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: Normals

- polygon:
  \[ N = (P_2 - P_1) \times (P_3 - P_1) \]

- assume vertices ordered CCW when viewed from visible side of polygon

- normal for a vertex
  - specify polygon orientation
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons
Review: Transforming Normals

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

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

given M, what should Q be?

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

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

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping → Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
Rendering Pipeline

Geometry Database → Mode/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
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
    glMatrixMode(GL_MODELVIEW);
    glMatrixMode(GL_PROJECTION);
Coordinate Systems

• result of a transformation
• names
  • convenience
    • armadillo: 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
W2V
viewing transformation
V2C
projection transformation
C2N
perspective divide
N2D
viewport transformation

O2W
world
object
Viewing Transformation

object: OCS
world: WCS
viewing: VCS

modeling transformation: $M_{mod}$
viewing transformation: $M_{cam}$

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 $d = \text{focal length}$
• can use rotate/translate/scale to move camera
  • demo: Nate Robins tutorial transformations
Viewing in Project 1

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

- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector
Convenient Camera Motion

- rotate/translateSCALE not intuitive
- arbitrary viewing position
  - 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
OpenGL Viewing Transformation

\[ \text{gluLookAt}(\text{ex}, \text{ey}, \text{ez}, \text{lx}, \text{ly}, \text{lz}, \text{ux}, \text{uy}, \text{uz}) \]

- postmultiplies current matrix, so to be safe:

\[
\text{glMatrixMode}(\text{GL}_\text{MODELVIEW}) ; \\
\text{glLoadIdentity}() ; \\
\text{gluLookAt}(\text{ex}, \text{ey}, \text{ez}, \text{lx}, \text{ly}, \text{lz}, \text{ux}, \text{uy}, \text{uz}) \\
// \text{now ok to do model transformations}
\]

- demo: Nate Robins tutorial \textit{projection}
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 \( g \)

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

• rotate around \( w \) to bring \( \text{up} \) into \( vw \)-plane
  • \( u \) should be perpendicular to \( vw \)-plane, thus perpendicular to \( w \) and \( \text{up} \) 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 \textbf{xyz} into \textbf{uvw} coordinate system with matrix that has rows \textbf{u}, \textbf{v}, \textbf{w}

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

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

• reminder: rotate from \textbf{uvw} to \textbf{xyz} coord sys with matrix \textbf{M} that has columns \textbf{u,v,w}
  • rotate from \textbf{xyz} coord sys to \textbf{uvw} coord sys with matrix \textbf{M}^T that has rows \textbf{u,v,w}
Deriving W2V Transformation

- $M = RT$

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

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

\[
M_{\text{world} \rightarrow \text{view}} = \begin{bmatrix}
    u_x & u_y & u_z & 0 \\
    v_x & v_y & v_z & 0 \\
    w_x & w_y & w_z & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & 0 & -e_x \\
    0 & 1 & 0 & -e_y \\
    0 & 0 & 1 & -e_z \\
    0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
    u_x & u_y & u_z & -u \cdot e \\
    v_x & v_y & v_z & -v \cdot e \\
    w_x & w_y & w_z & -w \cdot e \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]
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

\[ a = (1,1)_W \]

\[ b = (1,1)_{C1} = (5,3)_W \]

\[ c = (1,1)_{C2} = (1,3)_{C1} = (5,5)_W \]