Computer Viewing
Objectives

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

• There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    • Setting the model-view matrix
  - Selecting a lens
    • Setting the projection matrix
  - Clipping
    • Setting the view volume
The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
  - The default model-view matrix is an identity
- The camera is located at the origin and points in the negative $z$ direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - The default projection matrix is an identity
The Default Projection

The default projection is orthogonal

The default clipping volume is a cube of side = 2 centred at the origin.

Projection plane $z = 0$
Moving the Camera Frame

• If we want to visualize objects that have both positive and negative $z$ values we can either
  - Move the camera in the positive $z$ direction
    • Translate the camera frame
  - Move the objects in the negative $z$ direction
    • Translate the world frame

• Both of these views are equivalent and are determined by the model-view matrix
  o Want a translation ($\text{Translate}(0.0,0.0,-d);$)
  o Want $d > 0$, i.e., we move the objects in the $-z$ direction.
Moving Camera back from Origin

default frames

frames after translation by $-d$

where $d > 0$
Moving the Camera

• We can move the camera to any desired position by a sequence of rotations and translations

• Example: side view at the +x axis looking towards the origin
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix \( C = TR \)
Moving the Camera – OpenGL code

• Remember that the last transformation specified is first to be applied

```
// Using mat.h

mat4 t = Translate (0.0, 0.0, -d);
mat4 ry = RotateY (90.0);
mat4 m = t*ry;
```
The LookAt() Function

camera is located at a point \( e \) called the **eye point**, specified in the object frame, and it is pointed at a second point \( a \), called the **at point**.
The LookAt() Function

• The GLU library contains the function \texttt{gluLookAt} which can be used to form the required model-view matrix.
  \texttt{gluLookAt(eyeX, eyeY, eyeZ, centreX, centreY, centreZ, upX, upY, upZ)}

• We need to define the eye (camera) position, the centre (fixation point), and an up direction. All are of type \texttt{GLdouble}.

• Alternatively, we can use \texttt{LookAt()} defined in \texttt{mat.h}
  - The function returns a \texttt{mat4} matrix.
  - Can concatenate with modeling transformations

• Example:

\[
\texttt{mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);} \quad \text{Type: GLfloat}
\]
The LookAt() Function (cont.)

mat4 viewMat = LookAt(eye, at, up);
Other Viewing APIs

• The `LookAt()` function is only one possible API for positioning the camera

• Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll (angles)
  - Elevation, azimuth, twist (angles)
  - Direction angles
PROJECTION
Default Orthographic Projection

• The default projection in the eye (camera) frame is orthogonal
• For a point \( \mathbf{p} = (x, y, z, 1)^T \) within the default view volume, it is projected to \( \mathbf{p}_p = (x_p, y_p, z_p, w_p)^T \), where
  \[
  x_p = x, \quad y_p = y, \quad z_p = 0, \quad w_p = 1
  \]
• i.e., we can define
  \[
  \mathbf{M} = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
  and we can then write \( \mathbf{P}_p = \mathbf{M} \mathbf{p} \)
• In practice, we can let \( \mathbf{M} = \mathbf{I} \) and set \( z \) term to 0 later
Orthogonal Viewing

• The OpenGL orthogonal viewing function is:
  ```
  glOrtho(left, right, bottom, top, near, far)
  ```

• Alternatively, we can use `Ortho()` defined in `mat.h`:
  ```
  mat4 Ortho(left, right, bottom, top, near, far)
  ```

`near` and `far` are measured from camera

Type: GLfloat
Should be +ive

Type: GLdouble

Simple Perspective

- In orthographic projection, the camera’s focal length is considered to be infinite (i.e., the camera lens is at infinity).
- In perspective projection, the camera’s focal length $d$ is finite.
- A simple perspective projection:
  - Center of projection is at the origin.
  - Projection plane $z = d$, where $d < 0$. 

---

Simple Perspective (cont.)

Consider the top and side views

Recall: the OpenGL synthetic camera model in an earlier lecture

\[
\begin{align*}
x_p &= \frac{x}{z/d} \\
y_p &= \frac{y}{z/d} \\
z_p &= d
\end{align*}
\]
Simple Perspective (cont.)

Consider $q = Mp$ where

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

and $p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

$\Rightarrow q = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$

In OpenGL, this is the $w$ term
Perspective Division

• However, since \( w = \frac{z}{d} \neq 1 \), so we must divide by \( w \) to return back to non-homogeneous coordinates.

• This \textit{perspective division} yields

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

which are the desired perspective equations, as on slide 18.
Perspective Viewing

• To define a perspective transformation matrix for the camera, we can use

```c
mat4 Frustumum(left,right,bottom,top,near,far)
```
defined in `mat.h`:

All are of type `GLfloat`

Combines perspective projection and clipping
Perspective Viewing – Using Field of View

• With `Frustum()` it is often difficult to get the desired view. Another way to get perspective projection is:

```cpp
mat4 Perspective(fovy, aspect, near, far)
```

which often provides a better interface.

**Note:**
- `aspect = w/h`
- `fovy` is an angle in degrees

All are of type `GLfloat`
References


• Secs. 4.1. Classical and Computer Viewing; 4.1.2. Orthographic Projections; 4.1.5 Perspective Viewing
• Sec. 4.2. Viewing with a Computer
• Sec. 4.3.1. Positioning of the Camera Frame; 4.3.3. The Look-At Function
• Sec. 4.4.1. Orthographic Projections; 4.4.2. Parallel Viewing with OpenGL; 4.4.4. Orthogonal-Projection Matrices; (optional) 4.4.6 An Interactive Viewer
• Secs. 4.5. – 4.7. Projections – Perspective-Projection Matrices