view transformations:
How do we map from world coordinates to camera/view/eye
coordinates?

model transformations:
How do we map from object coordinates to world coordinates?

GL_MODELVIEW transformation
How do we map from object (to world) to view coordinates?

How can we specify the viewer’s coordinate system?

Define the z axis of the viewer by a vector from the 'look at' point to the viewer.

The z coordinate axis of the viewer is a unit vector in the direction is from the 'look at' point to the viewer.

Define any 3D vector \( V_{up} \) such that

\[
V_{up} \cdot \hat{z} \neq 0
\]

This defines a plane, containing \( V_{up} \) and \( \hat{z}_c \).

will be defined to lie in this plane.
What is the relationship between the world coordinate system and the viewer's coordinate system?

To re-map a general scene point \((x, y, z)\) from world coordinates to viewer coordinates, we translate and rotate.

Let the viewer's position be expressed in world coordinates. The matrix \(T\) translates the viewer's position to the origin.

\[
T = \begin{bmatrix}
1 & 0 & 0 & -P_c x \\
0 & 1 & 0 & -P_c y \\
0 & 0 & 1 & -P_c z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\(R\) rotates into the viewer's orientation.

\[
R = \begin{bmatrix}
-x_c & -y_c & z_c & 0 \\
y_c & -x_c & 0 & 0 \\
z_c & 0 & -y_c & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Recall slide 7 from lecture 2.

\(R\) maps to a new coordinate system by projecting onto new axes.

As a programmer using OpenGL, you don't have to compute these vectors. Instead you just define:

\[
\begin{align*}
\text{eye} &= ... \quad // \text{3D points} \\
\text{lookat} &= ... \\
\text{up} &= ...
\end{align*}
\]

\[
gluLookAt( \text{eye}[0], \text{eye}[1], \text{eye}[2], \\
\text{lookat}[0], \text{lookat}[1], \text{lookat}[2], \\
\text{up}[0], \text{up}[1], \text{up}[2] )
\]

What does this definition do ("under the hood")?

Coming soon...
view transformations:
How do we map from world coordinates to camera/view/eye coordinates?

model transformations:
How do we map from object coordinates to world coordinates?

GL_MODELVIEW transformation
How do we map from object (to world) to view coordinates?

```
glBegin(GL_LINES)
glVertex3f(x1, y1, z1)
glVertex3f(x2, y2, z2)
glVertex3f(x3, y3, z3)
glVertex3f(x4, y4, z4)
// more vertex pairs gives more lines
gLend()
```

```
// Quadric Surfaces: General

\[ ax^2 + by^2 + cz^2 + dxz + eyz + fyz + gzx + hxy + ixy + j = 0 \]

Recall homogeneous coordinates. Same quadric surface is represented if we scale 4D vector by a constant.

\[
\begin{bmatrix}
  \omega x \\
  \omega y \\
  \omega z \\
  \omega
\end{bmatrix}
Q
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
= 0
\]
```
Q: What is this surface? (if \(a, b, c > 0\))
\[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix}
\begin{bmatrix}
    a & 0 & 0 & 0 \\
    0 & b & 0 & 0 \\
    0 & 0 & c & 0 \\
    0 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
    x' \\
    y' \\
    z
\end{bmatrix} = 0
\]
A: rotated and translated ellipsoid.

How to define quadric surfaces in OpenGL?

GLUquadricObj myQuadric = gluNewQuadric()

gluSphere(myQuadric, ...)       // need to supply parameters

gluCylinder(myQuadric, ...)

Non-quadric surfaces from OpenGL Utility Toolkit (GLUT)

- glutSolidCube()
- glutWireCube()
- glutSolidTorus()
- glutWireTorus()
- glutSolidTeapot()
- glutWireTeapot()

How to transform objects in OpenGL?

- glRotatef(vx, vy, vz, angle)
- glTranslatef(x, y, z)
- glScalef(sx, sy, sz)

The parameters of each of these calls specify a 4x4 matrix. These transformations are not associated with (bound to) any particular object, however.

We'll see how this works next.

Recall how to transform from world coordinates to viewer coordinates:

\[
\tilde{x}_{\text{viewer}} = R\ T\ \tilde{x}_{\text{world}}
\]

\[
M_{\text{viewer}} = \text{world} \quad M_{\text{world}} = \text{dog}
\]

- gluLookAt( ... ) // transform from world coordinates to viewer/eye coordinates
- glTranslate( ... ) // transform position and orientation
- glRotate( ... ) // of dog to world coordinates
- glVertex( ) // etc. all the triangles of the dog object

...... // defined in dog coordinate system
OpenGL is a "state machine". One of its states is the GL_MODELVIEW matrix. This is a 4x4 matrix that transforms a vertex into eye coordinates.

We would like:

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\end{bmatrix}
\]

ASIDE: How to examine the GL_MODELVIEW matrix?

```python
m = (GLfloat * 16)()
glGetFloatv(GL_MODELVIEW_MATRIX, m)
gModelViewMatrix = [[], [], [], []]  # OpenGL stores in column major order
for i in range(16):
gModelViewMatrix[i % 4].append(m[i])  # OpenGL stores in column major order
print 'GL_MODELVIEW ', gModelViewMatrix
```

Q: What happens when you make these calls?

Answer:

```python
glMatrixMode(GL_MODELVIEW)
gLoadIdentity()
gluLookAt( ... )
M ← M_view \cdot M_world
M ← M_world \cdot M_model
M ← M_model \cdot M_view
```

Solution: use a stack of GL_MODELVIEW transformations.

```python
glMatrixMode(GL_MODELVIEW)
gLoadIdentity()
glGetMatrixf(GL_MODELVIEW_MATRIX, m)
glPushMatrix()  # push the current matrix to the stack
    gluLookAt( ... )
    M ← M_world \cdot M_view
    glRotatef( ... )
    M ← M_world \cdot M_view \cdot M_R
    glTranslatef( ... )
    M ← M_world \cdot M_view \cdot M_T
    glScalef( ... )
    M ← M_world \cdot M_view \cdot M_T \cdot M_S
    drawHouse()             // glVertex() etc...
    glPopMatrix()  # remove the matrix from the stack
```

Problem: the GL_MODELVIEW matrix only keeps track of one (model to view) transformation. But we may have hundreds of object models.

How do we keep track of all these transformations?

```python
glMatrixMode(GL_MODELVIEW)
gLoadIdentity()
glGetMatrixf(GL_MODELVIEW_MATRIX, m)
glPushMatrix()  # push the current matrix to the stack
    gluLookAt( ... )
    glRotatef( ... )
    glTranslatef( ... )
    drawDog()               // glVertex() etc...
    glPushMatrix()  # push the current matrix to the stack
    gluLookAt( ... )
    glRotatef( ... )
    glTranslatef( ... )
    drawHouse()             // glVertex() etc...
    glPopMatrix()  # remove the matrix from the stack
```