Computer Viewing

Objectives

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

Computer Viewing

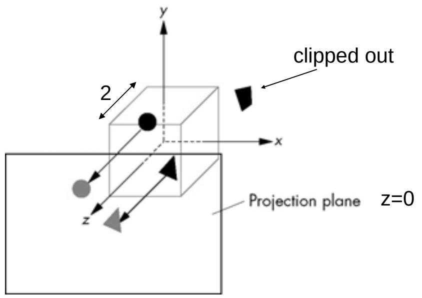
- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at 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
  - Default projection matrix is an identity

Default Projection

Default projection is orthogonal



Moving the Camera Frame

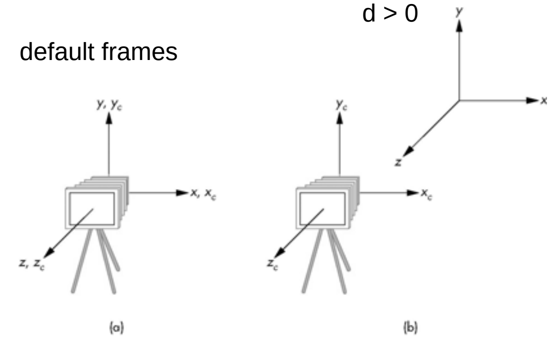
- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`glTranslatef(0.0,0.0,-d);`) `where d > 0`

Moving Camera back from Origin

frames after translation by –d

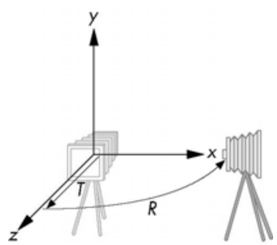
d > 0

default frames



Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix C = TR



OpenGL code

- Remember that last transformation specified is first to be applied

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(90.0, 0.0, 1.0, 0.0);
```

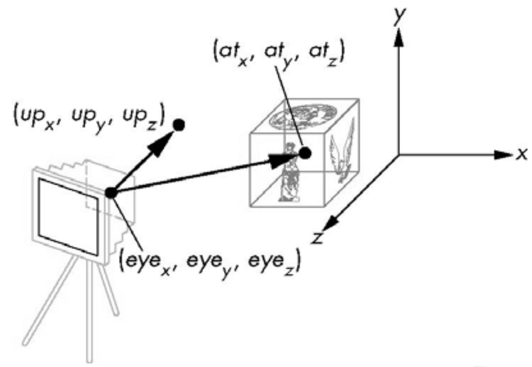
The LookAt Function

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

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


gluLookAt

```
glLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```



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) (see 5.3.2)
  - Yaw, pitch, roll (see 5.3.4)
  - Elevation, azimuth, twist (see 5.3.4)
  - Direction angles

Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

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

Homogeneous Coordinate Representation

default orthographic projection

$x_p = x$
$y_p = y$
$z_p = 0$
$w_p = 1$

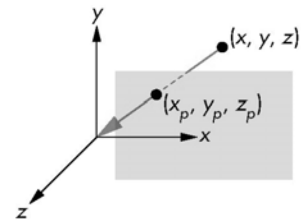
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

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

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later

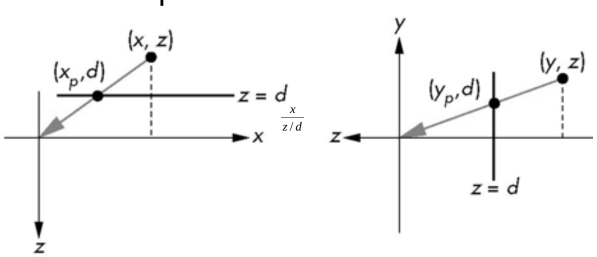
Simple Perspective

- Center of projection at the origin
- Projection plane $z = d,\ d < 0$



Perspective Equations

Consider top and side views



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Homogeneous Coordinate Form

consider $\mathbf{q} = \mathbf{M}\mathbf{p}$ where $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

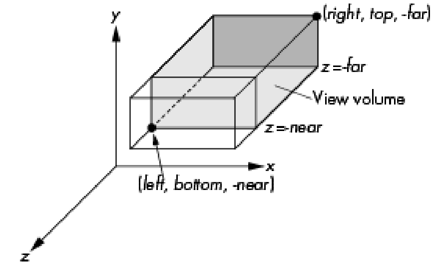
$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

- We will consider the corresponding clipping volume with the OpenGL functions

OpenGL Orthogonal Viewing

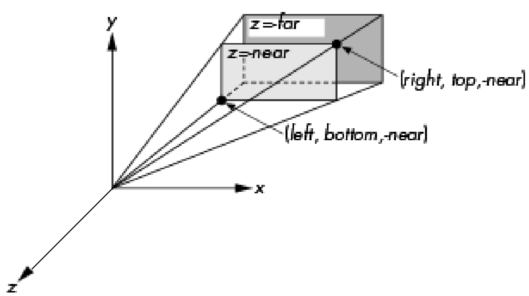
`glOrtho(left,right,bottom,top,near,far)`



`near` and **`far`** measured <u>from</u> camera

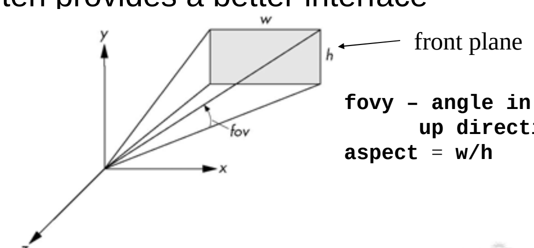
OpenGL Perspective

`glFrustum(left,right,bottom,top,near,far)`



Using Field of View

- With **`glFrustum`** it is often difficult to get the desired view
- **`gluPerpective(fovy, aspect, near, far)`** often provides a better interface



`fovy` – angle in up direction
`aspect` = w/h