COEN 290 Computer Graphics I

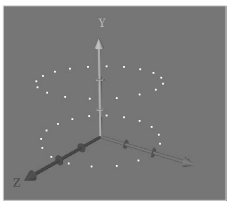
Santa Clara University

Evening's Goals

- Discuss viewing and modeling transformations
- Describe matrix stacks and their uses
- Show basic geometric rasterization and clipping algorithms

Modeling Objects

- Recall objects are composed of geometric primitives
  - each primitive is composed of vertices



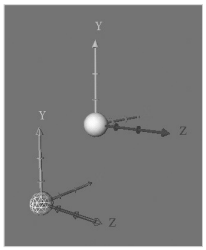
- *Model* around the origin
  - *model* just means "determine an object's vertices"

Modeling Transformations

- Position objects in world coordinates
- Move coordinate systems, not objects
- Affects all objects rendered after transformation
- Types
  - translation
  - scale
  - rotation
  - shear

Translation

- Move the origin to a new location



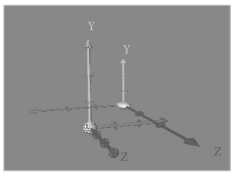
glTranslate[fd]()

```
glTranslatef( tx, ty, tz );
```

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scale

- Stretch, mirror or decimate a coordinate direction



$s > 1$	stretch
$0 > s \geq 1$	shrink
0	decimate
$-1 \leq s < 0$	reflect/shrink
$-1 > s$	reflect/strecth

Note, there's a translation applied here to make things easier to see

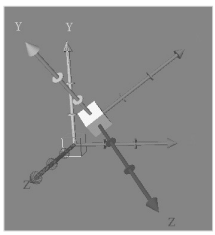
glScale[fd]()

glScalef(s_x, s_y, s_z);

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

- Rotate coordinate system about an axis in space



Note, there's a translation applied here to make things easier to see

glRotate[fd]()

glRotatef(*angle, x, y, z*);

$$\vec{v} = (x \;\; y \;\; z)$$
$$\vec{u} = \frac{\vec{v}}{\|\vec{v}\|} = (x' \;\; y' \;\; z')$$

$$M = \vec{u}^t\vec{u} + \cos(\theta)(I - \vec{u}^t\vec{u}) + \sin(\theta)S$$

$$S = \begin{pmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{pmatrix} \qquad R_r(\theta) = \begin{pmatrix} & & & 0 \\ & M & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Some Additional Rotation Examples

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

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

Shear

- Scaling in one dimension depends on another
- For example

$$x' = x + f_{xy}y + f_{xz}z$$

- No OpenGL command
  - load custom matrix

Shear

- Making a shear matrix

$$H = \begin{pmatrix} 1 & f_{xy} & f_{xz} & 0 \\ f_{yx} & 1 & f_{yz} & 0 \\ f_{zx} & f_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Other OpenGL Matrix Commands

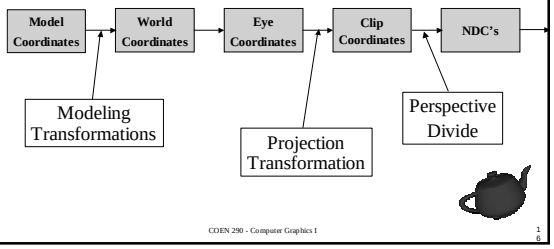
- **`glMultMatrix`**`( m )`
  - multiples the current matrix by *m*
- **`glLoadMatrix`**`( m )`
  - replaces the current matrix by *m*
- **`glLoadIdentity`**`()`
  - replace the current matrix with an identity matrix

OpenGL Matrix Format

```
GLfloat  m[16];
```

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

Modeling Transforms and our Pipeline



Using Multiple Modeling Transforms

- Multiple modeling transforms form a *composite* modeling transform
- Individual transform matrices are multiplied together
  - for example

$$M = T(t_x, t_y, t_z) \cdot R_{\vec{v}}(\theta) \cdot S(s_x, s_y, s_z)$$

Multiplying Matrices

- Matrix multiplication is not *commutative*
  - in general, $AB \neq BA$
  - order of operations is important
- OpenGL multiples matrices on the ***right***

$$A \cdot B \cdot C \cdot D = (((A \cdot B) \cdot C) \cdot D)$$

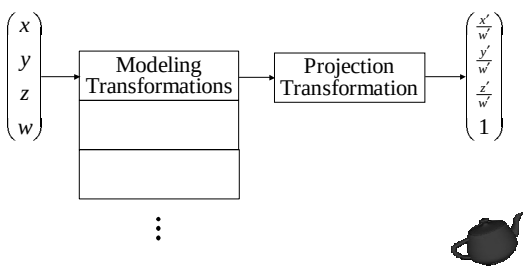
Independent vs. Dependent Transforms

- Every modeling transform affects the model coordinate system
- Transformations accumulate
  - we record every transformation every made
  - to undo a transform, we'd need to do the inverse transform
    - this is very inconvenient

A Stack Based Solution

- We create a stack containing matrices
- Any transform multiples the top of stack
- *Push* makes a copy and pushes it onto the stack
- *Pop* discards the top of stack

A Stack Based Solution (cont.)



OpenGL Matrix Stack Commands

- Top of stack matrix is called the *current matrix*
- **`glPushMatrix()`**
  - copy the current matrix and pushes it
- **`glPopMatrix()`**
  - pop the current matrix

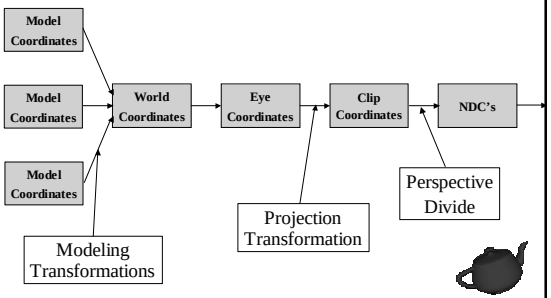
OpenGL Matrix Stacks

- Why multiple matrix stacks?
  - certain techniques are done in different spaces

`glMatrixMode(` *mode* **`)`**

  - choose which stack to manipulate
    - **`GL_MODELVIEW`**
    - **`GL_PROJECTION`**

Current Transformation Pipeline



Eye coordinates are nice, but ...

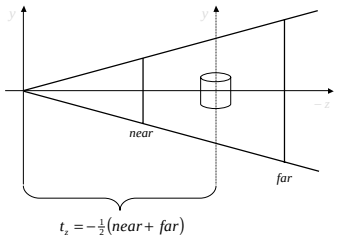
- Eye coordinates are restrictive
  - eye's positioned at the origin
  - looking down the -z axis
- What if we want to look at the scene from somewhere else?
  - use a *viewing transformation*

Viewing Transformations

- Reorient world coordinates to match eye coordinates
- Basically just a modeling transform
  - affects the entire scene
  - usually a translation and a rotation
- Usually set up after the projection transform, but before any modeling transforms

The Simplest Viewing Transform

- "Push" the origin into the viewing frustum



polarview()

- Useful if you want to view an entire object

```
polarview( dist, elev, azim, twist )
{
   glTranslatef( 0, 0, -dist );
   glRotatef( -twist, 0, 0, 1 );
   glRotatef( -elev, 1, 0, 0 );
   glRotatef( azim, 0, 0, 1 );
}
```

  - update *elev* and *azim* for interactive viewing

Transforming World to Eye Coordinates

- Viewing transform

```
gluLookAt( eyex, eyey, eyez,
           lookx, looky, lookz,
           upx, upy, upz );
```

- Creates an *orthonormal basis*
  - a set of linearly independent vectors of unit length

Creating an Orthonormal Basis

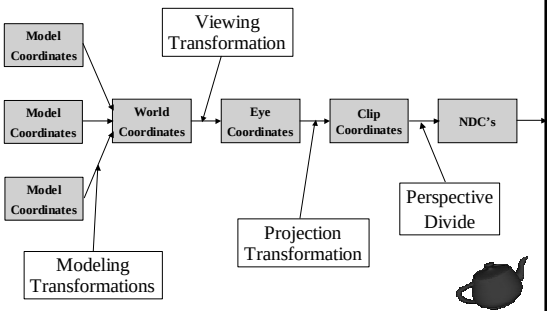
$$\hat{n} = \frac{\overrightarrow{look} - \overrightarrow{eye}}{\left\| \overrightarrow{look} - \overrightarrow{eye} \right\|}$$

$$\hat{u} = \frac{\hat{n} \times \overrightarrow{up}}{\left\| \hat{n} \times \overrightarrow{up} \right\|} \Rightarrow \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ -n_x & -n_y & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\hat{v} = \hat{u} \times \hat{n}$$

```
gluLookAt( eye_x, eye_y, eye_z,
           look_x, look_y, look_z,
           up_x, up_y, up_z )
 {
   /* Create matrix m */
   glMultMatrixf( m );
   glTranslatef( -eye_x, -eye_y, -eye_z );
 }
```

Our Final Transformation Pipeline



Retrieving Matrix information

```
GLfloat  m[16];
glGetFloatv( which, m );
```

- *which* is which matrix stack to access
  - `GL_MODELVIEW_MATRIX`
  - `GL_PROJECTION_MATRIX`

Putting it all Together

- General flow of transformations
  - ① projection transformation
  - ② viewing transformation
  - ③ push matrix
  - ④ modeling transformation
  - ⑤ render
  - ⑥ pop matrix
  - ⑥ goto ③ as necessary