

Fundamental Concepts

- Transformations are critical to most graphics applications
- It's important to understand how they work
- OpenGL maintains a "current transformation"
 - Vertices are sent through it for processing

FIGURE 5.7 The OpenGL pipeline.

1

Representation

- We'll start with 2D transformations
- Most common transformations:
 - Translation – moving an object from one position to another
 - Scaling – changing the size of an object
 - Rotation – revolving an object around a pivot point
- We use a column vector representation for points
- The point $P(x,y)$ is represented as:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$
- Operations on points thus become vector/matrix operations

2

Translation

- Implemented by addition
- Translation of $P(x,y)$ to $P'(x',y')$ is represented as

$$P' = T + P$$
- We define the translation as

$$x' = x + t_x \quad y' = y + t_y$$
- Representing these as vectors,

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

3

Translation

- For objects consisting of multiple vertices, we translate the object by applying the point translation to each vertex

4

Scaling

- Implemented with multiplication
- Unlike translation, scaling is relative to the origin
 - Translation is relative to the original position of the object
- Point $P(x,y)$ is scaled to $P'(x',y')$ with the products

$$x' = s_x \cdot x \quad y' = s_y \cdot y$$
- In matrix form, $P' = S \cdot P$ is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

5

Scaling

- If $s_x = s_y$, we have a *uniform* scaling:
- If $s_x \neq s_y$, we have a *differential* scaling:

6

Department of Computer Science **Rotation**

- More complicated
- We rotate points through an angle θ about the origin
 - Positive angles rotate counter-clockwise
- Defined mathematically by

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

$$y' = x \sin \theta + y \cos \theta$$
- In matrix form, $P' = R \cdot P$ is:

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

5/3/2007 7

Department of Computer Science **Rotation**

- Here is the result of rotating 45° about the origin:

5/3/2007 8

Department of Computer Science **Rotation**

- We derive our rotation equations as follows
- Consider points $P(x,y)$ and $P'(x',y')$
- Trigonometry tells us that

$$(A) \quad x = r \cos \phi \quad y = r \sin \phi$$
- Also,

$$(B) \quad x' = r \cos(\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta,$$

$$y' = r \sin(\theta + \phi) = r \cos \phi \sin \theta + r \sin \phi \cos \theta$$
- Substituting (A) into (B) yields

$$x' = x \cos \theta - y \sin \theta, \quad y' = x \sin \theta + y \cos \theta$$

5/3/2007 9

Department of Computer Science **Transformations**

- **Translation, Scaling, and Rotation** are examples of *affine* transformations
 - Preserve parallelism of lines, but not lengths or angles
- Other affine transformations include reflection and shearing

5/3/2007 10

Department of Computer Science **Reflection and Shearing**

- We reflect by inverting coordinate signs and/or reversing coordinate values:
 - Reflect around the y axis: $x' = -x, \quad y' = y$
 - Reflect around the x axis: $x' = x, \quad y' = -y$
 - Reflect around the positive diagonal ($m = 1$): $x' = y, \quad y' = x$
 - Reflect around the negative diagonal ($m = -1$): $x' = -y, \quad y' = -x$
- We shear by modifying each coordinate
 - Shear in y direction: $x' = x, \quad y' = bx + y$
 - Shear in x direction: $y' = y, \quad x' = x + ay$

5/3/2007 11

Department of Computer Science **Matrix Representations**

- Here are the "big three" transformations, in matrix form:

$$P' = T + P$$

$$P' = S \cdot P$$

$$P' = R \cdot P$$
- Unfortunately, translation is treated differently from rotation and scaling
- Ideally, we want to treat them all the same way
 - This will allow us to easily combine different operations

5/3/2007 12

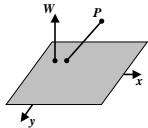
Homogeneous Coordinates

- Solution: use *homogeneous coordinates*
 - First developed in the mid 1940s in geometry
- Using homogeneous coordinates allows us to treat all three operations as multiplications
- Basic modification: add a third coordinate to a point
- The point (x,y) becomes (x,y,W)
- Two sets of homogeneous coordinates (x,y,W) and (x',y',W') represent the same point if and only if $x'=nx$, $y'=ny$, and $W'=nW$
 - Examples: $(2,4,6)$ $(4,8,12)$ $(1,2,3)$

5/3/2007 13

Homogeneous Coordinates

- Normally, we use triples to represent points in 3-space
- If we collect all triples (tx,ty,tW) with $t \neq 0$, we get a line



- We homogenize the point by dividing by W
- This gives us the point $(x,y,1)$
- Thus, homogenized points form the plane defined by $W = 1$

5/3/2007 14

Homogeneous Coordinates – Translation

- Points are now three-element vectors
- Transformation matrices must now be 3x3
- This now allows us to use multiplication for translation:

$$P' = T \cdot P \text{ becomes } P' = T(d_x, d_y) \cdot P$$
- In matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5/3/2007 15

Homogeneous Coords – Scaling, Rotation

- Scaling and rotation become

$$P' = S(s_x, s_y) \cdot P \quad P' = R(\theta) \cdot P$$
- In matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

5/3/2007 16

Homogeneous Coordinates – Reflection

- Around the x axis:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Around the y axis:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Around the diagonals:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5/3/2007 17

Homogeneous Coordinates – Shearing

- Shearing is a bit different:

$$SH_x \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$SH_y \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5/3/2007 18

Caution

- Note that some texts use row vectors rather than column vectors for points
- They also alter the multiplication sequence to pre-multiply by row vectors rather than post-multiplying by column vectors:
 $P \cdot M = P'$
- We must perform a transposition to switch forms:
 $(P \cdot M)^T = M^T \cdot P^T$

5/3/2007 19

Multiple Translations

- Intuitively, we know that translating a point twice is equivalent to a single, combined translation
 $P' = T(d_{x1}, d_{y1}) \cdot P$ then $P'' = T(d_{x2}, d_{y2}) \cdot P'$
 results in $P'' = T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1}) \cdot P$
 which is $P'' = T(d_{x1}+d_{x2}, d_{y1}+d_{y2}) \cdot P$

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 20

Multiple Scaling

- The same is true for scaling (multiplicative, not additive)
 $P' = S(s_{x1}, s_{y1}) \cdot P$ then $P'' = S(s_{x2}, s_{y2}) \cdot P'$
 results in $P'' = S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) \cdot P$
 which is $P'' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x2} \cdot s_{x1} & 0 & 0 \\ 0 & s_{y2} \cdot s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 21

Multiple Rotation

- Rotation, like translation, is additive
 $P' = R(\theta) \cdot P$ then $P'' = R(\phi) \cdot P'$
 results in $P'' = R(\phi) \cdot R(\theta) \cdot P$
 which is $P'' = R(\theta + \phi) \cdot P$

$$\begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta + \phi) & -\sin(\theta + \phi) & 0 \\ \sin(\theta + \phi) & \cos(\theta + \phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 22

Composition of Transformations

- Recall that rotation is about the origin
- What if we want to rotate an object about another point – say, an arbitrary point P ?
- We can combine a series of transformations into a *composite* transformation
- For example, to rotate about point P , we:
 - Translate P to the origin
 - Rotate
 - Translate the origin back to P

5/3/2007 23

Composition of Transforms – Rotation

- To translate point $P_i(x_i, y_i)$ to the origin, we translate by the negative coordinates, $T(-x_i, -y_i)$
- We then rotate this using $R(\theta)$
- Finally, we translate back by $T(x_i, y_i)$
- The full sequence: $T(x_i, y_i) \cdot R(\theta) \cdot T(-x_i, -y_i)$

$$\begin{bmatrix} 1 & 0 & x_i \\ 0 & 1 & y_i \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_i \\ 0 & 1 & -y_i \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos \theta & -\sin \theta & x_i(1 - \cos \theta) + y_i \sin \theta \\ \sin \theta & \cos \theta & y_i(1 - \cos \theta) - x_i \sin \theta \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 24

Composition of Transforms – Scaling

- Scaling is also relative to the origin
- To scale relative to an arbitrary point, we translate that point to the origin, scale by $S(s_x, s_y)$, and translate back
- The full sequence: $T(x_i, y_i) \cdot S(s_x, s_y) \cdot T(-x_i, -y_i)$

$$= \begin{bmatrix} 1 & 0 & x_i \\ 0 & 1 & y_i \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_i \\ 0 & 1 & -y_i \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & x_i(1-s_x) \\ 0 & s_y & y_i(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 25

Applications of Transformations

- So, where do we actually use these?
- Recall our window-to-viewport discussion
 - Mapping the contents of the window to the viewport is often called the *viewing transformation*
- We translate the window to the origin, scale it to the viewport size, and translate to the viewport position
- Note that this also involves a coordinate system change
 - World coordinates to view/screen coordinates

5/3/2007 26

Viewing Transformation

- Consider the world window and viewport locations:

- We can easily derive the necessary translation deltas from these coordinates
- What about the scaling values?

5/3/2007 27

Viewing Transformation

- Scaling values are the ratios of the dimensions

- So, the scaling values are:

$$s_x = \frac{u_{max} - u_{min}}{x_{max} - x_{min}} \quad s_y = \frac{v_{max} - v_{min}}{y_{max} - y_{min}}$$

5/3/2007 28

Viewing Transformation

- The sequence: $T(u_{min}, v_{min}) \cdot S(s_x, s_y) \cdot T(-x_{min}, -y_{min})$
- The matrices:

$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & -x_{min} \times \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & -y_{min} \times \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 29

Efficiency

- In general, composite matrices look like this:

$$\begin{bmatrix} r11 & r12 & d_x \\ r21 & r22 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Applying this to a point involves nine multiplications and six additions

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & d_x \\ r21 & r22 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5/3/2007 30

Efficiency

- We can cut this time significantly by noting the structure of the matrix
 - The bottom row, [0 0 1], always yields 1 as the bottom coordinate of the resulting point
 - The rightmost column entries are always multiplied against 1, the bottom coordinate of the original point
- So, we can eliminate five multiplications and two additions:

$$x' = r11 \cdot x + r12 \cdot y + d_x$$

$$y' = r21 \cdot x + r22 \cdot y + d_y$$
- This is especially significant, given that we may be applying this to hundreds or thousands of vertices

5/3/2007 31

Moving From 2D to 3D

- Not a lot of difference
- Again, we'll use homogenized coordinates
 - $P(x,y,z,W)$, with $W \neq 0$
- Add a z coordinate to our column vectors
- Add a fourth row and fourth column to our matrices
- Point $P(x,y,z)$: Identity matrix:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5/3/2007 32

Standard Transformation Matrices

- Translation:

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Scaling:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- We can verify these:

$$T(d_x, d_y, d_z) \cdot [x, y, z, 1]^T = [x+d_x, y+d_y, z+d_z, 1]^T$$

$$T(s_x, s_y, s_z) \cdot [x, y, z, 1]^T = [x \cdot s_x, y \cdot s_y, z \cdot s_z, 1]^T$$

5/3/2007 33

Rotation

- Rotation matrix differs for each axis of rotation
- Consider our 3D coordinate system:
 - Called a *righthanded* coordinate system
 - Looking along the axis from positive coordinates to the origin, rotation is counterclockwise

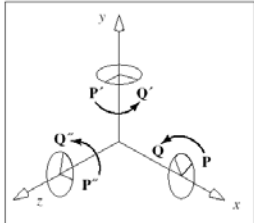


FIGURE 5.25 Positive rotations about the three axes.

5/3/2007 34

Rotation Around Z

- 2D rotation is just rotation around the z axis:

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

5/3/2007 35

Rotation Around X and Y

- Rotation around the x axis moves from y to z:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Rotation around the y axis moves from z to x:

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

5/3/2007 36

Composition of 3D Transformations

- Basically the same as in 2D
- Consider this diagram:

- Our goal is to transform it so that it looks like this:

5/3/2007 37

Composition of 3D Transforms

- This can be seen as a sequence of four transforms:
 - Translate P_1 to the origin
 - Rotate about y so that P_1P_2 lies in the (y,z) plane
 - Rotate about x so that P_1P_2 lies along the z axis
 - Rotate about z so that P_1P_3 lies in the (y,z) plane

5/3/2007 38

Composition of 3D Transforms

- Translation to the origin is just $T(-x_p, -y_p, -z_p)$
 - $T(-x_p, -y_p, -z_p) \cdot P_1(x_p, y_p, z_p) = P'_1(0, 0, 0)$
 - $T(-x_p, -y_p, -z_p) \cdot P_2(x_2, y_2, z_2) = P'_2(x_2 - x_p, y_2 - y_p, z_2 - z_p)$
 - $T(-x_p, -y_p, -z_p) \cdot P_3(x_3, y_3, z_3) = P'_3(x_3 - x_p, y_3 - y_p, z_3 - z_p)$
- Determining rotations is a bit more involved
- We begin by projecting P'_2 onto the (x,z) plane
 - Point $(x'_2, 0, z'_2)$
 - Distance D_1 from origin
 - Angle from the (x,y) plane is θ
 - Angle of rotation around y will be $-(90-\theta)$, which is just $\theta - 90$

5/3/2007 39

Composition of 3D Transforms

- Our rotation is $R(\theta - 90)$

$$\cos(\theta - 90) = \sin \theta = \frac{z'_2}{D_1} = \frac{z_2 - z_1}{D_1}$$

$$\sin(\theta - 90) = -\cos \theta = -\frac{x'_2}{D_1} = -\frac{x_2 - x_1}{D_1}$$

- Thus, $P_2'' = R_y(\theta - 90) \cdot P_2' = [0 \ y_2 \ y_1 \ D_1 \ 1]^T$

5/3/2007 40

Composition of 3D Transforms

- Next, we do the same thing for the x axis
- Our angle of rotation is ϕ

$$\cos \phi = \frac{z''_2}{D_2} \quad \sin \phi = \frac{y''_2}{D_2}$$

- Thus, $P_2''' = R_x(\phi) \cdot R_y(\theta) \cdot T \cdot P_2 = [0 \ 0 \ |P_1P_2| \ 1]^T$
- Rotation around the z axis is handled similarly

5/3/2007 41

Transforming Coordinate Systems

- Alternate view: when we apply a transformation, we're not changing the coordinates of a point within a coordinate system
- Instead, we're changing the coordinate system itself

FIGURE 5.31 Transforming a coordinate frame.

5/3/2007 42

Department of Computer Science **Transforming Coordinate Systems**

- We can apply multiple transformations:

FIGURE 5.32 Transforming a coordinate system twice.

5/3/2007 43

Department of Computer Science **Transforming Coordinate Systems**

FIGURE 5.33 Elementary changes between coordinate systems.

5/3/2007 44

Department of Computer Science **Transformations in OpenGL**

- OpenGL maintains a number of matrices:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} Modelview \\ matrix \end{bmatrix} \rightarrow \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} \rightarrow \begin{bmatrix} Projection \\ matrix \end{bmatrix} \rightarrow \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix}$$

$$Perspective\ division \rightarrow \begin{bmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{bmatrix} \rightarrow \dots \rightarrow \begin{bmatrix} Viewport \\ transformation \end{bmatrix} \rightarrow Window\ coordinates$$

5/3/2007 45

Department of Computer Science **Transformations in OpenGL**

- We switch between them with the `glMatrixMode()` routine
- The most recent call to `glMatrixMode()` selects OpenGL's *current transformation* (CT) matrix
- Transformations are applied by postmultiplying CT by the desired transformation:
 $CT = CT \cdot M$
- More on how this works in the next set of notes

5/3/2007 46