Computer Graphics

Unit 3 – Part 2

-By Manjula. S

3D Transformation

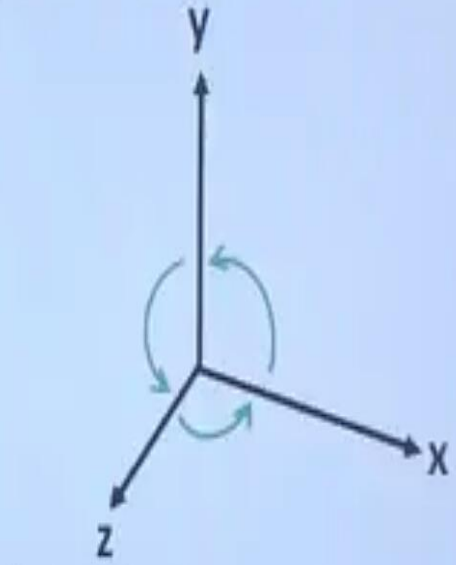
- Just as 2D transformations can be represented by 3 x 3 matrices using homogeneous coordinates, so 3D transformations can be represented by 4 X 4 matrices, provided we use homogeneous coordinate representations of points in 3-space as well.

- Thus, instead of representing a point as (x, y, z), we represent it as (x, y, z, W), where two of these quadruples represent the same point if one is a nonzero multiple of the other; the quadruple (0, 0, 0, 0) is not allowed. As in 2D, a standard representation of a point (x, y, z, W) with $W \neq 0$ is given by (x/W, y/W, z/W, 1). Transforming the point to this form is called **homogenizing**, as before.

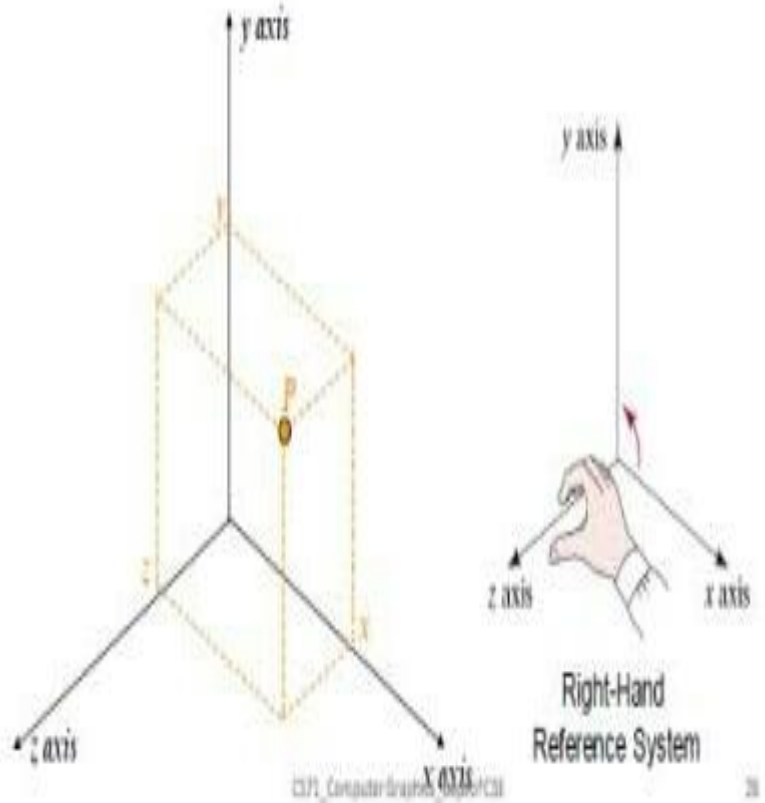
-By Manjula. S

3D Transformation

- **Right-handed System –**
  - Right-handed system gives the natural interpretation that larger z values are nearer to the viewer.



The right-handed coordinate system.



-By Manjula. S

Homogeneous Coordinate

- We use only 3 basis vector is not enough to make point and vector different
- For any point:

// General equation for point

In the frame specified by (v_1, v_2, v_3, P_0), any point P can be written uniquely as

The we can express this relation formally, using a matrix product, as

We may say that →

$$P = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix}$$

The yellow matrix called homogeneous-coordinate representation of point P

-By Manjula. S

Transformation in homogeneous coordinate

Most graphics APIs force us to work within some reference system. Although we can alter this reference system – usually a frame – we cannot work with high-level representations, such as the expression.

$$Q = P + \alpha v.$$

Instead, we work with representations in homogeneous coordinates, and with expressions such as

$$q = p + \alpha v.$$

Within a frame, each affine transformation is represented by a 4x4 matrix of the form

$$M = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

-By Manjula. S

2D Transformation

Translation

$$x_{ti} = x_i + a$$
$$y_{ti} = y_i + b$$

$$\begin{bmatrix} x_{ti} \\ y_{ti} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_i + a \\ y_i + b \end{bmatrix}$$

Rotation

$$x_{ri} = cos(\theta)x_i - sin(\theta)y_i$$
$$y_{ri} = sin(\theta)x_i + cos(\theta)y_i$$

$$\begin{bmatrix} x_{ri} \\ y_{ri} \end{bmatrix} = \begin{bmatrix} cos(\theta) - sin(\theta) \\ sin(\theta)cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x_i cos(\theta) - y_i sin(\theta) \\ x_i sin(\theta) - y_i cos(\theta) \end{bmatrix}$$

Scaling

$$x_{si} = cx_i$$
$$y_{si} = dy_i$$

$$\begin{bmatrix} x_{si} \\ y_{si} \end{bmatrix} = \begin{bmatrix} c & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} cx_i \\ dy_i \end{bmatrix}$$

Shear

$$x_{hi} = x_i + ey_i$$
$$x_{hi} = fx_i + y_i$$

$$\begin{bmatrix} x_{hi} \\ y_{hi} \end{bmatrix} = \begin{bmatrix} 1 & e \\ f & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

-By Manjula. S

2D Transformation Matrix Representation

Translation

$$\begin{bmatrix} x_{ti} \\ y_{ti} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_i + a \\ y_i + b \end{bmatrix}$$

$$\begin{bmatrix} x_{ti} \\ y_{ti} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} x_{ri} \\ y_{ri} \end{bmatrix} = \begin{bmatrix} cos(\theta) - sin(\theta) \\ sin(\theta)cos(\theta) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x_i cos(\theta) - y_i sin(\theta) \\ x_i sin(\theta) - y_i cos(\theta) \end{bmatrix}$$

$$\begin{bmatrix} x_{ri} \\ y_{ri} \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} x_{si} \\ y_{si} \end{bmatrix} = \begin{bmatrix} c & 0 \\ 0 & d \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} cx_i \\ dy_i \end{bmatrix}$$

$$\begin{bmatrix} x_{si} \\ y_{si} \\ 1 \end{bmatrix} = \begin{bmatrix} c & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

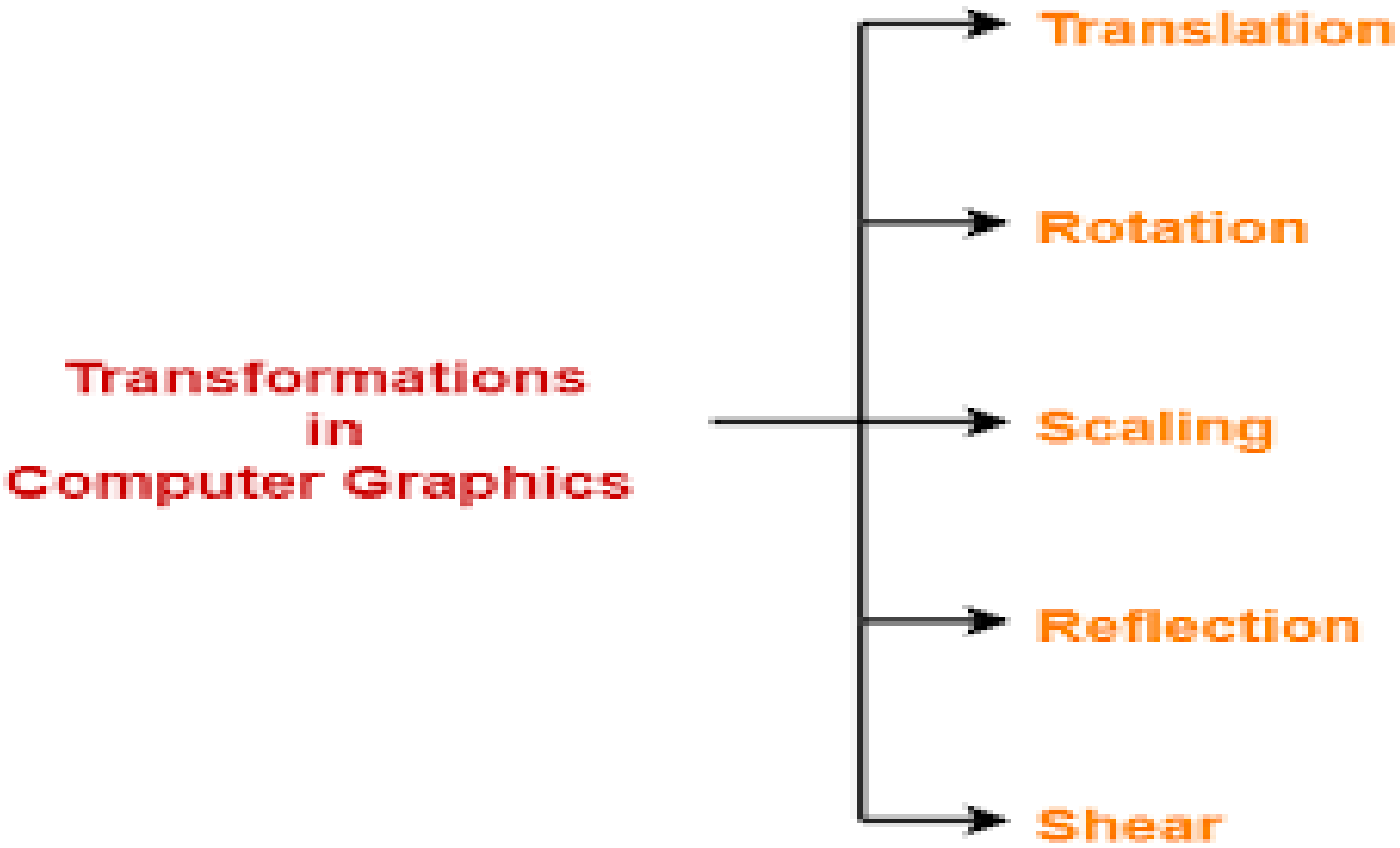
Shear

$$\begin{bmatrix} x_{hi} \\ y_{hi} \end{bmatrix} = \begin{bmatrix} 1 & e \\ f & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\begin{bmatrix} x_{hi} \\ y_{hi} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & e & 0 \\ f & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

-By Manjula. S

3D Transformations



-By Manjula. S

3D Translation

- We translate a 3D point by adding translation distances, t_x, t_y and t_z, to the original coordinate position (x, y, z):

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

- Alternatively, translation can also be specified by the transformation matrix in the following formula:

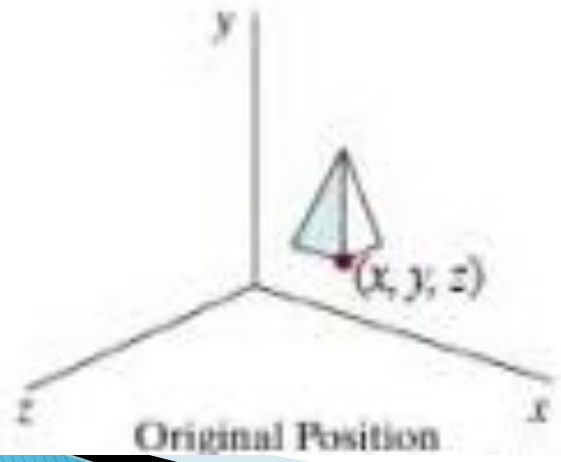
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

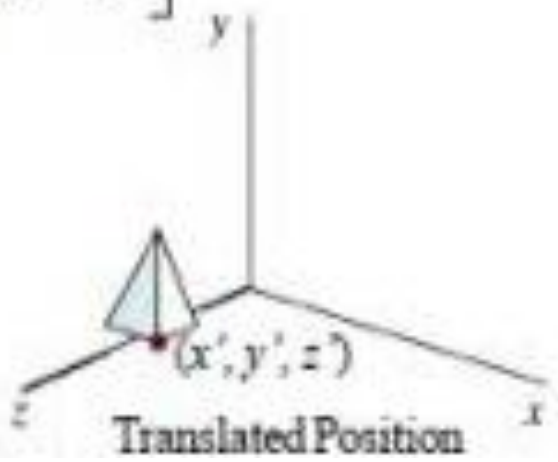
Translations In 3-D

To translate a point in three dimensions by dx, dy and dz .calculate the new points as follows:

$$x' = x + dx \quad y' = y + dy \quad z' = z + dz$$

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





-By Manjula. S

3D scaling

- **Scaling With Respect to the Origin:**
  - We scale a 3D object with respect to the origin by setting the scaling factors s_x, s_y and s_z, which are multiplied to the original vertex coordinate positions (x, y, z):

$$X' = X * S_X,\ Y' = Y * S_Y,\ Z' = Z * S_Z$$

- Alternatively, this scaling can also be specified by the transformation matrix in the following formula:

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

-By Manjula. S

3d scaling

- The equations for scaling :

$$x' = x \cdot sx$$

$$S_{sx,sy,sz} \rightarrow y' = y \cdot sy$$

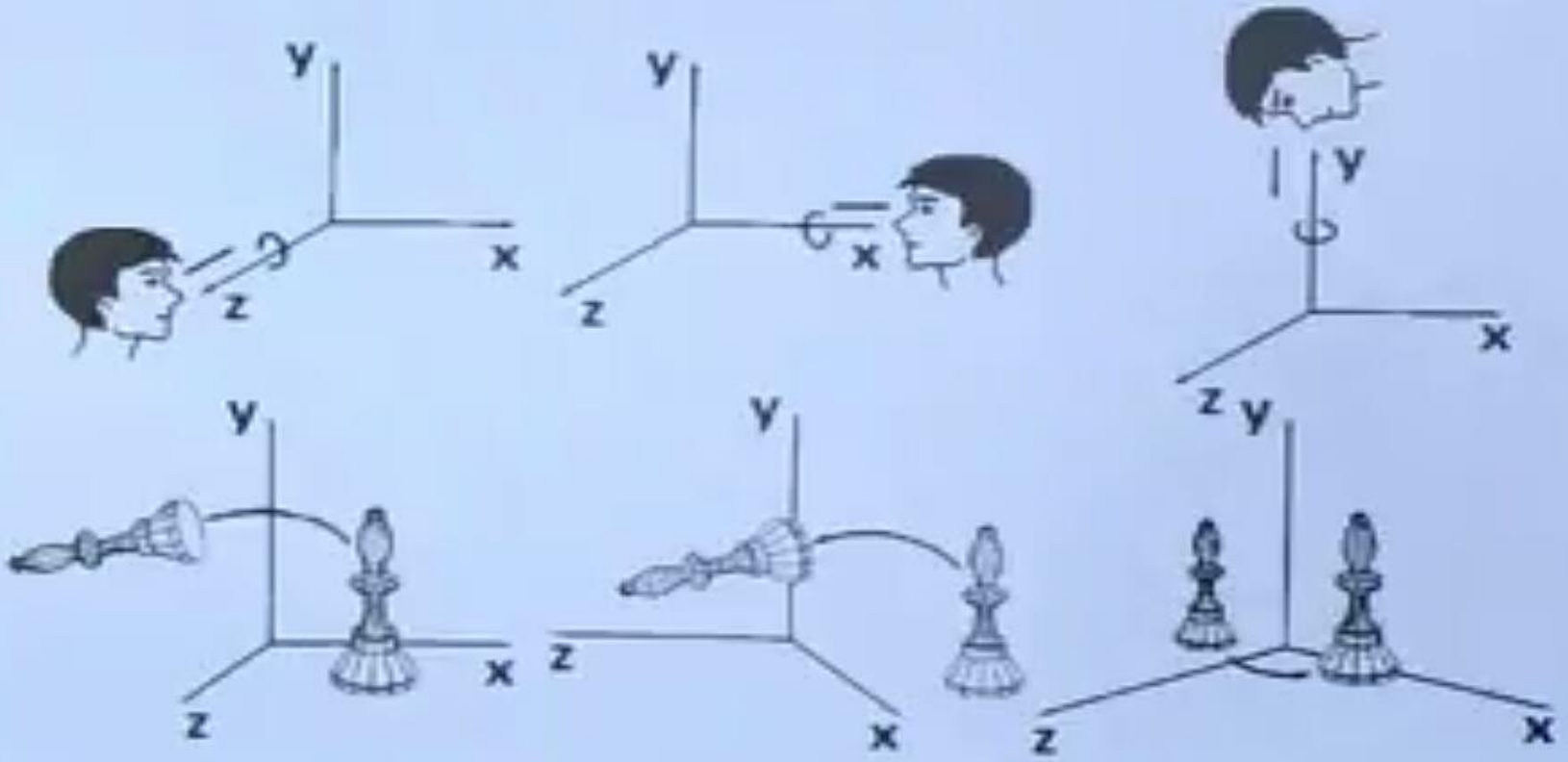
$$z' = z \cdot sz$$

fig name: After scaling

-By Manjula. S

3D Rotation

- **Coordinate-Axes Rotations:**
  - A 3D rotation can be specified around any line in space. The easiest rotation axes to handle are the coordinate axes.



-By Manjula. S

3D Rotation

- **3D Rotation matrix about the Z-axis is :**

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

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

$z' = z$

or

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

-By Manjula. S

3D Rotation

- **3D Rotation matrix about the X-axis is :**

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

$z' = y \sin\theta + z \cos\theta,$ and

$x' = x$

or

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

-By Manjula. S

3D Rotation

- **3D Rotation matrix about the Y-axis is :**

$z' = z \cos\theta - x \sin\theta$,

$x' = z \sin\theta + x \cos\theta$, and

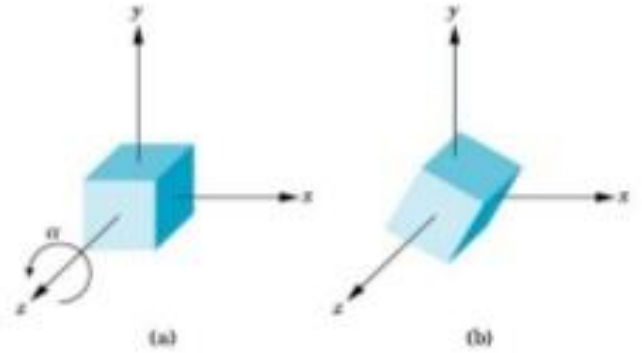
$y' = y$

or

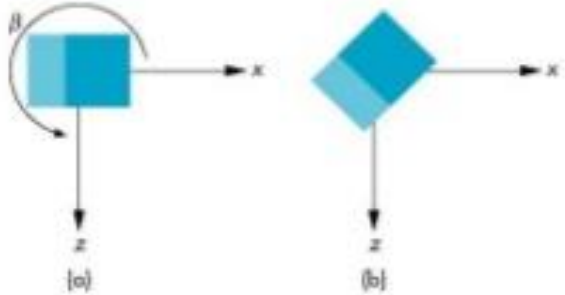
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

-By Manjula. S

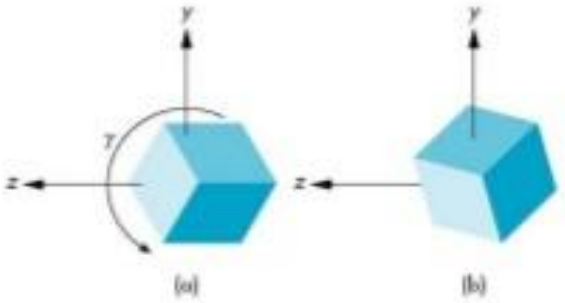
General rotation



Rotation of a cube about the z-axis. The cube is show (a) before rotation, and (b) after rotation



Rotation of a cube about the y-axis

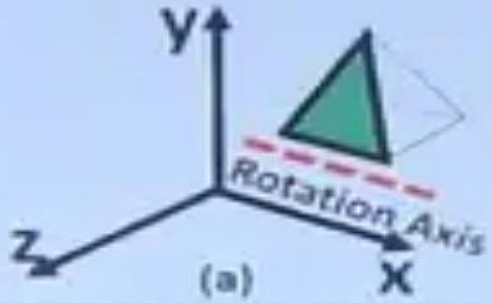


Rotation of a cube about the x-axis

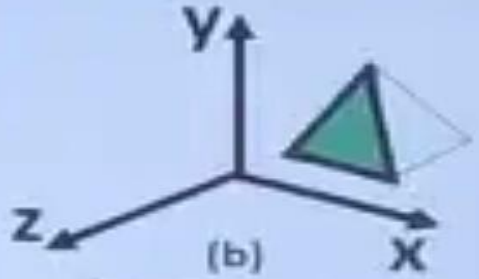
-By Manjula. S

3D Rotation About a Line

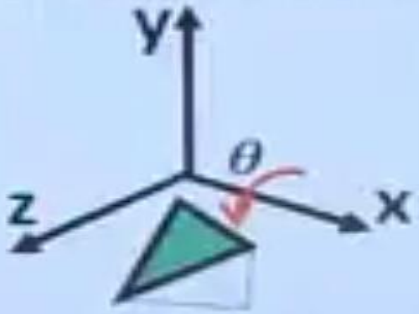
3D Rotation About a Line Which is Parallel to an Axis:



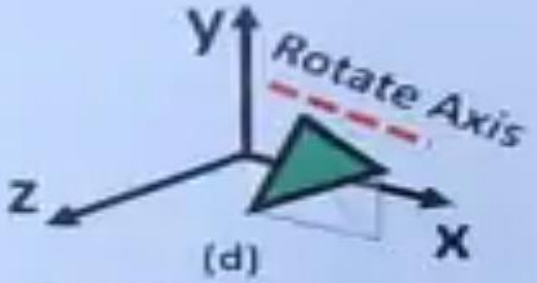
(a)
Original Position of Object



(b)
Translate Rotation Axis onto X Axis



(c)
Rotate Object Through Angle θ



(d)
Translate Rotation Axis to Original Position

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

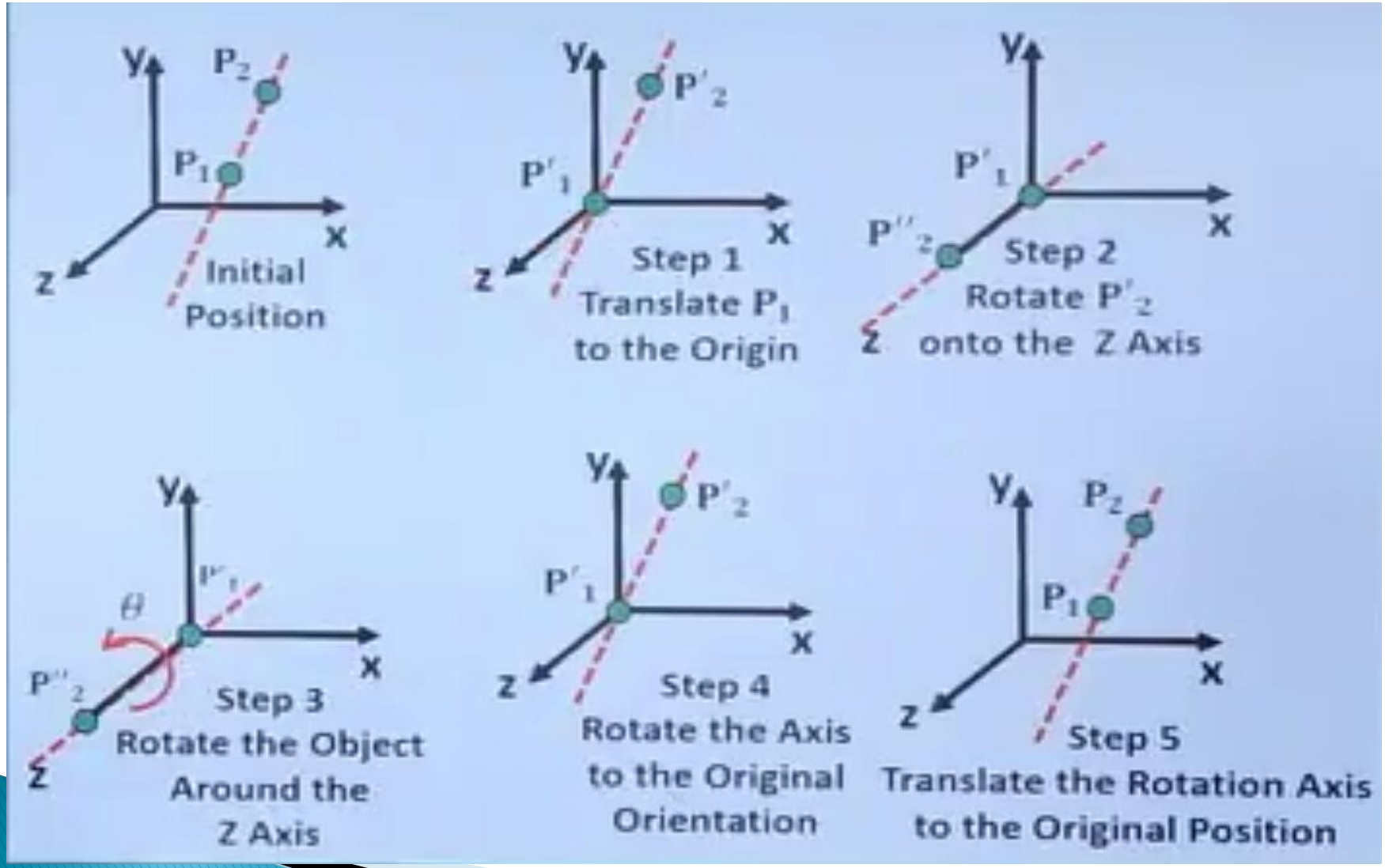
-By Manjula. S

General 3D Rotation

- **Step 1.** Translate the object so that the rotation axis passes through the coordinate origin.
- **Step 2.** Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
- **Step 3.** Perform the specified rotation about that coordinate axis.
- **Step 4.** Rotate the object so that the rotation axis is brought back to its original orientation.
- **Step 5.** Translate the object so that the rotation axis is brought back to its original position.

-By Manjula. S

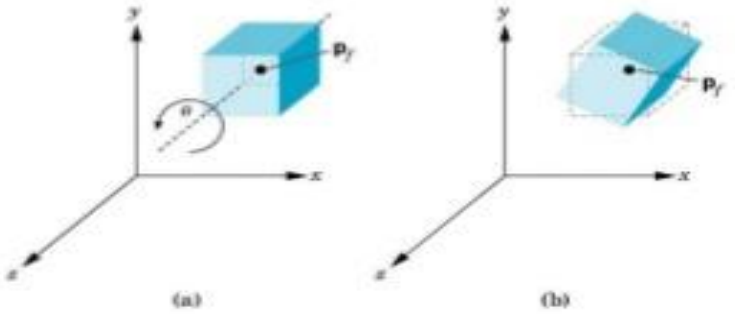
General 3D Rotation



-By Manjula. S

General 3D Rotation

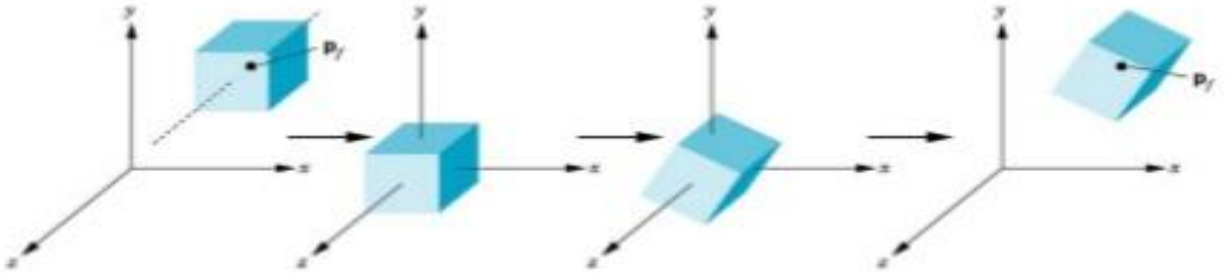
Derive example of M: rotation about a fixed point



Rotation of a cube about its center

<- This is what we try to do.

These are process that we choose to do



Sequence of transformation

-By Manjula. S

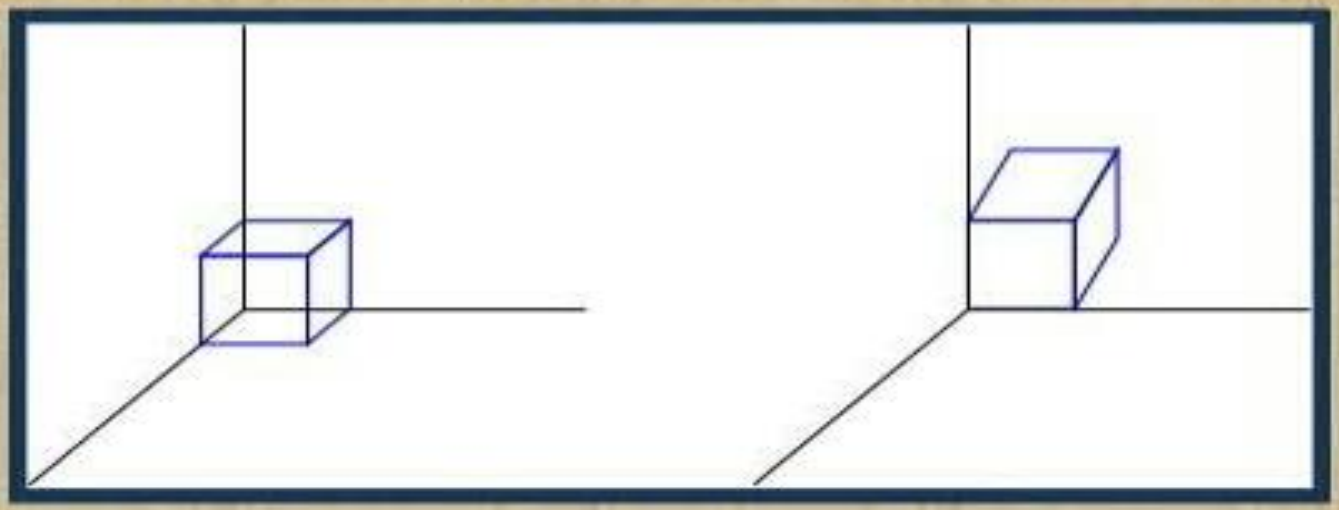
3D Shearing

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 shearing.

-By Manjula. S

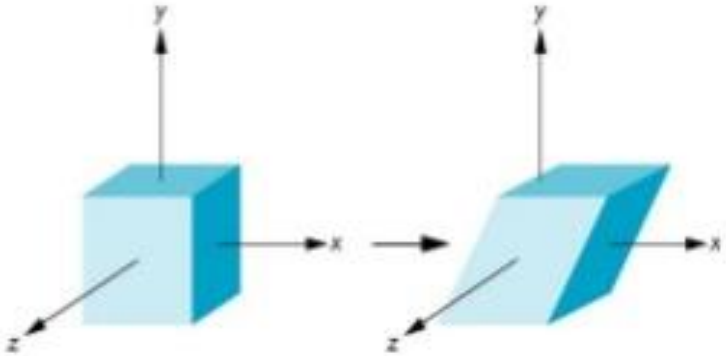
3D Shearing

- Modify object shapes
- Useful for perspective projections:
  - E.g. draw a cube (3D) on a screen (2D)
  - Alter the values for **x** and **y** by an amount proportional to the distance from z_{ref}



Shear

- Let pull in top right edge and bottom left edge
  - Neither y nor z are changed
  - Call *x* shear direction



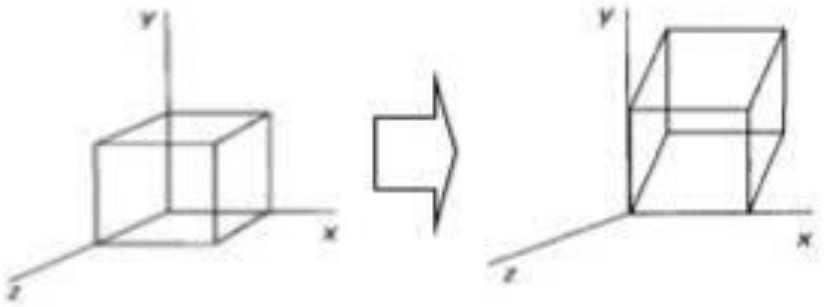
Shear

-By Manjula. S

3D Shearing

- **Shearing transformation** are used to modify the shape of the object and they are useful in 3-D viewing for obtaining **General Projection transformations.**
- **Z-axis 3-D Shear transformation**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- The effect of this transformation matrix is to alter the x and y co-ordinate values by an amount that is proportional to the z-value, while leaving z co-ordinate unchanged. Boundaries of the plane that are perpendicular to z-axis are thus shifted proportional to z-value.

Reflections

- The matrix expression for the reflection transformation of a position P = (x, y, z) relative to x-y plane can be written as:

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

- Transformation matrices for inverting x and y values are defined similarly, as reflections relative to yz plane and xz plane, respectively.

3D Reflection

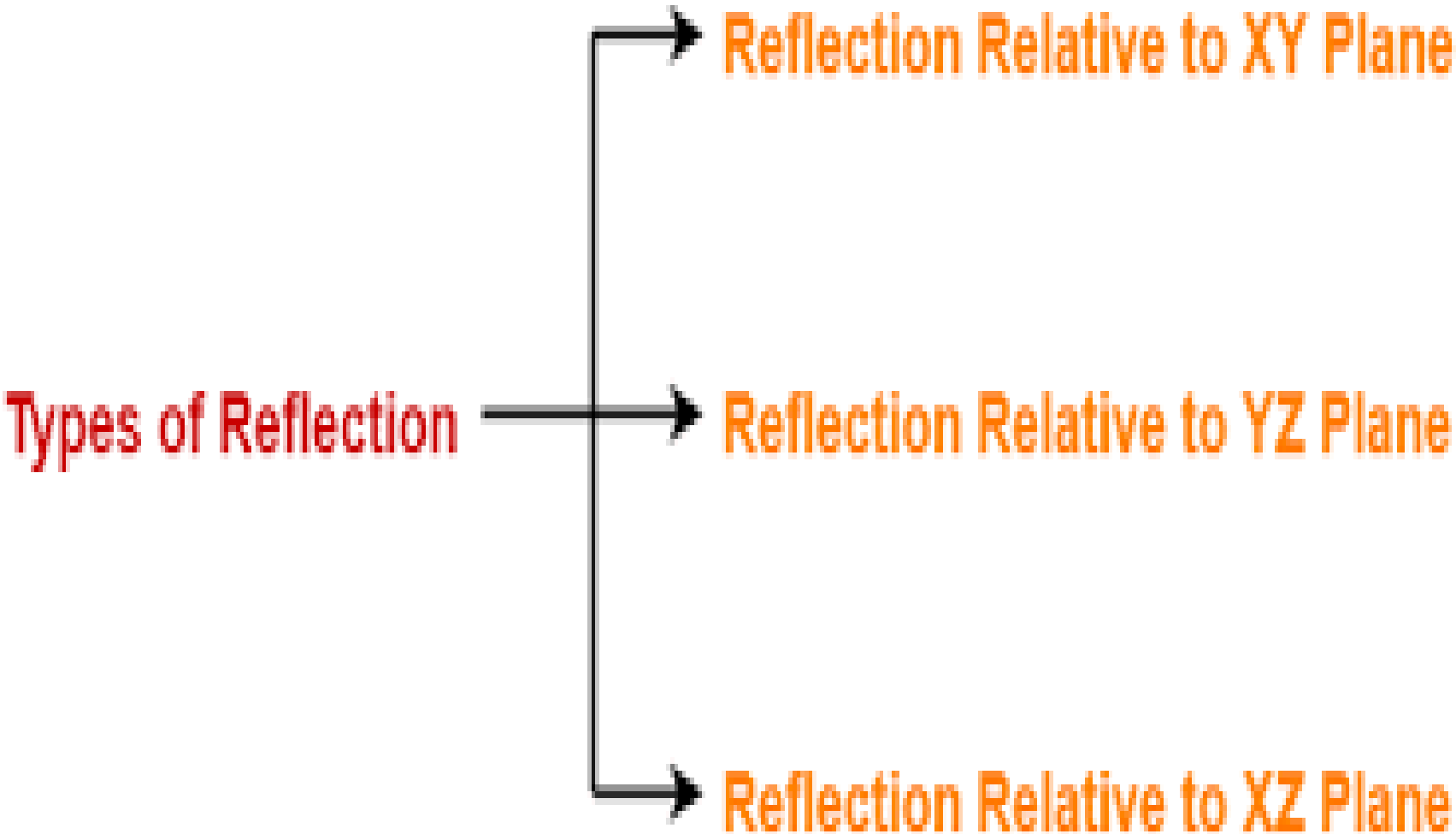
$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Reflection Matrix

(Reflection Relative to XY plane)

-By Manjula. S

3D Reflection



-By Manjula. S

3D reflection

- The matrix for reflection about y-axis:-

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

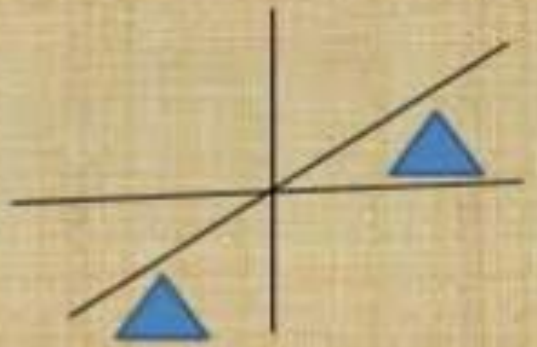


Fig: Z axis reflection

- **Reflection about z-axis:-**

$x' = -x \quad y' = -y \quad z' = z$

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

3d reflection

Reflection about x-axis:-

$x'=x \quad y'=-y \quad z'=-z$

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

Fig: X axis reflection

Reflection about y-axis:-

$y'=y \quad x'=-x \quad z'=-z$

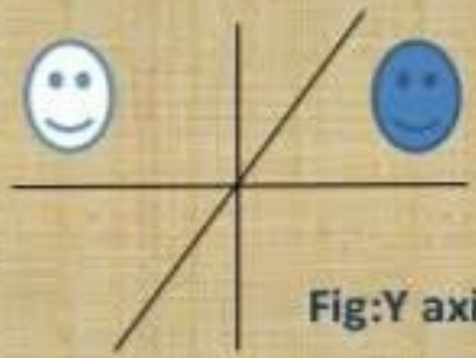
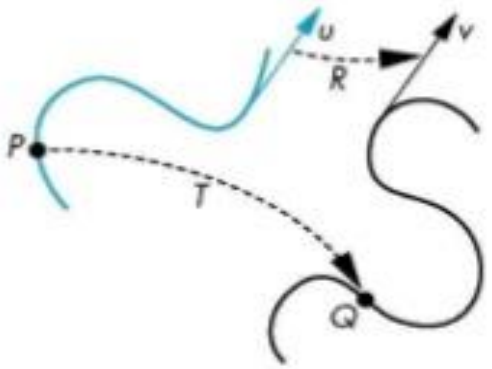


Fig:Y axis reflection

-By Manjula. S

Affine transformation

- Transformation:
  - A function that takes a point (or vector) and maps that points (or vector) in to another point (or vector)



Transformation

Affine Transformation

Affine transformation is a linear mapping method that preserves points, straight lines, and planes. Sets of parallel lines remain parallel after an **affine** transformation. The **affine** transformation technique is typically used to correct for geometric distortions or deformations that occur with non-ideal camera angles.

-By Manjula. S

Affine Transformation

- Affine Transformation (in 2D, 3D or higher dimension) have general properties that parallel lines are transformed into parallel lines and finite points.
- Translation, scaling, rotation, reflection and shearing are examples of affine transformations.
- Another example of an Affine transformation is the conversion of coordinate descriptions for a scene from one reference system to another because this transformation can be described as a combination of translation and rotation.
- An affine transformation involving only translation, rotation and reflection preserves angles and lengths, as well as parallel lines. For each of these three transformations, line lengths and angle between any two lines remain the same after the transformation.

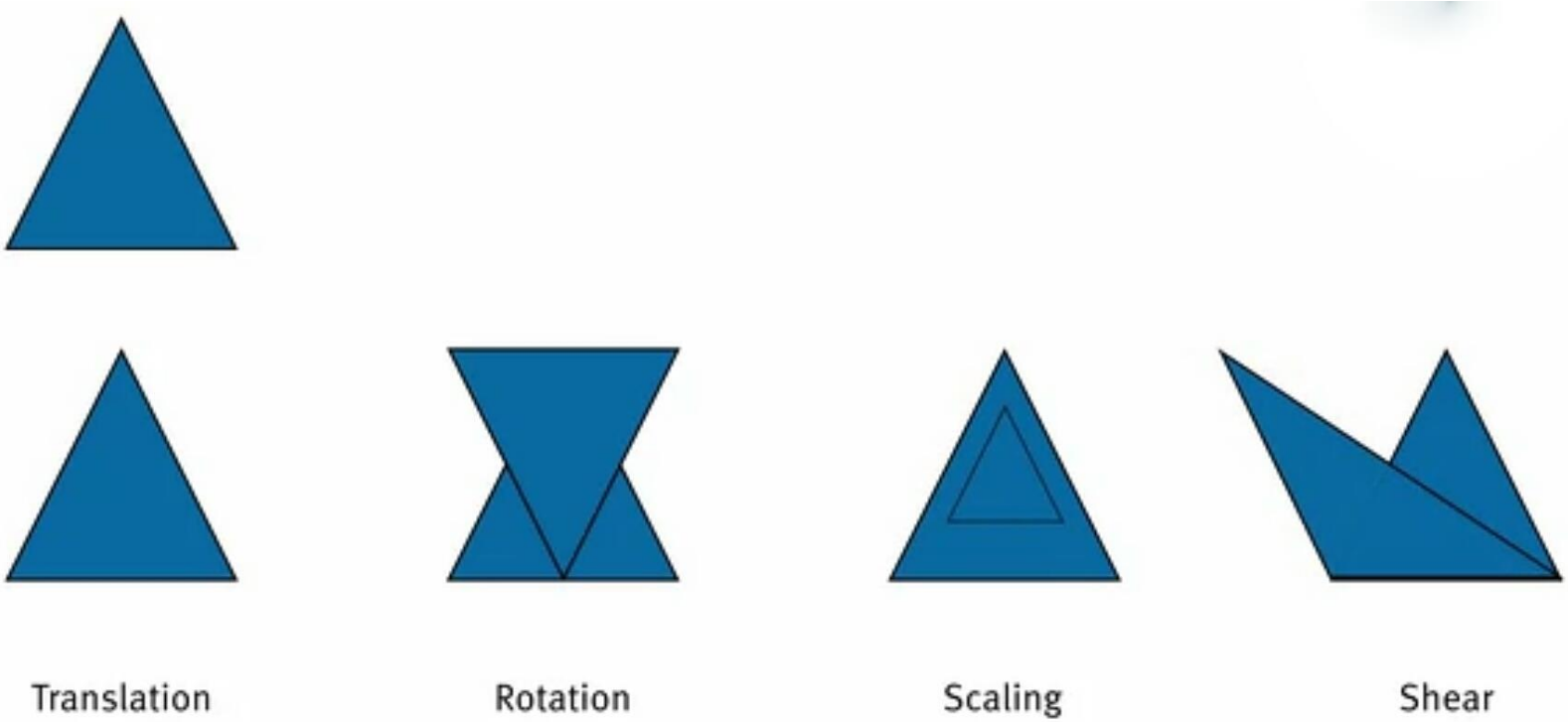
-By Manjula. S

Affine Transformations

A coordinate transformation of the form

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

is called an **affine transformation.** Each of the transformed coordinates x', y', and z' is a linear function of the original coordinates x, y, and z, and parameters a_{ij} and b_k are constants determined by the transformation type.



-By Manjula. S

Affine Transformations

- Line preserving
- Characteristic of many physically important transformations
  - Rigid body transformations: rotation, translation
  - Scaling, shear
- Importance in graphics is that we need only transform endpoints of line segments and let implementation draw line segment between the transformed endpoints

Affine Transformation

- **Affine Transformations are combinations of …**
  - Linear transformation
  - **Translation**

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- **Properties of affine transformations**
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel line remain parallel
  - Ratios are preserved
  - Closed under composition

-By Manjula. S

Concatenation of Transformation

Concatenation combines two affine **transformation** matrices by multiplying them together. You might perform several concatenations in order to create a single affine **transform** that contains the cumulative effects of several **transformations**.

-By Manjula. S

Concatenation of transformation

Application of transformations one at a time

The sequence of transformation is

q=CBAp

It can be seen that one do first must be near most to input

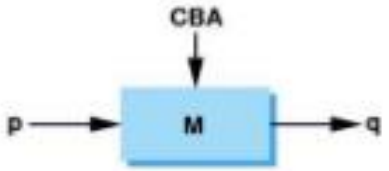
We may proceed in two steps

Calculate total transformation matrix:

M=CBA

Matrix operation :

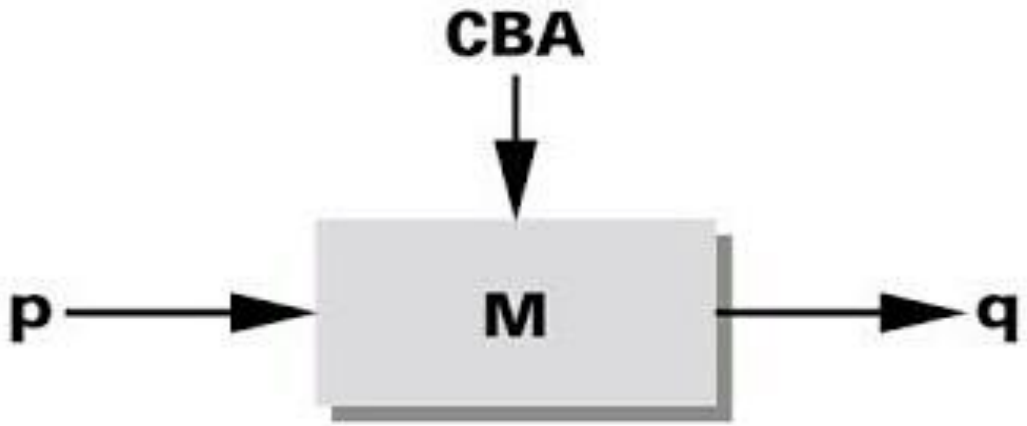
q=Mp



Pipeline transformation

-By Manjula. S

Concatenation of Transformations



-By Manjula. S

Concatenation.

The matrix product $T(d_{x1}, d_{y1}) \cdot T(d_{x2}, d_{y2})$ is:

$$\begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1}+d_{x2} \\ 0 & 1 & d_{y1}+d_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix product is variously referred to as compounding, concatenation, or composition.

This single matrix is called the Coordinate Transformation Matrix or CTM.

-By Manjula. S

OpenGL Functions

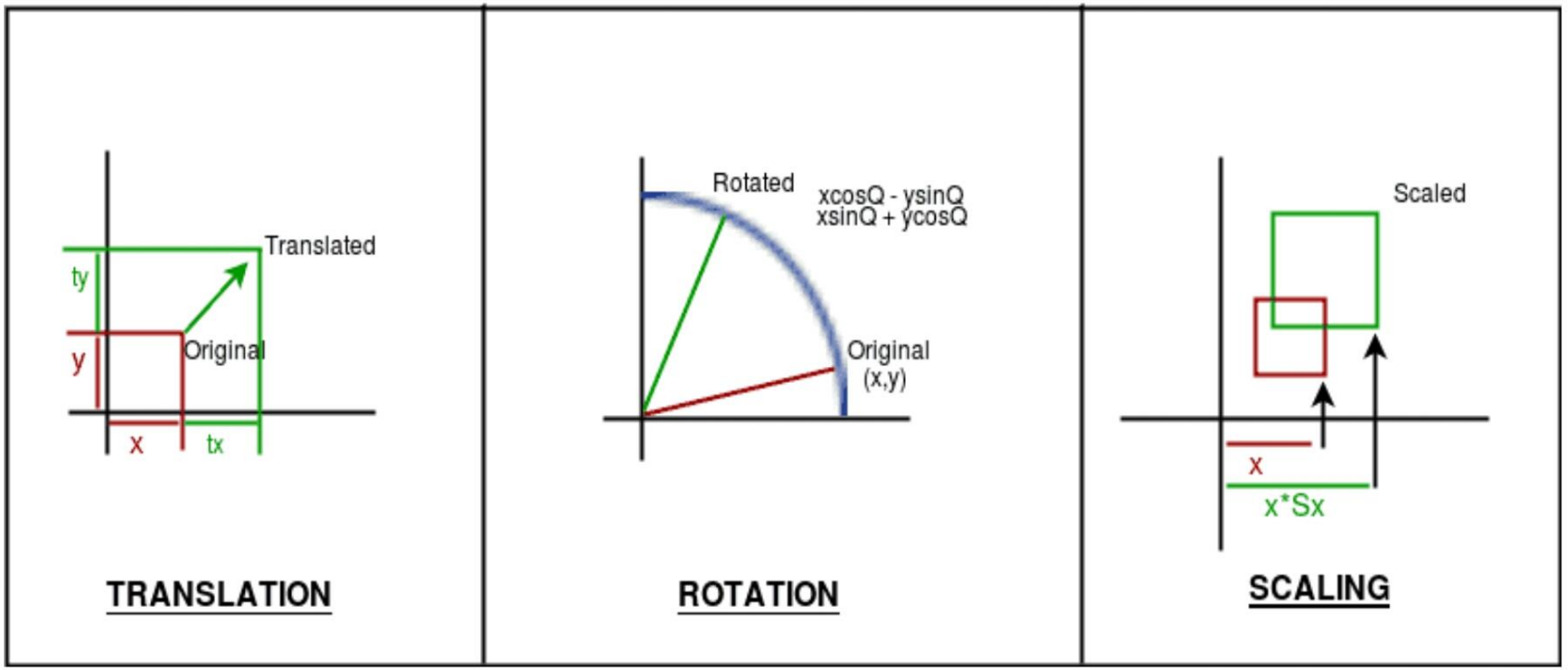
- Transformations in OpenGL are not drawing commands. They are retained as part of the graphics state.
- When drawing commands are issued, the current transformation is applied to the points drawn.
- Transformations are cumulative.

Transformations in OpenGL

- Translate
- Rotate
- Scale

- Push Matrix
- Pop Matrix

Transformation in OpenGL



-By Manjula. S

Translation

Offset (tx, ty, tz) is applied to all subsequent coordinates. Effectively moves the origin of coordinate system.

- $x' = x + \text{tx}\ , \quad y' = y + \text{ty}, \quad z' = z + \text{tz}$
- OpenGL function is `glTranslate`
- `glTranslatef`(tx, ty, tz);

EXAMPLE OF TRANSLATION

```
// this function will translate the point
void translatePoint(int px, int py, int tx, int ty)
{
    int fx = px, fy = py;
    while (1) {
        glClear(GL_COLOR_BUFFER_BIT);

        // update
        px = px + tx;
        py = py + ty;
```

Rotation

Expressed as rotation through angle θ about an axis direction (x,y,z) .

- OpenGL function – `glRotatef` (θ, x,y,z). So
  `glRotatef`(30.0, 0.0, 1.0, 0.0)
  rotates by 30° about y-axis.
- Note carefully:
  - `glRotate` wants angles in degrees.
  - C math library (sin, cos etc.) wants angles in radians.
  - $degs = rads * 180/\pi$; $rads = degs * \pi / 180$
- Positive angle? Right hand rule: if the thumb points along the vector of rotation, a positive angle has the fingers curling towards the palm.

Rotation (cont.)

- Frequently the axis is one of the coordinate axes. Common terms:
  - rotation about *y*-axis is heading/yaw
  - rotation about *x*-axis is pitch/elevation
  - rotation about *z*-axis is roll/bank
- 3-d rotation is an extremely difficult topic! There are several different mathematical formulations. Rotations do not commute – the order that transformations are done matters.

Rotation : Rotation refers to rotating a point.

Formula: X = xcosA - ysinA

Y = xsinA + ycosA,

A is the angle of rotation.

The above formula will rotate the point around the origin.

To rotate around a different point, the formula:

X = cx + (x-cx)*cosA - (y-cy)*sinA,

Y = cx + (x-cx)*sinA + (y-cy)*cosA,

cx, cy is centre coordinates,

A is the angle of rotation.

The OpenGL function is **glRotatef (A, x, y, z).**

-By Manjula. S

EXAMPLE OF ROTATION

```
void rotateAroundPt(int px, int py, int cx, int cy)
{
    float theta = 0.0;
    while (1) {
        glClear(GL_COLOR_BUFFER_BIT);
        int xf, yf;

        // update theta anticlockwise rotation
        theta = theta + thetaSpeed;
```

-By Manjula. S

Scaling

- Multiply subsequent coordinates by scale factors sx, sy, sz. (Note: these are not a point, not a vector, just 3 numbers)

  $x' = \text{sx} * x\ ,\quad y' = \text{sy} * y,\quad z' = \text{sz} * z$

- Often sx = sy = sz for a *uniform* scaling effect. If the factors are different, the scaling is called *anamorphic*.
- OpenGL function – `glScale` For example,

```
glScalef(0.5,0.5,0.5);
```

  would cause all objects drawn subsequently to be half as big.

EXAMPLE OF SCALING

```
// this function draws
void scalePoint(int px, int py, int sx, int sy)
{
    int fx, fy;
    while (1) {
        glClear(GL_COLOR_BUFFER_BIT);

        // update
        fx = px * sx;
        fy = py * sy;
```

FUNCTIONS CALL OF 3D TRANSFORMATION

```
case 1:
    translatePoint(100, 200, 1, 5);
    break;
case 2:
    rotateAroundPt(200, 200, maxWD / 2, maxHT / 2);
    // point will circle around
    // the centre of the window
    break;
case 3:
    scalePoint(10, 20, 2, 3);
    break;
```

-By Manjula. S

Order of transformations

- Transformations are cumulative and the order matters:
  - The sequence
    1. Scale 2, 2, 2
    2. Translate by (10, 0, 0)

    will scale subsequent objects by factor of 2 about an origin that is 20 along the *x*-axis
  - The sequence
    1. Rotate 90.0 deg about (0, 1, 0)
    2. Translate by (10, 0, 0)

    will set an origin 10 along the –ve *z*-axis
- For each object, the usual sequence is:
  1. Translate (move the origin to the right location)
  2. Rotate (orient the coordinate axes right)
  3. Scale (get the object to the right size)

Push and Pop

- `glMatrixMode(GL_MODELVIEW)`
- `glMatrixMode(GL_PROJECTION)`
- `glMatrixMode(GL_TEXTURE)`

- **`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.

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.

Thank You

-By Manjula. S