FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

MODULE 1

MODULE 2

33. Demonstrate Reflection of an object w.r.t the straight line y=x → 79
34. Explain the reflection and shearing. → 80
35. Explain with example ,vector method for splitting a polygon → 84
36. Describe OpenGL polygon fill area function with example → 87
37. Write a note on
    a. fill style b. color blended fill region → 89
38. Write a OpenGL program to rotate a triangle using composite matrix calculation. → 91
39. What are homogeneous coordinates? Write the matrix representation for translation, rotation and scaling. → 92
40. What is raster operation? Explain the raster methods for geometric transformation. → 94
41. Write a note on
    a. OpenGL fillpattern function.
    b. OpenGL texture and interpolation pattern.
    c. OpenGL wire frame methods.
    d. OpenGL front face function. → 96
42. Explain the composite 2D translation, Rotation and scaling. → 98
43. Explain the 2D OpenGL geometric transformations. → 100
44. Write the steps for rotation about pivot point and scaling about fixed point. → 102
45. Briefly explain Inverse transformation, composite transformation. → 104
46. Explain the OpenGL matrix operations and Matrix stacks. → 108
47. Explain the OpenGL 2D viewing functions. → 109
48. Translate a square with the following coordinate by 2units in both directions → 111
    A(0,0),B(2,0),C(2,2),D(0,2)
49. Rotate a triangle at A(0,0),B(6,0),C(3,3) by 90degrre about origin and fixed point (3,3) both → 112
    Anticlockwise and clockwise direction.
50. What are the polygon classifications? How to identify a convex polygon? Illustrate how to split a Concave polygon. → 115
51. What is stitching effect? How does OpenGL deals with it. → 117

MODULE 3

52. Imagine a 3 D cube object with rotation axis projected onto the Z-axis defined by the vector u. Rotate it and find the final rotation matrix R. Show all the 5 steps involved in it with 7 series of operations. → 119
53. Demonstrate the 3D Translation and Reflection with Homogenous coordinates. → 122
54. Demonstrate the 3D Scaling and Shearing with Homogenous coordinates. → 124
55. Explain the ambient light, diffuse reflection and specular reflection with equations. → 126
56. Explain OpenGL 3D Viewing Functions. → 129
57. Imagine you have a 3D object in front of you. Illustrate how to Normalize the transformation for an Orthogonal Projection? → 131
58. What is clipping and clipping window. → 133
59. Map the clipping window into a normalized viewport. → 138
60. Explain specular refection. → 142
61. Explain the 3D coordinate axis-Rotation → 143
62. Map the clipping window into a Normalized square. → 145
63. Explain the Cohen-Sutherland line-clipping algorithm. → 149
64. With neat diagram, illustrate Sutherland-Hodgeman polygon clipping algorithm. → 152
65. What is quaternion? Explain the quaternion methods for 3D rotations. → 156

Shankar R
Asst Professor,
CSE, BMSIT&M

(26) With neat diagram, explain the two commonly used algorithms for identifying interior areas of a plane figure.

⇒ There are two types of test

i) Crossing (or) odd-even test.

ii) Winding number test.

1) Odd-Even test:

This test is most widely used for making inside-outside decisions.

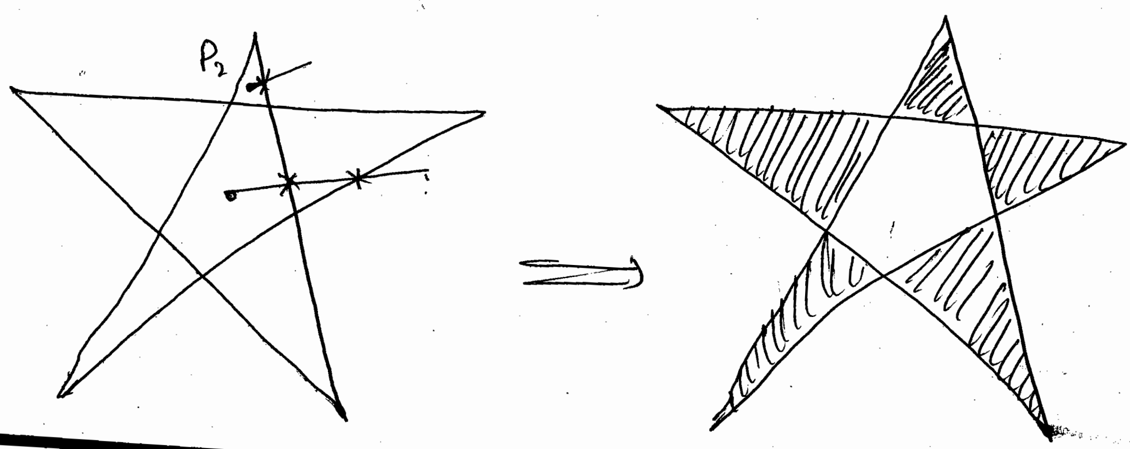
→ Drawing a line from any position 'P' to a distant point outside the co-ordinate extents of the closed polyline.

→ Count the number of line-segment crossings along this line.

→ If the number of segments crossed by this line is odd - P is considered to be interior point

→ Otherwise P is exterior.

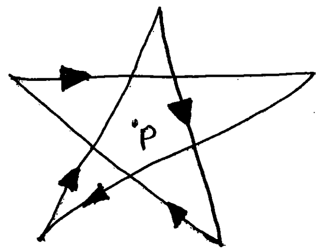
Ex:



Shankar R
Asst Professor,
CSE, BMSIT&M

Here, P_1 crosses 2 edges, hence outside or exterior.
P_2 crosses 1 edge hence interior / inside.

2. Winding number test [Non-Zero Winding number rule]
This test fills the complete star rather than in previous test.
→ To implement this test, we consider traversing the edges of the polygon from any starting vertex and going around the edge in a particular direction (any direction) until we reach the starting point.
→ We illustrate the path by labelling the edges, as shown in the below fig.



▶ → labeling the edges.

→ Consider an arbitrary point. Initial the winding number is set to zero.
→ Winding number, which counts the number of times the boundary of an object "winds" around a particular point in counter clockwise direction.

Shankar R
Asst Professor,
CSE, BMSIT&M

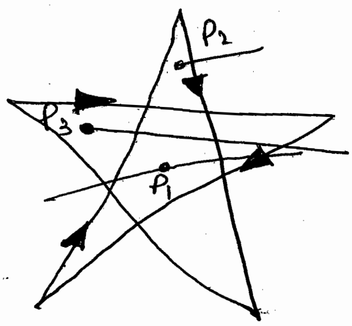
→ Count clockwise as +ve or add 1 to windowing number when it intersects a segment that crosses the line in clockwise direction.

→ Count counter anti-clockwise as −1 negative.

→ If windowing number is non-zero, P is interior.
If windowing number is zero, P is exterior.

→ & All points must cross edges not vertices.

Ex 1:



⇒ P_1 crosses 2 edges, 1st is from right to left.

$P_1 = +1 + 1 = 2$. (inside)

from vertex P_1 to left

$P_1 = -1 - 1 = -2$ (inside) from vertex P_1 to right.

⇒ P_2 = crosses 1 edge from right to left.

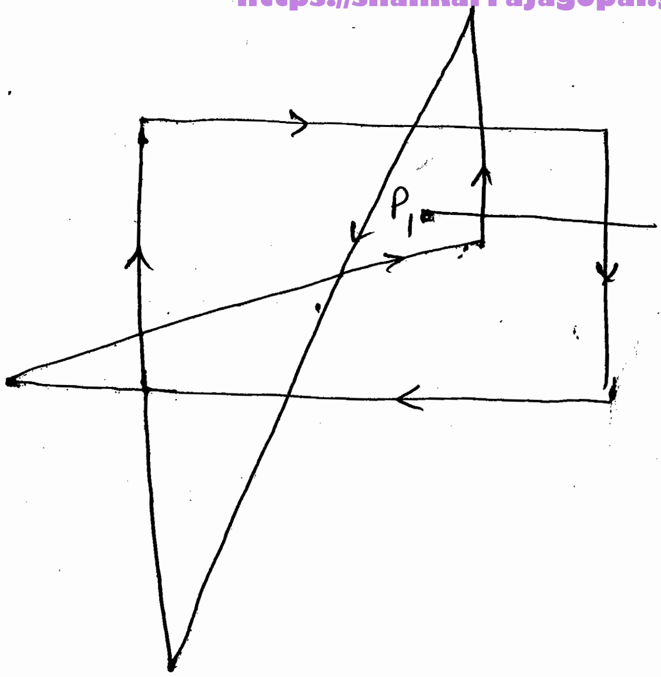
$P_2 = +1$ (inside)

⇒ P_3 = crosses 3 edges.

$= -1 + 1 + 1$

$P_3 = 1$ (inside)

Ex 2.

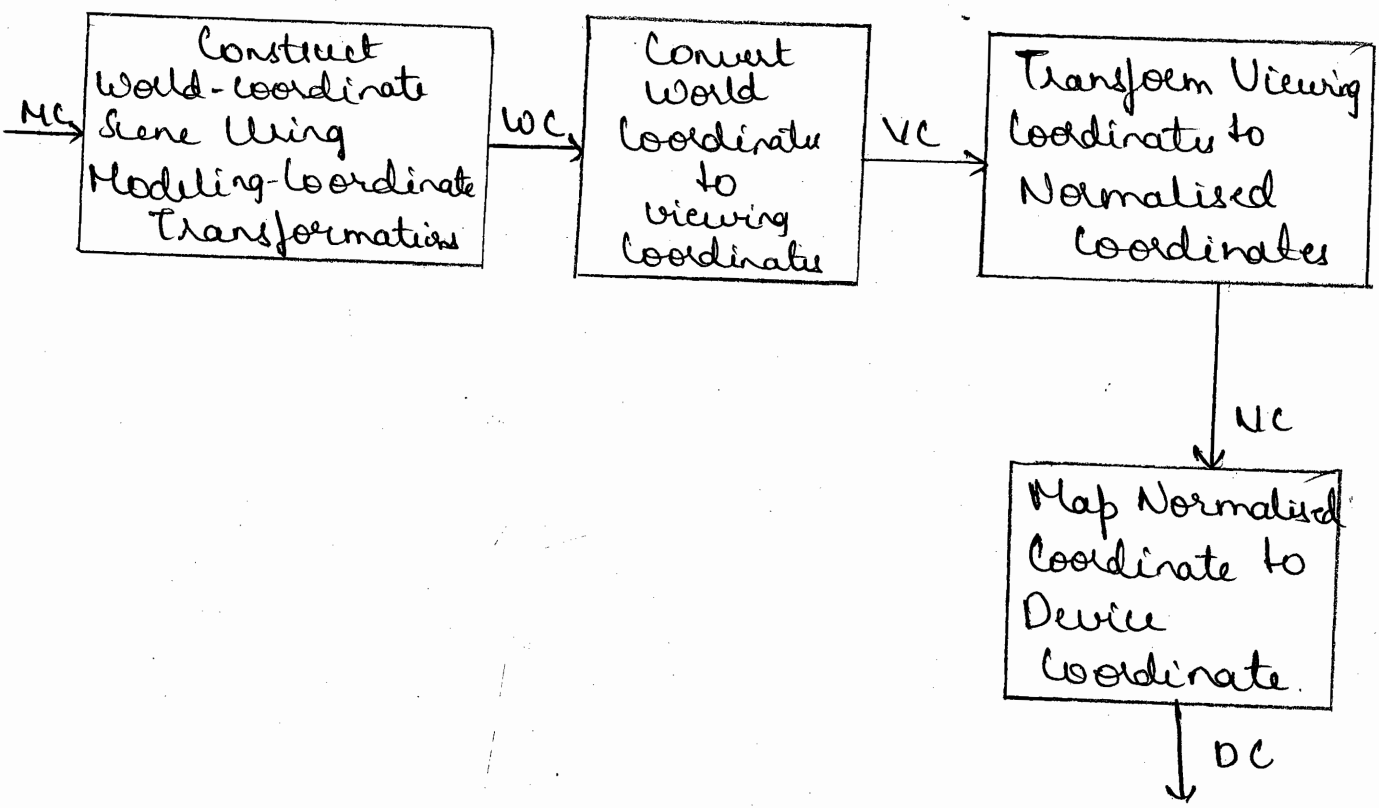


$P_1 = -1+1$

$= 0$ (outside)

(27) Explain two dimentional viewing transforma-tion pipeline.

⟹



Shankar R
Asst Professor,
CSE, BMSIT&M

A section of a two-dimensional scene that is selected for display is called a clipping window because all parts of the scene outside the selected section are "clipped" off.

The mapping of a two-dimensional, world co-ordinate scene description to device co-ordinates is called a two-dimensional viewing transformation. Sometimes this transformation is simply referred to as the window to viewport transformation or window transformation.

Once the world-coordinate scene has been constructed we could set up a seperate 2D viewing coordinate reference frame for specifying the clipping window. Viewing coordinates for 2D applications are the same as world coordinates.

To make the viewing process independent of the requirements of any output device, graphics systems convert object descriptions to normalized coordinates in the range from 0 to 1, and others use range from -1 to 1. Depending upon the graphics library in use, the viewport is defined either in normalised

Shankar R
Asst Professor,
CSE, BMSIT&M

co-ordinates or in screen coordinates after the normalization process. At the final step of the viewing transformation, the contents of the viewport are transferred to positions within the display window.

Shankar R
Asst Professor,
CSE, BMSIT&M

28. Show that successive scaling is multiplicative.

⇒ To alter the size of an object, we apply a scaling transformation. A simple two-dimensional scaling operation is performed by multiplying object positions (x, y) by scaling factors s_x and s_y to produce the transformed coordinates (x', y').

$$x' = x \cdot s_x \quad , \quad y' = y \cdot s_y$$

Scaling factor s_x, scales an object in the x direction while s_y scales in y direction.

Basic two-dimensional scaling equations are written as:

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

or

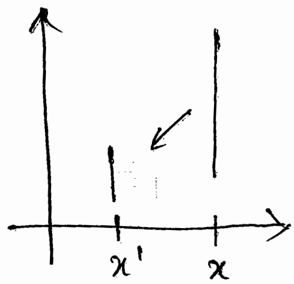
$$P' = S \cdot P$$

where S is 2×2 scaling matrix.

* Any positive values can be assigned to scaling factors s_x and s_y.
* Values less than 1 reduce the size of objects.
* Values greater than 1 produce enlargements.
* Specifying a value of 1 for both s_x and s_y leaves the size of objects unchanged.
* When s_x and s_y are assigned same value → uniform scaling

* Unequal values for s_x and s_y → differential scaling.
* In some systems, negative values can also be specified for the scaling parameters. This not only resizes an object, it reflects it about one or more of co-ordinate axes.

Figure below illustrates scaling of a line by assigning value 0.5 to both s_x and s_y



* Fixed point: controlling the location of a scaled object by choosing a position.

For a coordinate position (x, y), scaled coordinates (x', y') are calculated from following relationship:

$$x' - x_f = (x - x_f) s_x \quad , \quad y' - y_f = (y - y_f) s_y$$

We can rewrite equations as:

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y)$$

x_f and y_f are the fixed points

where additive terms $x_f(1-s_x)$ & $y_f(1-s_y)$ are constants for all points in the object.

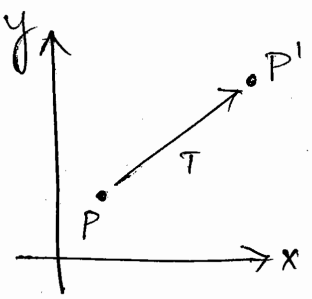
Shankar R
Asst Professor,
CSE, BMSIT&M

29. Show that successive translations are additive.

⇒ We perform a translation on a single co-ordinate point by adding offsets to its coordinates so as to generate a new coordinate position.

We are moving the original position along a straight-line path to its new location.

* To translate a two-dimensional position, we add translation distances t_x and t_y to the original co-ordinates (x, y) to obtain the new coordinate position (x', y') as shown below:



* translation values of x' and y' is calculated as

$$x' = x + t_x \quad , \quad y' = y + t_y$$

* translation distance pair (t_x, t_y) is called a translation vector (or) shift vector. column vector representation is given as:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

* This allows us to write the two dimensional translation equations in the matrix form:

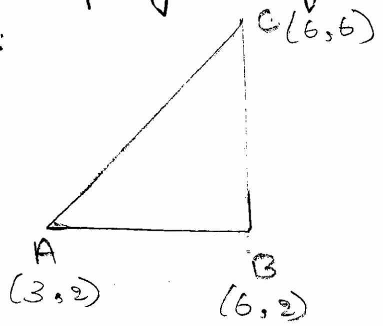
$$\boxed{P' = P + T}$$

* Translation is a rigid-body transformation that moves objects without deformation.

Shankar R
Asst Professor,
CSE, BMSIT&M

30) Design a polygon ABC – A(3, 2), B(6, 2) & C(6, 6) rotate in anticlockwise direction by 30° by keeping C fixed.

Ans.



$$P' = T(x,y) * R(\theta) * T(-x,-y) * P(x,y)$$

$$x = 6 \qquad \theta = 30^\circ$$
$$y = 6$$

$$P' = T(6,6) * R(30^\circ) * T(-6,-6) * P(x,y)$$

$$P' = \begin{bmatrix} 1 & 0 & 6 \\ 0 & 1 & 6 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & -6 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0 & 6 \\ 0 & 1 & 6 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.86 & -0.5 & 0 \\ 0.5 & 0.86 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & -6 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.86 & -0.5 & 6 \\ 0.5 & 0.86 & 6 \\ 0 & 0 & 6 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & -6 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

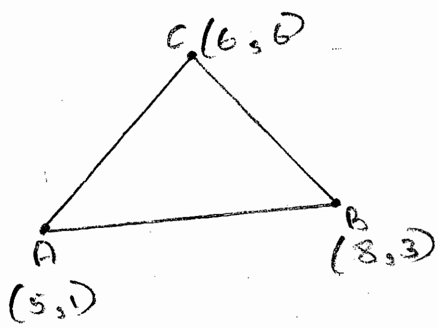
$$P' = \begin{bmatrix} 0.86 & -0.5 & 3.84 \\ 0.5 & 0.86 & -2.16 \\ 0 & 0 & 6 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$A : (3,2) \Rightarrow \begin{bmatrix} 0.86 & -0.5 & 3.84 \\ 0.5 & 0.86 & -2.16 \\ 0 & 0 & 6 \end{bmatrix} * \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.42 \\ 1.06 \\ 6 \end{bmatrix}$$

$$B : (6,2) \Rightarrow \begin{bmatrix} 0.86 & -0.5 & 3.84 \\ 0.5 & 0.86 & -2.16 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 3 \\ 6 \end{bmatrix}$$

$\therefore \quad A = (5, 1)$

$\quad\;\; B = (8, 3)$



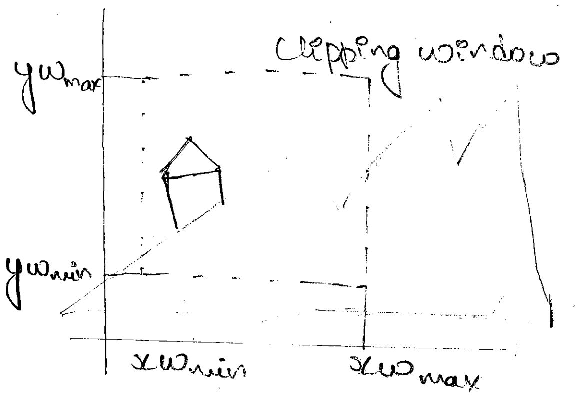
Shankar R
Asst Professor,
CSE, BMSIT&M

31) What are world coordinates & view port coordinates? Explain 2D viewing transformation pipeline.

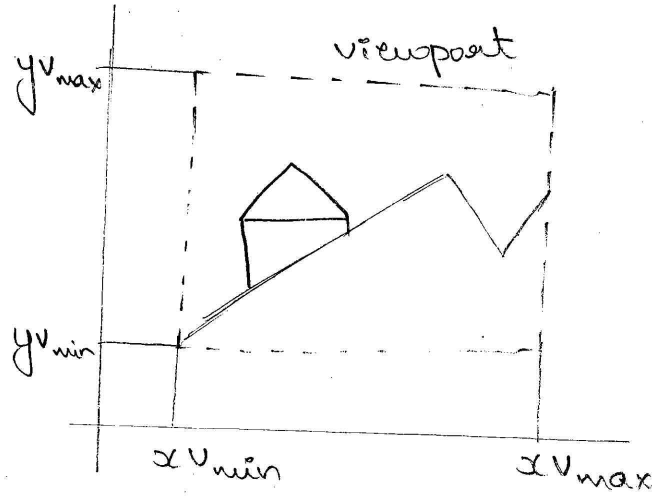
Ans. * We can define the shapes of individual objects such as trees or furniture, within a separate reference frame for each object. These reference frames are called modeling coordinates or local coordinates or master coordinates.

Once the individual object shapes have been specified, we can construct ("model") a scene by placing the objects into appropriate locations within a scene reference frame called world coordinates.

* The clipping window is mapped into a viewport. Viewing world has its own coordinates which may be a non-uniform scaling of world coordinates. An area on a display device to which a window is mapped is called a viewport.



World Coordinates



Viewport Coordinates

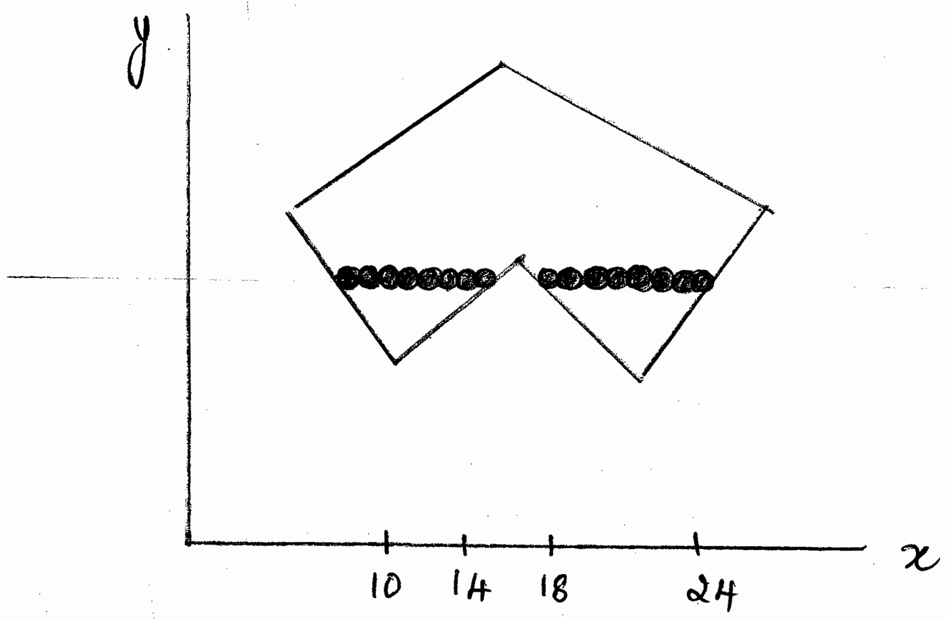
Refer to question 27 for 2D viewing transformation pipeline.

Shankar R
Asst Professor,
CSE, BMSIT&M

(32) Explain the scan line polygon fill algorithm

Ans:

Basic idea: For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are sorted from left to right. Then, we fill the pixels between each intersection pair



Steps:

→ Find minimum enclosed rectangle.
Number of scanlines = $Y_{max} - Y_{min} + 1$

→ For each scanline,

1. Obtain intersection points of scanline with polygon edges
2. Sort intersections from left to right

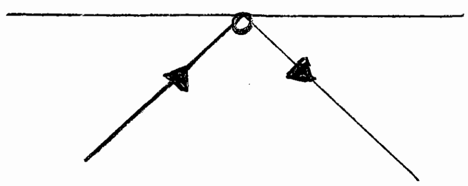
→ Form pairs of intersections from the list

→ Fill within pairs

→ Intersection points are updated for each scanline

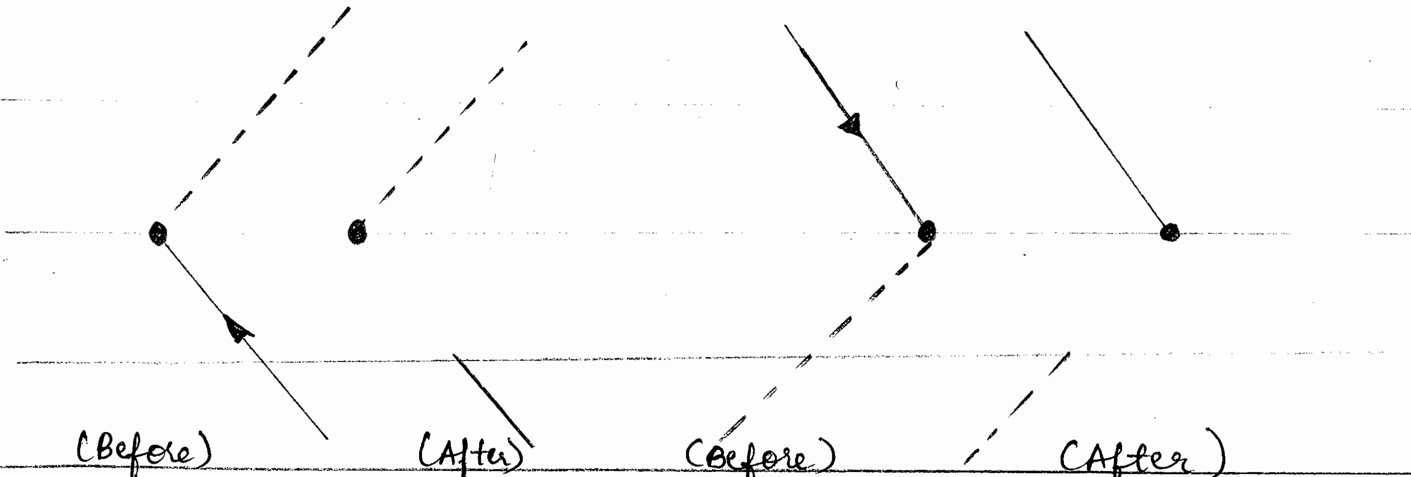
Some scan-line intersection at polygon vertices require special handling. A scan line passing through a vertex as intersecting the polygon twice

a) If the vertex is a local extrema, consider (or add) two intersections for the scan line corresponding to such a shared vertex.



⇒ 2 intersection points

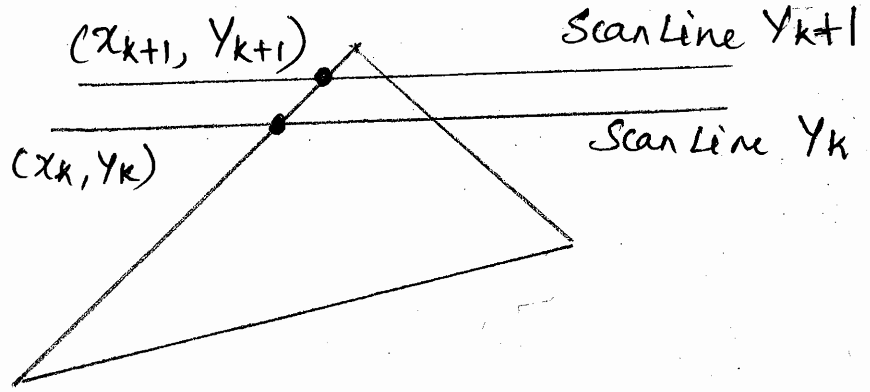
b) While processing edges non-horizontal (generally) along a edge in any order, check to determine the condition of monotonically changing (increasing or decreasing) endpoint Y. Shorten the lower edge to ensure only one intersection point at the vertex



(Before) (After) (Before) (After)

Shankar R
Asst Professor,
CSE, BMSIT&M

Coherence properties can be used in computer graphics to reduce processing. It often involve incremental calculations applied along a single scan line or between successive scan lines



$$\text{Slope} \Rightarrow m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

As the change in coordinates between two scan lines is $y_{k+1} - y_k = 1$

$$\therefore \; x_{k+1} = x_k + \frac{1}{m} \quad — (1)$$

Along the edge with slope m, the intersection x_k value for scan line k above initial scan line can be calculated as

$$x_k = x_0 + \frac{k}{m}$$

where, $m = \frac{\Delta y}{\Delta x}$

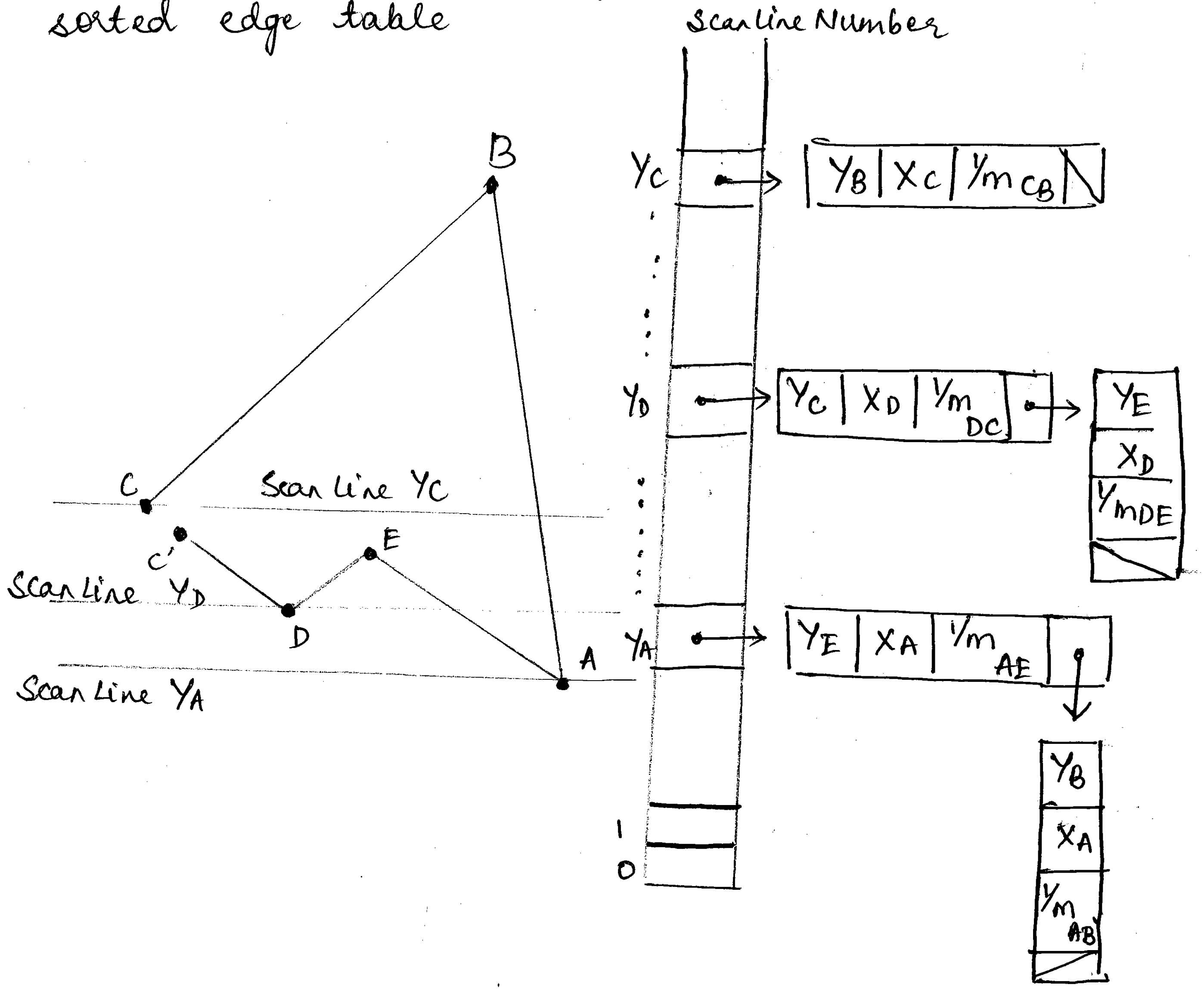
(1) can be expressed as

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}.$$

→ To perform a polygon fill efficiently, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan lines efficiently

→ Proceeding around the edges in either a clockwise or counterclockwise order, we can use a bucket sort to store the edges, sorted on the smallest y value of each edge, in the correct scan-line positions

→ Only nonhorizontal edges are entered into the sorted edge table



Shankar R
Asst Professor,
CSE, BMSIT&M

(33) Demonstrate Reflection of an object w.r.t the straight line $y=x$

Ans:- If we choose reflection axis as the diagonal line $y=x$, we could concatenate matrices for transformation sequence

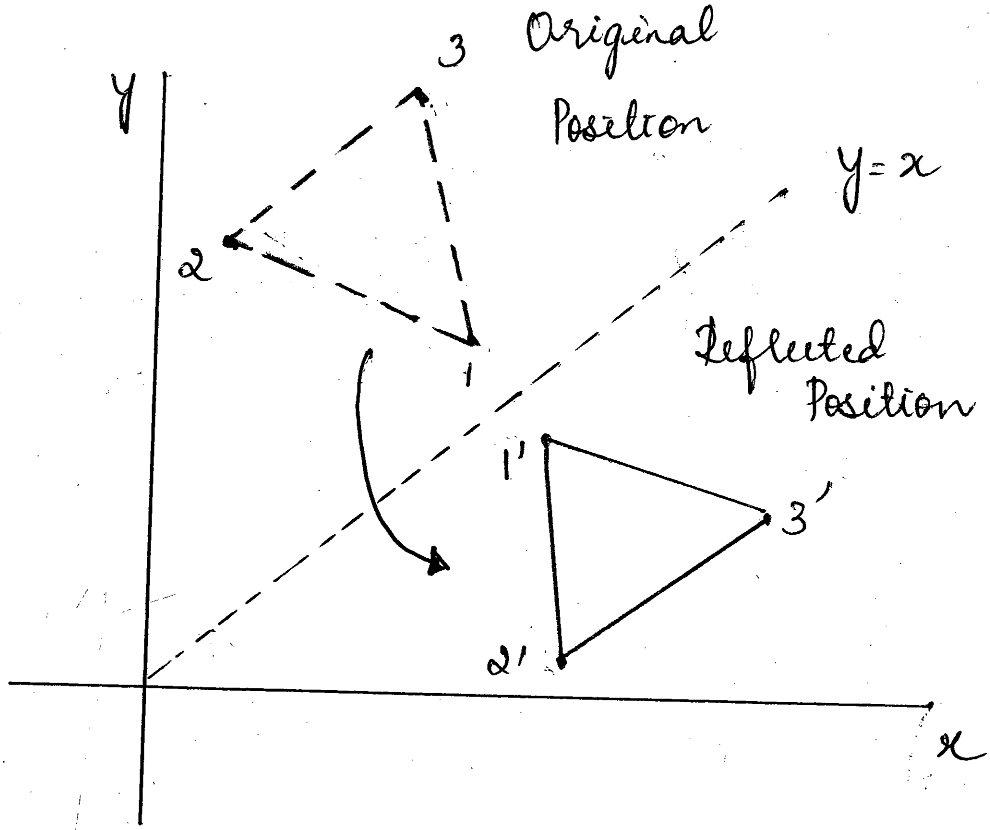
(1) Clockwise rotation by $45°$

(2) Reflection about x axis

(3) Counterclockwise rotation by $45°$

The resulting transformation matrix is

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



Shankar R
Asst Professor,
CSE, BMSIT&M

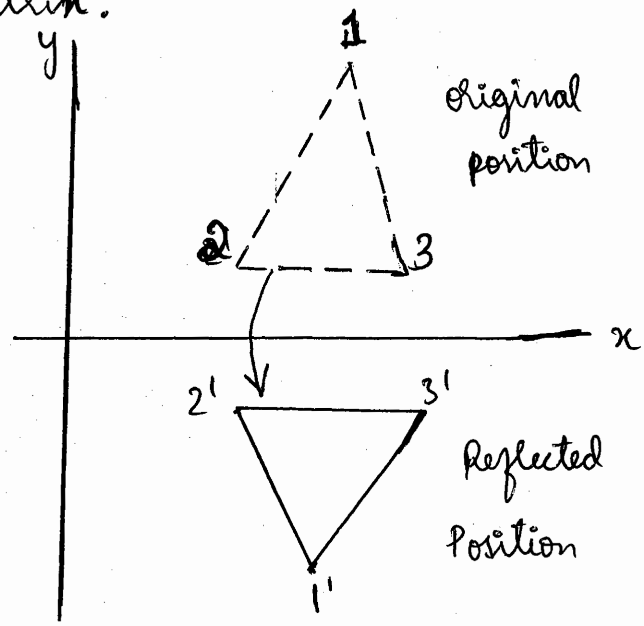
34. Explain Reflection and Shearing.

<u>Reflection</u>: A transformation that produces a mirror image of an object is called Reflection.

→ Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

(*) Reflection about line $y=0$ (the x axis) is accomplished with the transformation matrix:

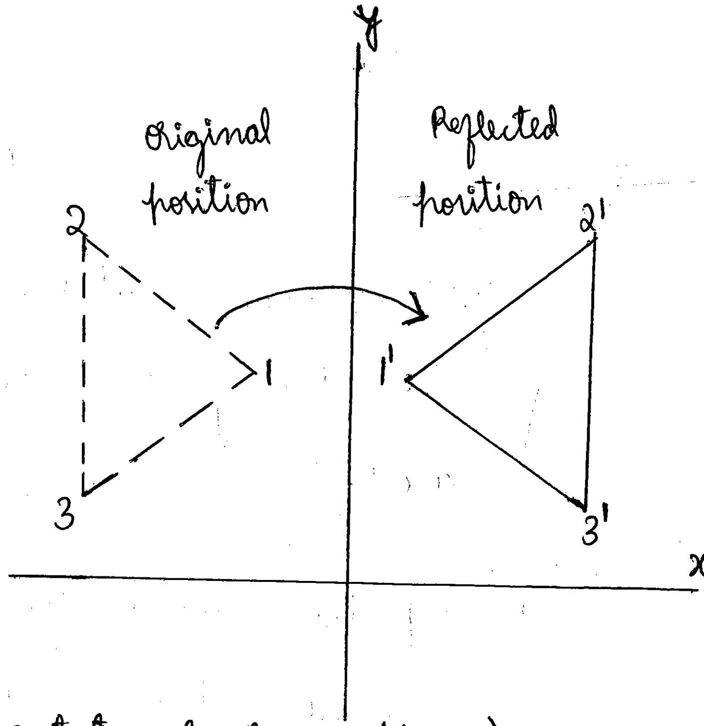
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



This transformation retains x value, but "flips" the y values of co-ordinates positions.

(*) Reflection about line $x=0$ (the y axis) is accomplished with the transformation matrix:

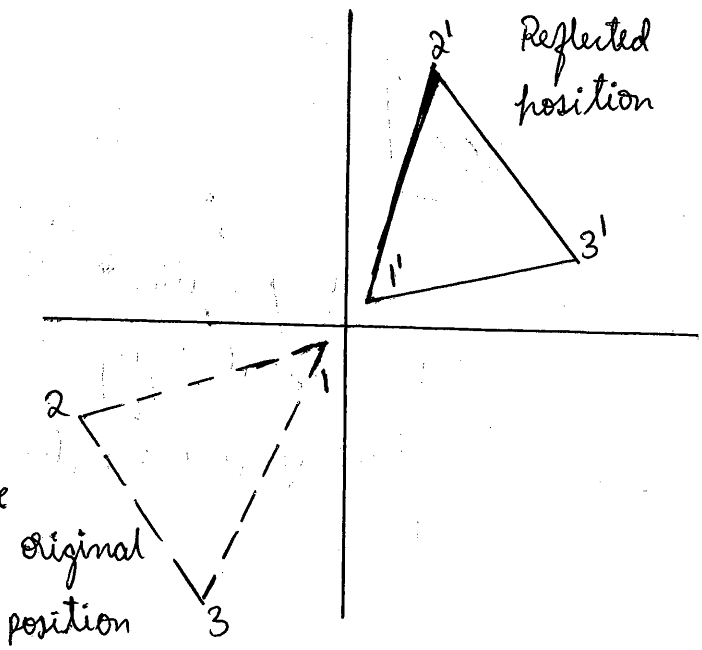
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Transformation retains y co-ordinates and flips x co-ordinates.

* Reflection about the origin (rotate about xy plane):

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

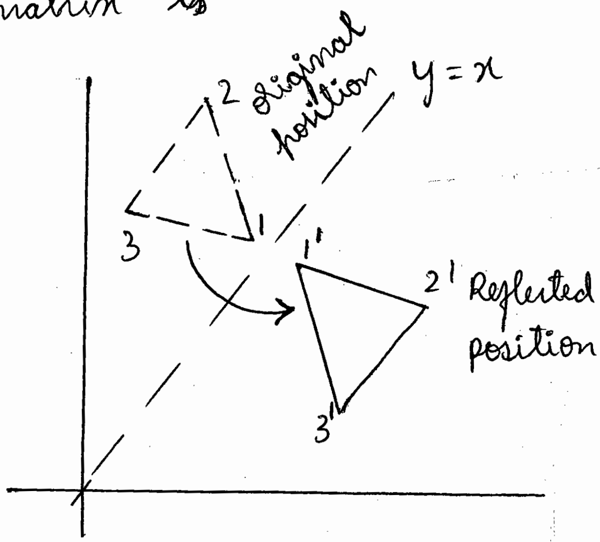


Flip both x & y co-ordinates of a point by reflecting relative to an axis that is perpendicular to xy plane and that passes through the co-ordinate origin.

Shankar R
Asst Professor,
CSE, BMSIT&M

* If we choose the reflection axis as diagonal line $y = x$, the reflection matrix is

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



To obtain above matrix for reflection, we could concatenate matrices for transformation sequence:

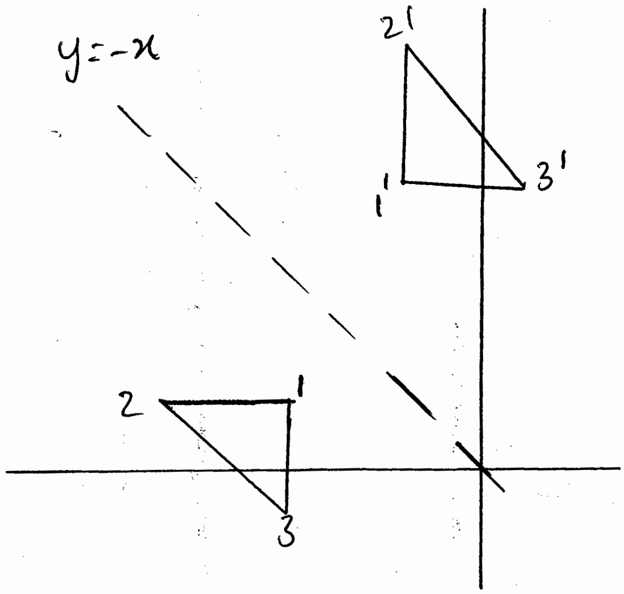
a) clockwise rotation by 45°.

b) Reflection about x-axis (for $y = x$) or reflection about y-axis (for $y = -x$).

c) counterclockwise rotation by 45°.

* for diagonal $y = -x$, the reflection matrix is

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



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.

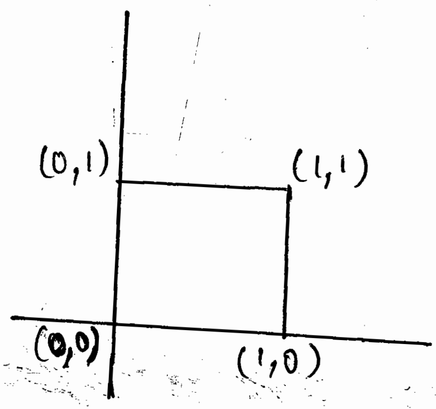
→ Two common shearing transformations are those that shift co-ordinate x values and those that shift y values. An x-direction shear relative to the x axis is produced with the transformation matrix.

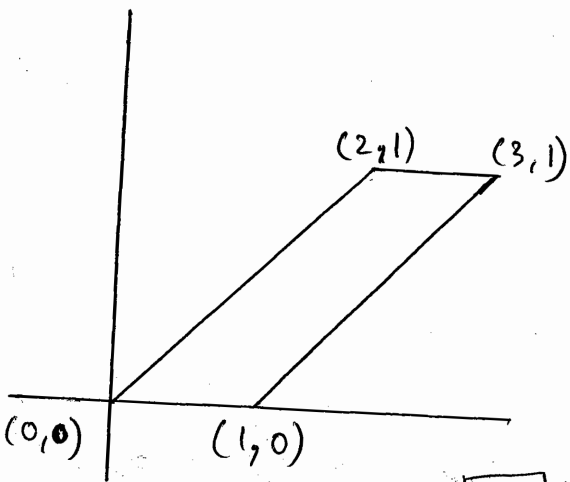
$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms co-ordinate positions as

$$x' = x + sh_x \cdot y, \quad y' = y$$

→ Any real number can be assigned to the shear parameter sh_x. Setting parameter sh_x to the value 2, for example, changes the square into a parallelogram is shown below. Negative values for sh_x shift coordinate positions to the left.





35. Explain with example, vector method of splitting a polygon?

Vector method

→ First need to form the edge vectors.

→ Given two consecutive vertex positions, V_K and V_{K+1}, we define the edge vector between them as

$$E_K = V_{K+1} - V_K$$

→ Calculate the cross-products of ~~successful~~ successive edge vectors in order around the polygon perimeter.

→ If the z component of some cross-products is positive while other cross-products have a negative z component, the polygon is concave.

→ We can apply the vector method by processing edge vectors in counterclockwise order if any cross-product has a negative z component, the polygon is concave and we can split it along the line of the first edge vector in the cross-product pair

Shankar R
Asst Professor,
CSE, BMSIT&M

→ We can generate x-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, coordinate positions are transformed as

$$x' = x + sh_x (y - y_{ref}), \qquad y' = y$$

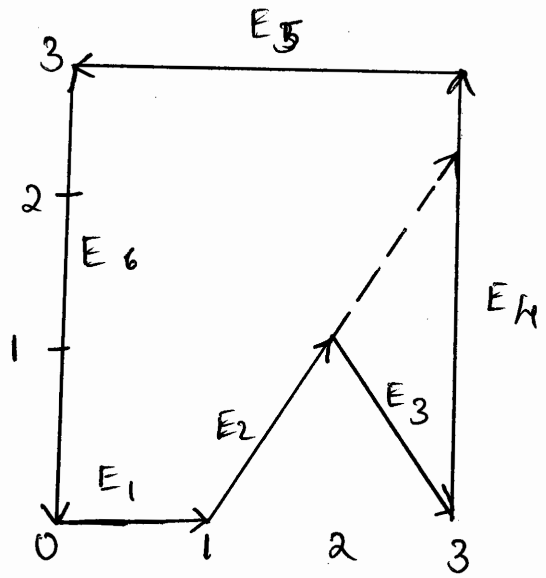
→ A y-direction shear relative to the line $x = x_{ref}$ is generated with the transformation Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

which generates the transformed co-ordinate values

$$x' = x, \quad y' = y + sh_y (x - x_{ref})$$

Shankar R
Asst Professor,
CSE, BMSIT&M



$E_1 = (1,0,0)$

$E_2 = (1,1,0)$

$E_3 = (1,-1,0)$

$E_4 = (0,2,0)$

$E_5 = (-3,0,0)$

$E_6 = (0,-2,0)$

→ Where the z component is 0, since all edges are in xy plane

→ The values for above fig is as follows

$E_1 \times E_2 = (0,0,1)$

$E_2 \times E_3 = (0,0,-2)$

$E_3 \times E_4 = (0,0,2)$

$E_4 \times E_5 = (0,0,6)$

$E_5 \times E_6 = (0,0,6)$

$E_6 \times E_1 = (0,0,2)$

→ Since the cross-product E2×E3 has a negative z component, we split the polygon along the line of vector E2.

→ The line equation for this edge has a slope of 1 and a y intercept of -1. No other edge cross-products are negative, so the two new polygons are both convex.

Shankar R
Asst Professor,
CSE, BMSIT&M

Q.36 Open gL Polygon fill area function

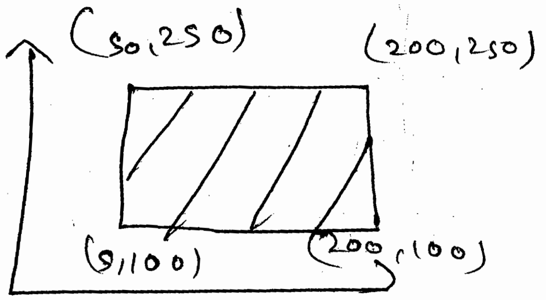
→ glRect (x_1, y_1, x_2, y_2)

One corner of Rectage is at coordinate point (x_1, y_1) & opposite corner of rectage at position (x_2, y_2)

Suffix code for glRect speifies the coordinate data type & whether co-ordinates are to expressed as array element

i = int
s = short
f = float
d = double
v = vector

glRect i(200,100, 50,250)

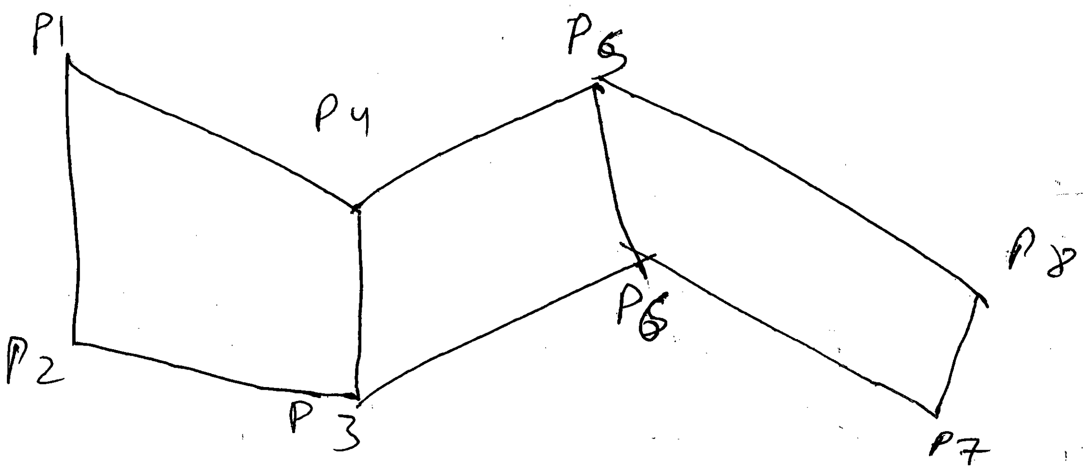


Polygon

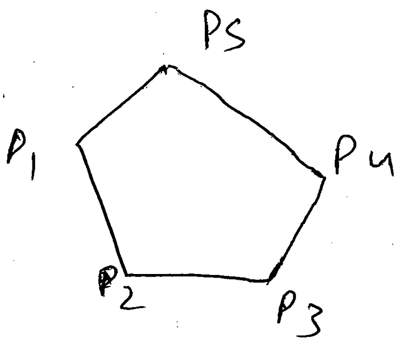
gl Begin (GL-POLYGON)
gL Vertex 2iv (P1) ;
gL Vertex 2iv (P2) ;
gL Vertex 2iv (P3) ;
gL Vertex 2iv (P4);
gL Vertex 2iv (P5);
gl END();

For list of N Vertices, we obtain N/2 -1 quadrilaterals provided that N≥4, Thus our first quad (n=1) is listed as having Vertex ordering of (P1, P2, P3, P4). The second quad (n=2) has Vertex order (P4, P3, P8, P5) Vertex ordering & third quad of N=3 is (P5, P6 P7, P8

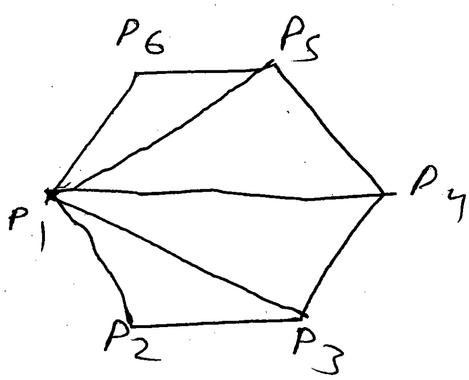
Shankar R
Asst Professor,
CSE, BMSIT&M



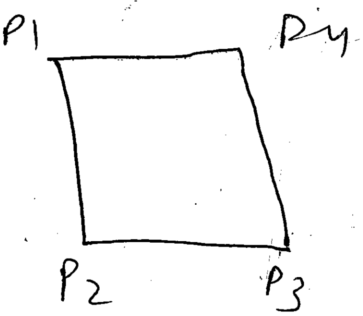
Polygon Vertex list must contain at least 3 vertices otherwise nothing will be displayed



GL-POLYGON



GL-TRIANGLE FAN



GL-QUAD

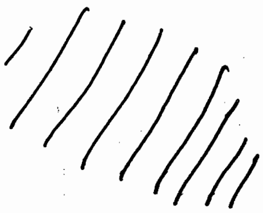
Shankar R
Asst Professor,
CSE, BMSIT&M

Q.37 Notion

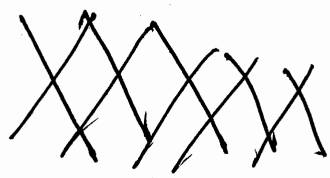
1) fill style →

Hollow. Solid Patter.

We can also fill selected region of scene using brush style, color blending combination & textures. We can also list different colour for different position in the array & fill pattern could be specific bit array that indicate which relative position are to be displayed in simple selected color.



Diagonal Hatch fill



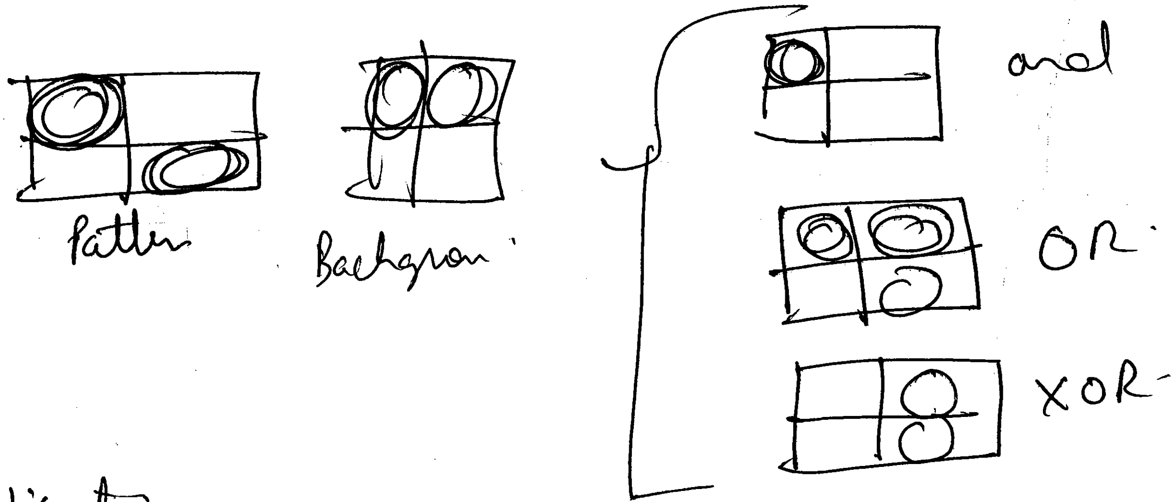
Diagonal crosshatch fill

Process of filling an area with rectangular pattern is called tiling & pattern is refered as tiling pattern

Shankar R
Asst Professor,
CSE, BMSIT&M

2) Color blended fill region

Combine a fill pattern with background color using a transparency factor that determines how much of background should be mixed with object color.



Application

Fill methods using blended colors have been referred to as soft fill

1) To soften the fill color at object borders
2) Allow To Repainting of a color area that was originally filled with semitransparent brush.

$$P = (tF) - (1-t)B$$

t → transparency factor (between 0 & 1)

P → RGB color

F → foreground color
B → Background color.

Shankar R
Asst Professor,
CSE, BMSIT&M

38): Write a OpenGL Program to rotate a triangle using Composite matrix calculation?

Ans:

```
#include <GL/glut.h>
#include <stdio.h>
int x, y;
float rotate_angle=0;
void triangle(int x, int y)
{   glColor3f(1,0,0);
    glBegin(GL_POLYGON);
     glVertex2f(x,y);
      glVertex2f(x+400, y+300);
      glVertex2f(x+300, y+0);
      glEnd();
 }
void display()
{  glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();
   glColor3f(1,1,1);
   rotate_angle+=1;
   glRotatef(rotate_angle,0,0,1);
   triangle(0,0);
   glutPostRedisplay();
   glutSwapBuffers();
 }
void Init()
{  glClearColor(0,0,0,1);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-800,800,-800,800)
```

Shankar R
Asst Professor,
CSE, BMSIT&M

```
glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{   glutInit(&argc,argv);
    glutInitWindowPosition(800,800);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Create and Rotate Triangle");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

39:- What are homogeneous coordinates? write matrix representation for Translation, rotation and Scaling.

Ans: A Standard technique for accomplishing 2D or 3D Transformation is to Expand each two-dimensional coordinate-Position representation (x,y) to a three-element representation (x_h, y_h, h), called homogeneous Coordinates.

where $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$

2D Translation:

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

$P' = T(t_x, t_y) \cdot P$

Shankar R
Asst Professor,
CSE, BMSIT&M

2D Scaling:-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = S(S_x, S_y) \cdot P$$

2D - Rotational matrix:-

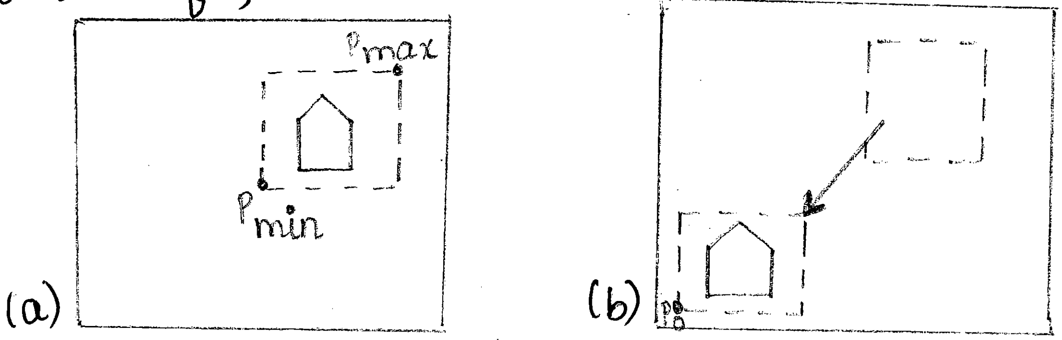
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = R(\theta) \cdot P$$

Shankar R
Asst Professor,
CSE, BMSIT&M

40) What is raster operation? Explain the raster methods for geometric transformation.

* Raster systems store picture information as color patterns in the frame buffer. Therefore, some simple object transformations can be carried out rapidly by manipulating an array of pixel values.
* Few arithmetic operations are needed, so the pixel transformations are particularly efficient.
* Functions that manipulate rectangular pixel arrays are called raster operations and moving a block of pixel values from one position to another is termed a block transfer, a bitblt or pixblt.



Translating an object from screen position (a) to the destination position shown in (b) by moving a rectangular block of pixel values. Coordinate positions Pmin and Pmax specifies the limit of the rectangular block to be moved and Po is the destination reference position.

* Rotations in 90-degree increments are accomplished easily by rearranging the elements of a pixel array.
* We can rotate a two dimensional object or pattern 90° counterclockwise by reversing the pixel values in each row of the array, then interchanging rows and columns.
* A 180° is obtained by reversing the order of the elements in each row of the array, then reversing the order

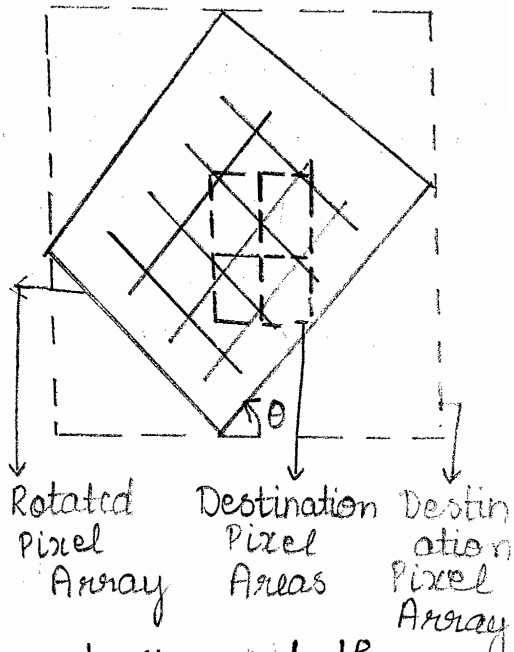
of the rows.

* Figure below demonstrates the array manipulations that can be used to rotate a pixel block by 90° and by 180°

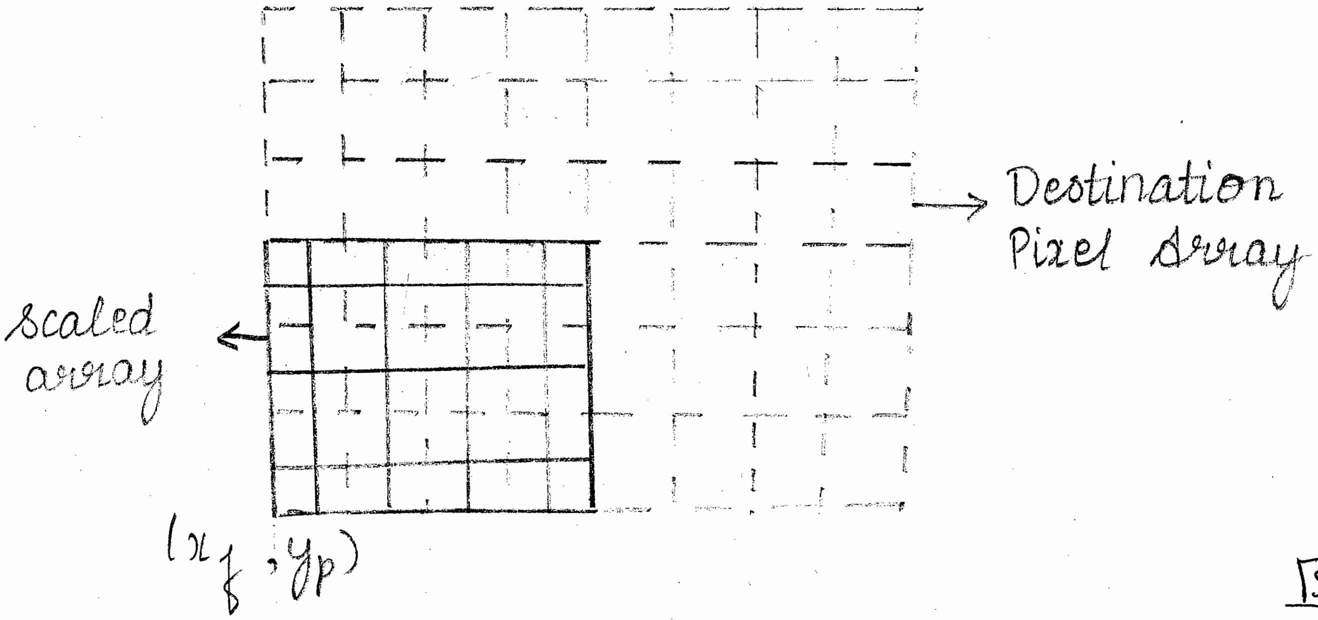
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \\ 10 & 11 & 12 \end{bmatrix} \qquad \begin{bmatrix} 3 & 6 & 9 & 12 \\ 2 & 5 & 8 & 11 \\ 1 & 4 & 7 & 10 \end{bmatrix} \qquad \begin{bmatrix} 12 & 11 & 10 \\ 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 3 & 1 \end{bmatrix}$$

(a) (b) (c)

* For array rotations that are not multiples of 90°, we need to do some extra processing.
* Each destination pixel area is mapped onto the rotated array and the amount of overlap with the rotated pixel areas is calculated.
* A color for a destination pixel can be computed by averaging the colors of the overlapped source pixels, weighted by their percentange of area overlap.



* Pixel areas in the original block are scaled, using specified values for sx and sy, and then mapped onto a set of destination pixels.
* The color of each destination pixel is then assigned according to its area of overlap with the scaled pixel areas.



Shankar R
Asst Professor,
CSE, BMSIT&M

41) Write a note on: a) OpenGL fillpattern function.
b) OpenGL texture and interpolation pattern.
c) OpenGL wire frame methods.
d) OpenGL font face function.

a) ① GLubyte fillPattern[ ]={0xFF,0x00,0xFF,0x00--};
* to fill the pattern in OpenGL, we use a 32 bit x 32 bit mask. value 1 in mask indicates the corresponding pixel is to be set to the current color.
* value 0 → leaves the value of the frame buffer position unchanged.

② Involve the polygon fill routine
glPolygonStipple(fillPattern);
we need to enable the fill routines before we specify the vertices for the polygons that are to be filled with the current pattern hence

③ we activate the polyon-fill feature of OpenGL
glEnable(GL_POLYGON_STIPPLE);
we turn off pattern filling with
glDisable(GL_POLYGON_STIPPLE);

④ Describe the polygons to be filled.

b) Use texture patterns to fill polygons. Similar simulate to the surface appearances of wood, brick, brushed steel.
* Interpolation fill of a polygon interior is used to produce realistice displays of shaded surface under various lighting conditions.

```
glShapeModel(GL_SMOOTH);
glBegin(GL_TRIANGLES);
  glColor3f(0.0,0.0,1.0);
  glVertex2i(50,50);
  glColor3f(0,0, 1,0,0.0);
```

glVertex2i(150, 50);
glColor3f(1.0, 0.0, 0.0);
glVertex2i(75, 150);
glEnd();

* GL_FLAT : fills the polygon with one color and
GL_SMOOTH : default shading.

c) OpenGL wire frame methods: To show only polygon edges
glPolygonMode (face, displayMode);

* Parameter 'face' which face of polygon we want to show edges: GL_FRONT, GL_BACK.
* Display mode: GL_LINE, GL_POINTS (polygon vertex points
* Stitching: Methods for displaying the edges of a filled polygon may produce gaps along the edges due to scanline fill. To eliminate the gap - shift the depth values calculated by fill routine so that they do not overlap with depth values for that polygon.

```
glColor3f(0, 0, 1.0, 0.0);
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0, 1.0);
glDisable(GL_POLYGON_OFFSET FILL);
glPolygonOffset(factor1, factor2);
depthOffset = factor1.maxSlope + factor2.const
```

d) Although the ordering of polygon vertices controls the identification of front and back faces. We can label the selected faces in the scene independently as front / back with the function:
glFrontFace (vertexOrder); The vertexOrder in OpenGL when set to GL_CW (close wise ordering) for its vertices will be considered to front face.

* If the vertex order in OpenGL, GL_CCW (counter clockwise ordering) of polygon vertices as front-facing which is the default-ordering.

Shankar R
Asst Professor,
CSE, BMSIT&M

42. Explain the composite 2D translation, Rotation and scaling.

* Composite Two-Dimensional Translations.

→ If two successive translation vectors (t_{1x}, t_{1y}) and (t_{2x}, t_{2y}) are applied to a two dimensional coordinate position P, the final transformed location P' is calculated as

$$P' = T(t_{2x}, t_{2y}) \cdot \{T(t_{1x}, t_{1y}) \cdot P\}$$
$$= \{T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y})\} \cdot P$$

where P and P' are represented as three-element, homogeneous-coordinate column vectors.

→ Also, the composite transformation matrix for this sequence of translations is

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x}+t_{2x} \\ 0 & 1 & t_{1y}+t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y}) = T(t_{1x}+t_{2x}, t_{1y}, t_{2y})$$

* Composite Two-Dimensional Rotations

→ Two successive rotations applied to a point P produce the transformed position.

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

where P and P' are represented as three-element, homogeneous-coordinate column vectors.

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

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

→ So that the final rotated coordinates of a point can be calculated with the composite rotation matrix as $P' = R(\theta_1 + \theta_2) \cdot P$

* <u>Composite Two-Dimensional Scalings</u>

→ Concatenating transformation matrices for two successive scaling operations in two dimensions produces the following composite scaling matrix

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S(s_{2x}, s_{2y}) \cdot S(s_{1x}, s_{1y}) = S(s_{1x} \cdot s_{2x}, s_{1y} \cdot s_{2y})$$

Shankar R
Asst Professor,
CSE, BMSIT&M

A3. Explain the 3D OpenGL geometric transformation.

→ To perform a translation, we invoke the translation routine and set the components for the three-dimensional translation vector.

→ In the rotation function, we specify the angle and the orientation for a rotation axis that intersects the coordinate origin.

→ In addition, a scaling function is used to set the three coordinate scaling factors relative to the coordinate origin. In each case, the transformation routine sets up a 4×4 matrix that is applied to the coordinates of objects that are referenced after the transformation call.

OpenGL Geometric Transformations.

→ A 4×4 translation matrix is constructed with the following routine:

glTranslate* (tx, ty, tz);

* Translation parameters tx, ty and tz can be assigned any real-number values, and the single suffix code to be affixed to this function is either f (float) or d (double).

* For two-dimensional applications we set tz = 0.0; and a two-dimensional position is represented as a four-element column matrix with the z component equal to 0.0.

* Example: glTranslatef (25.0, -10.0, 0.0);

→ Similarly, a 4 × 4 rotation matrix is generated with
glRotate* (theta, vx, vy, vz);

* where the vector v=(vx, vy, vz) can have any floating-point values for its components.
* This vector defines the orientation for a rotation axis that passes through the coordinate origin
* If v is not specified as a unit vector, then it is normalized automatically before the elements of the rotation matrix are computed.
* The suffix code can be either f or d, and parameter theta is to be assigned a rotation angle in degree.
* For example, the statement: glRotatef (90.0, 0.0, 0.0, 1.0);

→ We obtain a 4×4 scaling matrix with respect to the coordinate origin with the following routine:
glScale* (sx, sy, sz);

* The suffix code is again either f or d, and the scaling parameter can be assigned any real-number values.
* Scaling in a two-dimensional system involves changes in the x and y dimensions, so a typical two-dimensional scaling operation has a z scaling factor of 1.0

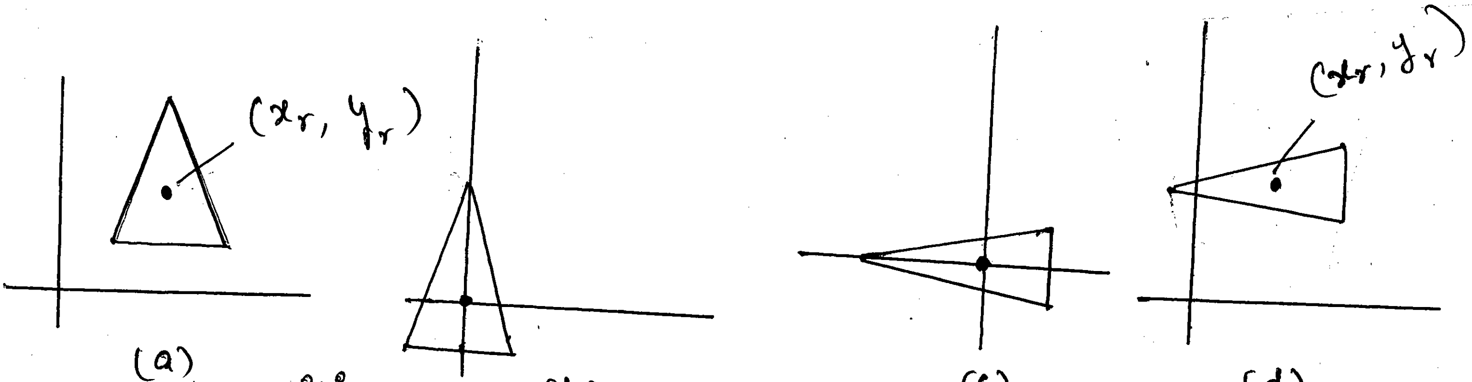
Example: glScalef (2.0, -3.0, 1.0);

Shankar R
Asst Professor,
CSE, BMSIT&M

44. Write the steps for rotation about pivot point and scaling about fixed point.

ANS:

* Two-Dimensional Pivot-Point Rotation:



(a) Original Position of Object and Pivot point.

(b) Translation of object so that Pivot Point (x_r, y_r) is at origin.

(c) Rotation about origin.

(d) Translation of object so that the Pivot Point is returned to position (x_r, y_r)

When a graphics package provides only a rotate function with respect to the coordinate origin, we can generate a two-dimentional rotation about any other pivot point (x_r, y_r) by preforming 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 coordinate origin.
3. Translate the object so that pivot point is returned to its original position.

The composite transformation matrix for this sequence is obtained with the concatenation.

Shankar R
Asst Professor,
CSE, BMSIT&M

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

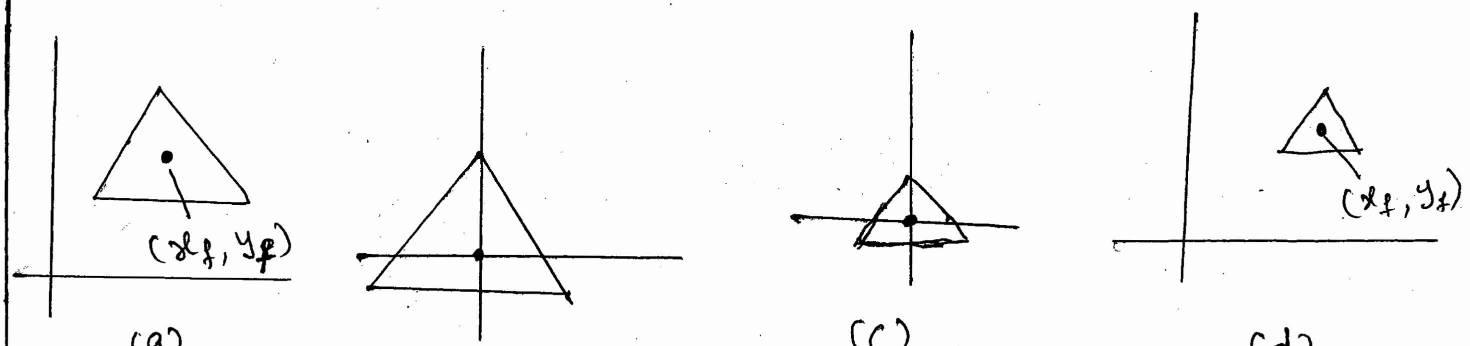
$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

which can be expressed in the form

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

where $T(-x_r, -y_r) = T^{-1}(x_r, y_r)$.

* Two-Dimentional Fixed-Point Scaling :



(a) Original Position of Object and Fixed Point

(b) Translate object so that fixed Point (x_f, y_f) is at origin.

(c) Scale object with Respect to origin

(d) Translate object so that the fixed Point is Returned to position (x_f, y_f).

When we have a function that can scale relative to the co-ordinate origin only. This sequence is

Shankar R
Asst Professor,
CSE, BMSIT&M

45. Briefly explain Inverse transformation, composite transformation.

ANS: * Inverse Transformation:

For translation, we obtain the inverse matrix by negating the translation distances. Thus, if we have two-dimensional translation distances t_x and t_y, the inverse translation matrix is

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

This produces a translation in the opposite direction, and the product of a translation matrix and its inverse produces the identity Matrix.

An inverse rotation is accomplished by replacing the rotation angle by its negative.

$$R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Negative values for rotation angles generate rotation in a clockwise direction, so the identity matrix is produced when any rotation matrix is multiplied by its inverse. Because only the sine function is affected by them change in sign of the rotation angle, the inverse matrix can also be obtained by interchanging rows and columns.

i.e $R^{-1} = R^T$

R is any rotation matrix.

Shankar R
Asst Professor,
CSE, BMSIT&M

1. Translate the 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 of the translation in step (1) to return the object to its original position.

Concatenating the matrices for these three operations produce the required ~~original position~~. Scaling Matrix:

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

or

$$T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f) = S(x_f, y_f, s_x, s_y)$$

This transformation is genrated automatically in systems that provide a scale function that accepts coordinates for the fixed point.

—— x —— x —— x ——

Shankar R
Asst Professor,
CSE, BMSIT&M

For two-dimentional scaling with parameters s_x and s_y applied relative to coordinate origin, the inverse transformation matrix is

$$S^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The inverse matrix genrates an opposite scaling transformation so the multiplication of any scaling matrix with its inverse produces the identity matrix.

* Composite Transformation:

1. Composite two-Dimentional translations

If two successive translation vectors (t_{1x}, t_{1y}) and (t_{2x}, t_{2y}) are applied to a two-dimentional coordinates Position P, the final transformed location P' is calculated as

$$P' = T(t_{2x}, t_{2y}) \cdot \{T(t_{1x}, t_{1y}) \cdot P\}$$

$$= \{T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y})\} \cdot P$$

where P and P' are represented as three-element, homogeneous-coordinate column vectors.

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x}+t_{2x} \\ 0 & 1 & t_{1y}+t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

or,

$$T(t_{2x}, t_{2y}) \cdot T(t_{1x}, t_{1y}) = T(t_{1x}+t_{2x}, t_{1y}+t_{2y})$$

which demonstrates that ~~who~~ two successive transations are additive.

Shankar R
Asst Professor,
CSE, BMSIT&M

Composite Two-Dimentional Rotations:

Two successive rotations applied to a point P produce the transformed position

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

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

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

So that the final rotated coordinates of a point can be calculated with the composite rotation matrix as

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

Composite Two-Dimentional scalings:

concatenating transformation matrices for two successive scaling operations in two ~~dimentional~~ dimentions produces the following composite scaling matrix:

$$\begin{bmatrix} S_{2x} & 0 & 0 \\ 0 & S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{1x} & 0 & 0 \\ 0 & S_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_{1x} \cdot S_{2x} & 0 & 0 \\ 0 & S_{1y} \cdot S_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or

$$S(S_{2x}, S_{2y}) \cdot S(S_{1x}, S_{1y}) = S(S_{1x} \cdot S_{2x}, S_{1y} \cdot S_{2y})$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative. ~~The fi~~ That is, if we were to triple size of an object twice in succession, the final size would be nine times that of the original.

Shankar R
Asst Professor,
CSE, BMSIT&M

Que 46 Explain the OpenGL matrix operation and matrix stacks.

Soln: → glm::translate :
Takes a matrix, translates it & returns it

Parameters:
glm::mat4 original - the matrix you want to translate
glm::vec3 dist - distance to move

→ glm::scale
Takes a matrix, and scales it

Parameters:
glm::mat4 original - the ~~ori~~ matrix you want to scale
glm::vec3 scale - the factors to scale by

→ glm::rotate
Takes a matrix & rotates it around an axis and returns it

Parameters:
glm::mat4 original - matrix you want to scale
double angle - angle you want to rotate by
glm::vec3 axis - axis to rotate by

Shankar R
Asst Professor,
CSE, BMSIT&M

47) Explain the openGL 2D viewing function.

We can use these two dimensional routines, along with the opengl view port function, all the viewing operations we need.

OpenGL Projection Mode

Before we select a clipping window and a viewport in openGL, we need to establish the appropriate mode for constructing the matrix to transform from world coordinates to screen coordinates.

```
glMatrixMode (GL_PROJECTION);
```

This designates the Projection matrix as the current matrix, which is originally set to the identity matrix.

GLU clipping-Window Function :-

To define a two-dimensional clipping window, we can use the OpenGL utility function

```
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
```

OpenGL ViewPort Function :-

```
glViewPort (xvmin, yvmin, vpwidth, vpHeight);
```

Create a Glut Display Window :-

```
glutInit (&argc, argv);
```

We have three functions in GLUT for definination a display window and choosing its dension and position.

```
glutInitWindowPosition (xTopLeft, yTopLeft);
glutInitWindowSize (dwidth, dwHeight);
glutCreateWindow ("Title of display Window");
```

Shankar R
Asst Professor,
CSE, BMSIT&M

Setting the GLUT Display-Window Mode & Color:-

Various display-window parameters are selected with the GLUT function:-

```
glutInitDisplayMode (mode);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glClearColor (red, green, blue, alpha);
glClearIndex (index);
```

Glut Display-Window identifier:-

```
windowID = glutCreateWindow ("A display Window");
```

Deleting a GLUT Display Window:-

```
glutDestroyWindow (windowID);
```

Current Glut Display Window:-

```
glutSetWindow (windowID);
```

Relocating and Resizing a Glut Display Window:-

```
glutPositionWindow (xNewTopLeft, yNewTopLeft);
glutReshapeWindow (dwNewWidth, dwNewHeight);
glutFullScreen ();
```

Managing multiple Glut Display Window

```
glutIconifyWindow ();
glutSetWindowTitle ("New Window Name");
```

(48) Translate a square with the following coordinate by 2 units in both directions A(0,0), B(2,0), C(2,2), D(0,2).

Shankar R
Asst Professor,
CSE, BMSIT&M

⇒ Two dimensional translation

To translate a 2-D position, we add translation distances t_x, t_y to the original coordinates (x, y) to obtain the new coordinate position (x', y') :

$$x' = x + t_x \quad ; \quad y' = y + t_y.$$

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$P' = P + T$$

Using homogeneous coordinates,

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

$$P' = T(t_x, t_y) \cdot P.$$

In the above eg, $t_x = t_y = 2$.

For A(0,0)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

∴ After translation A'(2,2).

For B(2,0)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$$

∴ After translation B'(4,2).

For C(2,2)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+2 \\ 2+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix}$$

∴ After translation C'(4,4).

Shankar R
Asst Professor,
CSE, BMSIT&M

For $D(0,2)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}$$

(49) Rotate a traingle at $A(0,0)$, $B(6,0)$, $C(3,3)$ by $90°$ about origin & fixed point $(3,3)$ both anticlockwise & clockwise direction.

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

$$P' = R(\theta).P$$

The above equation are for 2D rotation wrt origin,
$\theta = 90°$; $\cos 90° = 0$; $\sin 90° = 1$.

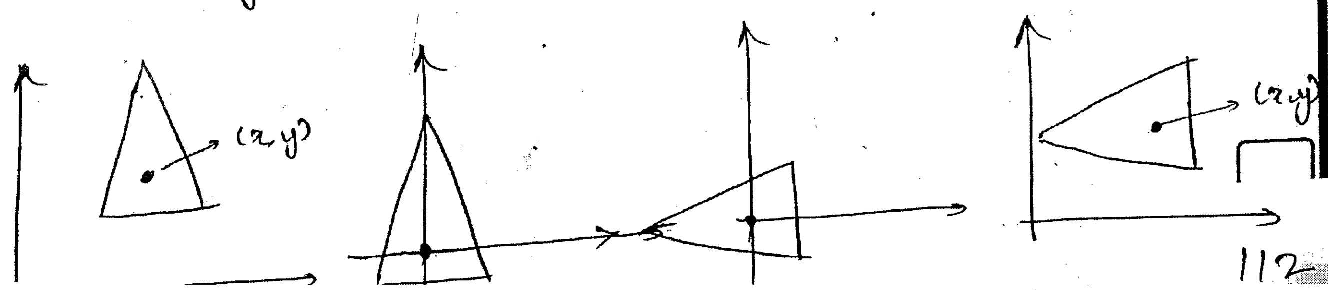
For $A(0,0)$.
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

For $B(6,0)$:
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 1 \end{bmatrix}$$

For $C(3,3)$:
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 3 \\ 1 \end{bmatrix}$$

2D rotation about pivot point:

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 is returned to its original position



Shankar R
Asst Professor,
CSE, BMSIT&M

(3)

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

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

which can be expressed in the form,

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

where $T(-x_r, -y_r) = T^{-1}(x_r, y_r)$; $\cos 90 = 0$; $\sin 90 = 1$

$x_r = 3$; $y_r = 3$;

For $A(0,0)$:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 3(1-0)+3(1) \\ 1 & 0 & 3(1-0)-3(1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix}$$

For $B(6,0)$:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 6 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 6 \\ 6 \\ 1 \end{bmatrix}$$

For $c(3,3)$:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 6 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} -3+6 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

Clockwise ~~θ~~: - we use -ve θ, $(-90°)$.

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Shankar R
Asst Professor,
CSE, BMSIT&M

$$\therefore \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

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

$x_r = y_r = 3$
$\theta = 90'$
$\cos 90' = 0$
$\sin 90' = 1.$

For $A(0,0)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 3(1-0)-3\times 1 \\ -1 & 0 & 3(1-0)+3\times 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 6 \\ 1 \end{bmatrix}$$

For $B(6,0)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -6+0+6 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

For $C(3,3)$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -3+0+6 \\ 0+1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

$\therefore$ Transformations:

$A(0,0) \longrightarrow A'(0,6)$
$B(6,0) \longrightarrow B'(0,0)$
$C(3,3) \longrightarrow C'(3,3)$ //

Shankar R
Asst Professor,
CSE, BMSIT&M

50. What are the polygon classifications? How to identify a convex polygon? Illustrate how to split a concave polygon.

→ Polygons can be classified into the following types:

i) Convex - If all interior angles of a polygon are less than or equal to 180°

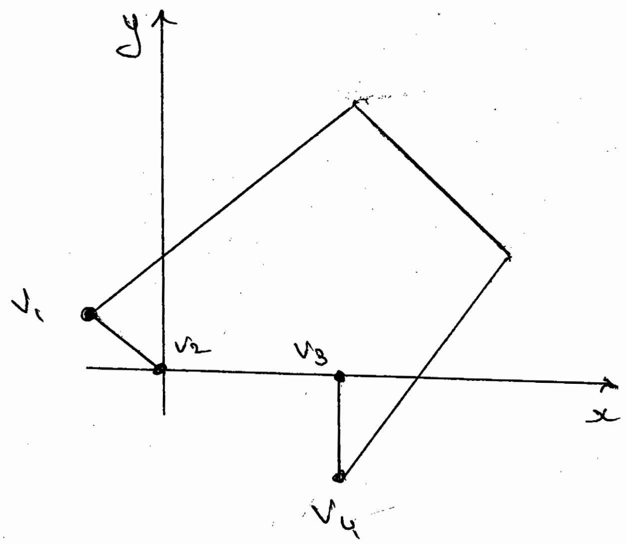
ii) Concave - A polygon that is not convex is called a concave polygon.

iii) Degenerate polygon - It is used to describe a set of vertices that are collinear or that have repeated coordinate positions.

Identifying Concave polygons - A concave polygon has atleast one interior angle greater than 180°. Also, the extension of some edges of a concave polygon will intersect other edges, and some pair of points will produce a line segment that intersects the polygon boundary. Therefore we can use any one of these charecteristics of a concave polygon for constructing identification algorithms.

If we set up a vector for each polygon edge, then we can use the cross product of adjacent edges to test for concavity. If some cross products yield a positive value and some a negative value, we have a concave polygon.

Splitting a Concave polygon - We can split a polygon using concave method. Proceeding counterclockwise around the polygon edges, we shift the position of the polygon so that each vertex V_k in turn is at the coordinate origin. Then, we rotate the polygon about the origin in a clockwise direction so that the next vertex V_{k+1} is on the axis. If the following vertex V_{k+2}, is below the x axis the polygon is concave. We then split the polygon along x axis to form two new polygons, and we repeat the concave test for each of the two new polygons. These steps are repeated until we have tested all vertices in the polygon list.



- Example, after moving V_2 to the coordinate origin and rotating V_3 onto x axis, we find that V_4 is below x axis. So we split the polygon along the line of $\overline{V_2 V_3}$ which is the x axis.

Shankar R
Asst Professor,
CSE, BMSIT&M

Q.51. What is stitching effect? How does OpenGL deals with it?

Ans: We might want to display a polygon with both an interior fill and a different color or pattern for its edges (or for its vertices). It is accomplished by using OpenGL Wire-Frame methods.

For a 3-D Polygon this method for displaying the edges of a filled polygon may produce gaps along the edges. This effect, sometimes referred to as stitching, is caused by difference between calculation in the scan line fill algorithm and calculations in the edge line-drawing of algorithm. As the interior of a 3-D Polygon is filled, the depth value (distance from the xy plane) is calculated for each (x,y) position. However, this depth value at an edge of the Polygon is often not exactly the same as the depth value calculated by the line drawing algorithm for the same (x,y) position.

One way to eliminate the gaps along displayed edges of a three-dimensional polygon is to shift the depth values calculated by the fill routine so that they do not overlap with the edge depth values for that polygon. We do this with the following two Open GL functions

- glEnable (GL_POLYGON_OFFSET_FILL);
- glPolygonOffset (factor1, factor2);

The 1st function activates the offset routine for scan-line filling and the 2nd function is used to set a couple of floating-point values factor1 and factor2 that are used to calculate the amount of depth offset. The calculation for this depth offset is

$$\text{depth offset} = \text{factor1} \cdot \text{maxSlope} + \text{factor2} \cdot \text{const}$$

where maxSlope is maximum slope of the polygon and const is an implementations const. for polygon in xy plane slope is 0. otherwise, the ~~depth~~ has to maximum slope has to be calculated.

As an example of assigning values to offset factors, we can modify the previous code segment as follows:

```
glColor3f (0.0, 1.0, 0.0);
glEnable (GL_POLYGON_OFFSET_FILL);
glPolygonOffset (1.0, 1.0);
glDisable (GL_POLYGON_OFFSET_FILL);
glColor3f (1.0, 0.0, 0.0);
glPolygonMode (GL_FRONT, GL_LINE);
```

It is possible to implement this method by applying the offset to the line-drawing algorithm, by changing the argument of the glEnable function to GL_POLYGON_OFFSET_LINE.