Reading for This and Next 2 Lectures

- FCG Chapter 7 Viewing
- FCG Section 6.3.1 Windowing Transforms
- RB rest of Chap Viewing
- RB rest of App Homogeneous Coords
Review: Display Lists

• precompile/cache block of OpenGL code for reuse
  • usually more efficient than immediate mode
    • exact optimizations depend on driver
  • good for multiple instances of same object
    • but cannot change contents, not parametrizable
  • good for static objects redrawn often
    • display lists persist across multiple frames
    • interactive graphics: objects redrawn every frame from new viewpoint from moving camera
• can be nested hierarchically
• snowman example: 3x performance improvement, 36K polys
Review: Computing Normals

- normal
  - direction specifying orientation of polygon
    - \( w=0 \) means direction with homogeneous coords
    - vs. \( w=1 \) for points/vectors of object vertices
  - used for lighting
    - must be normalized to unit length
  - can compute if not supplied with object

\[ N = (P_2 - P_1) \times (P_3 - P_1) \]
Review: Transforming Normals

- cannot transform normals using same matrix as points
  - nonuniform scaling would cause to be not perpendicular to desired plane!

\[ P \Rightarrow P' = MP \]
\[ N \Rightarrow N' = QN \]

given \( M \), what should \( Q \) be?

\[ Q = (M^{-1})^T \]

inverse transpose of the modelling transformation
Viewing
Using Transformations

- three ways
  - modelling transforms
    - place objects within scene (shared world)
    - affine transformations
  - viewing transforms
    - place camera
    - rigid body transformations: rotate, translate
  - projection transforms
    - change type of camera
    - projective transformation
Rendering Pipeline

Scene graph
Object geometry

Modelling
Transforms

Viewing
Transform

Projection
Transform
Rendering Pipeline

- result
  - all vertices of scene in shared 3D world coordinate system
Rendering Pipeline

- result
  - scene vertices in 3D view (camera) coordinate system
Rendering Pipeline

• result
  • 2D screen coordinates of clipped vertices
Viewing and Projection

• need to get from 3D world to 2D image
• projection: geometric abstraction
  • what eyes or cameras do
• two pieces
  • viewing transform:
    • where is the camera, what is it pointing at?
  • perspective transform: 3D to 2D
    • flatten to image
Rendering Pipeline
OpenGL Transformation Storage

• modeling and viewing stored together
  • possible because no intervening operations
• perspective stored in separate matrix

• specify which matrix is target of operations
  • common practice: return to default modelview mode after doing projection operations

```c
    glMatrixMode(GL_MODELVIEW);
    glMatrixMode(GL_PROJECTION);
```
Coordinate Systems

• result of a transformation
• names
  • convenience
    • mouse: leg, head, tail
  • standard conventions in graphics pipeline
    • object/modelling
    • world
    • camera/viewing/eye
    • screen/window
    • raster/device
Projective Rendering Pipeline

OCS - object/model coordinate system
WCS - world coordinate system
VCS - viewing/camera/eye coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device/display/screen coordinate system

- Modeling transformation: O2W
- Viewing transformation: W2V
- Projection transformation: V2C
- Perspective divide: C2N
- Viewport transformation: N2D
- Clipping: CCS
- Normalized device: NDCS
- Device: DCS
Viewing Transformation

- **VCS**
- **OCS**
- **WCS**

**Image Plane**

**Object Space (OCS)**

**World Space (WCS)**

**Viewing Space (VCS)**

**Modeling Transformation**

**Viewing Transformation**

$$M_{\text{mod}}$$

$$M_{\text{cam}}$$

OpenGL ModelView matrix
Basic Viewing

• starting spot - OpenGL
  • camera at world origin
    • probably inside an object
  • y axis is up
  • looking down negative z axis
    • why? RHS with x horizontal, y vertical, z out of screen
• translate backward so scene is visible
  • move distance d = focal length

• where is camera in P1 template code?
  • 5 units back, looking down -z axis
Convenient Camera Motion

- rotate/translateSCALE versus
  - eye point, gaze/lookat direction, up vector

- demo: Robins transformation, projection
OpenGL Viewing Transformation

gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)

- postmultiplies current matrix, so to be safe:

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)
  // now ok to do model transformations

- demo: Nate Robins tutorial  *projection*
Convenient Camera Motion

- rotate/translate/scale versus
  - eye point, gaze/lookat direction, up vector
From World to View Coordinates: W2V

- translate **eye** to origin
- rotate **view** vector (**lookat** – **eye**) to **w** axis
- rotate around **w** to bring **up** into **vw**-plane
Deriving W2V Transformation

- translate `eye` to origin

\[
T = \begin{bmatrix}
1 & 0 & 0 & e_x \\
0 & 1 & 0 & e_y \\
0 & 0 & 1 & e_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Deriving W2V Transformation

- rotate **view** vector (**lookat** – **eye**) to **w** axis
  - **w**: normalized opposite of **view/gaze** vector **g**

\[
\mathbf{w} = -\hat{\mathbf{g}} = - \frac{\mathbf{g}}{\|\mathbf{g}\|}
\]
Deriving W2V Transformation

- rotate around \( w \) to bring \textbf{up} into \( vw \)-plane
  - \( u \) should be perpendicular to \( vw \)-plane, thus perpendicular to \( w \) and \textbf{up} vector \( t \)
  - \( v \) should be perpendicular to \( u \) and \( w \)

\[
\begin{align*}
  u &= \frac{t \times w}{\|t \times w\|} \\
  v &= w \times u
\end{align*}
\]
Deriving W2V Transformation

- rotate from WCS \(xyz\) into \(uvw\) coordinate system with matrix that has columns \(u, v, w\)

\[
\begin{align*}
    u &= \frac{t \times w}{\|t \times w\|} \\
    v &= w \times u \\
    w &= -\hat{g} = - \frac{g}{\|g\|}
\end{align*}
\]

\[
R = \begin{bmatrix}
    u_x & v_x & w_x & 0 \\
    u_y & v_y & w_y & 0 \\
    u_z & v_z & w_z & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
T = \begin{bmatrix}
    1 & 0 & 0 & e_x \\
    0 & 1 & 0 & e_y \\
    0 & 0 & 1 & e_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
M_{W2V} = TR
\]

- reminder: rotate from \(uvw\) to \(xyz\) coord sys with matrix \(M\) that has columns \(u,v,w\)
W2V vs. V2W

- \( M_{W2V} = TR \)
- we derived position of camera in world
  - invert for world with respect to camera
- \( M_{V2W} = (M_{W2V})^{-1} = R^{-1}T^{-1} \)
- inverse is transpose for orthonormal matrices
- inverse is negative for translations
W2V vs. V2W

- $M_{W2V} = TR$
- $T = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- $R = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- we derived position of camera in world
  - invert for world with respect to camera
- $M_{V2W} = (M_{W2V})^{-1} = R^{-1}T^{-1}$

$M_{view2world} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -e_x \\ v_x & v_y & v_z & -e_y \\ w_x & w_y & w_z & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Moving the Camera or the World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but world moves to -7
- resulting image same either way
  - possible difference: are lights specified in world or view coordinates?
World vs. Camera Coordinates Example

\[ \mathbf{a} = (1,1)_{\mathbf{w}} \]
\[ \mathbf{b} = (1,1)_{\mathbf{c}_1} = (5,3)_{\mathbf{w}} \]
\[ \mathbf{c} = (1,1)_{\mathbf{c}_2} = (1,3)_{\mathbf{c}_1} = (5,5)_{\mathbf{w}} \]