Computer Graphics

(Viewing, or: the mystery of "glOrtho")

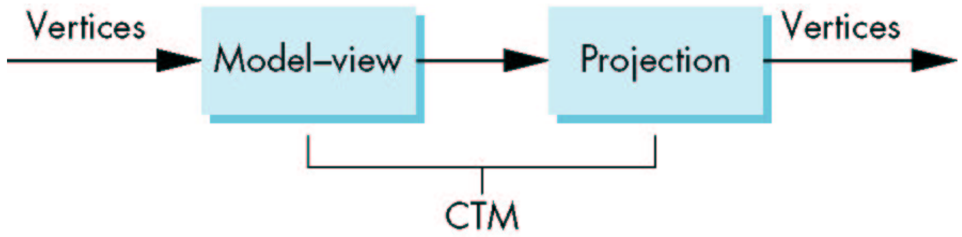
Thilo Kielmann
Fall 2003
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

http://www.cs.vu.nl/~graphics/

Outline for today

- Classical viewing
- Positioning the camera
- Projections in OpenGL
- Walking through a scene
- Projection matrices

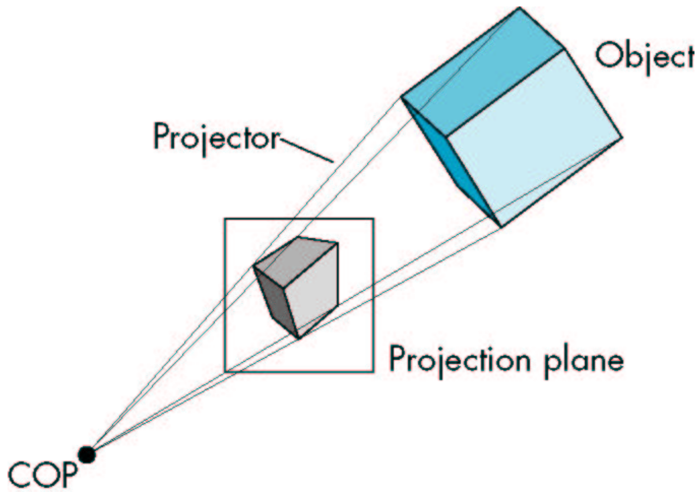
Model-View and Projection Matrices



CTM: Current Transformation Matrix

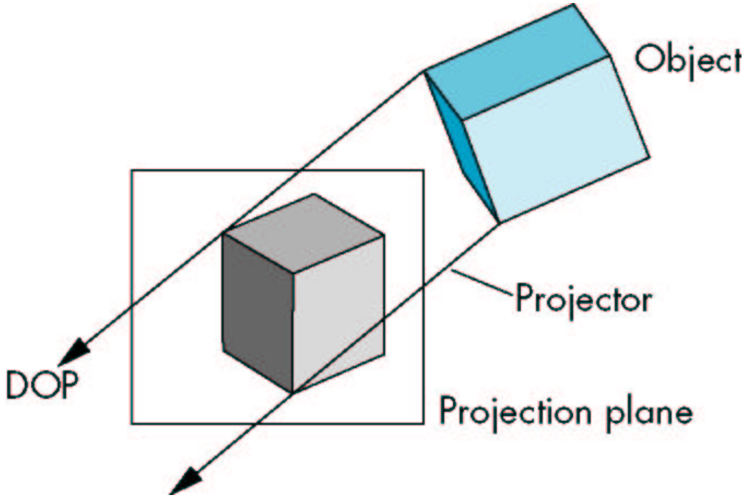
- model-view matrix: objects → world frame
- projection matrix: world frame → camera frame

Viewing



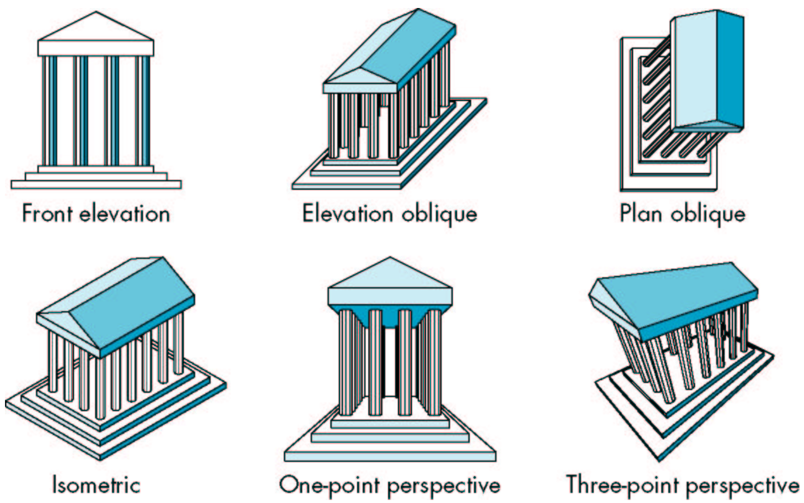
COP = center of projection

Moving the COP to Infinity

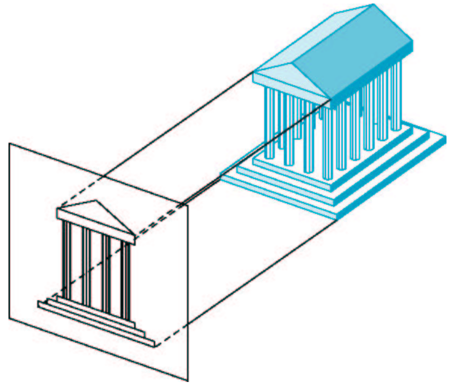


DOP = direction of projection

Classical Views

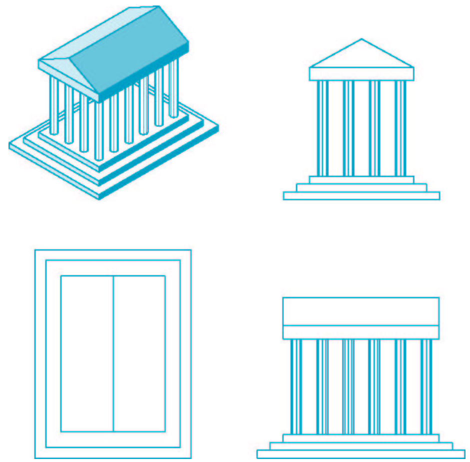


Orthographic Projection



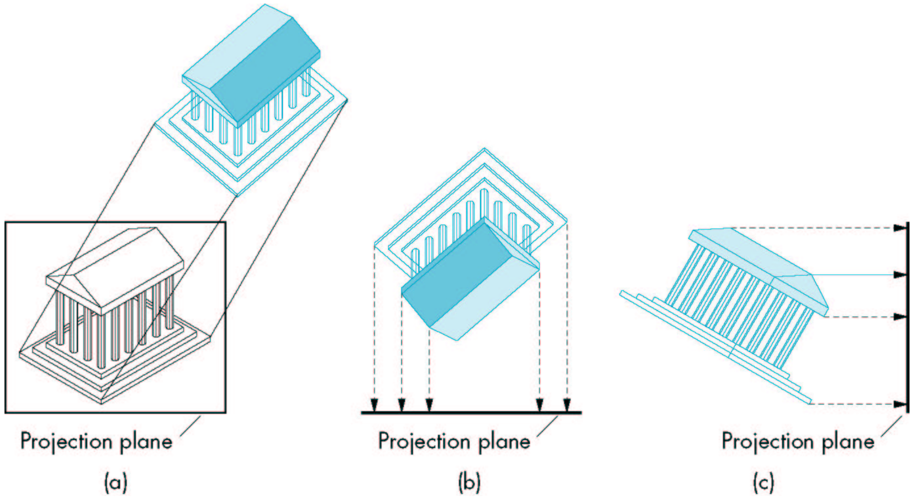
Projectors are perpendicular to the projection plane.

Multiple Orthographic Projections



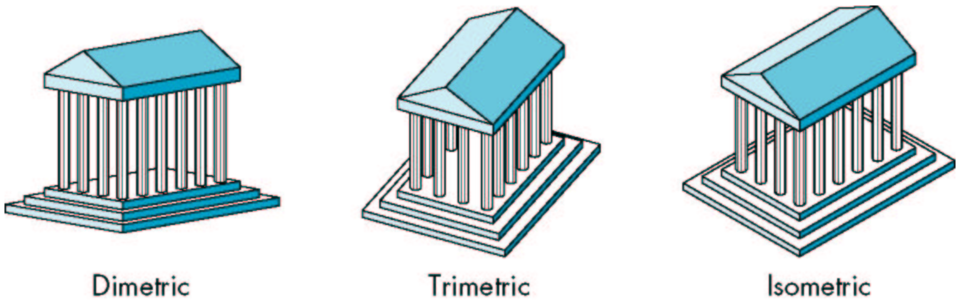
(temple and three principal faces)

Axonometric Projections



trimetric view top view side view

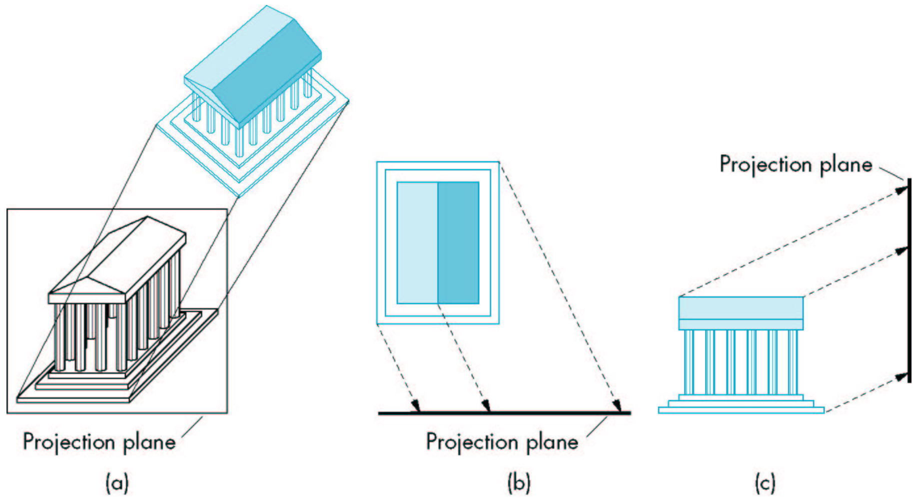
Axonometric Views



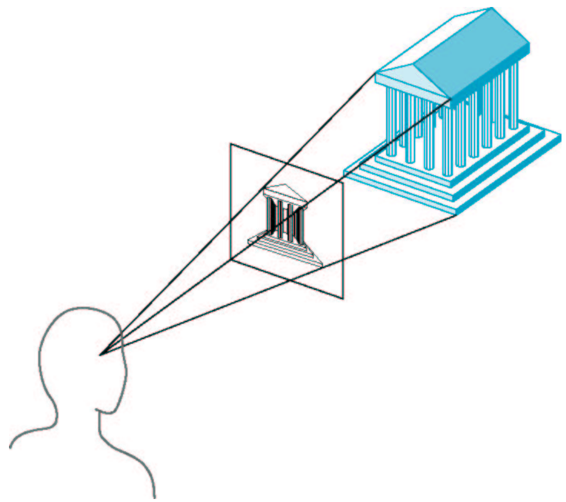
2 3 1

number of foreshortening factors

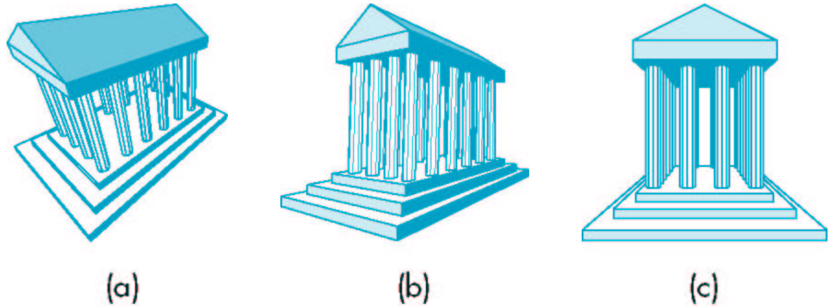
Oblique View



Perspective Viewing



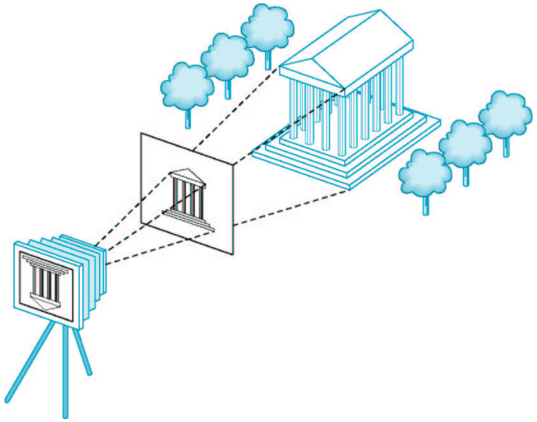
Three- Two- One-Point Perspectives



three two one

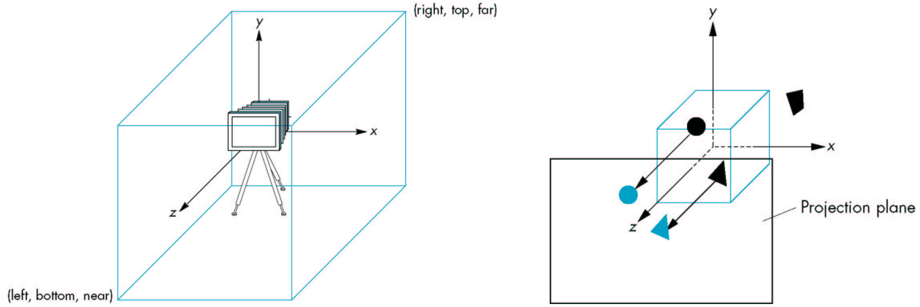
special cases of general perspective viewing

Viewing with a Computer



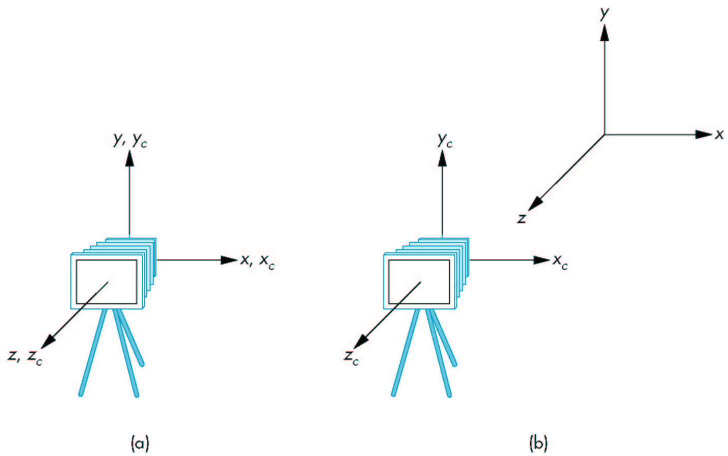
synthetic camera model

Default Camera with OpenGL

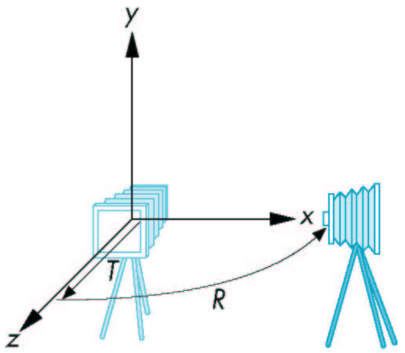


- Camera at origin, pointing to negative z direction
- Viewing volume, centered at origin, sides of length 2
- Orthographic projection

Moving the Camera with the Model-View Matrix

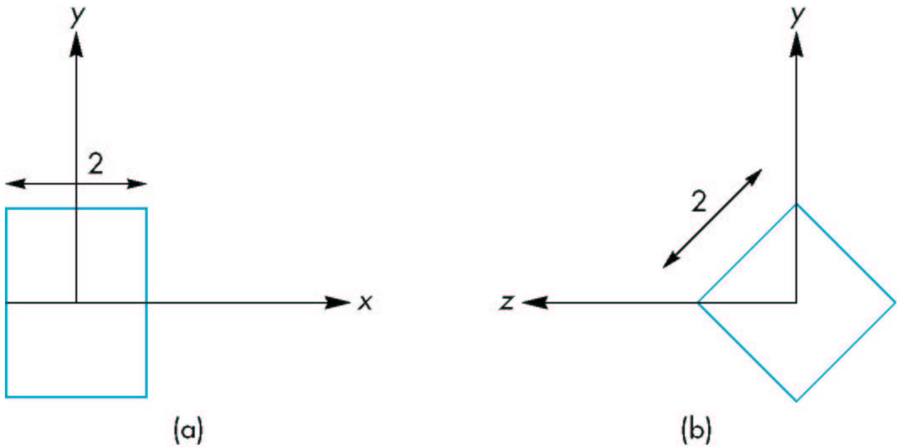


Positioning the Camera




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


Let's try an Isometric View



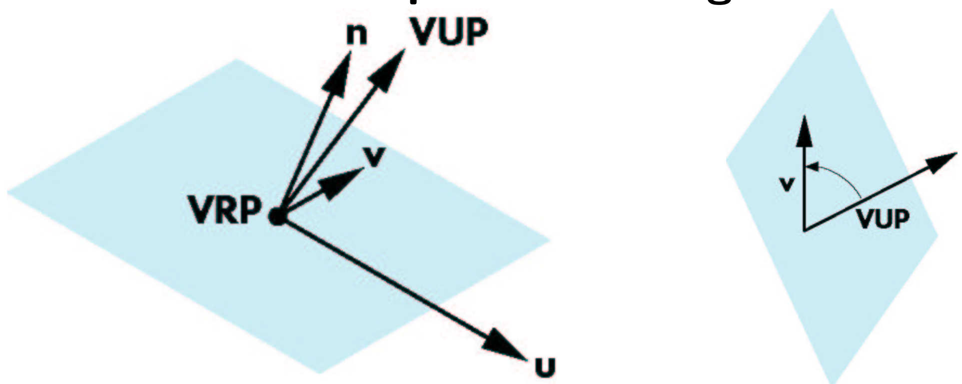
Isometric View (2)

We need to get: $M = TR_xR_y$

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(-35.26, 1.0, 0.0, 0.0);
glRotatef(45.0, 0.0, 1.0, 0.0);
```

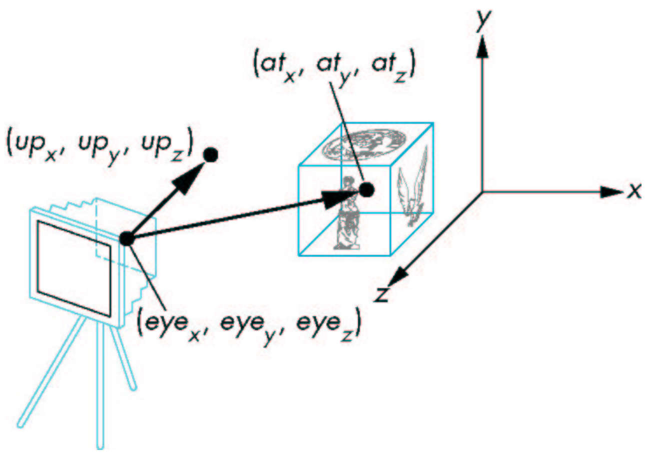
. . . not really convenient: we are mixing properties of viewing into the description of the objects :-(

Better: A Separate Viewing API



View Reference Point (VRP), View Plane Normal (n), View-Up Vector (VUP)
$u - v - n$ viewing coordinate system

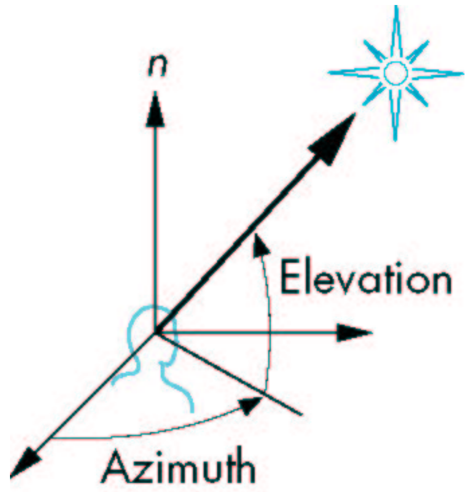
Look-at Positioning



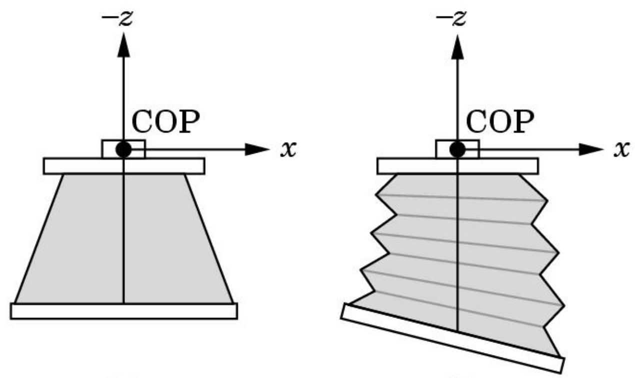
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);

Demo of gluLookAt

Alternative Positionings . . .



Cameras (Projection Planes etc.)

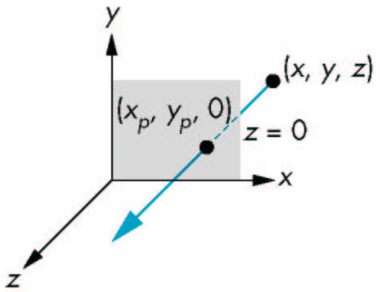


(a) we deal with this

(b) general case

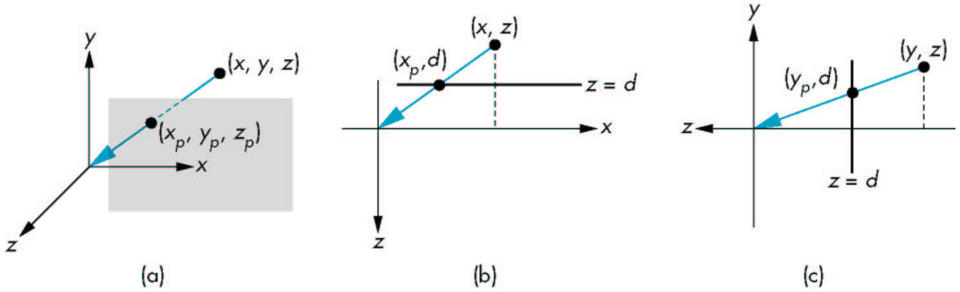
Orthogonal (Orthographic) Projections

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



$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection



3D-view   top view   side view

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

Perspective Projection (2)

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

- non-uniform foreshortening (parameter: z)
- perspective transformation is non-reversible
  - ⋆ all points along a projector are put on the same point in the plane
- preserving information: $p = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \qquad w \neq 0$

  divide w_x, w_y, w_z by w to retain 3d point

Perspective Projection

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

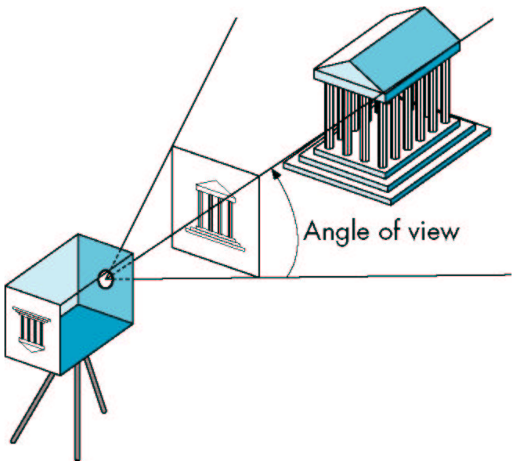
$$q' = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Projection Pipeline

(simple) perspective projection can be done by a 4×4 matrix **and** a perspective division

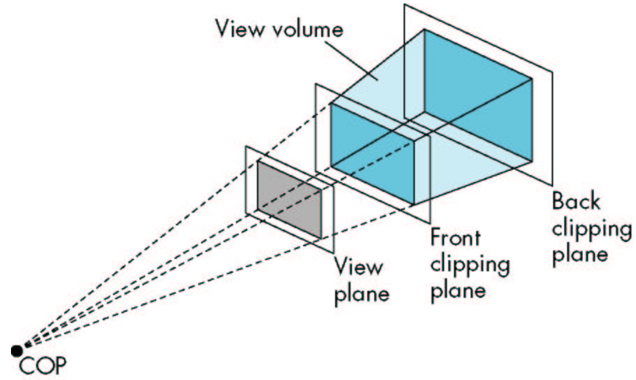
This adds another step (perspective division) to the projection pipeline.

Viewing (Clipping) Volume



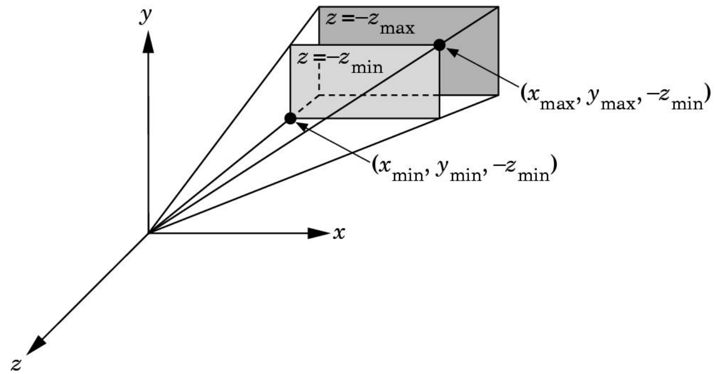
here: pyramid

Front and Back Clipping Planes



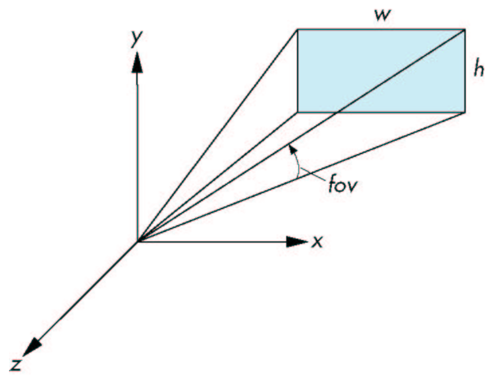
defines a **frustum**, a truncated pyramid

glFrustum



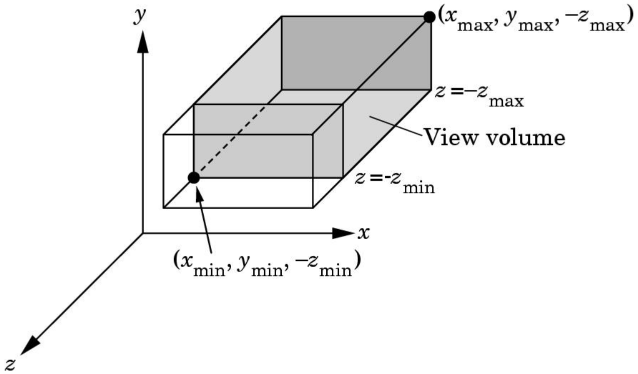

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(xmin, xmax, ymin, ymax, zmin, zmax);
```

gluPerspective




```
gluPerspective(fovy, aspect, near, far);
```

glOrtho




```
glOrtho(xmin, xmax, ymin, ymax, zmin, zmax);
```

Walking Through a Scene

```
void display(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  /* Update viewer position in modelview matrix */
  glLoadIdentity();
  gluLookAt(viewer[0],viewer[1],viewer[2],
            0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glRotatef(theta[0], 1.0, 0.0, 0.0);
  glRotatef(theta[1], 0.0, 1.0, 0.0);
  glRotatef(theta[2], 0.0, 0.0, 1.0);
  glScalef(scaling, scaling, scaling);
  colorcube();

  glFlush();
  glutSwapBuffers();
}
```


Walking Through a Scene (2)

```
void keys(unsigned char key, int x, int y){
/* Use x, X, y, Y, z, and Z keys to move viewer */
   if(key == 'q') exit(0);
   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   if(key == 's') scaling /= 2.0;
   if(key == 'S') scaling *= 2.0;

   glutPostRedisplay();
}
```

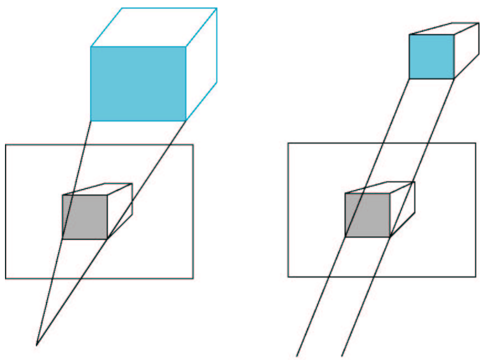
Walking Through a Scene (3)

```
void myReshape(int w, int h){
  glViewport(0, 0, w, h);

  /* Use a perspective view */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,
                     2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
  else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,
                 2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

  glMatrixMode(GL_MODELVIEW);
}
```

Predistortion of Objects



(a) (b)

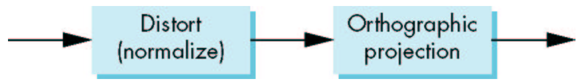
perspective view — orthographic view of predistorted object

Projection in the Rendering Pipeline

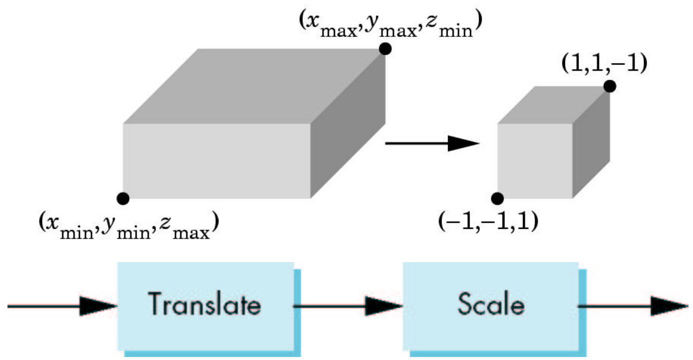
Goal: Efficient (uniform) implementation for all variations of viewing:

- Perspective
- Orthogonal
- Oblique (parallel)

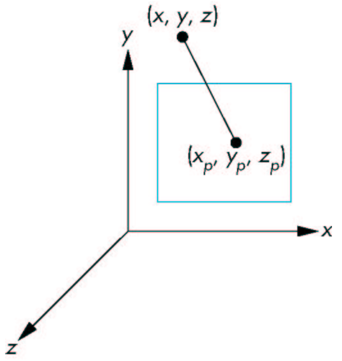
Predistortion as Normalization:

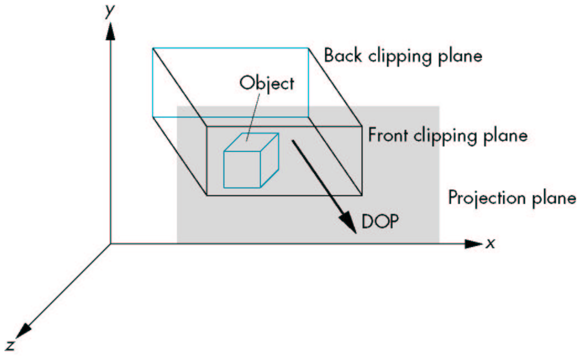


Mapping to the Canonical View Volume



Oblique (parallel) Projection



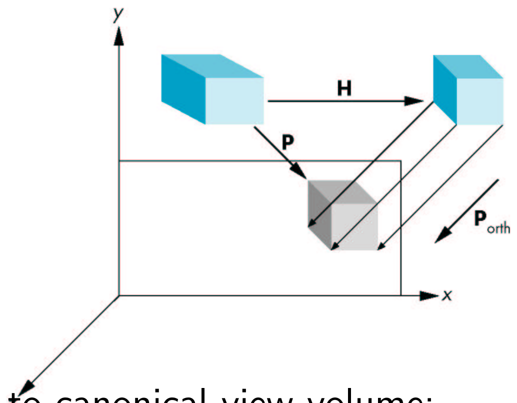


Oblique Projection (2)

$$P = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$P_{\text{orth}} H(\theta, \phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

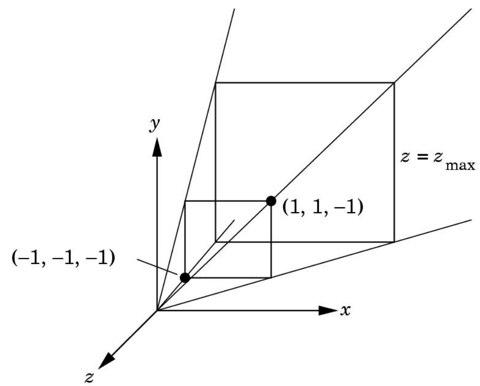
Effect of Shear Transformation



. . . and map to canonical view volume:

$P = P_{\text{orth}} S T H(\theta, \phi)$

Perspective Normalization



Idea: map perspective to orthogonal projection by "predistortion"

Perspective Projection Matrices

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad \alpha \neq 0 \quad \beta \neq 0$$

$$q = Np = N[x\ y\ z\ 1]^T = [x'\ y'\ z'\ w']^T$$

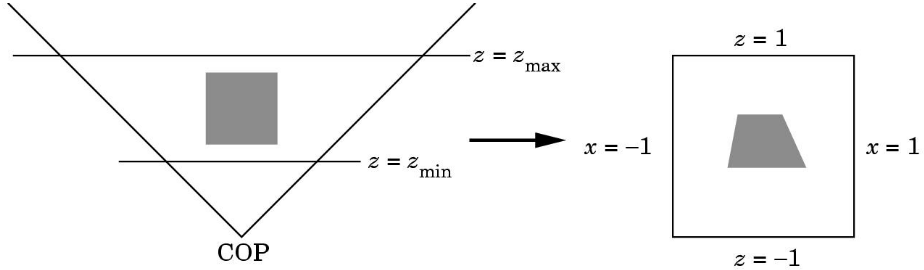
$$\begin{aligned} x' &= x & \qquad x'' &= -\tfrac{x}{z} \\ y' &= y & y'' &= -\tfrac{y}{z} \\ z' &= \alpha z + \beta & z'' &= -\left(\alpha + \tfrac{\beta}{z}\right) \\ w' &= -z \end{aligned}$$

Perspective Projection Matrices (2)

$$P_{orth}N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad p' = P_{orth}Np = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix}$$

after perspective division: $x_p = -\frac{x}{z}$ $y_p = -\frac{y}{z}$

Perspective Normalization of View Volume



Finally, select α, β such that:

$z_{min} \rightarrow z'' = -1 \qquad z_{max} \rightarrow z'' = 1$

very finally, if the frustum is not symmetrical, we first need a shear. . .

Summary

- For viewing, we need:
  - ⋆ camera position
  - ⋆ projection
  - ⋆ clipping volume
- OpenGL does most of that conveniently for us
- projection matrices implement it in the rendering pipeline
- Next week: Shading, adding light to the scene