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
- rotate view vector (lookat – eye) to w axis
  - w: normalized opposite of view/gaze vector $g$
  - $w = -\frac{g}{\|g\|}$

Convenient Camera Motion
- rotate/translate/scale versus
  - eye point, gaze/lookat direction, up vector
- demo: Robins transformation, projection

OpenGL Viewing Transformation
- $\text{gluLookAt}(ex, ey, ez, lx, ly, lz, ux, uy, uz)$
  - postmultiplies current matrix, so to be safe:
  - $\text{glMatrixMode}(\text{GL_MODELVIEW});$
  - $\text{glLoadIdentity}();$
  - $\text{gluLookAt}(ex, ey, ez, lx, ly, lz, ux, uy, uz)$
    // now ok to do model transformations
  - demo: Nate Robins tutorial projection

Review: Computing Normals
- normal
  - direction specifying orientation of polygon
  - w0 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

Deriving W2V Transformation
- rotate around w to bring up into vw-plane
  - u should be perpendicular to vw-plane, thus perpendicular to w and up vector t
  - v should be perpendicular to u and w
  - $u \cdot I \times w$
  - $v = w \times u$
Pinhole Camera

• non-zero sized hole
• blur: rays hit multiple points on film plane

Real Cameras

• pinhole camera has small aperture (lens opening)
• maximum blur
• problem: hard to get enough light to expose the film
• solution: lens
• permits larger apertures
• permits changing distance to film plane without actually moving lens
• cost: limited depth of field where image is in focus

Graphics Cameras

• real pinhole camera: image inverted
• computer graphics camera: convenient equivalent

General Projection

• image plane need not be perpendicular to view plane

Perspective Projection

• our camera must model perspective
**Perspective Projection**
- project all geometry
- through common center of projection (eye point)
- onto an image plane

**Basic Perspective Projection**
- desired result for a point $[x, y, z, 1]^T$ projected onto the view plane:
  \[
  \frac{x'}{d} = \frac{x}{z}, \quad \frac{y'}{d} = \frac{y}{z} \Rightarrow \frac{x'}{d} = \frac{x - d}{z}
  \]
  \[
  \text{but} \quad z' = d
  \]
- nonuniform foreshortening
- not affine

**Perspective Projection**
- expressible with 4x4 homogeneous matrix
- use previously untouched bottom row
- perspective projection is irreversible
- many 3D points can be mapped to same $(x, y, d)$ on the projection plane
- no way to retrieve the unique $z$ values

**Simple Perspective Projection Matrix**
\[
\begin{bmatrix}
  x \\ z/d \\ y \\ z/d \\ d
\end{bmatrix}
\]
is homogenized version of where $w = z/d$

**Perspective to Orthographic**
- transformation of space
- center of projection moves to infinity
- view volume transformed
- from frustum (truncated pyramid) to parallelepiped (box)

**Orthographic Camera Projection**
- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away $z$ values

**Canonical View Volumes**
- standardized viewing volume representation
- permits standardization
- clipping
  - easier to determine if an arbitrary point is enclosed in volume with canonical view volume vs. clipping to six arbitrary planes
  - rendering
    - projection and rasterization algorithms can be reused

**Why Canonical View Volumes?**
- convenient
  - viewing frustum mapped to specific parallelepiped
  - Normalized Device Coordinates (NDC)
    - same as clipping coords
    - only objects inside the parallelepiped get rendered
    - which parallelepiped?
      - depends on rendering system

**Normalized Device Coordinates**
- left/right $x = \pm 1$, top/bottom $y = \pm 1$, near/far $z = \pm 1$
Understanding Z
- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)

$VCS \rightarrow NDCS$

$$(-1,-1,-1)(1,1,1)$$

Orthographic Derivation
- scale, translate, reflect for new coord sys

$$y' = a \cdot y + b$$
$$y' = a \cdot y + b$$
$$y = top \rightarrow y' = 1$$
$$y = bottom \rightarrow y' = -1$$
$$a = \frac{2}{top - bot}$$
$$b = \frac{-2}{top - bot}$$

$$P' = \left[ \begin{array}{ccc} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -2 \end{array} \right]$$

Orthographic OpenGL
- glMatrixMode(GL_PROJECTION);
- glLoadIdentity();
- glOrtho(left, right, bot, top, near, far);