CST 304 Computer graphics & image processing

Module 3

Syllabus

(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, Scanline algorithm.

2D VIEWING

- 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.
- Windows and viewports are rectangles in standard position.

- Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process.

- In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.

- Sometimes the two-dimensional viewing transformation is simply referred to as the ***window-to-viewport transformation*** or the ***windowing transformation.***

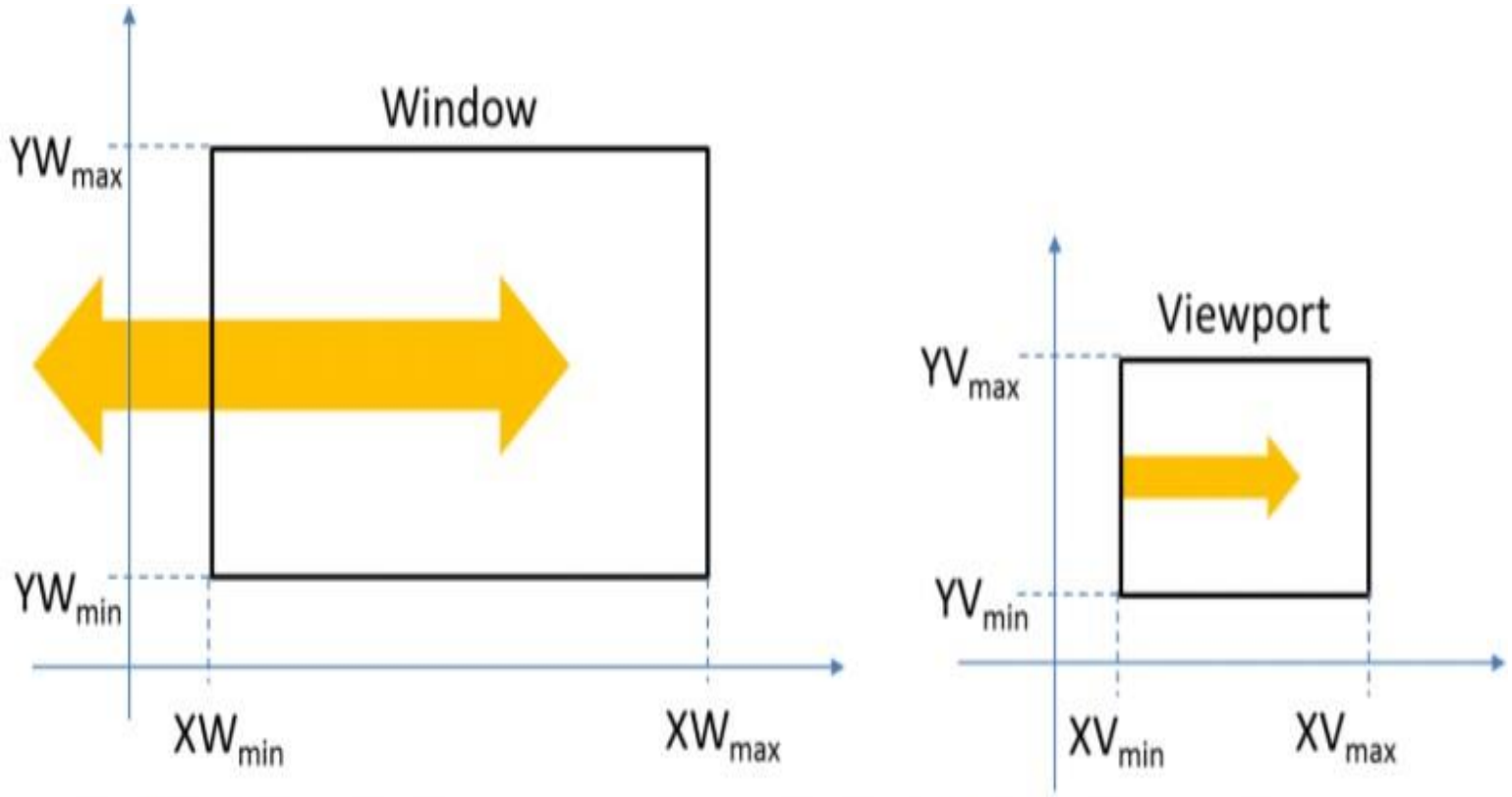


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

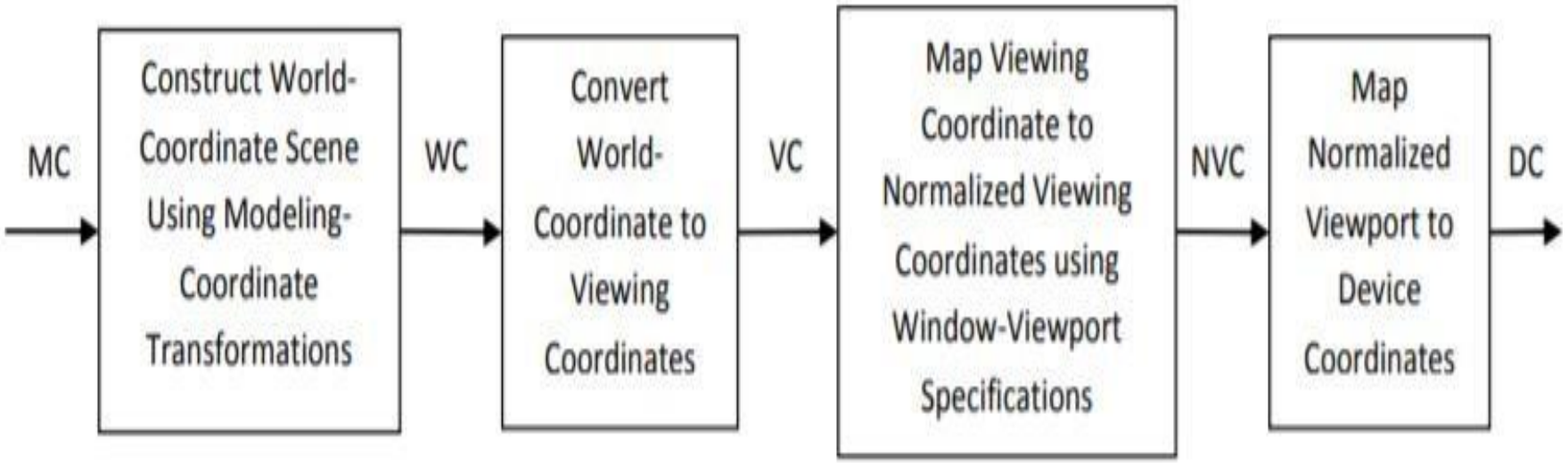


Fig. 3.2: - 2D viewing pipeline.

- Some graphics packages that provide window and viewport operations allow only standard rectangles.

- In this case, we carry out the **viewing transformation in several steps.**

- First, we construct the scene in world coordinates using the **output primitives and attributes.**

- Next, to obtain a particular orientation for the window, we can set up a two-dimensional viewing-coordinate system in the world-coordinate plane, and define a window in the viewing-coordinate system.

- We then define a viewport in normalized coordinates (in the range from **0** to 1) and map the viewing-coordinate description of the scene to normalized coordinates.

- At the final step, all parts of the picture that lies outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

- By changing the position of the viewport, we can view objects at different positions on the display area of an output device.

- Also, by varying the size of viewports, we can change the size and proportions of displayed objects.

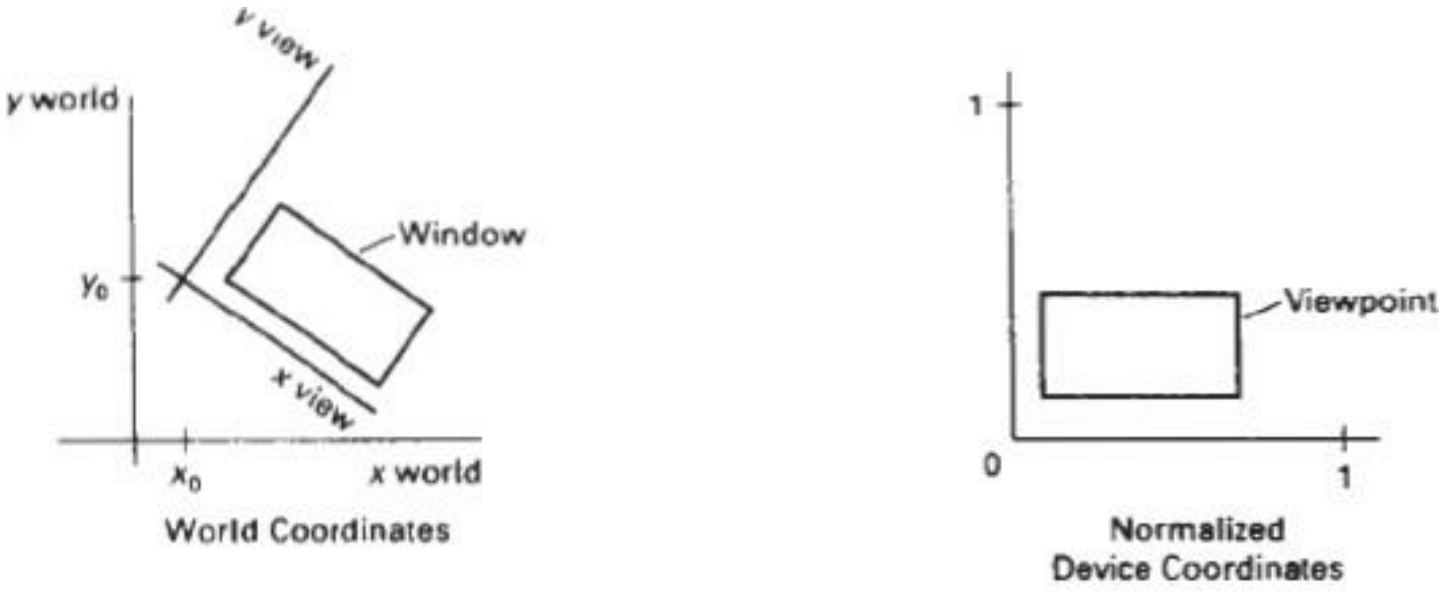
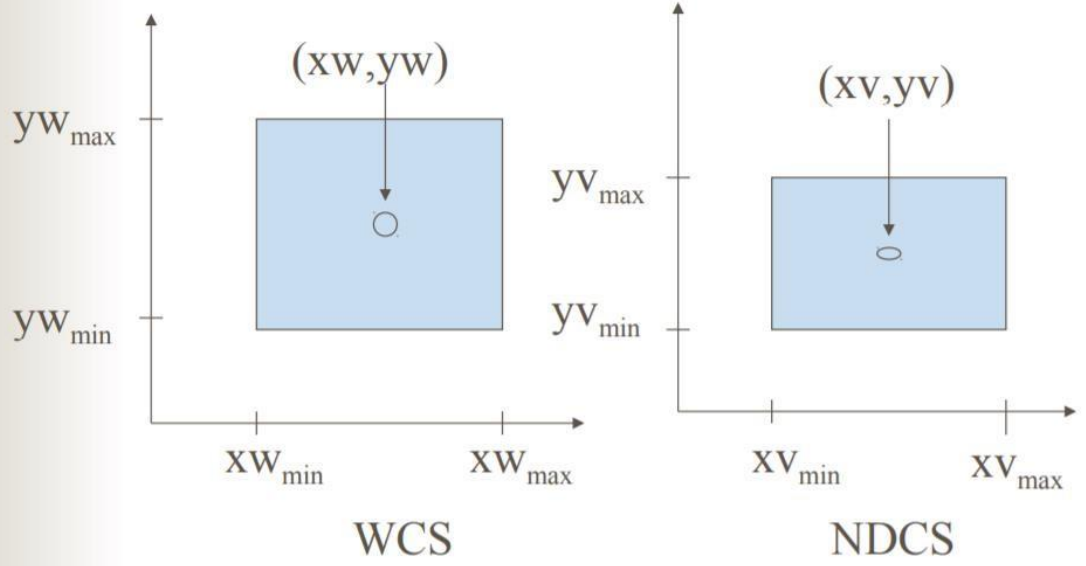


Figure 6-3
Setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport.

Window to viewport coordinate transformation

For eg.- if a coordinate position is at center of the viewing window, it will be displayed at the center of the viewport.



Problem

- for window, Xwmin = 20, Xwmax = 80, Ywmin = 40, Ywmax = 80.

- for viewport, Xvmin = 30, Xvmax = 60, Yvmin = 40, Yvmax = 60.

- Now a point (Xw, Yw) be (30, 80) on the window. We have to calculate that point on the viewport i.e (Xv, Yv).

$$S_x = (60 - 30) / (80 - 20) = 30 / 60$$
$$S_y = (60 - 40) / (80 - 40) = 20 / 40$$

•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 (Xw, Yw) = (30, 80) will be

(Xv, Yv) = (35, 60) on viewport.

Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called window to viewport transformation.
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.
- We find relative position by equation as follow:

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

- Solving by making viewport position as subject we obtain:

$$x_v = x_{vmin} + (x_w - x_{wmin})s_x$$

$$y_v = y_{vmin} + (y_w - y_{wmin})s_y$$

- Where scaling factor are :

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

- We can also map window to viewport with the set of transformation, which include following sequence of transformations:
  1. Perform a scaling transformation using a fixed-point position of (x_{Wmin}, y_{wmin}) that scales the window area to the size of the viewport.
  2. Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take $(s_x = s_y)$. in case if both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way one way is simply maintain relative position like other primitive and other is to maintain standard character size even though viewport size is enlarged or reduce.

- Number of display device can be used in application and for each we can use different window-to-viewport transformation. This mapping is called the **workstation transformation.**

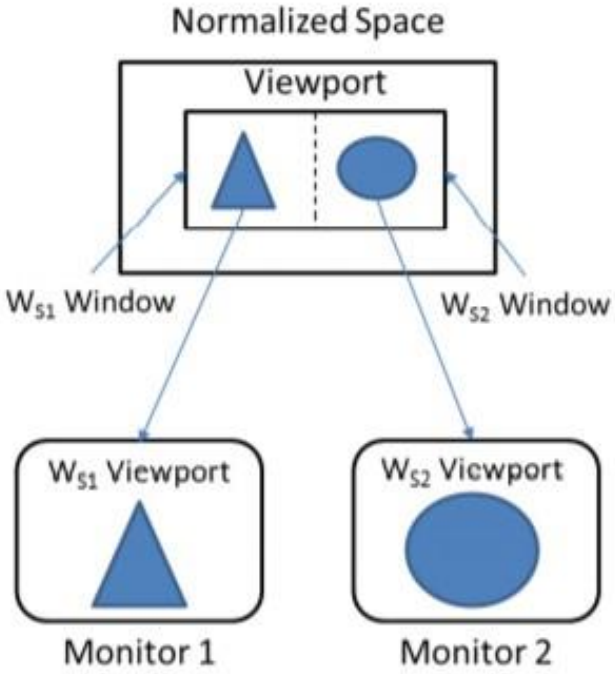


Fig. 3.4: - workstation transformation.

CLIPPING OPERATIONS

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm, or simply clipping.

- The region against which an object is to clipped is called a clip window.
- For the viewing transformation, we want to display only those picture parts that are within the window area .

- Everything outside the window is discarded.
- Clipping algorithm can be applied in world coordinates, so that only the contents of the window interior are mapped to device coordinates.

Different types of clipping algorithms

- **Point Clipping**
- **Line Clipping (straight-line segments)**

Cohen- Sutherland line clipping

Liang – Barsky line clipping

Nicholl-Leo-Nicholl line clipping

Midpoint subdivision algorithm

- **Area Clipping (polygons)**

Sutherland-Hodgeman polygon clipping

Weiler- Atherton polygon clipping

- **Curve Clipping**
- **Text Clipping**

Point Clipping

- Assuming that the clip window is a rectangle in standard position, we save a point P = **(*x*, y)** for display if the following inequalities are satisfied:

  - Here we consider clipping window is rectangular boundary with edge $(x_{wmin}, x_{wmax}, y_{wmin}, y_{wmax})$.
  - So for finding wether given point is inside or outside the clipping window we use following inequality:

    $x_{wmin} \leq x \leq x_{wamx}$

    $y_{wmin} \leq y \leq y_{wamx}$
  - If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

LINE CLIPPING

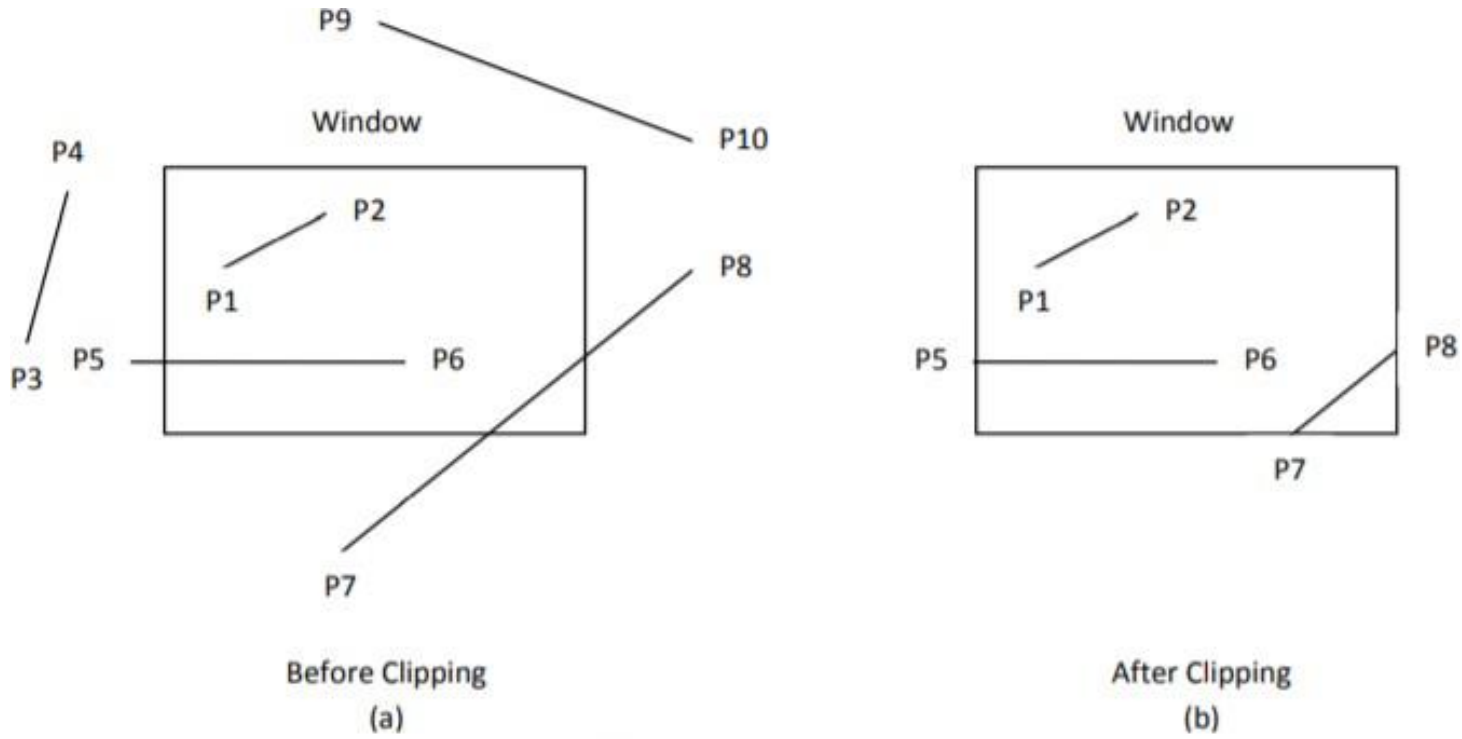


Fig. 3.5: - Line clipping against a rectangular window.

Procedure

- A line clipping procedure involves several parts.
- First, we can test a given line segment to determine whether it lies **completely inside** the clipping window.
- If it does not, we try to determine whether it lies **completely outside** the window.
- Finally, if we cannot identify a line as completely inside or completely outside, we must perform **intersection calculations** with one or more clipping boundaries.

- We process lines through the "inside-outside" tests by checking the line endpoints.
- A line with both endpoints inside all clipping boundaries, such as the line from P1 to P2 is saved.
- A line with both endpoints outside any one of the clip boundaries line P3 to P4 is discarded.
- All other lines cross one or more clipping boundaries, and may require calculation of multiple intersection points.

Cohen- Sutherland line clipping

- This is one of the oldest and most popular line clipping procedures.
- Generally, the method **speeds** up the processing of line segments by performing initial test that reduce the number of intersections that must he calculated.
- Every line end point 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 rectangle.

Region code

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom.
- By numbering the bit positions in the region code as **1** through 4 from right to left, the coordinate regions can be correlated with the bit positions as
- bit **1:** left
- bit 2: right
- bit 3: below
- bit 4: above

bit **1:** sign bit of x-xwmin
bit 2: sign bit of xwmax-x
bit 3: sign bit of y-ywmin
bit 4: sign bit of ywmax-y

- A value of 1 in any bit position indicates that the point is in that relative position; otherwise, the bit position is set to **0.**
- If a point is within the clipping rectangle, the region code is **0000.**
- A point that is below and to the left of the rectangle has a region code of **0101**

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 and we will keep this line

Step 3 : If step 2 fails, 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.a : Choose an endpoint of the line that is outside the given rectangle.

Step 3.2.b : Find the intersection point of the rectangular boundary (based on region code).

Step 3.2.c : Replace endpoint with the intersection point and update the region code.

Step 3.2.d : Repeat step 2 until we find a clipped line either trivially accepted or rejected.

Step 4 : Repeat step 1 for all lines

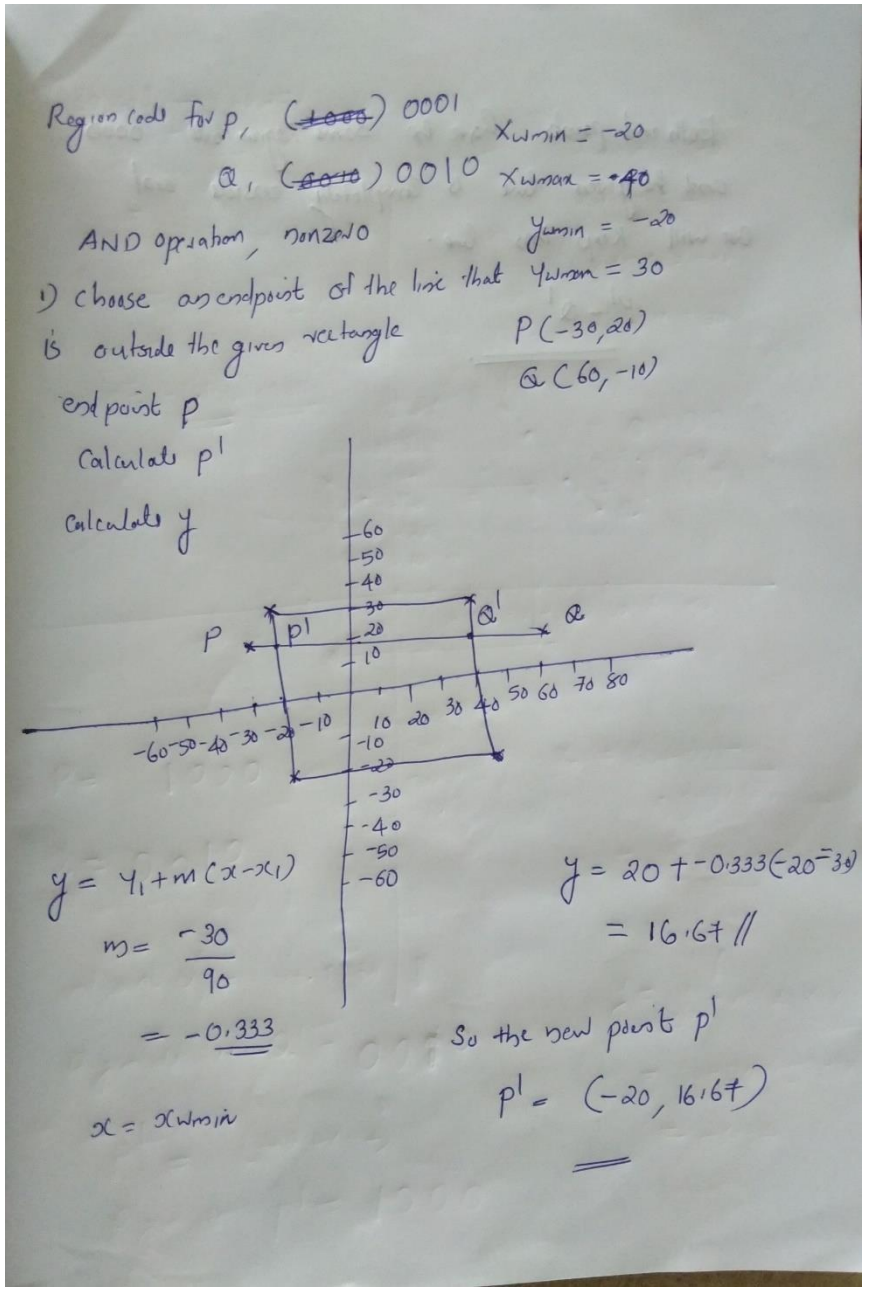
Intersection points calculation with clipping window boundary

- For intersection calculation we use line equation "$y = mx + b$".
- 'x' is constant for left and right boundary which is:
  - for left "$x = x_{wmin}$"
  - for right "$x = x_{wmax}$"
- So we calculate y coordinate of intersection for this boundary by putting values of x depending on boundary is left or right in below equation.

  $$y = y_1 + m(x - x_1)$$
- 'y' coordinate is constant for top and bottom boundary which is:
  - for top "$y = y_{wmax}$"
  - for bottom "$y = y_{wmin}$"
- So we calculate x coordinate of intersection for this boundary by putting values of y depending on boundary is top or bottom in below equation.

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

Given a clipping window A(-20,-20), B(40,-20), C(40,30) and D(-20,30). Using Cohen Sutherland line clipping algorithm, find the visible portion of the line segment joining the points P(-30,20) and Q(60,-10).


Region code for P, (1000) 0001
Q, (0010) 0010
AND operation, non zero
1) choose an endpoint of the line that is outside the given rectangle
endpoint P
Calculate P'
Calculate y
Xwmin = -20
Xwmax = 40
Ywmin = -20
Ywmax = 30
P(-30, 20)
Q(60, -10)
P
P'
Q'
Q
y = y1 + m(x - x1)
m = -30/90
= -0.333
x = xwmin
y = 20 + -0.333(-20 - -30)
= 16.67 //
So the new point P'
P' = (-20, 16.67)

Replace endpoint with intersection point and update region code.

$p'(-20, 16.67)$ $Q = (60, -10)$

find the region code

$p' = (0000)$ $Q = (0010)$

~~Step~~. perform AND operation, nonzero, line is partially visible choose an endpoint of the line that is outside the given rectangle.

find the intersection Q',

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

$= -10 + -0.333(40 - 60)$ $= 40$

$= 3.34$ //

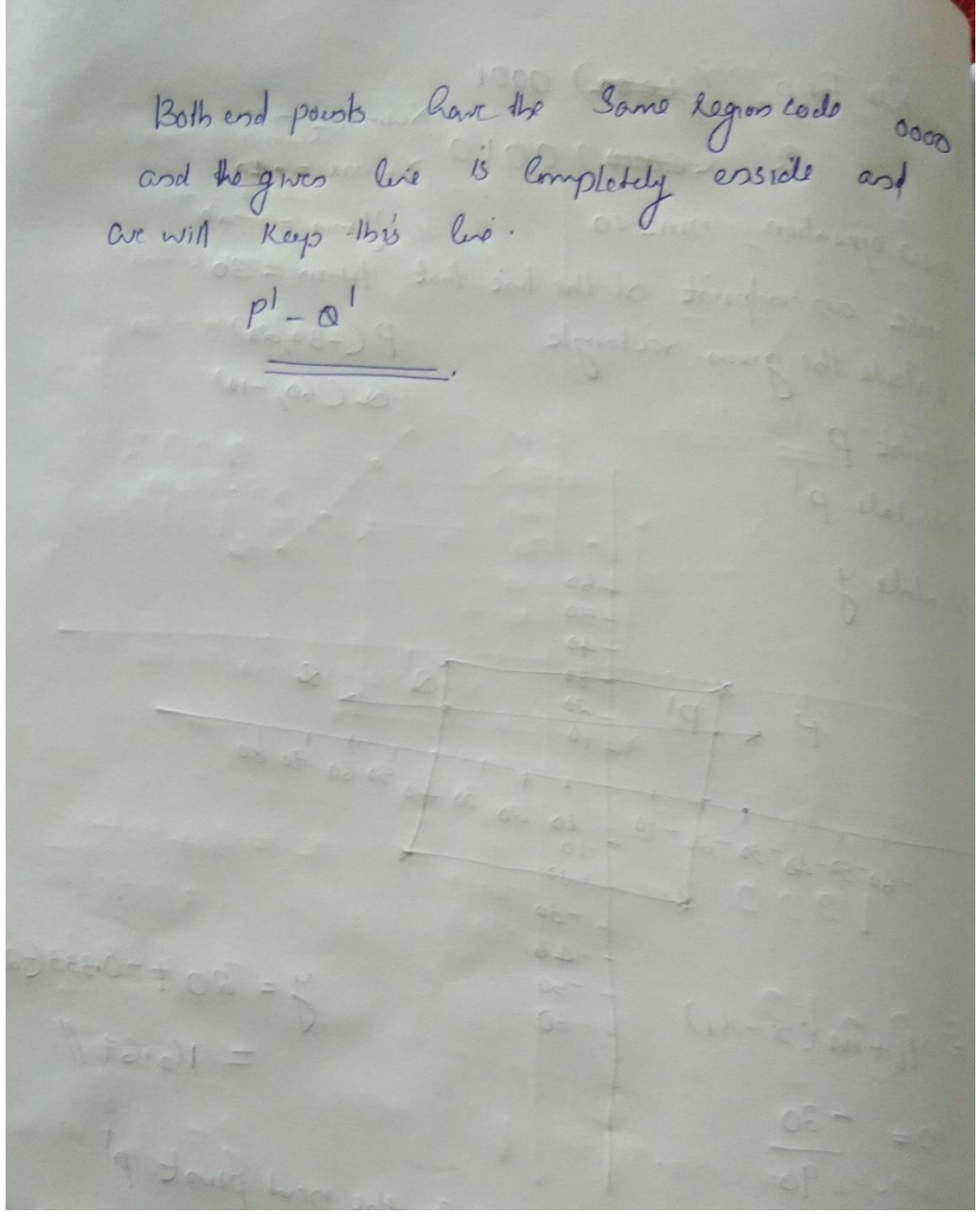
$Q' = (40, 3.34)$

Replace endpoint with intersection point Q'

$p' = (-20, 16.67)$ $Q' = (40, 3.34)$

find the region code

$p' = 0000$ $Q' = 0000$


Both end points have the same region code 0000
and the given line is completely visible and
we will keep this line.
P' - Q'

Clip a line p(-1,5) and q(3,8) using Cohen-Sutherland line clipping algorithm with window coordinates (-3,1) and(2,6)

- **Solve ……**

Polygon clipping

- Polygon clipping is a process in which we only consider the part which is inside the view pane or window.
- We will remove or clip the part that is outside the window.

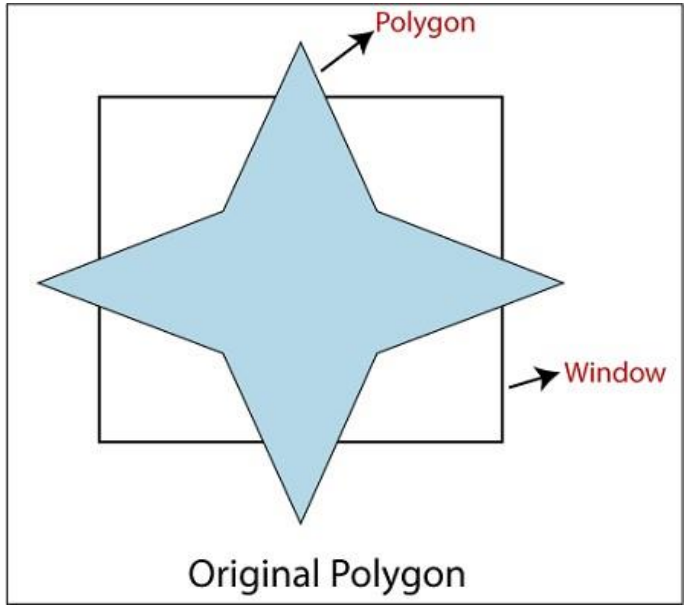
We will use the following algorithms for polygon clipping

- Sutherland-Hodgeman polygon clipping algorithm
- Weiler-Atherton polygon clipping algorithm

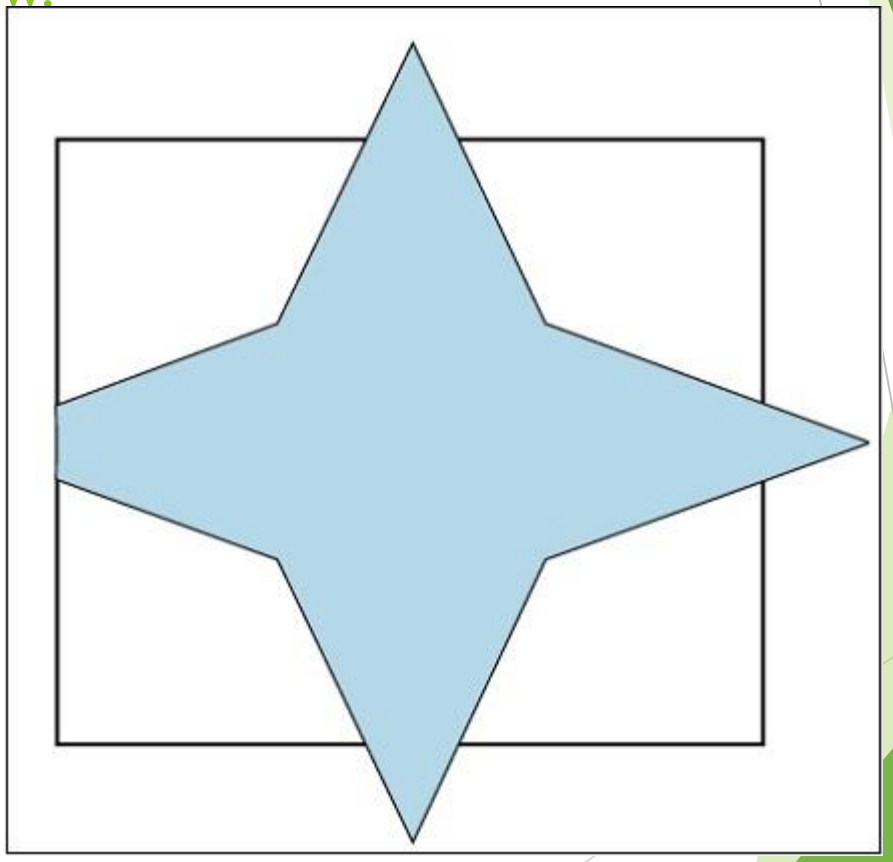
Sutherland-Hodgeman polygon clipping algorithm

- The polygon clipping algorithm deals with four different clipping cases.

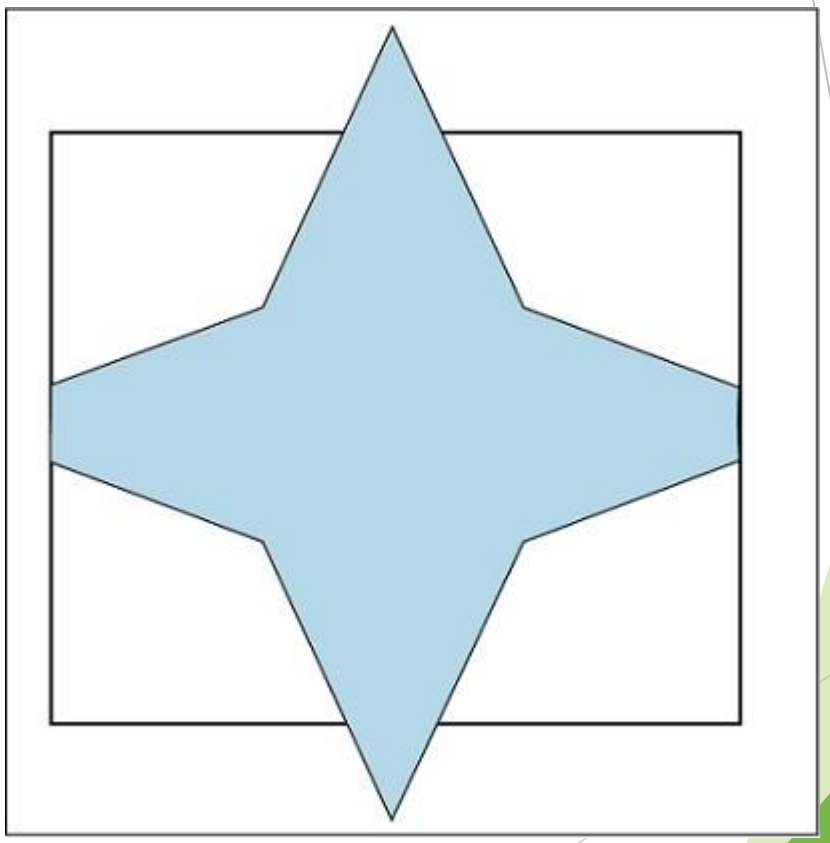
- The output of each case is input for the next case.



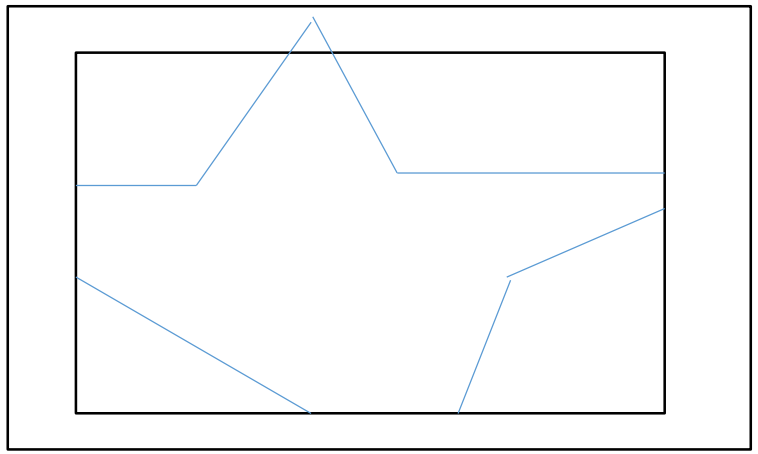
Case1) Left clip: In the left side polygon clipping, we only remove the left part of the polygon, which is outside the window. We only save the portion which is inside the window.



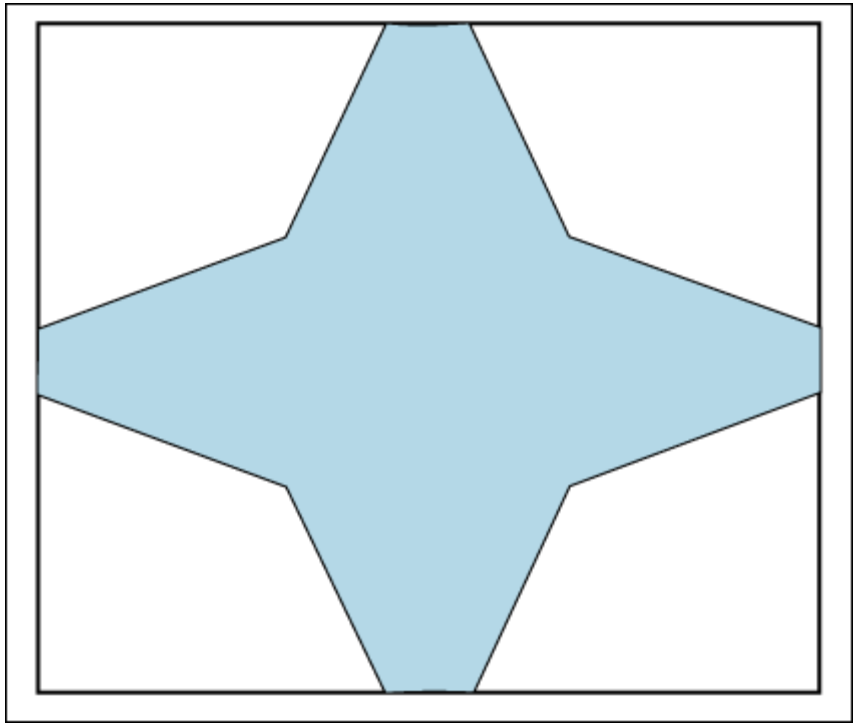
Case2) Right clip: In the right-side polygon clipping, we only remove the right part of the polygon, which is outside the window. We only save the portion which is inside the window.



Case3) Bottom clip: In the bottom side polygon clipping, we only remove the bottom part of the polygon, which is outside the window. We only save the portion which is inside the window.

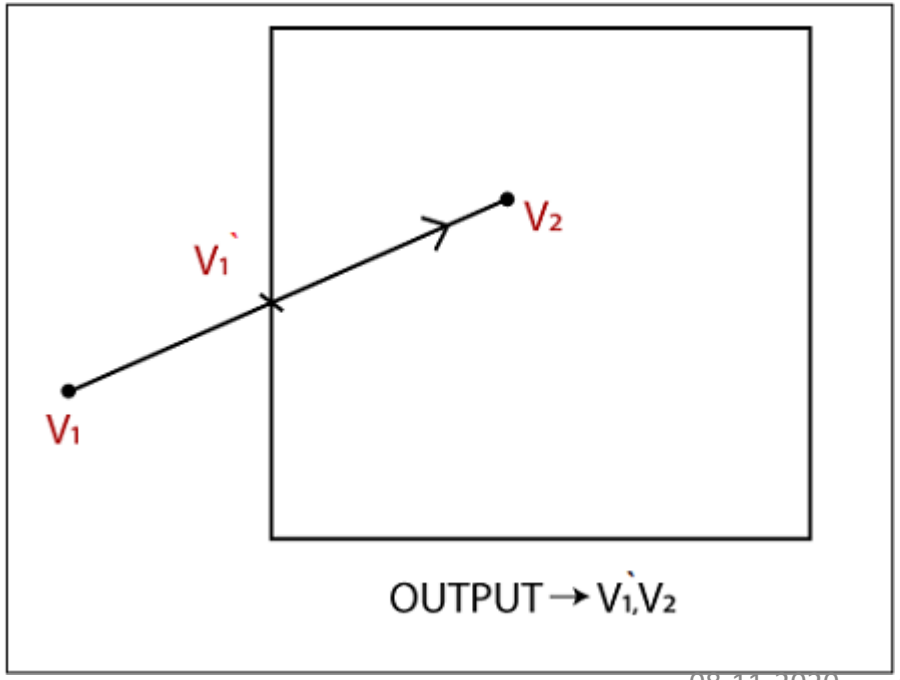


Case4) Top clip: On the top side polygon clipping, we only remove the top part of the polygon, which is outside the window. We only save the portion which is inside the window.

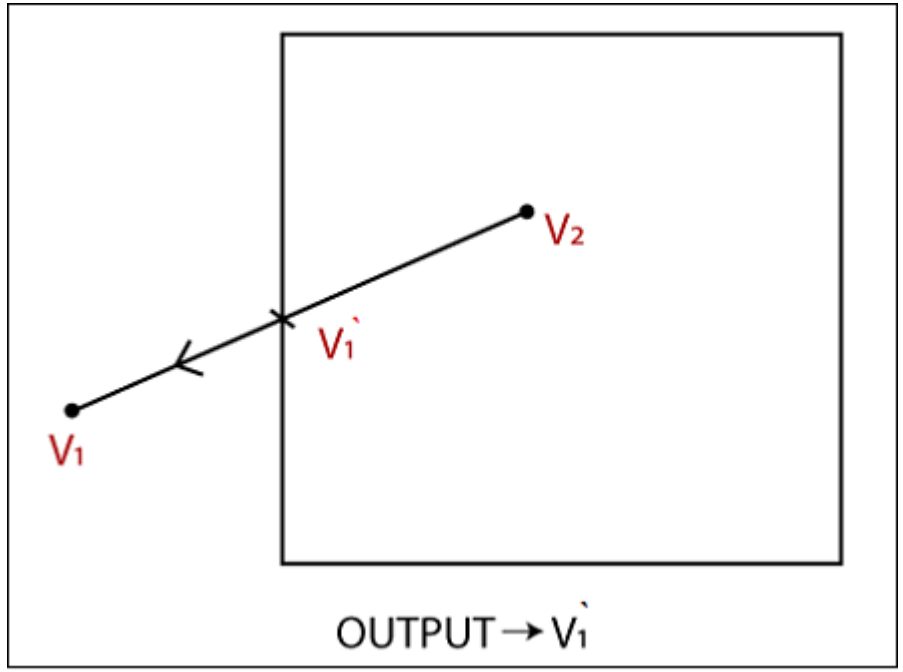


There should be the following conditions while we clip a polygon.

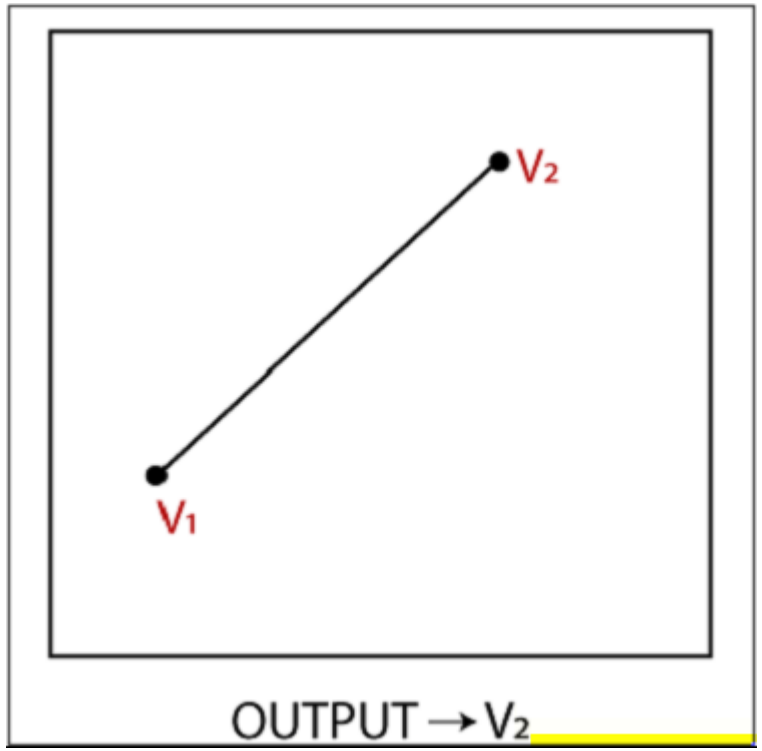
Condition 1(Out-In): If the first vertex of the polygon is **outside** and the second vertex is **inside** the window, then the output will be the **intersection point and second vertex.**



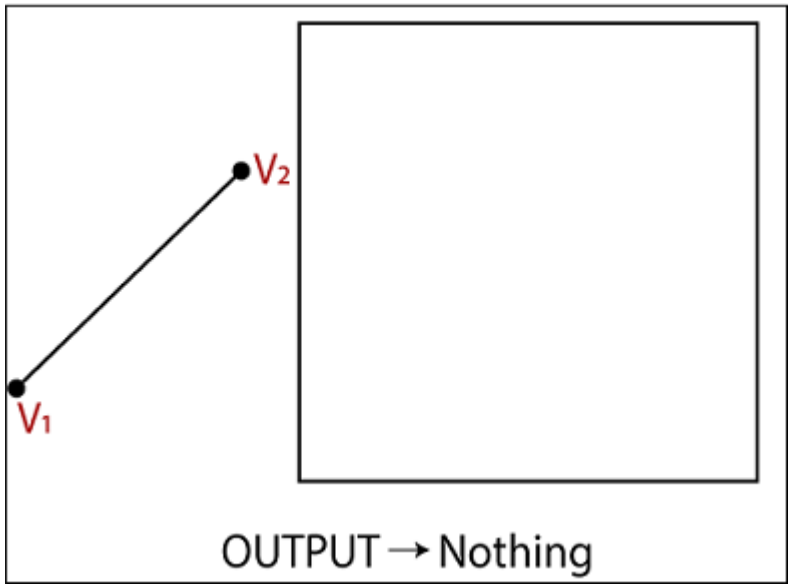
Condition 2(In-Out): If the first vertex of the polygon is inside and the second vertex is outside the window, then the output will be the intersection point.



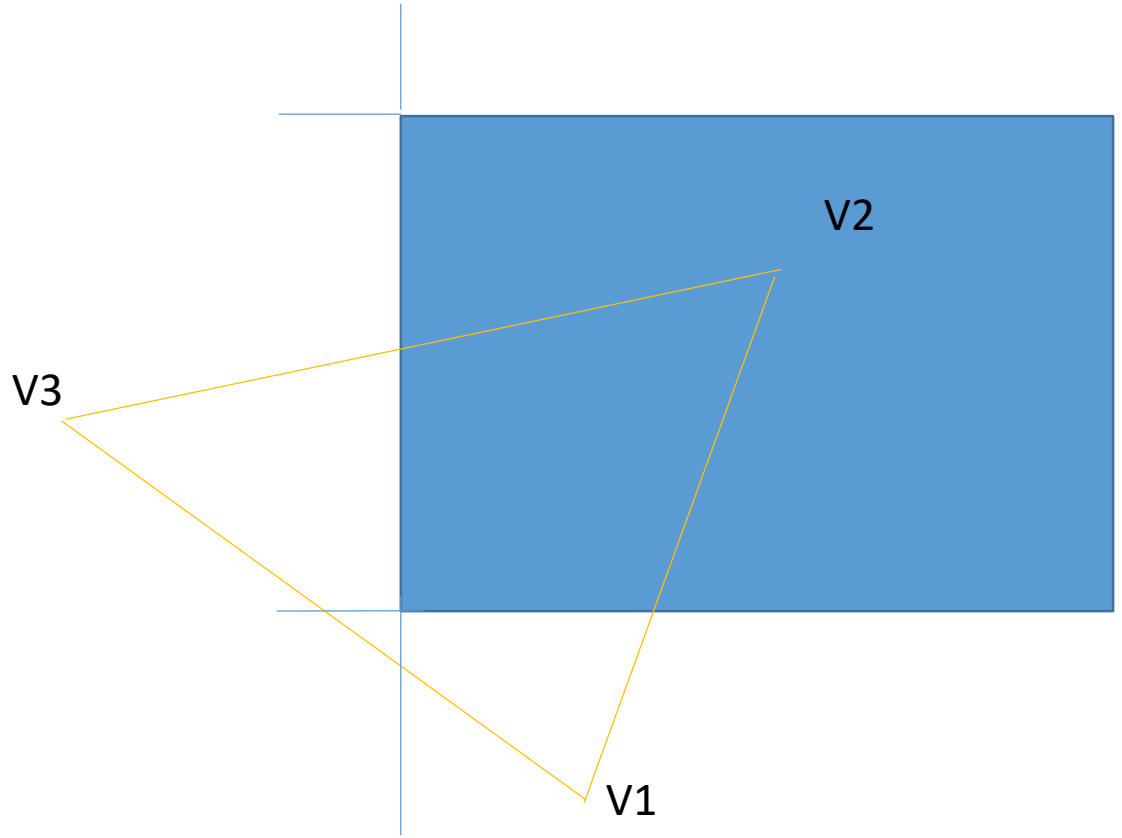
Condition 3(In-In): If both vertices of the polygon are inside the window, then the output will be the second vertex.



Condition 4(Out-Out): If both vertices of the polygon are outside the window, then the output will be nothing. It means we found a clipped polygon.

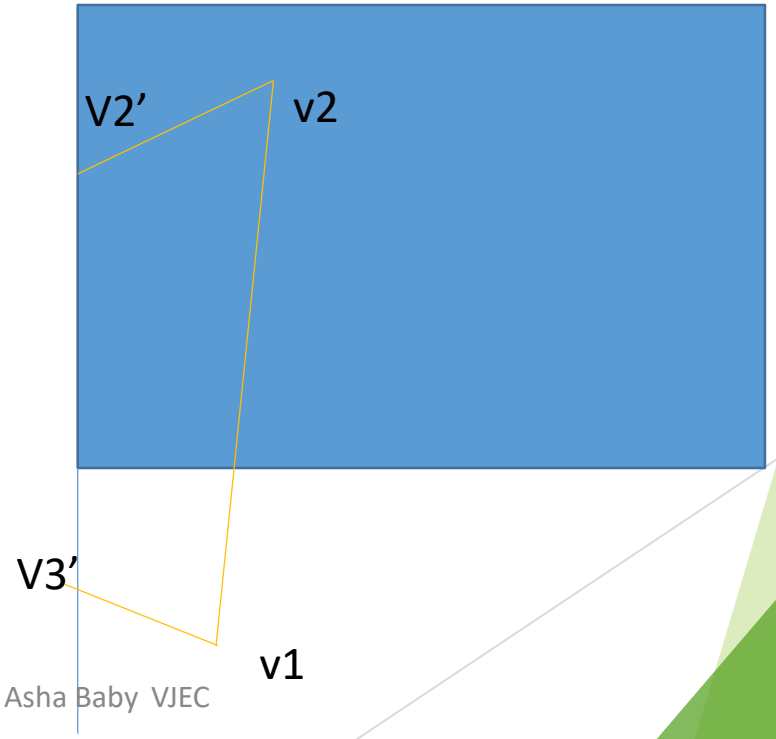


Clip the given polygon using Sutherland-Hodgeman algorithm



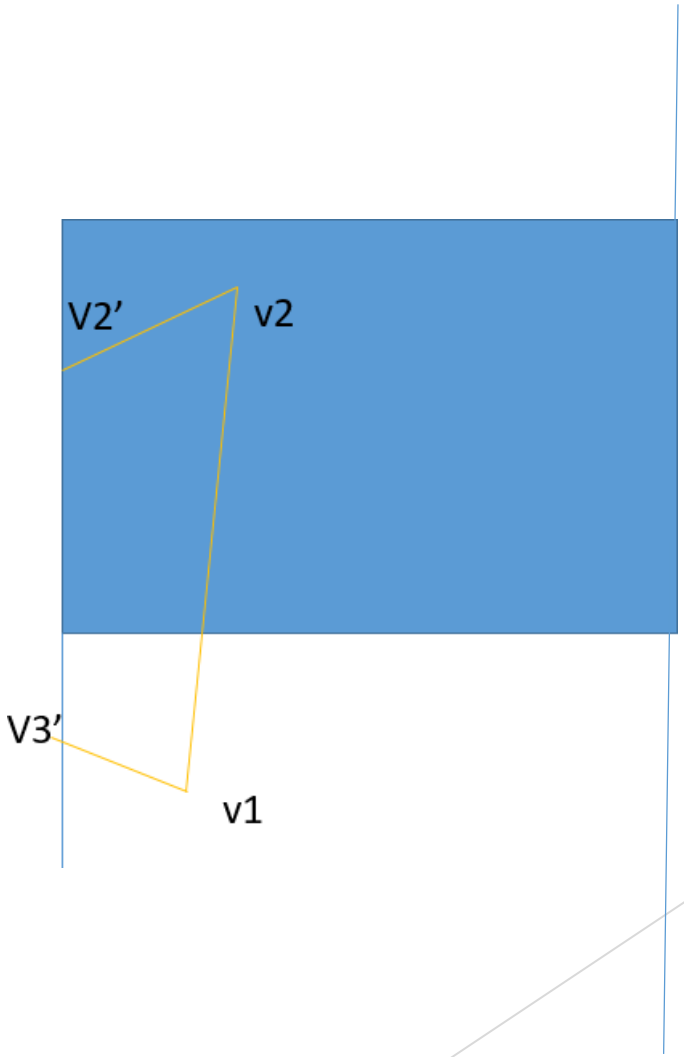
Left clip

- v1v2 = in-in {v2}
- v2v3 = in-out {v2’}
- v3v1 =out-in {v3’,v1}



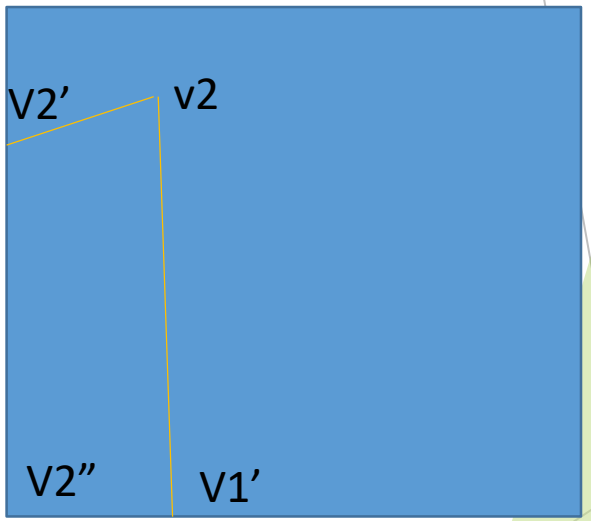
Right clip

- v1v2 =in-in {v2}
- v2v2’=in-in {v2’}
- v2’v3’=in-in {v3’}
- v3’v1=in-in {v1}



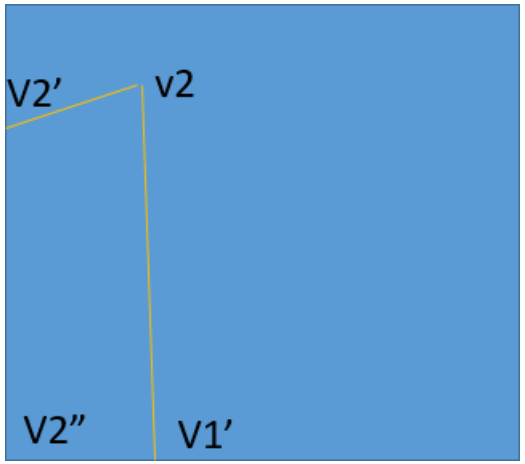
Bottom clip

- v1v2 =out-in {v1’,v2}
- v2v2’=in-in {v2’}
- v2’v3’=in-out {v2”}
- v3’v1=out-out{}

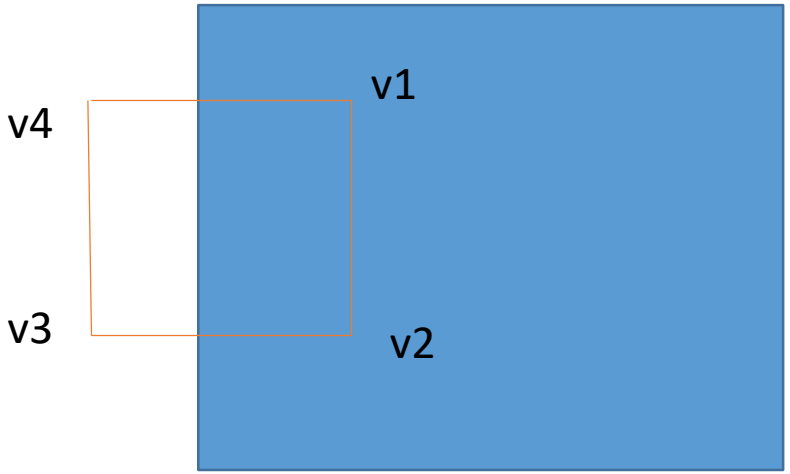


Top clip

- v1’v2 =in-in {v2}
- v2v2’=in-in {v2’}
- v2’v2”=in-in {v2”}
- v2”v1’=in-in{v1’}



Clip the given polygon using Sutherland-Hodgeman algorithm



Disadvanatges

1. It clips to each window boundary one at a time.

2. It has a “Random” edge choice

3. It has Redundant edge-line cross calculations

Three dimensional viewing pipeline

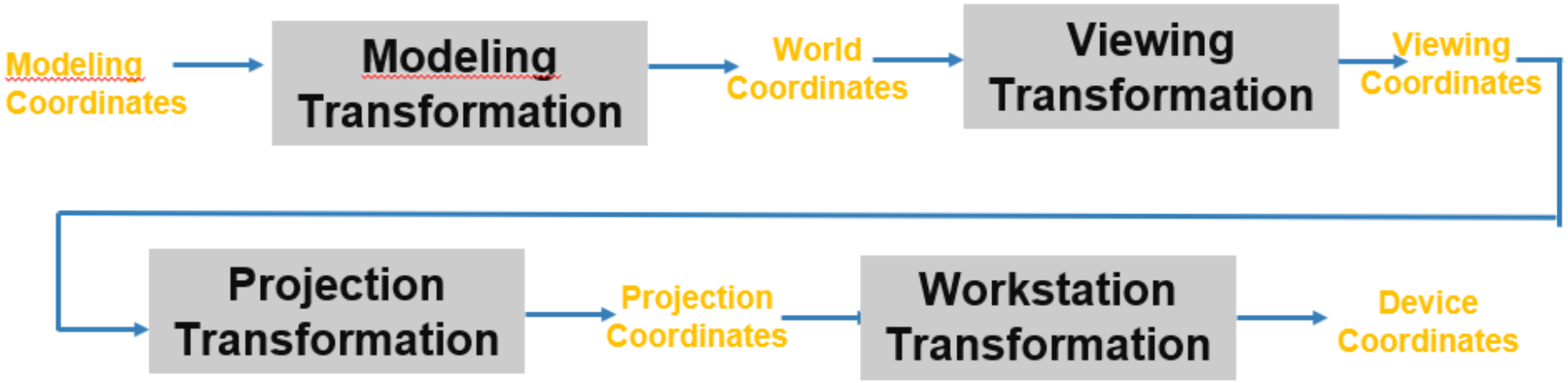
- The viewing-pipeline in 3 dimensions is almost the same as the 2D-viewing-pipeline.

- Only after the definition of the viewing direction and orientation (i.e., of the camera) an additional projection step is done, which is the reduction of 3D-data onto a projection plane

- The steps for computer generation of a view of 3D scene are analogous to the process of taking photograph by a camera.

- For a snapshot, we need to position the camera at a particular point in space and then need to decide camera orientation.

- Finally when we snap the shutter, the seen is cropped to the size of window of the camera and the light from the visible surfaces is projected into the camera film.


Modeling Coordinates
Modeling Transformation
World Coordinates
Viewing Transformation
Viewing Coordinates
Projection Transformation
Projection Coordinates
Workstation Transformation
Device Coordinates

Steps

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

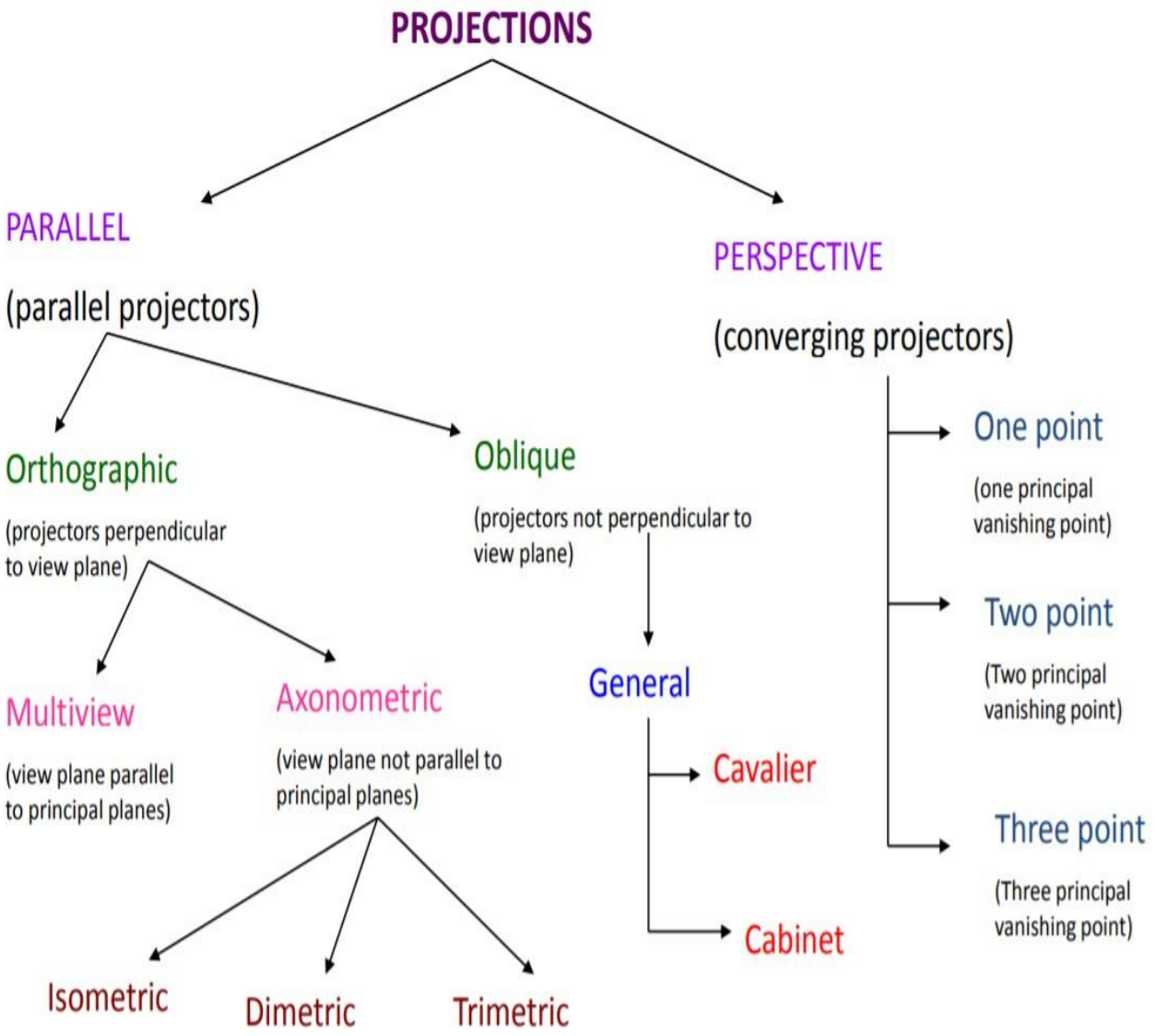
Projections

- 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.
- It is also defined as mapping or transformation of the object in projection plane or view plane.
- The view plane is displayed surface.
- We can project the 3-D objects onto the 2-D plane.

- So Projection can be defined as a mapping of point P onto its image P’ in the projection plane or view plane.

There are two basic projection methods:

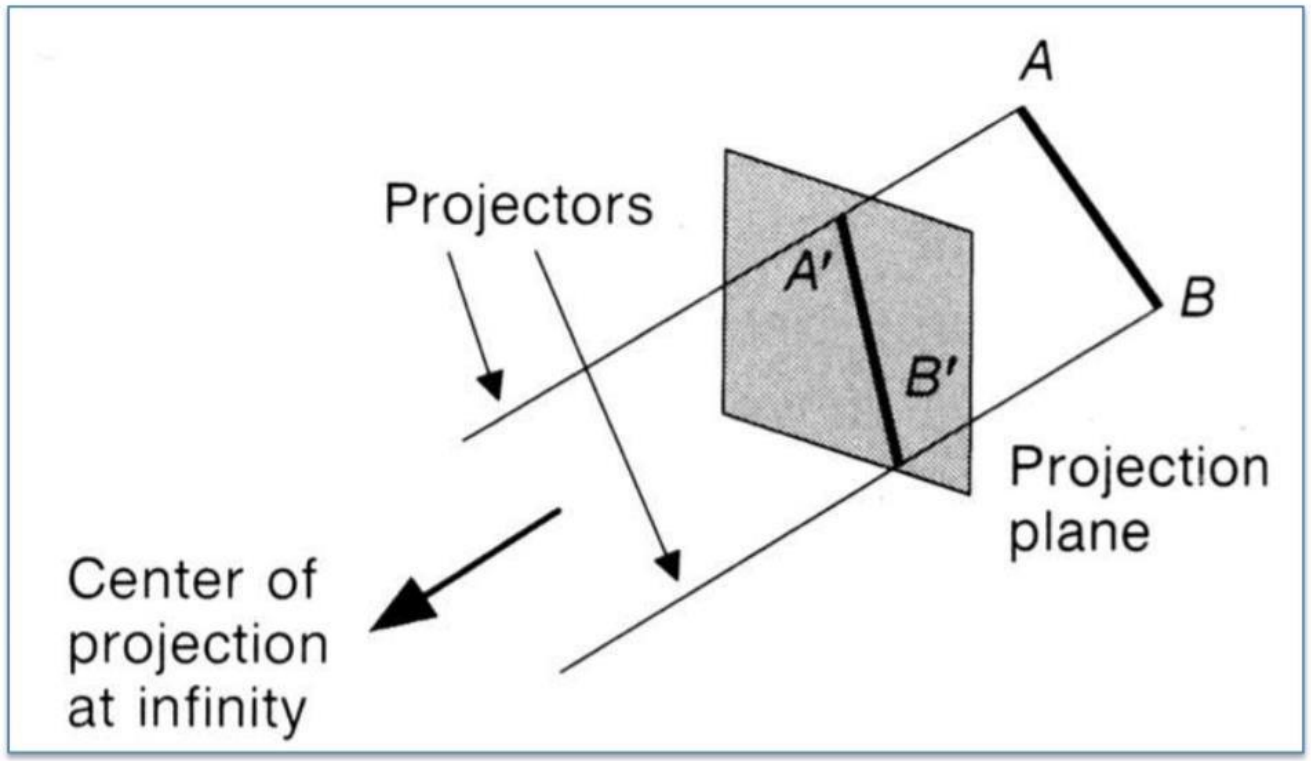
- Parallel projection
- Perspective projection


PROJECTIONS
PARALLEL
(parallel projectors)
PERSPECTIVE
(converging projectors)
Orthographic
(projectors perpendicular to view plane)
Oblique
(projectors not perpendicular to view plane)
Multiview
(view plane parallel to principal planes)
Axonometric
(view plane not parallel to principal planes)
General
Cavalier
Cabinet
Isometric
Dimetric
Trimetric
One point
(one principal vanishing point)
Two point
(Two principal vanishing point)
Three point
(Three principal vanishing point)

SL.NO	PARALLEL PROJECTIONS	PERSPECTIVE PROJECTION
1	Coordinate positions are transformed to the view plane along parallel lines. If COP is located at infinity, all the projectors are parallel and the result is a parallel positions.	Object positions are transformed to the view plane along lines that converge to a point called centre of projection(COP)
2	It does not provide realistic representation but provide accurate views of the various sides of an object.	It produce realistic representation of a 3D object.
3	In parallel projection, these effects are not created.	In perspective projection, objects that are far away appear smaller, and objects that are near appear bigger.
4	The distance of the object from the center of projection is infinite.	The distance of the object from the center of projection is finite.
5	Parallel projection can give the accurate view of object.	Perspective projection cannot give the accurate view of object.
6	The lines of parallel projection are parallel.	The lines of perspective projection are not parallel.
7	Projector in parallel projection is parallel.	Projector in perspective projection is not parallel.
8	Two types of parallel projection : 1.Orthographic, 2.Oblique	Three types of perspective projection: 1.one point perspective, 2.Two point perspective, 3. Three point perspective,
9	It does not form realistic view of object.	It forms a realistic view of object.

Parallel projection

- In parallel projection, coordinate positions are transformed to the view plane along parallel lines.



- The parallel lines are known as projection line and the vector which defines the direction of projection lines called as projector vector.

2 types:

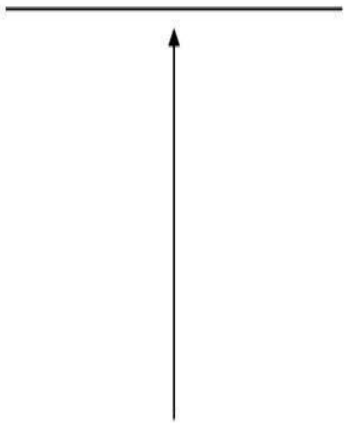
Orthographic :

- When the projection is perpendicular to the view plane.
- In short, direction of projection = normal to the projection plane.
- The projection is perpendicular to the view plane.

Oblique :

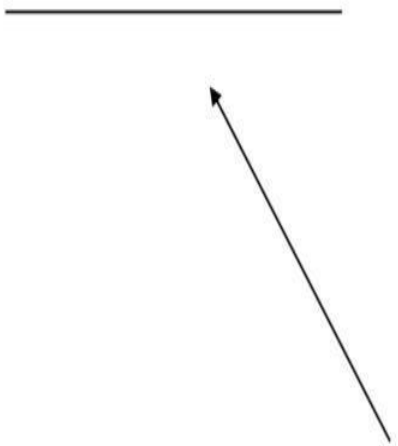
- When the projection is not perpendicular to the view plane.
- In short, direction of projection not equal to the normal to the projection plane.
- Not perpendicular.

- Orthographic projection



when the projection is perpendicular to the view plane

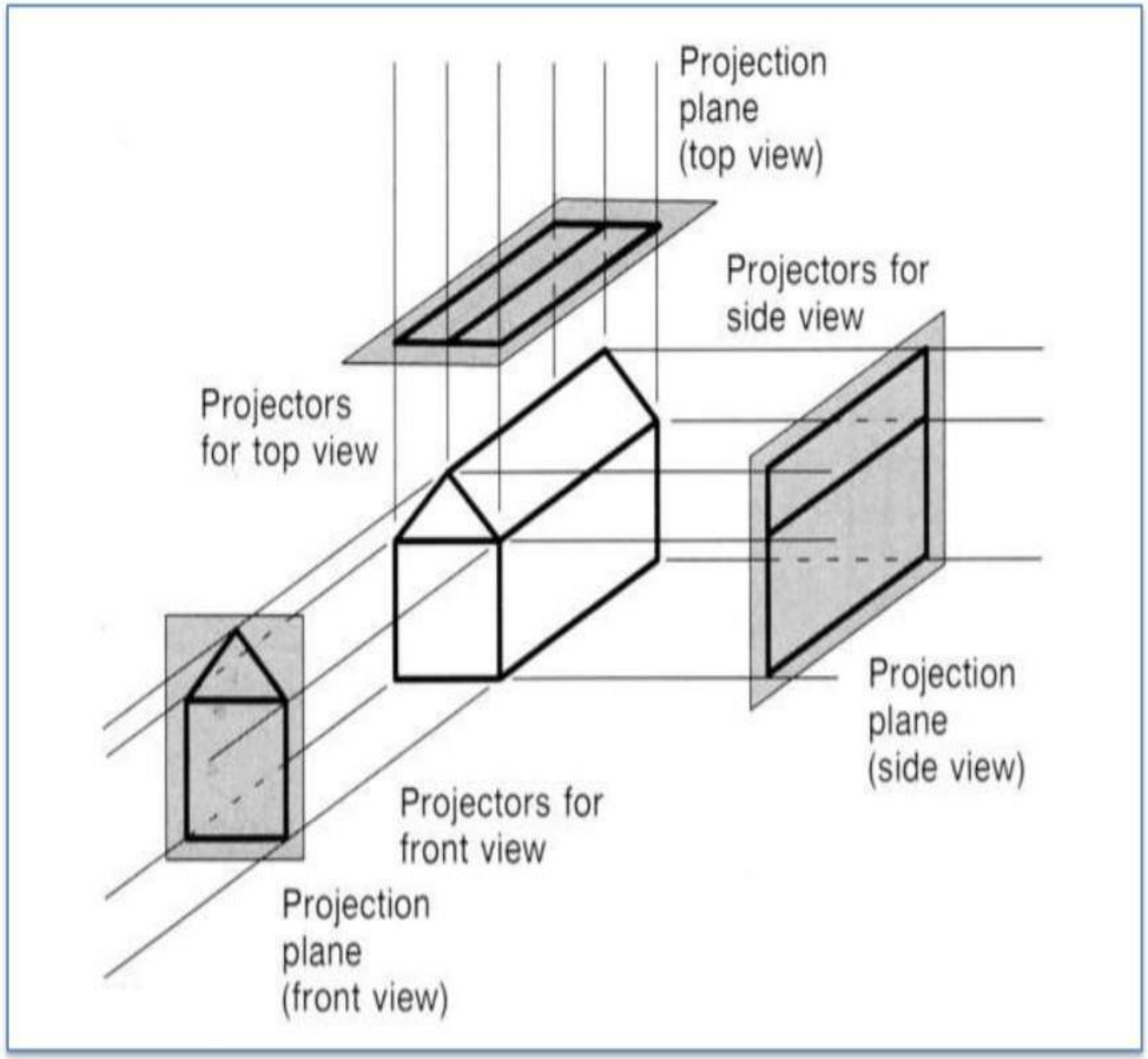
Oblique projection



when the projection is not perpendicular to the view plane

Orthographic (or orthogonal) projections

- **Front, side and rear** orthographic projection of an object are called **elevations** and the top orthographic projection is called **plan view.**
- All have projection plane perpendicular to a principle axes.
- Here length and angles are accurately depicted and measured from the drawing, so engineering and architectural drawings commonly employee this.
- However, **As only one face** of an object is shown, it can be hard to create a mental image of the object, even when several views are available.


Projection
plane
(top view)
Projectors for
side view
Projectors
for top view
Projection
plane
(side view)
Projectors for
front view
Projection
plane
(front view)

Axonometric orthographic projections

- Orthographic projections that **show more than one face of an object** are called axonometric orthographic projections.
- In this projection it display more than one face of an object that is at least three faces are shown by manipulating the object. (applying basic transformations on the object)
- It is divided into 3 types based on the foreshortening factor.

Isometric , dimetric and trimetric

Foreshortening factor

- It is the ratio of the projected line to true length of a line.

Trimetric

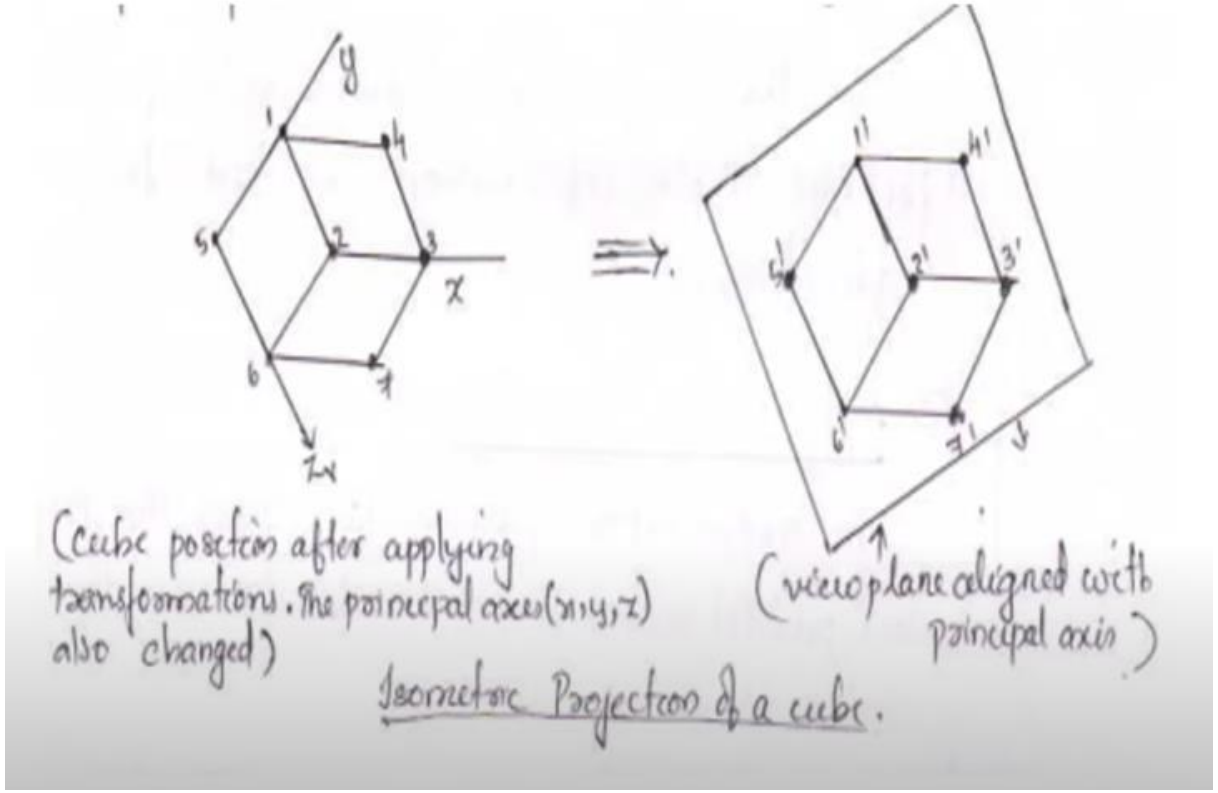
- Here all the 3 foreshortening factors (corresponding to each principle axis) are different

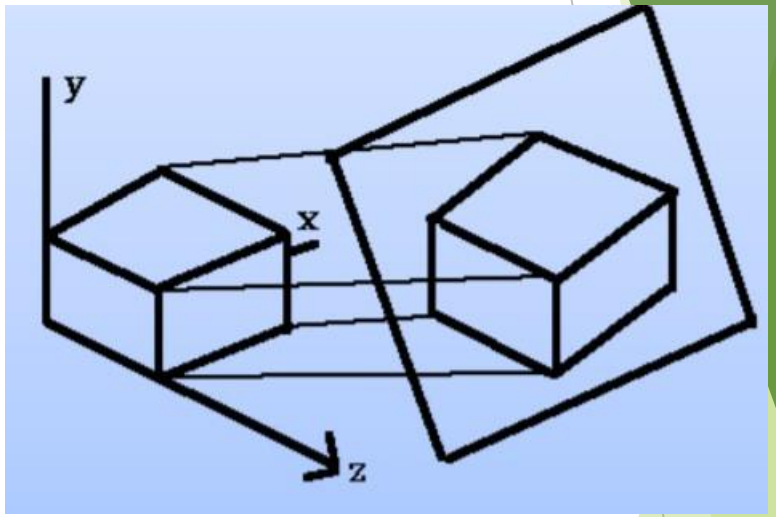
Dimetric

- 2 foreshortening factors are same.

Isometric

- All the 3 foreshortening factors (corresponding to each principle axis) are equal.
- This isometric projection is obtained by aligning the projection plane with the principle axis in which the object is defined.

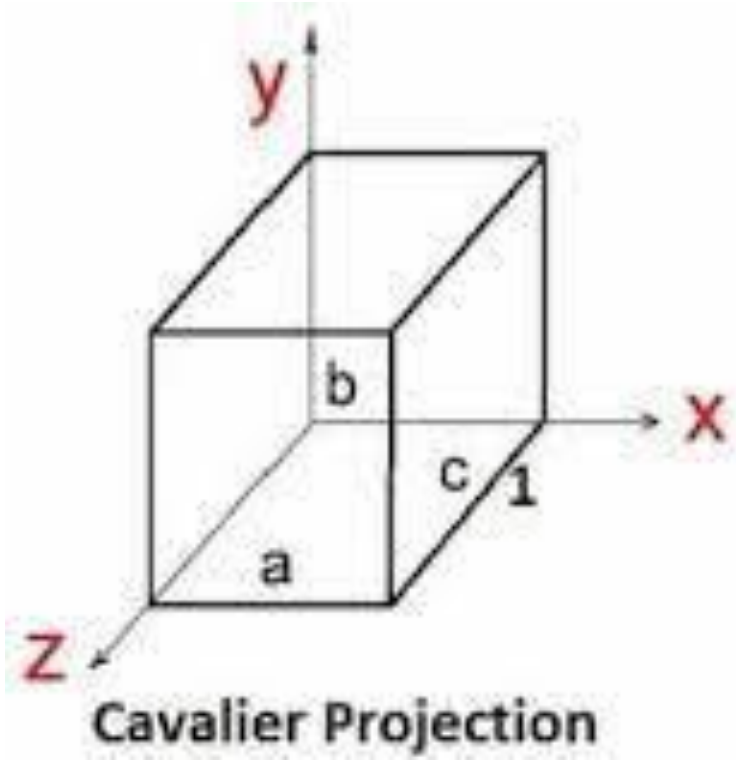

y
x
z
1
2
3
4
5
6
7
1'
2'
3'
4'
5'
6'
7'
(cube position after applying transformations. The principal axes (x,y,z) also changed)
(viewplane aligned with principal axis)
Isometric Projection of a cube.


y
x
z

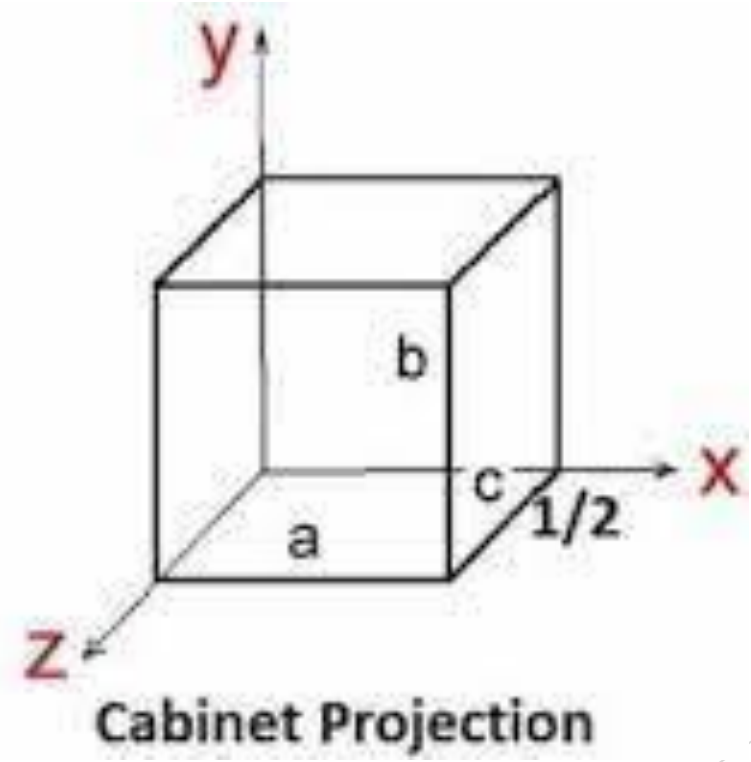
2 common oblique parallel projections: Cavalier and Cabinet

- In oblique projection, the direction of projection is not normal to the projection of plane.
- In oblique projection, we can view the object better than orthographic projection.
- There are two types of oblique projections − **Cavalier** and **Cabinet**.

Cavalier and cabinet projection



Cavalier Projection



Cabinet Projection

Cavalier projection

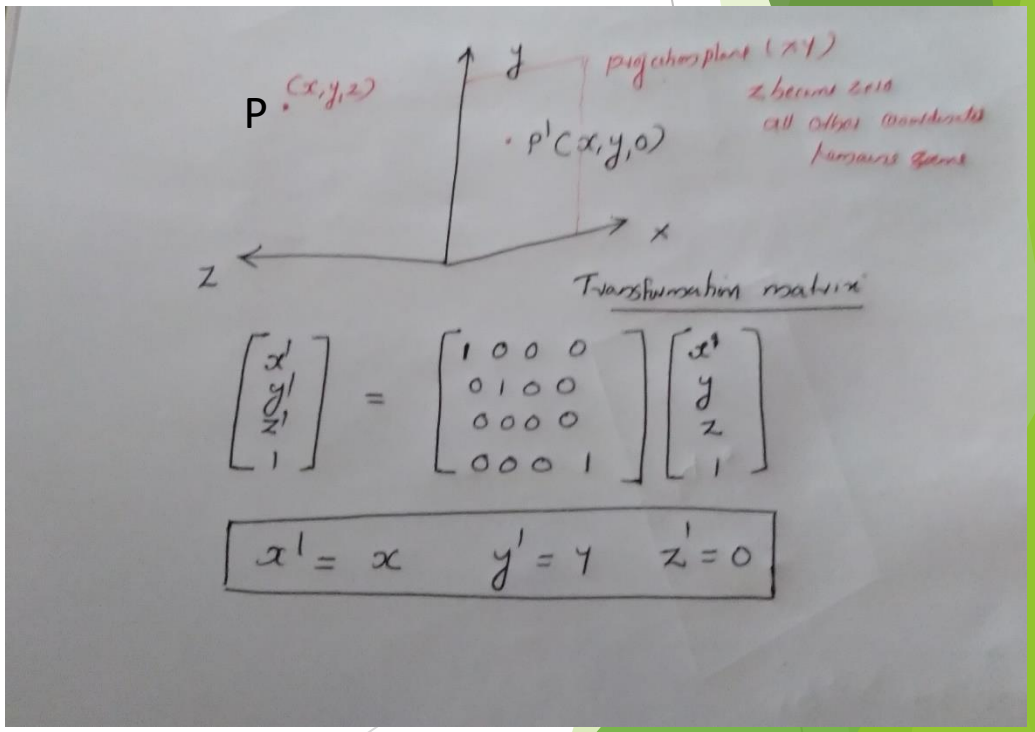
- Cavalier projection: All lines of projections perpendicular to the projection plane are projected with no change in length.
- The **Cavalier projection makes 45° angle with the projection plane**.
- The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection.
- In a cavalier projection, the foreshortening factors for all three principal directions are equal.

Cabinet projection

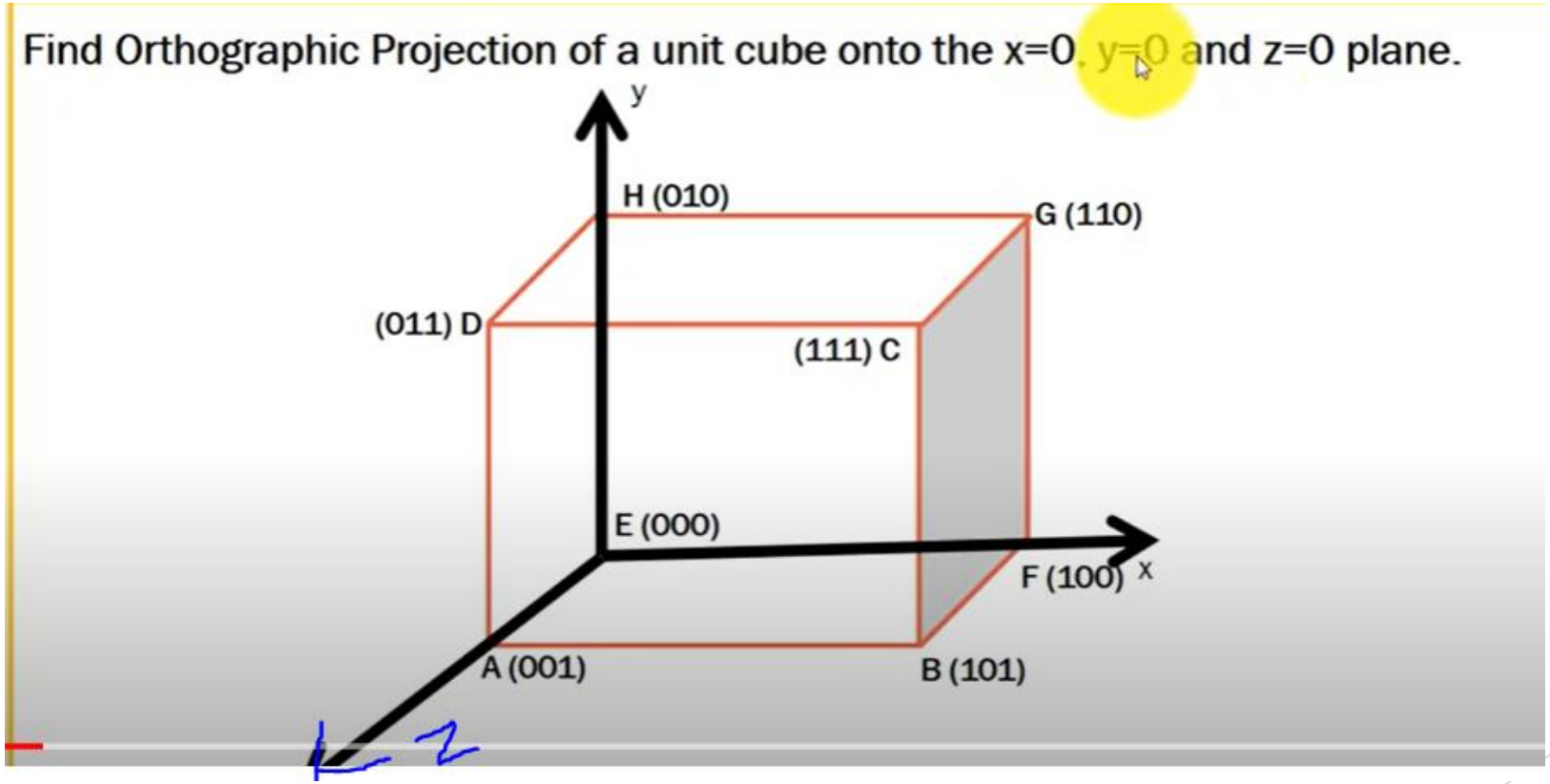
- The **Cabinet projection makes 63.4° angle** with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length.
- Lines which are perpendicular to the projection plane (viewing surface) are projected at 1 / 2 the length .
- This results in foreshortening of the z axis, and provides a more "realistic" view.

Transformation equation for an orthographic parallel projections

- Consider any point (x,y,z) in viewing coordinates , it is transformed to the projection coordinates in XY plane , then z value become zero and other coordinates become unchanged.
- Point p(x,y,z) transformed to the projection plane x-y (x-y plane , z value become zero and x,y remains unchanged).

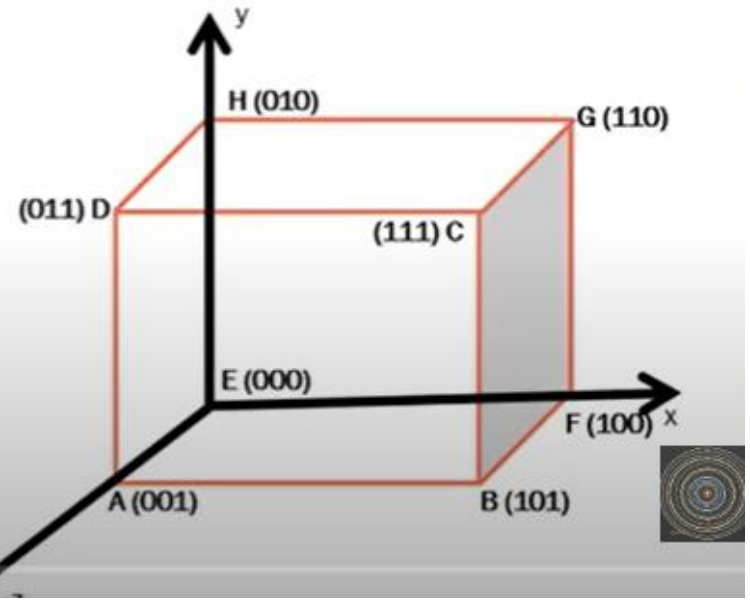


Find the orthographic projection of a unit cube onto the x=0, y=0 and z=0 plane.



Step 01: Write co-ordinates of the unit cube in Matrix Notation.

$$\begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$



Orthographic Projection

1. Matrix for projection when [x=0]

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Matrix for projection when [y=0]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Matrix for projection when [z=0]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic Projection

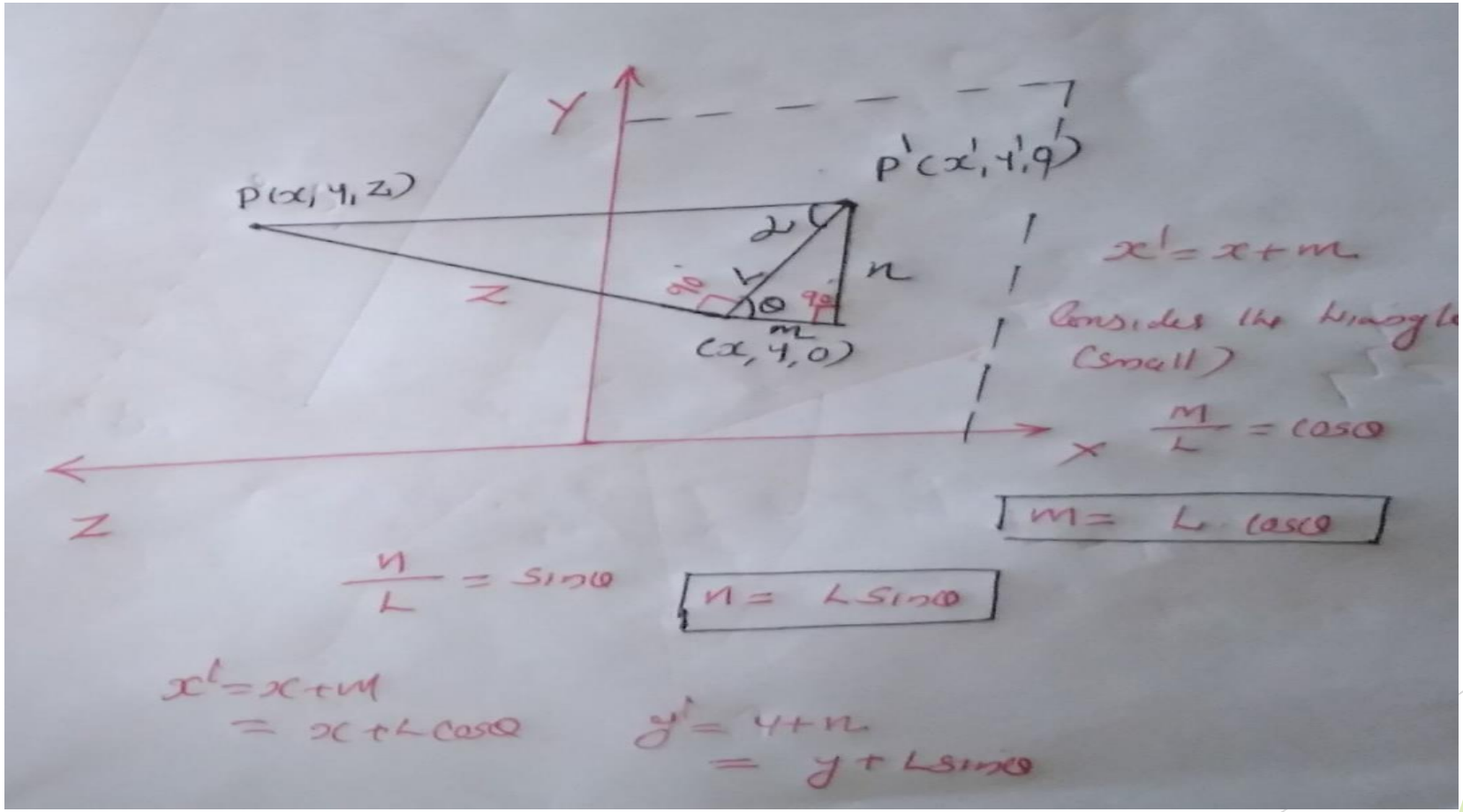
1. Matrix for projection when [x=0] PLANE IS $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Therefore Orthogonal Projection performed as:

$$\begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

- Similarly compute orthographic projection when y=0 and z=0

Transformation equation for an oblique parallel projections



- In order remove L , consider large triangle

tan α = z/L

L= z.cot α

Substitute L in the above equation

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

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

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

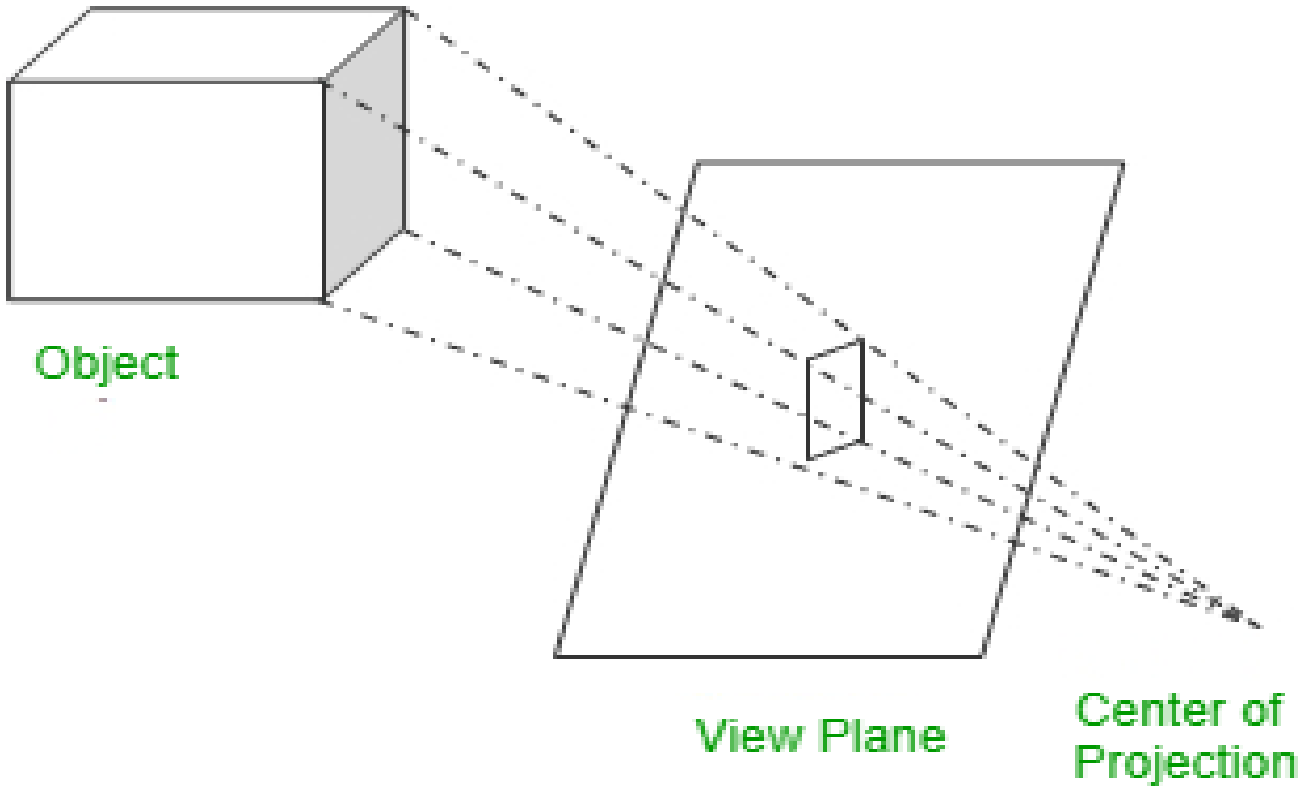
$y' = y + z.\cot\alpha .\sin\theta$

$z' = 0$

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

Perspective projections

- Object positions are transformed to the view plane along lines that converge to a point called **center of projection(COP)**
- In **Perspective Projection** the center of projection is at **finite distance** from projection plane.
- This projection produces **realistic views** but does **not preserve relative proportions of an object dimensions.**
- Projections of distant object are smaller than projections of objects of same size that are closer to projection plane.


Object
View Plane
Center of
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.

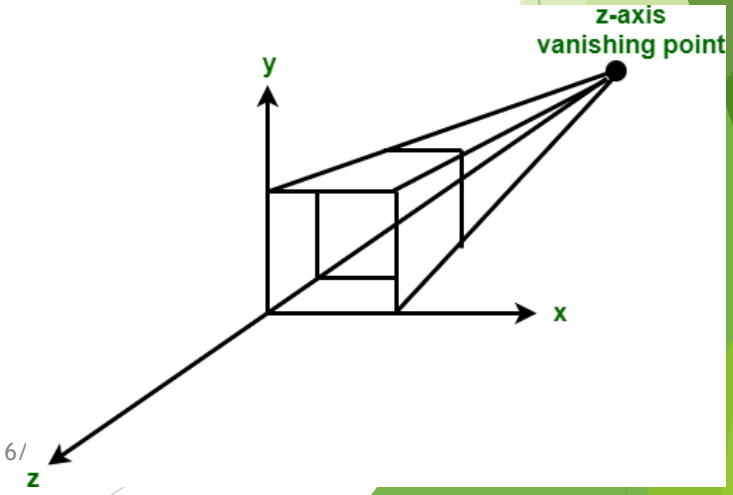
- Parallel lines that are parallel to the view plane will be projected as parallel lines.
- Any set of parallel lines of objects that are not parallel to the projection plane are projected into converging lines .
- The point at which set of projected parallel lines appears to converge is called **vanishing 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.**

A **vanishing point** can be seen at the far end of this railroad.

Types of perspective projection

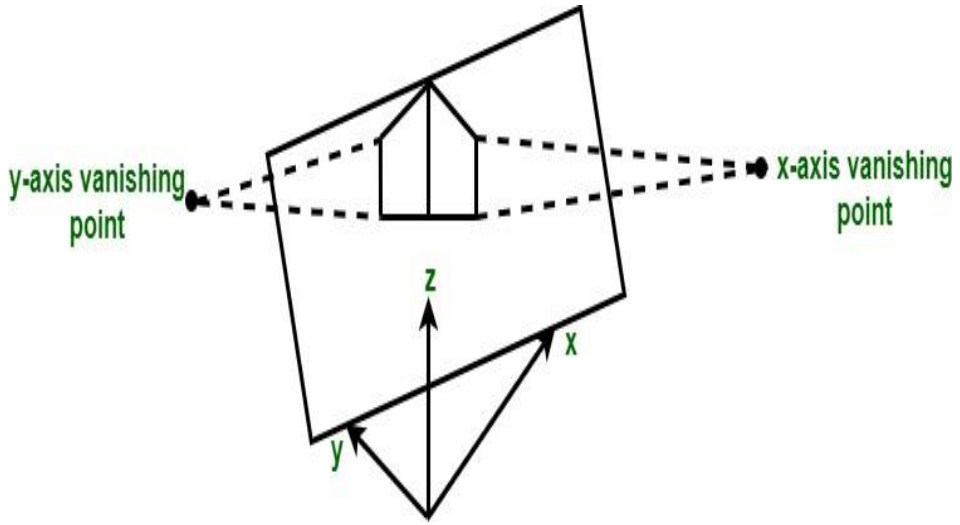
One Point Perspective Projection

- One point perspective projection occurs when any one of principal axes intersects with projection plane or we can say when projection plane is perpendicular to principal axis.
- In the figure, z axis intersects projection plane whereas x and y axis remain parallel to projection plane.



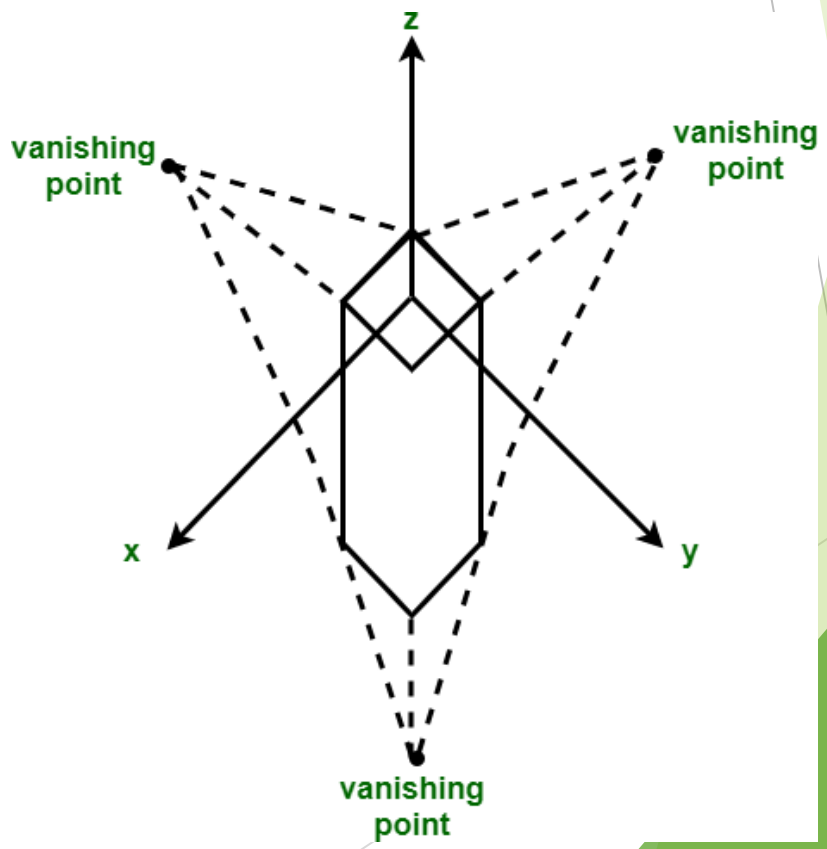
Two Point Perspective Projection

- Two point perspective projection occurs when projection plane intersects two of principal axis.
- In the figure, projection plane intersects x and y axis where as z axis remains parallel to projection plane.

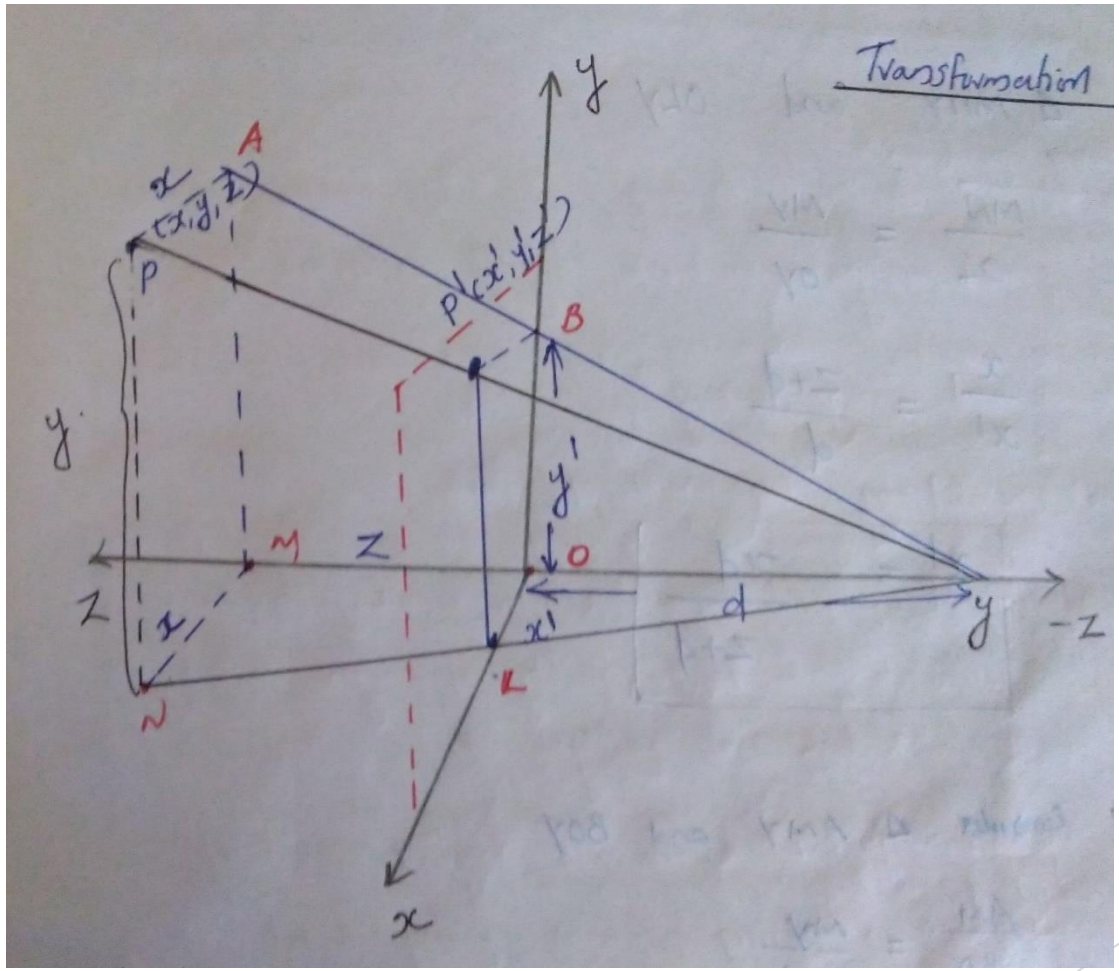


Three Point Perspective Projection

- Three point perspective projection occurs when all three axis intersects with projection plane. There is no any principle axis which is parallel to projection plane.



Transformation equation for perspective projection



- **P** be the point of an object, after projection the point P will be at xy plane which are not equal to x,y z because all the rays are not parallel to each other.
- New projected point **P' =(x',y',z')** where z'=0 here all the rays from the object converge at a point that point is y.
- Draw a perpendicular to zy plane.
- Draw a perpendicular to z axis.
- Again draw perpendicular to z axis, which is the **reflection of x**. using the property of congruent triangle , we should get the value of x' and y'.
- Using the property of similar triangle the ratio of similar side will be same.
- Consider the triangle △ MNY and OLY
- $\frac{MN}{OL} = MY/OY$
- $\frac{x}{x'} = (z+d)/d$
- $\boldsymbol{x' = xd/(z+d)}$

- Again Consider the triangle △ AMY and BOY
- $\frac{AM}{BO} = MY/OY$
- $\frac{y}{y'} = (z+d)/d$
- $\boldsymbol{y' = yd/(z+d)}$

For a homogenous matrix for this,

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

Here it is homogenous coordinate system , h not equal to 1 , for getting the exact divide x', y' by h

h=z+d

$\boldsymbol{x' = xd/(z+d)}$ $\boldsymbol{y' = yd/(z+d)}$ **z'=0**

Visible surface detection methods

Problem definition of Visible-Surface Detection Methods

To identify those parts of a scene that are visible from a chosen viewing position. Surfaces which are obscured by other opaque surfaces along the line of sight (projection) are invisible to the viewer.

Characteristics of approaches:

- Require large memory size?
- Require long processing time?
- Applicable to which types of objects?

Considerations

- Complexity of the scene
- Type of objects in the scene
- Available equipment
- Static or animated?

Classification of Visible-Surface Detection Algorithms

Object-space Methods

- Compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible:

•For each object in the scene do

Begin

▶ Determine those parts of the object whose view is unobstructed by other parts of it or any other object with respect to the viewing specification.

▶ Draw those parts in the object color.

▶ End

▶Compare each object with all other objects to determine the visibility of the object parts.
- If there are n objects in the scene, **complexity = O(n2)**

- **Display is more accurate but computationally more expensive** as compared to image space methods because step 1 is typically more complex, eg. Due to the possibility of intersection between surfaces.

- Suitable for scene with **small number of objects and objects with simple relationship with each other.**

Image-space Methods (Mostly used)

- Visibility is determined point by point at each pixel position on the projection plane.

For each pixel in the image do

Begin

Determine the object closest to the viewer .

Draw the pixel in the object color.

End

- For each pixel, examine all n objects to determine the one closest to the viewer.
- If there are p pixels in the image, complexity depends on n and p (O(np)).
- Accuracy of the calculation is bounded by the display resolution.
- A change of display resolution requires re-calculation.

Object space method	Image space method
Compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible	Visibility is determined point by point at each pixel position on the projection plane.
The Object-space method is implemented in physical coordinate system.	Image-space method is implemented in screen coordinate system.
Compare each object with all other objects to determine the visibility of the object parts. - If there are n objects in the scene, complexity = O(n2)	If there are p pixels in the image, complexity depends on n and p (O(np))
If the number of objects in the scene increases, computation time also increases.	In this method complexity increase with the complexity of visible parts.
It is performed at the precision with which each object is defined, No resolution is considered.	It is performed using the resolution of the display device.
These were developed for vector graphics system.	These are developed for raster devices.
Example : Back face removal algorithm	Example : Depth buffer algorithm, A- Buffer algorithm , Scan line algorithm

- **Object space methods-** Back face removal

- **Image space methods-** Z- Buffer algorithm(Depth buffer algorithm) A-Buffer algorithm, Scan line algorithm
- **Combined** – Depth sorting algorithm

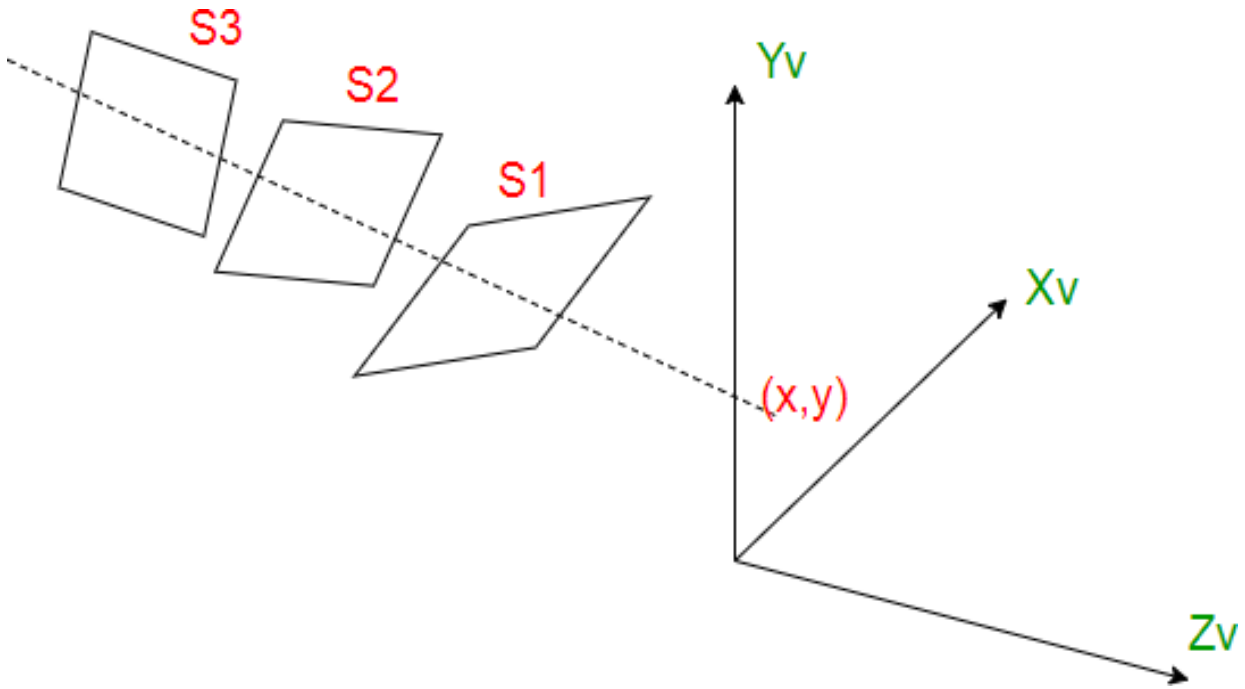
Depth buffer algorithm (Z-buffer algorithm)

- A commonly used image-space approach to detecting visible surfaces is the depth-buffer method, which compares surface depths at each pixel position on the projection plane.

- This procedure is also referred to as the z-buffer method, since object depth is usually measured from the view plane along the z axis of a viewing system.

- Each surface of a scene is processed separately, one point at a time across the surface.

- The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement.
- With object descriptions converted to projection coordinates, each (x, y, z) position on a polygon surface corresponds to the orthographic projection point (x,y) on the view plane.
- Therefore, for each pixel position (x, y) on the view plane, object depths can be compared by comparing z values.


S3
S2
S1
Yv
Xv
Zv
(x,y)

- We can implement the depth-buffer algorithm in normalized coordinates, so that z values range from 0 at the back clipping plane to Z_{max} at the front clipping plane.

- The value of Z_{max} , can be set either to 1 (for a unit cube) or to the largest value that can be stored on the system.

Two buffer areas are required.

- A **depth buffer** is used to store depth values for each (x, y) position as surfaces are processed, and the **refresh buffer** stores the intensity values for each position

- Initially, all positions in the depth buffer are set to 0 (minimum depth), and the refresh buffer is initialized to the background intensity.

- Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z value) at each (x, y) pixel position.

- The calculated depth is compared to the value previously stored in the depth buffer at that position.

- If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the refresh buffer.

- **There are two cases**
- First – viewer is placed at the positive z- axis and viewing alone negative z- axis.
- Second - viewer is placed at the negative z- axis and viewing alone positive z- axis.

Algorithm- viewer placed at positive z-axis

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x, y),

$$\text{depth}(x, y) = 0, \qquad \text{refresh}(x, y) = I_{\text{backgnd}}$$

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

$$\text{depth}(x, y) = z, \qquad \text{refresh}(x, y) = I_{\text{surf}}(x,y)$$

where I_{backgnd} is the value for the background intensity, and $I_{\text{surf}}(x,y)$ 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 the corresponding intensity values for those surfaces.

- Depth values for a surface position (x, y) are calculated from the plane equation for each surface

$$z = \frac{-Ax - By - D}{C}$$

- For any scan line , adjacent horizontal positions across the line differ by 1, and a vertical y value on an adjacent scan line differs by 1.
- If the depth of position (x, y) has been determined to be z, then the depth z' of the next position (x + 1, y) along the scan line is obtained as

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

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

- The ratio –A/C is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single addition.

- On each scan line we start by calculating the depth on a left edge of the polygon that intersect that scan line .

Algorithm- viewer placed at negative z-axis

- ??????????????????

Advantages:

- Simple to use and implement
- It process one object at a time.
- Can be implemented easily in image space.
- Can be executed quickly, even with many polygons

Disadvantages:

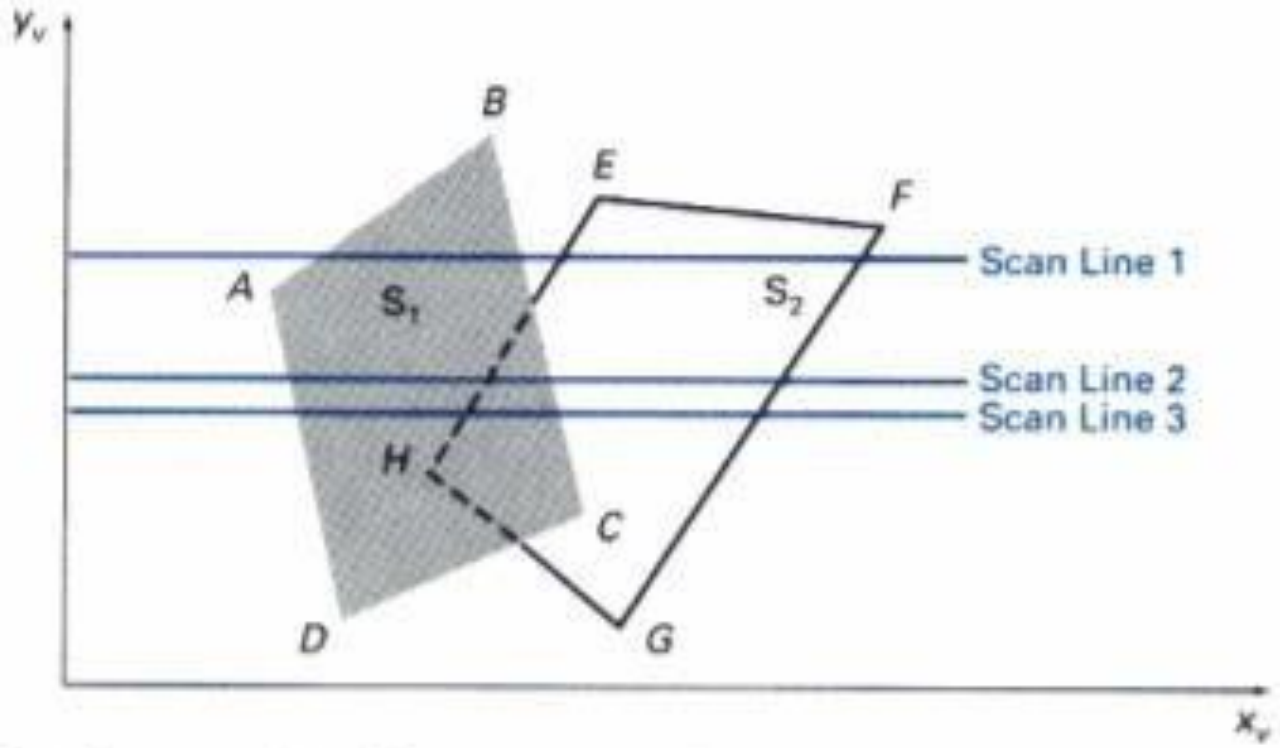
- Takes up a lot of memory because of the two buffers.
- Can't do transparent surfaces without additional code.
- It is time consuming process.

Scan line algorithm

- It is an **image-space method** to identify visible surface and which is an extension of scan line algorithm for filling polygon interiors.
- In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible.
- Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane.
- When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

Tables are set up for the various surfaces

1. Edge table
2. Polygon table
3. Active list of edges



Scan lines crossing the projection of two surfaces, S_1 and S_2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

- The **edge table** contains coordinate endpoints for each line in the scene, the inverse slope of each line, and pointers into the polygon table to identify the surfaces bounded by each line.

- The **polygon table** contains coefficients of the plane equation for surface, intensity information for the surfaces, and possibly pointers into the edge table.

Active list of edges

- To facilitate the search for surfaces crossing a given scan line, **active list of edges** can be used.
- This active list will contain only edges that cross the current scan line, sorted in order of increasing x.
- In addition , we define a flag for each surface that is set on or off to indicate whether a position along a scan is inside or outside of the surface.
- Scan lines are processed from left to right.
- At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

- Figure illustrates the scan-line method for locating visible portions of surfaces for pixel positions along the line.

- The active list for **scan line 1** contains information from the edge table for edges AB, BC, EH, and FG.

- For positions along this scan line between edges AB and BC, only the flag for surface S1 is on.

- Therefore, no depth calculations are necessary, and intensity information for surface S1, is entered from the polygon table into the refresh buffer.

- Similarly, between edges EH and FG, only the flag for surface S2 is on.
- NO other positions along scan line 1 intersect surfaces, so the intensity values in the other areas are set to the background intensity.
- The background intensity can be loaded throughout the buffer in an initialization routine

- For scan lines 2 and 3 in Fig. , the active edge list contains edges AD, EH, BC, and FG.
- Along scan line 2 from edge AD to edge EH, only the flag for surface S1, is on. But between edges EH and BC, the flags for both surfaces are on.
- In this interval, depth calculations must be made using the plane coefficients for the two surfaces.

- For this example, the depth of surface S1 is assumed to be less than that of S2, so intensities for surface S1, are loaded into the refresh buffer until boundary BC is encountered.

- Then the flag for surface S1 goes off, and intensities for surface S2 are stored until edge FG is passed.

- We can take **advantage of coherence** along the scan lines as we pass from one scan line to the next.
- In Fig., scan line 3 has the same active list of edges as scan line2.
- Since no changes have occurred in line intersections, it is unnecessary again to make depth calculations between edges EH and BC.
- The two surfaces must be in the same orientation as determined on scan line 2, so the intensities for surface S1, can be entered without further calculations.
- Any number of overlapping polygon surfaces can be processed with this scan-line method.

- Flags for the surfaces are set to indicate whether a position is inside or outside, and depth calculations are performed when surfaces overlap.
- When these **coherence methods are used**, we need to be careful to keep track of which surface section is visible on each scan line.
- This works only if surfaces do not cut through or otherwise cyclically overlap each other .
- If any kind of cyclic overlap is present in a scene, we can divide the surfaces to eliminate the overlaps.
- The dashed lines in this figure indicate where planes could be subdivided to form two distinct surfaces, so that the cyclic overlaps are eliminated.

- **Advantages**
- Simple
- Potentially fewer quantization errors.
- Each pixel only drawn once

- **Disadvantages:**
- It is time consuming process.

University questions

- Briefly describe the various classification of the visible–surface detection algorithms.
- Is there any point at which a set of projected parallel lines appears to converge? Justify your answer.
- Compare object space and image space method of visible surface detection technique.
- Describe in detail the depth buffer visible surface detection technique.
- Derive the equation to find the depth values for a surface position (x, y).
- Distinguish between cavalier and cabinet projection.
- Explain scan line algorithm with suitable example.

- What is parallel projection? Describe orthographic and oblique parallel projection in detail.
- Distinguish between parallel and perspective projections.
- Describe about the depth-sorting method to display the visible surfaces of any given object with plane faces. Also explain the tests to identify overlapping surfaces.
- Derive the transformation matrix for oblique parallel projection with the help of a neat diagram.
- Explain about different types of parallel projections.
- Write the Z-buffer algorithm for hidden surface removal.
- What is parallel projection? Describe orthographic and oblique parallel projection in detail.

- Explain in detail the scan line algorithm for visible surface detection by pointing out the data structures used in this algorithm.
- How the cyclic overlaps of surfaces are eliminated in scan line algorithm?
- Derive the transformation matrix for perspective projection with the projection reference point at position Zprp along the Zv axis and the view plane at Zvp. Write the perspective transformation equations (i) if the view place is taken to be the uv plane (ii) if the projection reference point is taken to be at the viewing co-ordinate origin.

- Describe about the depth-sorting method to display the visible surfaces of any given object with plane faces. Also explain the tests to identify overlapping surfaces.
- Derive the transformation matrix for oblique parallel projection with the help of a neat diagram.
- Explain about different types of parallel projections.
- Write the Z-buffer algorithm for hidden surface removal.
- Find the window to viewport normalization transformation with window lower left corner at (1,1) and upper right corner at (2,6).
- Find the orthographic projection of a unit cube onto the x=0, y=0 and z=0 plane.
- Explain the need of using vanishing points in projections.
- Describe Sutherland Hodegman polygon clipping algorithm and what are the limitations.