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

Convenient Camera Motion

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

Placing Camera in World Coords: V2W

- treat camera as if it's just an object
  - translate from origin to eye
  - rotate view vector (lookat - eye) to w axis
  - rotate around w to bring up into vw-plane

Coordinate Systems

- result of a transformation
- names
  - convenience
    - animal: leg, head, tail
  - standard conventions in graphics pipeline
    - object/modelling
    - world
    - camera/viewing/eye
    - screen/window
    - raster/device

Viewing and Projection

- need to get from 3D world to 2D image
  - projection: geometric abstraction
  - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
  - flatten to image

Projective Rendering Pipeline

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

Viewing Transformation

- starting spot - GL
  - 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

Rendering Pipeline

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

Basic Viewing

- 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
Deriving V2W Transformation

- rotate view vector (lookat – eye) to w axis
- w: normalized opposite of view/gaze vector g
  \[ w = \vec{g} \]
- 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

\[
\begin{bmatrix}
ux & vy & wz & 0 \\
uy & vy & wz & 0 \\
uw & vz & wz & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Moving the Camera or the World?

- two equivalent operations
- move camera one way vs. move world other way
- example
  - initial GL 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 GL models viewing in left-hand coordinates
  - camera stays put, but world moves -7
  - resulting image same either way
  - possible differences: are lights specified in world or view coordinates?

World vs. Camera Coordinates Example

\[
\begin{bmatrix}
a = (1,1)_w \\
b = (1,1)_{c1} = (5,3)_w \\
c = (1,1)_{c2} = (1,3)_{c1} = (5,5)_w \\
a = (1,1)_{w1} \\
b = (1,1)_{c1} = (5,3)_{w1} \\
c = (1,1)_{c2} = (1,3)_{c1} = (5,5)_{w1}
\end{bmatrix}
\]