Computer Graphics

Chapter 9

Three-Dimensional Geometric Transformations

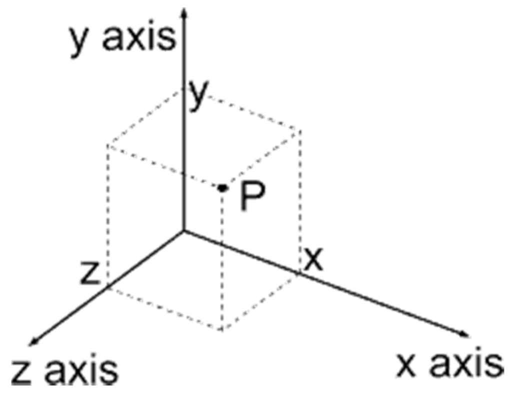
Outline

- 3D Translation
- 3D Rotation
- 3D Scaling
- Transformations between 3D Coordinate Systems
- OpenGL Geometric Transformation Functions
- OpenGL 3D Geometric Transformation Programming Examples

3D Transformation

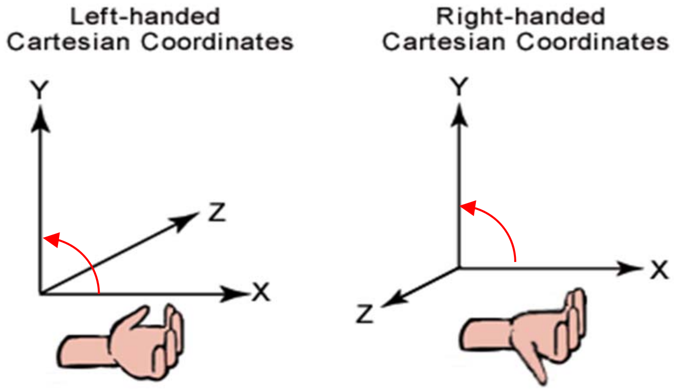
- Same as 2D.
- Add z-axis and z-coordinate.
- Use 4X4 homogenous matrix.

(x, y, z, w)

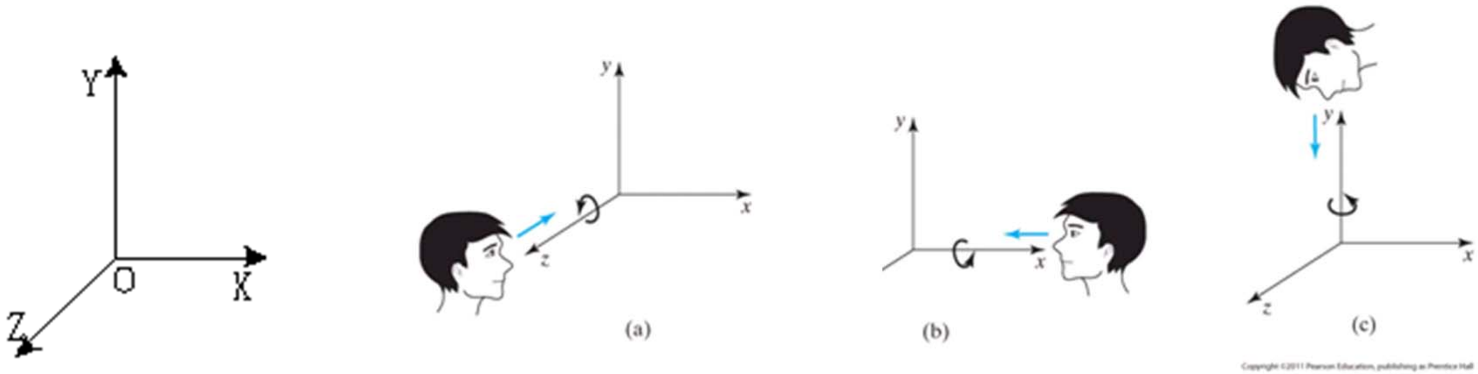


$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Right-hand coordinate system



- OpenGL: **right-hand**
  - The positive x and y axes point right and up, and the z axis points to the viewer.
  - Positive rotation is **counterclockwise** about the axis of rotation when looking along the positive half of the axis toward the origin.



3D Translation

- A position P=(x, y, z) in 3D space is translated to a location P'=(x', y', z') by adding translation distances t_x, t_y, and t_z:

$$x' = x + t_x \qquad y' = y + t_y \qquad z' = z + t_z \tag{9-1}$$

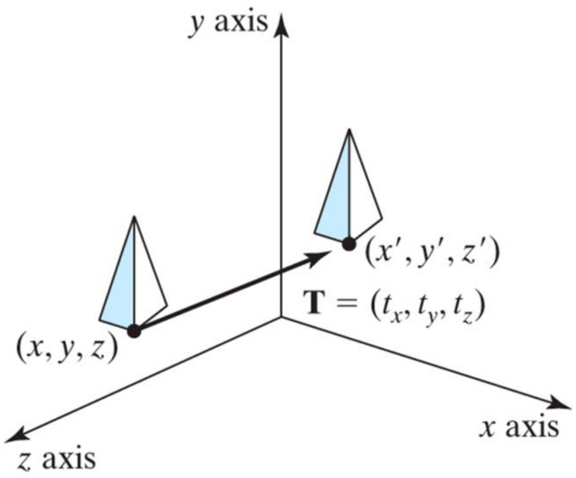


FIGURE 9-2 Shifting the position of a three-dimensional object using translation vector T.

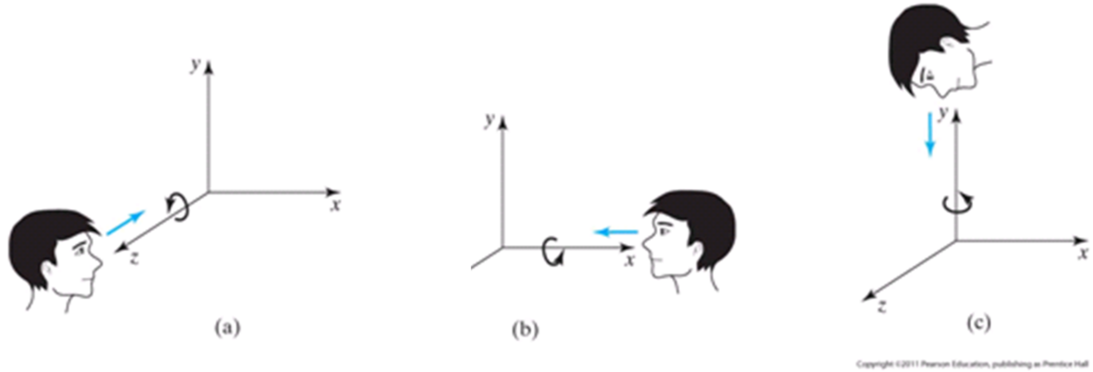
By matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{9-2}$$

or

$$\mathrm{P}' = \mathrm{T} \cdot \mathrm{P} \tag{9-3}$$

3D Rotation

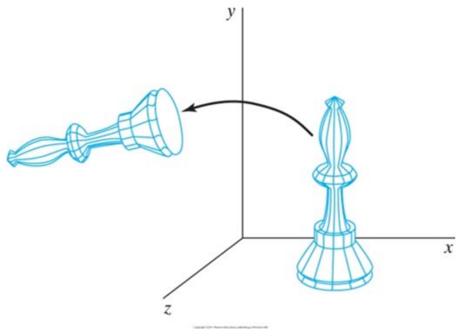


Positive rotations: **counterclockwise** when looking along the positive half of the axis toward the origin

- Coordinate-Axes Rotations
  - X-axis, Y-axis or Z-axis rotation
  - Rotation about an axis that is in parallel to one of the coordinate axes
- General 3D Rotations
  - Rotation about an arbitrary axis

3D Coordinate Axis Rotation

- Rotation of an object about the z axis

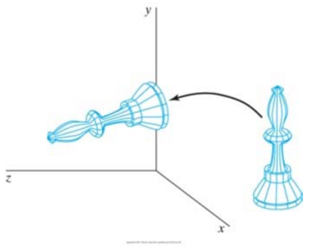


$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

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

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

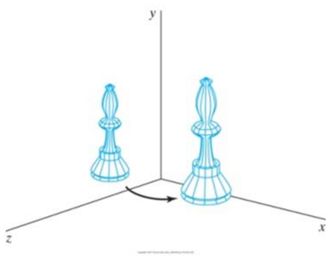
- Rotation of an object about the x axis



z → x; x → y; y → z.

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

- Rotation of an object about the y axis

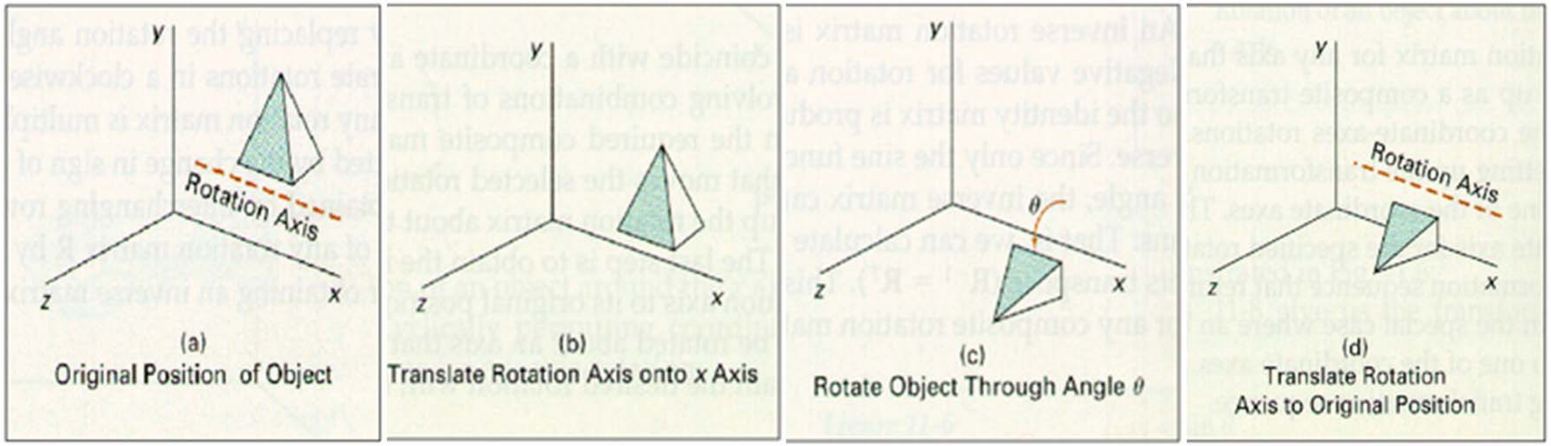


x → y; y → z; z → x.

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

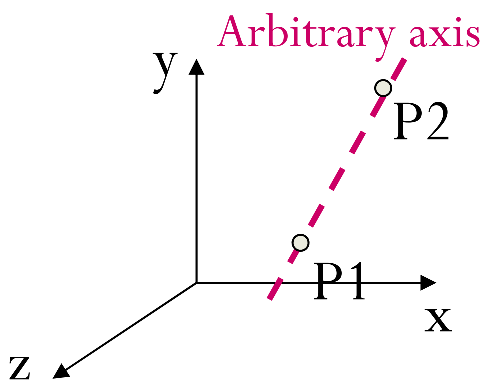
3D Rotations Parallel to Axes

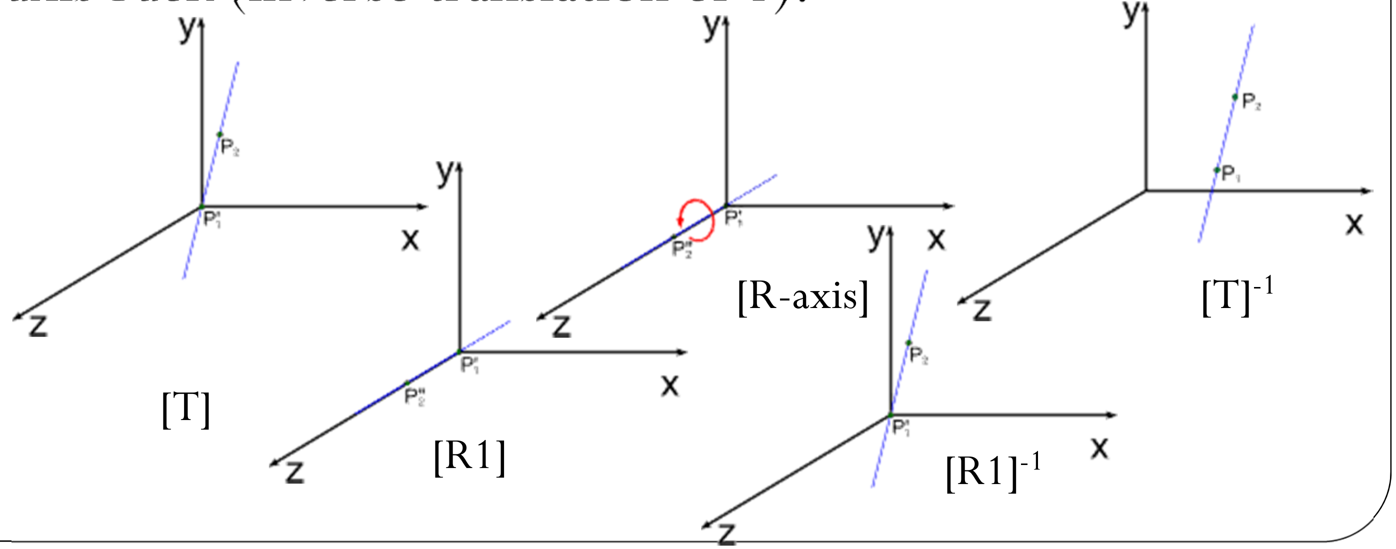
- Rotation an axis that is parallel to one of the coordinate axes
  - Translate the object so that the rotation axis coincides with the parallel coordinate axis
  - Perform the specified rotation about that axis
  - Translate the object so that the rotation axis is moved back to its original position



3D Rotations about Arbitrary Axis

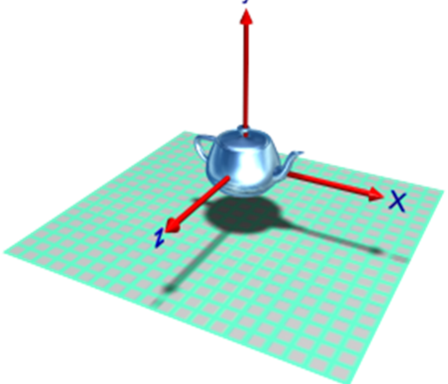
- Rotate about the arbitrary axis through P1 and P2:
  1. Translate P1 to origin.
  2. Rotate so that the rotation axis is aligned with one of the principle coordinate axes.
  3. Perform the desired rotation about coordinate axis.
  4. Rotate axis back (inverse rotation of 2).
  5. Translate axis back (inverse translation of 1).



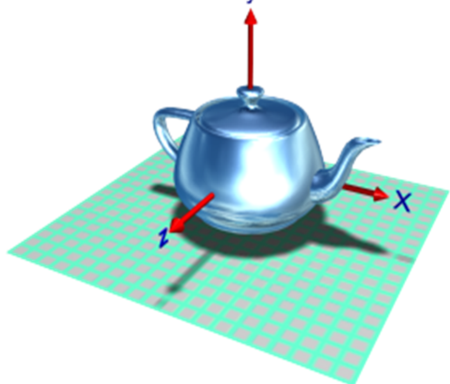


3D Scaling

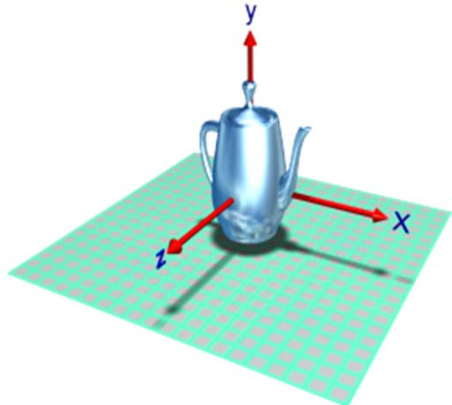
- Scale objects relative to the coordinate origin (0, 0, 0)
  - All vectors are scaled from the origin



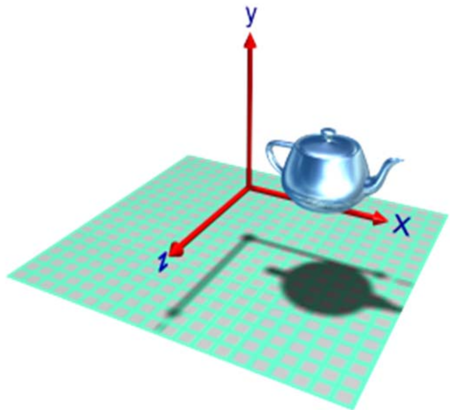
Original



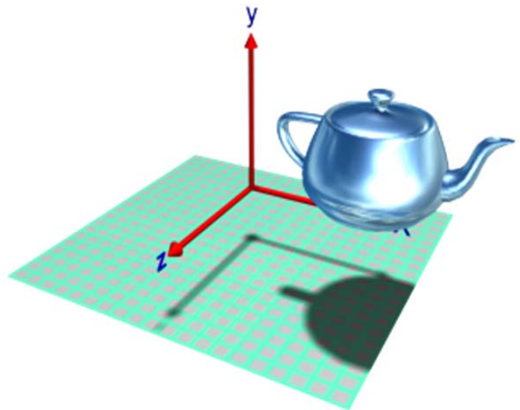
scale all axes



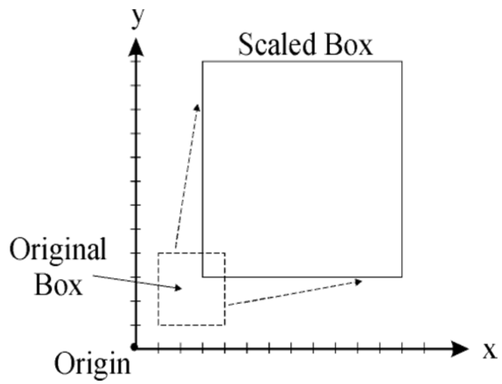
scale Y axis



offset from origin



distance from origin also scales

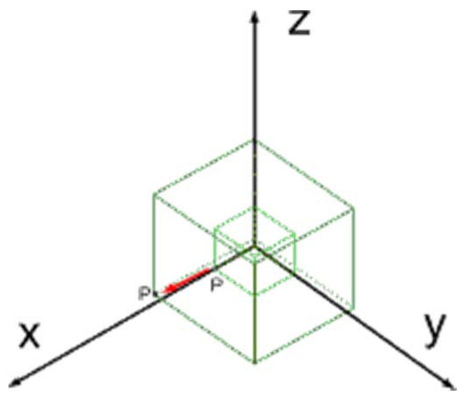


3D Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \Rightarrow \quad \mathrm{P}' = \mathbf{S} \cdot \mathrm{P}$$

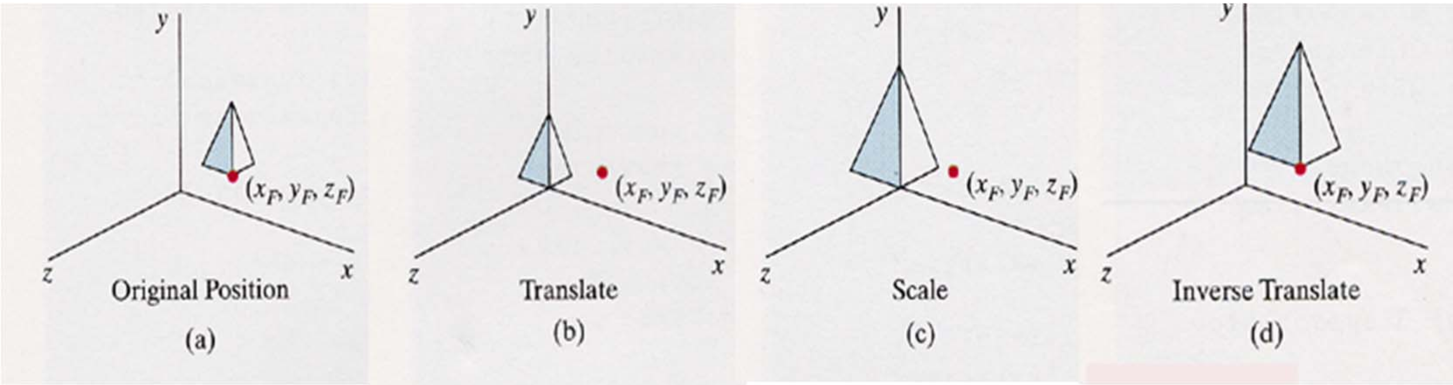
or in 3D homogeneous coordinates

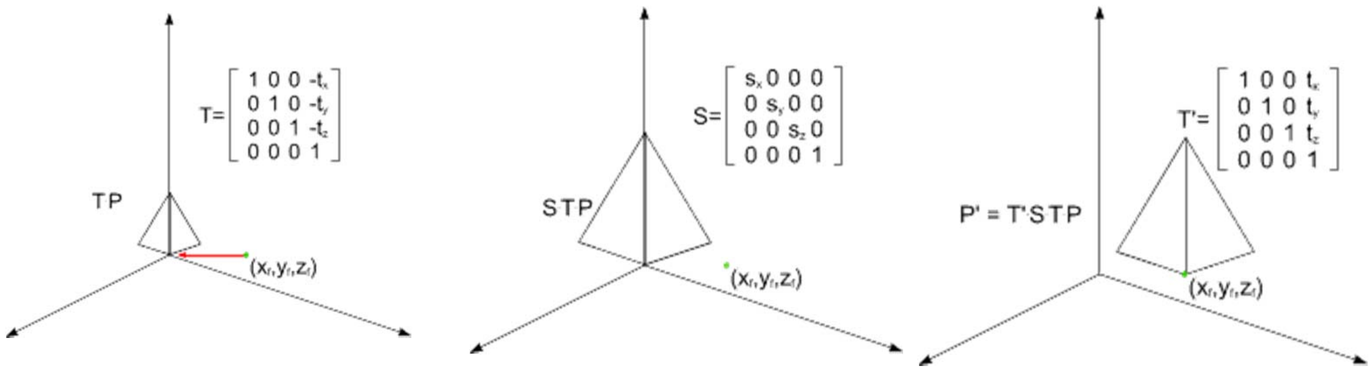
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad (9\text{-}41)$$



3D Scaling

- Scaling objects relative to a selected fixed point (x_f, y_f, z_f)

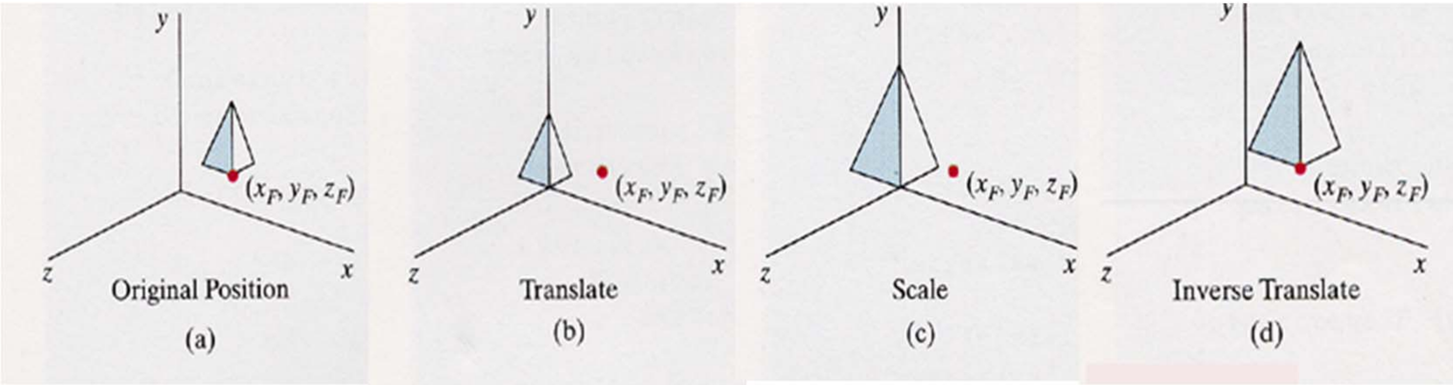




Translate fixed point to origin   Scale   Translate fixed point back

3D Scaling

- Scaling objects relative to a selected fixed point (x_f, y_f, z_f) (cont.)



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

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

Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\mathbf{p}' \quad = \quad T(t_x,t_y) \qquad R(Q) \qquad S(s_x,s_y) \qquad \mathbf{p}$$

$$\boxed{\begin{aligned} \mathbf{p}' &= (T * (R * (S*\mathbf{p}))) \\ \mathbf{p}' &= (T*R*S) * \mathbf{p} \end{aligned}}$$

- **Order of transformations**
  - Matrix multiplication is not commutative

$$\mathbf{p}' = T * R * S * \mathbf{p}$$

"Global" "Local"

Transformations Between 3D Coordinate Systems

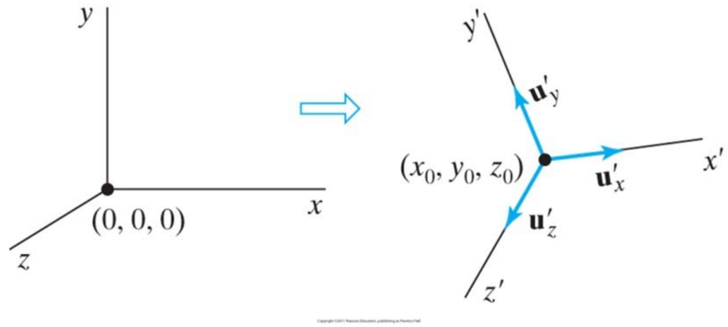


FIGURE 9-21 A new x'y'z' coordinate system defined within an xyz system. A scene description is transferred to the new coordinate reference using a transformation sequence that superimposes the x'y'z' frame on the xyz axes.

- To transfer the ***xyz*** coordinate descriptions -> ***x'y'z'*** coordinate system
  - **Translation**: bring the ***x'y'z'*** coordinate origin to the position of the ***xyz*** origin.
  - **Transform *x'y'z'*** onto the corresponding axes ***xyz***: the coordinate-axis rotation matrix formed by the unit axis vectors.

$$R = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(P313)

which transforms unit vector $\boldsymbol{u'}_x$, $\boldsymbol{u'}_y$, $\boldsymbol{u'}_z$ onto the x, y and z axes.

$$\mathbf{M}_{xyz,\, x'y'z'} = \mathbf{R} \bullet \mathbf{T}(-x_0, -y_0, -z_0)$$

  - If different scales are used in the two coordinate systems, the scaling transformation may also be needed.

OpenGL Geometric Transformation Functions

- Be careful of manipulating the matrix in OpenGL
  - OpenGL uses 4X4 matrix for transformation.
  - The 16 elements are stored as 1D in *column-major order*

$$\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

OpenGL transform matrix

  - C and C++ store matrices in *row-major order*
  - If you declare a matrix to be used in OpenGL as

    GLfloat M[4][4]; to access the element in row i and column j, you need to refer to it by M[j][i]; or, as

    GLfloat M[16]; and then you need to convert it to conventional *row-major order*.

OpenGL Transformations

- All the transformations done by OpenGL can be described as a **multiplication of two or more matrices**.
  - The mathematics behind these transformations are greatly simplified by the mathematical notation of the matrix.
  - Each of the transformations can be achieved by multiplying a **matrix** that contains the vertices, by a **matrix** that describes the transformation.

OpenGL Geometric Transformation Functions

- Basic OpenGL geometric transformations on the matrix:

glTranslate* (tx, ty, tz);

[glTranslatef (25.0, -10.0, 10.0);

- Post-multiplies the current matrix by a matrix that moves the object by the given x-, y-, and z-values

glScale* (sx, sy, sz);

[glScalef (2.0, -3.0, 1.0);]

- Post-multiplies the current matrix by a matrix that scales an object about the origin. None of sx, sy or sz is zero.

glRotate* (theta, vx, vy, vz);

[glRotatef (90.0, 0.0, 0.0, 1.0);]

- Post-multiplies the current matrix by a matrix that rotates the object in a counterclockwise direction. vector v=(vx, vy, vz) defines the orientation for the rotation axis that passes through the coordinate origin. (the rotation center is (0, 0, 0))

OpenGL: Order in Matrix Multiplication

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );      //Set current matrix to the identity.
glMultMatrixf (elemsM2); //Post-multiply identity by matrix M2.
glMultMatrixf (elemsM1); //Post-multiply M2 by matrix M1.
glBegin (GL_POINTS)
    glVertex3f (vertex);
glEnd( );
```

Modelview matrix successively contains:

I(identity), $M2$, $M2 \cdot M1$

The concatenated matrix is:

$M = M2 \cdot M1$

The transformed vertex is:

$M2 \cdot (M1 \cdot \text{vertex})$

In OpenGL, a transformation sequence is applied in reverse order of which it is specified.

OpenGL: Order in Matrix Multiplication

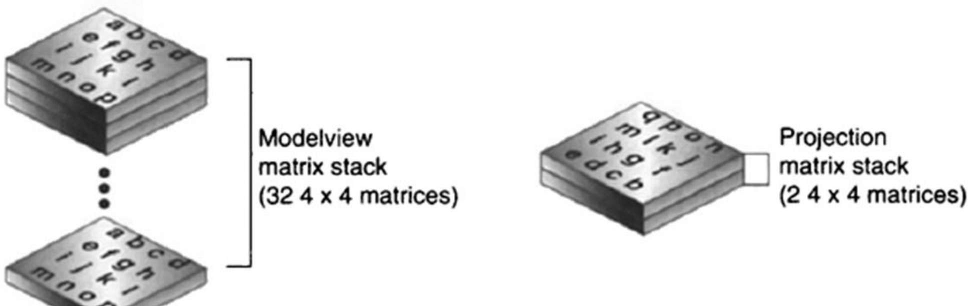
- Example

```
// rotate object 30 degrees around X-axis
glRotatef(30.0, 1.0, 0.0, 0.0);
// move object to (x, y, z)
glTranslatef(x, y, z);
drawObject();
```

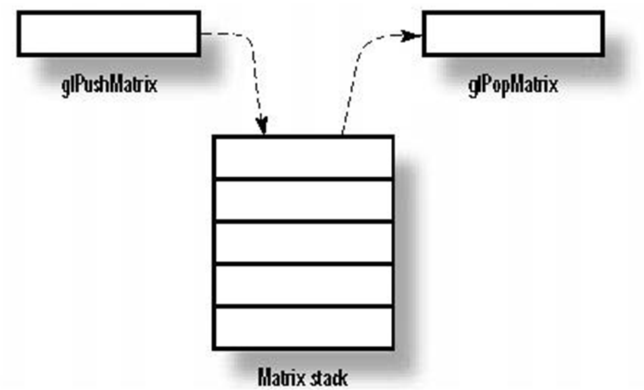
The object will be **translated** first then **rotated**.

Independent Models: Matrix Stacks

- How OpenGL implement the independent models?
- OpenGL maintains a **stack** of matrices.
  - Each type of the matrix modes has a matrix stack (modelview, projection, texture, and color)
  - Initial value is identity matrix
  - The top matrix on the stack at any time: the current matrix
  - New matrix transformation function is applied to the current matrix
  - To use push or pop functions to modify it.



Modelview and Projection Matrix Stacks



Functions About Matrix Stack Operations

- Find the maximum allowable number of matrices in stack

  glGetIntegerv (GL_MAX_MODELVIEW_STACK_DEPTH, stackSize**);**

  glGetIntegerv (GL_MAX_PROJECTION_STACK_DEPTH, stackSize**);**

- Find out how many matrices are currently in the stack

  glGetIntegerv (GL_MODELVIEW_STACK_DEPTH, numMats**);**

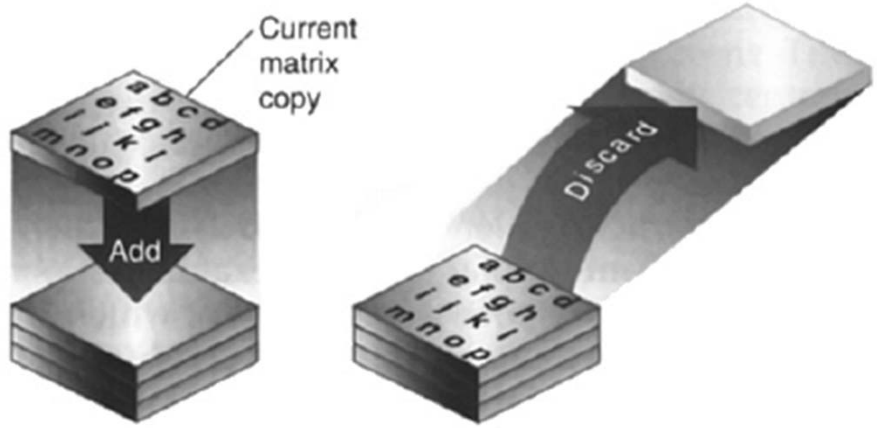
Functions About Matrix Stack Operations

glPushMatrix ( )

- Push the current matrix down one level and copy the current matrix

glPopMatrix ( )

- Pop the top matrix off the stack



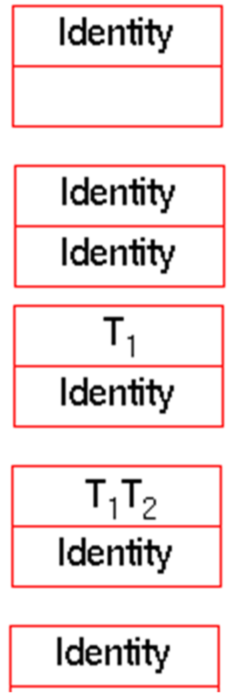
Pushing and Popping the Matrix Stack

- Matrix stack is very useful for creating hierarchical model (body, car …).
  - Save the current position (modelview)
  - Load a previous position or new ones

Matrix Stack Operations

- Example

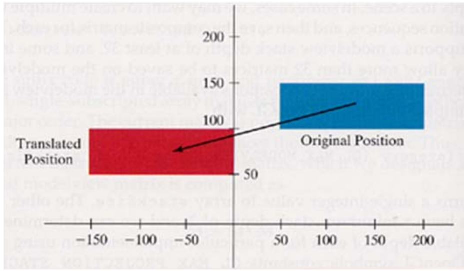
```
glMatrixMode (GL_MODELVIEW);
glPushMatrix ();
    glTranslatef ( 0.0, 0.0, -8.0 );
    glTranslatef ( 1.0, 0.0, 0.0 );
    DrawObj ();
glPopMatrix ();
```

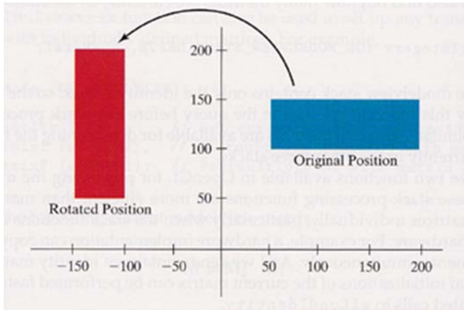


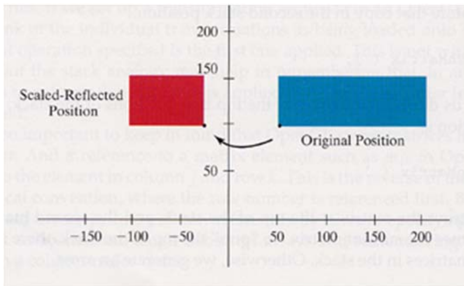
modelview matrix stack

OpenGL Geometric Trans. Programming Examples

```
glMatrixMode (GL_MODELVIEW); //Identity matrix
glColor3f (0.0, 0.0, 1.0);          // Set current color to blue
glRecti (50, 100, 200, 150);      // Display blue rectangle.

glColor3f (1.0, 0.0, 0.0);          // Red
glTranslatef (-200.0, -50.0, 0.0); // Set translation parameters.
glRecti (50, 100, 200, 150);      // Display red, translated rectangle.

glLoadIdentity ();             // Reset current matrix to identity.
glRotatef (90.0, 0.0, 0.0, 1.0);    // Set 90-deg, rotation about z axis.
glRecti (50, 100, 200, 150);      // Display red, rotated rectangle.

glLoadIdentity ();              // Reset current matrix to identity.
glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);      // Display red, transformed rectangle.
```



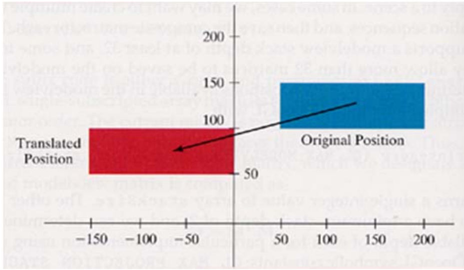


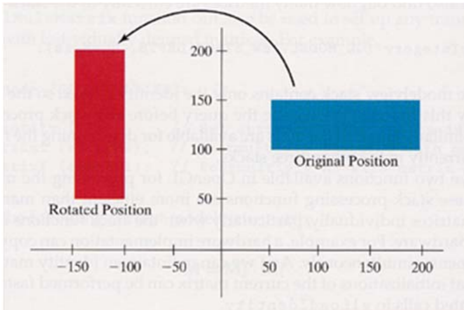


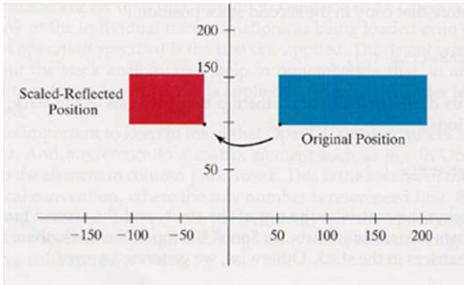
OpenGL Geometric Trans. Programming Examples

- More efficient way: glPushMatrix/glPopMatrix

```
glMatrixMode (GL_MODELVIEW);
glColor3f (0.0, 0.0, 1.0);          // Set current color to blue.
glRecti (50, 100, 200, 150);        // Display blue rectangle.

glPushMatrix ();                    // Make copy of identity (top) matrix.
  glColor3f (1.0, 0.0, 0.0);        // Set current color to red.
  glTranslatef (-200.0, -50.0, 0.0); // Set translation parameters.
  glRecti (50, 100, 200, 150);      // Display red, translated rectangle.
glPopMatrix ();                     // Throw away the translation matrix.

glPushMatrix ();                    // Make copy of identity (top) matrix.
  glRotatef (90.0, 0.0, 0.0, 1.0);  // Set 90-deg, rotation about z axis.
  glRecti (50, 100, 200, 150);      // Display red, rotated rectangle.
glPopMatrix ();                     // Throw away the rotation matrix.

glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);        // Display red, transformed rectangle.
```





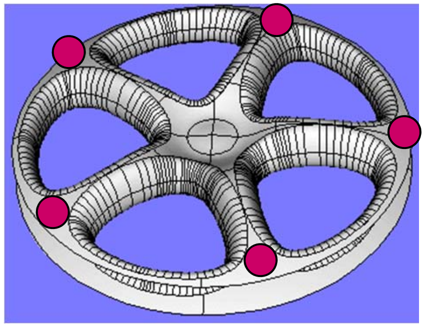


Example

The wheels and bolt axes are coincident with z-axis; the bolts are evenly spaced every 72 degrees, 3 units from the center of the wheel.

- Drawing a car'wheels with bolts

```
draw_wheel( );
for (j=0; j<5; j++) {
        glPushMatrix ();
                glRotatef(72.0*j, 0.0, 0.0, 1.0);
                glTranslatef (3.0, 0.0, 0.0);
                draw_bolt ( );
        glPopMatrix ( );
}
```

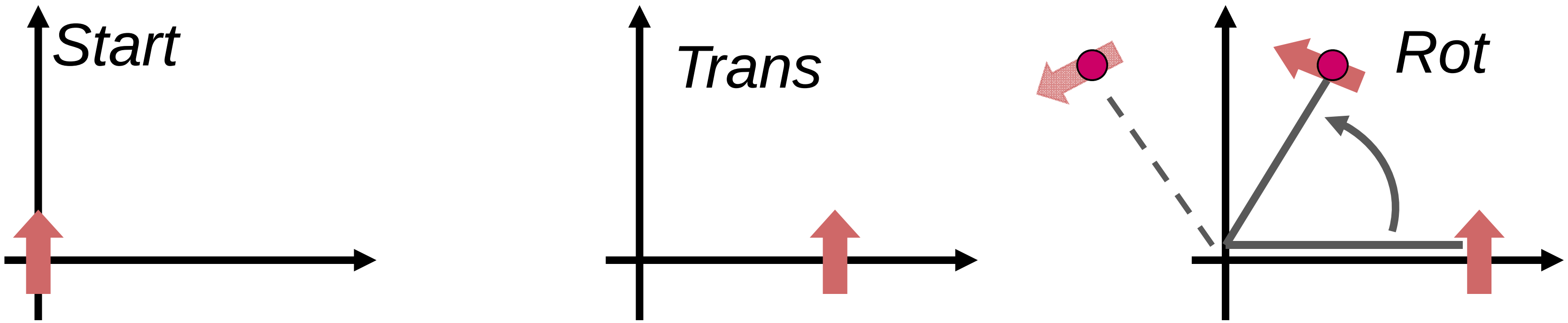


R

RT

RTv

Global – Bottom Up



Summary

- Basic 3D geometric transformations
  - Translation
  - Rotation
  - Scaling
  - Combination of these transformations
- OpenGL 3D geometric transformation functions
  - GL_MODELVIEW matrix
  - Order in multiple matrix multiplication
- Matrix stack
  - glPushMatrix ()
  - glPopMatrix ()