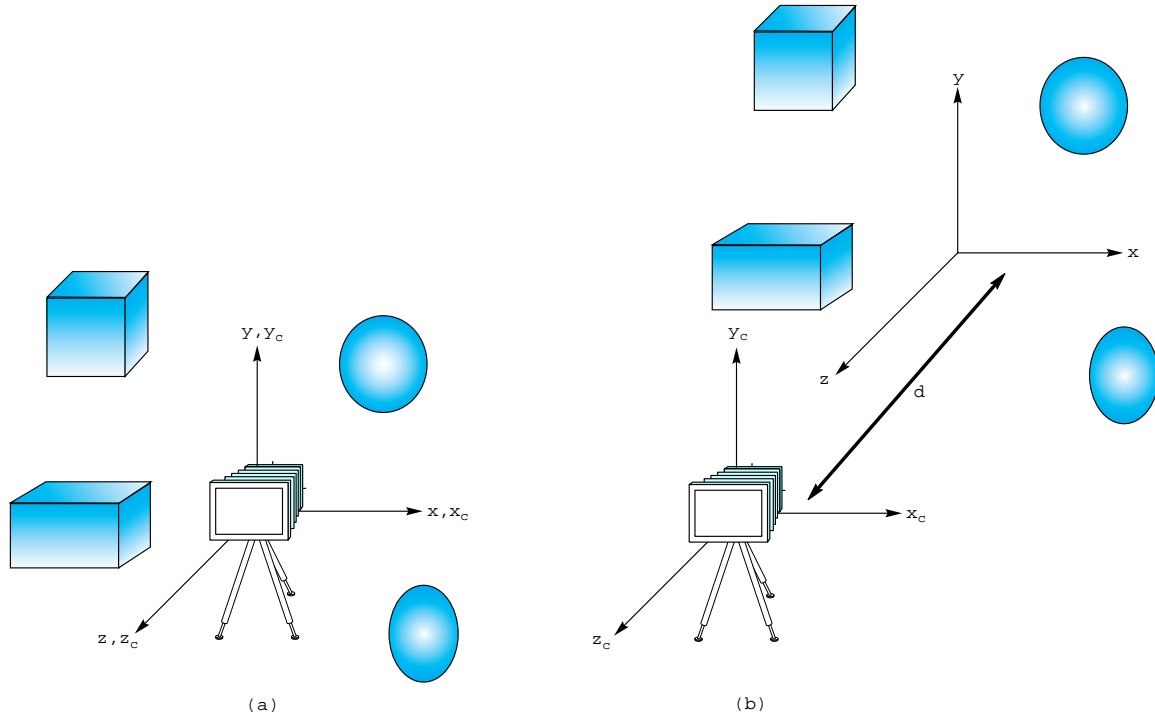
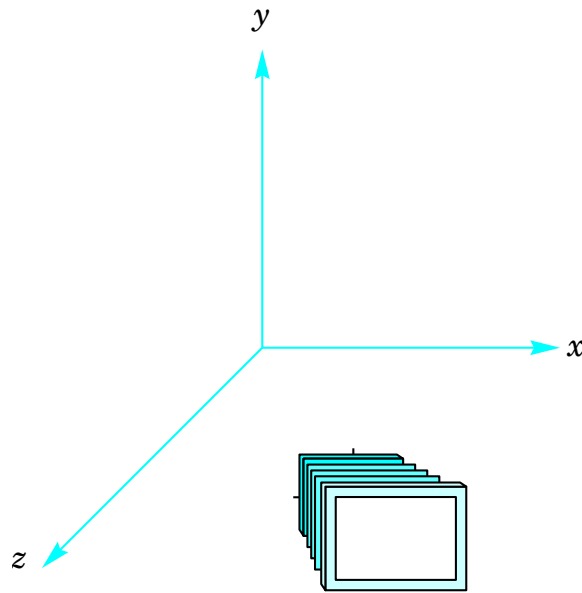


A simple modelview matrix:



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A less-simple modelview matrix:



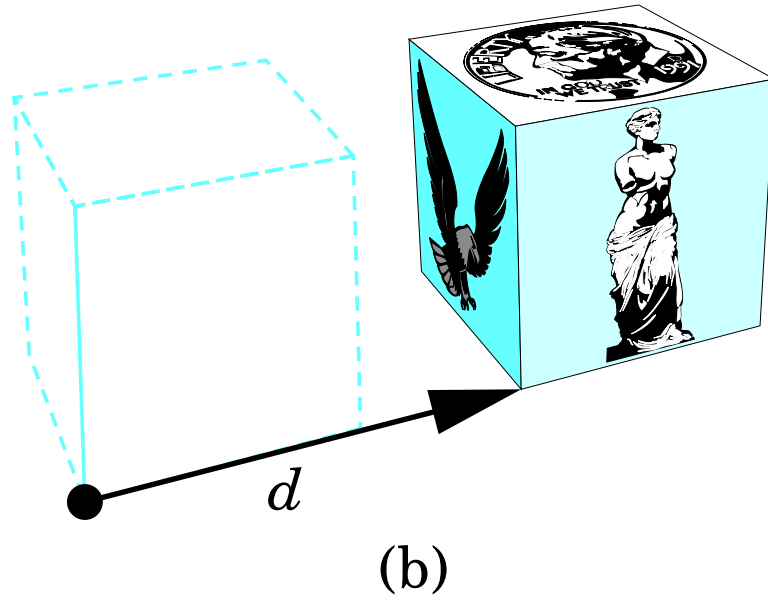
camera:

- centered at $(1, 0, 1, 1)^T$ in world coordinates
- points at origin, so $\vec{n} = (-1, 0, -1, 0)^T$
- up is $\vec{v} = (0, 1, 0, 0)$
- get third vector for camera frame by taking cross product: $\vec{u} = (1, 0, -1, 0)^T$

$$(M^T)^{-1} = \begin{bmatrix} 1 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

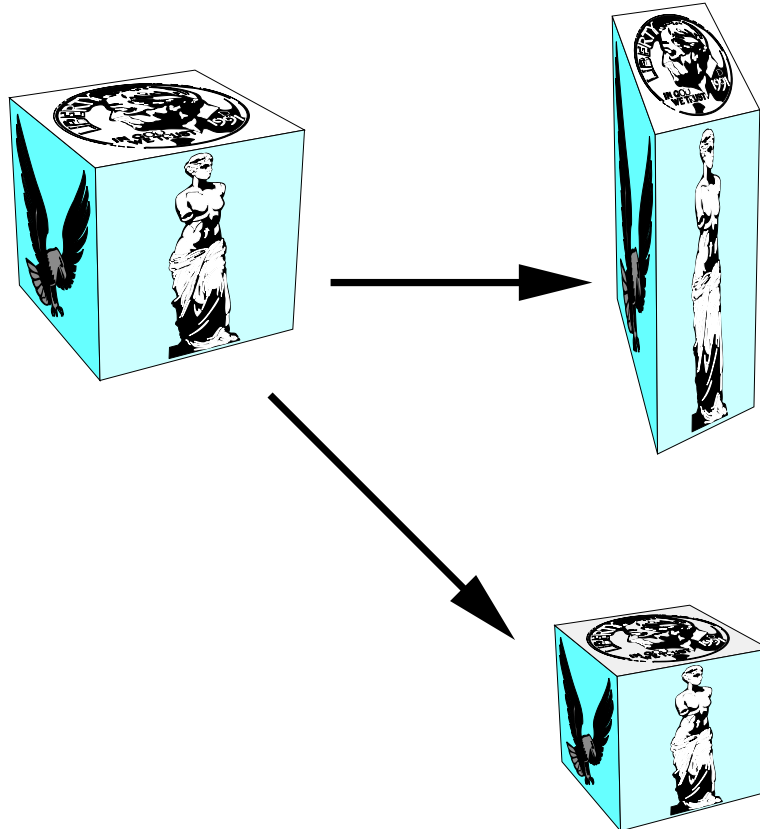
$$= \begin{bmatrix} 1/2 & 0 & -1/2 & 0 \\ 0 & 1 & 0 & 0 \\ -1/2 & 0 & -1/2 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation:



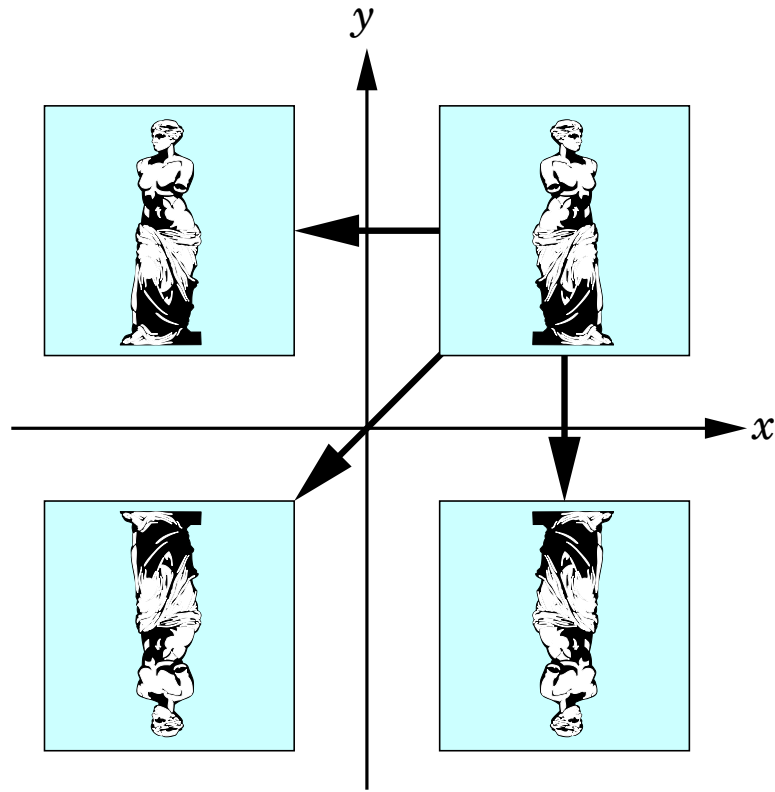
$$M^T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling:



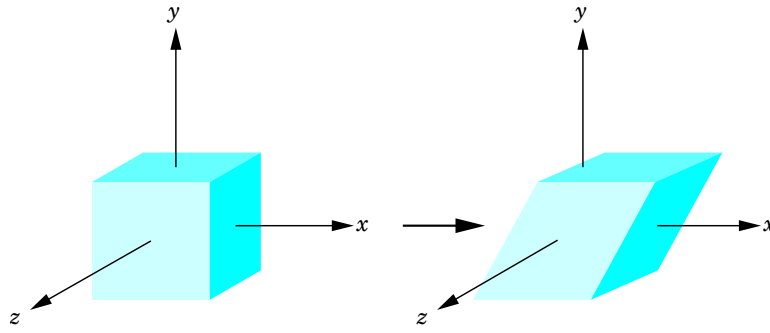
$$M^T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection:



$$M^T = ???$$

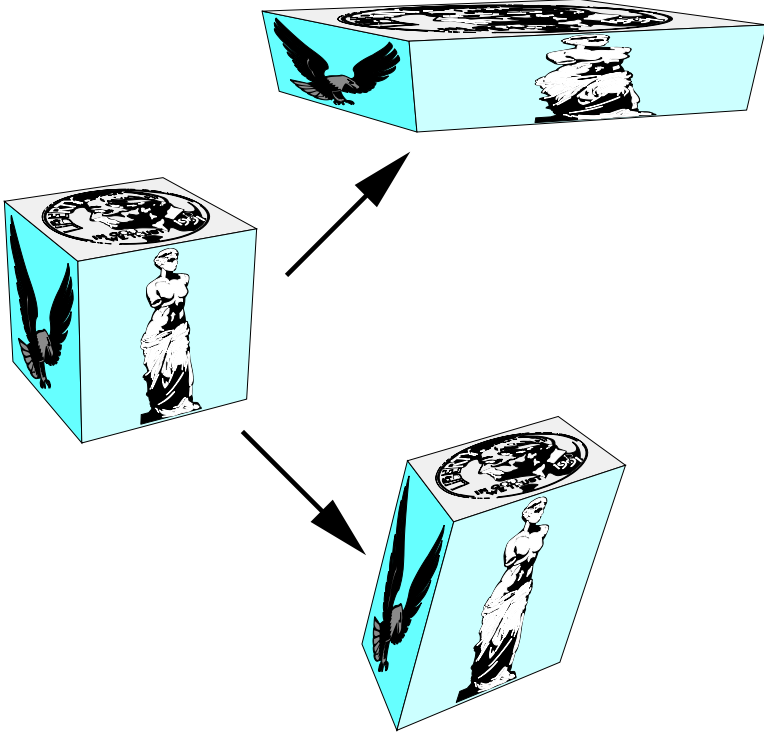
Shear:



$$M^T = \begin{bmatrix} 1 & \cot \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

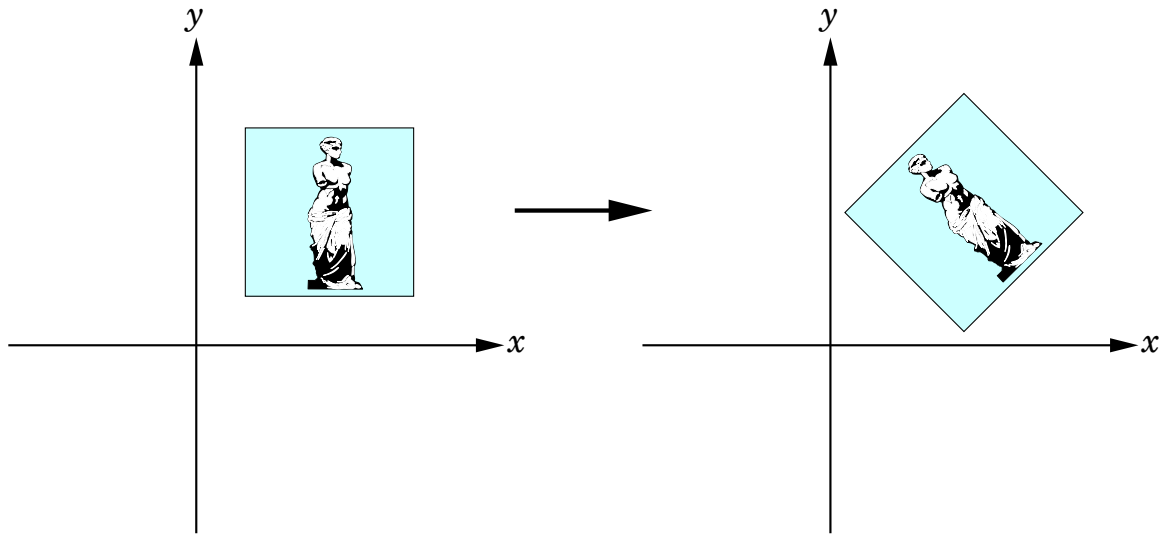
How would you shear in the y direction instead?

Non-rigid body transformations:

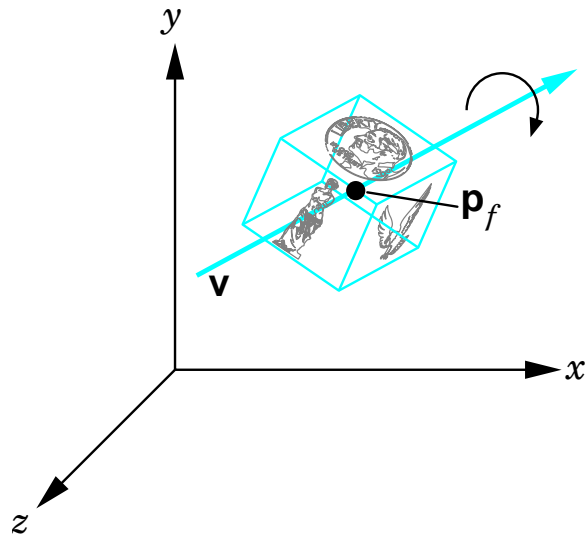


Rotation:

Easy in 2D:



Harder in 3D:



3D rotation matrices do not commute

Euler angles:

- rotation about “principal axes” of object:

$$R_x(\beta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta & 0 \\ 0 & \sin \beta & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\beta) = \begin{bmatrix} \cos \beta & -\sin \beta & 0 & 0 \\ \sin \beta & \cos \beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(all assuming right-handed coord systems)

Euler angles, cont.:

- all of those rotate around an (implicit) origin
- what if you want to rotate around the center of the object instead?

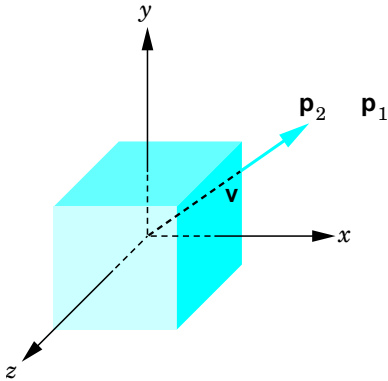
$$R_z(\beta) = \begin{bmatrix} \cos \beta & -\sin \beta & 0 & x_f - x_f \cos \beta + y_f \sin \beta \\ \sin \beta & \cos \beta & 0 & x_f - x_f \sin \beta - y_f \cos \beta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ick.

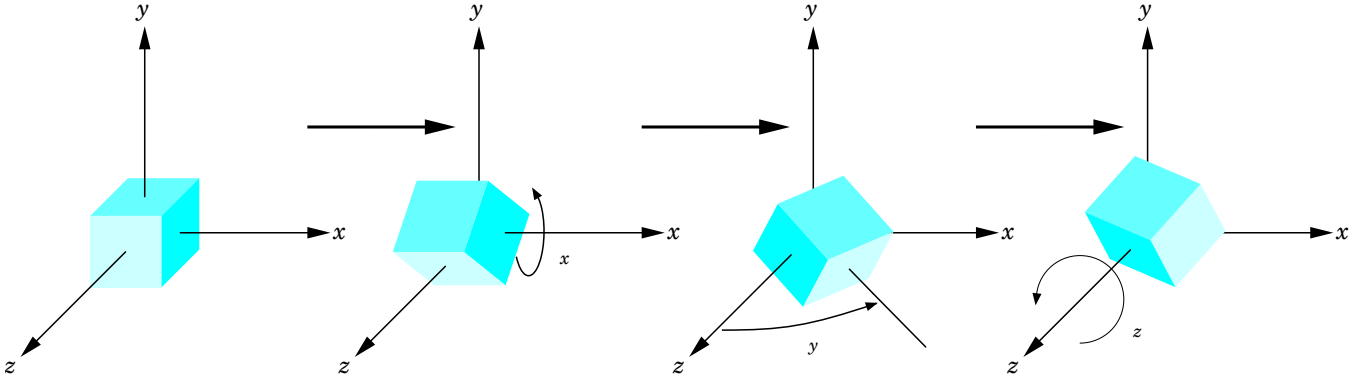
The idea: move to origin; rotate; move back. The matrix above is the product of the associated three transformations (move, rotate, move back). See section 4.8.1.

Rotation about arbitrary axis:

First, move the fixed point to the origin:



Then, perform individual constituent rotations:



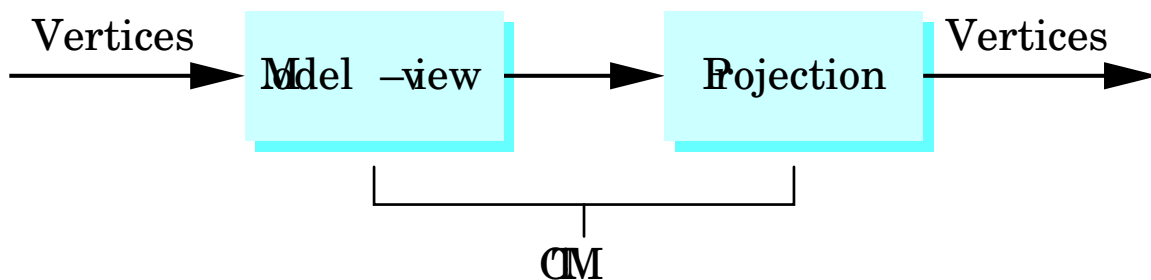
but finding θ_x , θ_y , and θ_z can be hard

Issues w.r.t. Euler angles:

- rotations are order-dependent and there are no conventions about which order to use
- hard to deal with rotation about arbitrary axis
- interpolation for animation violates movement “motif”
- widely used anyway, because they’re “simple”
- better idea: quaternions (later)

Transformation book-keeping:

- OpenGL: “current transformation matrix” is the product of the *modelview* matrix and the *projection* matrix:



- modelview: positions world relative to camera
- projection: projects onto viewport (chapter 5)

Manipulating the modelview matrix:

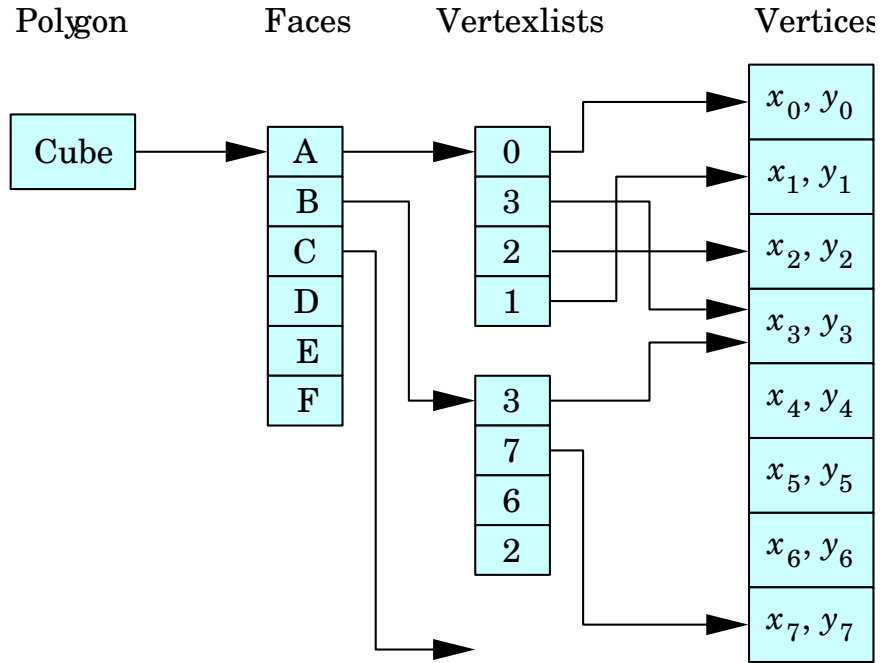
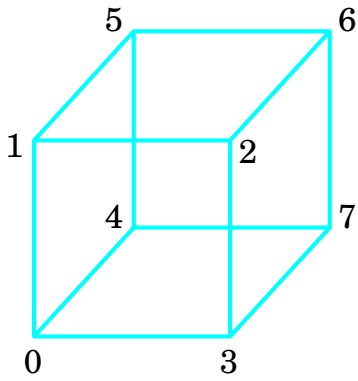
- can set/manipulate directly...but *be careful* (`glLoadMatrixf`, `glMultMatrixf`, `glLoadMatrix`)
- or do the usual “load identity and then frob” stuff:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(4.0, 5.0, 6.0);
glRotatef(45.0, 1.0, 2.0, 3.0);
    // args to glRotate are: angle and
    // vect coords of rotation axis
glTranslatef(-4.0, -5.0, -6.0);
```

What does this do?

- pushing, popping: `glPushMatrix`, `glPopMatrix` (to save modelview matrix while you're messing around)

Capturing the topology of the cube:



Does cube.c do this?