# BCT 2405 Computer Graphical systems

Karanja Mwangi

# Lesson objective

- **At the end of this class you will**
  - Perform Basic  2-Dimensional transformation namely translation, rotation, reflection, scaling and shear
  - Describe various components of coordinate systems used in computer graphics pipelining
  - Appreciate the role of homogenous coordinates
  - Implement Raster Methods for Transformations and OpenGL
  - Understand the transformations between 2D Coordinate Systems and OpenGL

# Geometric transformations

- **Also known as Modeling Transformations**
- **Geometric transformations: Changing an object's position (translation), orientation (rotation) or size (scaling)…. reflection and shearing operations**
- **Modeling transformations: Constructing a scene or hierarchical description of a complex object through a combination of various geometric transformations**

# 2D - Transformations

- **Why are they called  Rigid Body Transformations** - transformations that do not change the object.
- Translate
  - If you translate a rectangle, it is still a rectangle
- Scale
  - If you scale a rectangle, it is still a rectangle
- Rotate
  - If you rotate a rectangle, it is still a rectangle

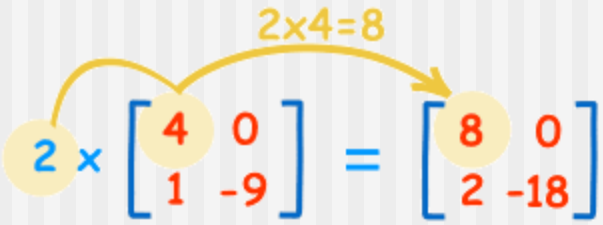
# Basics about Vertices and matrices -1

- We have always represented vertices as (x,y)
- But …..

$$(x, y) = \begin{bmatrix} x \\ y \end{bmatrix}$$

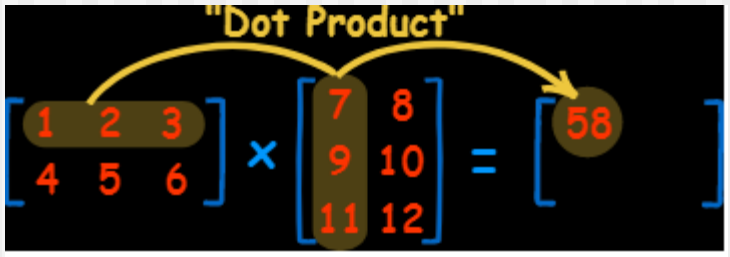
$$(3.3, 7.5) = \begin{bmatrix} 3.3 \\ 7.5 \end{bmatrix}$$

- Matrix is an array of numbers
- Various operations on Matrix
  - Scalar multiplication

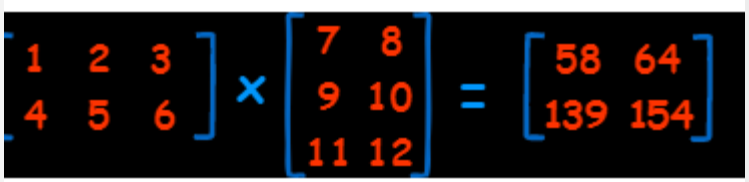


# Basics about Vertices and matrices -2

- To multiply to Matrices (they need to be compatible)- number of columns in the first matrix must be equal to the number of rows in the second matrix.
- The operation is a **Dot Product operation**




- (1, 2, 3) • (7, 9, 11) = 1×7 + 2×9 + 3×11
  = 58




- For more exercises on matrices …https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:multiplying-matrices-by-matrices/a/multiplying-matrices

# Basics about Vertices and matrices -3

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = ax + by$$

$$y' = cx + dy$$

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

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x' = ax + by + cz$$

$$y' = dx + ey + fz$$

$$z' = gx + hy + iz$$

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

# Practice

$$A = \begin{bmatrix} 1 & 7 \\ 2 & 4 \end{bmatrix}, B = \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix}$$

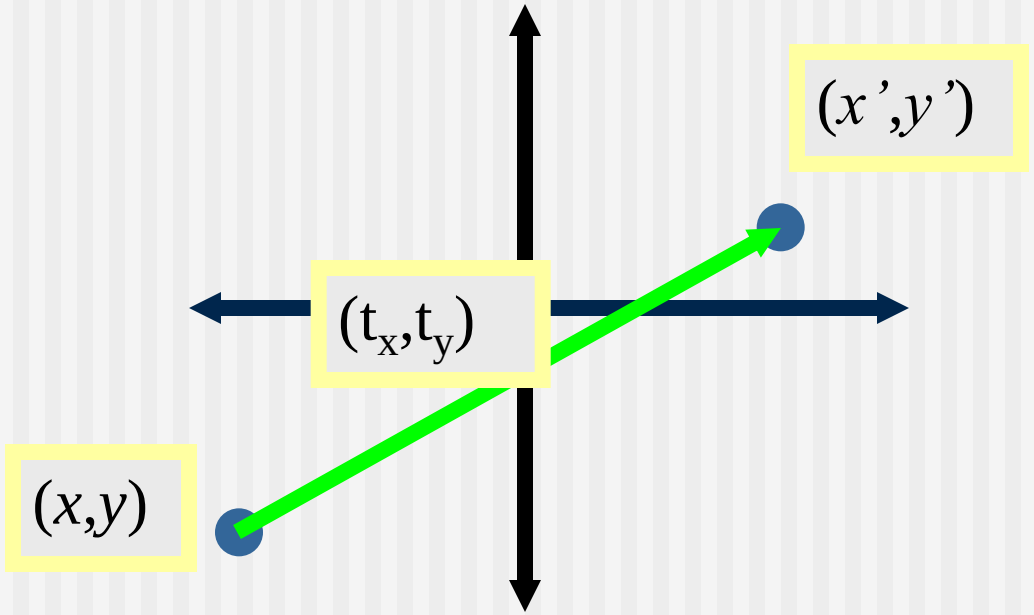
$$AB = ?$$

# Practice (You got this?)

$$AB = \begin{bmatrix} 38 & 17 \\ 26 & 14 \end{bmatrix},$$

# 2D Translation

- Translation - repositioning an object along a straight-line path (the **translation distances**) from one coordinate location to another.



# 2D Translation

- 2D Translation
  - To move a line segment, apply the transformation equation to each of the two line endpoints and redraw the line between new endpoints
  - To move a polygon, apply the transformation equation to coordinates of each vertex and regenerate the polygon using the new set of vertex coordinates

The process is as follows

- Initial coordinates of the object O = (x, y)
- New coordinates of the object O after translation = (x′, Y′)
- Translation vector or Shift vector = ($t_x$, $t_y$)

# 2D Translation

- Given:
- We want:
- Matrix form:

$$P = (x, y)$$
$$T = (t_x, t_y)$$

$$x' = x + t_x$$
$$y' = y + t_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$P' = P + T$$

$$P = (-3,-4)$$
$$T = (7,8)$$

$$x' = -3 + 7$$
$$y' = -4 + 8$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} + \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

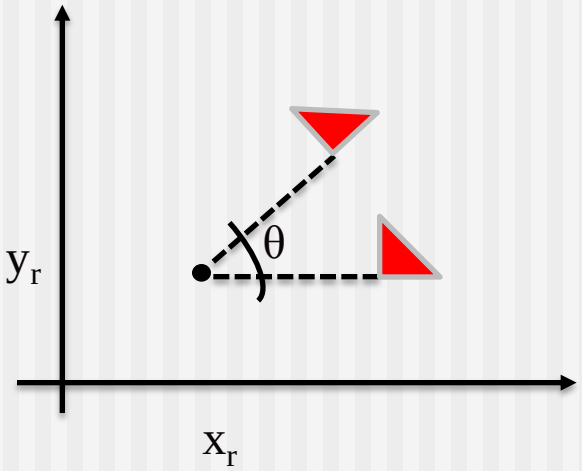
$$x' = 3$$
$$y' = 4$$

# Translation Exercises

- Given a square with coordinate points A(0, 3), B(3, 3), C(3, 0), D(0, 0). Apply the translation with distance 2 towards X axis and 2 towards Y axis. Obtain the new coordinates of the square.
- Can you implement the same in OpenGL (try out)

# 2D Rotation-1

- 2D Rotation
  - Rotation axis
  - Rotation angle
  - rotation point or pivot point ($x_r$,$y_r$)



# 2D Translation-2

- 2D Rotation
  - If θ is positive → counterclockwise rotation
  - If θ is negative → clockwise rotation
  - Remember:
    - ***cos(a + b) = cos a cos b - sin a sin b***
    - ***cos(a - b) = cos a sin b + sin a cos b***

# 2D Translation-2

$$P = (x, y)$$

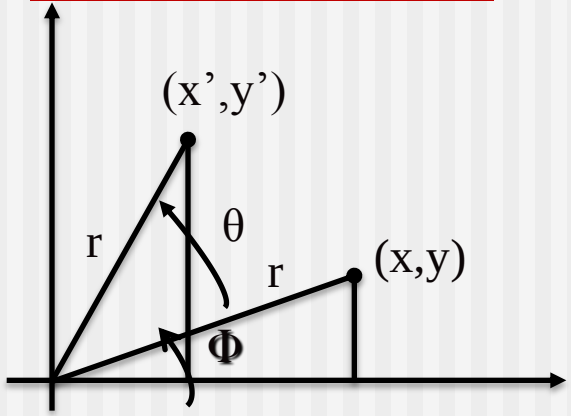
$$R = (\theta)$$

$$x = r\cos\phi$$

$$y = r\sin\phi$$

$$x' = r\cos(\phi + \Theta)$$

$$y' = r\cos(\phi + \Theta)$$




$$x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$

$$y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

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

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

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

$$P' = R \cdot P$$

# 2D Translation-3

- Rotation of a point about any specified position $(x_r, y_r)$

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

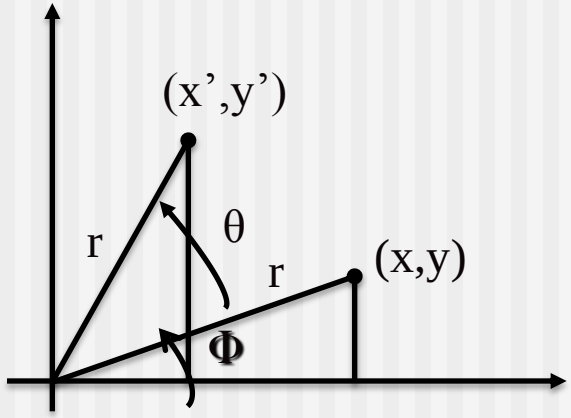
  $y' = y_r + (x - x_r) \sin\theta + (y - y_r) \cos\theta$

- Rotations also move objects without deformation
- A line is rotated by applying the rotation formula to each of the endpoints and redrawing the line between the new end points
- A polygon is rotated by applying the rotation formula to each of the vertices and redrawing the polygon using new vertex coordinates

# 2D Translation-4

- 2D Rotation
  - P′=R·P

$$R = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix}$$



# 2D rotation Exercises-1

- Given  line segment with starting point as (0, 0) and ending point as (4, 4). Apply 30 degree rotation anticlockwise direction on the line segment and find out the new coordinates of the line.

Always remember that

$$R = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix}$$

# 2D rotation Exercises-2

- $X_{new}$
  - $= X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
  - $= 4 \times \cos30^o - 4 \times \sin30^o$
  - $= 4 \times (\sqrt{3} / 2) - 4 \times (1 / 2)$
  - $= 2\sqrt{3} - 2$
  - $= 2(\sqrt{3} - 1)$
  - $= 2(1.73 - 1)$
  - $= 1.46$
- $Y_{new}$
  - $= X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
  - $= 4 \times \sin30^o + 4 \times \cos30^o$
  - $= 4 \times (1 / 2) + 4 \times (\sqrt{3} / 2)$
  - $= 2 + 2\sqrt{3}$
  - $= 2(1 + \sqrt{3})$
  - $= 2(1 + 1.73)$
  - $= 5.46$
- Thus, New ending coordinates of the line after rotation = (1.46, 5.46).

# 2D rotation Exercises-1

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos 30 & -\sin 30 \\ \sin 30 & \cos 30 \end{bmatrix} \times \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30 - 4 \times \sin 30 \\ 4 \times \sin 30 + 4 \times \cos 30 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 4 \times \cos 30 - 4 \times \sin 30 \\ 4 \times \sin 30 + 4 \times \cos 30 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1.46 \\ 5.46 \end{bmatrix}$$

# 2D rotation Exercises-2

- Given a triangle with corner coordinates (0, 0), (1, 0) and (1, 1). Rotate the triangle by 90 degree anticlockwise direction and find out the new coordinates.

- Always remember that
  - We rotate a polygon by rotating each vertex of it with the same rotation angle.

$$R = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix}$$

# 2D rotation Exercises-2

## For Coordinates A(0, 0)

$X_{new}$
$= X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
$= 0 \times \cos 90^o - 0 \times \sin 90^o$
$= 0$

$Y_{new}$
$= X_{old} \times \sin\theta + Y_{old} \times \cos\theta$
$= 0 \times \sin 90^o + 0 \times \cos 90^o$
$= 0$

Thus, New coordinates of corner A after rotation = (0, 0).

## For Coordinates B(1, 0)

Xnew
$= Xold \times \cos\theta - Yold \times \sin\theta$
$= 1 \times \cos 90^o - 0 \times \sin 90^o$
$= 0$

Ynew
$= Xold \times \sin\theta + Yold \times \cos\theta$
$= 1 \times \sin 90^o + 0 \times \cos 90^o$
$= 1 + 0 = 1$

Thus, New coordinates of corner B after rotation = (0, 1)

## For Coordinates C(1, 1)

Xnew
$= Xold \times \cos\theta - Yold \times \sin\theta$
$= 1 \times \cos 90^o - 1 \times \sin 90^o$
$= 0 - 1$
$= -1$

Ynew
$= Xold \times \sin\theta + Yold \times \cos\theta$
$= 1 \times \sin 90^o + 1 \times \cos 90^o$
$= 1 + 0$
$= 1$

Thus, New coordinates of corner C after rotation = (-1, 1).

**Thus, New coordinates of the triangle after rotation = A (0, 0), B(0, 1), C(-1, 1).**

# 2D Scaling -1

- Scale - Alters the size of an object. If scaling factor > 1, then the object size is increased. If scaling factor < 1, then the object size is reduced.
- When *sx* and *sy* are assigned the same value, a **uniform scaling** is produced, which maintains relative object proportions.
- Unequal values for *sx* and *sy* result in a **differential scaling** that is often used in design applications, where pictures are constructed from a few basic shapes that can be adjusted by scaling and
- 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 the coordinate axes. (see more on page 194 of the **Hearn, Baker and Carithers book called Computer Graphics with Open GL**)

# 2D Scaling -2

- Given:
- We want:
- Matrix form:

$$P = (x, y)$$
$$S = (s_x, s_y)$$

$$x' = s_x x$$
$$y' = s_y y$$

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

$$P' = S \cdot P$$

$$P = (1,2)$$
$$S = (3,3)$$

$$x' = 3*1$$
$$y' = 3*2.$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$x' = 3$$
$$y' = 6$$

# 2D Scaling -3

- 2D Scaling
- We can control the location of the scaled object by choosing a position called the fixed point (xf,yf)
  - $x' - xf = (x - xf)\, sx \qquad y' - yf = (y - yf)\, sy$
  - $x' = x \cdot sx + xf\,(1 - sx)$
  - $y' = y \cdot sy + yf\,(1 - sy)$
- Polygons are scaled by applying the above formula to each vertex, then regenerating the polygon using the transformed vertices

# Combining Transformations

- Many graphics applications involve sequences of geometric transformations
  - Animations
  - Design and picture construction applications
- We will now consider matrix representations of these operations
  - Sequences of transformations can be efficiently processed using matrices
  - To produce a sequence of operations, such as scaling followed by rotation then translation, we could calculate the transformed coordinates one step at a time.
  - A more efficient approach is to combine transformations, without calculating intermediate coordinate values

# Combining Transformations-2

Translation

- $T(t_x, t_y)$
- Translation distances

Scale

- $S(s_x, s_y)$
- Scale factors

Rotation

- $R(\theta)$
- Rotation angle

# Combining Transformations-3

$$P(x, y)$$

$$T(t_x, t_y)$$

$$R(\theta)$$

$$P' = P + T$$

$$P'' = R \bullet P'$$

$$x' = x + t_x$$

$$y' = y + t_y$$

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

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

$$x'' = (x + t_x)\cos\theta - (y + t_y)\sin\theta$$

$$y'' = (x + t_x)\sin\theta + (y + t_y)\cos\theta$$

$$P' = R \bullet P$$

$$P'' = P' + T$$

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

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

$$x'' = (x\cos\theta - y\sin\theta) + t_x$$

$$y'' = (x\sin\theta + y\cos\theta) + t_y$$

# Combining them

- We must do each step in turn.  First we rotate the points, then we translate, etc.
- Since we can represent the transformations by matrices, **why don't we just combine them?**

$$P' = P + T$$

$$P' = R \bullet P$$

$$P' = S \bullet P$$

# Homogeneous Coordinates

- A point $(x, y)$ can be re-written in homogeneous coordinates as $(x_h, y_h, h)$
- The homogeneous parameter $h$ is a non- zero value such that:

$$x = \frac{x_h}{h} \qquad y = \frac{y_h}{h}$$

- We can then write any point $(x, y)$ as $(hx, hy, h)$
- We can conveniently choose $h = 1$ so that $(x, y)$ becomes $(x, y, 1)$

# Why Homogeneous Coordinates?

- Mathematicians commonly use homogeneous coordinates as they allow scaling factors to be removed from equations
- We will see in a moment that all of the transformations we discussed previously can be represented as 3*3 matrices
- Using homogeneous coordinates allows us use matrix multiplication to calculate transformations – extremely efficient!

# Homogenous Coordinates

- Combine the geometric transformation into a single matrix with 3by3 matrices
- Expand each 2D coordinate to 3D coordinate with homogenous parameter
- **Two-Dimensional translation matrix**

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

- **Two-Dimensional rotation matrix**

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

- **Two-Dimensional scaling matrix**

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

# Inverse transformations

- **Inverse translation matrix**

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

$$T^{-1} * T = I$$

- **Two-Dimensional Rotation matrix**

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

$$R^{-1} * R = I$$

- **Two-Dimensional Scaling matrix**

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

$$S^{-1} * S = I$$

# Final Transformations - Compare Equations

$$T(t_x,t_y)=\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}t_x\\t_y\end{bmatrix}+\begin{bmatrix}x\\y\end{bmatrix}\rightarrow\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}1&0&t_x\\0&1&t_y\\0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

$$P=T(t_x,t_y)+P\rightarrow P=T(t_x,t_y)\bullet P$$

$$S(s_x,s_y)=\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}s_x&0\\0&s_y\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}\rightarrow\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

$$P=S(s_x,s_y)\bullet P\rightarrow P=S(s_x,s_y)\bullet P$$

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

$$P=R(\theta)\bullet P\rightarrow P=R(\theta)\bullet P$$

# Matrix Operations on OpenGL

- glMatrixMode(.);
  - Projection Mode: Determines how the scene is projected onto the screen
  - Modelview Mode: Used for storing and combining geometric transformations
  - Texture Mode: Used for mapping texture patterns to surfaces
  - Color Mode: Used to convert from one color mode to another

# OpenGL Matrix Operations

- Modelview matrix, used to store and combine geometric transformations
  - glMatrixMode(GL_MODELVIEW);
- A call to a transformation routine generates a matrix that is multiplied by the current matrix
- To assign the identity matrix to the current matrix
  - glLoadIdentity();

# OpenGL Matrix Operations (cont.)

- Alternatively:
  - glLoadMatrix* (elements16);
  - To assign other values to the elements of the current matrix
  - In column-major order:
    - First four elements in first column
    - Second four elements in second column
    - Third four elements in third column
    - Fourth four elements in fourth column

# OpenGL Matrix Operations (cont.)

- Concatenating a specified matrix with current matrix:
  - glMultMatrix* (otherElements16);
  - Current matrix is postmultiplied (right-to-left) by the specified matrix

- **By now you know that**
- Matrix notation $m_{jk}$ means:
  - In OpenGL: $j$ → column, $k$ → row
  - In mathematics: $j$ → row, $k$ → column

# OpenGL Transformation Functions

TABLE 7-1

Summary of OpenGL Geometric Transformation Functions

| Function | Description |
|---|---|
| `glTranslate*` | Specifies translation parameters. |
| `glRotate*` | Specifies parameters for rotation about any axis through the origin. |
| `glScale*` | Specifies scaling parameters with respect to coordinate origin. |
| `glMatrixMode` | Specifies current matrix for geometric-viewing transformations, projection transformations, texture transformations, or color transformations. |
| `glLoadIdentity` | Sets current matrix to identity. |
| `glLoadMatrix*` (elems); | Sets elements of current matrix. |
| `glMultMatrix*` (elems); | Postmultiplies the current matrix by the specified matrix. |
| `glPixelZoom` | Specifies two-dimensional scaling parameters for raster operations. |

# Summary

- **You have learnt how to**
  - Perform Basic 2-Dimensional transformation namely translation, rotation, **reflection**, scaling and **shear (read about the ones in red)**
  - Describe various components of coordinate systems used in computer graphics pipelining
  - Appreciate the role of homogenous coordinates
  - Implement Raster Methods for Transformations and OpenGL
  - Understand the transformations between 2D Coordinate Systems and OpenGL

- Read more from **Page 189** -Two-Dimensional Geometric Transformations –**Hearn, Baker and Carithers book called Computer Graphics with Open GL**)