15-462 Computer Graphics I
Lecture 5

Viewing and Projection

Shear Transformation
Camera Positioning
Simple Parallel Projections
Simple Perspective Projections
[Angel, Ch. 5.2-5.4]
[Red's Dream, Pixar, 1987]

January 31, 2002
Frank Pfenning
Carnegie Mellon University

http://www.cs.cmu.edu/~fp/courses/graphics/

Transformation Matrices in OpenGL

- Transformation matrices in OpenGl are vectors of 16 values (column-major matrices)
- In glLoadMatrixf(GLfloat *m);

  m = {m_1, m_2, ..., m_{16}} represents

$$\mathrm{M} = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

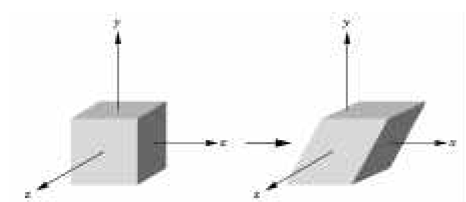
- Some books transpose all matrices!

Pondering Transformations

- Derive transformation given some parameters
  - Choose parameters carefully
  - Consider geometric intuition, basic trigonometry
- Compose transformation from others
  - Use translations to and from origin
- Test if matrix describes some transformation
  - Determine action on basis vectors
- Meaning of dot product and cross product

Shear Transformations

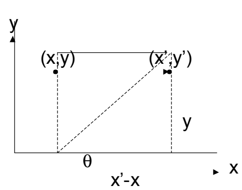
- x-shear scales x proportional to y
- Leaves y and z values fixed



Specification via Angle

- cot(θ) = (x'-x)/y
- x' = x + y cot(θ)
- y' = y
- z' = z

$$\mathrm{H}_x(\theta) = \begin{bmatrix} 1 & \cot(\theta) & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Specification via Ratios

- Shear in both x and z direction
- Leave y fixed
- Slope α for x-shear, γ for z-shear
- Solve

$$\mathrm{H}_{xz}(\alpha,\gamma) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + \alpha y \\ y \\ z + \gamma y \\ 1 \end{bmatrix}$$

- Yields

$$\mathrm{H}_{xz}(\alpha,\gamma) = \begin{bmatrix} 1 & \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \gamma & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composing Transformations

- Every affine transformation is a composition of rotations, scalings, and translations
- How do we compose these to form an x-shear?
- Exercise!

Thinking in Frames

- Action on frame determines affine transfn.
- Frame given by basis vectors and origin
- xz-shear: preserve basis vectors u_x and u_z

$$\mathrm{M}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \qquad \mathrm{M}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

- Move $u_y = [0\ \ 1\ \ 0\ \ 0]^T$ to $u_y' = [\alpha\ \ 1\ \ \gamma\ \ 0]^T$

$$\mathrm{M}\begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}\alpha\\1\\\gamma\\0\end{bmatrix}$$

Preservation of Origin

- Preserve origin P_0

$$\mathrm{M}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

- Results comprise columns of the transfn. matrix

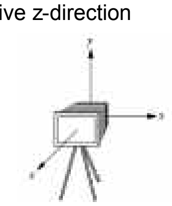
$$\mathbf{H}_{xz}(\alpha, \gamma) = \begin{bmatrix}1 & \alpha & 0 & 0\\0 & 1 & 0 & 0\\0 & \gamma & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

Outline

- Shear Transformation
- Camera Positioning
- Simple Parallel Projections
- Simple Perspective Projections

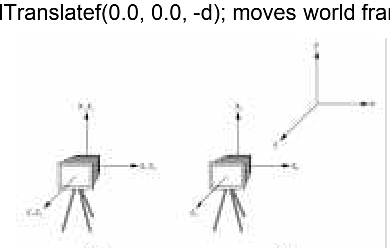
Camera in Modeling Coordinates

- Camera position is identified with a frame
- Either move and rotate the objects
- Or move and rotate the camera
- Initially, pointing in negative z-direction
- Initially, camera at origin



Moving Camera and World Frame

- Move world frame relative to camera frame
- glTranslatef(0.0, 0.0, -d); moves world frame



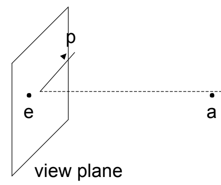
Order of Viewing Transformations

- Think of moving the world frame
- Viewing transfn. is inverse of object transfn.
- Order opposite to object transformations

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);        /*T*/
glRotatef(-90.0, 0.0, 1.0, 0.0); /*R*/
```

The Look-At Function

- Convenient way to position camera
- gluLookAt(e_x, e_y, e_z, a_x, a_y, a_z, p_x, p_y, p_z);
- e = eye point
- a = at point
- p = up vector

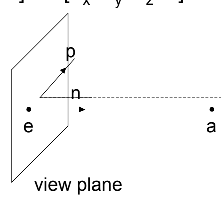


Implementing the Look-At Function

- (1) Transform world frame to camera frame
- Compose a rotation R with translation T
- W = T R
- (2) Invert W to obtain viewing transformation V
- $V = W^{-1} = (T R)^{-1} = R^{-1} T^{-1}$
- Derive R, then T, then $R^{-1} T^{-1}$

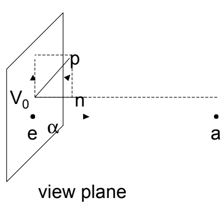
World Frame to Camera Frame I

- Camera points in negative z direction
- $n = (a - e) / |a - e|$ is unit normal to view plane
- R maps $[0\ 0\ -1\ 0]^T$ to $[n_x\ n_y\ n_z\ 0]^T$



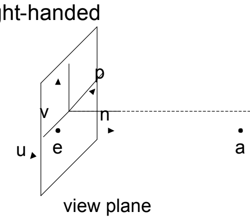
World Frame to Camera Frame II

- R maps y to projection of p onto view plane
- $\alpha = (p \cdot n) / |n| = p \cdot n$
- $v_0 = p - \alpha\, n$
- $v = v_0 / |v_0|$



World Frame to Camera Frame III

- x is orthogonal to n and v in view plane
- $u = n \times v$
- (u, v, -n) is right-handed



Summary of Rotation

- gluLookAt(e_x, e_y, e_z, a_x, a_y, a_z, p_x, p_y, p_z);
- n = (a – e) / |a – e|
- v = (p – (p · n) n) / |p – (p · n) n|
- u = n × v

$$\mathbf{R} = \begin{bmatrix} u_x & v_x & -n_x & 0 \\ u_y & v_y & -n_y & 0 \\ u_z & v_z & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

World Frame to Camera Frame IV

- Translation of origin to e = $[e_x \ e_y \ e_z \ 1]^T$

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

Camera Frame to World Frame

- $V = W^{-1} = (T R)^{-1} = R^{-1} T^{-1}$
- R is rotation, so $R^{-1} = R^T$

$$\mathbf{R}^{-1} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ -n_x & -n_y & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- T is translation, so T^{-1} negates displacement

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Putting it Together

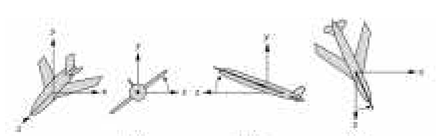
- Calculate $V = R^{-1} T^{-1}$

$$\mathbf{V} = \begin{bmatrix} u_x & u_y & u_z & -u_x e_x - u_y e_y - u_z e_z \\ v_x & v_y & v_z & -v_x e_x - v_y e_y - v_z e_z \\ -n_x & -n_y & -n_z & n_x e_x + n_y e_y + n_z e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This is different from book [Angel, Ch. 5.2.2]
- See errata for 2nd and/or 3rd printing

Other Viewing Functions

- Roll (about z), pitch (about x), yaw (about y)



- Assignment 2 poses related problem

Outline

- Shear Transformation
- Camera Positioning
- Simple Parallel Projections
- Simple Perspective Projections

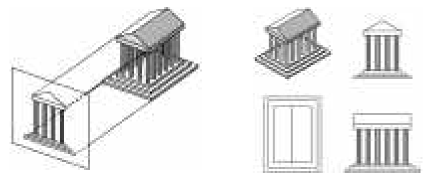
Projection Matrices

- Recall geometric pipeline

- Projection takes 3D to 2D
- Projections are not invertible
- Projections also described by matrix
- Homogenous coordinates crucial
- Parallel and perspective projections

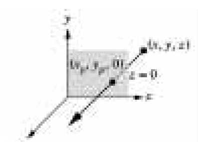
Orthographic Projections

- Parallel projection
- Projectors perpendicular to projection plane
- Simple, but not realistic
- Used in blueprints (multiview projections)



Orthographic Projection Matrix

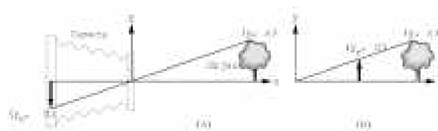
- Project onto z = 0
- $x_p = x$, $y_p = y$, $z_p = 0$
- In homogenous coordinates



$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Viewing

- Characterized by foreshortening
- More distant objects appear smaller



- $y/z = y_p/d$ so $y_p = y/(z/d)$
- Note this is non-linear!

Exploiting the 4th Dimension

- Perspective projection is not affine:

$$M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$ has no solution for M

- Idea: represent point $[x \;\; y \;\; z \;\; 1]^T$ by line in 4D

$$p = w \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ for arbitrary $w \neq 0$

Perspective Projection Matrix

- Represent multiple of point

$$(z/d) \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

- Solve

$$M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$ with $$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

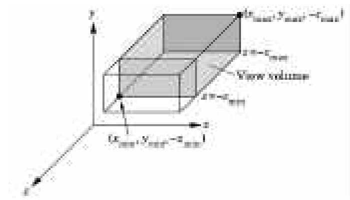
Perspective Division

- Normalize $[x\ \ y\ \ z\ \ w]^T$ to $[(x/w)\ \ (y/w)\ \ (z/w)\ 1]^T$
- Perform perspective division after projection

- Projection in OpenGL is more complex

Parallel Viewing in OpenGL

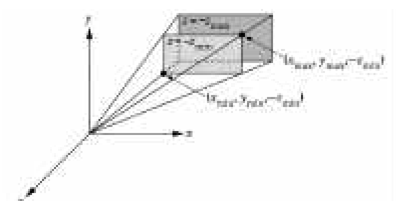
- glOrtho(xmin, xmax, ymin, ymax, near, far)



z_{min} = near, z_{max} = far

Perspective Viewing in OpenGL

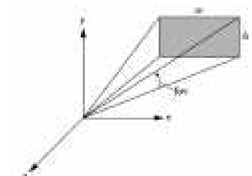
- Two interfaces: glFrustum and gluPerspective
- glFrustum(xmin, xmax, ymin, ymax, near, far);



z_{min} = near, z_{max} = far

Field of View Interface

- gluPerspective(fovy, aspect, near, far);
- near and far as before
- Fovy specifies field of view as height (y) angle



Matrices for Projections in OpenGL

- Next lecture:
  - Use shear for predistortion
  - Use projections for “fake” shadows
  - Other kinds of projections

Announcements

- Assignment 1 due tonight (100 pts)
- Late policy
  - Up to 1 day late, 20% penalty
- Assignment 2 out today, due in 1 week (50 pts)
- Extra credit policy
  - Up to 20% of assignment value
  - Recorded separately
  - Weighed for “borderline” cases
- Remember: no collaboration on assignments!

Looking Ahead

- Lighting and shading
- Video: *Red's Dream*, John Lasseter, Pixar,1987