Computer Viewing
Objectives

- Introduce mathematics of projection
- Introduce OpenGL viewing functions
- Look at alternate viewing APIs
Computer Viewing

- Three aspects of viewing process
- All implemented in pipeline
  - Position camera
    - Set model-view matrix
  - Select lens
    - Set projection matrix
  - Clip
    - Set view volume
The OpenGL Camera

- In OpenGL, initially object and camera frames same
  - Default model-view matrix = identity
- Camera
  located at origin
  points in negative z direction
- OpenGL also specifies default view volume
  cube with sides of length 2
  centered at origin
  - Default projection matrix = identity
Default Projection

Default projection is orthogonal

![Diagram of default projection]

- $z = 0$
- Clipped out
- Projection plane
- $z = 0$
Moving the Camera Frame

- If want to visualize object with both positive and negative z values can either
  - Move camera in positive z direction
    - Translate camera frame
  - Move objects in negative z direction
    - Translate world frame
- Both views equivalent and determined by model-view matrix
  - Want translation \( \text{glTranslatef}(0.0, 0.0, -d); \)
  \[ d > 0 \]
Moving Camera back from Origin

default frames

frames after translation by \(-d\)

\(d > 0\)
Moving the Camera

- Can move camera to any desired position by sequence of Rs and Ts
- Example: side view
  - Rotate camera
  - Move it away from origin
  - Model-view matrix $C = TR$
OpenGL code

- Remember that last transformation specified is first to be applied

```c
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(90.0, 0.0, 1.0, 0.0);
```
The LookAt Function

- GLU library contains function gluLookAt form required modelview matrix through simple interface
- Note need to set an up direction
- Still need to initialize
  - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0., 1.0, 0.0);
```
gluLookAt

\[ \text{glLookAt}(\text{eyex}, \text{eyey}, \text{eyez}, \text{atx}, \text{aty}, \text{atz}, \text{upx}, \text{upy}, \text{upz}) \]
Other Viewing APIs

• LookAt function only one possible API for positioning camera
• Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles
Projections and Normalization

- Default projection in eye (camera) frame orthogonal
- For points within default view volume

\[
\begin{align*}
    x_p &= x \\
    y_p &= y \\
    z_p &= 0
\end{align*}
\]

- Most graphics systems use view normalization
  - All other views converted to default view by transformations that determine projection matrix
  - Allows use of same pipeline for all views
Homogeneous Coordinate Representation

default orthographic projection

\[ x_p = x \]
\[ y_p = y \]
\[ z_p = 0 \]
\[ w_p = 1 \]

\[ p_p = Mp \]

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

In practice, can let \( M = I \) and set \( z \) term to zero later.
Simple Perspective

- Center of projection at origin
- Projection plane $z = d, d < 0$
Perspective Equations

Consider top and side views

\[
\begin{align*}
    x_p &= \frac{x}{z/d} \\
    y_p &= \frac{y}{z/d} \\
    z_p &= d
\end{align*}
\]
Homogeneous Coordinate Form

consider \( q = Mp \)

where

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\]

\[
q = \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} \Rightarrow p = \begin{bmatrix}
x \\
y \\
z \\
z/d
\end{bmatrix}
\]
Perspective Division

• However $w \neq 1$, so must divide by $w$ to return from homogeneous coordinates

• *perspective division* $===>$

\[
\begin{align*}
    x_p &= \frac{x}{z/d} \\
    y_p &= \frac{y}{z/d} \\
    z_p &= d
\end{align*}
\]

desired perspective equations

• Will consider corresponding clipping volume with OpenGL functions
OpenGL Orthogonal Viewing

\[ \text{glOrtho}(left, right, bottom, top, near, far) \]

near and far measured from camera
OpenGL Perspective

$\text{glFrustum}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{near}, \text{far})$
Using Field of View

- With `glFrustum` often difficult to get desired view
- `gluPerspective(fovy, aspect, near, far)` often provides better interface

![Diagram](image)

- `aspect = w/h`
- Front plane