Three-Dimensional Geometric Transformations

GKV-2020

#Chapter 7

1 Three-Dimensional Translation

A position $\mathbf{P} = (x, y, z)$ in three-dimensional space is translated to a location $\mathbf{P}' = (x', y', z')$ by adding translation distances t_x, t_y, and t_z to the Cartesian coordinates of $\mathbf{P}$:

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z \tag{1}$$

We can express these three-dimensional translation operations in matrix form. But now the coordinate positions, $\mathbf{P}$ and $\mathbf{P}'$, are represented in homogeneous coordinates with four-element column matrices, and the translation operator $\mathbf{T}$ is a 4×4 matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

Matrix for translation

$$\left\{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{matrix}\right\} \text{ or } \left\{\begin{matrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{matrix}\right\}$$

Matrix representation of point translation

Point shown in fig is (x, y, z). It become (x^1, y^1, z^1) after translation. T_x T_y T_z are translation vector.

$$\begin{pmatrix} x^1 \\ y^1 \\ z^1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

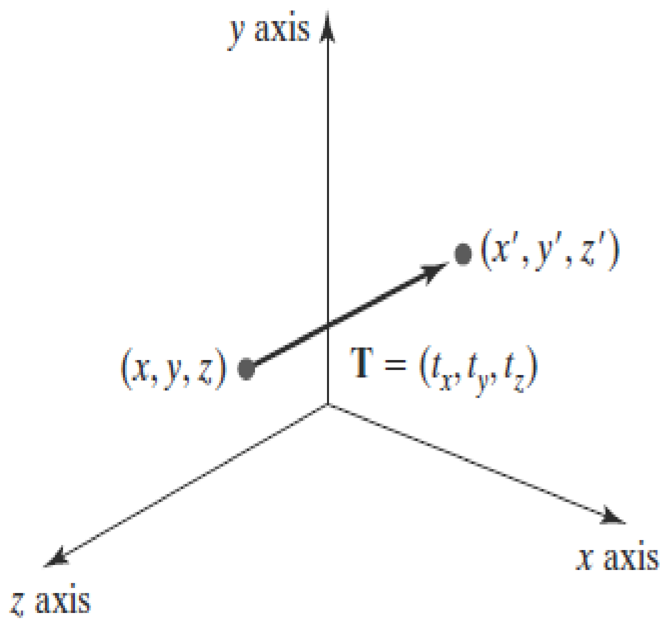


FIGURE 1
Moving a coordinate position with translation vector $\mathbf{T} = (t_x, t_y, t_z)$.

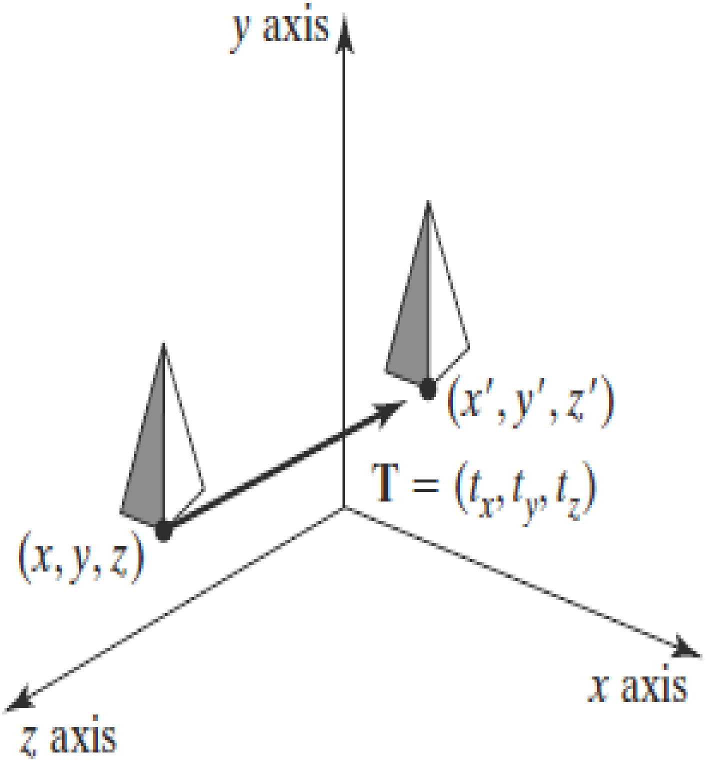
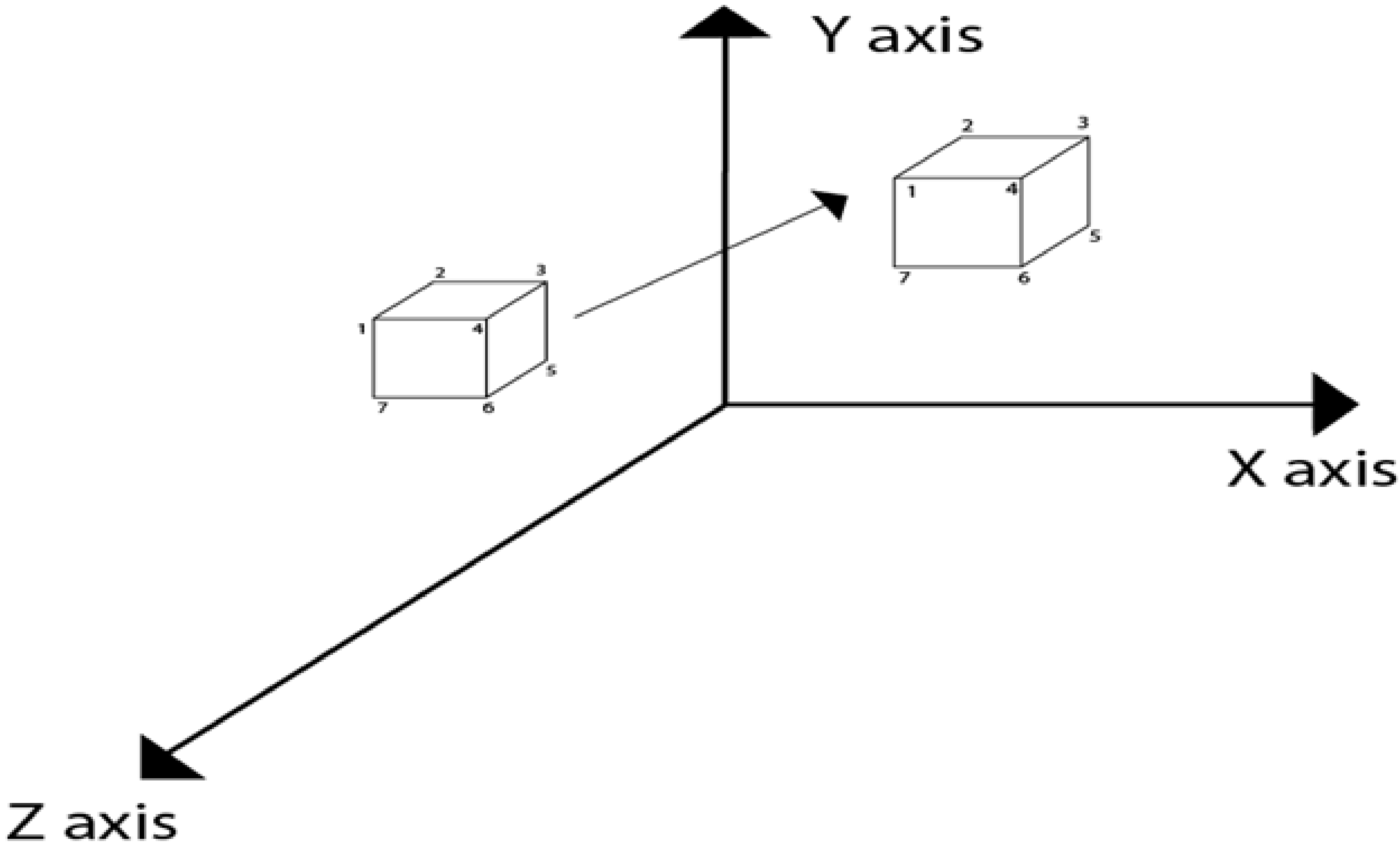


FIGURE 2
Shifting the position of a three-dimensional object using translation vector **T**.


Y axis
X axis
Z axis
1
2
3
4
5
6
7
1
2
3
4
5
6
7

Example

Example: A point has coordinates in the x, y, z direction i.e., (5, 6, 7). The translation is done in the x-direction by 3 coordinate and y direction. Three coordinates and in the z- direction by two coordinates. Shift the object. Find coordinates of the new position.

Solution: Co-ordinate of the point are (5, 6, 7)

Translation vector in x direction = 3

Translation vector in y direction = 3

Translation vector in z direction = 2

Translation matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Multiply co-ordinates of point with translation matrix

$$(x^1 y^1 z^1) = (5,6,7,1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 3 & 2 & 1 \end{pmatrix}$$

= [5+0+0+30+6+0+30+0+7+20+0+0+1] = [8991]

x becomes x^1=8

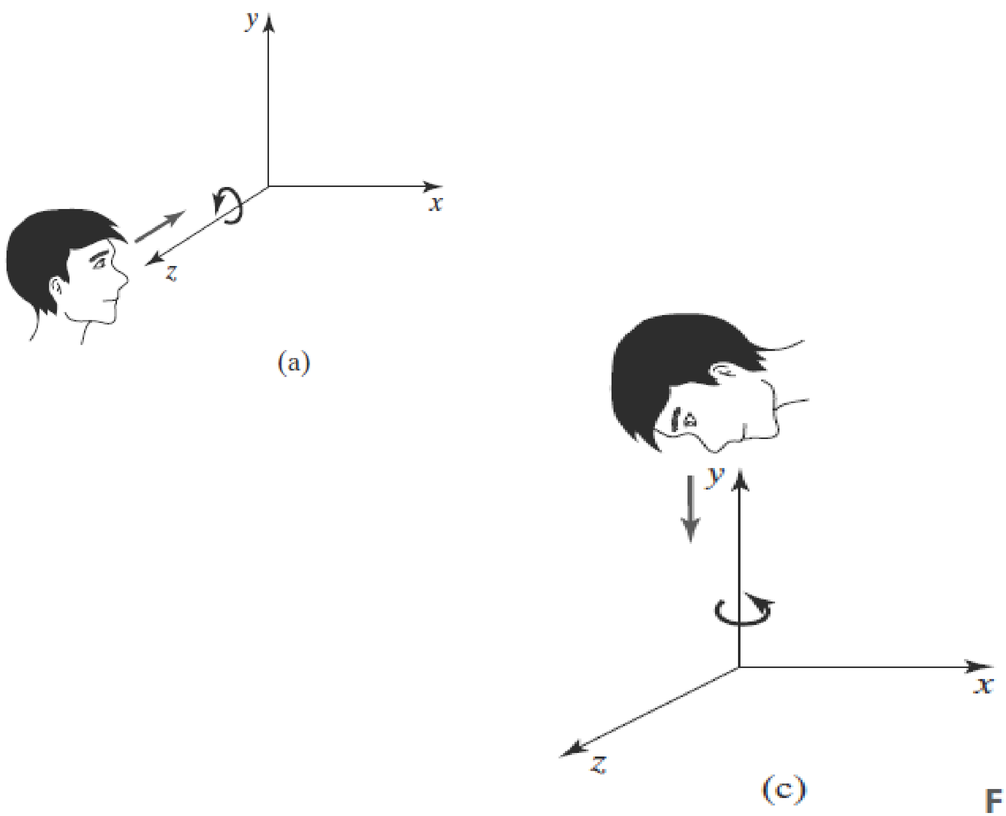
y becomes y^1=9

z becomes z^1=9

```
typedef GLfloat Matrix4x4 [4][4];

/* Construct the 4 x 4 identity matrix. */
void matrix4x4SetIdentity (Matrix4x4 matIdent4x4)
{
   GLint row, col;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matIdent4x4 [row][col] = (row == col);
}

void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)
{
   Matrix4x4 matTransl3D;

   /*  Initialize translation matrix to identity.  */
   matrix4x4SetIdentity (matTransl3D);

   matTransl3D [0][3] = tx;
   matTransl3D [1][3] = ty;
   matTransl3D [2][3] = tz;
}
```

2 Three-Dimensional Rotation

We can rotate an object about any axis in space, but the easiest rotation axes to handle are those that are parallel to the Cartesian-coordinate axes. Also, we can use combinations of coordinate-axis rotations (along with appropriate translations) to specify a rotation about any other line in space. Therefore, we first consider the operations involved in coordinate-axis rotations, then we discuss the calculations needed for other rotation axes.



F I G U R E 3
Positive rotations about a coordinate axis are counterclockwise, when looking along the positive half of the axis toward the origin.

Three-Dimensional Coordinate-Axis Rotations

The two-dimensional **z-axis rotation** equations are easily extended to three dimensions, as follows:

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned} \tag{4}$$

Parameter θ specifies the rotation angle about the z axis, and z-coordinate values are unchanged by this transformation. In homogeneous-coordinate form, the three-dimensional z-axis rotation equations are

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5}$$

Three-Dimensional Coordinate-Axis Rotations

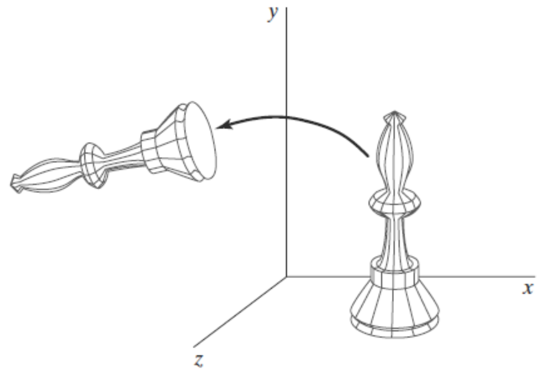


FIGURE 4
Rotation of an object about the *z* axis.

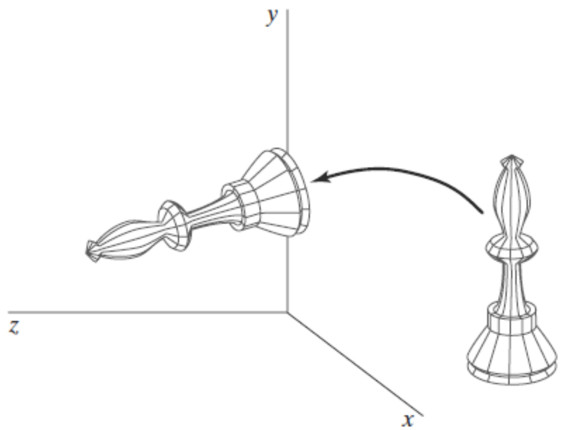


FIGURE 6
Rotation of an object about the *x* axis.

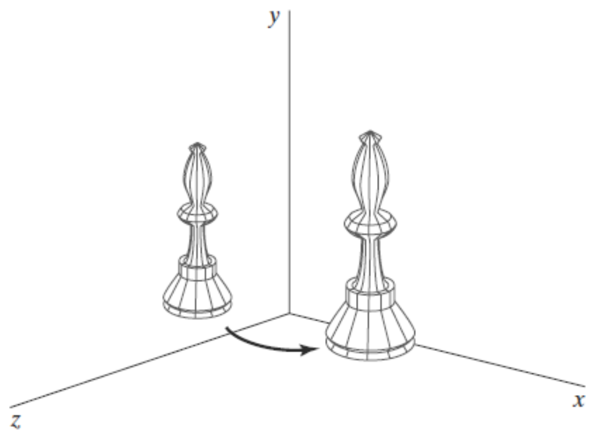


FIGURE 7
Rotation of an object about the *y* axis.

Similar for x or y axis rotation

***x*-axis rotation:**

$$
\begin{aligned}
y' &= y \cos\theta - z \sin\theta \\
z' &= y \sin\theta + z \cos\theta \\
x' &= x
\end{aligned}
\tag{8}
$$

***y*-axis rotation:**

$$
\begin{aligned}
z' &= z \cos\theta - x \sin\theta \\
x' &= z \sin\theta + x \cos\theta \\
y' &= y
\end{aligned}
\tag{9}
$$

All of 3 axis rotations

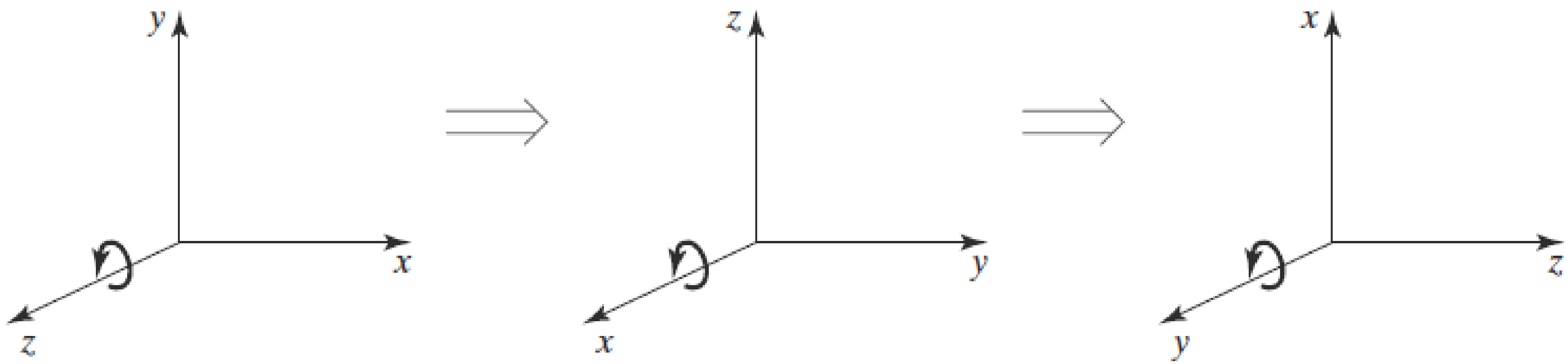


FIGURE 5
Cyclic permutation of the Cartesian-coordinate axes to produce the three sets of coordinate-axis rotation equations.

Routine 3-D rotation (1)

```
class wcPt3D {
   public:
      GLfloat x, y, z;
};
typedef float Matrix4x4 [4][4];

Matrix4x4 matRot;

/* Construct the 4 x 4 identity matrix. */
void matrix4x4SetIdentity (Matrix4x4 matIdent4x4)
{
   GLint row, col;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matIdent4x4 [row][col] = (row == col);
}
```

Routine 3-D rotation (2)

```
/* Premultiply matrix m1 by matrix m2, store result in m2. */
void matrix4x4PreMultiply (Matrix4x4 m1, Matrix4x4 m2)
{
   GLint row, col;
   Matrix4x4 matTemp;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
                              m2 [1][col] + m1 [row][2] * m2 [2][col] +
                              m1 [row][3] * m2 [3][col];
   for (row = 0; row < 4; row++)
      for (col = 0; col < 4; col++)
         m2 [row][col] = matTemp [row][col];
}
```

Routine 3-D rotation (3)

```
void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)
{
   Matrix4x4 matTransl3D;

   /*  Initialize translation matrix to identity.  */
   matrix4x4SetIdentity (matTransl3D);

   matTransl3D [0][3] = tx;
   matTransl3D [1][3] = ty;
   matTransl3D [2][3] = tz;
```

```
    /*  Concatenate translation matrix with matRot.  */
    matrix4x4PreMultiply (matTransl3D, matRot);
 }
```

Routine 3-D rotation (4)

```
void rotate3D (wcPt3D p1, wcPt3D p2, GLfloat radianAngle)
{
   Matrix4x4 matQuaternionRot;

   GLfloat axisVectLength = sqrt ((p2.x - p1.x) * (p2.x - p1.x) +
                         (p2.y - p1.y) * (p2.y - p1.y) +
                         (p2.z - p1.z) * (p2.z - p1.z));
   GLfloat cosA = cos (radianAngle);
   GLfloat oneC = 1 - cosA;
   GLfloat sinA = sin (radianAngle);
   GLfloat ux = (p2.x - p1.x) / axisVectLength;
   GLfloat uy = (p2.y - p1.y) / axisVectLength;
   GLfloat uz = (p2.z - p1.z) / axisVectLength;
```

Routine 3-D rotation (5)

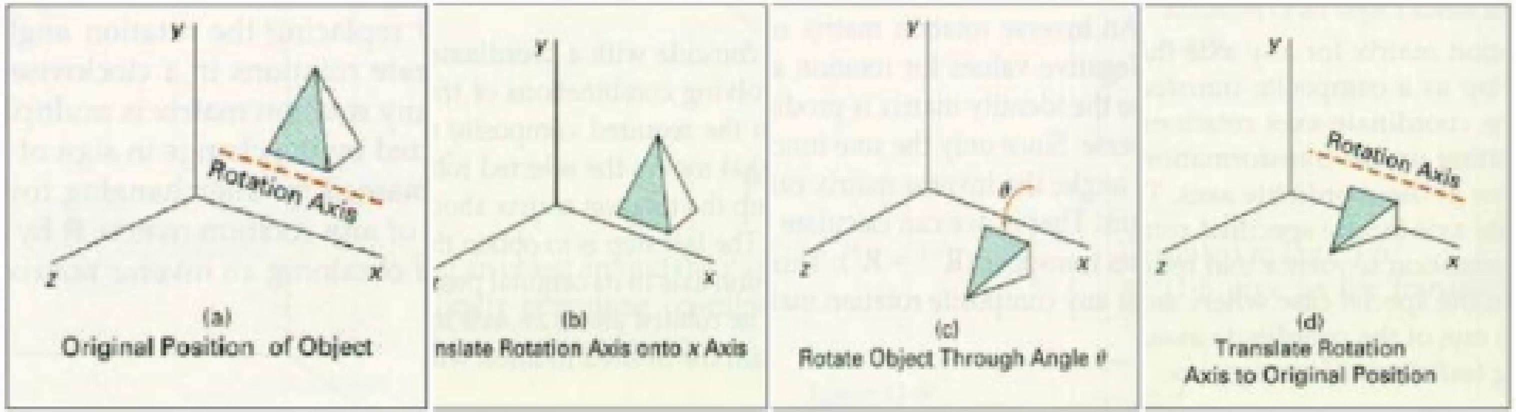
```
    /*  Set up translation matrix for moving p1 to origin.  */
    translate3D (-p1.x, -p1.y, -p1.z);

    /*  Initialize matQuaternionRot to identity matrix.  */
    matrix4x4SetIdentity (matQuaternionRot);

    matQuaternionRot [0][0] = ux*ux*oneC + cosA;
    matQuaternionRot [0][1] = ux*uy*oneC - uz*sinA;
    matQuaternionRot [0][2] = ux*uz*oneC + uy*sinA;
    matQuaternionRot [1][0] = uy*ux*oneC + uz*sinA;
    matQuaternionRot [1][1] = uy*uy*oneC + cosA;
    matQuaternionRot [1][2] = uy*uz*oneC - ux*sinA;
    matQuaternionRot [2][0] = uz*ux*oneC - uy*sinA;
    matQuaternionRot [2][1] = uz*uy*oneC + ux*sinA;
    matQuaternionRot [2][2] = uz*uz*oneC + cosA;

    /*  Combine matQuaternionRot with translation matrix.  */
    matrix4x4PreMultiply (matQuaternionRot, matRot);

    /*  Set up inverse matTransl3D and concatenate with
     *  product of previous two matrices.
     */
    translate3D (p1.x, p1.y, p1.z);
}
```

Routine 3-D rotation (6)

```
void displayFcn (void)
{
   /*  Input rotation parameters.  */

   /*  Initialize matRot to identity matrix:  */
   matrix4x4SetIdentity (matRot);

   /*  Pass rotation parameters to procedure rotate3D.  */

   /*  Display rotated object.  */
}
```

General 3D Rotations : **CASE 1**

- **Rotation about an Axis that is Parallel to One of the Coordinate Axes**
  - **Translate** the object so that the rotation axis coincides with the parallel coordinate axis
  - Perform the specified **rotation** about that axis
  - **Translate** the object so that the rotation axis is moved back to its original position



  - Any coordinate position **P** on the object in this fig. is transformed with the sequence shown as below

$$\mathbf{P}' = T^{-1} \cdot \mathbf{R}_x(\theta) \cdot T \cdot \mathbf{P}$$

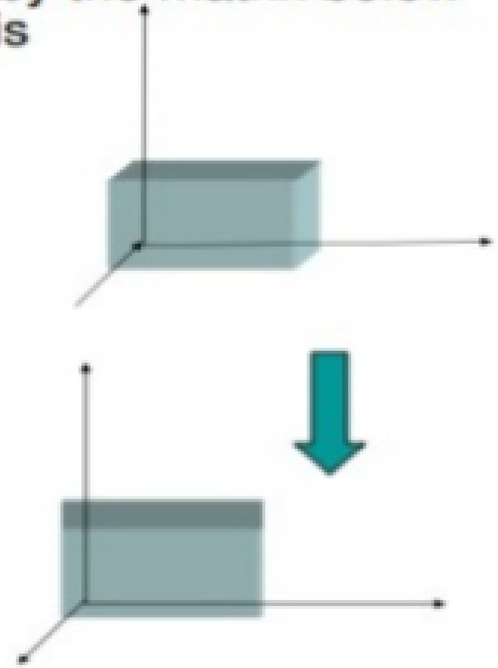
3D rotation Example

- Example – for the RPP given by the matrix below obtain -90 rotation about x -axis

$$[X]=\begin{bmatrix} 0 & 3 & 3 & 0 & 0 & 3 & 3 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[T_{Rx}]=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[X']=\begin{bmatrix} 0 & 3 & 3 & 0 & 0 & 3 & 3 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & -2 & 0 & 0 & -2 & -2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



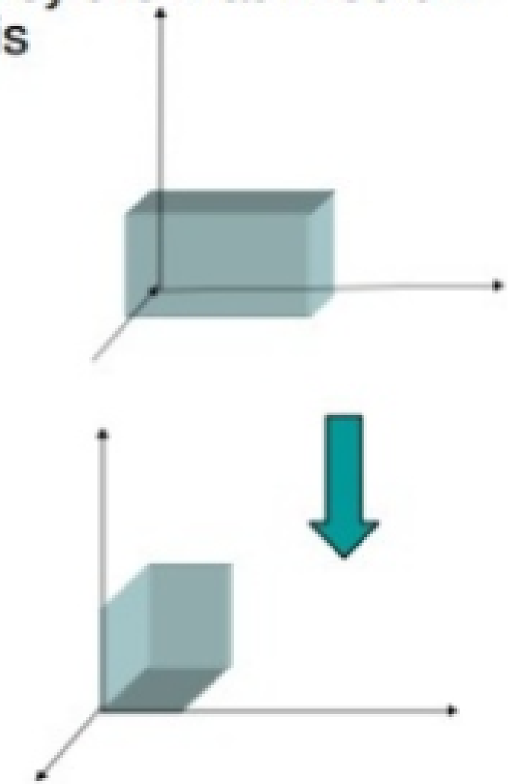
3D rotation Example

➢ Example – for the RPP given by the matrix below obtain 90 rotation about y -axis

$$[X]=\begin{bmatrix} 0 & 3 & 3 & 0 & 0 & 3 & 3 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

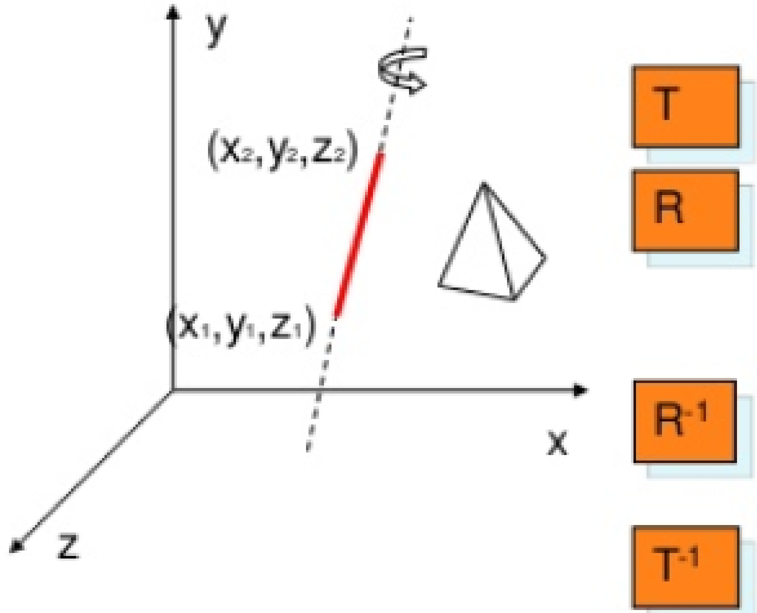
$$[T_{Ry}] = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[X'] = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & -3 & -3 & 0 & 0 & -3 & -3 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



General 3D Rotations : CASE 2

- Rotation about an Arbitrary Axis



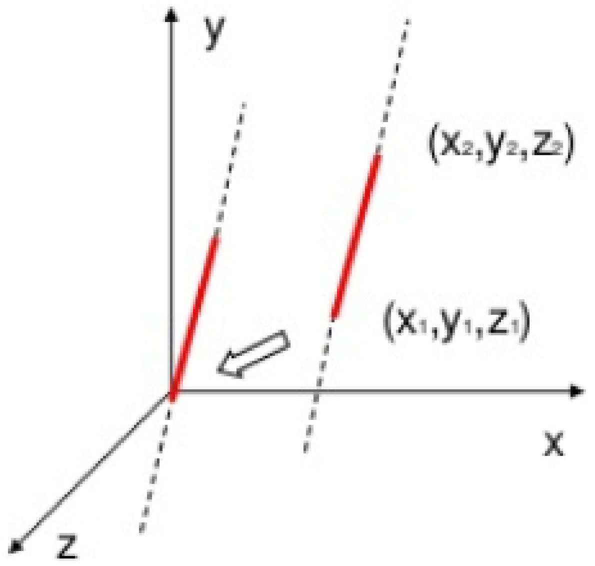
Basic Idea

1. Translate (x1, y1, z1) to the origin
2. Rotate (x'2, y'2, z'2) on to the z axis
3. Rotate the object around the z-axis
4. Rotate the axis to the original orientation
5. Translate the rotation axis to the original position

$$\mathbf{R}(\theta)=\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

General 3D Rotations

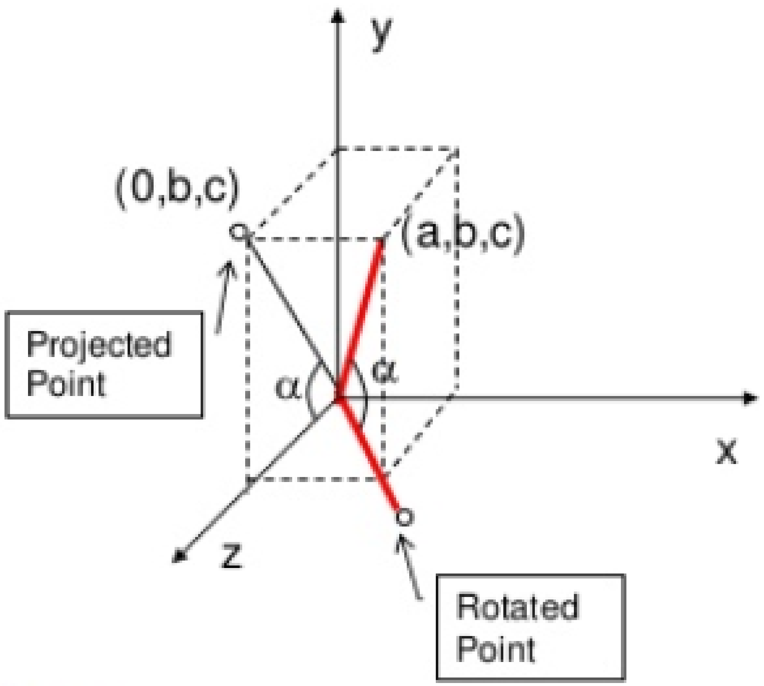
- Step 1. Translation



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General 3D Rotations

- Step 2. Establish $[\,T_R\,]^{\alpha}_{x}$ $\quad x$ axis



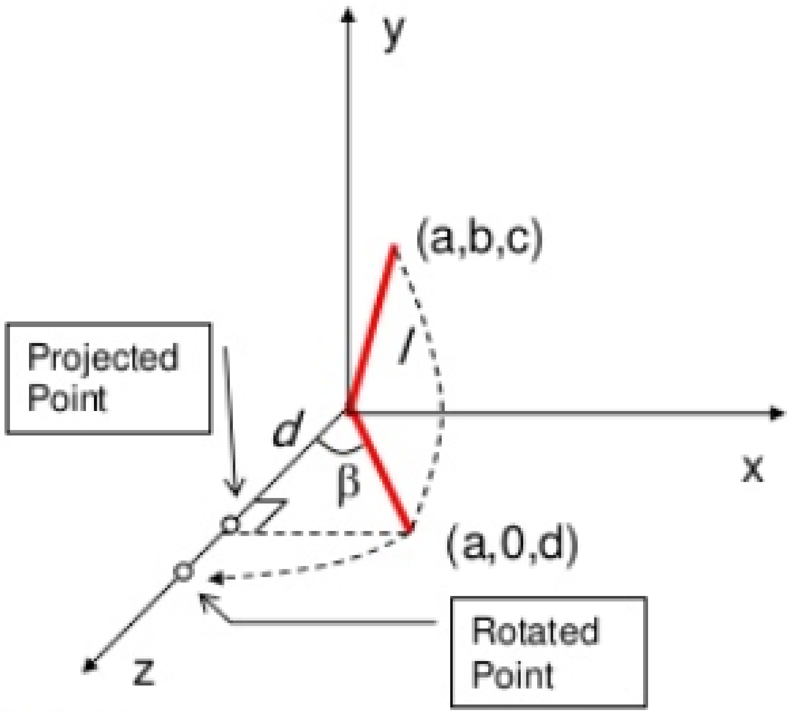
$$\sin\alpha = \frac{b}{\sqrt{b^2+c^2}} = \frac{b}{d}$$

$$\cos\alpha = \frac{c}{\sqrt{b^2+c^2}} = \frac{c}{d}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

- Step 3. Rotate about *y* axis by ϕ



$$\sin\beta = \frac{a}{l}, \quad \cos\beta = \frac{d}{l}$$

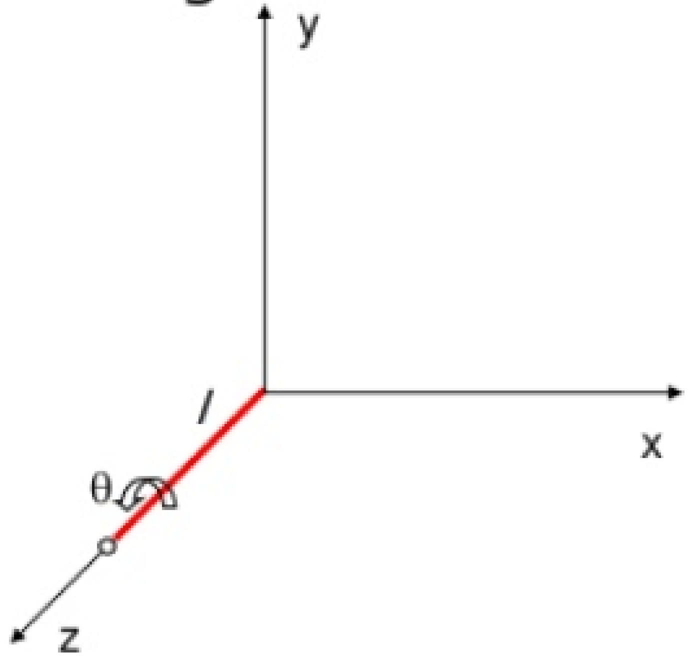
$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

$$d = \sqrt{b^2 + c^2}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & -a/l & 0 \\ 0 & 1 & 0 & 0 \\ a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

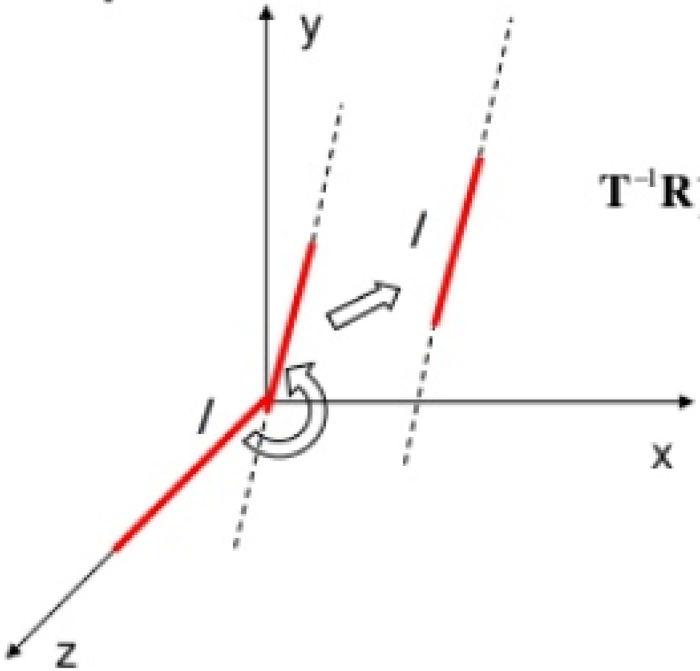
- Step 4. Rotate about *z* axis by the desired angle **θ**



$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

- Step 5. Apply the reverse transformation to place the axis back in its initial position



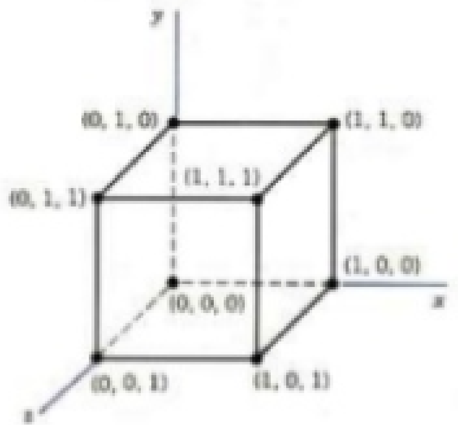
$$\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)=\begin{bmatrix}1&0&0&-x_1\\0&1&0&-y_1\\0&0&1&-z_1\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\cos\alpha&\sin\alpha&0\\0&-\sin\alpha&\cos\alpha&0\\0&0&0&1\end{bmatrix}$$

$$\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\-\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}$$

$$\mathbf{R}(\theta)=\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

Example

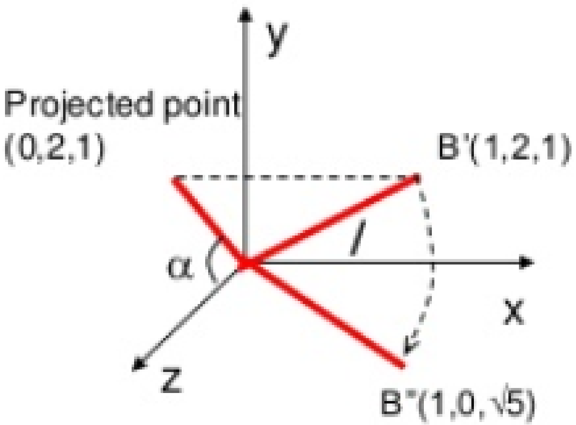
Find the new coordinates of a unit cube 90º-rotated about an axis defined by its endpoints A(2,1,0) and B(3,3,1).



A Unit Cube

Example

- Step 2. Rotate axis *A'B'* about the x axis by and angle α, until it lies on the *xz* plane.



$$\sin\alpha = \frac{2}{\sqrt{2^2+1^2}} = \frac{2}{\sqrt{5}} = \frac{2\sqrt{5}}{5}$$

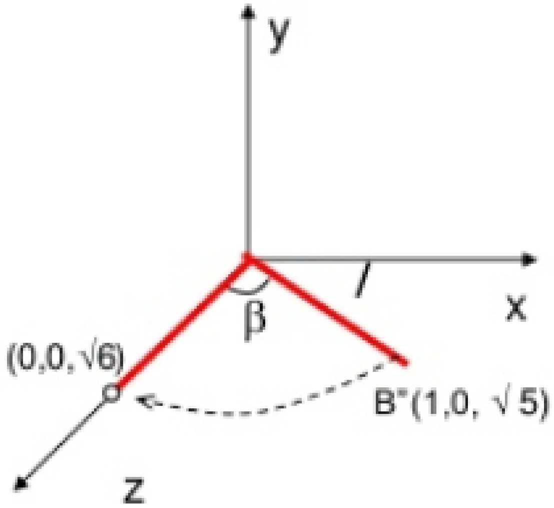
$$\cos\alpha = \frac{1}{\sqrt{5}} = \frac{\sqrt{5}}{5}$$

$$l = \sqrt{1^2+2^2+1^2} = \sqrt{6}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} & 0 \\ 0 & \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

- Step 3. Rotate axis *A'B''* about the y axis by and angle ϕ, until it coincides with the z axis.



$$\sin\beta = \frac{1}{\sqrt{6}} = \frac{\sqrt{6}}{6}$$

$$\cos\beta = \frac{\sqrt{5}}{\sqrt{6}} = \frac{\sqrt{30}}{6}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \frac{\sqrt{30}}{6} & 0 & -\frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

- Step 4. Rotate the cube 90° about the z axis

$$\mathbf{R}_z(90°)=\begin{bmatrix}0 & -1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

  - Finally, the concatenated rotation matrix about the arbitrary axis AB becomes,

$$\mathbf{R}(\theta)=\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(90°)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

Example

$$\mathbf{R}(\theta)=\begin{bmatrix}1&0&0&2\\0&1&0&1\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\frac{\sqrt{5}}{5}&\frac{2\sqrt{5}}{5}&0\\0&-\frac{2\sqrt{5}}{5}&\frac{\sqrt{5}}{5}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\frac{\sqrt{30}}{6}&0&\frac{\sqrt{6}}{6}&0\\0&1&0&0\\-\frac{\sqrt{6}}{6}&0&\frac{\sqrt{30}}{6}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&-1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$\begin{bmatrix}\frac{\sqrt{30}}{6}&0&-\frac{\sqrt{6}}{6}&0\\0&1&0&0\\\frac{\sqrt{6}}{6}&0&\frac{\sqrt{30}}{6}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\frac{\sqrt{5}}{5}&-\frac{2\sqrt{5}}{5}&0\\0&\frac{2\sqrt{5}}{5}&\frac{\sqrt{5}}{5}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&-2\\0&1&0&-1\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$=\begin{bmatrix}0.166&-0.075&0.983&1.742\\0.742&0.667&0.075&-1.151\\-0.650&0.741&0.167&0.560\\0&0&0&1\end{bmatrix}$$

Example

- Multiplying $\mathbf{R}(\theta)$ by the point matrix of the original cube

$$[\mathbf{P}'] = \mathbf{R}(\theta)\cdot[\mathbf{P}]$$

$$[\mathbf{P}'] = \begin{bmatrix} 0.166 & -0.075 & 0.983 & 1.742 \\ 0.742 & 0.667 & 0.075 & -1.151 \\ -0.650 & 0.741 & 0.167 & 0.560 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.650 & 1.667 & 1.834 & 2.816 & 2.725 & 1.742 & 1.909 & 2.891 \\ -0.558 & -0.484 & 0.258 & 0.184 & -1.225 & -1.151 & -0.409 & -0.483 \\ 1.467 & 1.301 & 0.650 & 0.817 & 0.726 & 0.560 & -0.091 & 0.076 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3 Three-Dimensional Scaling

The matrix expression for the three-dimensional scaling transformation of a position $\mathbf{P} = (x, y, z)$ relative to the coordinate origin is a simple extension of two-dimensional scaling. We just include the parameter for z-coordinate scaling in the transformation matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{41}$$

The three-dimensional scaling transformation for a point position can be represented as

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \tag{42}$$

where scaling parameters s_x, s_y, and s_z are assigned any positive values. Explicit expressions for the scaling transformation relative to the origin are

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y, \qquad z' = z \cdot s_z \tag{43}$$

Scaling an object with transformation 41 changes the position of the object relative to the coordinate origin. A parameter value greater than 1 moves a point farther from the origin in the corresponding coordinate direction. Similarly, a parameter value less than 1 moves a point closer to the origin in that coordinate direction. Also, if the scaling parameters are not all equal, relative dimensions of a transformed object are changed. We preserve the original shape of an object with a *uniform scaling*: $s_x = s_y = s_z$. The result of scaling an object uniformly, with each scaling parameter set to 2, is illustrated in Figure 17.

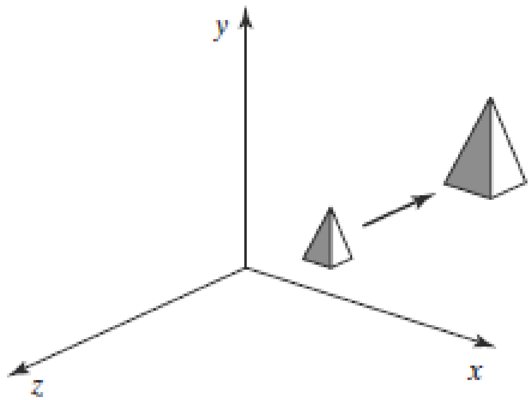


FIGURE 17
Doubling the size of an object with transformation 41 also moves the object farther from the origin.

Because some graphics packages provide only a routine that scales relative to the coordinate origin, we can always construct a scaling transformation with respect to any selected *fixed position* (x_f, y_f, z_f) using the following transformation sequence:

1. Translate the fixed point to the origin.
2. Apply the scaling transformation relative to the coordinate origin using Equation 41.
3. Translate the fixed point back to its original position.

$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1 - s_x)x_f \\ 0 & s_y & 0 & (1 - s_y)y_f \\ 0 & 0 & s_z & (1 - s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{44}$$

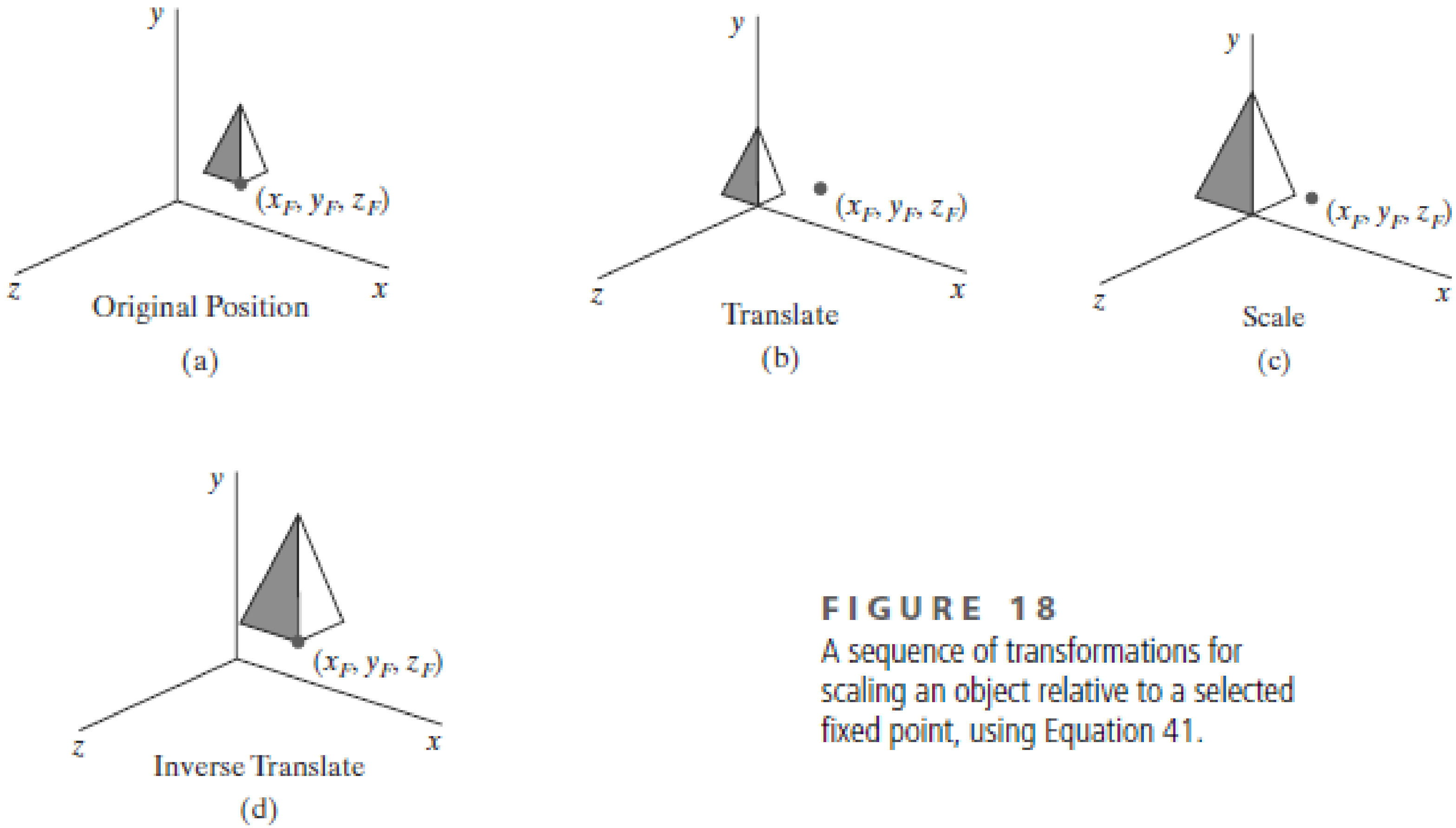
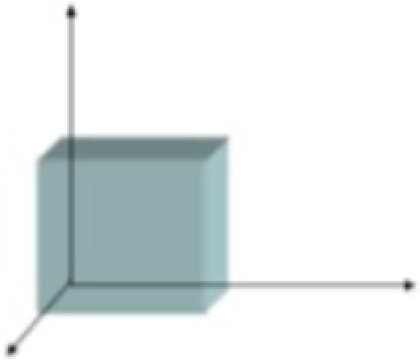


FIGURE 18
A sequence of transformations for scaling an object relative to a selected fixed point, using Equation 41.

3D scaling

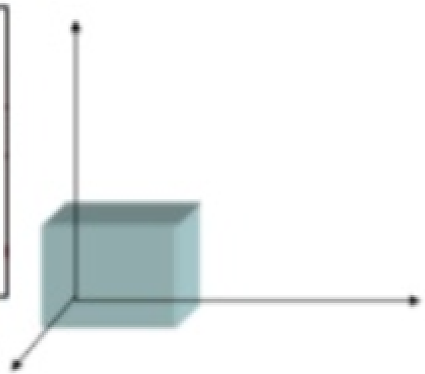
- 3D Scaling

$$[T_s] = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Ex: Required scaling to scale the RPP to a unit cube is ½, 1/3, 1

$$[X'] = [T_s] \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



Routine of 3-D Scaling

```
class wcPt3D
{
   private:
      GLfloat x, y, z;

   public:
   /*  Default Constructor:
    *  Initialize position as (0.0, 0.0, 0.0).
    */
   wcPt3D ( ) {
      x = y = z = 0.0;
   }

   setCoords (GLfloat xCoord, GLfloat yCoord, GLfloat zCoord) {
      x = xCoord;
      y = yCoord;
      z = zCoord;
   }

   GLfloat getx ( ) const {
      return x;
   }

   GLfloat gety ( ) const {
      return y;
   }

   GLfloat getz ( ) const {
      return z;
   }
};
```

```
typedef float Matrix4x4 [4][4];

void scale3D (GLfloat sx, GLfloat sy, GLfloat sz, wcPt3D fixedPt)
{
   Matrix4x4 matScale3D;

   /*  Initialize scaling matrix to identity.  */
   matrix4x4SetIdentity (matScale3D);

   matScale3D [0][0] = sx;
   matScale3D [0][3] = (1 - sx) * fixedPt.getx ( );
   matScale3D [1][1] = sy;
   matScale3D [1][3] = (1 - sy) * fixedPt.gety ( );
   matScale3D [2][2] = sz;
   matScale3D [2][3] = (1 - sz) * fixedPt.getz ( );
}
```

3D Shearing

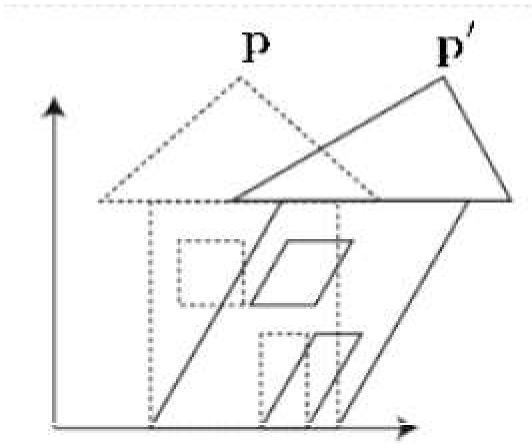
$$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot Sh$$

$$X' = X + Sh_x^y Y + Sh_x^z Z$$

$$Y' = Sh_y^x X + Y + sh_y^z Z$$

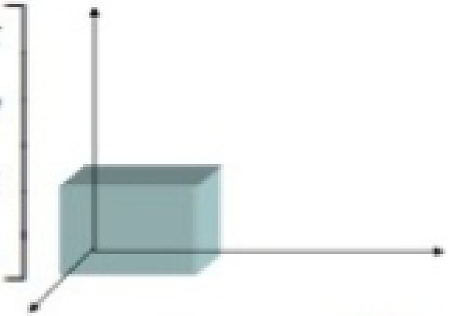
$$Z' = Sh_z^x X + Sh_z^y Y + Z$$



3D shearing

➢ 3D SHEARING

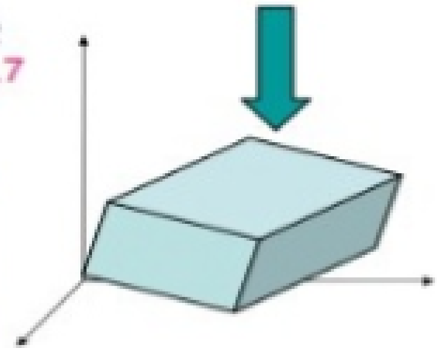
$$[T_{SH}] = \begin{bmatrix} 1 & d & g & 0 \\ b & 1 & i & 0 \\ c & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



$$[T_s][X] = [x + yd + gz \quad bx + y + iz \quad cx + fy + z \quad 1]^T$$

Ex: Uniformly scale the unit cube by a factor of 2 requires d= -0.75, g=0.5,i=1, b=-0.85,c=0.25,f=0.7

$$[X'] = \begin{bmatrix} 0.5 & 1.5 & 0.75 & -0.25 & 0 & 1 & 0.25 & -0.75 \\ 1 & 0.15 & 1.15 & 2 & 0 & -0.85 & 0.15 & 1 \\ 1 & 1.25 & 1.95 & 1.7 & 0 & 0.25 & 0.95 & 0.7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



Combine 3D Transformation

- COMBINATION OF TRANSFORMATIONS – As in 2D, we can perform a sequence of 3D linear transformations.
- This is achieved by concatenation of transformation matrices to obtain a combined transformation matrix

A combined matrix $[T][X] = [X][T_1][T_2][T_3][T_4]\ldots[T_n]$

Where $[T_i]$ are any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

linear trans. but not perspective transformation

(Results in loss of info)

3D Combine Transformations

- Example – Transform the given position vector [3 2 1 1] by the following sequence of operations
(i) Translate by –1, -1, -1 in x, y, and z respectively
(ii) Rotate by +30° about x-axis and +45° about y axis
The concatenated transformation matrix is:

$$[T] = [T_{tr}][T_{rx(30)}][T_{ry(45)}] =$$

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

$$[T][X] = \begin{bmatrix} 0.707 & 0.354 & 0.612 & -1.673 \\ 0 & 0.866 & -0.5 & -0.366 \\ -0.707 & 0.354 & 0.612 & -0.259 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.768 \\ 0.866 \\ -1.061 \\ 1 \end{bmatrix}$$