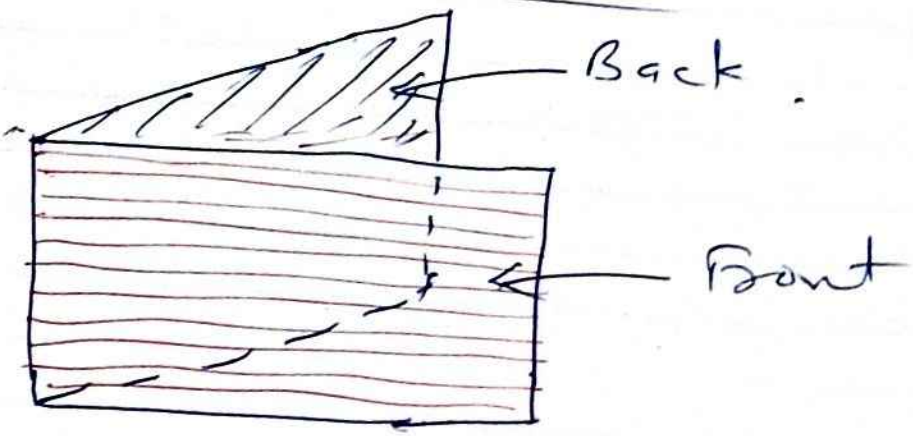
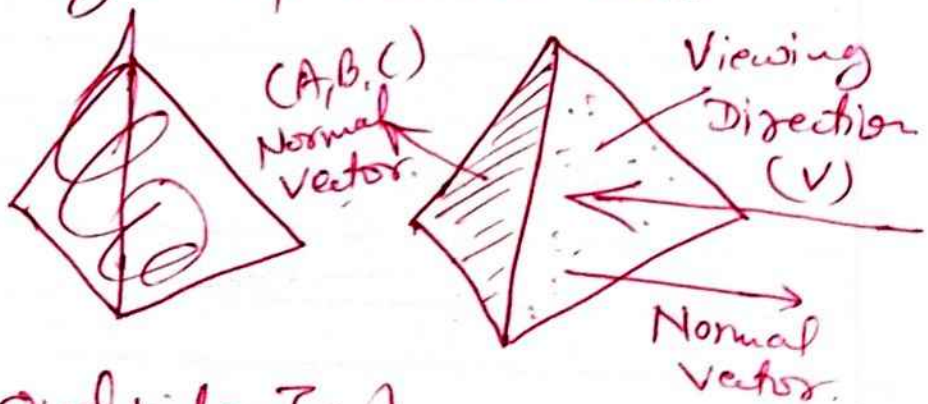
Back-Face Detection Algo.



Visible Surface Detection
or
Hidden Surface Elimination

Back face Detection:

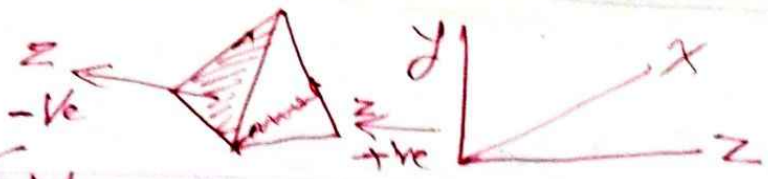
↳ (a) object Space Method
(b) Image Space Method.



Inside-outside Test.

↳ $Ax+By+Cz+D < 0$ → Inside
otherwise - outside.

$V.N > 0$ than Backface
$V.N < 0$ than Front Face



Viewing Vector $(V) = (0, 0, V_z)$

Parallel to z-axis.

$V \cdot N = V_z \cdot C$

$(0,0,V_z)$ (A,B,C) Imp.

$\therefore C < 0$ than Back face.

$V \cdot N > 0$ than Back face.

$V_z = -V_z$

$-V_z * (-C) = V_z C$

If $C \leq 0$

$V \cdot N \geq 0$ than Back Face.

All is Right Handed System

In Left Handed System

For $V \cdot N > 0$

than We have required

$C \geq 0$

Method 2:-

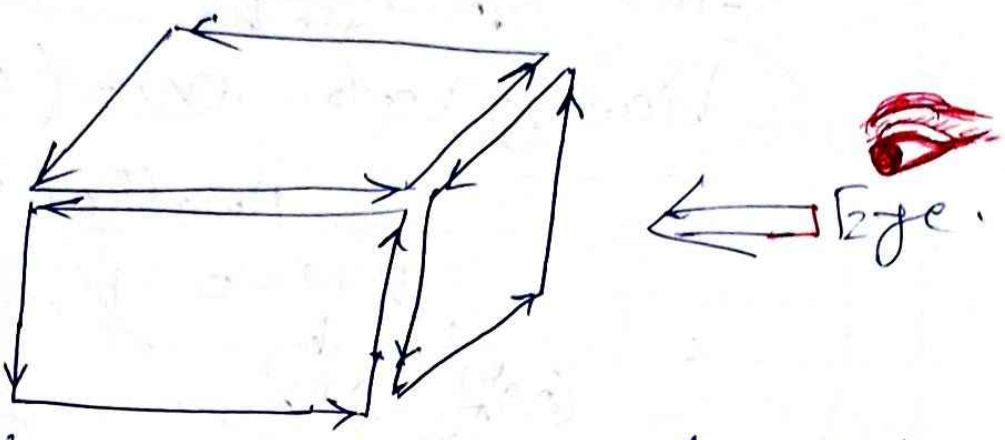


Fig: A polygon visited in counter clockwise Direction

This method was developed by Newell. The problem with the first approach to finding the vector normal to the polygon is that, some searching and checking is required to find a vertex at a convex corner of noncollinear sides. The calculation is -

If the n vertices of the polygon are (x_i, y_i, z_i), then from the sum of overall vertices which is $[a\ b\ c]$ is a vector normal to the polygon. Each of these sums gives twice the area of the projections of the polygon on a plane. If we project the polygon along the x-direction to yz-plane, then the area of that projected polygon is $a/2$.

The value $b/2$ and $c/2$ are the areas of projections on the xz and xy planes respectively. So, a, b, c describe the projection of the polygon in the x, y, and z directions. The amount of area projected along the z-direction. Normal vector $[a\ b\ c]$ is a normal vector to the plane of the polygon.

This method is that there are no special cases to check; we can simply compute the sums.

Suppose that we make the rule that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to us, the dark faces meet the material inside the object. This means that when we look at an object's face from the outside it will appear to be drawn counter-clockwise. If a polygon is visible, the light surface should face towards us and the dark surface should face away from us. Since cross product or sum can be formed which gives the direction of the light face this vector should point towards us.

So, if normal vector points towards the viewer, the face is visible, i.e. a "front face"; otherwise the face is hidden (a back face) and should be removed.

The z-component of the normal vector. If z-component is +ve, then the polygon faces towards the viewer.

If -Ve, the polygon faces away from the viewer. If we have two vector R & S to compare their direction we use the vector dot product

$$a = R \cdot S = |R| \cdot |S| \cos q$$

The cosine factor is important to use because if the vector is in the

same direction ($0 \leq \theta \leq P/2$) then the cosine is +ve and overall dot product is -ve. The Formula for computing dot product is:

$$a = R \cdot S$$
$$= [R_x\ R_y\ R_z] \cdot [S_x\ S_y\ S_z]$$
$$= R_x S_x + R_y S_y + R_z S_z$$

For the back face, check one vector is normal to the polygon and the other is the depth direction $[0\ 0\ 1]$. So, the test for a back face is then a check on the sign of the z-component of the normal vector.

Back-Face Detection:-

Back-face detection is a fast object space algo. based upon the inside-outside test for identifying the back-face of a polyhedron. Suppose a point (x, y, z) is inside a polygon surface with plane parameters A, B, C, and D. If

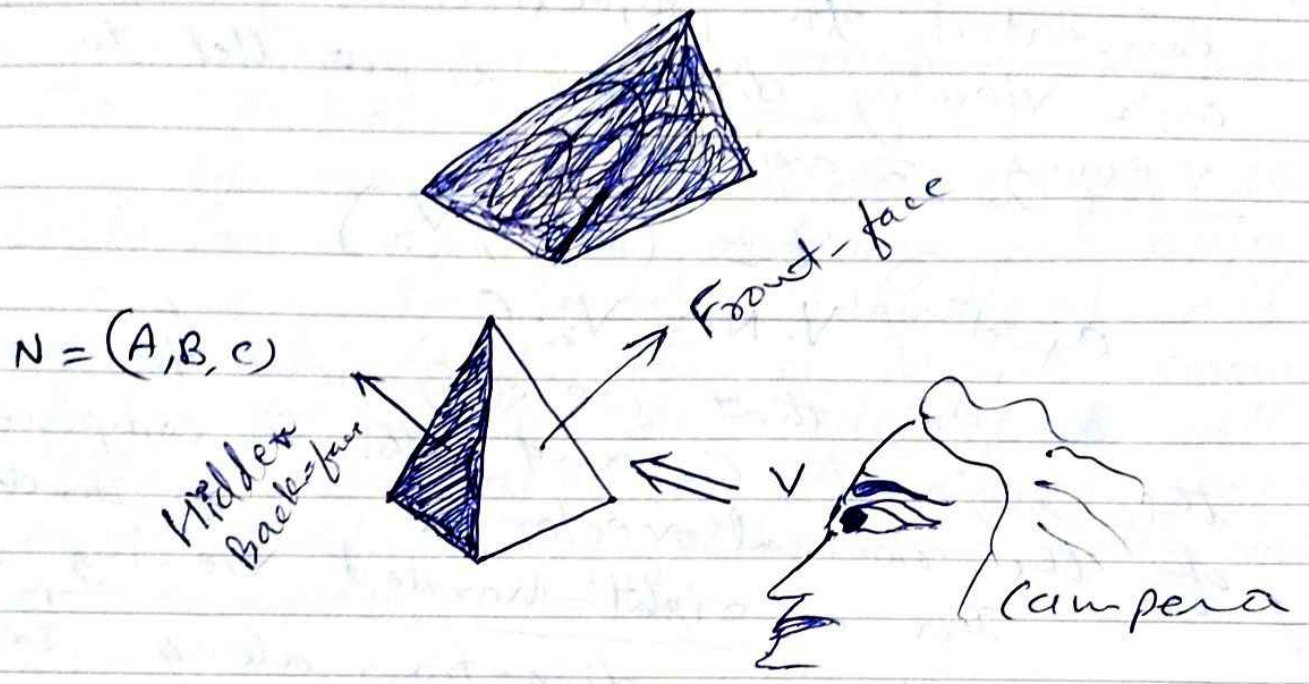
$$Ax + By + Cz + D < 0 \quad \text{---(1)}$$

1) When an inside point is along the line of sight to the surface, the polygon must be a back face.

2) The direction of the light face can be identified by examining the result

$$N \cdot V > 0$$ where

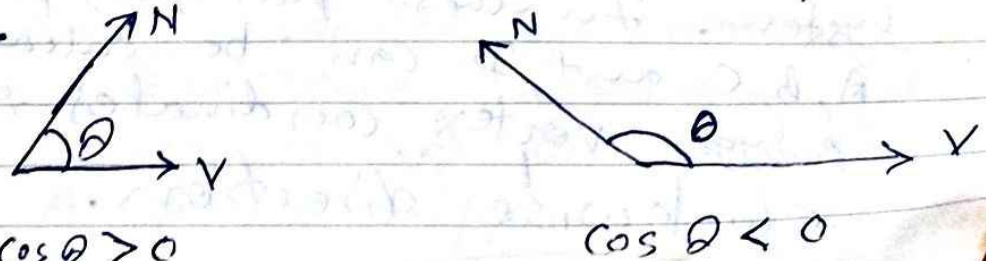
N is the normal vector to the polygon surface with cartesian components A, B, C. and V is a vector in the viewing direction from the eye (or "camera") position.



We know that, the dot product of two vectors, gives the product of the lengths of the two vectors times the cosine of angle between them.

If the vectors are in the same direction i.e. $0 \leq \theta \leq \pi/2$, then the cosine is +ve and the overall dot product is positive.

However, if the directions are opposite i.e. $\pi/2 < \theta \leq \pi$, then the cosine and the overall dot product is -ve.



If the dot product is +ve, we can say that the polygon faces towards the viewer, otherwise, faces away and should be removed.

If object descriptions have been converted to projection co-ordinates and our viewing direction is parallel to the viewing z-axis, then

$$V = (0, 0, V_z)$$

and $$V \cdot N = V_z \cdot C$$

So, that we only need to consider the sign of C and the Z component of the normal vector N.

In a right-handed viewing system (counter-clockwise direction) with viewing direction along the negative Z_v axis, the polygon is a back-face if $C < 0$.

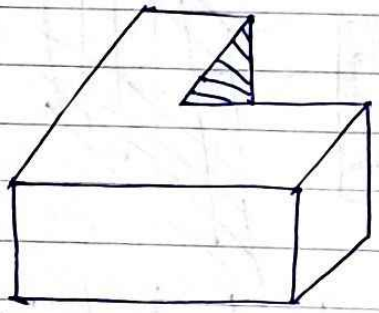
Also, we cannot see any face whose normal has Z component $C = 0$.

Thus, in general, we can label any polygon as a back face if its normal vector has a Z component value $C \leq 0$.

Similary method can be use in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in clockwise direction.

Inequality eq. (1), i.e. $Ax + By + Cz + D < 0$ then remains a valid test for inside points. Also, back faces have normal vectors that point away from the viewing position and are identified by $C \geq 0$ when the viewing direction is along the positive (+ve) z-axis.

By examining parameter C for the different planes defining an object, we can immediately identified all the back faces. For a single convex polyhedron, this test identifies all the hidden surfaces on the object, since each surface is either completely visible or completely hidden..



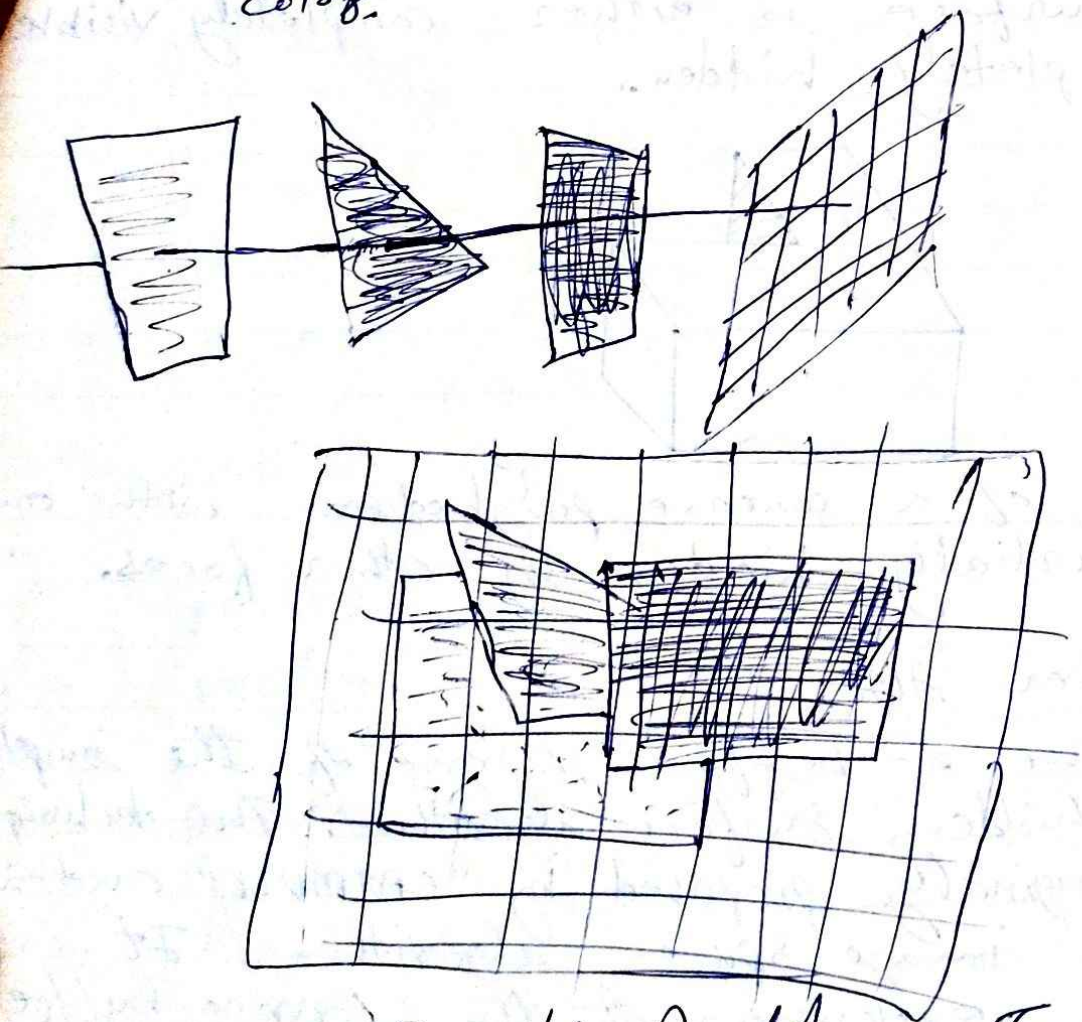
View of a concave polyhedron with one face partially hidden by other faces.

Z-buffer Algo:-

The z-buffer is one of the simplest of the hidden surface algorithms. This technique was originally proposed by "CATMULL" and it is an image space algorithm. It is simple extension of the frame buffer idea. A frame buffer is used to store

to store the intensity (attributes) of each pixel in image space. The z-buffer is a separate depth buffer used to store the z-coordinate or depth of every visible pixel in image space.

In this algo. a buffer of the same size as the framebuffer is set up which holds depth information. Each element of the depth buffer corresponds to a pixel in the framebuffer and initially holds the maximum depth in the scene. The framebuffer is cleared to the background color.



Depth Buffer Technique.

As each polygon is scan-converted, the depth at each pixel is calculated and compared with the corresponding depth in the depth Buffer. If the depth is less than that stored in the depth buffer (i.e. near the viewer) then the pixel is set in the frame buffer with the polygon color at that point and the depth buffer is set to the depth. If the polygon depth is greater (i.e. further away from the viewer) than the depth buffer depth at that point then that pixel is not written to the framebuffer.

Unit - III

Segments :- In the real world, the image contains most of the times, many pictures. An image may contain several views of an object. This may contains the informations, instructions for user and other information too. Now, if an image has a single picture, then we do not face any problems in displaying it. However, when the image has several views, we have many questions in our mind.

(1) How to display all the pictures simultaneously?
(2) Do all the pictures are going to appear simultaneously?
(3) How to handle the individual informations?

Example

D.E. AR-96

Question: in our mind are
1) When and what objects get displayed is a major question?
2) How we have to handle these displays?
3) How the transformations are to be applied to the individual objects?

So that, we need some mechanism for organizing the image.

we know that the image is stored in a display file.

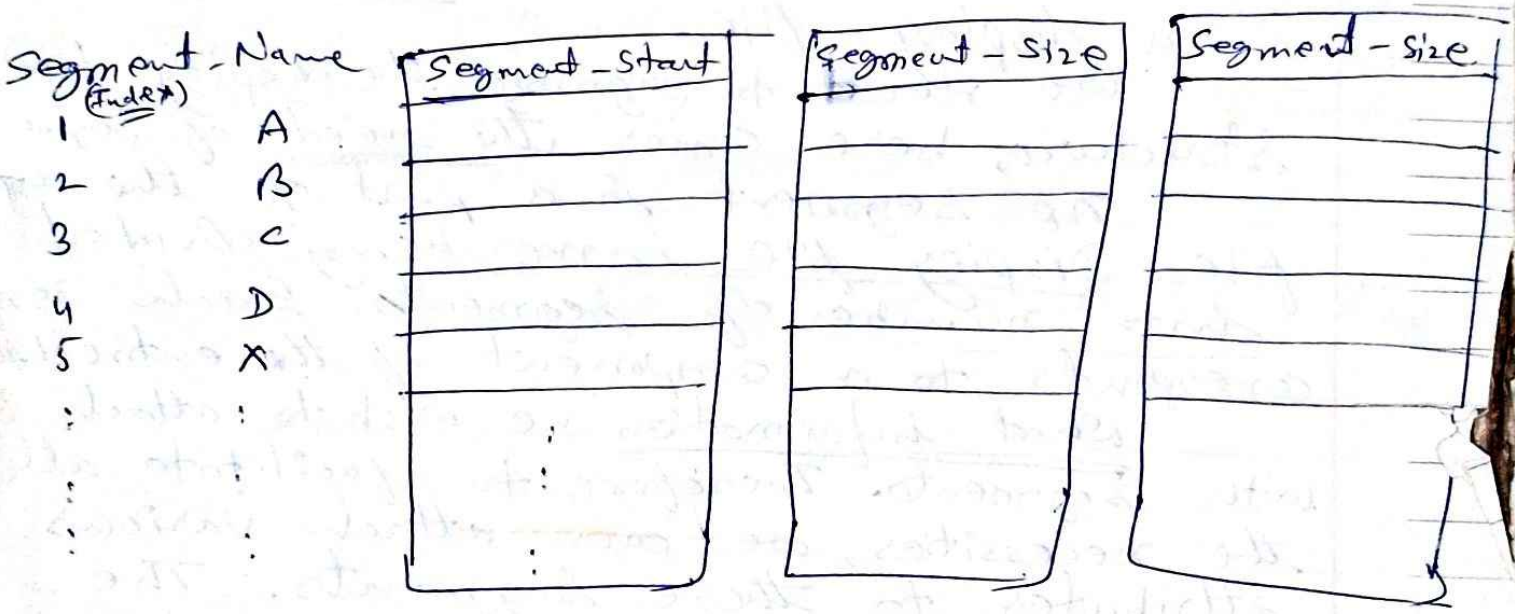
we need to organize the display file structure, here comes the need of segments.

The segment is a part of the display file. Display file is now being divided into number of segments. Each segment coresponds to a component of the entire display.

What information we wish to attach with segments. Therefore to facilitate all the necessities, we can attach various attributes to these segments. The attributes include visibility segment is visible or not. It is obvious that the visible segment will be displayed and non visible segment will not be displayed. The visibility attribute settings can be altered such that we can make the segments visible or invisible can be altered. As the user starts moving from one position, it is obvious that we have to change the visibility attributes of the objects. which are relevent and display them.

To facilitate the individual picture transformation we have an attribute image transformation. for example, a ~~fan~~ fan in a class room, we ~~wish~~ wish to show it as rotating. So, set the image transformation attribute for that segment containing fan, appropriately.

Segment Table: →



Some information is associated with each segment. Necessarily, we realize that, the information should contain:

1) the name (unique) of segment, so that one can access the segment correctly, the naming also helps when we need to change the attributes of the segments.

2) when we refer to a particular display file segment, we must know where the display file instructions for a segment start and end.

5 the class room in computer ?

(25), (26).

This requirement suggests that we need a proper organization of the information. The organization of information helps us to obtain the segment attributes, alter them, interpret the corresponding display file instructions.

It is accomplished by means of segment table. The segment table can be prepared by using arrays. The segment name can be kept in one array. The segment properties can be stored in separate arrays.

In a diagram, suppose if we wish to refer to the segment 'x', then we can search in the array segment name. we find that 'x' is at index 5. We now change the visibility status of 'x', then we can do this by simply giving to the 5th location in the array visibility and change the content. The use of arrays for implementing segment table simplifies the work.

Normally an index '0' can be reserved for unnamed segment.

Segment Creations :→

The segment creation process starts with naming a segment. However one must ensure that there is no other segment open at that time. One more care has to be taken to ensure that no other segments have

the same name as the new one and also the name given to the segment is valid. Now whatever the commands we write next, they refer or belong to the newly created segment. The first instruction within this segment will be located at the next free space in the display file. The other attributes are set to default values and size is set to zero as we have not written anything in the segment.

The procedure is described in the form of an algorithm:

Step 1: Check whether there is any segment open, if so then print "Error - segment still open" and goto Step 9.

Step 2: Read the name for new segment.

Step 3: Check whether segment name is valid (it is based on what restriction you have on naming the segment), if not valid print appropriate error message and goto Step 9.

Step 4: Check whether the segment name is present in the segment name array and if so, display an appropriate error message and goto Step 9.

Step 5: In the segment start array initialize the corresponding entry to the free place in display file.

age class room in computer!

Step 6: Initialize the segment size for this new segment to zero.

Step 7: Initialize all the other parameters/ attributes to default values.
(Note that default values are stored in zeroth entry of each array).

Step 8: Indicate that this new segment is open.

Step 9: Stop.

As mentioned earlier once the segment is opened all operations following it become the members of that segment. When we want to indicate that we are no longer using the segment, then one needs to close the segments. So for closing the segment what we need to do is simply change the value of indicator to default value. The reader is clear of one fact that we have an unnamed segment as zeroth entry. So, while closing the segment we can set the indicator value to zero. However, to ensure that there won't be two unnamed segment instructions, we opt to delete those. But we keep the unnamed segment instruction in a ready position to receive the instructions in next free display file location.

Closing & Deleting a segment algo.

close segment :-

For these purpose following the following steps :-

Step 1: Check whether any segment is open (This can be done by checking the value of an indicator). If no then display "Error. NO segment is open". And goto Step 4

Step 2: Read the name for new segment.

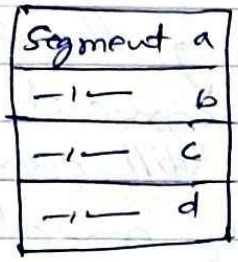
Step 3: Initialize Start Size open segment indicators to the original values.
i.e. Start as free.
Size as 2000.
Open of segment as null.

Step 4: STOP.

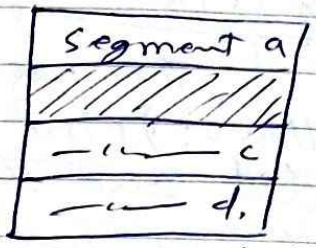
As we indicated that the segment is to be deleted while closing the segment one must understand what happens while deleting. Now what is required is to:

1) Recover the space occupied by the segment which is not required.
2) Reuse the space for other segments.
3) Keep intact the rest of the display file contents.

Since, we are descussing ~~for~~ the display file structure by means of an array, the deletion is just shifting of every element one position up.

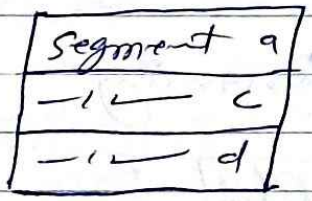


Before Deletion



After Deletion

Then ↓



After shifting.

Note

Remaining Flow chart are in after turn two pages.

P.T.O. ⇒

~~*~~ Renaming a Segment :→ This operation is very useful. When we are using motion pictures or animated pictures, in that we are first displaying an object, then deleting that object, changing its position and again displaying the object. The problem with this is during the time after first image is deleted and next image is completed, only partially completed image may get displayed. So to avoid this problem we should not delete a segment till the replacement for it is completed. This means that both segments must exist in the display file

at the same time. We do this by building the new invisible image under some temporary name. When it is completed we can delete the original image and make replacement image visible. The idea of maintaining two images, one to show and one to build is called as "Double Buffering".

The method of renaming the segment is as follows: 1) First checks the segment names are valid and they are not still open. 2) It also checks against using the name of an already existing segment. 3) If these conditions are met, the segment table entries for the old name are copied into the new name positions and the size of the old segment is set back to 0.

Method

(a) Segment table before Renaming:

Segment Name	segment start	segment size	Visibility	...
1	1	3	ON	
2	4	10	ON	...
3	14	5	ON	

(b) Segment table having both segments 2 and four:

Segment Name	Segment Starts	segment Size	Visibility	...
1	1	3	ON	
2	4	10	ON	...
3	14	5	ON	
4	-	-	OFF	

(c) Segment table after renaming:

segment Name	segment start	segment size	visibility	---
1	1	3	ON	. . .
2	4	0	ON/OFF	. . -
3	14	5	ON	. . .
4	4	10	ON	- -

suppose we are maintaining three segments and if we want to rename the second segment by new segment then we are creating new segment four as a temporary segment with visibility as OFF, so that it will not get displayed. Once the segment four is completed then we are deleting segment two and copying all the attributes of segment two to new segment four. After that, we have to make the size of segment two as zero to indicate that it is deleted.

~~Posting of Copying a segment :→~~

The renaming algo:-

Step 1) Varify that the segment names are valid and are still open.

Step 2) check for the new name which you are trying to give for the existing segment is valid (i.e. this new name is not present in segment table).

Step 3) If the conditions in step 1 and 2 are valid then copy the segment table contents (of the segment which is to be renamed into the new name's entry).

Step 4) STOP.

Posting & Unposting A segment :-

We can extend the usefulness of our graphics package considerably by allowing segments to become temporarily invisible. For this we use two additional functions :

1) Posting & 2) Unposting.

1) "Posting" is the action of including a segment in the display refresh cycle, thus posting makes a segment visible and

2) "Unposting" remove the segment from the refresh cycle, so that it is no longer visible. Thus,

Post segment (n) add segment n to the refresh cycle

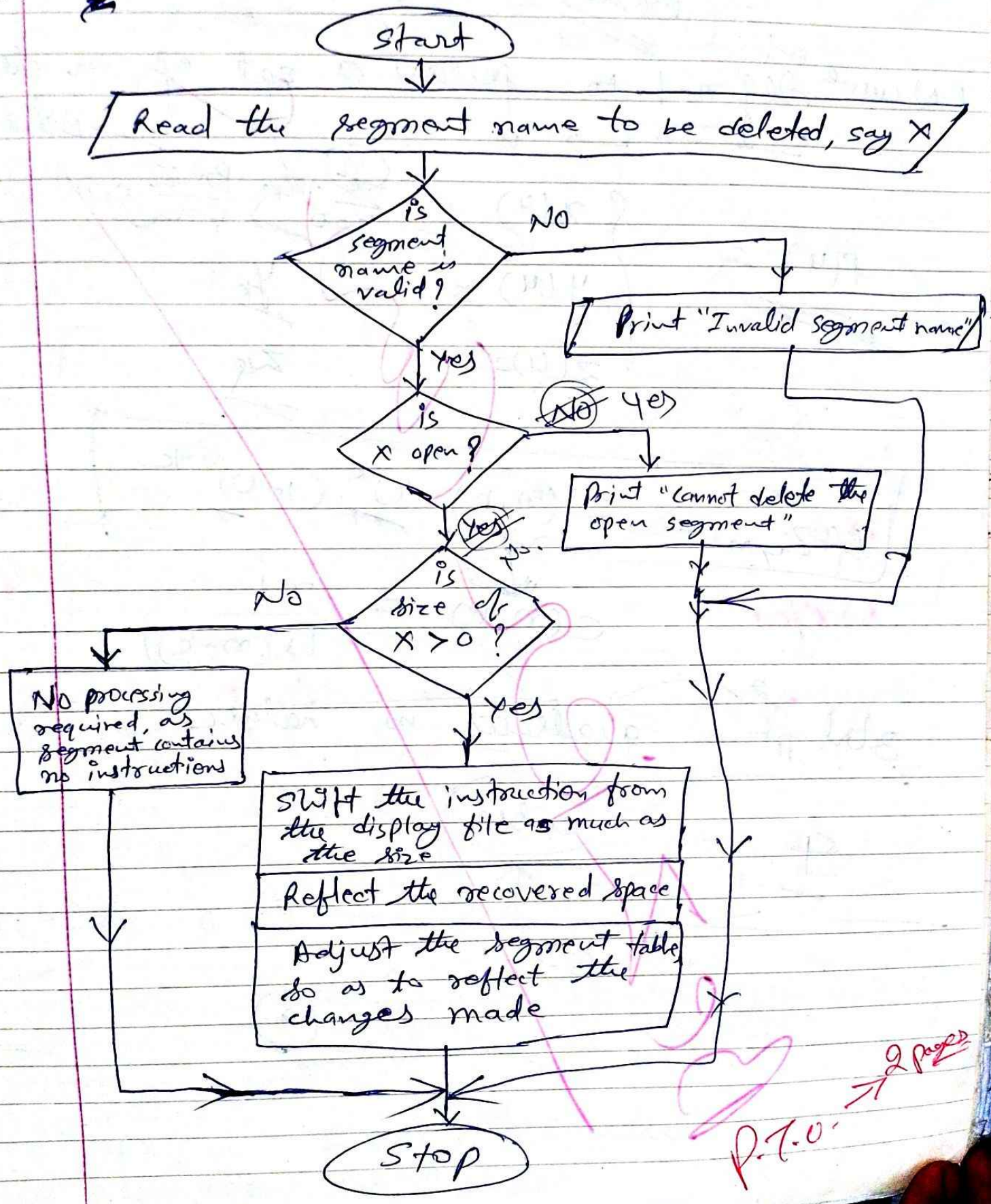
Unpost segment (n) remove segment n from the refresh cycle.

By unposting, the segment is not destroyed and so it can be rendered visible again without redefining it, simply by posting it.

Posting & Unposting are particularly useful in avoiding the need for repeatedly redefining graphics overlays, command menus and other segments that are removed only temporarily from the screen.

Remainning Part

Flow chart for Deleting a segment



P.T.O. → 2 pages

Unit-IV

Projection

Expt. No. ..... Page No. ① Date .....

Three - Dimensional Concepts

Object boundaries can be constructed with various combinations of plane and curved surfaces, and we sometimes need to specify information about object interiors.

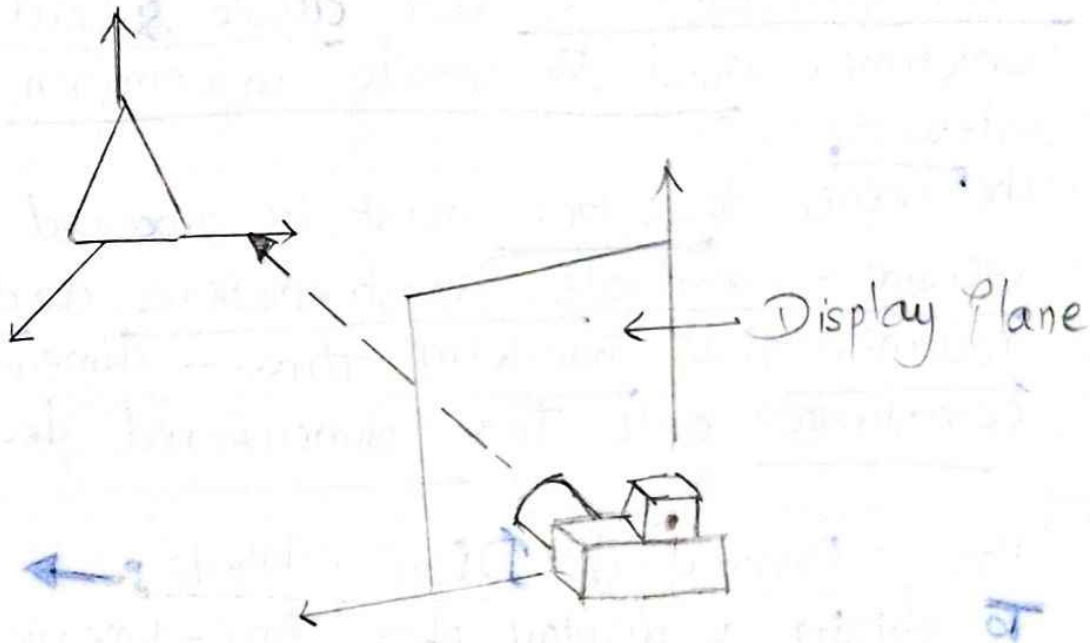
The scene description must be processed through viewing - coordinate transformations and projection routines that transform three - dimensional viewing co-ordinates onto two - dimensional device coordinates.

Three - Dimensional Display Methods :→

To obtain a display of a three - dimensional scene that has been ~~modeled~~ modeled in world coordinates, we must first set up a coordinate reference for the "camera". This coordinate reference defines the position and orientation for the plane of the camera film (fig a), which is the plane we want to use to display a view of the objects in the scene. Object descriptions are then transferred to the camera reference co-ordinates and projected onto the selected display plane. We can display the objects in wireframe (outline) form, or we can apply lighting and surface - rendering techniques to shade the visible surfaces.

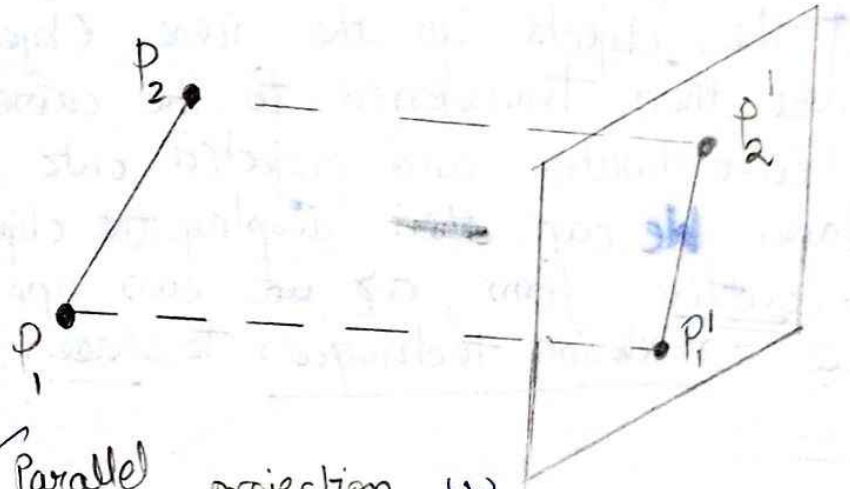
Teacher's Signature : .....

Fig. (a)



Coordinate reference for obtaining a particular view of a three-dimensional scene

Fig (b)



Parallel projection (b) of an object to the view plane.

Parallel Projection

In a parallel projection, parallel lines in the world-coordinate scene project into parallel lines on the two-dimensional display plane. This technique is used in engineering and architectural drawings to represent an object with a set of views that maintain relative proportions of the object. (fig 6)

Perspective Projection

In this type of projection, parallel lines in a scene that are not parallel to the display plane are projected into converging lines. Scenes displayed using perspective projections appear more realistic. For a perspective projection, object positions are transformed to the view plane along lines that converge to a point called the projection reference point (or center of projection). (fig 6)

A parallel projection preserves relative proportions of objects, and this is the method used in drafting to produce scale drawings of three-dimensional objects.

A perspective projection, on the other hand, produces realistic views but does not preserve relative proportions.

Teacher's Signature : .....

fig (c) Perspective projection of an object to the view plane

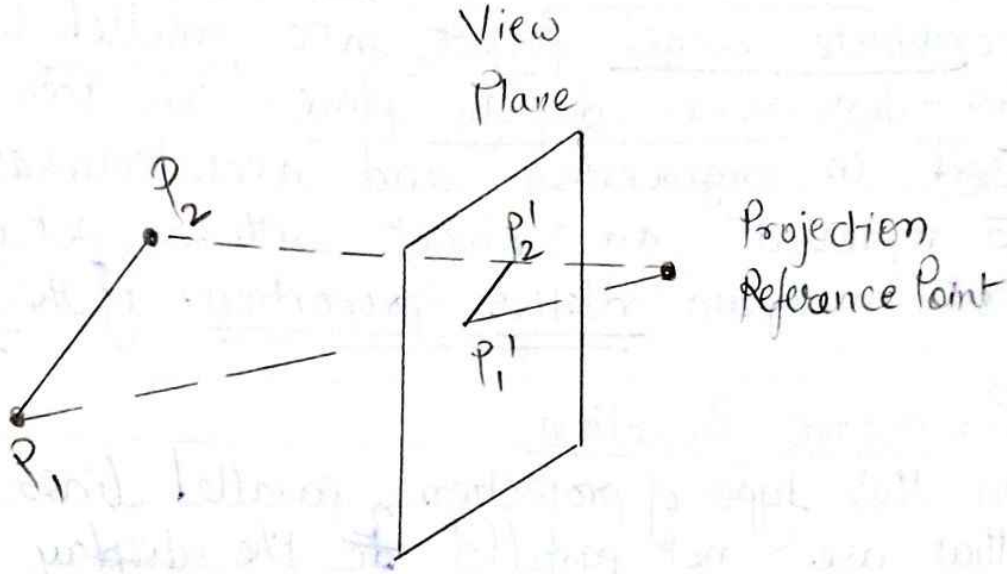


fig (d) Orthographic Projection

fig (e) Oblique Projection

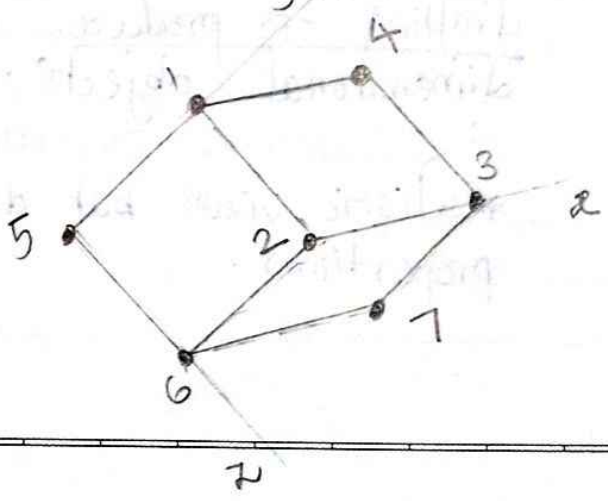
Orientation of the projection vector V_p to produce an orthographic projection (d) & an oblique projection (e)

fig (f)

Isometric projection for a cube

or

axonometric orthographics projection



We can specify a parallel projection with a projection vector that defines the direction for the projection lines. When the projection is perpendicular to the view plane, we have an orthographic parallel projection. ~~Otherwise, we have an orthographic parallel projection.~~ Otherwise, we have an oblique parallel projection (fig e)

Orthographic Projections

These are most often used to produce the front, side and top views of an object (fig d). Front, side and rear orthographic projections of an object are called elevations; and a top orthographic projection is called a plan view or Top views.

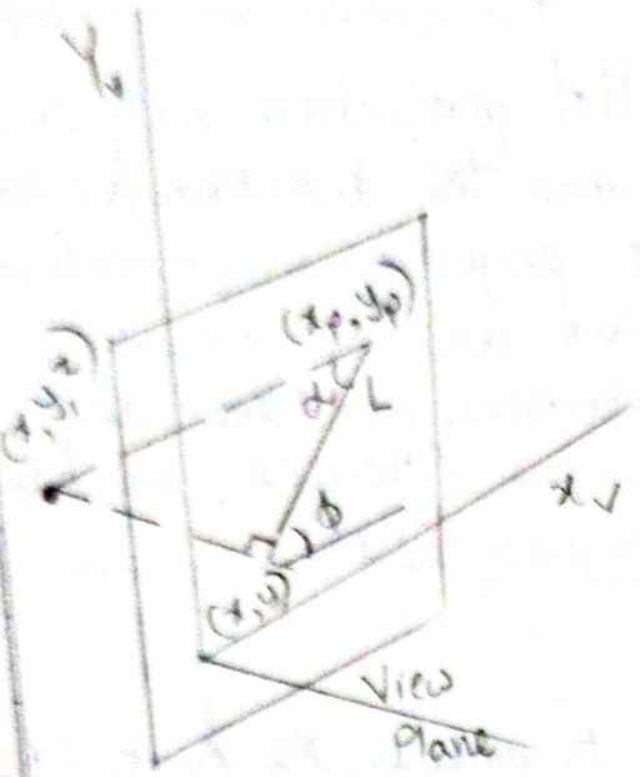
Those orthographic projections that display more than one face of an object are called axonometric orthographic projections. The most commonly used axonometric projection is the isometric projection (fig f).

We can express the projection co-ordinates of an oblique projection in terms of x, y, L and ϕ as in (fig g) as

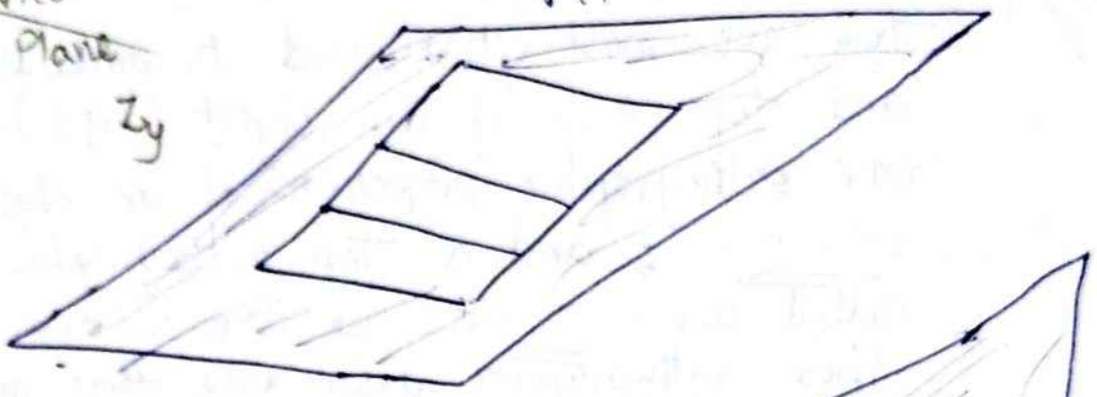
$$x_p = x + L\cos\phi$$
$$y_p = y + L\sin\phi \quad \text{---- (1)}$$

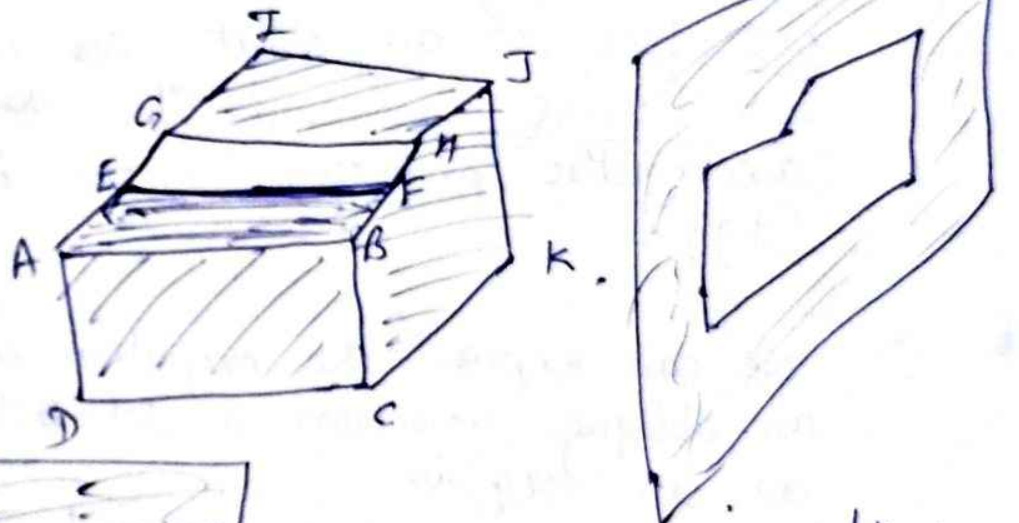
Length L depends on the angle α and the

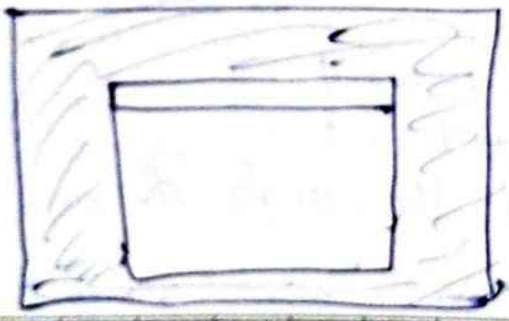
fig(g) Oblique projection of coordinate position (x, y, z) to position (x_p, y_p) on the view plane



Top View or Plan View







Front Elevation View.

Side Elevation view.

Fig: Orthographic Projection of an object displaying Plan and elevation view

z – co-ordinate of the point to be projected:

$$\tan\alpha = \frac{z}{L}$$

Thus,

$$L = \frac{z}{\tan\alpha}$$

$$= z L_1$$

where L_1 is the inverse of $\tan\alpha$, which is also the value of L when $z = 1$. We can write the oblique projection equations as

$$x_p = x + z(L_1 \cos\phi)$$

$$y_p = y + z(L_1 \sin\phi)$$

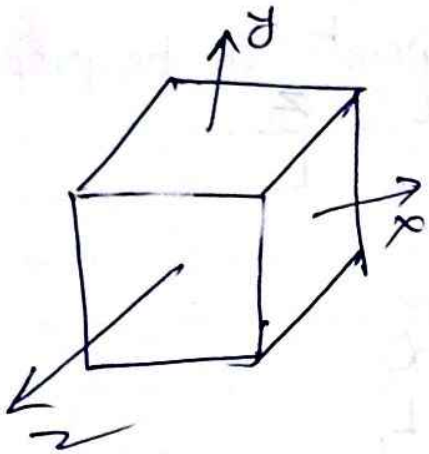
The transformation matrix for producing any parallel projection onto the $x_v y_v$ plane can be written as

$$M_{parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

An orthographic projection is obtained when $L = 0$ (which occurs at a projection angle α of 90°). Oblique projections are generated with nonzero values for L_1.

Teacher's Signature : .....

Fig (h)(i) Coordinate description



Vanishing Point

Fig (ii) One-Point Perspective Projection

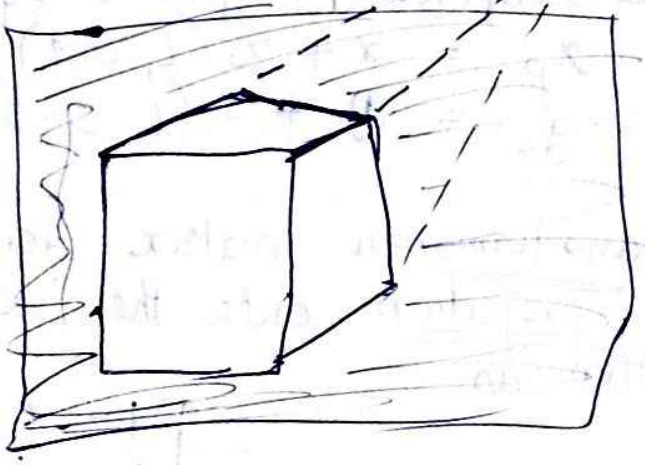
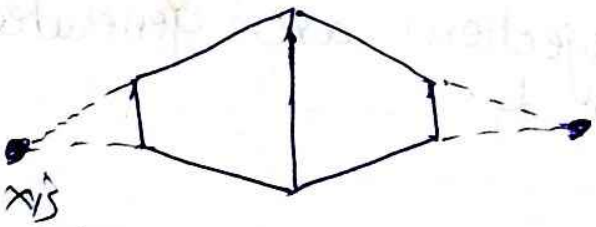


Fig h(iii) Two-Point Perspective projection



p-wub

Expt. No. ..... Page No. ..... Date .....

When a three-dimensional object is projected onto a view plane using perspective transformation equations, any set of parallel lines in the object that are not parallel to the plane are projected into converging lines. Parallel lines that are parallel to the view plane will be projected as parallel lines. The point at which a set of projected parallel lines appears to converge is called a vanishing point.

Each such set of projected parallel lines will have a separate vanishing point; and, in general, a scene can have any number of vanishing points, depending on how many sets of parallel lines there are in the scene.

The vanishing point of for the any set of lines that are parallel to one of the principal axees of an object is referred to as a principal vanishing point. We control the number of principal vanishing points (one, two or three) with the orientation of the projection plane, and perspective projections are accordingly classified as one-point, two-point, or three-point projections. The number of principal vanishing points in a projection is determined by the number of principal axees intersecting the view plane. (Fig 4 (i)) illustrates the appearance of one-

Teacher's Signature : .....

point and two-point perspective projections for a cube. In fig h(ii), the view plane is aligned parallel to the xy object plane so that only the object z axis is intersected. This orientation produces a one-point perspective projection with a z-axis's vanishing point. For the view shown in fig h(iii), the projection plane intersects both the x and z axes but not the y axis. The resulting two-point perspective projection contains both x-axis and z-axis's vanishing points.

Basic Transformations

1) Translation.
2) Rotation.
3) Scaling.

1) Translation :- A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding translation distances, t_x and t_y, to the original coordinate position (x, y) to move the point to a new position (x', y').

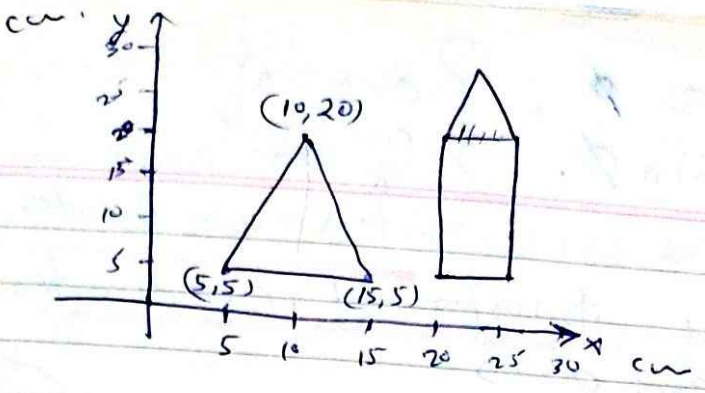
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = [P] \cdot [T]$$

$$x' = x + t_x \quad -(1)$$

$$y' = y + t_y \quad -(2)$$

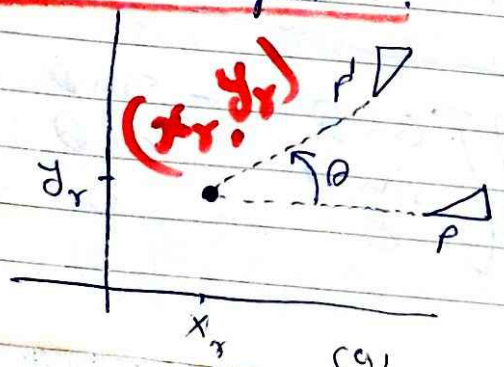
The translation distance pair (t_x, t_y) is called a translation vector or shift vector.



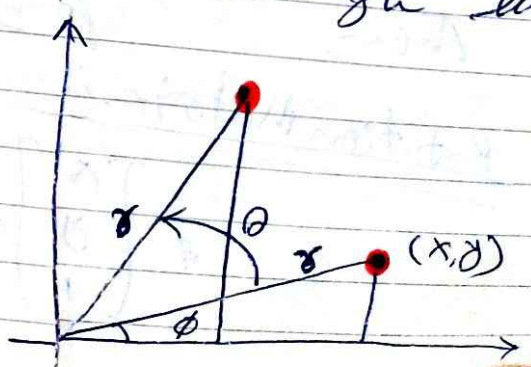
Q: (1). Translate in x-direction is 10 cm.
Q. (2) —/— y-direction is 05.
Q (3) —/— x-direction is 5 &
y-direction is 10.

(2) Rotation :- A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy-plane. To generate a rotation, we specify a rotation angle θ and the position (x_r, y_r) of the rotation point (or pivot point) about which the object is to be rotated.

+ve values for the rotation angle define counterclockwise rotations about the pivot-point and −ve values rotate objects in the clockwise direction. This transformation can also be described as a rotation about a rotation axis that is perpendicular to the xy-plane and passes through the pivot point.



(a)



(b)

$x = r \cos\phi$,
$y = r \sin\phi$.

} Original coordinate of the point in Polar coordinates. — (3) (4)

we express the transformed coordinates in terms of angles θ and ϕ as

$$x' = r \cdot \cos(\phi + \theta) \quad \text{— (5)}$$

$$x' = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta \quad \text{— (6)}$$

$$y' = r \cdot \sin(\phi + \theta) \quad \text{— (7)}$$

$$y' = r \cdot \cos\phi \cdot \sin\theta + r \cdot \sin\phi \cdot \cos\theta \quad \text{— (8)}$$

Substitute the eq. (3) & (4) into eq. (6) & (8), we obtain the transformation eq. for rotating a point at position (x, y) through an angle θ about the origin:

$$x' = x \cdot \cos\theta - y \cdot \sin\theta \quad \text{— (9)}$$

$$y' = x \cdot \sin\theta + y \cdot \cos\theta \quad \text{— (10)}$$

Then, $P' = R \cdot P$

Rotation Matrix is:

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

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\cdot\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cdot\cos\theta$$

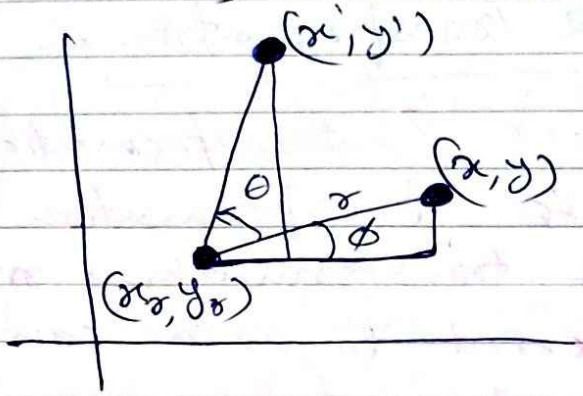


Fig:- Rotation a point from position (x,y) to position (x',y') through an angle θ about rotation point (x_r, y_r)

we can generalize eq - (9) & (10) to obtain the transformation equations for rotation of a point about any specified rotation position (x_r, y_r) :

Scaling:- A scaling transformation alerts the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x,y) of each vertex by scaling factors S_x and S_y to produce the transformed coordinates (x', y'):

$$x' = x \cdot S_x \quad \text{---(1)}$$

$$y' = y \cdot S_y \quad \text{---(2)}$$

The transformation eq. (1) & (2) can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{---(3)}$$

Q. $S_x = 2, S_y = 1$

Q. $S_x = S_y = 0.5$.

Composite Transformation :-

- Sequence of transformation is called a composite transformation. Forming products of transformations matrices is often referred to as a concatenation, or Composition of matrices.

1) **Translation:-** If two successive translation vectors (t_{x_1}, t_{y_1}) and (t_{x_2}, t_{y_2}) are applied to a coordinate position P, the final transformed location P' is calculated as

$$P' = T(t_{x_2}, t_{y_2}) \cdot \{T(t_{x_1}, t_{y_1}) \cdot P\}$$

$$= T\{(t_{x_2}, t_{y_2}) \cdot (t_{x_1}, t_{y_1})\} \cdot P$$

where, P and P' are represented as homogeneous-coordinate column vectors. The composite transformation matrix for this sequence of translation is

Q. $S_x = 2, \; S_y = 1$

Q. $S_x = S_y = 0.5$.

Composite Transformation:-

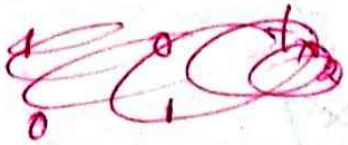
- Sequence of transformation is called a composite transformation. Forming products of transformations matrices is often referred to as a concatenation, or composition of matrices.

1) Translations:- If two successive translation vectors (t_{x_1}, t_{y_1}) and (t_{x_2}, t_{y_2}) are applied to a coordinate position P, the final transformed location P' is calculated as

$$P' = T(t_{x_2}, t_{y_2}) \cdot \{T(t_{x_1}, t_{y_1}) \cdot P\}$$

$$= T\{(t_{x_2}, t_{y_2}) \cdot (t_{x_1}, t_{y_1})\} \cdot P$$

where, P and P' are represented as homogeneous-coordinate column vectors. The composite transformation matrix for this sequence of translation is



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx_2 \\ 0 & 1 & ty_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & tx_1 \\ 0 & 1 & ty_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x_1} + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

which demonstrates that two successive translations are additive.

$$T(tx_2, ty_2) \cdot T(tx_1, ty_1) = T(t_{x_1} + tx_2, ty_1 + ty_2)$$

2) Rotations :—

Two successive rotations applied to point P produce the transformed position

$$P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$
$$= \{R(\theta_2) \cdot R(\theta_1)\} P$$

By multiplying the two rotation matrices, we can verify that two successive rotations are additive :

$$R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

$$\therefore P' = R(\theta_1 + \theta_2) \cdot P.$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

~~Rotation~~ Also)

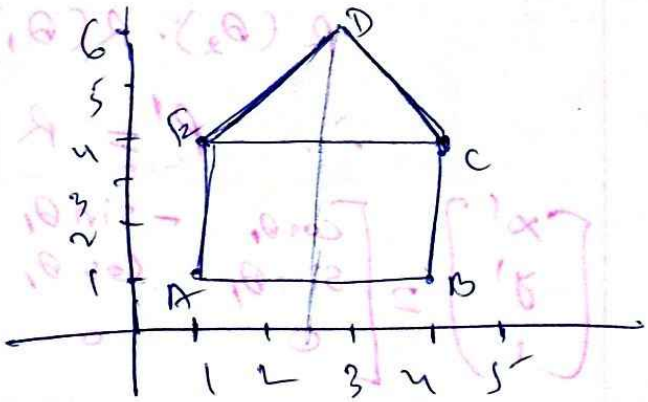
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3) **Scaling:** In this transformation two successive scaling operations produces the following composite scaling matrix:

$$\begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} (S_{x_1} \cdot S_{x_2}) & 0 & 0 \\ 0 & (S_{y_1} \cdot S_{y_2}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

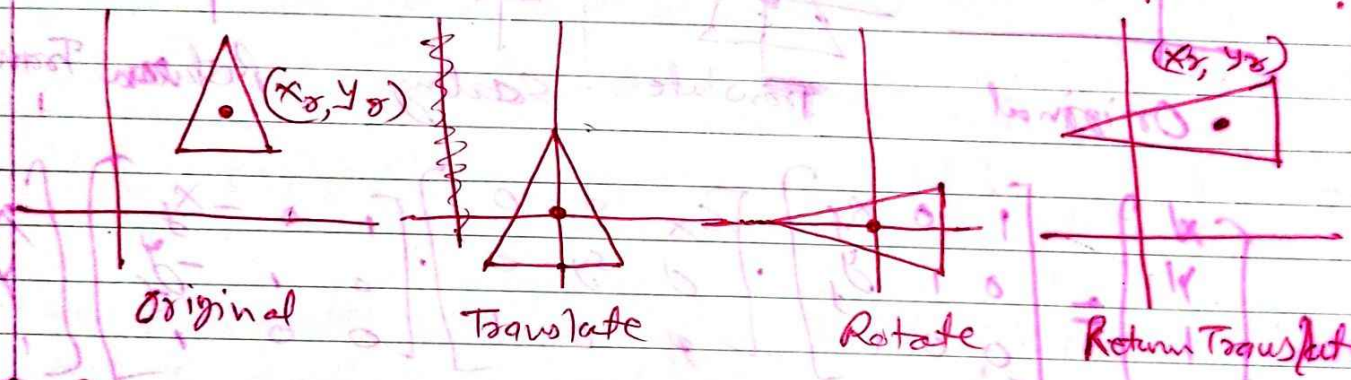
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_1} \cdot S_{x_2} & 0 & 0 \\ 0 & S_{y_1} \cdot S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



General Pivot-Point Rotation:→

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of translate-rotate-translate operations:-

1) Translate the object so that the pivot-point position is moved to the coordinate origin.
2) Rotate the object about the coordinate origin.
3) Translate the object so that the pivot point is returned to its original position.



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

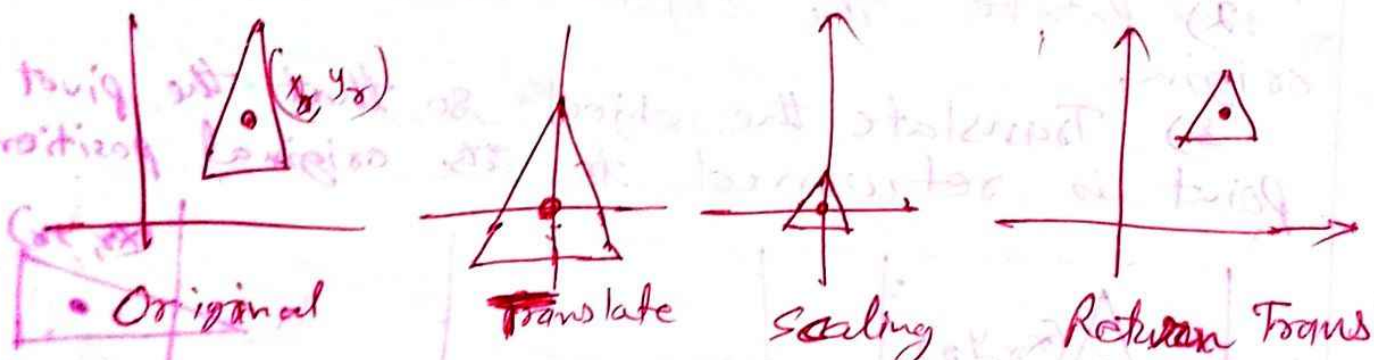
$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \cdot \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \cdot \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

General Fixed-Point scaling :-

1) Translate object so that the fixed point coincides with the coordinate origin.

2) Scale the object with respect to the coordinate origin.

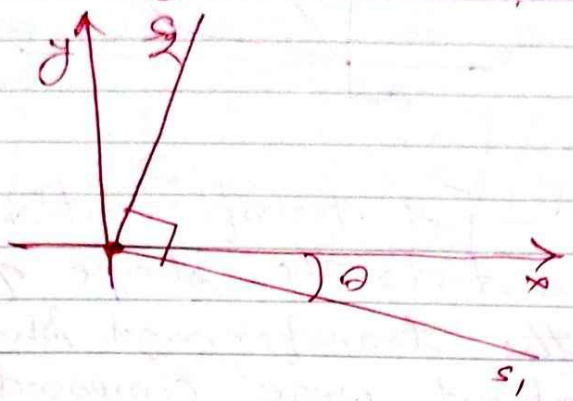
3) Use the inverse translation of Step 1 to return the object to its original position.



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

General Scaling Directions:-



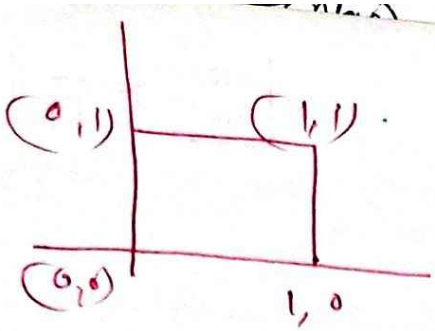
Rotate – Scaling – Rotate.

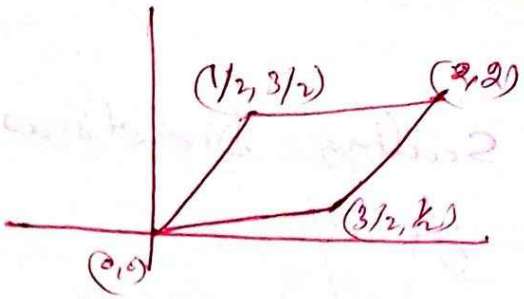
$$R^{-1}(\theta) \cdot S(s_1, s_2) \cdot R(\theta)$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 \cdot \cos^2\theta + s_2 \cdot \sin^2\theta & (s_2 - s_1) \cdot \cos\theta \cdot \sin\theta & 0 \\ (s_2 - s_1) \cos\theta \cdot \sin\theta & s_1 \cdot \sin^2\theta + s_2 \cdot \cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

→ Parameters s_x and s_y scale objects along the x and y directions. We can scale an object in other directions by rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.

Q. If we turn a unit square into a parallelogram by stretching it along the diagonal from (0,0) to (1,1). Rotate the diagonal onto the y-axis and double its length with the transformation parameters $\theta = 45°$, $s_1 = 1$, $s_2 = 2$.

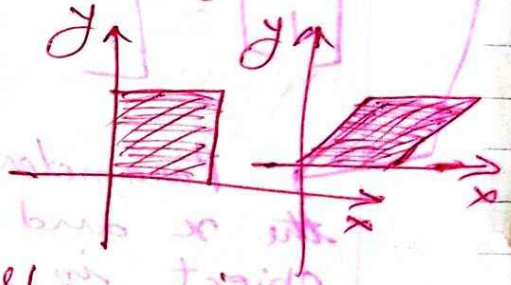




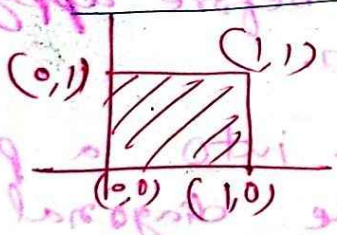
Shear :- [A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a "Shear".] Two common shearing transformations are those that shift coordinate x-values and those that shift y values.

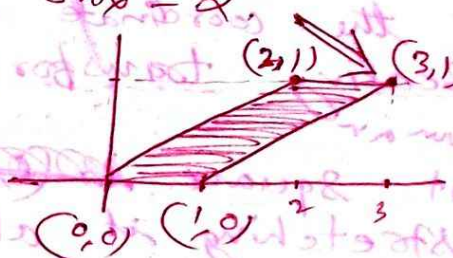
(1) An x-direction shear relative to the x-axis is produced with the transformation matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



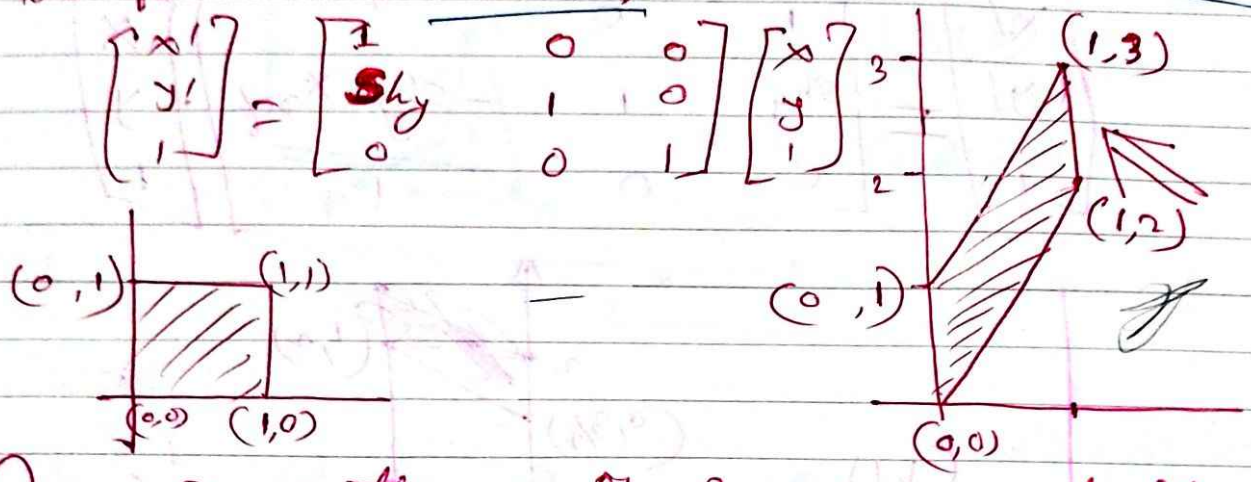
(2) Draw the unit square and shear in x-direction, if $sh_x = 2$.





(2) A- y-direction shear relative to the ~~line~~ x y-axis is produced with the transformation matrix.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

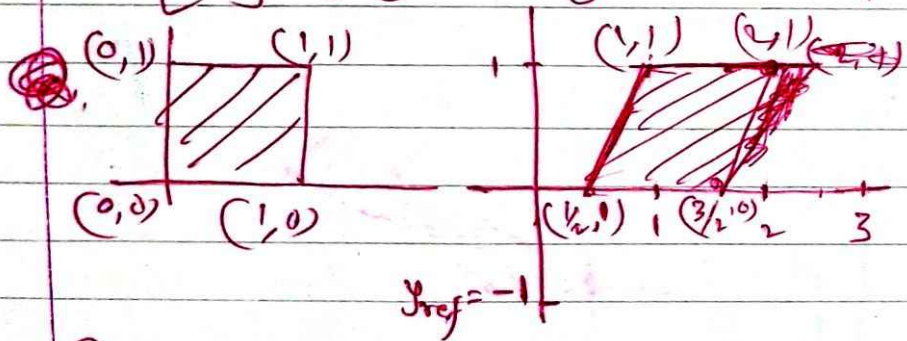


Q. Draw the unit square and shear in y-direction if $Sh_y = 2$.

(3) Shear & shifting :-

x-direction shears relative to other reference lines with the matrix are :-

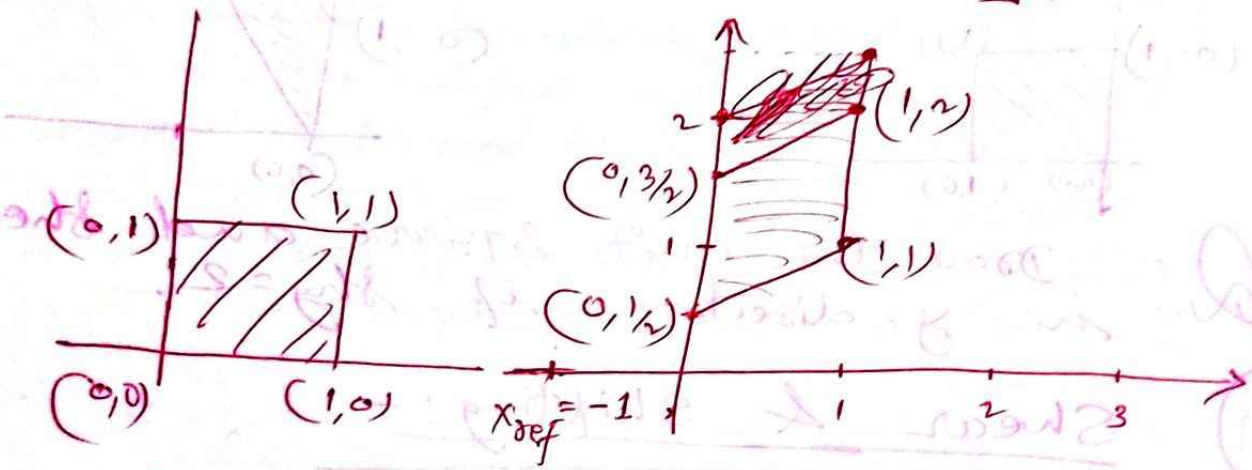
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & -Sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Q. In Unit Square, $Sh_x = 1/2$ and $y_{ref} = -1$

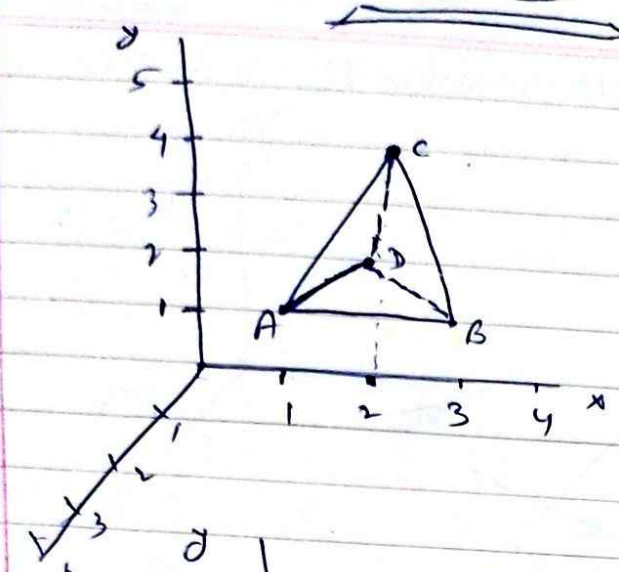
y - direction shears relative to other reference lines with the matrix are :-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

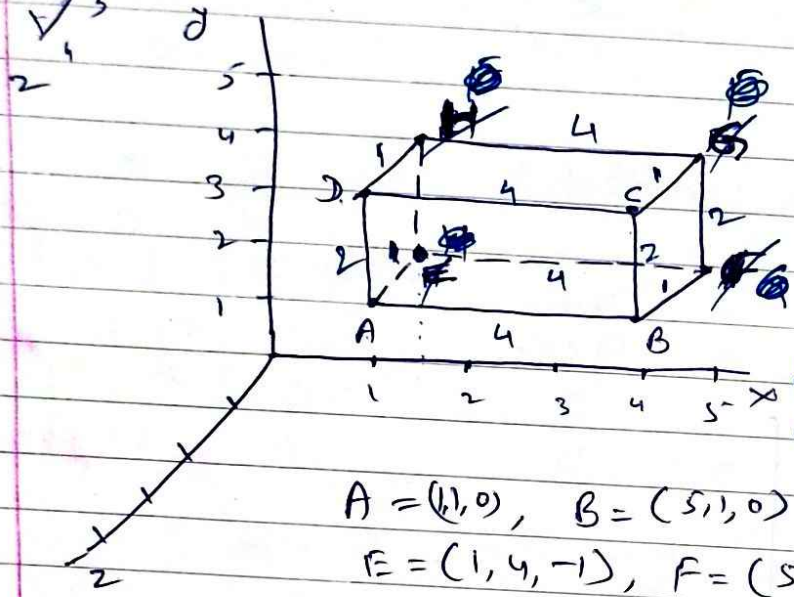


Q. if $sh_y = 1/2$, $x_{ref} = -1$ in the y-direction (Shear & Shift).

3-D



A = (1, 1, 0)
B = (3, 1, 0)
C = (2, 4, 0)
D = (2, 1, -1)



A = (1, 1, 0) ✓
B = (4, 1, 0)
C = (4, 3, 0)
D = (1, 3, 0)
E = (1, 1, -1)
F = (4, 1, -1)
G = (4, 3, -1)
H = (1, 3, -1)

A = (1, 1, 0), B = (5, 1, 0), C = (5, 4, 0), D = (1, 4, 0),
E = (1, 1, -1), F = (5, 1, -1), G = (5, 4, -1), H = (1, 4, -1)

Translation :-

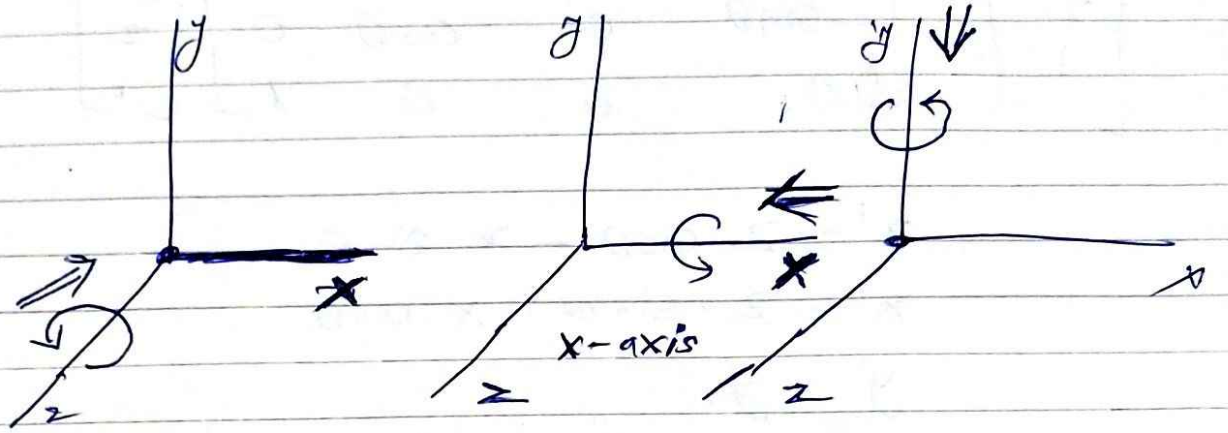
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = x + t_x$.
$y' = y + t_y$.
$z' = z + t_z$.

Rotation in 3-D.

~~X-axis~~

~~Z-axis rotation eq:-~~



Z-axis Rotation:-

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

$x' = x \cdot \cos\theta - y \cdot \sin\theta$

$y' = x \cdot \sin\theta + y \cdot \cos\theta$

$z' = z$

X-axis Rotation:-

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

$y' = y \cdot \cos\theta - z \cdot \sin\theta$

$z' = y \cdot \sin\theta + z \cdot \cos\theta$

$x' = x.$

Y - axis Rotation:

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

$$z' = z \cdot \cos\theta - x \cdot \sin\theta$$
$$x' = z \cdot \sin\theta + x \cdot \cos\theta.$$
$$y' = y.$$

3-D. rotation

1) Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2) Perform the specified rotation about that axis.
3) Translate the object so that the rotation axis is moved back to its original position.

Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

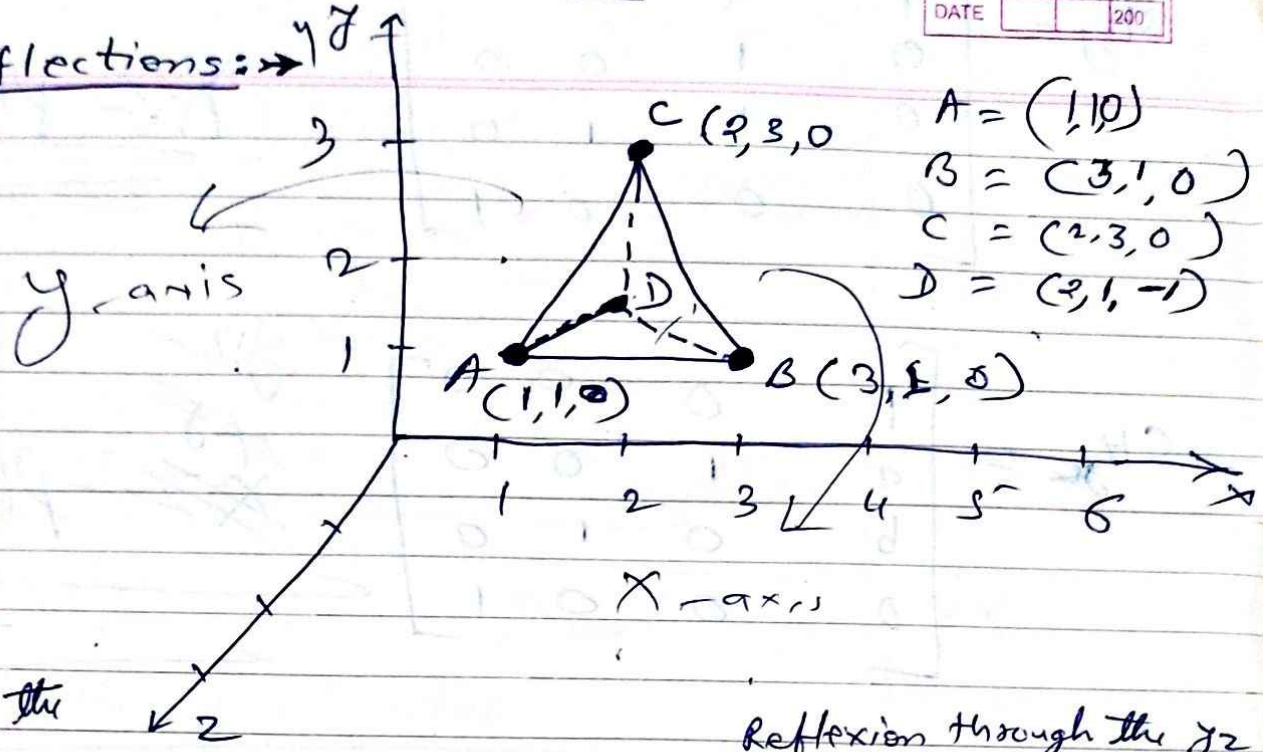
Procedure

1. Translate the fixed point to the origin.
2. Scale the object relative to the coordinate origin.
3. Translate the fixed point back to its original position.

Q.

Other Transformations :→

Reflections :→



A = (1,1,0)
B = (3,1,0)
C = (2,3,0)
D = (2,1,−1)

Reflexion through the xy plane is :-

$$[T]_{xy} \quad Rf_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflexion through the yz plane is :-

$$[T]_{yz} \quad Rf_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rf_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflexion through the xz plane is

Shear :

z-axis shear.

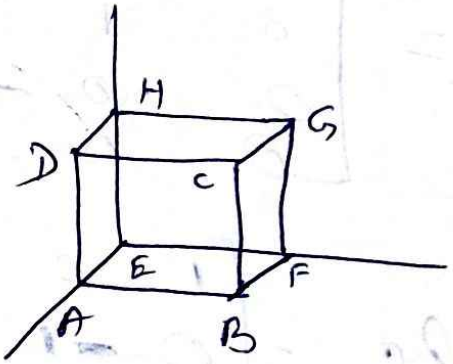
$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

xy-plane

$$SH_y = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ ← X-Z-plane

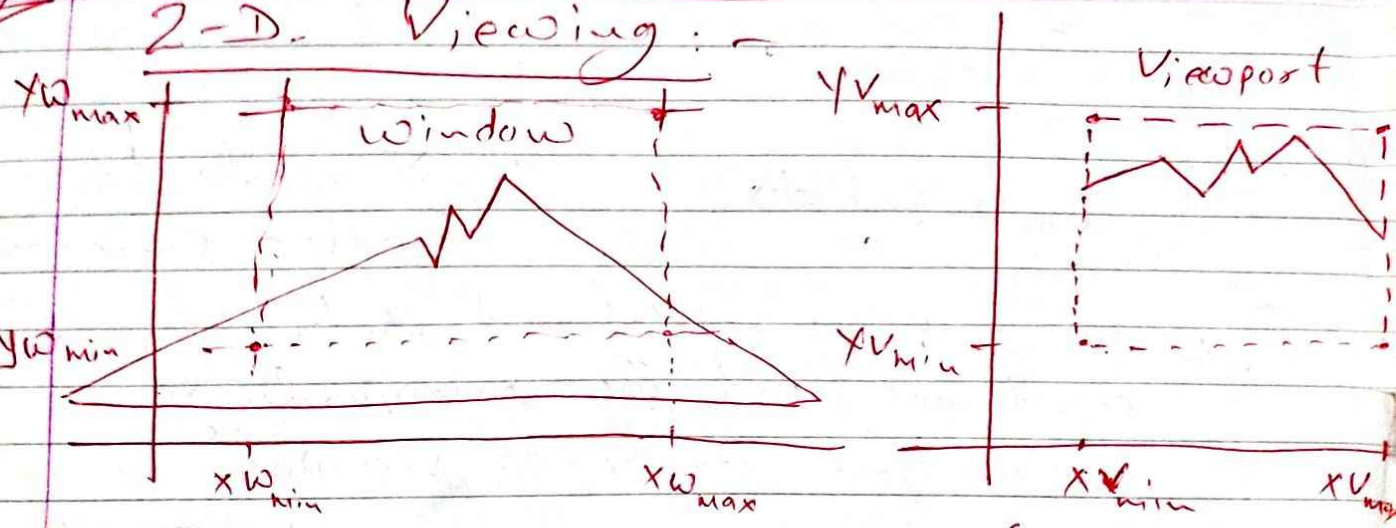
$$SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ yz ~~xy~~ ~~xz~~ -plane

Q. Perform reflection of Unit cube about the xy-plane.



Unit. - 3

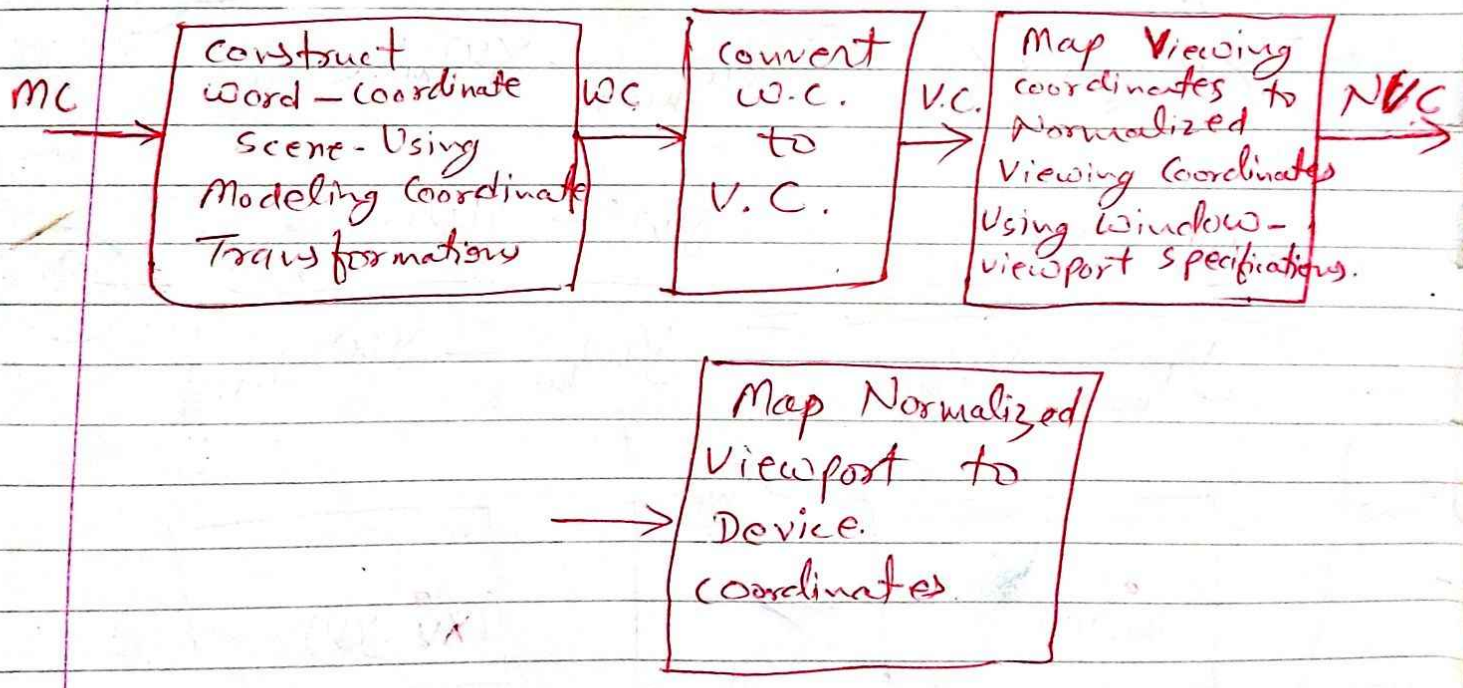
2-D. Viewing :-



World Coordinates.

Device Coordinate

2-D. viewing - transformation pipeline :-



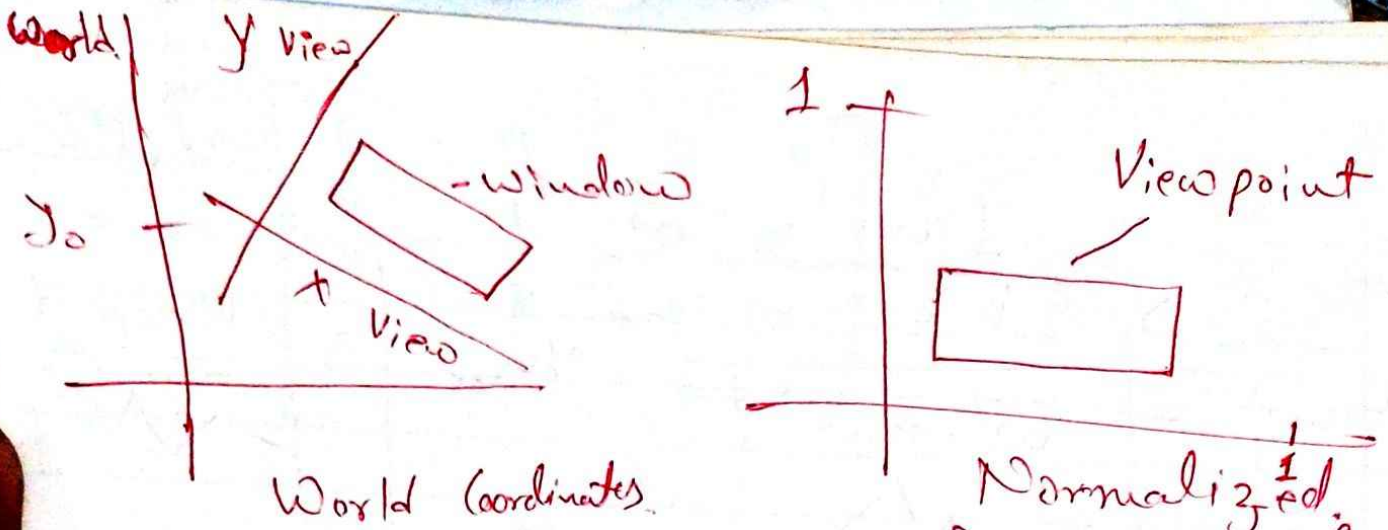
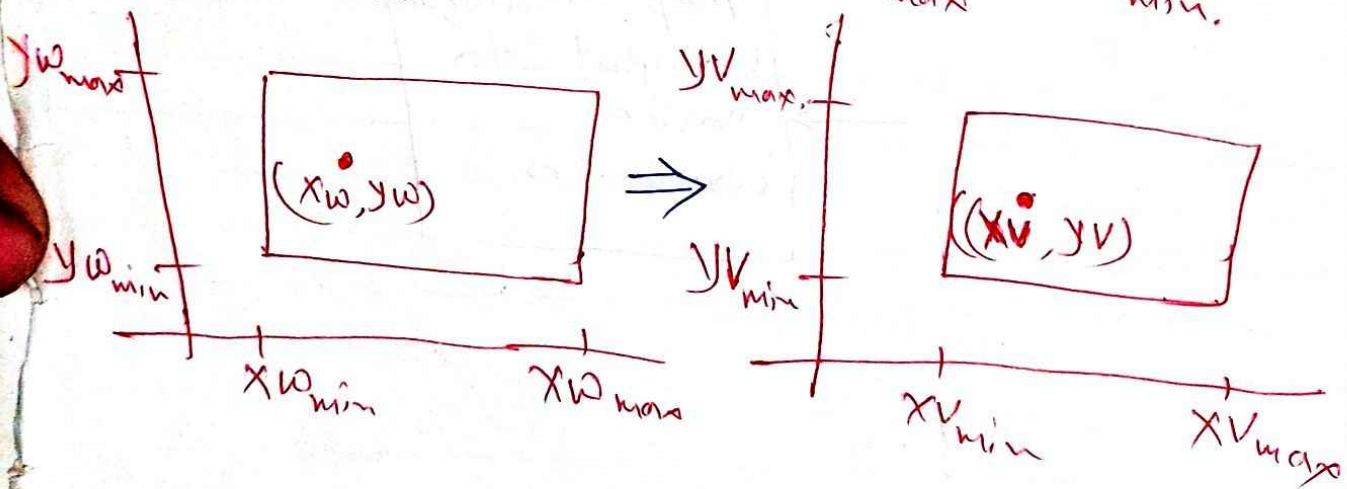


Fig:- Setting up a rotated word window in viewing coordinates and the corresponding normalized-coordinate viewport.

Window-To-Viewport coordinate Transformation:

$$\frac{XV - XV_{min}}{XV_{max} - XV_{min}} = \frac{XW - XW_{min}}{XW_{max} - XW_{min}} \quad \text{---(1)}$$

$$\frac{YV - YV_{min}}{YV_{max} - YV_{min}} = \frac{YW - YW_{min}}{YW_{max} - YW_{min}} \quad \text{---(2)}$$



Solving these expressions for the viewport position (xv, yv), we have.

$$xv = xv_{min} + (xw - xw_{min}) s_x \quad \text{---(3)}$$

$$yv = yv_{min} + (yw - yw_{min}) s_y. \quad \text{---(4)}$$

where the scaling factors are

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}. \quad \text{---(5)}$$

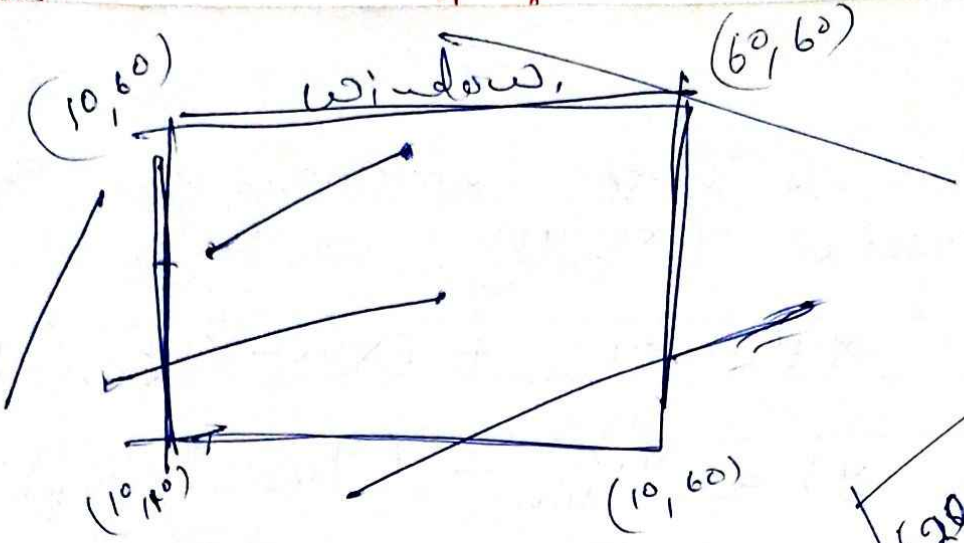
$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}. \quad \text{---(6)}$$

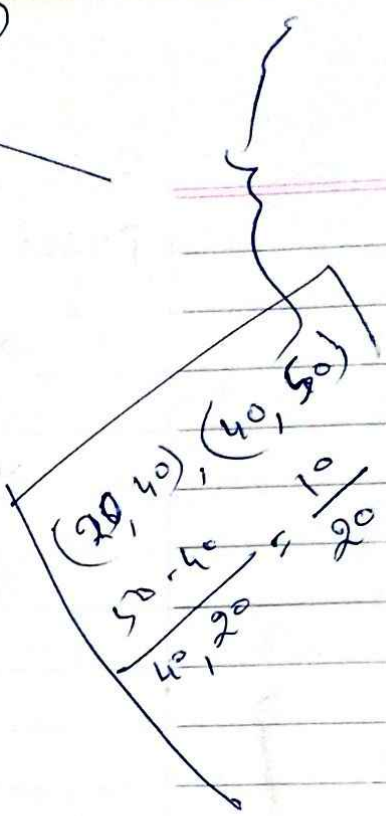
Clipping Operations:-

$P = (x, y)$.

$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}.$$

Line - clipping :-





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

$$y = y_1 + u(y_2 - y_1)$$

$$\{\because 0 \le u \le 1\}$$

$$\therefore u = \frac{y_2 - y_1}{x_2 - x_1} \qquad \because u = \text{intersection}$$

(1) $\because$ u ~~>~~ 0 to 1

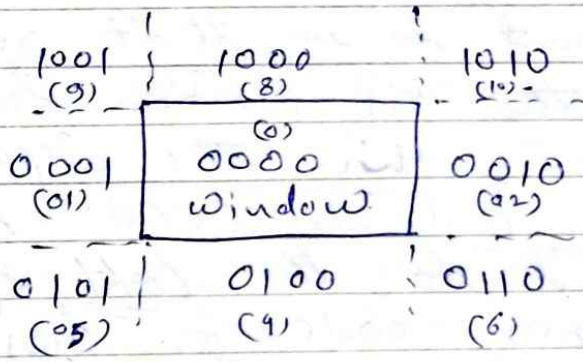
$\therefore$ outside the window boundary.

(2) $\because u \Rightarrow 0 \le u \le 1$

$\therefore$ The line segment does indeed cross into the clipping area.

If the value of "u" for an intersection with a rectangle boundary edge is outside the range 0 to 1, the line does not enter the interior of the window at that boundary. If the value of u is within the range from 0 to 1, the line segment does indeed cross into the clipping area.

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 tests that reduce the number of intersections that must be calculated. Every line endpoint in a picture is assigned a four-digit boundary code, called a "Region code", that identifies the location of the point relative to the boundaries of the clipping rectangle. Regions are set up in reference to the boundaries. Each

In figure, 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.

bit 1 : Left $\{ x - xw_{min} \}$
bit 2 : Right $\{ xw_{max} - x \}$
bit 3 : Below $\{ y - yw_{min} \}$
bit 4 : Above $\{ yw_{max} - y \}$

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.

Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries.

1) Bit 1 is set to 1 if $x < xw_{min}$.
2) Bit 2 is set to 1 if $xw_{max} > x$
3) Bit 3 is set to 1 if $y < yw_{min}$
4) Bit 4 is set to 1 if $yw_{max} > y$.

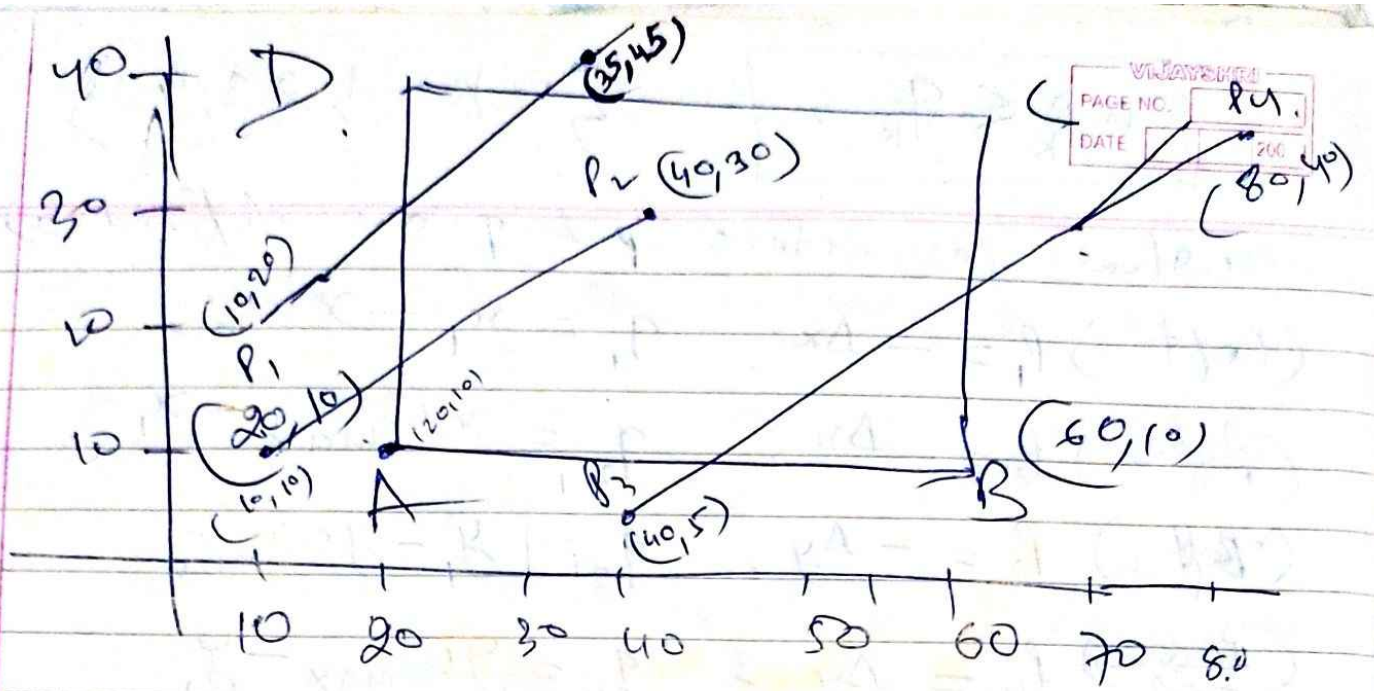
1) $\boxed{y = y_1 + m(x - x_1)}$

$\therefore x$ is set either xw_{min} or xw_{max}

$\therefore m = \frac{y_2 - y_1}{x_2 - x_1}$

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

$\therefore y$ is set either yw_{min} or to yw_{max}.



Liang-Barsky Line clipping:-

Faster line clippers have been developed that are based on analysis of the parametric eq. of a line segment, which we can write in the form

$$x = x_1 + u\,\Delta x, \quad \{\because \Delta x = x_2 - x_1\}$$
$$y = y_1 + u\,\Delta y, \quad \{\because \Delta y = y_2 - y_1\}$$
$$\therefore 0 \le u \le 1$$

$\because u = \frac{y_2 - y_1}{x_2 - x_1}$

This clipping algorithm is generally more efficient than the Cohen-Sutherland algo. Following the Liang-Barsky approach we first write the point-clipping conditions in the parametric form:

$$x w_{min} \le x_1 + u\,\Delta x \le x w_{max}$$
$$y w_{min} \le y_1 + u\,\Delta y \le y w_{max}$$

Each of these four inequalities can be expressed as

$$u p_k \leq q_k, \quad \{ \forall : k = 1, 2, 3, 4 \}$$

where parameters p & q are defined as

(Left) $p_1 = -\Delta x$, $\quad q_1 = x_1 - x w_{min}$

(Right) $p_2 = \Delta x$, $\quad q_2 = x w_{max} - x_1$

(Bottom) $p_3 = -\Delta y$, $\quad q_3 = y_1 - y w_{min}$

(~~Bottom~~ Top) $p_4 = \Delta y$, $\quad q_4 = y w_{max} - y_1$

Any line that is parallel to one of the clipping boundaries has $p_k = 0$ for the value of k corresponding to that boundary $k = 1, 2, 3, 4$ corresponds to the Left, Right, Bottom and Top boundaries respectively.)

1) If for that value of k, we also find $q_k < 0$, then line is completely outside the boundary and can be eliminated from ~~from~~ further consideration.

2) If $q_k \geq 0$, the line is inside the parallel clipping boundary.

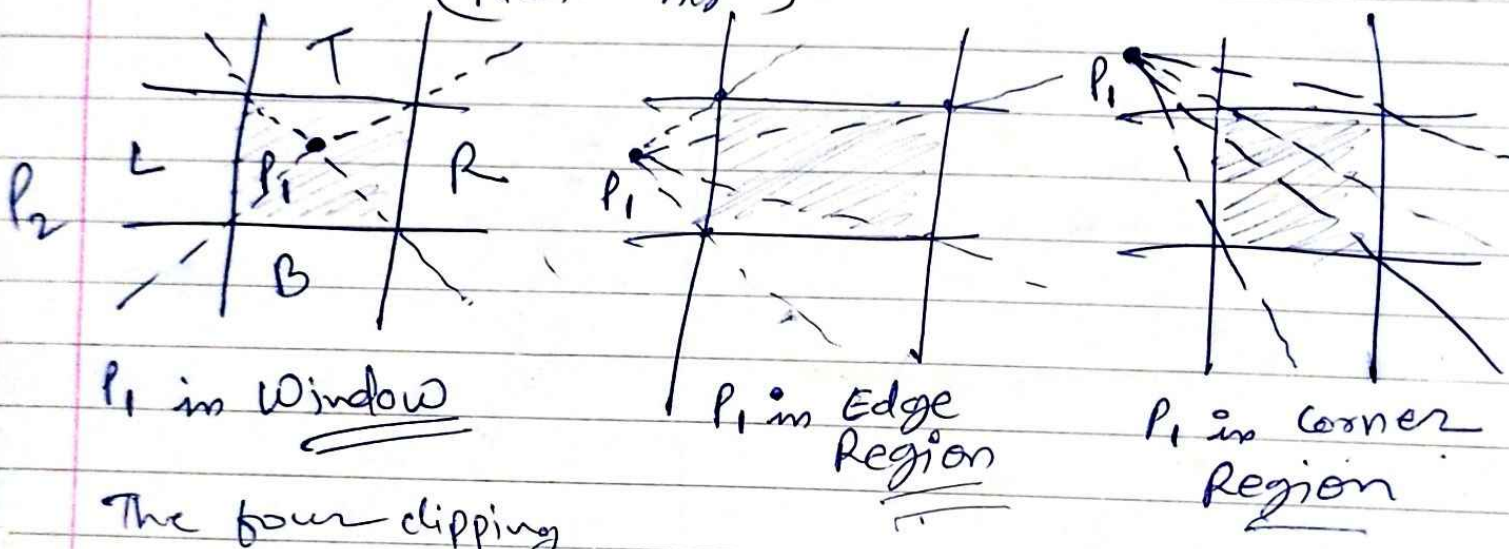
3) When $p_k < 0$, ~~then~~ the infinite extension of the line proceeds from the outside to the inside of the infinite extension of this particular clipping boundary.

4) If $p_k > 0$, the line proceeds from the inside to the outside.

5) For a nonzero value of p_k, we can calculate the value of u that corresponds to the point where the infinitely extended line intersects the extensions of boundary k as

$$u = q_k / p_k$$

Nicholl - Lee - Nicholl line clipping: → (NLN - Algo.)

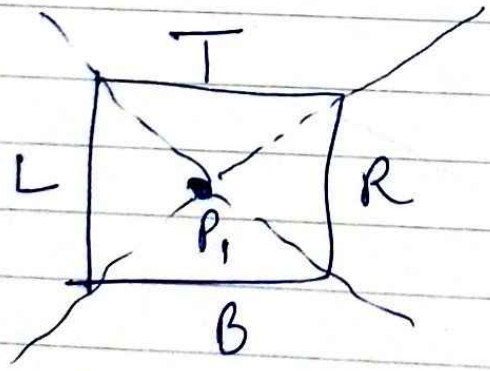


P_1 in Window

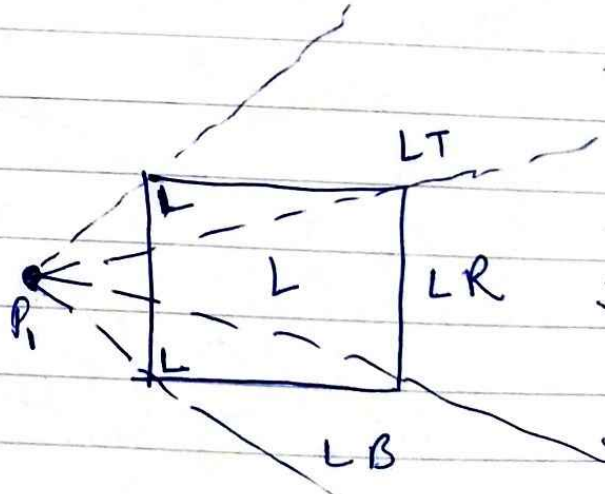
P_1 in Edge Region

P_1 in corner Region

The four clipping regions used in the NLN algo. when P_1 is inside the clip window and P_2 is outside.



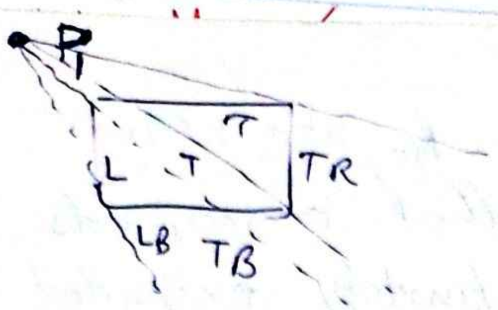
four regions L, T, R, B. (P_1 is inside)

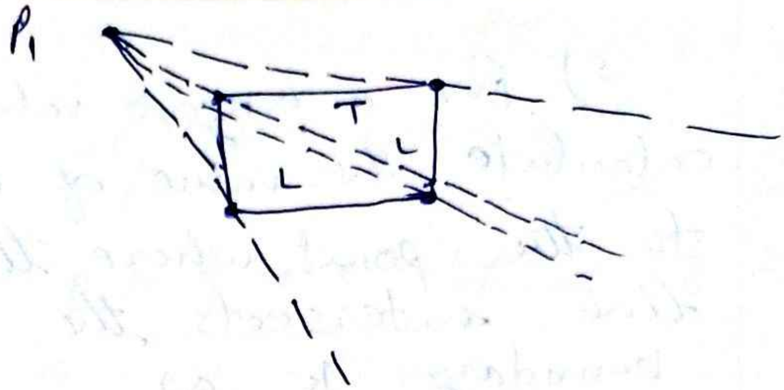


Four regions. L, LT, LR, LB

The four clipping regions used in the NLN algo. when P_1 is inside the clip window.

= Unique Boundary





P1 is above and to the left of the clip window.

Remain of Unit-III are one

[illegible] ...t Page after (11) pages