Lecture 10: Viewing Transformation

Rui Wang
Before we start

☐ Assignment 2 due today
  - how are we doing?

☐ Next week:
  - Tuesday: Guest lecture by John Bowers (OpenGL + iOS apps)
  - Thursday: Midterm (in-class), 15% of grade!
Where are we?

- Model Transformation
  - Basic 2D/3D transformation
  - Matrix representation and composition
  - Translation and homogeneous coordinates
  - Normal transformation
  - Transformation of coordinate system

- Viewing Transformation
  - Orthographic projection
  - Perspective projection

- 2D → 3D demos

3D → 3D

3D → 2D

(x, y, z) → (x_p, y_p).
Viewing

- Camera
  - Map 3D to 2D

Orthographic  Perspective  Perspective with hidden lines removed
Orthographic Projection

- Assume the eye is looking at $-z$;
- Project everything along the $Z$ axis to the $x$-$y$ plane (drop the $z$ component)
The camera sensor has finite size, so the portion of the world being imaged is bounded. Objects outside this volume will not be displayed.

$$(x, y, z) \in [-1, 1]^3$$

Why the range in z?
Orthographic Projection

- Project everything in volume $[-1,1]^3$ to square $[-1,1]^2$ the x-y plane.

\[
\begin{bmatrix}
x \\ y \\ z
\end{bmatrix} \overset{\text{R}}{\rightarrow} \begin{bmatrix}
x \\ y \\ 0
\end{bmatrix}
\]
Orthographic Projection

- What if the camera wants to look at some other portions of the world?
Orthographic Projection

What if the camera wants to look at some other portions of the world?

If the camera center is at \((cx, cy)\), you can translate everything in the world by \((-cx, -cy)\). This can be done via a simple translation matrix.

Note that the camera transformation and object transformation are relative to each other. If the camera is transformed by \(M\), it’s equivalent to say the camera is still, but the world is transformed by \(M^{-1}\).
Perspective Projection

- Look out the window!
- Common in vision systems
- Further objects are smaller (size, inverse distance)
- Parallel lines are not necessarily parallel after projection; converge to single point
Perspective Projection
Perspective Projection
Perspective Projection
Perspective Projection

- Generated image depends on:
  - the 3D world being imaged
  - camera intrinsic parameters
    - focal length
    - pixel size
    - skewness of axis
  - camera extrinsic parameters
    - position in 3D space
    - orientation in 3D space
Perspective Projection

- Pinhole Camera
  - Aperture is a single point
  - No lens to focus light
Perspective Projection

- Simulate a Pinhole Camera

Camera Center
Perspective Projection

- Simulate a Pinhole Camera

Camera center at origin
Looking along -z
Overhead View of Our Screen

Orthographic Projection

(0, 0, 0) 

(x, y, d) 

(x, y, z) 

d
Overhead View of Our Screen

Perspective Projection

\((0, 0, 0)\) \(\rightarrow\) \((x', y', d)\) \(\rightarrow\) \((x, y, z)\)
Overhead View of Our Screen

Use similar triangles

Perspective Projection
Overhead View of Our Screen

Perspective Projection

Use similar triangles

\[
\frac{x}{z} = \frac{x'}{d} \Rightarrow x' = \frac{d \times x}{z}
\]

\[
\frac{y}{z} = \frac{y'}{d} \Rightarrow y' = \frac{d \times y}{z}
\]
Overhead View of Our Screen

Use similar triangles

\[
\frac{x}{z} = \frac{x'}{d} \Rightarrow x' = \frac{dx}{z}
\]

\[
\frac{y}{z} = \frac{y'}{d} \Rightarrow y' = \frac{dy}{z}
\]

Perspective Projection
Overhead View of Our Screen

Use similar triangles

\[
\frac{x}{z} = \frac{x'}{d} \Rightarrow x' = \frac{d \times x}{z}
\]

\[
\frac{y}{z} = \frac{y'}{d} \Rightarrow y' = \frac{d \times y}{z}
\]
Perspective Projection

- Matrix Representation of Perspective Projection

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{d} & 0
\end{pmatrix}
\]
Perspective Projection

Matrix Representation of Perspective Projection

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{d} & 0
\end{pmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Matrix Representation of Perspective Projection

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{pmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
z/d
\end{bmatrix}
\]

Homogeneous Coords
Perspective Projection

- Matrix Representation of Perspective Projection

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{d} & 0 \\
\end{pmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
x \\
y \\
z \\
\frac{z}{d} \\
\end{bmatrix}
=
\begin{bmatrix}
d \times x \\
z \\
d \times y \\
z \\
d \\
1 \\
\end{bmatrix}
\]
Perspective Projection

The View Frustum

- Cameras have limited field of view.
Perspective Projection

☐ The View Frustum

- After projection, points whose xy fall outside of \([-1,1]^2\) will be clipped.

Vertical Field-of-View (FOV)
Perspective Projection

Matrix Representation

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{d} & 0
\end{pmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
dx \\
z \\
d \times y \\
z \\
d
\end{bmatrix}
\]

Problem: depth information is lost after projection.
Perspective Projection

Matrix Representation

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & f + n & -f \times n \\
0 & 0 & 1 & 0
\end{pmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} =
\begin{bmatrix}
x \\
y \\
z \\
(f + n) - \frac{f \times n}{z}
\end{bmatrix}
\]

Depth info. preserved. Maps z=n to n and z=f to f.
OpenGL Projection Primitives

- Orthographic Projection
  ```
  void glOrtho (left, right, bottom, top, zNear, zFar);
  ```

- Perspective Projection
  ```
  gluPerspective(fovy, aspecratio, zNear, zFar);
  ```
OpenGL Projection Primitives

- Orthographic Projection
  ```c
  void glOrtho (left, right, bottom, top, zNear, zFar);
  ```

- Perspective Projection
  ```c
  gluPerspective(fovy, aspectratio, zNear, zFar);
  ```

  - fovy: vertical field of view (in angles, i.e. 45.f)
  - aspect = window width / height
  - zNear, zFar refer to distance from the camera center to clipping planes, must be strictly >0
OpenGL Projection Primitives

- When setting up projection, must call `glMatrixMode(GL_PROJECTION);` first to indicate that you are going to modify projection matrix.
- Then call `glOrtho` or `gluPerspective`
- After you are done, call `glMatrixMode(GL_MODELVIEW);`
- Otherwise calls to `glTranslatef`, `glRotatef` etc. will end up changing the projection matrix!!
Summary: OpenGL Rendering Pipeline

Model coordinates

Model transformation

Camera transformation (gluLookAt)

World coordinates

Perspective transformation (gluPerspective)

Eye coordinates

Viewport transformation

Screen coordinates

Window coordinates

Raster transformation

Device coordinates

Slide courtesy Greg Humphreys
2D → 3D

- Infinite possibilities (pic adapted from D. Hoeim’s PhD thesis)

- What can we do?