# 2D AND 3D GRAPHICS WITH OPENGL

***Syllabus:***

***2D and 3D graphics with OpenGL:*** *2D Geometric Transformations: Basic 2D Geometric Transformations, matrix representations and homogeneous coordinates, 2D Composite transformations, other 2D transformations, raster methods for geometric transformations, OpenGL raster transformations, OpenGL geometric transformations function,*

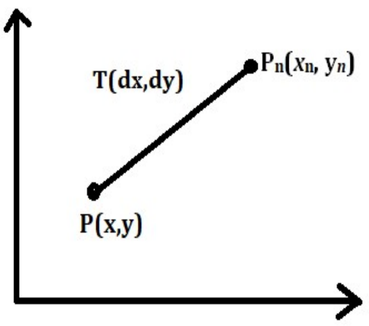
***3D Geometric Transformations:*** *Translation, rotation, scaling, composite 3D transformations, other 3D transformations, OpenGL geometric transformations functions*

***Resources:***

1. *Donald D Hearn, M Pauline Baker and Warren Carithers: Computer Graphics with OpenGL 4th Edition, Pearson, 2014*

- Operations that are applied to the geometric description of an object to change its position, orientation, or size are called **geometric transformations.**
- The basic geometric-transformation functions available are
    1. **Translation,**
    2. **Rotation,** and
    3. **Scaling.**
- Other useful transformations are **Reflection** and **Shearing.**
- To introduce the general concepts associated with geometric transformations, we first consider operations in two dimensions.

## 2-Dimensional Translation



- The transformation that changes the position from one point to other along the straight line path is called **Translation.**
- To translate a two-dimensional position, we add **translation distances *dx*** and ***dy*** to the original coordinates **(*x*, *y*)** to obtain the new coordinate position **($x_n$, $y_n$)** as shown in Figure below.
- From Figure we have,

$$x_n = x + dx$$
$$y_n = y + dy$$

The translation distance pair **(*dx*, *dy*)** is called a translation vector or shift vector.

We can express the above equations as a single matrix equation by using the following column vectors to represent coordinate positions and the translation vector:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad T = \begin{bmatrix} dx \\ dy \end{bmatrix} \quad P_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

Thus the 2D transformation equation in the matrix form is given by:

$$\boxed{P_n = P + T}$$

$$\boxed{\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}}$$

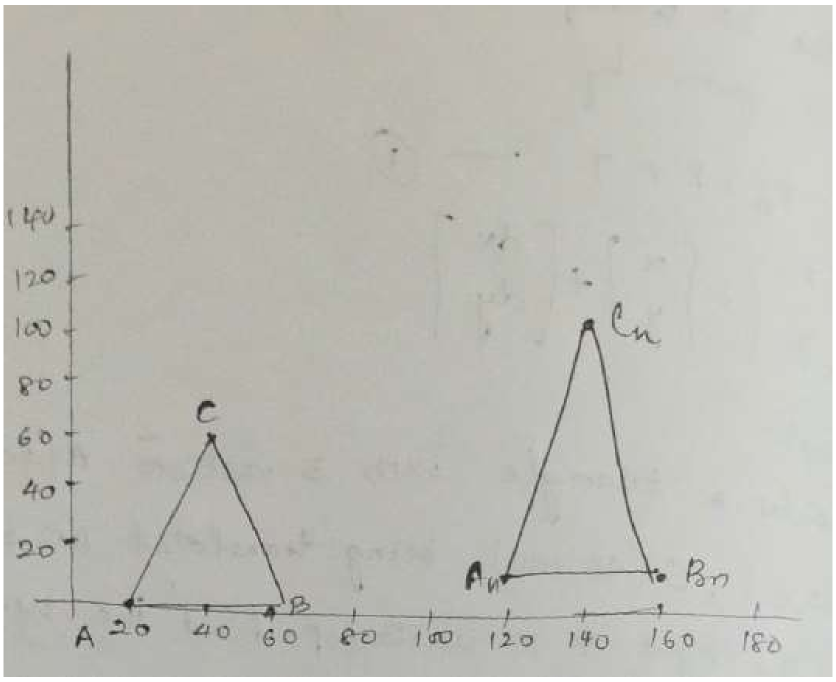
**Example:**

Consider a triangle with 3 vertices **A(20,0), B(60,0), (40,100),** being translated **100 units to the right & 10 units up**. Find the new Vertices.

**Solution:**

dx= 100, dy= 10

| **To find $A_{new}$, Given A (20,0)** | **To find $B_{new}$ Given B (60,0)** | **To find $C_{new}$, Given C(40,100)** |
|---|---|---|
| Xn= x + dx = 20 + 100 = 120 | Xn= x + dx = 60 + 100 = 160 | Xn=x+dx = 40+100 =140 |
| Yn = y + dy =0 + 10 = 10 | Yn = y + dy =0 + 10 = 10 | Yn=y+dy = 100+10 =110 |
| **$A_{new}$ (120, 10)** | **$B_{new}$ (160,10)** | **$C_{new}$ (140,110)** |

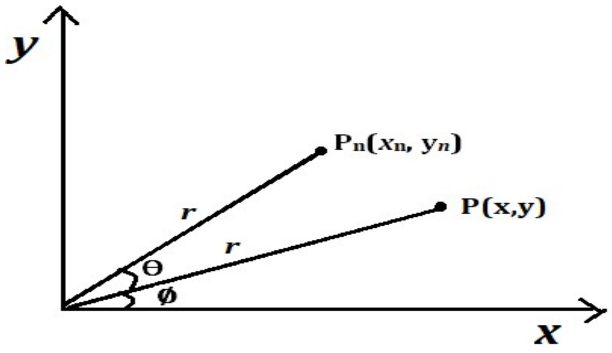


**Note:**

1. Translation is a rigid body that moves objects without deformation.
2. Polygons are translated by adding translation distances for every vertex of the polygon.
3. Circles or ellipse are translated by adding the translation distance to the center.

## 2-Dimensional Rotation

- A 2D rotation is applied to an object by repositioning it along a circular path in the xy plane.
- Rotation can be generated by specifying the rotation angle **‘θ’**. By default the equations we have is for the rotation about an origin.
- **Positive** values of **θ** specify **anticlockwise rotation**.
- **Negative** values of **θ** specify **clockwise rotation.**
- Transformation equation for rotation of a point **P** with center of rotation is origin is as shown in the figure.



-Here, **‘r’** is the distance of a point from the origin.
-angle **‘Ø’** is the original angular position of the Point **P** and **‘θ’** is the rotation angle.
- Using polar coordinate equations in terms of **‘θ’** and **r** we have ,

$$\left.\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}\right\} \textcircled{1} \text{ for point } P$$

For point $P_n$ we have,

$$x_n = r\cos(\phi+\theta)$$
$$y_n = r\sin(\phi+\theta)$$

Using the formula:

sin (A + B) = sin A cos B + cos A sin B

cos (A + B) = cos A cos B – sin A sin B

$$\Rightarrow \left.\begin{aligned} x_n &= r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y_n &= r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{aligned}\right\} \textcircled{2}$$

put $\textcircled{1}$ in $\textcircled{2}$, we have

$$\boxed{\begin{aligned} x_n &= x\cos\theta - y\sin\theta \\ y_n &= x\sin\theta - y\cos\theta \end{aligned}}$$

Thus 2D rotation equation in the matrix form is given by:

$$\boxed{\begin{aligned} x_n &= x\cos\theta - y\sin\theta \\ y_n &= x\sin\theta - y\cos\theta \end{aligned}}$$

$$\boxed{P_n = R(\theta)\cdot P}$$

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

**Example:**

Consider a triangle with 3 vertices **A(0,0), B(60,0), (40,100),** being translated **100 units to the right & 10 units up**. Find the new Vertices.

**Solution:**

**Θ= - 45°**

**<u>To find $A_{new}$, Given A (0,0)</u>**

Xn= 0*cos(-45) -0*sin(-45)=0

Yn = 0*sin(-45)+0*cos(-45)=0

**$A_{new}$ (0,0)**

**<u>To find $B_{new}$ Given B (60,0)</u>**

Xn= 60*cos(-45) -0*sin(-45)=42

Yn=60*sin(-45)+0*cos(-45)=-42

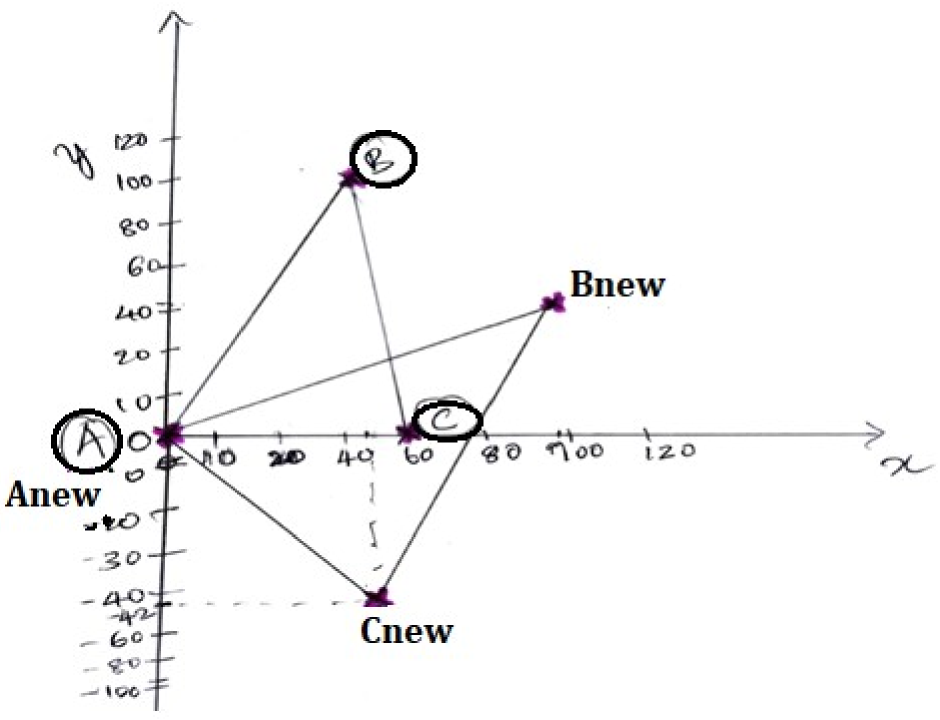
**$B_{new}$ (42,-42)**

**<u>To find $C_{new}$, Given C(40,100)</u>**

Xn=40*cos(-45)-100*sin(-45)=99

Yn=40*sin(-45)+100*cos(-45)=42

**$C_{new}$ (99,42)**



## 2-Dimensional Scaling

- A Scaling transformation is used to alter the size of an object.
- Scaling of a polygon is done by computing the product of **(x, y)** of each vertex with scaling factors **(Sx, Sy)** to product the transformed co-ordinates **($x_n$ , $y_n$).**
- The equations are:

$$\boxed{\begin{aligned} x_n &= x \cdot S_x \\ y_n &= y \cdot S_y \end{aligned}}$$

- The Transformation of in the matrix form is given by:

$$\boxed{P_n = S \cdot P}$$

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

**Note:**

1. If **(Sx, Sy) < 1,** reduces the size of the object.
2. If **(Sx, Sy) > 1**, produce an enlargement.
3. If **(Sx, Sy) = 1**, No change in the size of the object.
4. If (**Sx ≠ Sy)**, distort the picture.

## Homogeneous co-ordinates & Matrix Representation of 2D Transformations:

- The matrix representation for translation, rotation & scaling respectively are:
    1. **$P_n = P + T$**
    2. **$P_n = R.P$**
    3. **$P_n = S.P$**
- Here, translation is treated differently from scaling & rotation. We should treat all the transformations in a consistant way.
- If the points are expressed in homogeneous coordinates, all the 3 transformations can be treated multiplications.
- If points are expressed as homogeneous coordinates, we add 3rd coordinate to a point and hence instead of being represented by a pair **(x,y)** each point is represented by a triplet **(x,y,w)**. For 2D w = 1, we have **(x, y,1)**
- Because points are now 3 element column vectors, transformation matrices which multiply a point vector must be 3x3.

- In the 3x3 matrix form for homogeneous coordinates, the transformation equations are:

Translation:

$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \boxed{\therefore P_n = T \cdot P}$$

Rotation:

$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \boxed{\therefore P_n = R \cdot P}$$

Scaling:

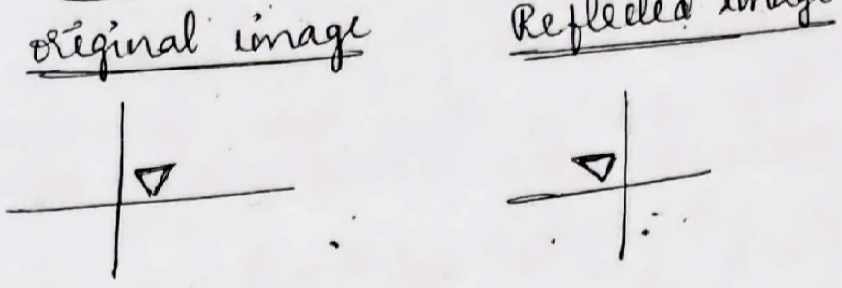
$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \boxed{\therefore P_n = S \cdot P}$$

Other transformations are **REFLECTION** and **SHEAR**.

## Reflection

- A reflection is a transformation that produces a minor image of an object relative to an axis of reflection.
- Scaling transformation with negative scaling factors gives us reflection.

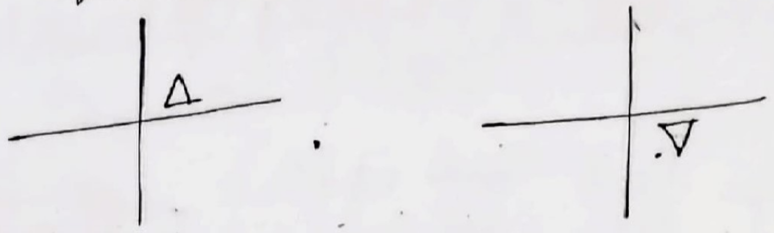
Reflection abt y-axis



Transformation Matrix

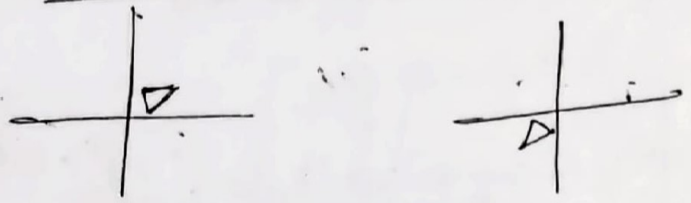
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{where } S_x = -1,\ S_y = 1$$

Reflection abt x-axis



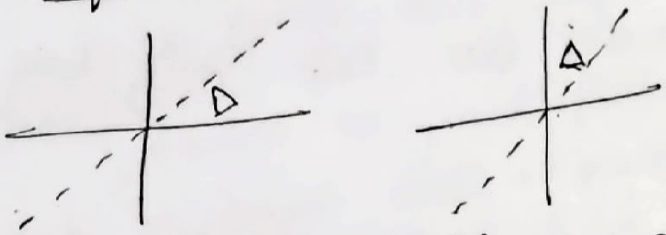
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{where } S_x = 1,\ S_y = -1$$

Reflection abt origin



$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ where $S_x = -1$, $S_y = -1$

Reflection abt line $y = x$ :-



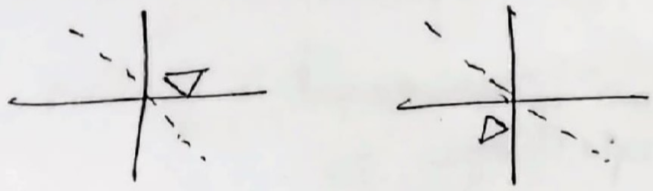
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$y = x, S_x = S_y = 1$

$x_n = y S_x$

$y_n = x S_y$

Reflection abt line $y = -x$ :-



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

$y = -x, S_x = S_y = 1$

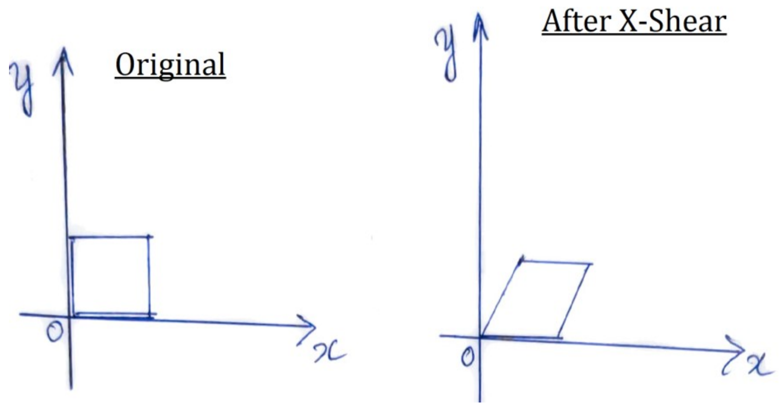
$x_n = -y S_x$

$y_n = -x S_y$

## Shear:

- A transformation that slants the shape of an object is called shear transformation.
- Two common shearing transformations are:
  1. **1.** X-shear(Shifts X coordinate values)
  2. **2.** Y-shear(Shifts Y coordinate values)

**X-Shear**



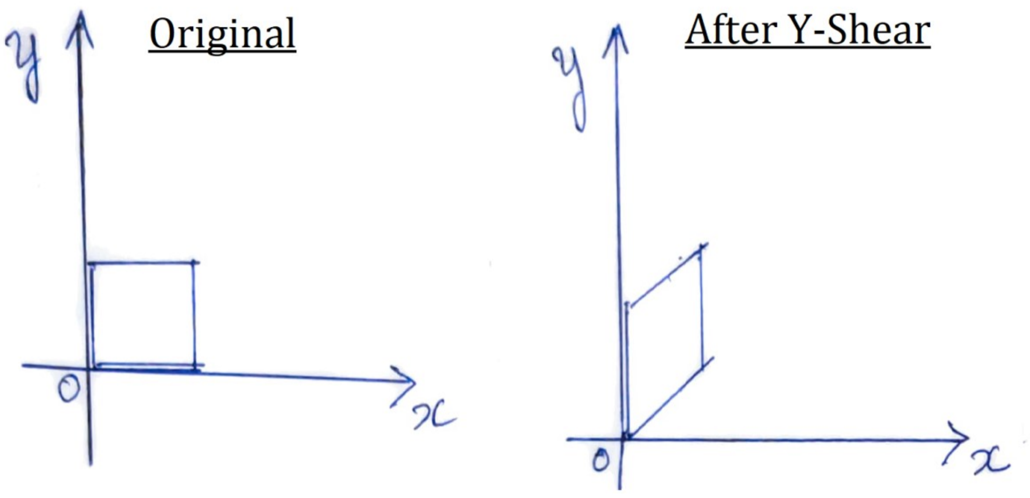
The equations are:

$$\boxed{\begin{aligned} x_n &= x + Sh_x \cdot y \\ y_n &= y \end{aligned}}$$

$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\boxed{P_n = H_x \cdot P}$$

**Y-Shear**



The equations are:

$$\boxed{\begin{aligned} x_n &= x \\ y_n &= y + Sh_y \cdot x \end{aligned}}$$

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

$$\boxed{P_n = H_y \cdot P}$$

## Inverse Transformation:

It refers to the process of undoing the transformation that has been applied to an object or set of vertices.

1. For translation,weobtain the inverse matrix by negating the translation distances. Thus, for inverse translation we have, **T (-dx, -dy)**
2. An inverse rotation is accomplished by replacing the rotation angle by its negative. Thus, for inverse rotation we have, **R(-Θ)**
3. An inverse scaling transformation is accomplished by replacing the scaling parameters with their reciprocals. Thus, for inverse scaling we have, $S\left(\frac{1}{S_x}, \frac{1}{S_y}\right)$

## Composition (Concatenation) of 2D transformation:

- The basic purpose of composing transformation is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.
- Thus, if we want to apply two transformations to point position **P**, the transformed location would be calculated as

$$P_n = M2 \cdot M1 \cdot P$$
$$= M \cdot P$$

- The coordinate position is transformed using the composite matrix **M**, rather than applying the individual transformations **M1** and then **M2**.
-

**Case 1: Rotation about arbitrary point**

We know that how to do rotation only about the origin. To rotate an object about an arbitrary point **($x_r, y_r$)** we have to carry out the following 3 steps

**1.** Translate such that **($x_r, y_r$)** is at origin. i. e, **T ($-x_r, -y_r$)**
**2.** Rotate i.e, **R(Θ)**
**3.** Apply inverse translations such that **($x_r, y_r$),** is shifted to its from origin to its original position i.e, **T($x_r, y_r$).**

Therefore the net transformation matrix is:

**T($x_r, y_r$)* R(Θ) * T ($-x_r, -y_r$)**

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

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

**Case 2: Scaling about an arbitrary point.**

We know that how to scale relative only about the origin. To scale an object about an arbitrary point **($x_f, y_f$)** we have to carry out the following 3 steps:

1. Translate such that **($x_f, y_f$)** is at origin. **i. e, T($-x_f, -y_f$)**
2. Apply scaling. i. e, **S(Sx, Sy)**
3. Apply inverse translations such that **($x_r, y_r$)** is shifted from origin to the original position **i.e, T($x_f, y_f$).**

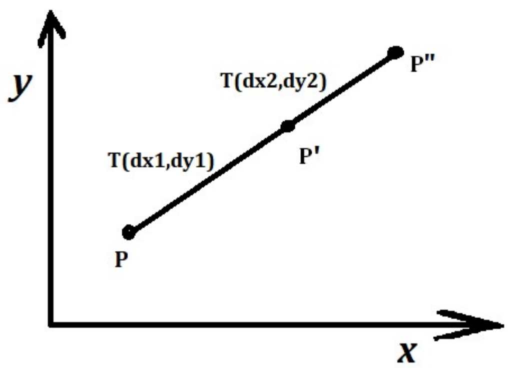
Therefore the net transformation matrix is:

**T($x_f$, $y_f$)* S(Sx, Sy) * T (-$x_f$,- $y_f$)**

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

## Composite Two-Dimensional Translations
## (Prove that 2 successive translations are additive.)

Let two successive translation vectors **(*dx1,dy1*)** and **(*dx2,dy2*)** are applied to a 2D coordinate position **P** as shown in figure:



From Figure,

**P'= T*(dx1,dy1)* • P→(1)**

**P"= T*(dx2,dy2)* · P'→(2)**

**Put (1) in (2), we get,**

**P"= T*(dx2,dy2)* · T*(dx1,dy1)* · P**

$$\therefore P'' = \begin{bmatrix}1 & 0 & dx_2\\0 & 1 & dy_2\\0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & dy_1\\0 & 1 & dy_2\\0 & 0 & 1\end{bmatrix}\cdot P$$

$$P'' = \begin{bmatrix}1 & 0 & dx_2+dx_1\\0 & 1 & dy_2+dy_1\\0 & 0 & 1\end{bmatrix}\cdot P$$

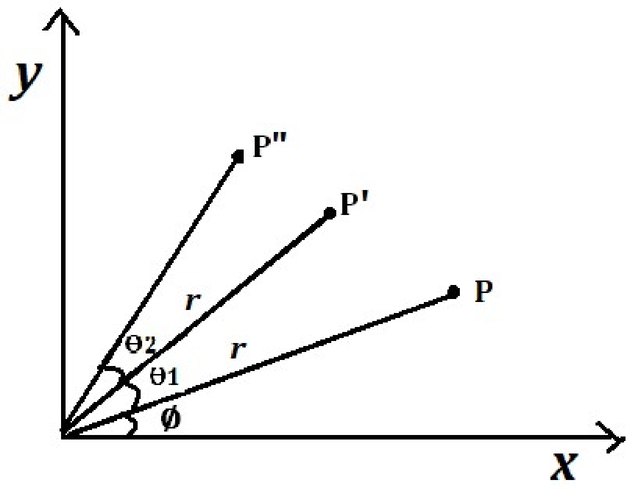
$$\boxed{\therefore P'' = T(dx_1+dx_2,\ dy_1+dy_2)\cdot P}$$

which demonstrates that two successive translations are additive.

## Composite Two-Dimensional Rotations

**(Prove that 2 successive rotations are additive.)**

Let two successive rotations are applied to a point **P** as shown in figure:



From Figure,

**P'= R(θ1) · P→(1)**

**P"= R(θ2) · P'→(2)**

**Put (1) in (2), we get,**

**P"= R(θ2) · R(θ1) · P**

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

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

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

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

## Composite Two-Dimensional Scalings

**(Prove that 2 successive scalings are multiplicative)**

Let $P' = S(Sx_1, Sy_1) \cdot P \longrightarrow \textcircled{1}$

$P'' = S(Sx_2, Sy_2) \cdot P' \longrightarrow \textcircled{2}$

put $\textcircled{1}$ in $\textcircled{2}$

$$P'' = S(Sx_2, Sy_2) \cdot S(Sx_1, Sy_1) \cdot P$$

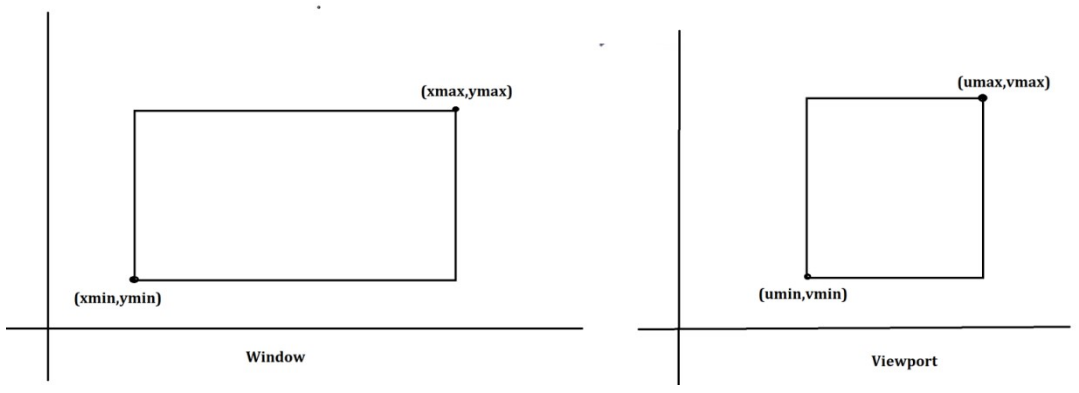
$$= \begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

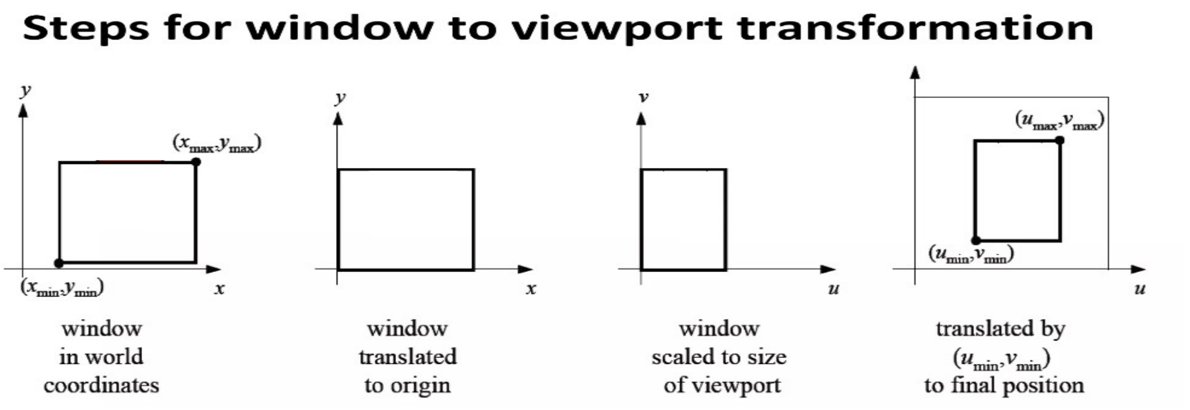
$$= \begin{bmatrix} Sx_1 Sx_2 & 0 & 0 \\ 0 & Sy_1 Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$\therefore \boxed{P'' = S(Sx_1 \cdot Sx_2, Sy_1 \cdot Sy_2) \cdot P}$$

Hence proved.

## Find the net transformation matrix for window to viewport transformation shown below:




Steps for window to viewport transformation
y
x
$(x_{max}, y_{max})$
$(x_{min}, y_{min})$
window in world coordinates
window translated to origin
v
u
window scaled to size of viewport
$(u_{max}, v_{max})$
$(u_{min}, v_{min})$
translated by $(u_{min}, v_{min})$ to final position

**1.** Translate such that **(*xmin, ymin*)** is at origin. **i. e, T(-*xmin, -ymin*)**
**2.** Apply scaling. i. e, **S(Sx, Sy) ,where**

$$Sx = \frac{umax - umin}{xmax - xmin} \quad \text{and} \quad Sy = \frac{vmax - vmin}{ymax - ymin}$$

**3.** Translations to the position **(umin,vmin) i.e, T(umin,vmin).**

Therefore, the net transformation matrix **M** is given by:

$$M = T(umin, vmin) * S\left(\frac{umax - umin}{xmax - xmin}, \frac{vmax - vmin}{ymax - ymin}\right) * T(-xmin, -ymin)$$

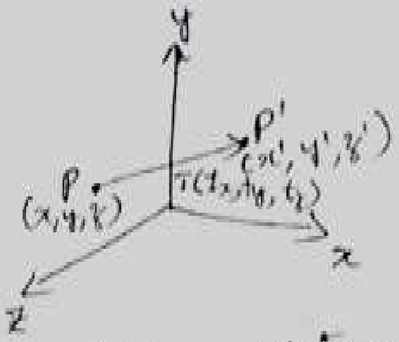
$$M = \begin{bmatrix} 1 & 0 & umin \\ 0 & 1 & vmin \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \frac{umax - umin}{xmax - xmin} & 0 & 0 \\ 0 & \frac{vmax - vmin}{ymax - ymin} & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -xmin \\ 0 & 1 & -ymin \\ 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} \frac{umax - umin}{xmax - xmin} & 0 & -xmin\left(\frac{umax - umin}{xmax - xmin}\right) + umin \\ 0 & \frac{vmax - vmin}{ymax - ymin} & -ymin\left(\frac{vmax - vmin}{ymax - ymin}\right) + vmin \\ 0 & 0 & 1 \end{bmatrix}$$

## 3D Transformation and its Homogeneous matrix representation:

Translation:- In a 3-D homogeneous coordinate system a point is translated from position $P=(x,y,z)$ to position $P=(x',y',z')$ with matrix operations

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



parameters $t_x, t_y, t_z$ specify translation distance for the coordinate direction $x, y, z$. The matrix representation contains 3 eqns:

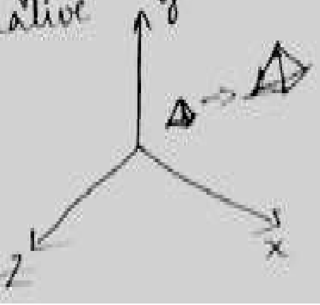
$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$

(2) Scaling:- A point in a 3-D coordinate representation is scaled from position $P(x,y,z)$ to the point $P'(x',y',z')$ with matrix operation:

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

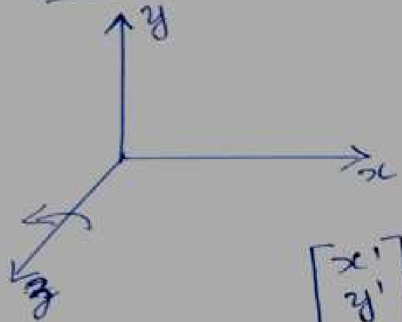
where $S_x, S_y, S_z$ are scaling parameters. The expressions for scaling, relative to the origin are:

$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$
$$z' = z \cdot S_z$$



## ③ Rotation:-

**Case 1: Rotation about z-axis:**



$$\left.\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned}\right\} ①$$

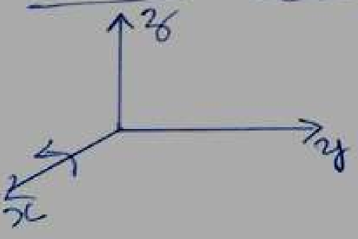
matrix form:

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

i.e $\boxed{P' = R_z(\theta) \cdot P}$

**Case 2: Rotation about x-axis:-**

change x to y & y to z in eqn ①



$$\left.\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned}\right\} ②$$

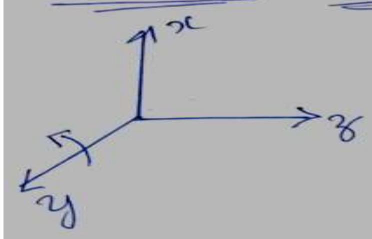
matrix form:

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

i.e $\boxed{P' = R_x(\theta) \cdot P}$

**Case 3: Rotation about y-axis:**

change z to x and y to z in eqn ②



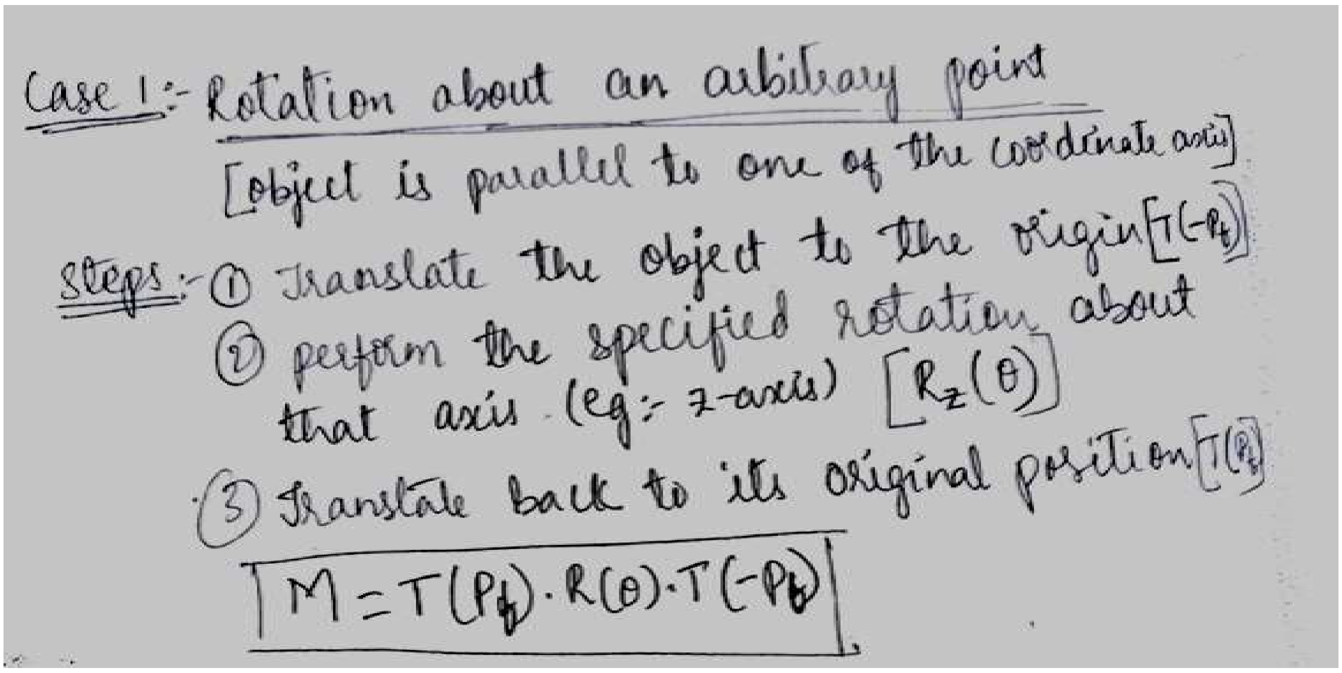
$$\left.\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \\ y' &= y \end{aligned}\right\} ③$$

matrix form

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

i.e $\boxed{P' = R_y(\theta) \cdot P}$

## Composition (Concatenation) of 3D transformation:



Case 1:- Rotation about an arbitrary point

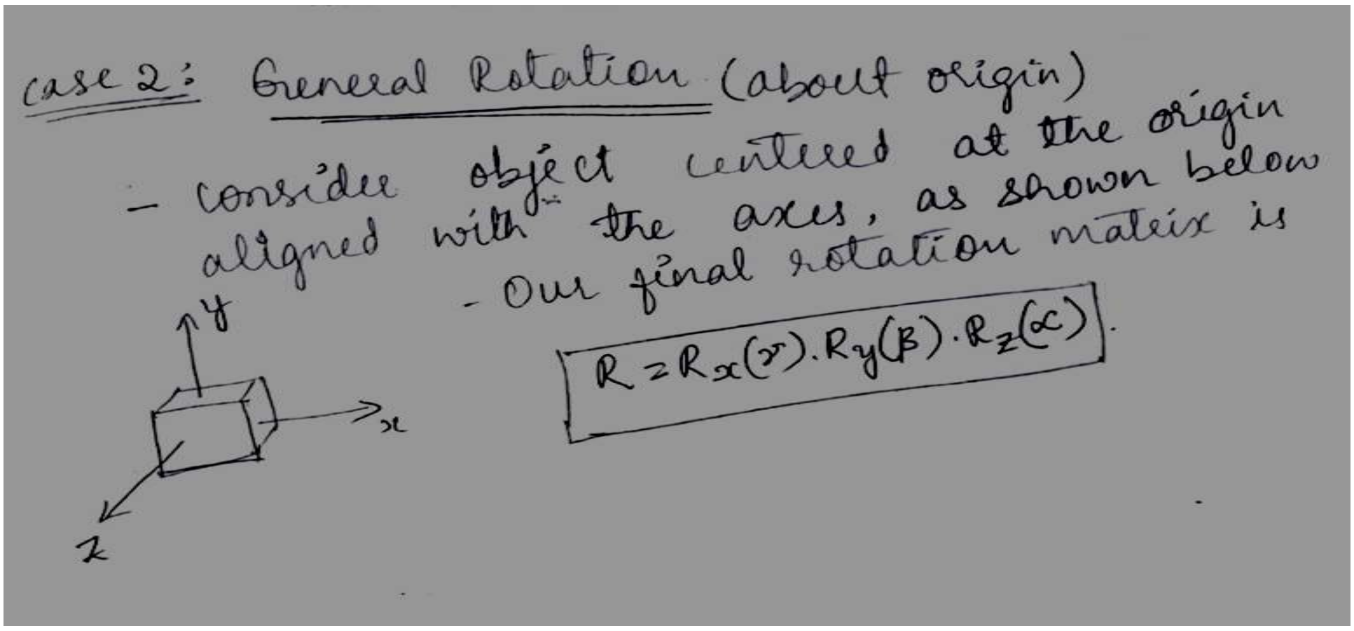
[object is parallel to one of the coordinate axis]

steps:- ① Translate the object to the origin [$T(-P_f)$]

② perform the specified rotation about that axis (eg:- z-axis) [$R_z(\theta)$]

③ Translate back to its original position [$T(P_f)$]

$$\boxed{M = T(P_f)\cdot R(\theta)\cdot T(-P_f)}$$



case 2: General Rotation (about origin)

- consider object centered at the origin aligned with the axes, as shown below

- Our final rotation matrix is

$$\boxed{R = R_x(\gamma)\cdot R_y(\beta)\cdot R_z(\alpha)}$$

### Case-3: Rotation about an arbitrary axis:

When an object is to be rotated about an axis that is not parallel to one of the co-ordinate axis (About which the rotation has to take place). We have to perform some additional transformations. The sequence is given below:

**Step 1:** Translate the object so that rotation axis specified passes through the co-ordinate origin.**[T(-P)]**

**Step 2:** Rotate the object that axis of rotation aligns with the specified coordinate axis (eg: z-axis)

a. Perform rotation about x to bring in x-z plane i.e, **[Rx (ϴx)].**
b. Perform rotation about y to bring into the Z plane i.e, **[Ry (ϴy)].**

**Step 3:** Rotate about the specified axis. **[Rz(θz)]**
**Step 4:** Inverse rotation about **y-axis** and then about **x-axis** and bring back to original orientation **[Ry (-θy). Rx (-θx)].**
**Step 5:** Inverse translation to move to original position.**[T(P)]**
**Finally we have,**

$$\boxed{N = T(-P)*Ry(-\theta y)*Rx(-\theta x)*Rz(\theta z)*Ry(\theta y)*Rx(\theta x)*T(p)}$$

## Case 4: Scaling about an arbitrary point.

We know that how to scale relative only about the origin. To scale an object about an arbitrary point **($x_f$, $y_f$)** we have to carry out the following 3 steps:
1. Translate such that point **P** is at origin. **i. e, T(-*P*)**
2. Apply scaling. i. e, **S(Sx, Sy)**
3. Apply inverse translations such that **point P** is shifted from origin to the original position **i.e, T(*P*).**

Therefore the net transformation matrix is:

$$\boxed{N = T(P)*S(Sx, Sy)*T(-P)}$$

## OpenGL Transformation Matrices

-In Open GL, several matrices are part of the state.
-The matrix state is manipulated by a common set of functions, & we use the function **glMatrixMode** to select the matrix to which the operations apply.
-In OpenGL, the model-view matrix normally is an affine-transformation matrix & has only 12 degree of freedom.
**The Current Transformation Matrix :-**
-The generalization common to most graphics systems is the current transformation matrix, (**CTM**). It is the matrix that is applied to any vertex that is defined subsequent to it's setting. If we change the **CTM**, we change the state of the system.
- The **CTM** is part of the pipeline; thus if **p** is a vertex specified in the application, then the pipeline produces **Cp**.
-The CTM is a 4x4 matrix; it can be altered a set of functions provided by the by graphics package.
-Let **C** denote the **CTM.** Initially, it is Set to the 4X4 identity matrix. We write this as

$$C \leftarrow I$$

-The functions that alter **C** are of 2 forms:
**(a)** Load it with some matrix
**(b)** Pre-multiplication (or) post-multiplication by a matrix
- OpenGL uses only post-multiplication.
- We can write **3 transformation operations in post-multiplication** form as

1. **C←CT //Translation**
2. **C←CR //Rotation**
3. **C←CS //Scaling**

and in **Load Form** as

1. **C←T //Translation**
2. **C←R //Rotation**
3. **C←S //Scaling**

Most systems allow us to load the **CTM** with an arbitrary matrix M,

**C←M //M is Transformation Matrix**

(or) to post-multiply by an arbitrary Matrix M

**C←CM //M is Transformation Matrix**

## Rotation, Translation, and Scaling:

-In openGL, the matrix that is applied to all primitives is the product of the model-view matrix (**GL-MODELVIEW**) & the projection matrix (**GL_PROJECTION**).

- We can manipulate each individually by selecting the desired matrix by

**glMatrixMode(GL-MODELVIEW)**

**glMatrixMode(GL-PROJECTION)**

-We can load a matrix with a function

**glLoadMatrixf(M)**-The matrix **M** is a one dimension array of 16 elements which are the components of the desired 4×4 matrix stored by **columns**.

-We can load an Identity matrix with the function

**glLoadIdentity()**

-We Can alter the **CTM** selected matrix with the Function

**glMultMatrixf(m)**

**m** is one dimension array of 16 elements

-Rotation, translation & scaling are provided through the following 3 functions

**glRotatef(angle, Vx, Vy, Vz)**

-**angle** in degrees.

- **(Vx, Vy, VZ)** define axis of rotation

**glTranslatef (dx, dy, dz)**

- (dx, dy, dz), components of displacement vector

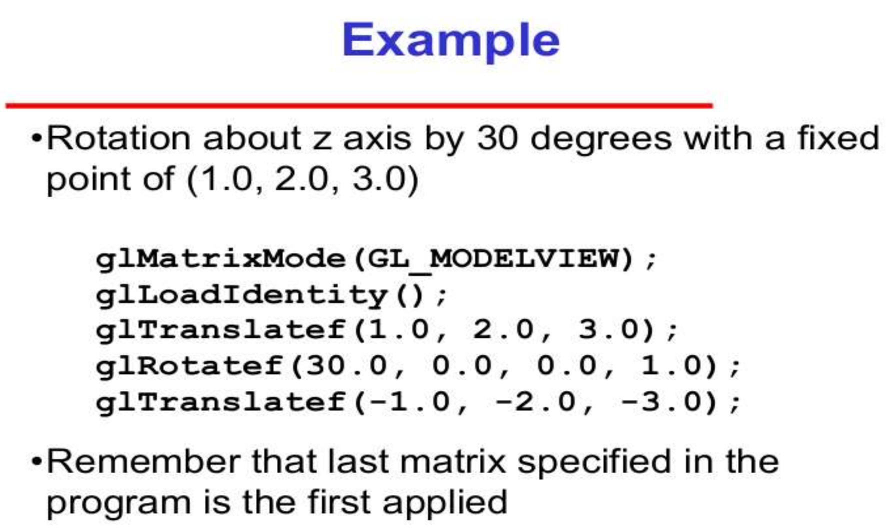
**glScalef (Sx, Sy, Sz)**

--(Sx, Sy, Sz) Specifies the scale factor.

-Each has a double (**d**) and a float **(f)** format

**Other Functions**

- **glPushMatrix();**
  - Save the state.
  - Push a copy of the CTM onto the stack.
  - The CTM itself is unchanged.
- **glPopMatrix();**
  - Restore the state, as it was at the last Push.
  - Overwrite the CTM with the matrix at the top of the stack.
- **glLoadIdentity();**
  - Overwrite the CTM with the identity matrix.

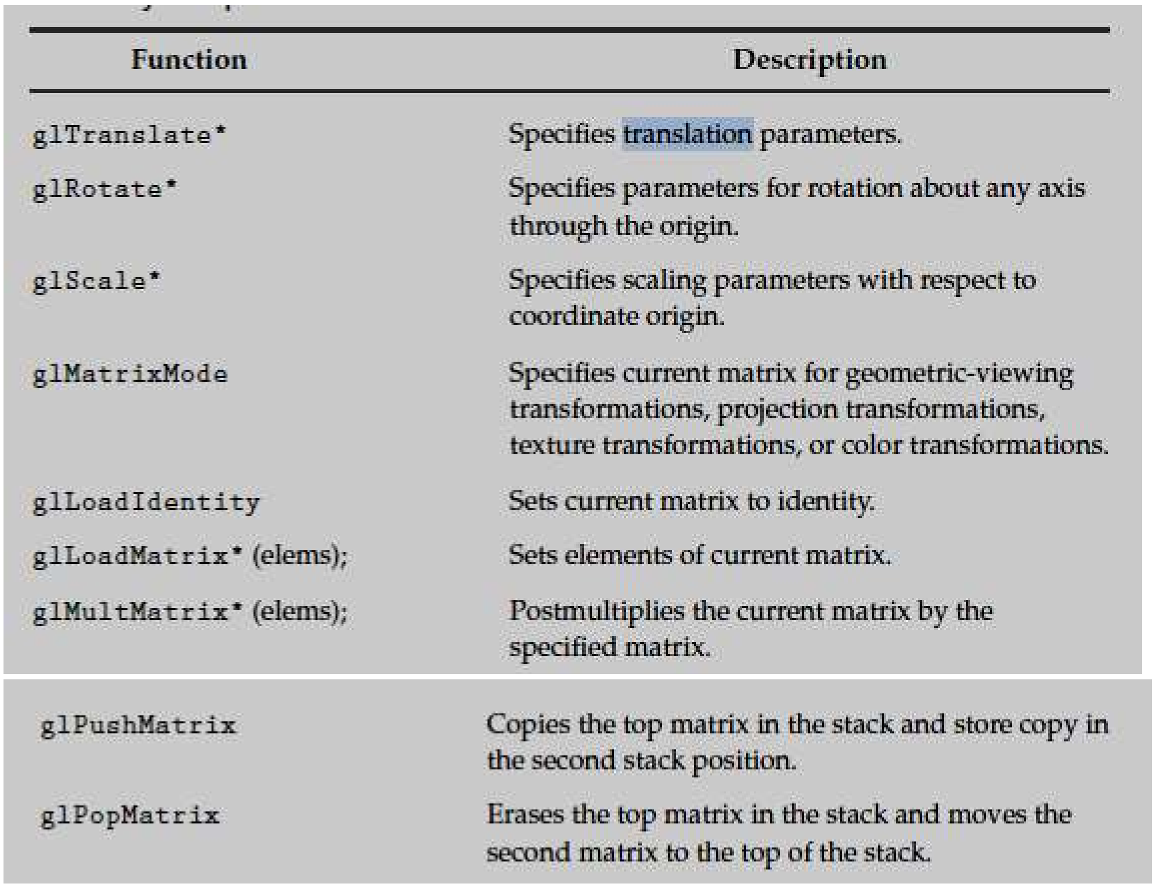


## Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);
```

- Remember that last matrix specified in the program is the first applied



| 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. |
| `glPushMatrix` | Copies the top matrix in the stack and store copy in the second stack position. |
| `glPopMatrix` | Erases the top matrix in the stack and moves the second matrix to the top of the stack. |

## Questions on Unit-2

1. Explain rotation in 2D.Show that two successive rotations are additive.
2. Briefly explain the 3 basic transformations in 3D. Obtain the homogeneous coordinate matrix for the same.
3. Explain rotation, translation and scaling with respect to 2D.
4. Consider on object ABC with co-ordinates A (1,1) ,B (10,1) ,C (5,5) Rotate the object by 90 Degree in counter clockwise direction and give co-ordinates of transformed object.
5. Apply following transformations on polygon A(10,10) ,B(10,40),C(30,10), D(20,50)and E(30,40).
   a. Translation 10, 20 units along X&Y directions.
   b. Rotate 45 degrees about the origin.
   c. Scale with scaling factor(2,2)
6. Explain window, view port and window - to - view port transformation.
7. Obtain the matrix representation for rotation of a object about an arbitrary axis.
8. Design a transformation matrix for window to viewport transformation.
9. With the help of a suitable diagram explain basic 3D Geometric transformation techniques and give the transformation matrix.
10. Design transformation matrix to rotate an 3D object about an axis that is parallel to one of the co-ordinate axis.
11. What is concatination of Transformations? Explain rotation about a fixed point.
12. All proofs
13. Examples
14. Homogeneous Transformations.
15. Explain reflection with transformation matrix.
16. Define and represent the following 2-D transformations in homogenous coordinate system.
    a. Translation
    b. Rotation
    c. Scaling
    d. Reflection
17. Explain the basic transformations in 3D and represent them in matrix form.
18. What is the need of homogeneous coordinates? Give 2-dimension homogeneous coordinate matrix for translation, rotation and scaling.
19. Obtain a matrix representation for rotation of a object about a specified pivot point in 2-dimension.
20. Explain OpenGL geometric transformation functions.
21. Explain translation, rotation and scaling of 2D transformation with suitable diagrams, equations and matrix.
22. Explain any two of the 3D geometrical transformation.