3D Modelling Transformations

Intended Learning Outcomes

- Understand the use of homogeneous coordinates
- Learn different types of 3D transforms and the concept of composite transform
- Able to use coordinate transform to switch between one coordinate frame to another
- Able to use OpenGL to implement coordinate transform

Homogeneous coordinates

- Represent a n-dimensional entity as a (n+1)-dimensional entity
- Allow all linear transforms to be expressed as matrix multiplications; eliminate matrix addition/subtraction

Linear Transform

- $\mathbf{P}_2 = \mathbf{M}_1\mathbf{P}_1 + \mathbf{M}_2$

  $\mathbf{P}_1$ n-dimensional points (n x 1 column vector)
  $\mathbf{P}_2$ Transformed n-dimensional points
  (n x 1 column vector)
  $\mathbf{M}_1$ n x n square transform matrix
  $\mathbf{M}_2$ n x 1 column transform vector

- Homogeneous coordinates allow us to express the multiplicative term $\mathbf{M}_1$ and the addition term $\mathbf{M}_2$ in a common 4 x 4 matrix. This is achieved by adding one dimension w.

3D Point

A 3D point (n = 3) can be expressed as

- (X, Y, Z) Euclidean coordinates
- (X_W, Y_W, Z_W, W) Homogeneous coordinates

$$X = \frac{X_W}{W} \quad Y = \frac{Y_W}{W} \quad Z = \frac{Z_W}{W}$$

- W can be any non-zero value.

3D Translation

- Euclidean

$$\mathbf{P}_2 = \mathbf{P}_1 + \mathbf{T}(t_X, t_Y, t_Z)$$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} + \begin{pmatrix} t_X \\ t_Y \\ t_Z \end{pmatrix}$$

- Homogeneous

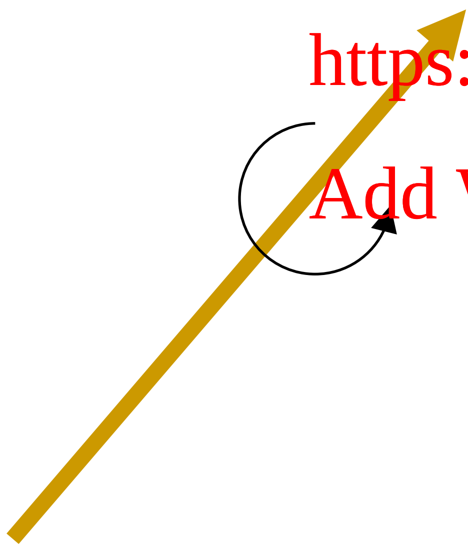
$$\mathbf{P}_2 = \mathbf{T}(t_X, t_Y, t_Z)\mathbf{P}_1$$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_X \\ 0 & 1 & 0 & t_Y \\ 0 & 0 & 1 & t_Z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Note : $W_2 = W_1 = 1$

3D Rotations

- Rotation about an axis

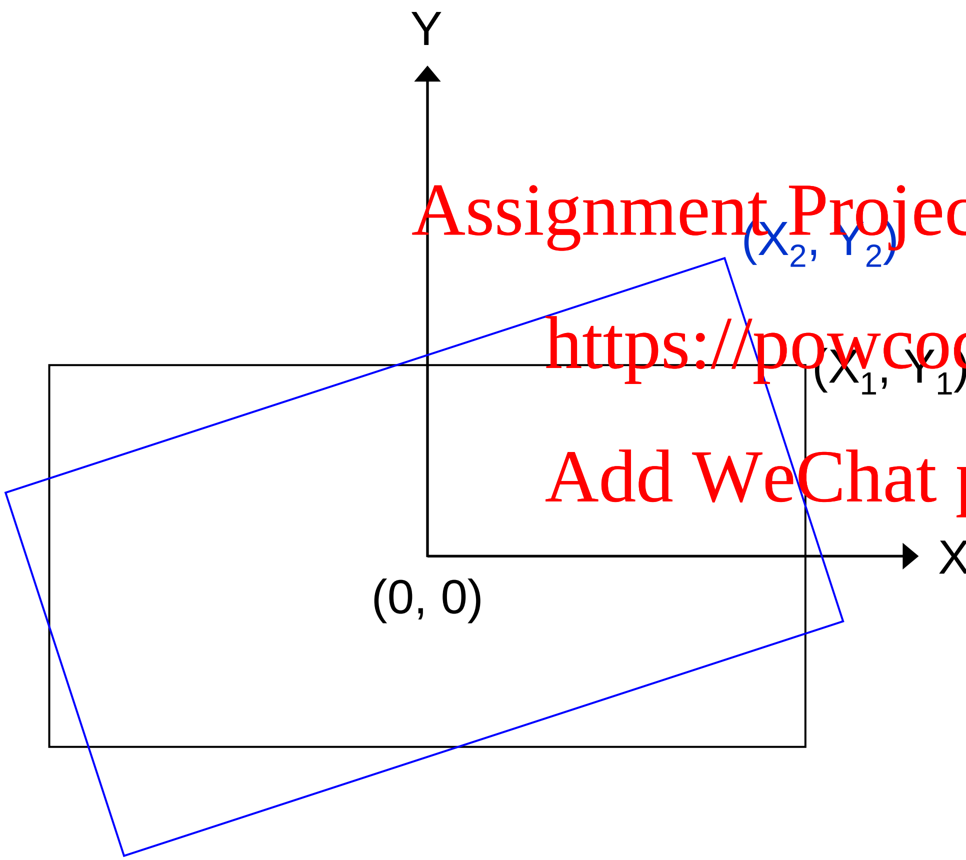


CCW $\Rightarrow$ POSITIVE rotation

Right Hand Rule

2D Rotations about the origin

- About a common coordinate system X-Y



$$\begin{pmatrix} X_2 \\ Y_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \end{pmatrix}$$

Equivalent to rotation about Z axis, which is pointing out of paper

Rotation about Z

- Euclidean

  $\mathbf{P}_2 = \mathbf{R}_Z(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2 = \mathbf{R}_Z(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Rotation about X

- Euclidean

  $\mathbf{P}_2$=$\mathbf{R}_X(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2$=$\mathbf{R}_X(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Rotation about Y

- Euclidean

  $\mathbf{P}_2=\mathbf{R}_Y(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2=\mathbf{R}_Y(\theta)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Scaling about the origin

- Euclidean

  $\mathbf{P}_2 = \mathbf{S}(s_X, s_Y, s_Z)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} s_X & 0 & 0 \\ 0 & s_Y & 0 \\ 0 & 0 & s_Z \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2 = \mathbf{S}(s_X, s_Y, s_Z)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} s_X & 0 & 0 & 0 \\ 0 & s_Y & 0 & 0 \\ 0 & 0 & s_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Reflection about the X-Y plane

- Euclidean

  $\mathbf{P}_2 = \mathbf{RF}_Z\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2 = \mathbf{RF}_Z\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Shearing about the Z axis

- Euclidean

  $\mathbf{P}_2 = \mathbf{Sh}_z(a,b)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix}$$

- Homogeneous

  $\mathbf{P}_2 = \mathbf{Sh}_Z(a,b)\mathbf{P}_1$

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \\ W_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \\ W_1 \end{pmatrix}$$

Affine Transform

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- a_{ij} and b_i are constants.
- a linear transformation
- // lines are transformed to // lines
- Translation, rotation, scaling, reflection, shearing are special cases
- Any affine transform can be expressed as composition of the above 5 transforms

Composite Transformation

- A number of (relative) transformations applied in sequence
- Models the complex movement of an object in the world coordinate system
- The transformation is pre-computed where possible.
- In practice, ONLY the final 4 x 4 composite transformation needs to be stored.

E.g. 1 Rotation about an axis // to X axis.

- Let (X_f, Y_f, Z_f) be a point on the axis. The composite rotation is

$$\mathbf{P}_2 = \mathbf{T}^{-1}\mathbf{R}_x(\theta)\mathbf{T}\,\mathbf{P}_1$$

$$\mathbf{T} = \mathbf{T}(-X_f, -Y_f, -Z_f)$$

- For the composite transformation

$$\begin{pmatrix} 1 & 0 & 0 & X_f \\ 0 & 1 & 0 & Y_f \\ 0 & 0 & 1 & Z_f \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -X_f \\ 0 & 1 & 0 & -Y_f \\ 0 & 0 & 1 & -Z_f \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Only the product

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & -Y_f\cos\theta + Z_f\sin\theta + Y_f \\ 0 & \sin\theta & \cos\theta & -Y_f\sin\theta - Z_f\cos\theta + Z_f \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

is stored

E.g. 2 Scaling about (X_f, Y_f, Z_f)

- $\mathbf{P}_2 = \mathbf{T}^{-1}\mathbf{S}(s_X, s_Y, s_Z)\mathbf{T}\ \mathbf{P}_1$

$\mathbf{T} = \mathbf{T}(-X_f, -Y_f, -Z_f)$

Similarly, only the final 4 x 4 composite transformation is stored

Concept

- A composite transformation may have two physical meaning:

- Either

It represents a physical action

- Or

It represents a change of coordinate system

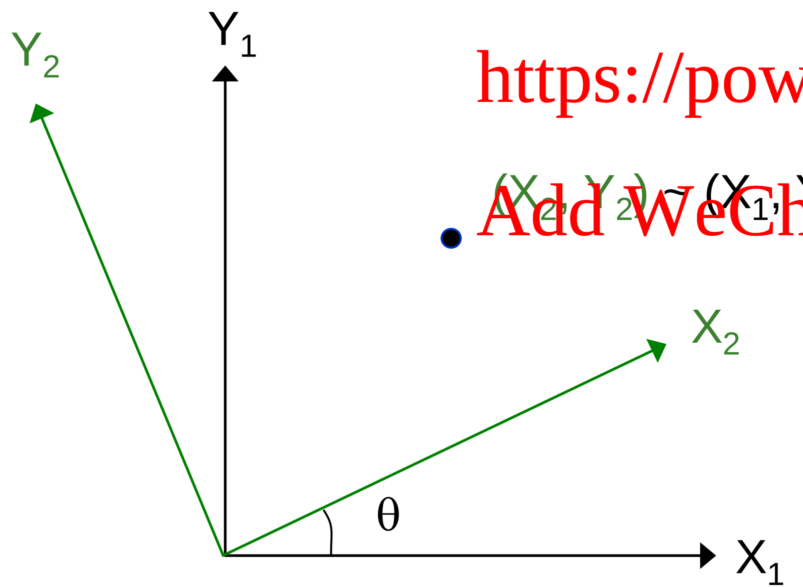
3 Kinds of Coordinate System in CG

- Each object defined in their own natural coordinate system – Modelling coordinate system (MC)
- All objects being placed in a common world coordinate system (WC)
- For correct viewing by a camera, objects need to be expressed in a common viewer or camera coordinate system (VC, CC)

MC → WC → VC/CC

A point in two different coordinate sy.

- The SAME point has DIFFERENT coordinates in DIFFERENT coordinate systems



$$\mathbf{P}^{(2)} = \mathbf{R}(-\theta)\mathbf{P}^{(1)}$$

$$\begin{pmatrix} X_2 \\ Y_2 \end{pmatrix} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \begin{pmatrix} X_1 \\ Y_1 \end{pmatrix}$$

- $\mathbf{P}^{(i)}$ A point in coordinate system i
- $\mathbf{M}_{j \leftarrow i}$ 4 x 4 transformation that transforms a point in coordinate system i to coordinate system j

- $\mathbf{P}^{(j)} = \mathbf{M}_{j \leftarrow i}\, \mathbf{P}^{(i)}$

- Rule 1 for computing $\mathbf{M}_{j\leftarrow i}$:

$\mathbf{M}_{j\leftarrow i}$ is the inverse of the transformation that takes the i^{th} coordinate system frame as if it is an object to the j^{th} coordinate system frame position, all the time using the i^{th} coordinate system as the reference coordinate system

As $\mathbf{M}_{j\leftarrow i} = \mathbf{M}_{i\leftarrow j}^{-1}$, we have the alternative rule:

- Alternative rule (rule 2) for computing $\mathbf{M}_{j \leftarrow i}$:

  $\mathbf{M}_{j \leftarrow i}$ is the transformation that takes the **j**$^{\text{th}}$ coordinate system frame as if it is an object to the **i**$^{\text{th}}$ coordinate system frame position, all the time using the **j**$^{\text{th}}$ coordinate system as the reference coordinate system

- which rule to use depends on which coordinate system is easier to get on hand

$M_{j\leftarrow i}$ is the INVERSE of the transformation that takes the ith coordinate system frame as if it is an object to the jth coordinate system frame

Proof

Suppose we have two coordinate systems x_i-y_i-z_i and x_j-y_j-z_j. Treat x_i-y_i-z_i and x_j-y_j-z_j as two objects that consist of two sets of points, both defined in the x_i-y_i-z_i coordinate system. Let

$$x_i = (1, 0, 0)^T \rightarrow x_j = (a_{11}, a_{21}, a_{31})^T + (t_x, t_y, t_z)^T$$
$$y_i = (0, 1, 0)^T \rightarrow y_j = (a_{12}, a_{22}, a_{32})^T + (t_x, t_y, t_z)^T$$
$$z_i = (0, 0, 1)^T \rightarrow z_j = (a_{13}, a_{23}, a_{33})^T + (t_x, t_y, t_z)^T$$

where all the coordinates are defined in the x_i-y_i-z_i coordinate system. $\rightarrow$ means "corresponds to".

The transformation T that transforms the three points x_i, y_i, z_i to x_j, y_j, z_j in the x_i-y_i-z_i coordinate system is thus

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, it can also be interpreted as changing from coordinate system j to coordinate system i. Thus

$$\mathbf{P}^{(j)} = (1, 0, 0)^T \rightarrow \mathbf{P}^{(i)} = (a_{11}, a_{21}, a_{31})^T + (t_x, t_y, t_z)^T$$
$$\mathbf{P}^{(j)} = (0, 1, 0)^T \rightarrow \mathbf{P}^{(i)} = (a_{12}, a_{22}, a_{32})^T + (t_x, t_y, t_z)^T$$
$$\mathbf{P}^{(j)} = (0, 0, 1)^T \rightarrow \mathbf{P}^{(i)} = (a_{13}, a_{23}, a_{33})^T + (t_x, t_y, t_z)^T$$

Since any arbitrary $\mathbf{P}^{(j)}$ can be written as $\lambda_1(1,0,0)^T + \lambda_2(0,1,0)^T + \lambda_3(0,0,1)^T$, where $\lambda_1, \lambda_2, \lambda_3$ are constants, it follows that

$$\mathbf{M}_{i\leftarrow j} = \mathbf{T}$$

Since $\mathbf{M}_{j\leftarrow i} = \mathbf{M}_{i\leftarrow j}{}^{-1}$,

$$\mathbf{M}_{j\leftarrow i} = \mathbf{T}^{-1}$$

This gives the rule

$\mathbf{M}_{j\leftarrow i}$ is the INVERSE of the transformation that takes the ith coordinate system frame as if it is an object to the jth coordinate system frame

OpenGL Geometric Transformations

- 4 x 4 translation matrix
  glTranslatef (tx, ty, tz);
- 4 x 4 rotation matrix
  glRotatef (theta, vx, vy, vz);

- 4 x 4 scaling matrix

  glScalef (sx, sy, sz);
- 4 x 4 reflection matrix

  glScalef (1, 1, -1); // reflection about Z axis
- 4 x 4 shearing matrix
  glMultMatrixf (matrix); // matrix is a 16 element
  // matrix in column-major order

OpenGL Matrix Operations

- Calls the current matrix, responsible for geometrical transformation

  glMatrixMode (GL_MODELVIEW);

  (do not confuse with *glMatrixMode (GL_PROJECTION)*, which is responsible for projection transformation)

- Assign identity matrix to current matrix

  glLoadIdentity ( );

- Current matrix is modified by (relative) transformations
  - E.g. glTranslatef, glScalef, glRotatef
  - The meaning of the relative transformations may either be physical action or coordinate transformations

- Current matrix are *postmultiplied. Last operation specified is first operation performed, like a LIFO stack*

Let **C** be the composite matrix

- Example 1

glMatrixMode (GL_MODELVIEW)

glLoadIdentity ( ); // **C** = identity matrix

glTranslatef (-25, 50, 25); // **C** = **T**(-25,50,25)

glRotatef (45, 0, 0, 1); // **C** = **T** (-25,50,25)$\mathbf{R}_Z(45^o)$

glScalef (1, 2, 1); // **C** = **T** (-25,50,25)$\mathbf{R}_Z(45^o)$ **S**(1,2,1)

- Example 2

glMatrixmode (GL_MODELVIEW)

glLoadIdentity ( );

glScalef (1, 2, 1);

glRotatef (45, 0, 0, 1);

glTranslatef (-25, 50, 25); // $\mathbf{C} = \mathbf{S}(1,2,1)\mathbf{R}_Z(45^o)\mathbf{T}(-25,50,25)$

Note: the order of the transformation is important

OpenGL Matrix Stacks

- OpenGL has a stack for storing the relative transformations
- Stack is a LIFO data structure
- Stores intermediate results

- Push the current matrix into the stack
  glPushMatrix ( );
- Pop the current matrix from the stack
  glPopMatrix   ( );

Note: Very useful for modelling hierarchical structures

- Example

glMatrixMode (GL_MODELVIEW)
glLoadIdentity ( ); // ***MV*** = identity matrix

glTranslatef (-25, 50, 25); // **MV** = **T**(-25,50,25)
glRotatef (45, 0, 0, 1); // **MV** = **T** (-25,50,25)$\mathbf{R}_Z(45^o)$
glPushMatrix ( ); // **MV** is pushed to the stack
glScalef (1, 2, 1); // **MV** = **T** (-25,50,25)$\mathbf{R}_Z(45^o)$ **S**(1,2,1)
glTranslatef (0, 0, 10); // **MV** = **T** $\mathbf{R}_Z(45^o)$**S**(1,2,1) **T**(0, 0, 10)
glPopMatrix ( );
// **MV** = **T** (-25,50,25)$\mathbf{R}_Z(45^o)$

References

- Text: Sec 7.2 -7.3, 9.1 – 9.7 (except quaternion method), 9.8. The text uses a different exposition of the coordinate transformation method.
- Our discussion of coordinate transformation follows: Foley et. al., Computer Graphics, 2nd Ed., 222-226
- The two methods of coordinate transformation are conceptually the same.