Foundations of Computer Graphics (Fall 2012)

CS 184, Lecture 4: Transformations 2

http://inst.eecs.berkeley.edu/~cs184

To Do

- Start doing HW 1
  - Time is short, but needs only little code [Due Wed Sep 12]
  - Ask questions or clear misunderstandings by next lecture
- Specifics of HW 1
  - Last lecture covered basic material on transformations in 2D Likely need this lecture to understand full 3D transformations
  - Last lecture had full derivation of 3D rotations. You only need final formula
  - gluLookAt derivation this lecture helps clarifying some ideas
- Read and post on Piazza re questions

Outline

- *Translation: Homogeneous Coordinates*
- Combining Transforms: Scene Graphs
- Transforming Normals
- Rotations revisited: coordinate frames
- gluLookAt (quickly)

Translation

- E.g. move x by +5 units, leave y, z unchanged
- We need appropriate matrix. What is it?

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} & & \\ & ? & \\ & & \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x+5 \\ y \\ z \end{pmatrix}$$

Transformations game demo

Homogeneous Coordinates

- Add a fourth homogeneous coordinate (w=1)
- 4x4 matrices very common in graphics, hardware
- Last row always 0 0 0 1 (until next lecture)

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+5 \\ y \\ z \\ 1 \end{pmatrix}$$

Representation of Points (4-Vectors)

Homogeneous coordinates

- Divide by 4th coord (w) to get (inhomogeneous) point $P = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix}$
- Multiplication by w > 0, no effect
- Assume w ≥ 0. For w > 0, normal finite point. For w = 0, point at infinity (used for vectors to stop translation)

Advantages of Homogeneous Coords

- Unified framework for translation, viewing, rot…
- Can concatenate any set of transforms to 4x4 matrix
- No division (as for perspective viewing) till end
- Simpler formulas, no special cases
- Standard in graphics software, hardware

General Translation Matrix

$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} I_3 & T \\ 0 & 1 \end{pmatrix}$$

$$P' = TP = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+T_x \\ y+T_y \\ z+T_z \\ 1 \end{pmatrix} = P + T$$

Combining Translations, Rotations

- Order matters!! TR is not the same as RT (demo)
- General form for rigid body transforms
- We show rotation first, then translation (commonly used to position objects) on next slide. Slide after that works it out the other way
- Demos with applet, homework 1

Combining Translations, Rotations

$$P' = (TR)P = MP = RP + T$$

$$M = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \left(\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix}$$

Transformations game demo

Combining Translations, Rotations

$$P' = (RT)P = MP = R(P+T) = RP + RT$$

$$M = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{3\times3} & R_{3\times3}T_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix}$$

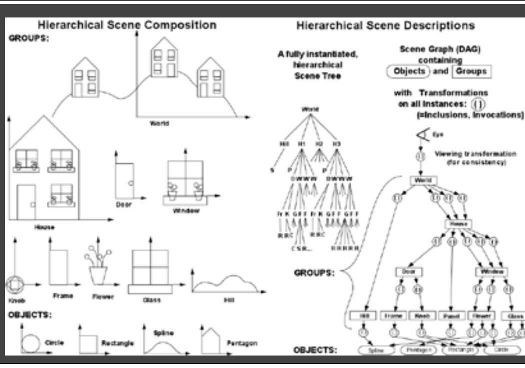
Transformations game demo

Outline

- Translation: Homogeneous Coordinates
- *Combining Transforms: Scene Graphs*
- Transforming Normals
- Rotations revisited: coordinate frames
- gluLookAt (quickly)

Slides for this part courtesy Prof. O' Brien

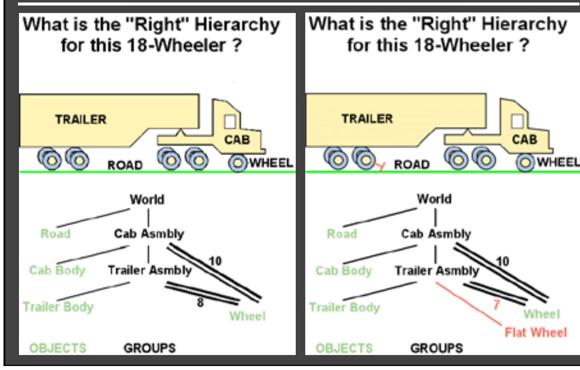
Hierarchical Scene Graph



Drawing a Scene Graph

- Draw scene with pre-and-post-order traversal
  - Apply node, draw children, undo node if applicable
- Nodes can carry out any function
  - Geometry, transforms, groups, color, …
- Requires stack to “undo” post children
  - Transform stacks in OpenGL
- Caching and instancing possible
- Instances make it a DAG, not strictly a tree

Example Scene-Graphs

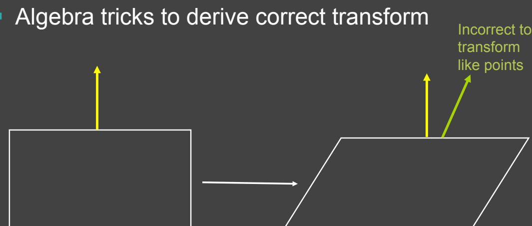


Outline

- Translation: Homogeneous Coordinates
- Combining Transforms: Scene Graphs
- *Transforming Normals*
- Rotations revisited: coordinate frames
- gluLookAt (quickly)

Normals

- Important for many tasks in graphics like lighting
- Do not transform like points e.g. shear
- Algebra tricks to derive correct transform



Finding Normal Transformation

$$t \rightarrow Mt \qquad n \rightarrow Qn \qquad Q = ?$$

$$n^T t = 0$$

$$n^T Q^T M t = 0 \quad \Rightarrow \quad Q^T M = I$$

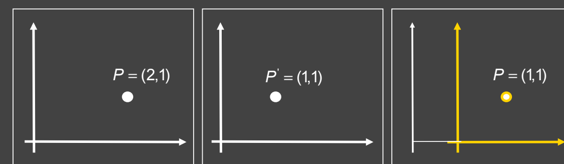
$$Q = (M^{-1})^T$$

Outline

- Translation: Homogeneous Coordinates
- Combining Transforms: Scene Graphs
- Transforming Normals
- *Rotations revisited: coordinate frames*
- gluLookAt (quickly)

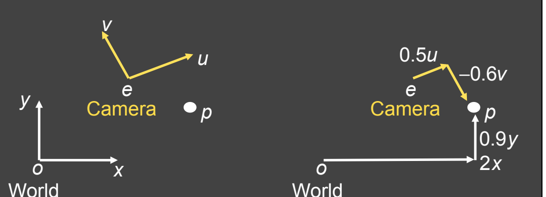
Coordinate Frames

- All of discussion in terms of operating on points
- But can also change coordinate system
- Example, motion means either point moves backward, or coordinate system moves forward

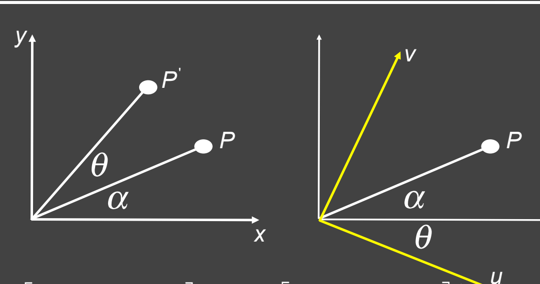


Coordinate Frames: In general

- Can differ both origin and orientation (e.g. 2 people)
- One good example: World, camera coord frames (H1)



Coordinate Frames: Rotations



$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Outline

- Translation: Homogeneous Coordinates
- Combining Transforms: Scene Graphs
- Transforming Normals
- *Rotations revisited: coordinate frames*
- gluLookAt (quickly)

Geometric Interpretation 3D Rotations

- Rows of matrix are 3 unit vectors of new coord frame
- Can construct rotation matrix from 3 orthonormal vectors

$$R_{uvw} = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{pmatrix} \qquad u = x_u X + y_u Y + z_u Z$$

Axis-Angle formula (summary)

$$(b \setminus a)_{ROT} = (I_{3\times3}\cos\theta - aa^T\cos\theta)b + (A^*\sin\theta)b$$

$$(b \rightarrow a)_{ROT} = (aa^T)b$$

$$R(a,\theta) = I_{3\times3}\cos\theta + aa^T(1-\cos\theta) + A^*\sin\theta$$

$$R(a,\theta) = \cos\theta\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} + (1-\cos\theta)\begin{pmatrix}x^2 & xy & xz\\ xy & y^2 & yz\\ xz & yz & z^2\end{pmatrix} + \sin\theta\begin{pmatrix}0 & -z & y\\ z & 0 & -x\\ -y & x & 0\end{pmatrix}$$

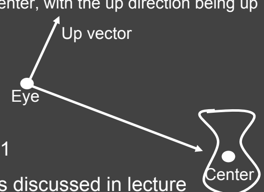
Outline

- Translation: Homogeneous Coordinates
- Combining Transforms: Scene Graphs
- Transforming Normals
- Rotations revisited: coordinate frames
- *gluLookAt (quickly)*

Case Study: Derive gluLookAt

Defines camera, fundamental to how we view images

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera is at eye, looking at center, with the up direction being up



- May be important for HW1
- Combines many concepts discussed in lecture
- Core function in OpenGL for later assignments

Steps

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera is at eye, looking at center, with the up direction being up

- *First, create a coordinate frame for the camera*
- Define a rotation matrix
- Apply appropriate translation for camera (eye) location

Constructing a coordinate frame?

We want to associate **w** with **a**, and **v** with **b**
- But **a** and **b** are neither orthogonal nor unit norm
- And we also need to find **u**

$$w = \frac{a}{\|a\|}$$

$$u = \frac{b \times w}{\|b \times w\|}$$

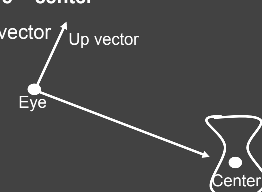
$$v = w \times u$$

from lecture 2

Constructing a coordinate frame

$$w = \frac{a}{\|a\|} \qquad u = \frac{b \times w}{\|b \times w\|} \qquad v = w \times u$$

- We want to position camera at origin, looking down –Z dirn
- Hence, vector **a** is given by **eye** – **center**
- The vector **b** is simply the **up** vector



Steps

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera is at eye, looking at center, with the up direction being up

- First, create a coordinate frame for the camera
- *Define a rotation matrix*
- Apply appropriate translation for camera (eye) location

Geometric Interpretation 3D Rotations

- Rows of matrix are 3 unit vectors of new coord frame
- Can construct rotation matrix from 3 orthonormal vectors

$$R_{uvw} = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{pmatrix} \qquad u = x_u X + y_u Y + z_u Z$$

Steps

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera is at eye, looking at center, with the up direction being up

- First, create a coordinate frame for the camera
- Define a rotation matrix
- *Apply appropriate translation for camera (eye) location*

Translation

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera is at eye, looking at center, with the up direction being up

- *Cannot* apply translation after rotation
- The translation must come first (to bring camera to origin) before the rotation is applied

Combining Translations, Rotations

$$P' = (RT)P = MP = R(P+T) = RP + RT$$

$$M = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{3\times 3} & R_{3\times 3}T_{3\times 1} \\ 0_{1\times 3} & 1 \end{pmatrix}$$

gluLookAt final form

$$\begin{pmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x_u & y_u & z_u & -x_u e_x - y_u e_y - z_u e_z \\ x_v & y_v & z_v & -x_v e_x - y_v e_y - z_v e_z \\ x_w & y_w & z_w & -x_w e_x - y_w e_y - z_w e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$