Transformations II

Ronald A. Metoyer
Oregon State University
How do we implement and apply transformations in OpenGL?

Must create your own transformation matrices

Angel.h

glm.h

Must compute composite transformations

modelview_matrix

modeling transforms

viewing transforms
How do we implement and apply transformations in OpenGL?

In the old fixed function pipeline, the modeling and viewing transformations were composed to create the **model_view** matrix

- **view**: from the LookAt function
- **model**: from modeling transformations
TransformOrderDemo
Storing Transformations

In modeling, we typically start with a simple object centered at the origin, oriented with the axis, and at some initial size. We apply an \textit{instance transformation} to its vertices to:

- **Scale**
- **Orient**
- **Locate**

The transformation matrix \( M = \text{TRS} \) combines these transformations, where \( S \) represents scaling, \( R \) represents rotation, and \( T \) represents translation.
Instance Transform

Model objects in object space
Apply S,R,T
  Translate
  Rotate
  Scale
  \( P' = TRSP \)

How do we update the instance transform when the objects can be changed interactively (as in Assignment#4)?
Updating Instance Transform

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

\[
S' = S_{\text{update}} \times S_{\text{current}};
\]

\[
T' = T_{\text{update}} \times T_{\text{current}};
\]

\[
R' = R_{\text{update}} \times R_{\text{current}};
\]
In Practice

For each ‘manipulable’ object

Store Rcurrent, Scurrent, Tcurrent

For any user interaction

Determine the mode (T,R,S)
Update T,R, or S with dT, dR, or dS
Apply update to ‘current’

In Display

Apply Instance Transform with ‘current’
P’ = TRSP
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
Moving the Camera Frame

If we want to visualize objects with both positive and negative z values (ie. 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

In reality, there is not actual ‘camera’ so we take the second option!
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)

Translate Camera Away
Rotate about Y

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

Model-view matrix for scene:

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

Model transformations position, scale, and orient objects

View transformation positions and orients the camera (but really repositioning objects with respect to the camera)

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

\[ P' = T^{-1}_{\text{cam}} R^{-1}_{\text{cam}} T R S P \]
Next Week…

How does LookAt produce the correct View Transformation?