CSCI 420 Computer Graphics
Lecture 5

Viewing and Projection

Shear Transformation
Camera Positioning
Simple Parallel Projections
Simple Perspective Projections
Angel, Ch. 5

Reminder: Affine Transformations
• Given a point \([x y z]\), form homogeneous coordinates \([x y z 1]\).

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} = \begin{bmatrix}
  m_{11} & m_{12} & m_{13} & m_{14} \\
  m_{21} & m_{22} & m_{23} & m_{24} \\
  m_{31} & m_{32} & m_{33} & m_{34} \\
  m_{41} & m_{42} & m_{43} & m_{44}
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]
• The transformed point is \([x' y' z']\).

Transformation Matrices in OpenGL
• Transformation matrices in OpenGL are vectors of 16 values (column-major matrices).
• In \text{glLoadMatrixf(GLfloat *m)},

\[
m = \{m_1, m_2, \ldots, m_{16}\}
\]
represents

\[
\begin{bmatrix}
  m_1 & m_5 & m_9 & m_{13} \\
  m_2 & m_6 & m_{10} & m_{14} \\
  m_3 & m_7 & m_{11} & m_{15} \\
  m_4 & m_8 & m_{12} & m_{16}
\end{bmatrix}
\]
• Some books transpose all matrices!

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

Specification via Shear Angle
• \(\cot(\theta) = (x' - x) / y\)
• \(x' = x + y \cot(\theta)\)
• \(y' = y\)
• \(z' = z\)

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} \rightarrow \begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix}
\]

\[
H_{x}(\theta) = \begin{bmatrix}
  1 & \cot(\theta) & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

Specification via Ratios
• For example, shear in both x and z direction
• Leave y fixed
• Slope \(\alpha\) for x-shear, \(\gamma\) for z-shear

\[
H_{xz}(\alpha, \gamma) = \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} \rightarrow \begin{bmatrix}
  x + \alpha y \\
  y \\
  z + \gamma y
\end{bmatrix}
\]
• Yields

\[
H_{xz}(\alpha, \gamma) = \begin{bmatrix}
  1 & \alpha & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & \gamma & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]
Composing Transformations

• Let \( p = A q \), and \( q = B s \).

• Then \( p = (A B) s \).

\[
\begin{array}{c}
\text{AB} \\
\text{matrix multiplication}
\end{array}
\]

Composing Transformations

• Fact: Every affine transformation is a composition of rotations, scalings, and translations.

• So, how do we compose these to form an x-shear?

• Exercise!

Outline

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

Transform Camera = Transform Scene

• Camera position is identified with a frame.

• Either move and rotate the objects:

• Or move and rotate the camera:

• Initially, camera at origin, pointing in negative z-direction.

The Look-At Function

• Convenient way to position camera.

• gluLookAt(ex, ey, ez, fx, fy, fz, ux, uy, uz):

• \( e \) = eye point
• \( f \) = focus point
• \( u \) = up vector

OpenGL code

```c
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(ex, ey, ez, fx, fy, fz, ux, uy, uz);
    glTranslatef(x, y, z);
    ...
    renderBunny();
    glutSwapBuffers();
}
```
Implementing the Look-At Function

Plan:

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

World Frame to Camera Frame I

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

World Frame to Camera Frame II

- $R$ maps $[0,1,0]^T$ to projection of $u$ onto view plane
- This projection $v$ equals:
  - $\alpha = (u \cdot n) / |n| = u \cdot n$
  - $v_x = u - \alpha n$
  - $v = v_x / |v_x|$

World Frame to Camera Frame III

- Set $w$ to be orthogonal to $n$ and $v$
- $w = n \times v$
- $(w, v, -n)$ is right-handed

World Frame to Camera Frame IV

- Translation of origin to $e = [e_x \ e_y \ e_z]^T$
- $T = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Summary of Rotation
Camera Frame to Rendering Frame

- \( V = W^{-1} = (T R)^{-1} = R^T T^{-1} \)
- \( R \) is rotation, so \( R^{-1} = R^T \)
- \( T \) is translation, so \( T^{-1} \) negates displacement

\[
R^{-1} = \begin{bmatrix}
w_x & w_y & w_z & 0 \\
v_x & v_y & v_z & 0 \\
-n_x & -n_y & -n_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Putting it Together

- Calculate \( V = R^{-1} T^{-1} \)
- This is different from book [Angel, Ch. 5.3.2]
- There, \( u, v, n \) are right-handed (here: \( u, v, -n \))

Other Viewing Functions

- Roll (about z), pitch (about x), yaw (about y)
- Assignment 2 poses a 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 are described by a \( 4 \times 4 \) matrix
- Homogenous coordinates crucial
- Parallel and perspective projections

Parallel Projection

- Project 3D object to 2D via parallel lines
- The lines are not necessarily orthogonal to projection plane
Parallel Projection
- Problem: objects far away do not appear smaller
- Can lead to “impossible objects”:

Penrose stairs

Orthographic Projection
- A special kind of 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

Perspective
- Perspective characterized by foreshortening
- More distant objects appear smaller
- Parallel lines appear to converge
- Rudimentary perspective in cave drawings:

Lascaux, France

Discovery of Perspective
- Foundation in geometry (Euclid)

Mural from Pompeii, Italy

Middle Ages
- Art in the service of religion
- Perspective abandoned or forgotten

Ottonian manuscript, ca. 1000
Renaissance

- Rediscovery, systematic study of perspective

Filippo Brunelleschi
Florence, 1415

Projection (Viewing) in OpenGL

- Remember: camera is pointing in the negative z direction

Orthographic Viewing in OpenGL

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

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

Perspective Viewing in OpenGL

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

Field of View Interface

- gluPerspective(fovy, aspectRatio, near, far);
- near and far as before
- aspectRatio = w / h
- Fovy specifies field of view as height (y) angle

OpenGL code

```c
void reshape(int x, int y)
{
glViewport(0, 0, x, y);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0, 1.0 * x / y, 0.01, 10.0);
glMatrixMode(GL_MODELVIEW);
}"
```
Perspective Viewing Mathematically

- $d$ = focal length
- $y/z = y/(z/d) = y/d/z$
- Note that $y_p$ is non-linear in the depth $z$!

Exploiting the 4th Dimension

- Perspective projection is not affine:
  $$M = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ \frac{z}{d} \\ 1 \end{bmatrix}$$
  has no solution for $M$

- Idea: exploit homogeneous coordinates
  $$p = w \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
  for arbitrary $w \neq 0$

Perspective Projection Matrix

- Use multiple of point
  $$\begin{bmatrix} \frac{x}{d} \\ \frac{y}{d} \\ \frac{z}{d} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
- Solve
  $$M = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
  with $M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Projection Algorithm

Input: 3D point $(x,y,z)$ to project

1. Form $[x \ y \ z \ 1]^T$
2. Multiply $M$ with $[x \ y \ z \ 1]^T$; obtaining $[X \ Y \ Z \ W]^T$
3. Perform perspective division:
   $$X/W, Y/W, Z/W$$

Output: $(X/W, Y/W, Z/W)$
(last coordinate will be $d$)

Perspective Division

- Normalize $[x \ y \ z \ w]^T$ to $[(x/w) \ (y/w) \ (z/w) \ 1]^T$
- Perform perspective division after projection
  $$\text{Model-view} \rightarrow \text{Projection} \rightarrow \text{Perspective division}$$
- Projection in OpenGL is more complex
  (includes clipping)