set  
$$\text{depth}(x,y)=z , \quad \text{refresh}(x,y) = I_{\text{surf}}(x,y)$$

$I_{\text{background}}$  –value of background intensity

$I_{\text{surf}}(x,y)$  is the projected intensity value for the surface at pixel position  $(x,y)$ . After all surfaces have been processed , the depth buffer contains depth values for the visible surfaces and the refresh buffer contains

## Surface Equation

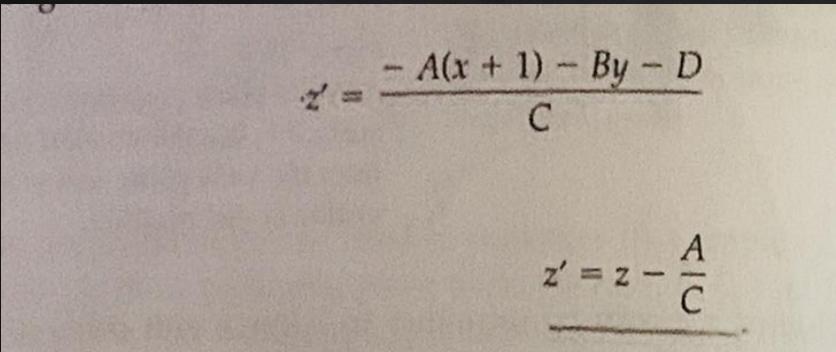
$$AX+BY+CZ+D=0$$

Depth calculation along z axis

$$Z=(-AX-BY-D)/C$$

Once obtained depth at a pixel position, in the same scan line ( horizontal line) next point depth can be obtained by substituting 'x ' as 'x+1'

Current point is(x,y) then next point in the same scan line is (x+1,y)


$$z' = \frac{-A(x+1) - By - D}{C}$$

$$z' = z - \frac{A}{C}$$

$$Z' = (-Ax - A - By - D)/C$$

$$Z' = (-Ax - By - D)/C - A/C$$

$$= Z - A/C$$

After finishing one scan line we don't want to start from the beginning of next scan line  
start from polygon edge point of of next line

Vertically next polygon point depth where is  $(x-1/m, y-1)$  , after substituting

### Depth calculation along z axis

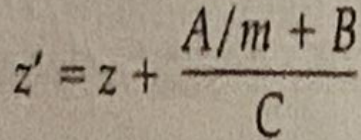
$$Z=(-AX-BY-D)/C$$

$$=-((A(x-1/m)-B(y-1)-D)/C$$

$$=(-Ax+A/m-By+B-D)/C$$

$$=(-Ax-By-D)/C+(A/m+B)/C$$

$$=z +(A/m +B)/C$$


$$z' = z + \frac{A/m + B}{C}$$

### Advantages:

- Simple to use
- Can be implemented easily in object or image space

### Disadvantages:

- Takes up a lot of memory
- It is a time-consuming process

## SCAN LINE ALGORITHM

It is an image-space method to identify visible surfaces.

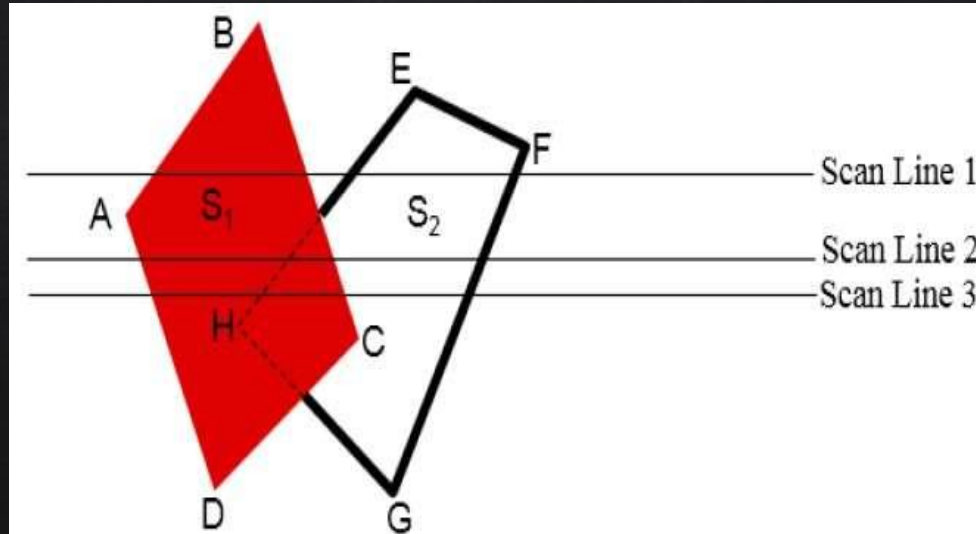
This is an extension of the scan line algorithm for the polygon filling algorithm. we can process only one surface at a time if we use the depth buffer method. But scan line method can process more than one surfaces at a time.

This method has depth information for the only single scan line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line.

Two important tables, **edge table** and **polygon table**, are maintained for this

**The Edge Table** – It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

- **The Polygon Table** – It contains the plane coefficients, Color information of the particular polygon, and a flag that is initialized to false.



## Scan Line Algorithm:

1. For all polygons (surfaces) intersecting a scan line , the faces are processed from left to right.
2. If there are overlapping faces, we should determine the depth so that nearest face to the view plane is identified and the intensities of that face is entered into the refresh buffer.
3. For each scan line an **active list of edges {edges intersecting by scan line}** of faces is maintained and they are sorted in the order of increasing x values.

In the figure, there are two polygon faces ABCD and EFGH.

**For scanline 1, Active Edge List { AB,BC,EH,FG }**

It is processed from left to right .

{ At left most position the face flag is turned On and on the rightmost position, it is set to OFF. }



Between edges, AB and BC flag for surface S1 is ON, and intensity of surface s1 is entered into refresh buffer.

Between BC and EH both flags of S1 and S2 are OFF and no intensity is stored.

Between EH and EG only flag for S2 is ON and the intensity of Surface S2 is entered into the refresh buffer.

**For scan line 1 there is no depth calculation since there is no overlapping region.**

**For scanline 2 ,Active Edge List { AD,EH,BC,FG}**

Between AD and BC flag for S1 is ON, and the intensity of S1 is entered into the refresh buffer.

Between EH and BC flag for S1 is ON and the flag for S2 is ON, in this case, calculate depth from the viewpoint to these two surfaces S1 and S2, and the intensity of the surface which is nearest to the viewpoint is entered into the refresh buffer .

**For scan line 3** we can follow the coherence property here, because scan line 2 and scan line 3 has same active edge list and other regularities also, in such case we don't want to calculate the depth separately for scan line 3 again, instead follow depth information calculated previously in scan line 2 and store the same intensity details as in case of scan line 2 .

### Advantages of scan line algorithm

Any number of overlapping surfaces can be processed at a time.  
Take advantage of coherence properties.

### Disadvantages

It does not work for surfaces that **cut through or cyclically overlap** each other.

