lecture 3

view transformations

model transformations

GL_MODELVIEW transformation

view transformations:

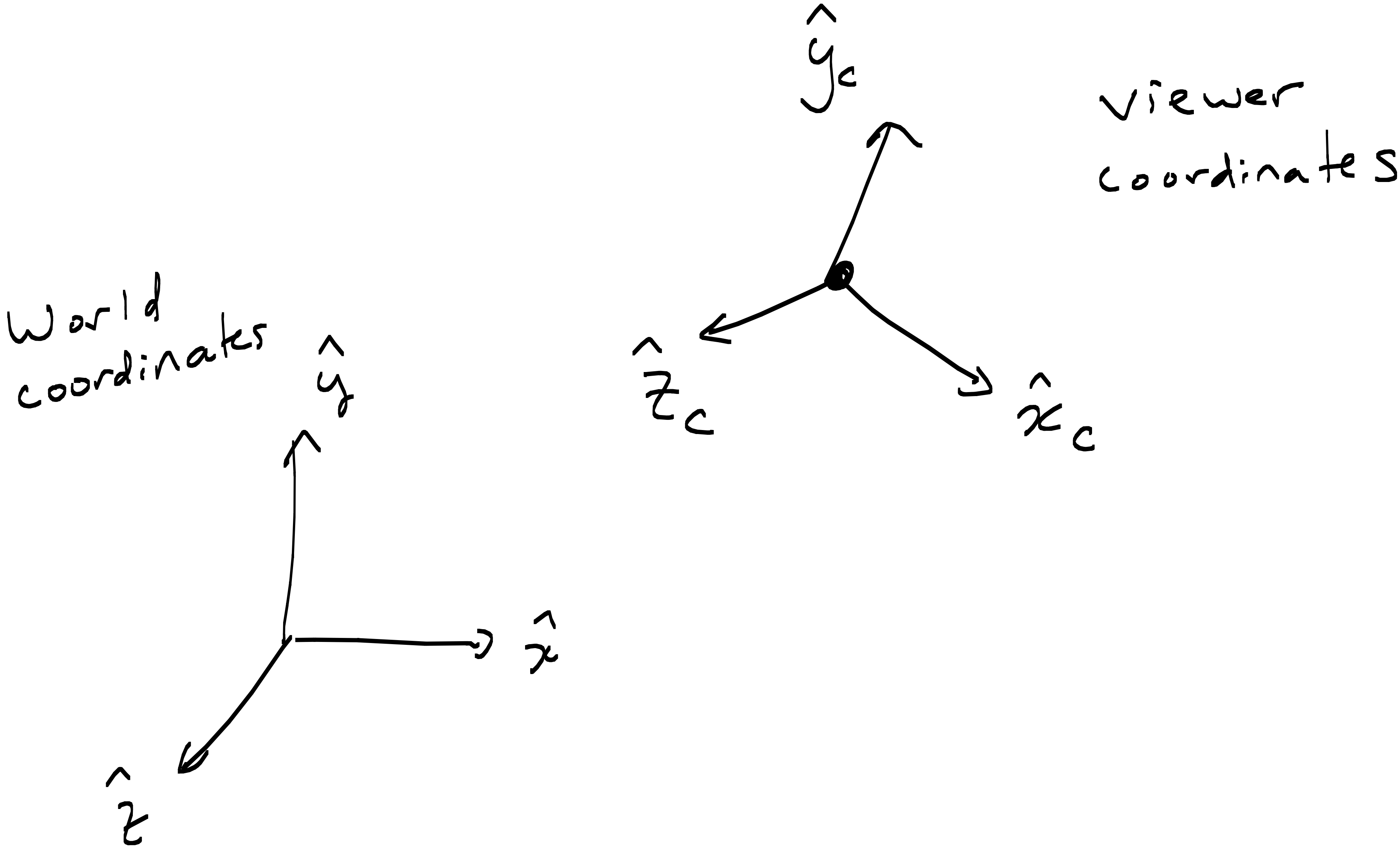
How do we map from world coordinates to camera/view/eye coordinates ?

model transformations:

How do we map from object coordinates to world coordinates ?

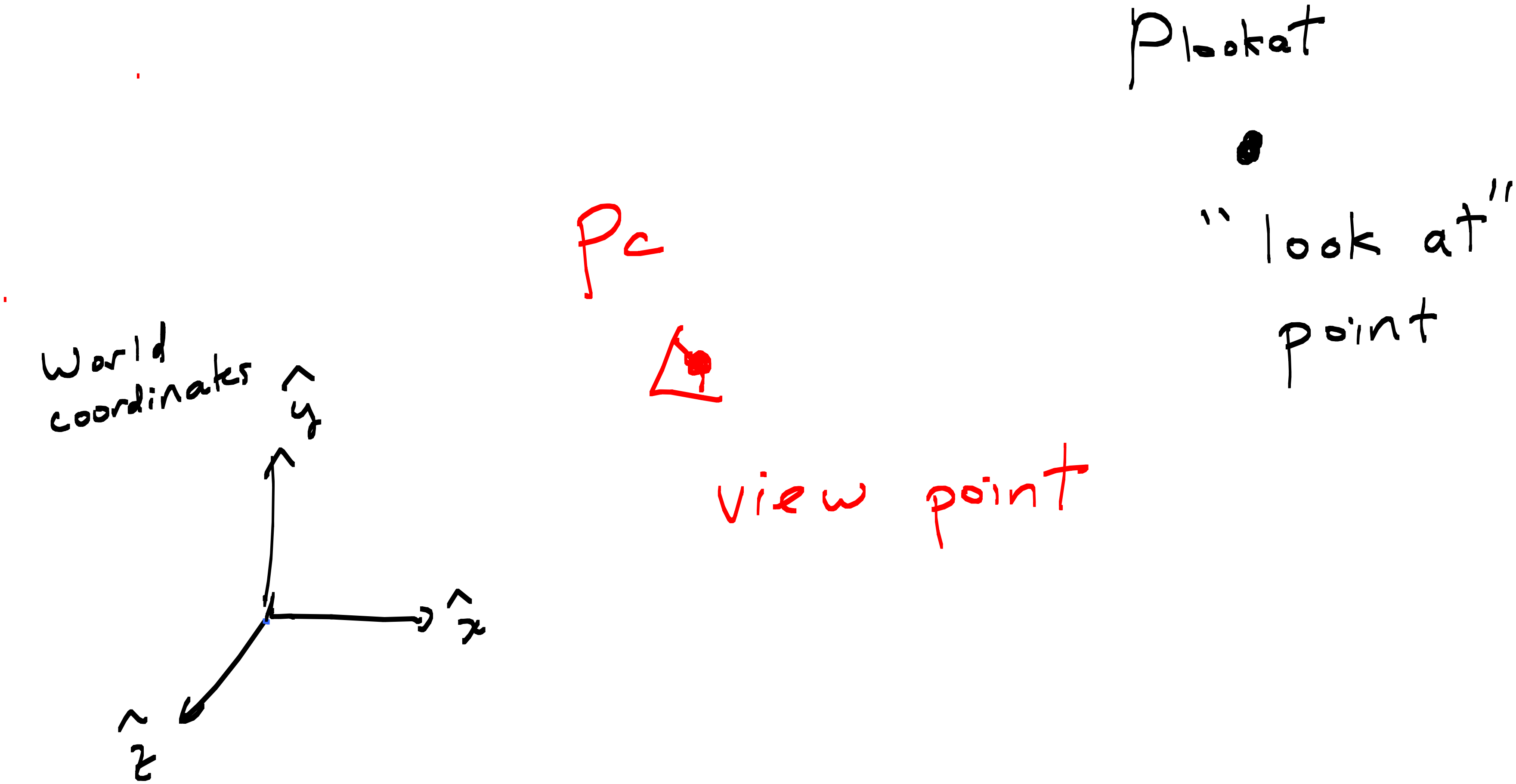
GL_MODELVIEW transformation

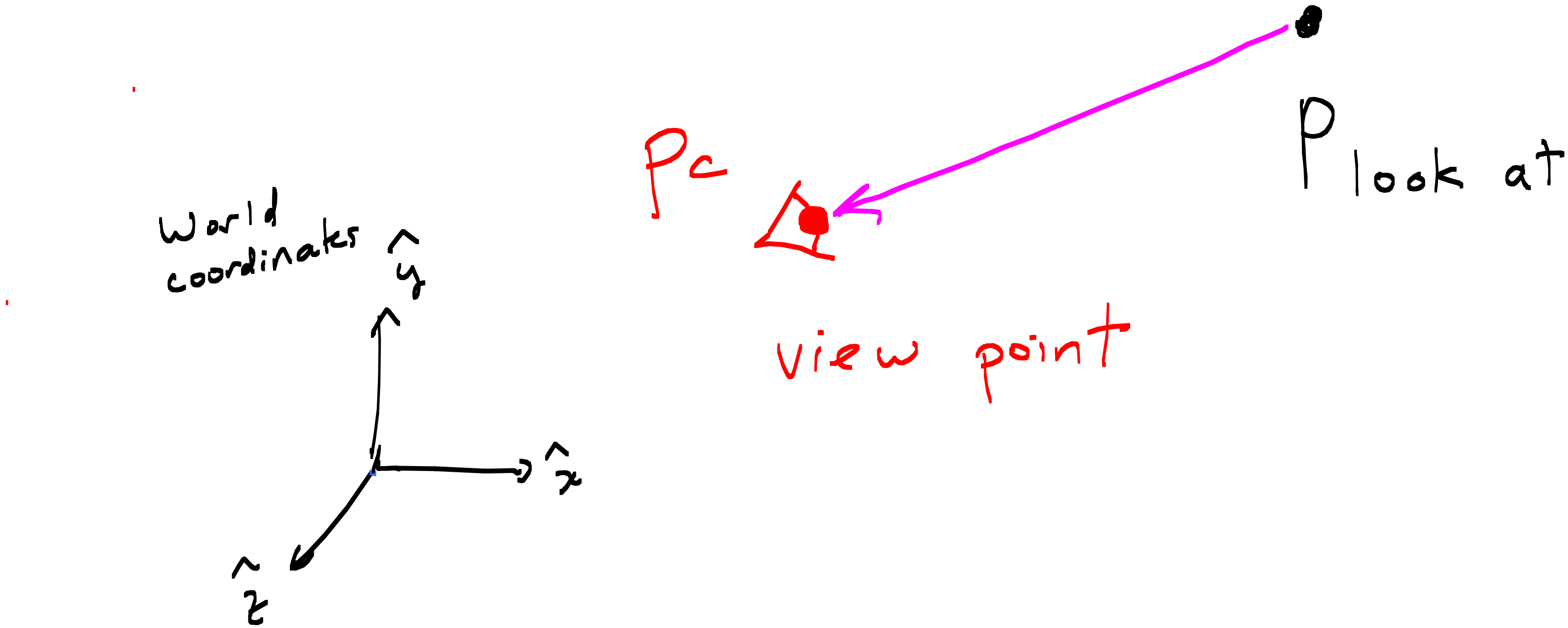
How do we map from object (to world) to view coordinates?



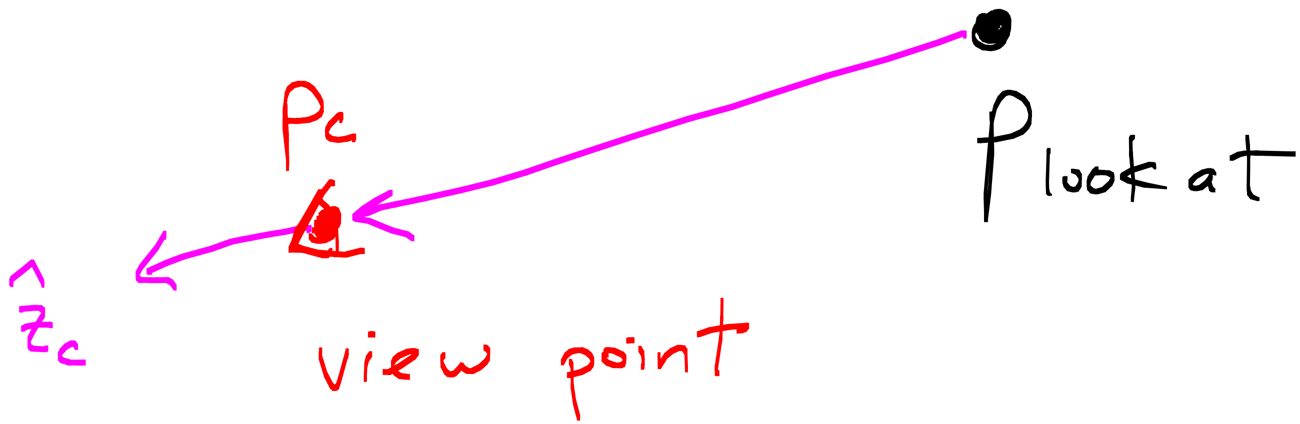
Viewer = camera = eye

How can we specify the viewer's coordinate system ?



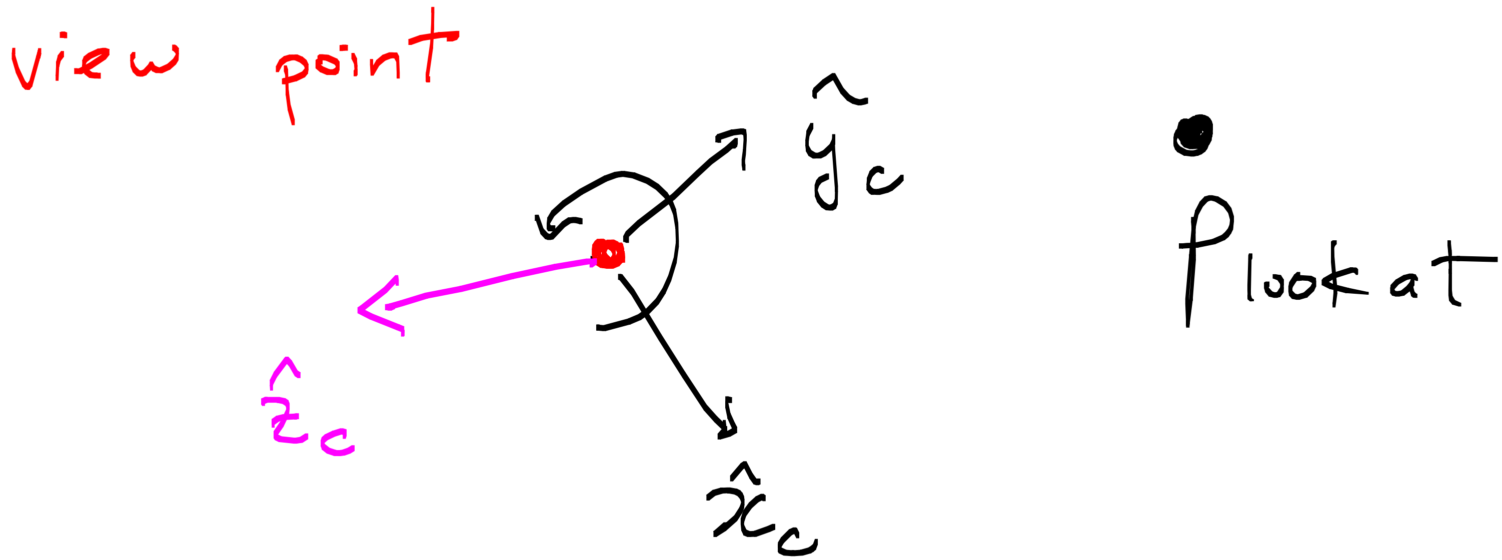


Define the z axis of the viewer by a vector from the 'look at' point to the viewer.



$$\hat{z}_c = \frac{p_c - p_{lookat}}{|p_c - p_{lookat}|}$$

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

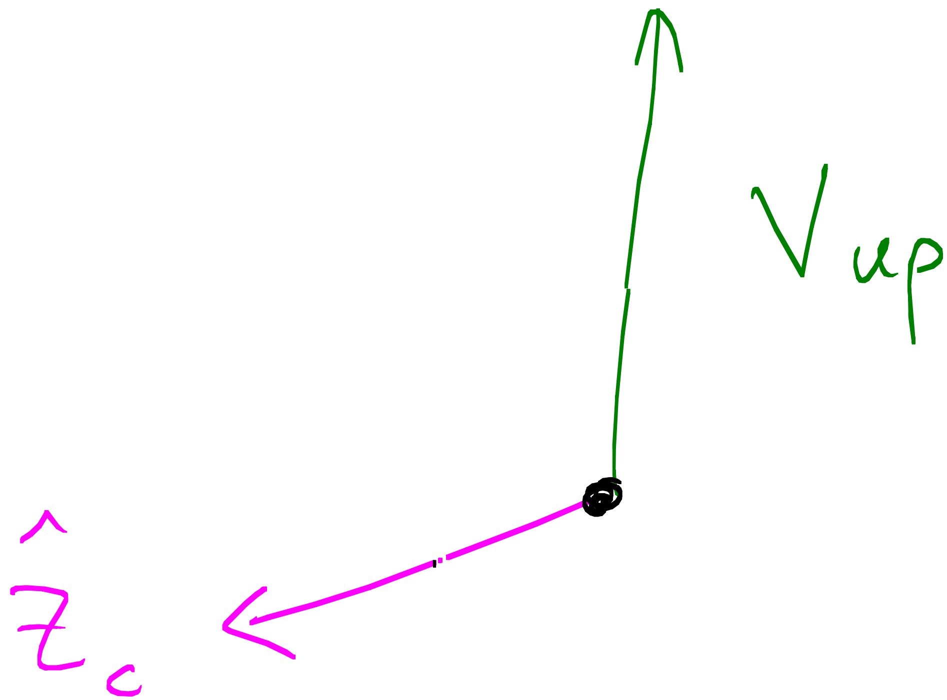


To specify the viewer's x and y coordinate axes, we need to choose from 360 degrees of possibilities.

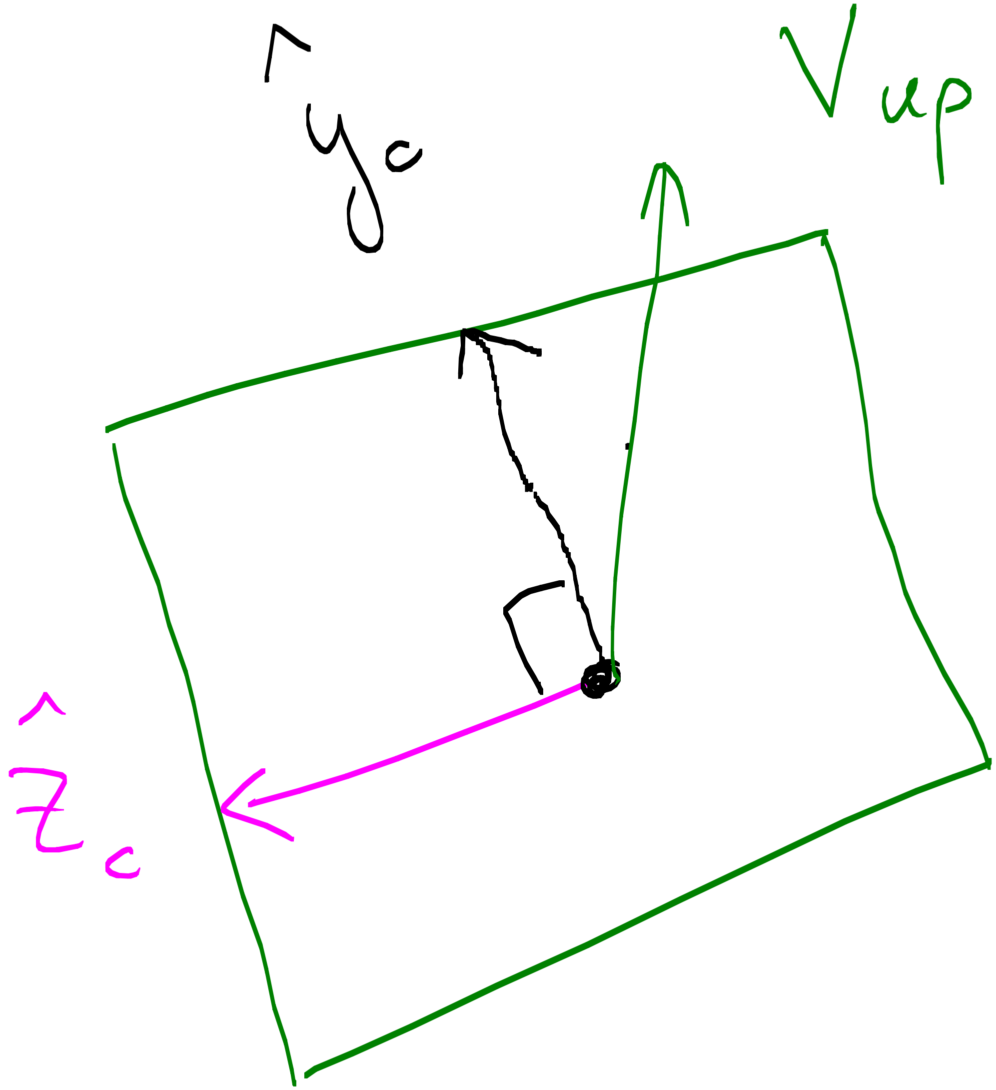
Which way is up ?

Define *any* 3D vector **Vup** such that

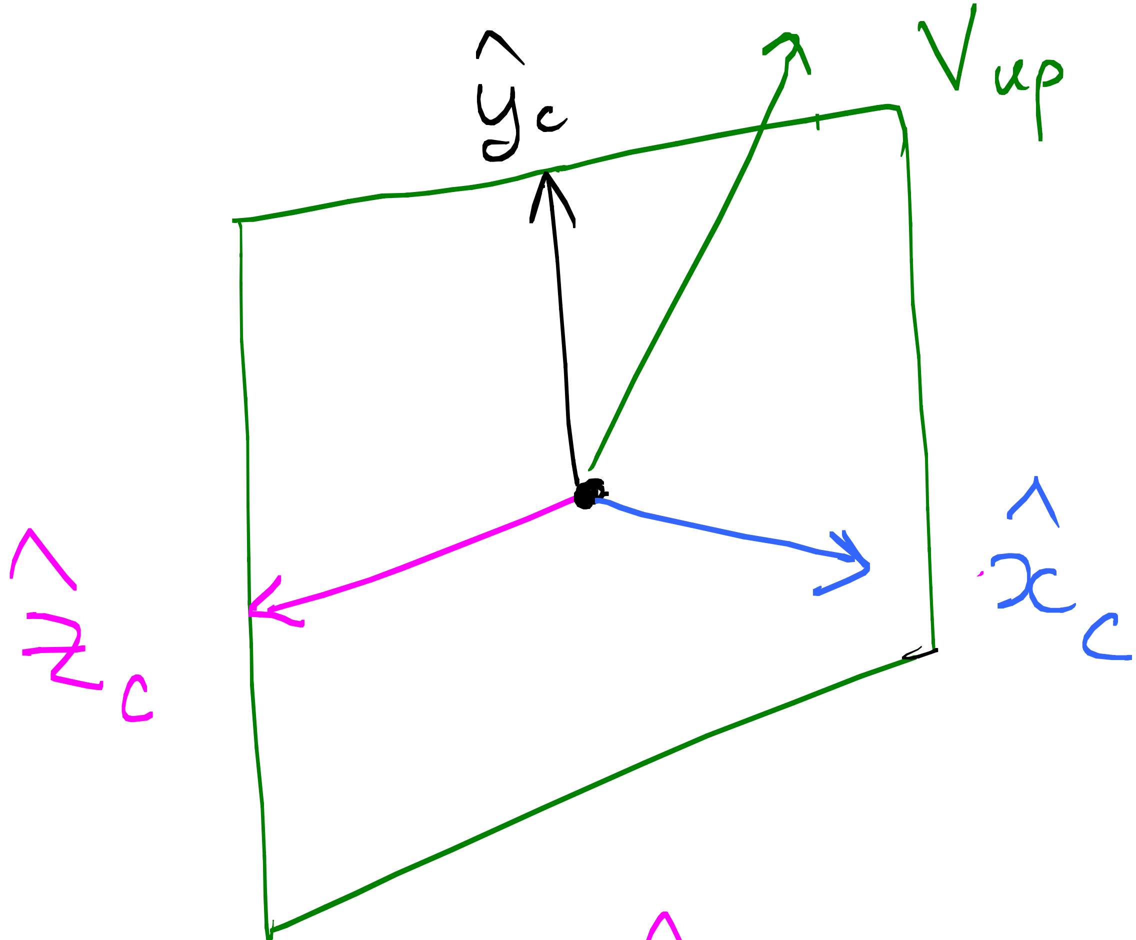
$$V_{up} \cdot \hat{z} \neq 0 .$$



This defines a plane, containing **Vup** and **z_c** .



$\hat{y}_c$ will be defined to lie in this plane.



$$\hat{x}_c = \frac{V_{up} \times \hat{z}_c}{\left| V_{up} \times \hat{z}_c \right|}$$

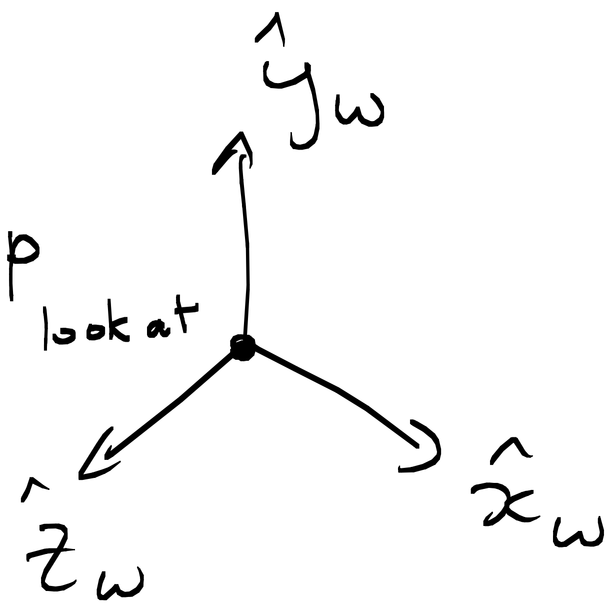
$$\hat{y}_c = \hat{z}_c \times \hat{x}_c$$

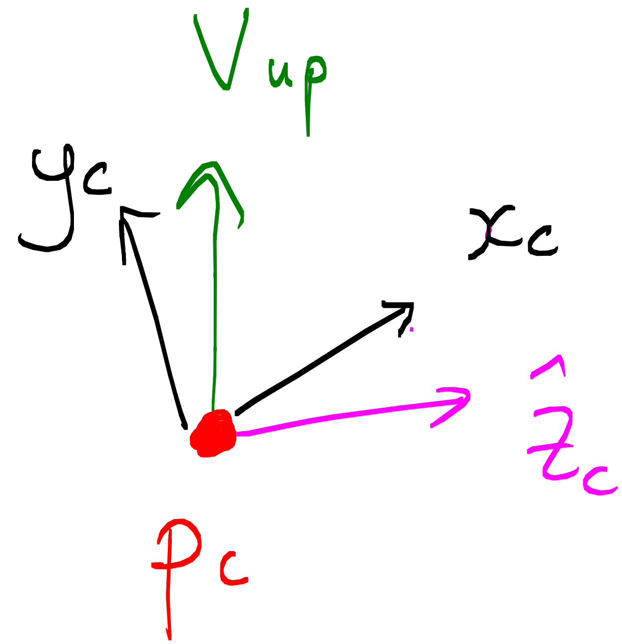
Example

$$\text{viewer} = (2, 1, 1)$$

$$\text{look at} = (0, 0, 0)$$

$$V_{up} = (0, 1, 0)$$



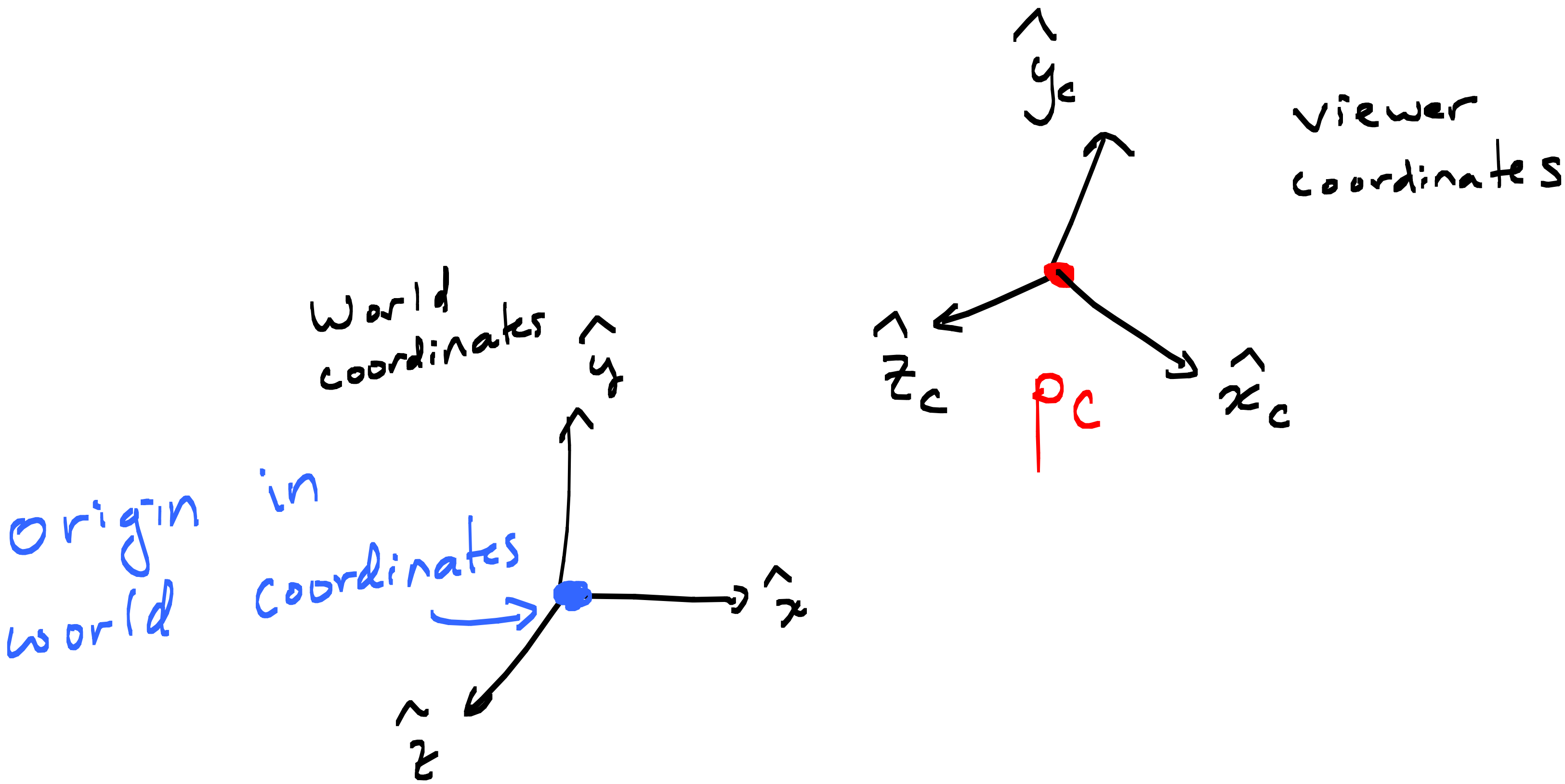


See lecture notes for the calculation.

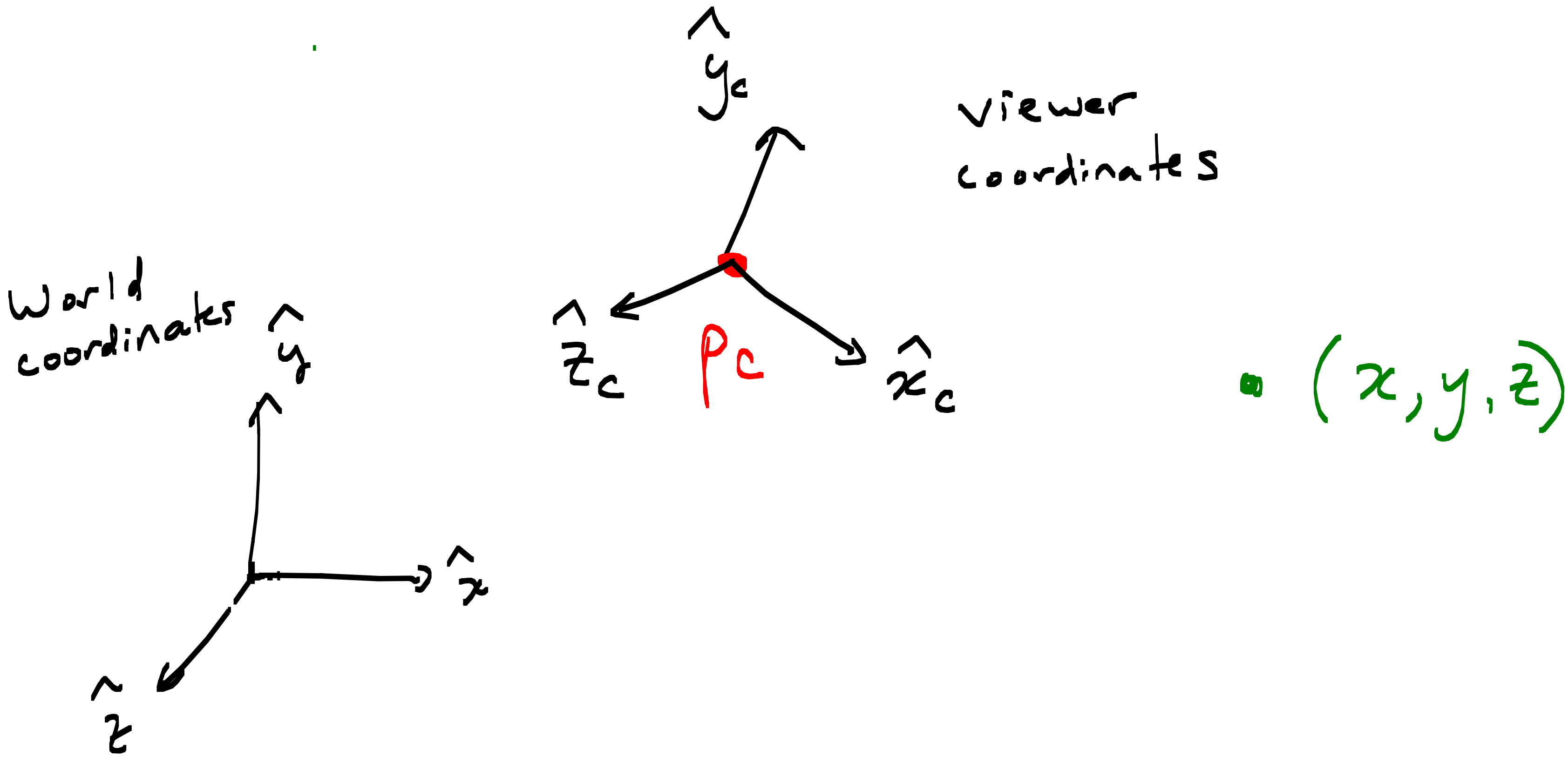
As a programmer using OpenGL, you don't have to compute these vectors. Instead you just define:

```
eye     =  ...     //  3D points
lookat  =  ...
up      =  ...

gluLookAt( eye[0], eye[1], eye[2],

           lookat[0], lookat[1], lookat[2],

           up[0], up[1], up[2] )
```

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



What is the relationship between the world coordinate system and the viewer's coordinate system?



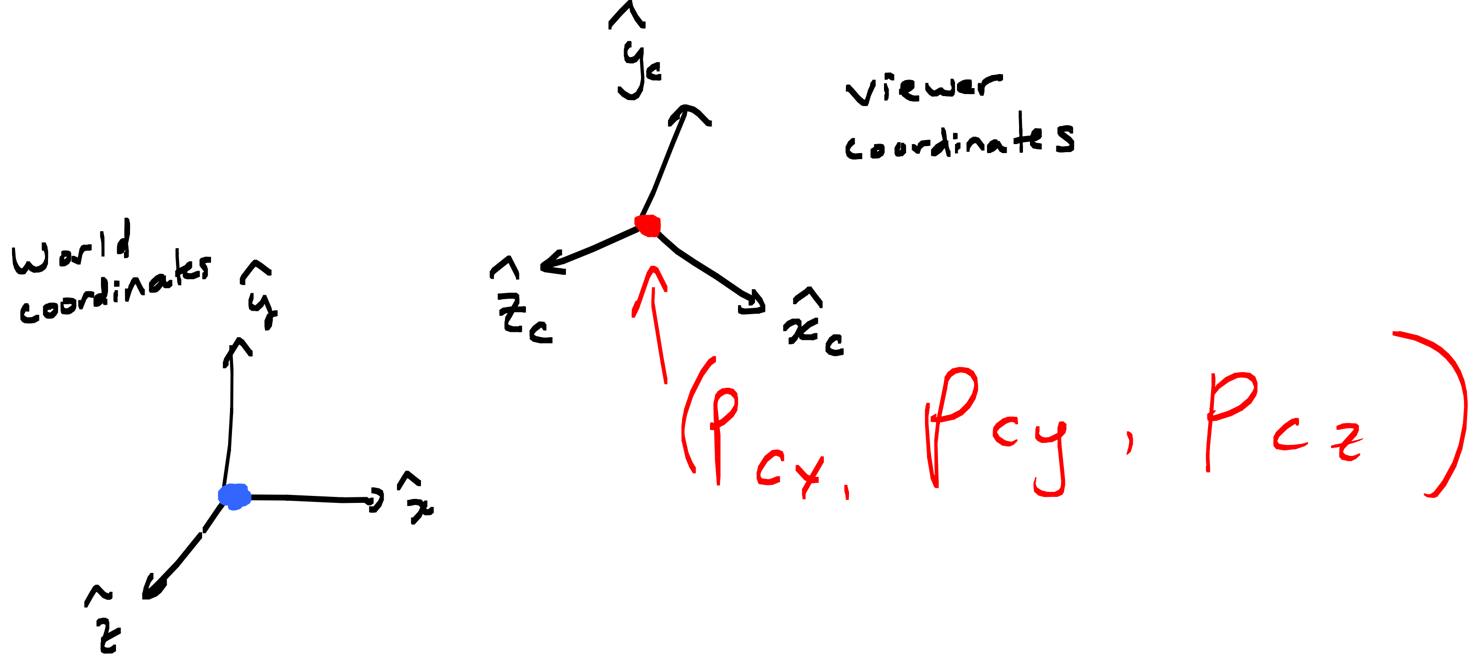
To re-map a general scene point (x,y,z) from world coordinates to viewer coordinates, we translate and rotate.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}_{\text{viewer}} = R \quad T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{world}}$$

$$M_{viewer \leftarrow world} = R \quad T$$

viewer/eye/camera $= \vec{p_c}$

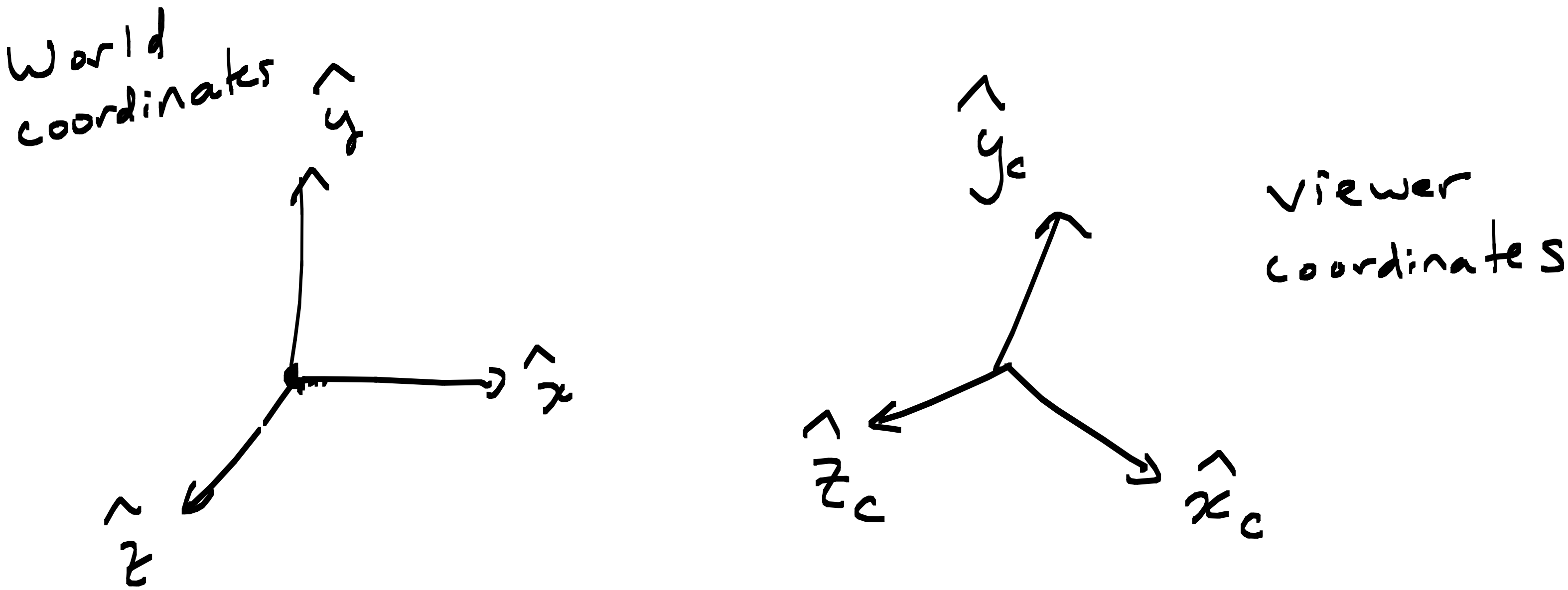
$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{cx} \\ 0 & 1 & 0 & -p_{cy} \\ 0 & 0 & 1 & -p_{cz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{cx} \\ 0 & 1 & 0 & -p_{cy} \\ 0 & 0 & 1 & -p_{cz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

R rotates into the viewer's orientation.

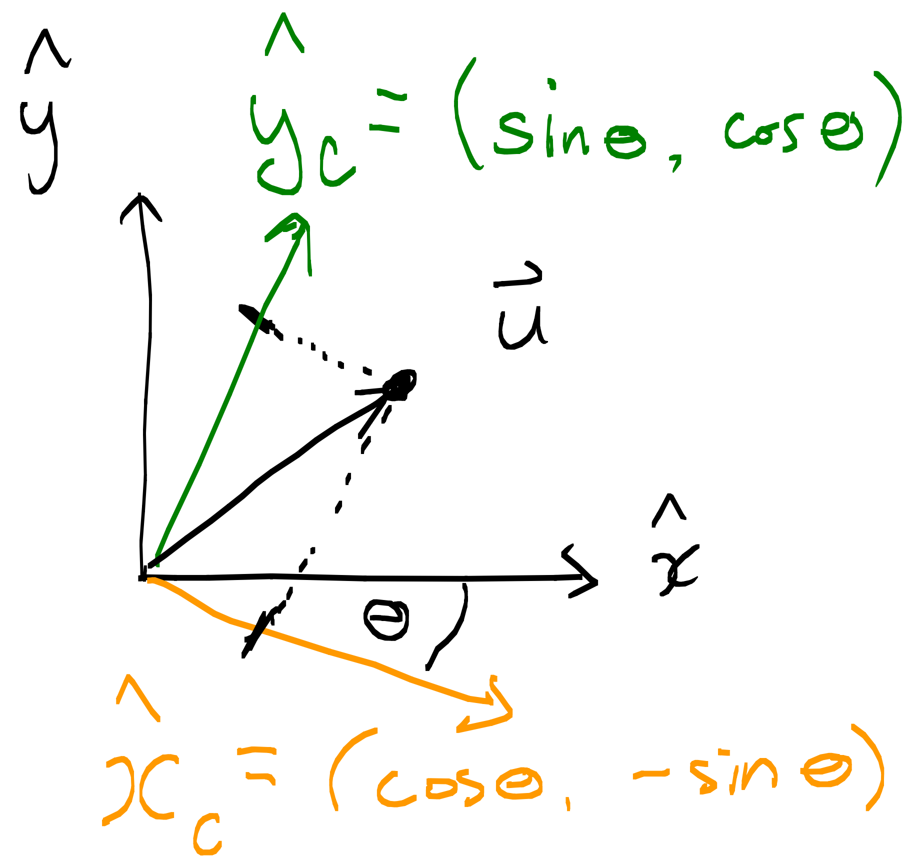


$$R = \begin{bmatrix} - & \hat{x}_c & - \\ - & \hat{y}_c & - \\ - & \hat{z}_c & - \end{bmatrix}_{3 \times 3}$$

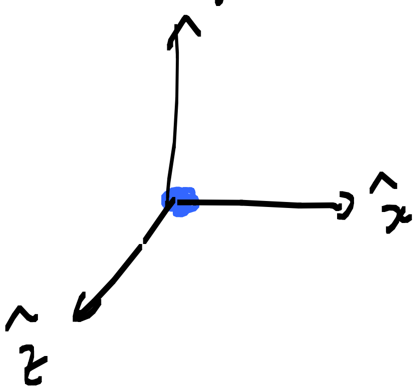
Recall slide 7 from lecture 2.

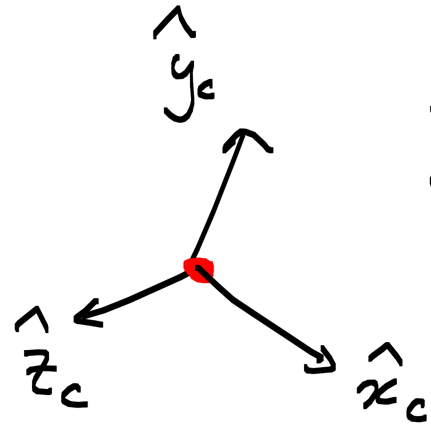
R maps to a new coordinate system by projecting onto new axes.

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



World coordinates





viewer coordinates

$$R \qquad\qquad T$$

$$\left[\begin{array}{ccc|c} \leftarrow & \hat{x}_c & \rightarrow & 0 \\ \leftarrow & \hat{y}_c & \rightarrow & 0 \\ \leftarrow & \hat{z}_c & \rightarrow & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \quad \begin{bmatrix} 1 & 0 & 0 & -p_{cx} \\ 0 & 1 & 0 & -p_{cy} \\ 0 & 0 & 1 & -p_{cz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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?

OpenGL Geometric "Primitives"

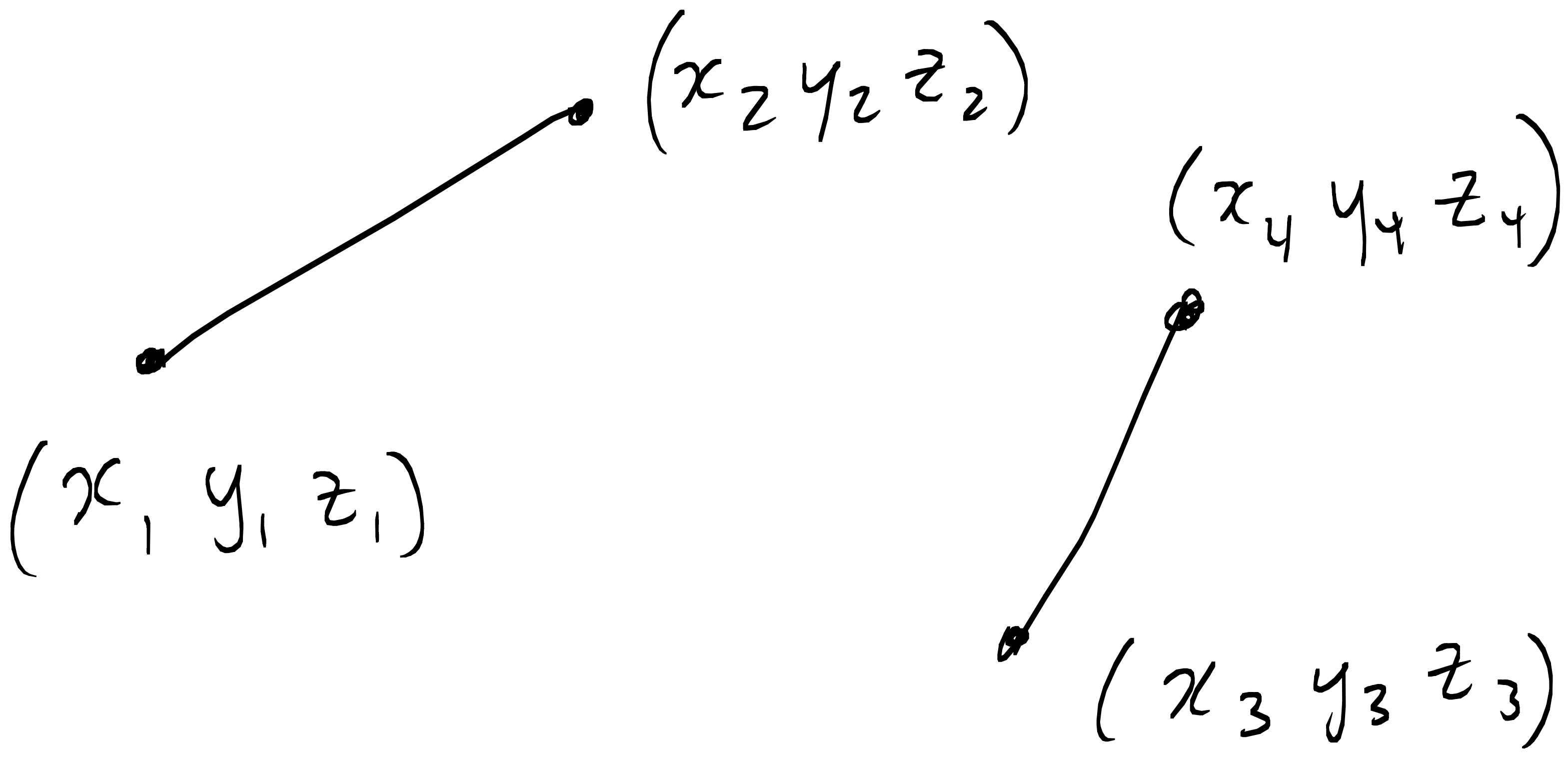
```
glVertex3f(x1,  y1, z1)
glVertex3f(x2,  y2, z2)
glVertex3f(x3,  y3, z3)
```

$(x_2\ y_2\ z_2)$

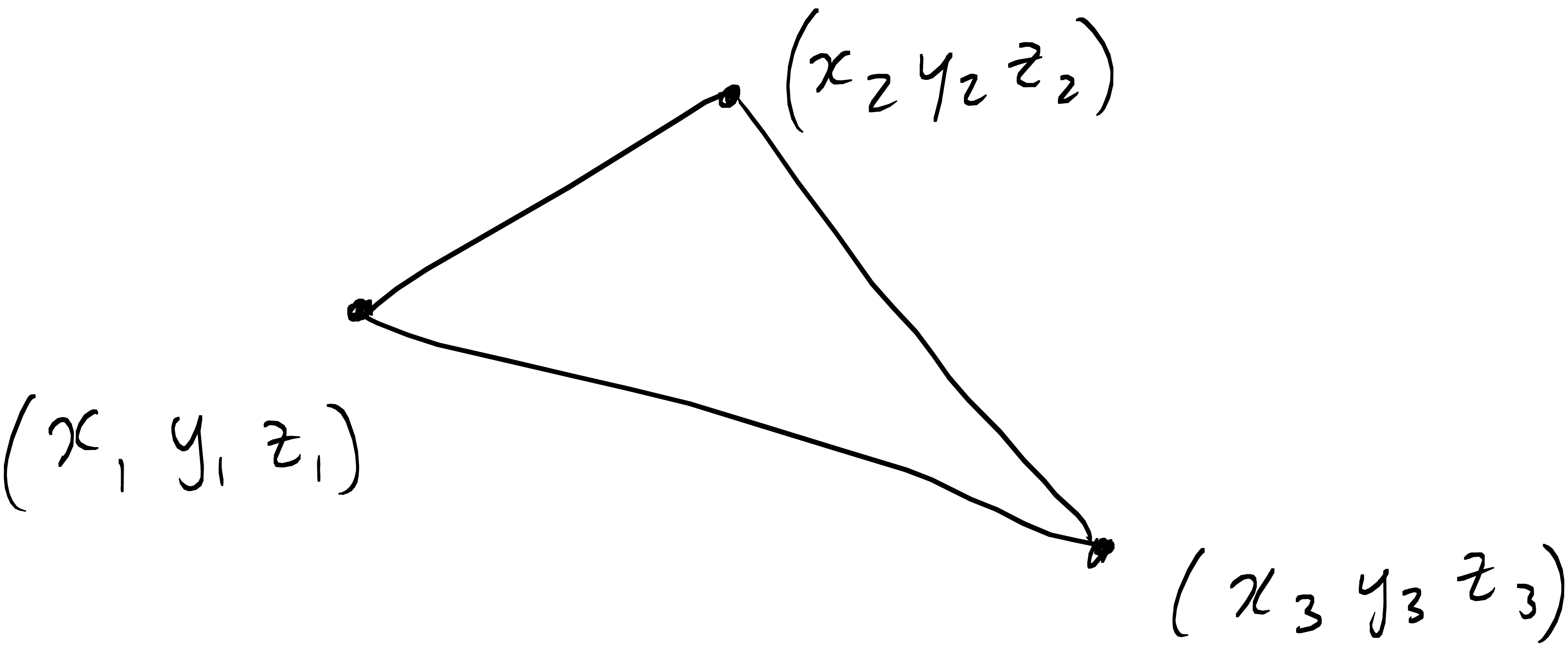
$(x_1,\ y_1\ z_1)$

$(x_3\ y_3\ z_3)$

```
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()
```

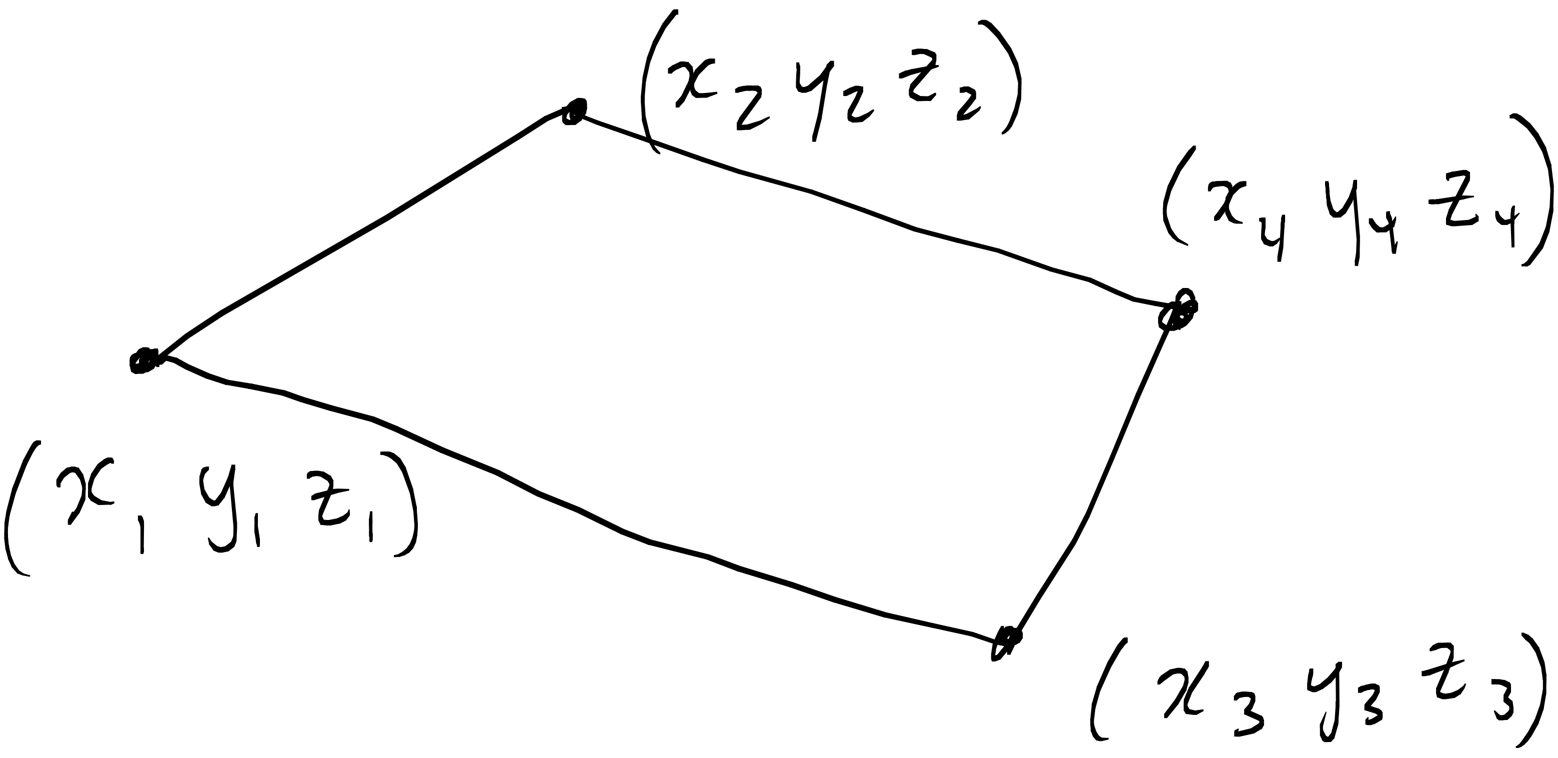


```
glBegin( GL_TRIANGLES )
   glVertex3f(x1,  y1 , z1)
   glVertex3f(x2,  y2 , z2)
   glVertex3f(x3,  y3, z3)
          //  more vertex triples gives more triangles
glEnd()
```



```
glBegin( GL_POLYGON )
    glVertex3f(x1,  y1 , z1)
    glVertex3f(x2,  y2 , z2)
    glVertex3f(x4,  y4 , z4)
    glVertex3f(x3,  y3 , z3)
glEnd()
```

problems if order is swapped



"Quadric" (Quadratic) Surfaces: examples

ellipsoid

$$a\,(x-x_0)^2 + b(y-y_0)^2 + c(z-z_0)^2 = 1$$

Cone

$$a(x-x_0)^2 + b(y-y_0)^2 = c(z-z_0)^2$$

paraboloid

$$ax = b(y-y_0)^2 + c(z-z_0)^2$$

Quadric Surfaces: General

$$ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0$$

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \underset{4\times 4}{Q} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

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

$$[\, wx,\ wy,\ wz,\ w \,] \quad \underset{4\times 4}{Q} \begin{bmatrix} wx \\ wy \\ wz \\ w1 \end{bmatrix} = 0$$

Q: What is this surface?

(if $a, b, c > 0$)

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

A: ellipsoid centered at origin

Q: What is this surface ? (a, b, c > 0)

$$\underbrace{\left(RT \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \right)}_{\vec{x}\,'}^{T} \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \underbrace{RT \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{\vec{x}\,'} = 0$$

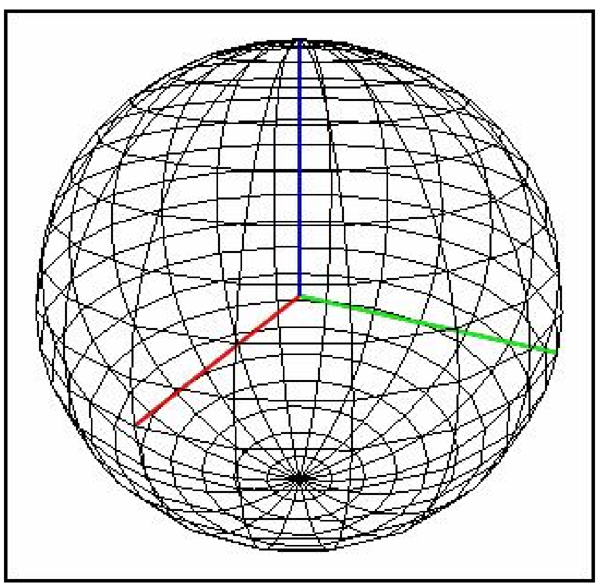
A: rotated and translated ellipsoid.

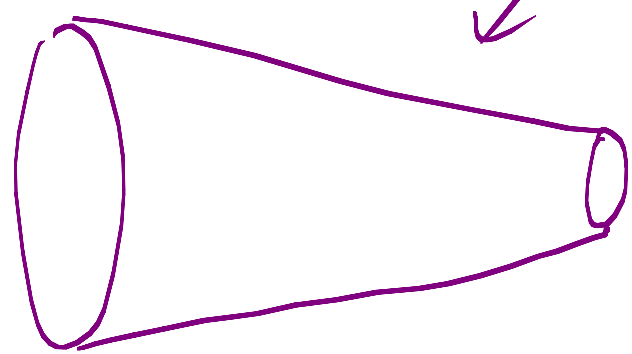
How to define quadric surfaces in OpenGL ?

GLUquadricObj myQuadric = gluNewQuadric()

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

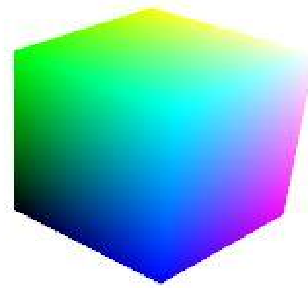
gluCylinder(myQuadric,   ...)

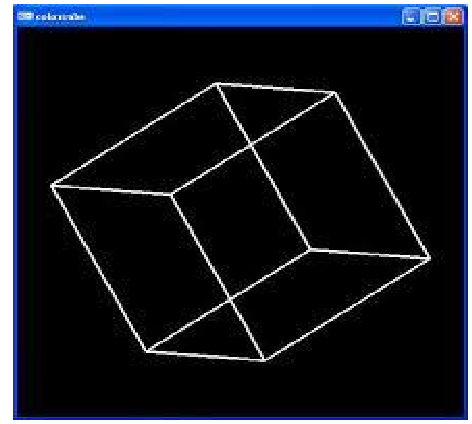




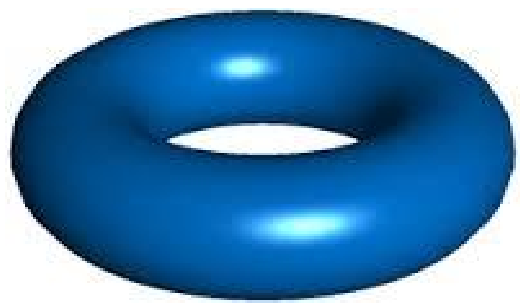
Non-quadric surfaces from OpenGL Utility Toolkit (GLUT)

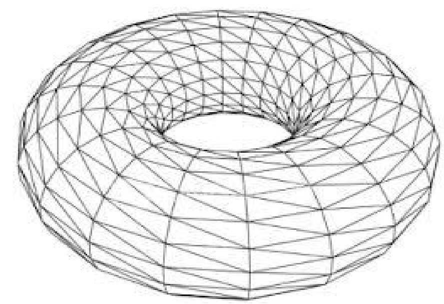
glutSolidCube()
glutWireCube()



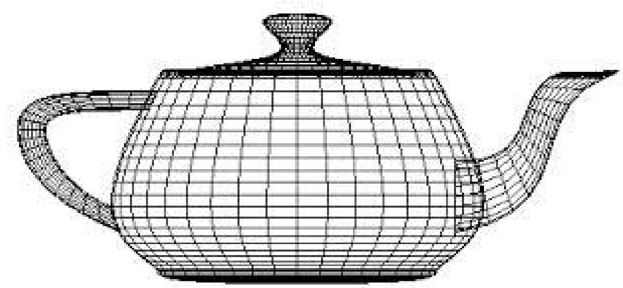


glutSolidTorus()
glutWireTorus()





glutSolidTeapot()
glutWireTeapot()



How to transform objects in OpenGL ?

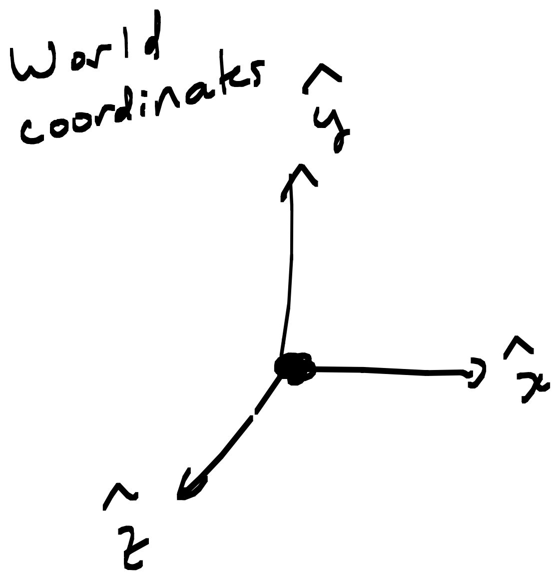
glRotatef(vx, vy, vz, angle)
glTranslatef(x, y, z)
glScalef(sx, sy, sz)

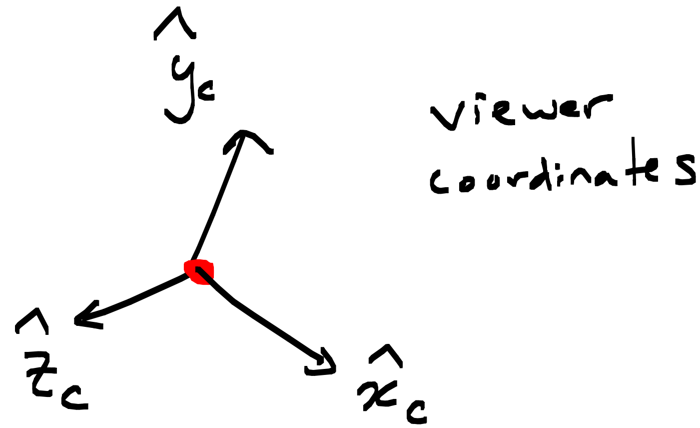
The parameters of each of these calls specify a 4x4 matrix.

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:

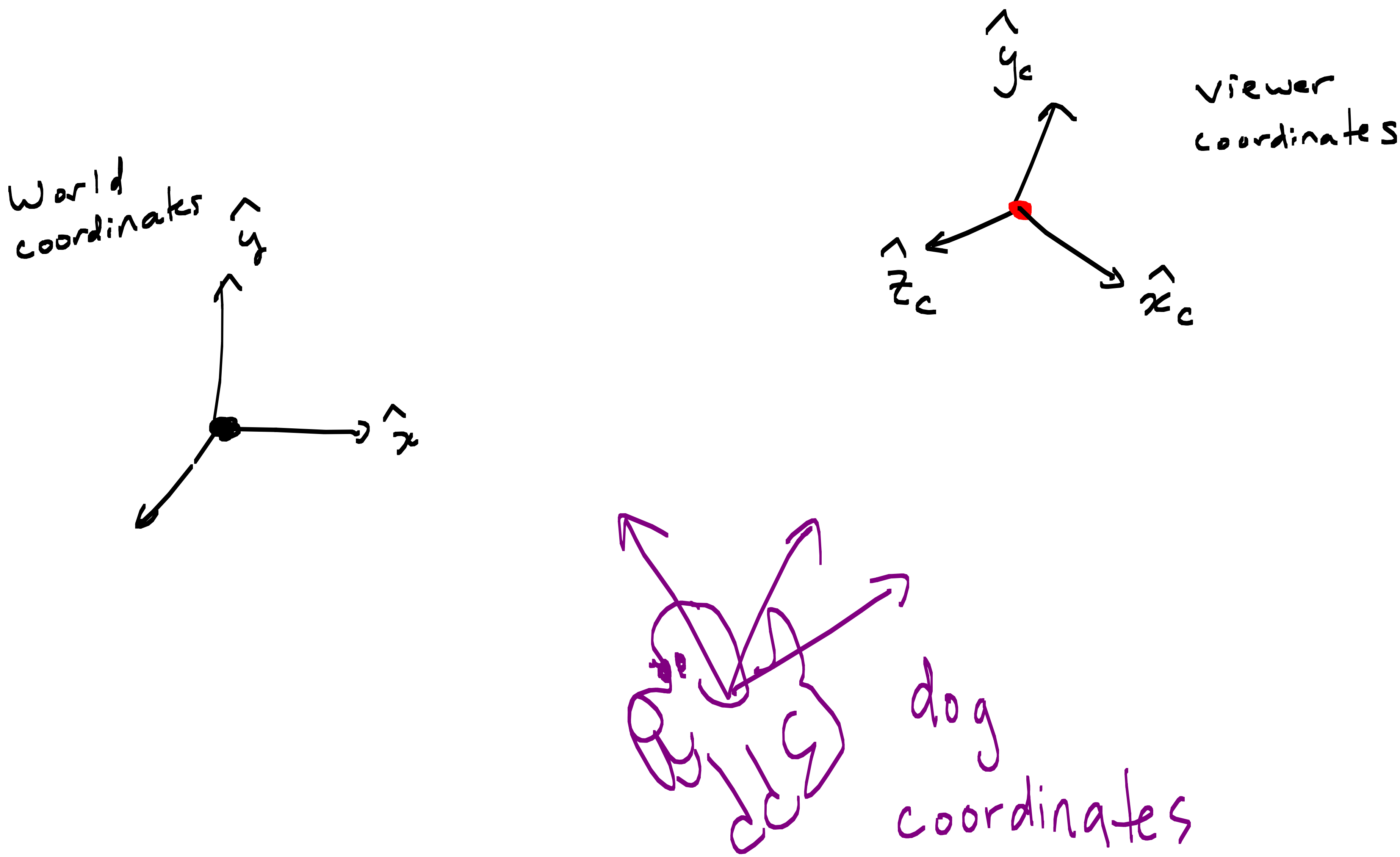




$$\tilde{x}_{\text{viewer}} = \underbrace{R \quad T}_{M_{\text{viewer} \leftarrow \text{world}}} \tilde{x}_{\text{world}}$$

(eye/camera)

How to transform from dog (object) coordinates to viewer coordinates?



$$\underbrace{M_{\text{viewer} \leftarrow \text{world}}}_{\text{gluLookAt(...)}} \qquad \underbrace{M_{\text{world} \leftarrow \text{dog}}}_{\substack{\text{glTranslate(...)} \\ \text{glRotate(...)}}}$$

$$M_{\text{viewer} \leftarrow \text{world}} \; M_{\text{world} \leftarrow \text{dog}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\text{dog}}$$

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

$$M_{\text{GL\_MODELVIEW}}$$

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* :

$$M_{\text{GL\_MODELVIEW}} = M_{\text{viewer} \leftarrow \text{world}} \, M_{\text{world} \leftarrow \text{obj}}$$

glMatrixMode(GL_MODELVIEW)
glLoadIdentity()

initializes: $M_{\text{GL\_MODELVIEW}} \leftarrow I$

ASIDE: How to examine the GL_MODELVIEW matrix ? (python)

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

Let M denote $M_{\text{GL\_MODELVIEW}}$

Q: What happens when you make these calls ?

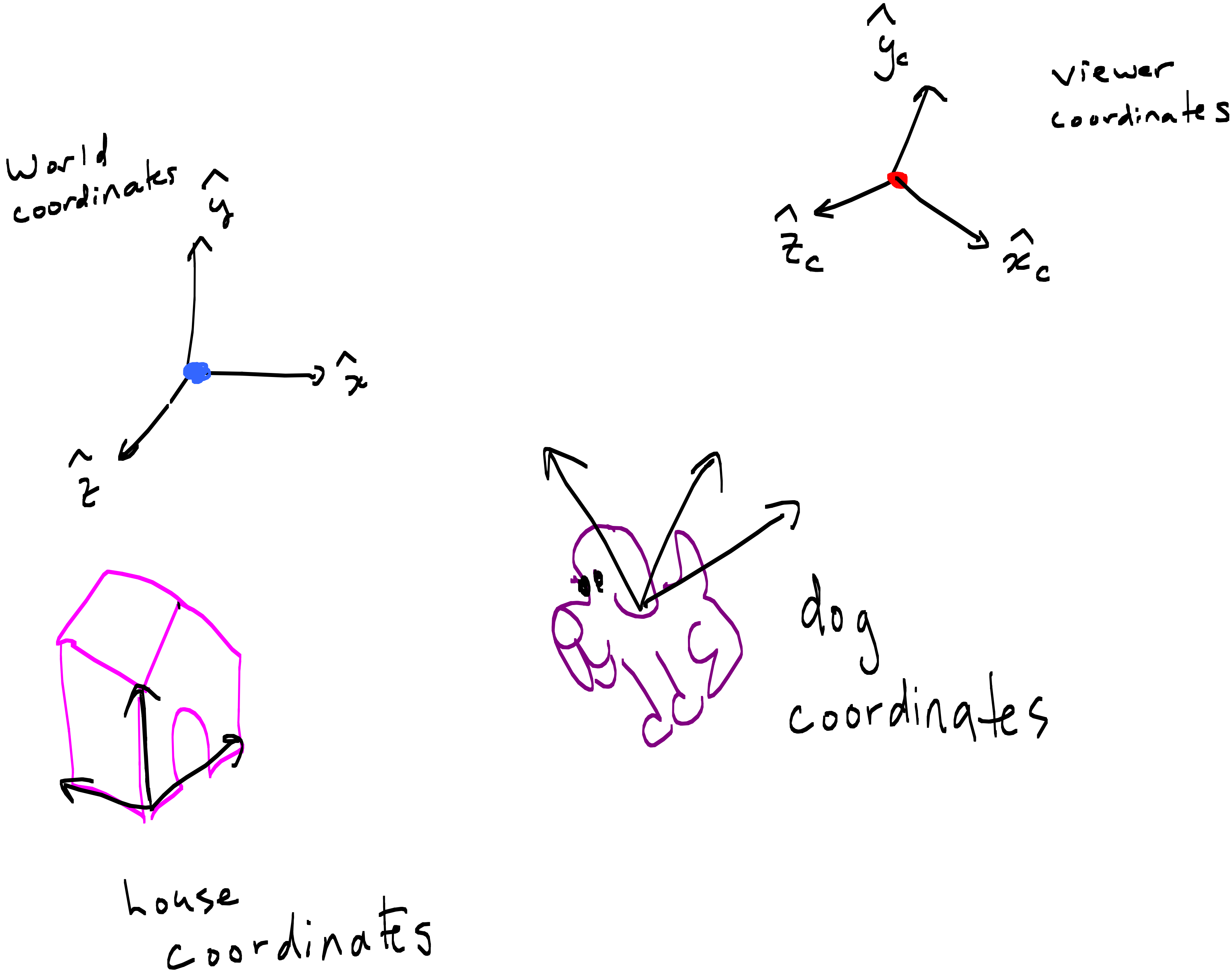
Answer:

gluLookAt(...) $M \leftarrow M \, M_{\text{viewer} \leftarrow \text{world}}$

glRotatef(...) $M \leftarrow M \, R$

glTranslatef(...) $M \leftarrow M \, T$

glScalef(...) $M \leftarrow M \, S$


World coordinates
$\hat{y}$
$\hat{x}$
$\hat{z}$
$\hat{y}_c$
viewer coordinates
$\hat{z}_c$
$\hat{x}_c$
dog coordinates
house coordinates

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?

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
gluLookAt( eye ... , lookat..., up ...)

glTranslate( ...)
glRotate(...)
drawDog()              //  glVertex() etc...

glTranslate( ...)
glRotate(...)
drawHouse()            //  glVertex() etc...
```

no!
this is relative to dog

Solution: use a stack of GL_MODELVIEW transformations.

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
gluLookAt( eye ... , lookat..., up ...)

glPushMatrix()
  glTranslate( ...)
  glRotate(...)
  drawDog()
glPopMatrix()

glPushMatrix()
  glTranslate( ...)
  glRotate(...)
  drawHouse()
glPopMatrix()
```

Summary of Today

viewer coordinate systems

view transformations : gluLookAt()

model transformations : glRotate(), glTranslate(), glScale()

GL_MODELVIEW transformation