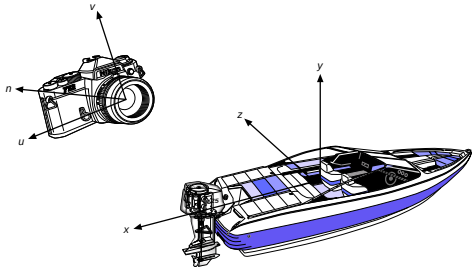
Specifying a View

- How does a programmer specify the view parameters to a computer?
- Similar to taking a picture.



Slide 1

View Parameters

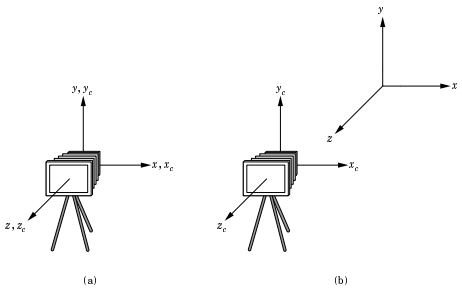
- Position of the camera.
  $pos = (pos_x, pos_y, pos_z)$.
- Orientation
  - Point at which camera is focused.
    $look = (look_x, look_y, look_z)$.
  - Orientation of the camera.
    $UP = (UP_x, UP_y, UP_z)$.
- Field of view.
  - Aspect ratio, angle of view.
- Depth of field
  - Near and far distances.
- Perspective/parallel projection.
- Determine focal distance.

Slide 2

View Parameters

Default OpenGL parameters

- $pos = (0.0, 0.0, 0.0)$
- $look = (0.0, 0.0, -1.0)$
- $UP = (0.0, 1.0, 0.0)$



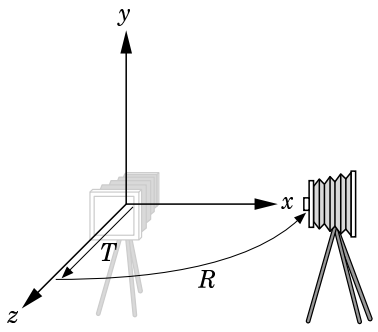
Change the view point to $(0, 0, 10)$

- Translate every object by $(0, 0, -10)$
  `glTranslated` $(0.0, 0.0, -10.0)$

Slide 3

View Parameters

Change $pos = (5, 0, 0)$ and $look = (-1, 0, 0)$!

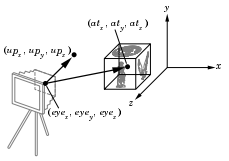


`glMatrixMode` (GL_MODELVIEW)
`glLoadIdentity` ()
`glTranslatef` $(0.0, 0.0, -5)$
`glRotatef` $(-90.0, 0.0, 1.0, 0.0)$

Slide 4

Viewing in OpenGL

`gluLookAt`(*eye*, *at*, *up*)



- ☆ *eye = (eyex, eyey, eyez)*: Viewpoint position
- ☆ *at = (atx, aty, atz)*: Any point along the line of sight; typically the center of the image.

  $look = at - eye$
- ☆ *up = (upx, upy, upz)*: Up direction

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

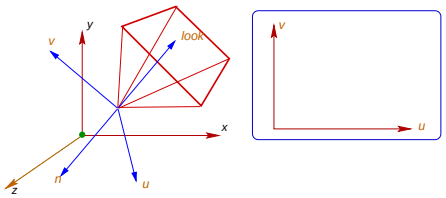
Default command

`gluLookAt`(0.0, 0.0, 0.0, 0.0, 0.0, −1.0, 0.0, 1.0, 0.0)

Typically appears at the beginning of the program.

Slide 5

Coordinate Systems



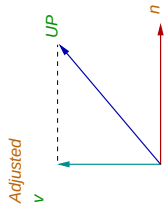
- ☆ *World coordinate system*

  Standard $x-$, $y-$, z-coordinates.
- ☆ *Viewing coordinate system*

  (u, v, n) coordinates.
  - Origin at *position*.
  - *look* is in $-n$-direction.
  - v is determined by *UP* vector. *UP* is not necessarily normal to n, so a correction is required.
  - u is normal to *UP* and n.
  - Coordinate system satisfies the right-hand rule.

Slide 6

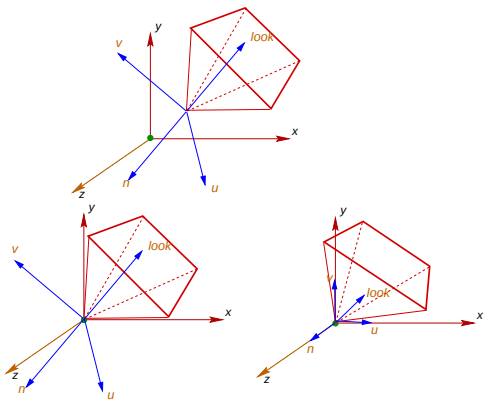
Computation with u, v, n



$$n = \frac{-look}{\|look\|} \qquad u = \frac{look \times UP}{\|look \times UP\|} \qquad v = \frac{u \times look}{\|u \times look\|}$$

Slide 7

World to View Coordinates



Slide 8

World to View Coordinates

- $pos = (pos_x, pos_y, pos_z)$,
- $u = (u_x, u_y, u_z)$,
- $v = (v_x, v_y, v_z)$,
- $n = (n_x, n_y, n_z)$

- Perform a translation T_C so that pos maps to the origin.

$$T_C = \begin{bmatrix} 0 & 0 & 0 & -pos_x \\ 0 & 0 & 0 & -pos_y \\ 0 & 0 & 0 & -pos_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Perform a rotation R_C so that $u \to x$, $v \to y$, and $n \to z$.

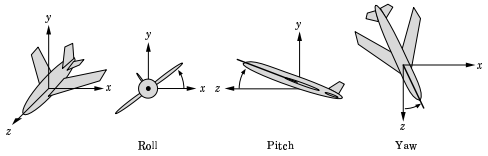
$$R_C = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $M_C = R_C \cdot T_C$.

Slide 9

Viewing Transforms

Example: Flight simulator



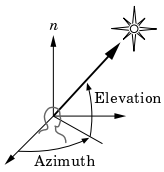

```
void pilotview { GLdouble planex, GLdouble planey
                 GLdouble planez, GLdouble roll
                 GLdouble pitch,  GLdouble yaw }
  {
   glRotated (roll, 0.0, 0.0, 1.0);
   glRotated (pitch, 1.0, 0.0, 0.0);
   glRotated (yaw, 0.0, 1.0, 0.0);
   glTranslated (-planex, -planey, -planez);
   }
```


Slide 10

Viewing Transforms

Example: Oribiting around an object centered at the origin.

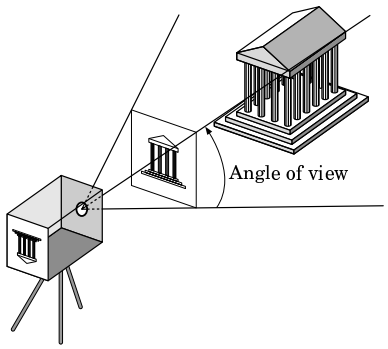
- radius
- azimuth
- elevation
- twist

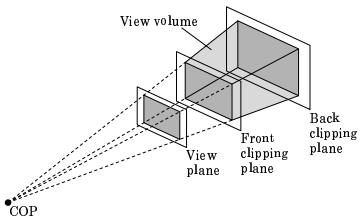



```
void polarview {GLdouble radius, GLdouble azimuth
                GLdouble elevation, GLdouble twist}
  {
   glTranslated (0.0, 0.0, -radius);
   glRotated (-twist, 0.0, 0.0, 1.0);
   glRotated (-elevation, 1.0, 0.0, 0.0);
   glRotated (-azimuth, 0.0, 1.0, 0.0);
   }
```


Slide 11

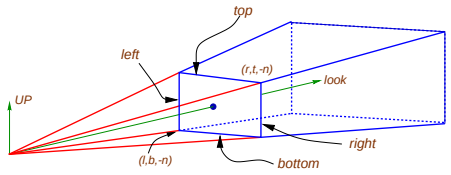
Perspective Projection





Slide 12

Specifying Views in OpenGL



Viewing volume:

- ☆ Defines the region of visible objects.
- ☆ Objects lying outside the viewing volume are discarded.

  `glFrustum`(l, r, b, t, n, f)
- ☆ Frustum need not be symmetric with respect to the line of sight.

Slide 13

Perspective Projection

l: x-coordinate of the left edge of the near plane.

r: x-coordinate of the right edge of the near plane.

b: y-coordinate of the bottom edge of the near plane.

t: y-coordinate of the top edge of the near plane.

n: z-coordinate of the near plane.

f: z-coordinate of the far plane.

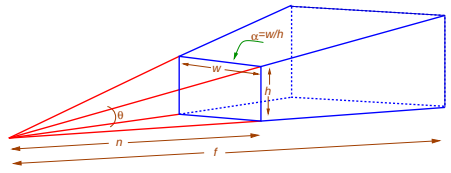
$(l, b, -n)$: left-bottom corner of the near plane.

$(r, t, -n)$: top-right corner of the near plane.

Slide 14

Perspective Projection

Estimating l, r, b, t is difficult!



`gluPerspective`(θ, α, n, f)

θ: Angle of the field view; $\theta \in [0.0, 180.0]$.

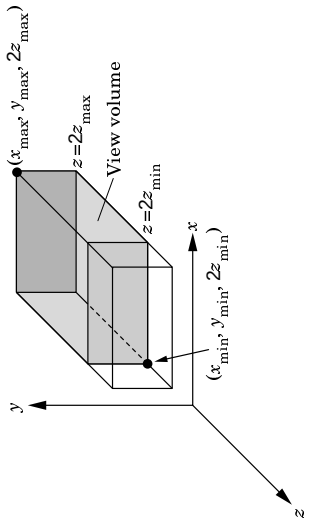
α: Aspect ratio (x-length/y-length).

Line of sight is the $(-z)$-axis.

Frustum symmetric with respect to the line of sight.

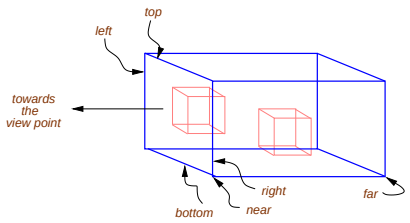
Slide 15

Otrhographic Projection



Slide 16

Otrhographic Projection

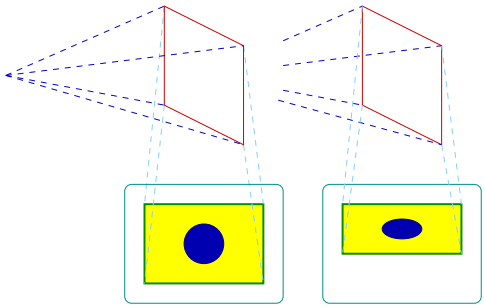


- ☆ `glOrtho` (l, r, b, t, n, f)
  Arguments are the same as in `glFrustum`.
- ☆ `glOrtho2d` (l,r,b,t)
  Same as `glOrtho` (l, r, b, t, 0, 0).
  Useful for 2D viewing.

Slide 17

ViewPort Transform in OpenGL

Projection plane to screen transformation.



`glViewport`(x, y, w, h)

$x, y,$: Lower left corner of the viewport.

w: Width of the viewport.

h: Height of the viewport.

Slide 18

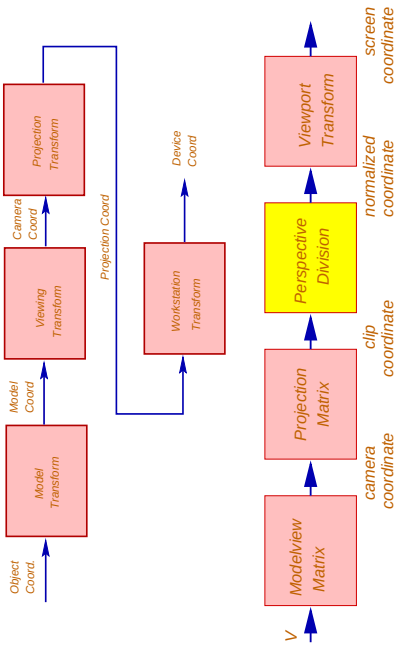
OpenGL Matrices

- ☆ Modelview matrix
- ☆ Projection matrix
- ☆ Texture matrix

- ☆ `glMatrixMode` (GLenum mode);
  - GL_MODELVIEW, GL_PROJECTION
    GL_TEXTURE
- ☆ OpenGl maintains a stack of each of three matrices
  - GL_MAX_MODELVIEW_STACK_DEPTH
    gives the maximum depth of the stack of modelview matrix.
  - Similar commands for other matrices.
  - `glPushMatrix`, `glPopMatrix`

Slide 19

View pipeline

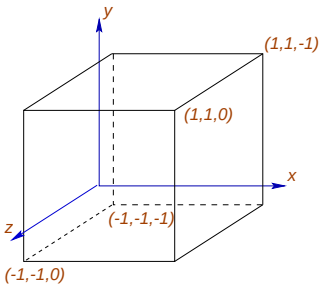


Slide 20

Canonical View

☆ $eye = (0,0,0)$, $look = (0,0,-1)$, $UP = (0,1,0)$.

☆ Viewing volume is always the parallel box $[-1,+1] \times [-1,+1] \times [-1,0]$



Simplifies clipping, projection, hidden surface removal

☆ Projection: ignore the z-coordinate

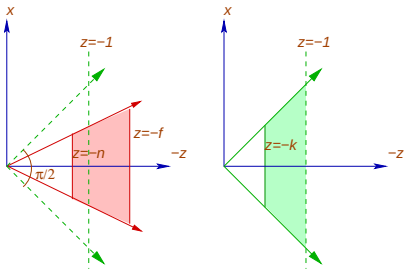
☆ Hidden surface removal: compare z-coordinates

Slide 21

Canonical View

Perspective transform

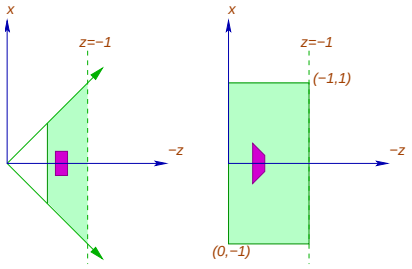
☆ Perform a transformation so that
- The far clipping plane is $z = -1$
- Corners of the clipping rectangle on the far plane are $(\pm 1, \pm 1, -1)$



Slide 22

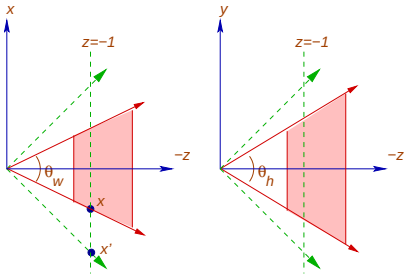
Canonical View

☆ Perform a perspective transformation so that
- Viewing volume is a parallel box
- near clipping plane is $z = 0$, far plane is at $z = -1$
- Distorts the objects



Slide 23

Canonical View: Step I



$x = (\tan(\theta_w/2), 0, -1)$, $x' = (1, 0, -1)$
$y = (0, \tan(\theta_h/2), -1)$, $x' = (0, 1, -1)$

☆ Need to scale x by $1/\tan(\theta_w/2) = \cot(\theta_w/2)$

☆ Need to scale y by $1/\tan(\theta_h/2) = \cot(\theta_h/2)$

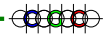
$$S_{xy} = \begin{bmatrix} \cot(\theta_w/2) & 0 & 0 & 0 \\ 0 & \cot(\theta_h/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slide 24

Canonical View: Step I

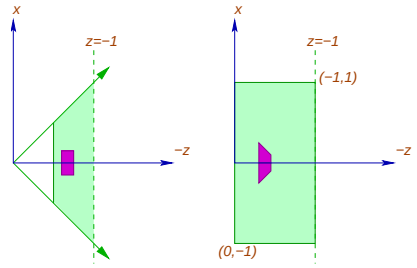
- ☆ Far clipping plane may lie at $z \neq 1$
- ☆ Scale so that the far clipping plane is $z = -1$
- ☆ S_{xy} does not scale the z-coordinates, so far plane is at $z = -f$.
- ☆ Perform a uniform scaling by $1/f$

$$S_2 = \begin{bmatrix} 1/f & 0 & 0 & 0 \\ 0 & 1/f & 0 & 0 \\ 0 & 0 & 1/f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Slide 25

Canonical View: Step II



- ☆ Suppose that the near clipping plane ($z = -n$) after applying $S_2 \cdot S_{xy}$ is $z = -k$.
- ☆ Show that $k = n/f$!

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1-k) & k/(1-k) \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\text{PROJ} = D \cdot S_2 \cdot S_{xy}$$

Slide 26

Reversing Transforms

```
gluUnproject(GLdouble winx, GLdouble winy,
        GLdouble winz,
        const GLdouble modelmatrix[16],
        const GLdouble projectmatrix[16],
        const GLint viewport[4], GLdouble *objx
        GLdouble *objy, GLdouble *objz)
```

- ☆ Returns the world coordinate of the point (winx, winy, winz) at the window.
- ☆ Useful for interactive system.
- ☆ Window has only x and y coordinates. Need to provide the third coordinate winz.

```
gluProject(GLdouble objx, GLdouble objy,
              GLdouble objz,
              const GLdouble modelmatrix[16],
              const GLdouble projectmatrix[16],
              const GLint viewport[4], GLdouble *winx
              GLdouble *winy, GLdouble *winz)
```

- ☆ Returns the window coordinates of the point (objx, objy, objz) in the world coordinates.

Slide 27