University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2016

Tamara Munzner

Viewing 1

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
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
**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

The pipeline consists of:

1. **Modeling Transformation** (O2W)
2. **Viewing Transformation** (W2V)
3. **Projection Transformation** (V2C)
4. **Clipping Transformation** (C2N)
5. **Viewport Transformation** (N2D)

**Transformation Details:**
- `O2W`: object to world coordinate transformation
- `W2V`: world to viewing coordinate transformation
- `V2C`: viewing to clipping coordinate transformation
- `C2N`: clipping to normalized device coordinate transformation
- `N2D`: normalized device to device coordinate transformation

**Coordinate Systems:**
- **OCS**: object or model coordinate system
- **WCS**: world coordinate system
- **VCS**: viewing, camera, or eye coordinate system
- **CCS**: clipping coordinate system
- **NDCS**: normalized device coordinate system
- **DCS**: device, display, or screen coordinate system
Viewing Transformation

OCS

WCS

VCS

modeling transformation

transformation

M_{mod}

M_{cam}

modelview matrix
Basic Viewing

• 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
Convenient Camera Motion

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

- `lookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)`
Convenient Camera Motion

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

![Diagram of camera motion with WCS, eye, lookat, view arrows]
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
Deriving V2W Transformation

- translate origin to eye

\[
T = \begin{bmatrix}
1 & 0 & 0 & e_x \\
0 & 1 & 0 & e_y \\
0 & 0 & 1 & e_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Deriving V2W Transformation

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

\[ w = -\hat{g} = -\frac{g}{\|g\|} \]
Deriving V2W 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 $w$

$$u = \frac{t \times w}{||t \times w||} \quad v = w \times u$$
Deriving V2W Transformation

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

\[
\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\left| \mathbf{t} \times \mathbf{w} \right|}
\]

\[
\mathbf{v} = \mathbf{w} \times \mathbf{u}
\]

\[
\mathbf{w} = -\mathbf{\hat{g}} = -\frac{\mathbf{g}}{\left| \mathbf{g} \right|}
\]

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

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

\[
\mathbf{M}_{V2W} = \mathbf{TR}
\]

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

- $M_{V2W} = TR$

- we derived position of camera as object in world
  - invert for lookAt: go from world to camera!

- $M_{W2V} = (M_{V2W})^{-1} = R^{-1}T^{-1}$

- inverse is transpose for orthonormal matrices
- inverse is negative for translations
V2W vs. W2V

- $M_{W2V} = (M_{V2W})^{-1} = R^{-1}T^{-1}$

$$M_{world2view} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -e \cdot u \\ v_x & v_y & v_z & -e \cdot v \\ w_x & w_y & w_z & -e \cdot w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{W2V} = \begin{bmatrix} u_x & u_y & u_z & -e_x \cdot u_x + -e_y \cdot u_y + -e_z \cdot u_z \\ v_x & v_y & v_z & -e_x \cdot v_x + -e_y \cdot v_y + -e_z \cdot v_z \\ w_x & w_y & w_z & -e_x \cdot w_x + -e_y \cdot w_y + -e_z \cdot w_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Moving the Camera or the World?

• two equivalent operations
  • move camera one way vs. move world other way

• example
  • initial 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 to -7

• resulting image same either way
  • possible difference: are lights specified in world or view coordinates?
World vs. Camera Coordinates Example

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

\[ b = (1,1)_{c_1} = (5,3)_w \]

\[ c = (1,1)_{c_2} = (1,3)_{c_1} = (5,5)_w \]