Transformations II

Ronald A. Metoyer
Oregon State University
Goals

Storing and updating transforms for objects
Camera/Viewing Transformation
In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size. We apply an instance transformation to its vertices to

- Scale
- Orient
- Locate
Instance Transform

Model objects in object space
Apply S,R,T
  - Translate
  - Rotate
  - Scale

P' = TRSP

How do we update the instance transform when they objects can be changed interactively (as in Assignment#3)?
Updating Instance Transform

A user interaction is simply a new transform to be composed with the existing transformations

\[
S' = S_{\text{update}} * S;
\]
\[
T' = T_{\text{update}} * T;
\]
\[
R' = R_{\text{update}} * R;
\]

Examples: Matlab Script
OpenGL Camera Revisited

We have been specifying the camera using LookAt

Sometimes more convenient to take an alternative view of the camera specification

Remember, by default, Camera is at origin looking down –z axis
If we want to visualize objects with both positive and negative z values (i.e. Behind the default camera) we can either

- Move the camera in the positive z direction
- or
- Move the objects in the negative z direction

Both of these views are equivalent and are determined by the model-view matrix.
Camera Example

We can move the camera to any desired position by a sequence of rotations and translations.

Must reverse the order and negate parameters (i.e. invert the transform for objects)

Example: side view

Translate Camera Away

Rotate about Y

\[ M_{\text{cam}} = RT \]

Model-view matrix for scene:

\[ M_{\text{Scene}} = T^{-1}R^{-1} \]
Combining Modeling + Camera

Model transforms position objects

View transforms position camera (but really repositioning objects with respect to the camera)

To build the final ModelView Matrix: combine instance transform with viewing transform

\[ P' = T_{\text{cam}}^{-1} R_{\text{cam}}^{-1} T R S P \]

If using the scene rotation model (instead of camera)

\[ P' = T_{\text{scene}} R_{\text{scene}} T R S P \]
Selection

If the user clicks, the mouse callback should...

1. Render Pass 1 – with a unique color for each object
2. Read Pixels at mouse location
3. Match color to object with that unique color
Selection

Assume you have assigned a unique color to each selectable object (i.e. unsigned integer value counter)

Clear buffers
For each object
1. Use a uniform variable to sync object’s unique color to the shader
2. Render (e.g. call glDrawArrays)
3. glutPostRedisplay – to make sure it forces a render
4. Read pixel at x,y location
5. Compare pixel color to unique object colors
6. If match is found, that’s the object that is being clicked!
Useful Functions

```c
void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,
                   GLenum format, GLenum type, GLvoid * data);
```

e.g.  glReadPixels(x, viewport[3] - y, 1, 1, GL_RGBA,
                      GL_UNSIGNED_BYTE, pixel);

```c
void glGetIntegerv(GLenum pname, GLint * data);
```

e.g.  GLint viewport[4];
       glGetIntegerv(GL_VIEWPORT, viewport);
```