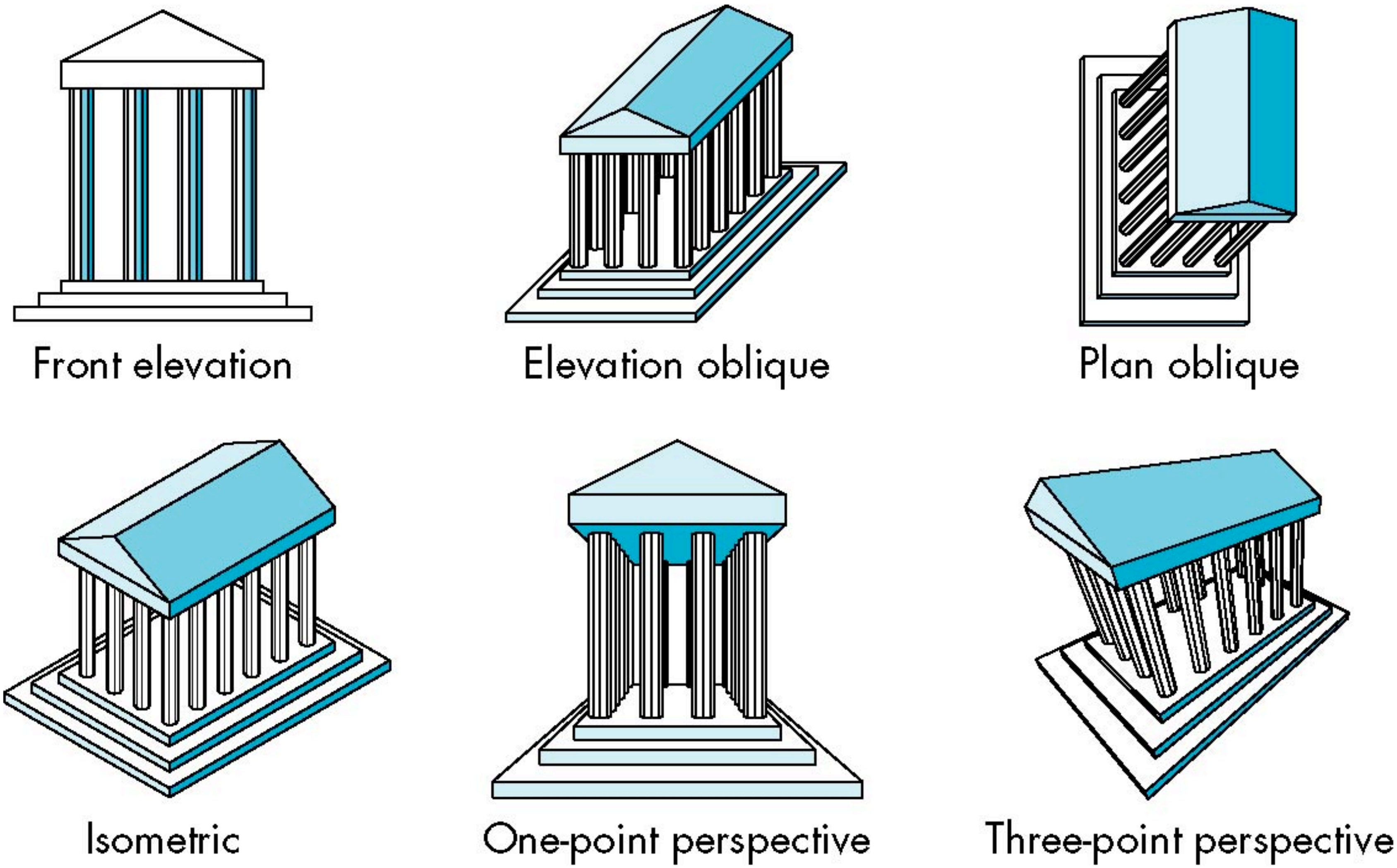
# Supplement to Lecture 8

## Viewing/Projections in OpenGL

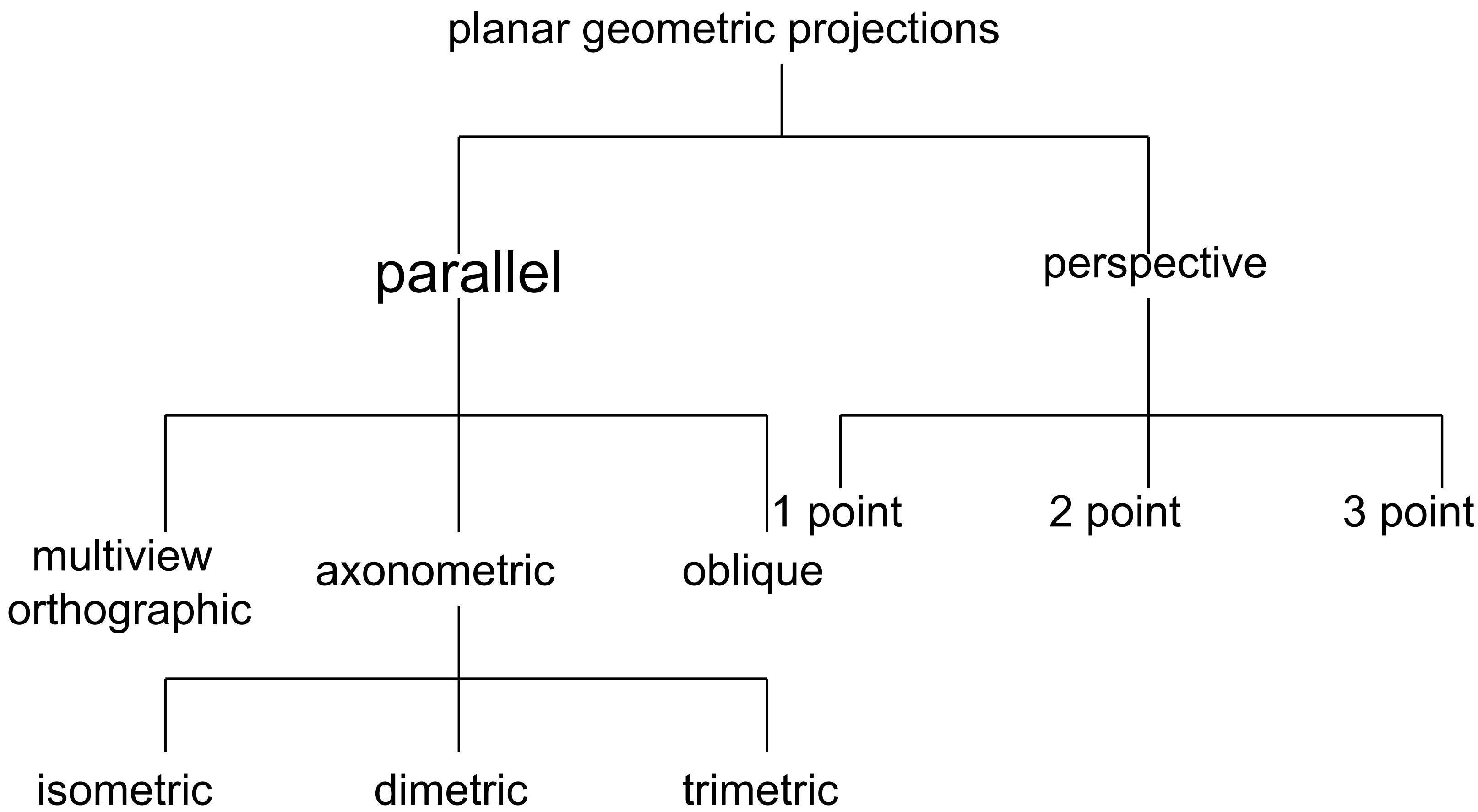
# Classical Projections



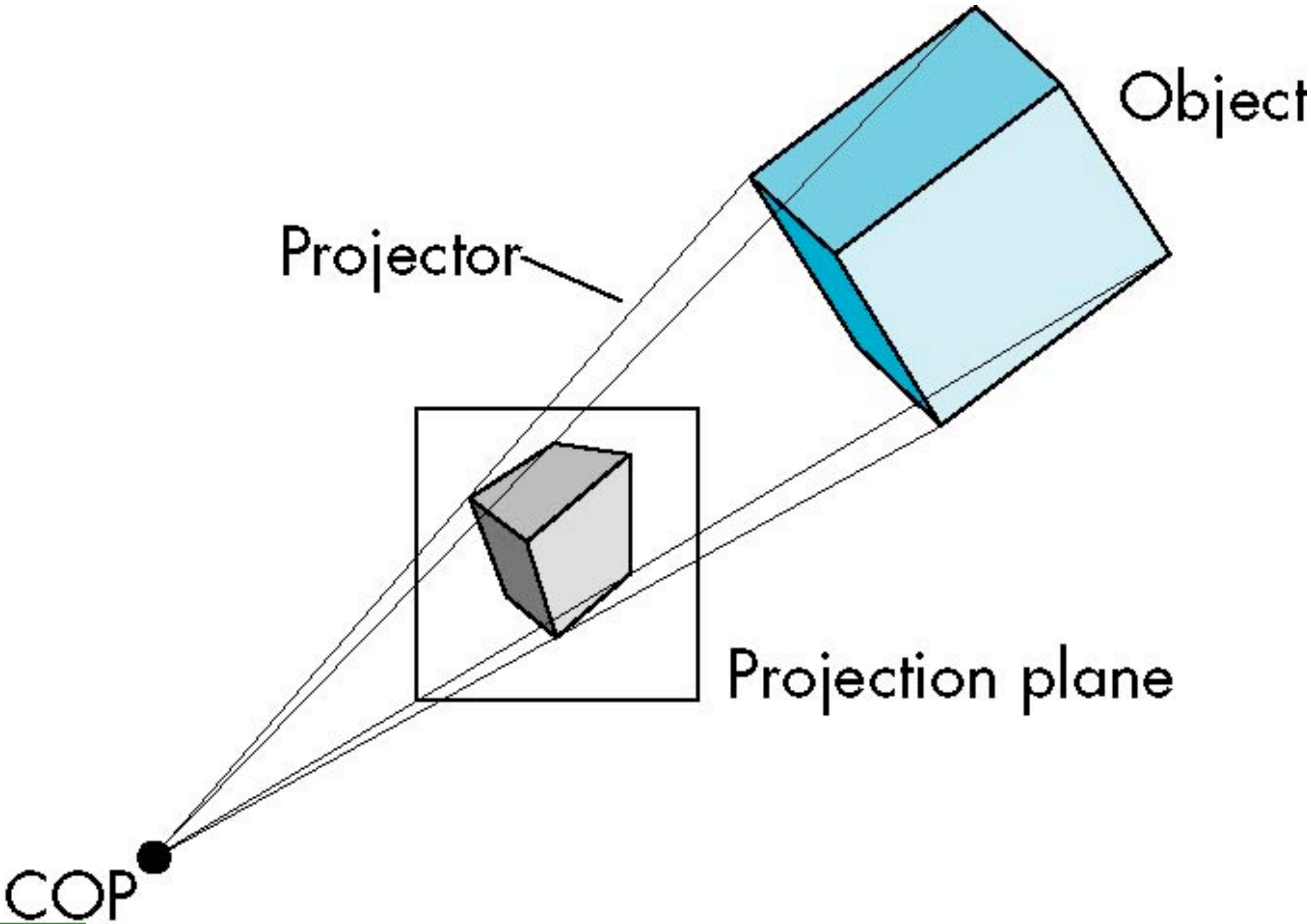
# Perspective vs Parallel

- Computer graphics treats all projections the same and implements them with a single pipeline
- Classical viewing developed different techniques for drawing each type of projection
- Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing

# Taxonomy of Planar Projections



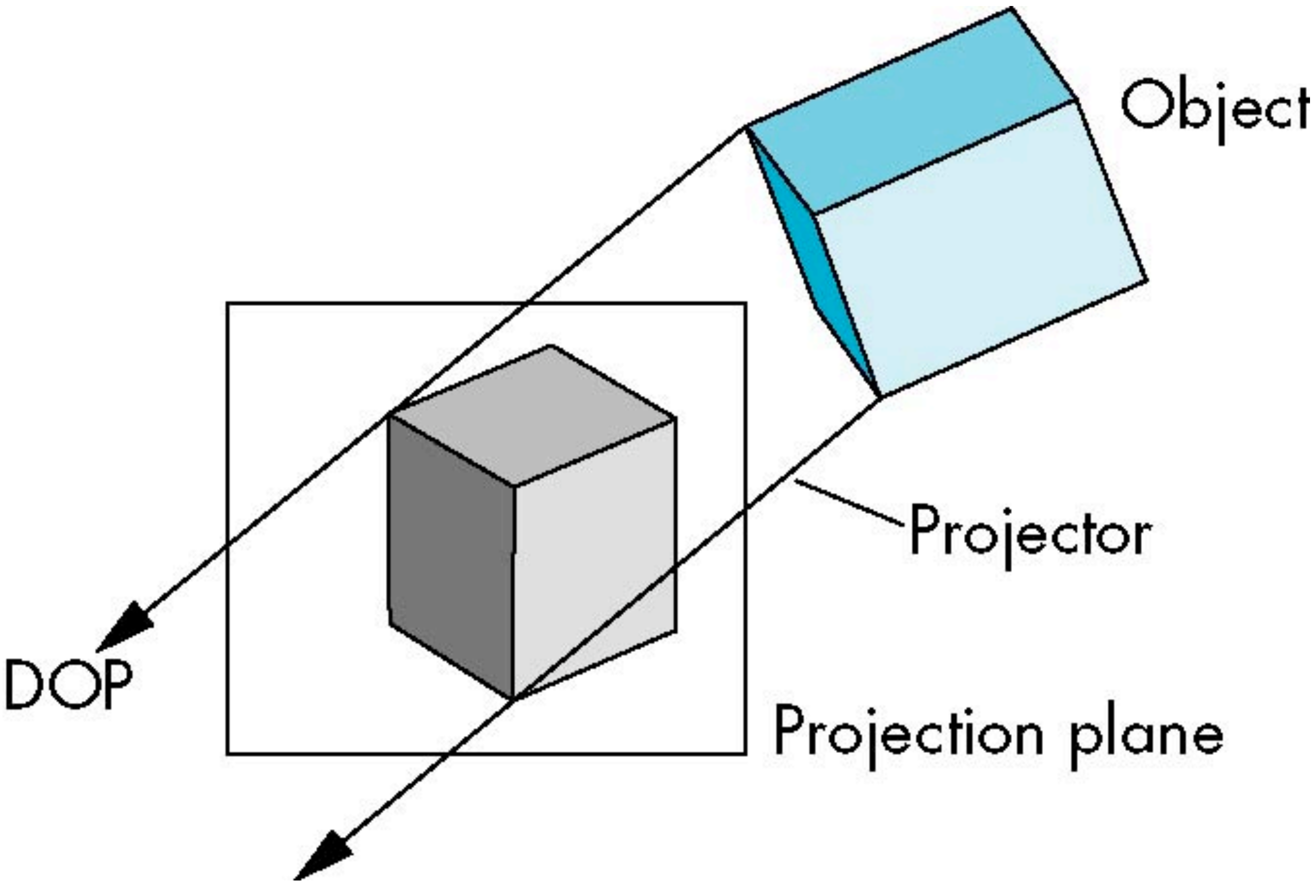
# Perspective Projection





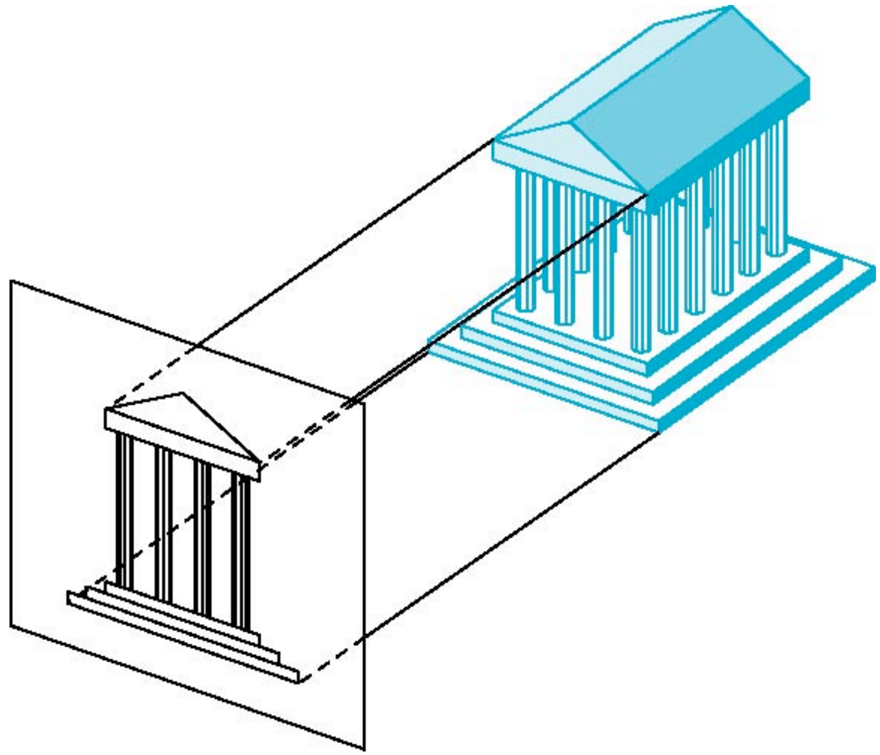

Notes and figures from *Ed Angel, D. Shreiner: Interactive Computer Graphics, 6th Ed., 2012 © Addison Wesley*

# Parallel Projection



# Orthographic Projection

Projectors are orthogonal to projection surface



# Multiview Orthographic Projection

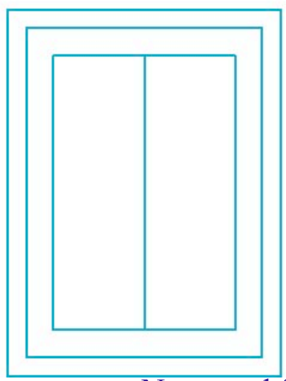
- Projection plane parallel to principal face
- Usually form front, top, side views

isometric (not multiview orthographic view)

front

in CAD and architecture, we often display three multiviews plus isometric

top



side

# Plus & Minus

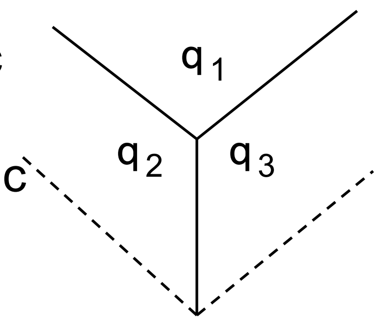
- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

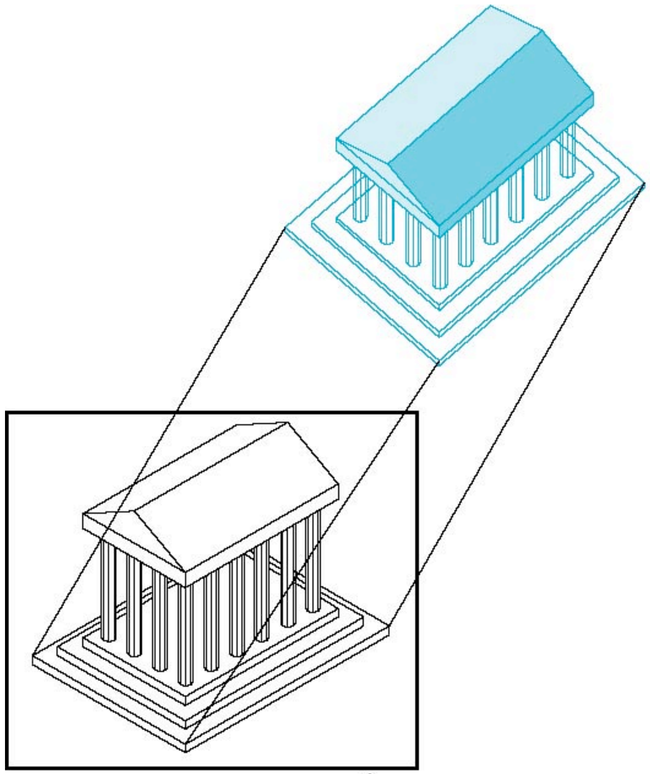
# Axonometric Projections (AP)

Allow projection plane to move relative to object

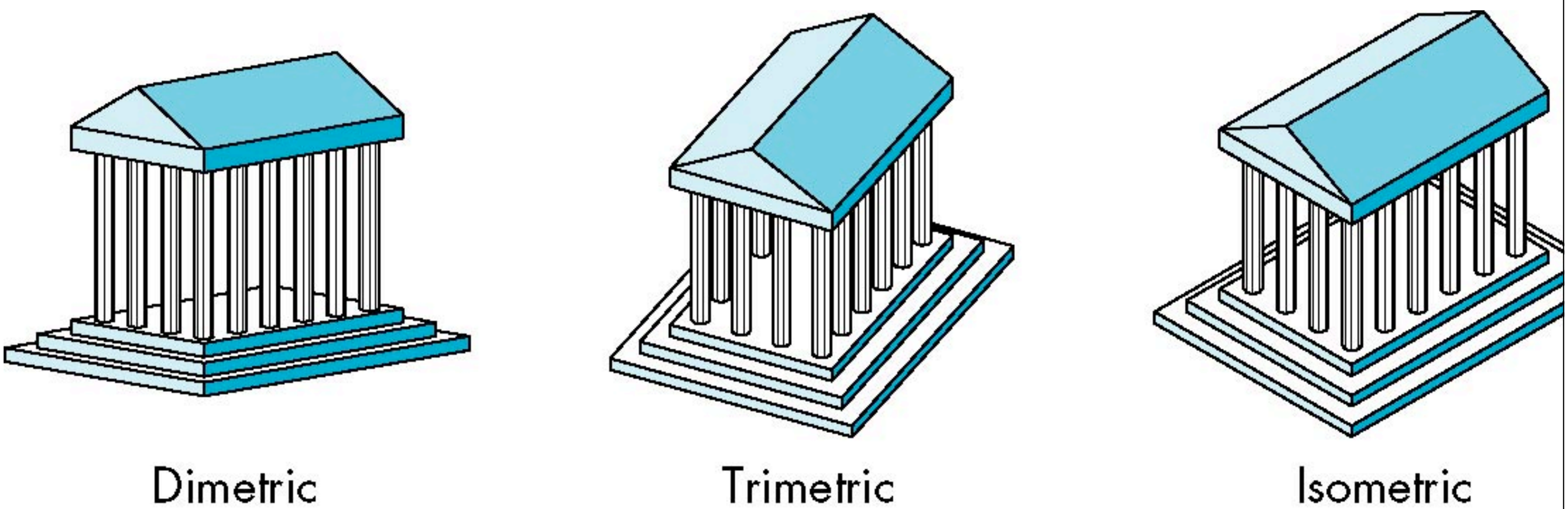
classify by how many angles of a corner of a projected cube are the same

none: trimetric
two: dimetric
three: isometric



# Types of AP

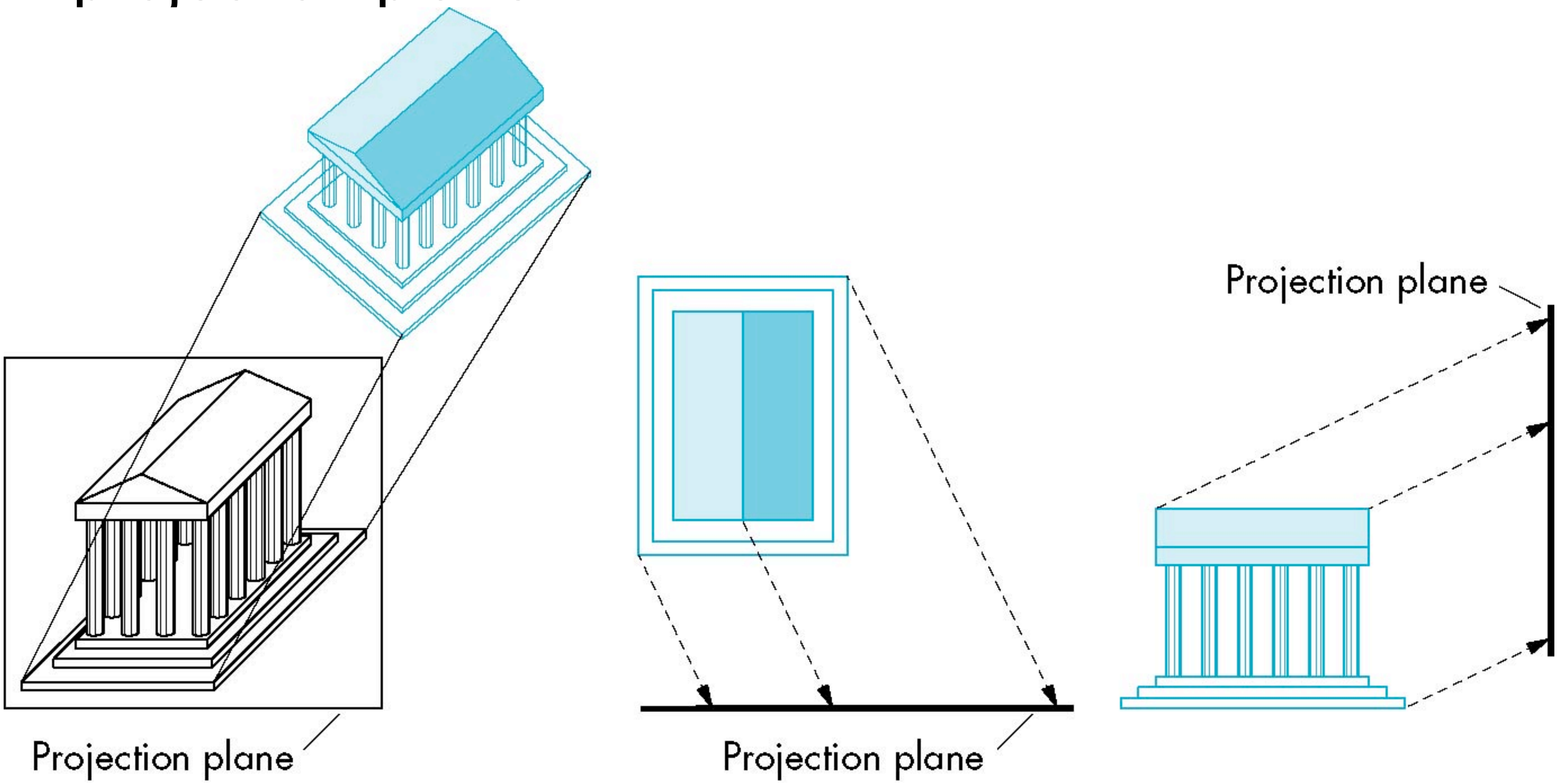


# Plus & Minus

- Lines are scaled (*foreshortened*) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse
- Can see three principal faces of a box-like object
- Some optical illusions possible
  - Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications

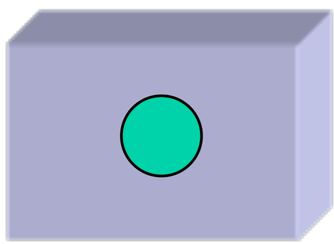
# Oblique Projection

Arbitrary relationship between projectors and projection plane



# Plus & Minus

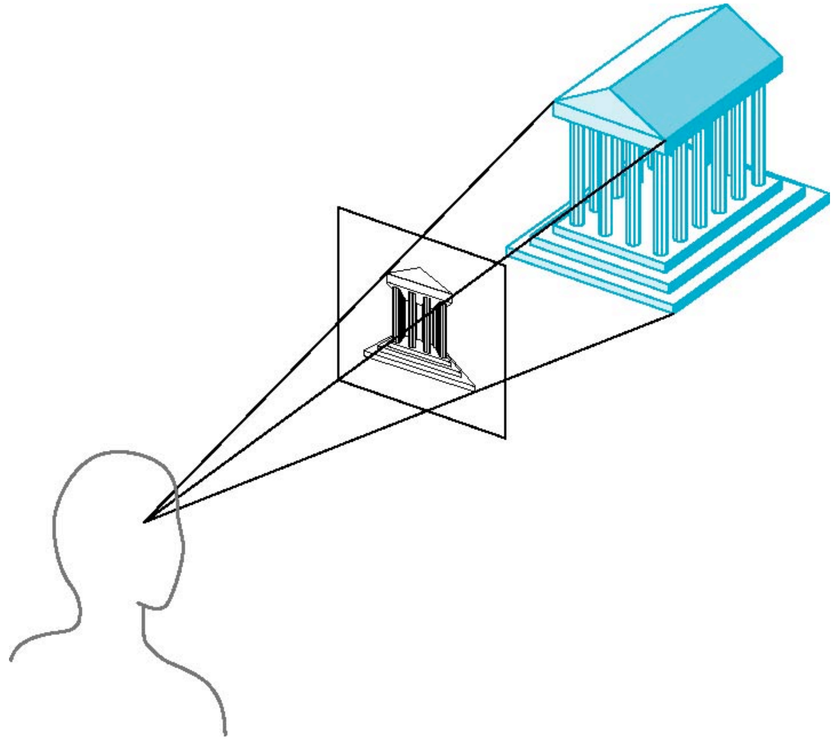
- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique
- Angles in faces parallel to projection plane are preserved while we can still see "around" side



- In physical world, cannot create with simple camera; possible with bellows camera or special lens (architectural)

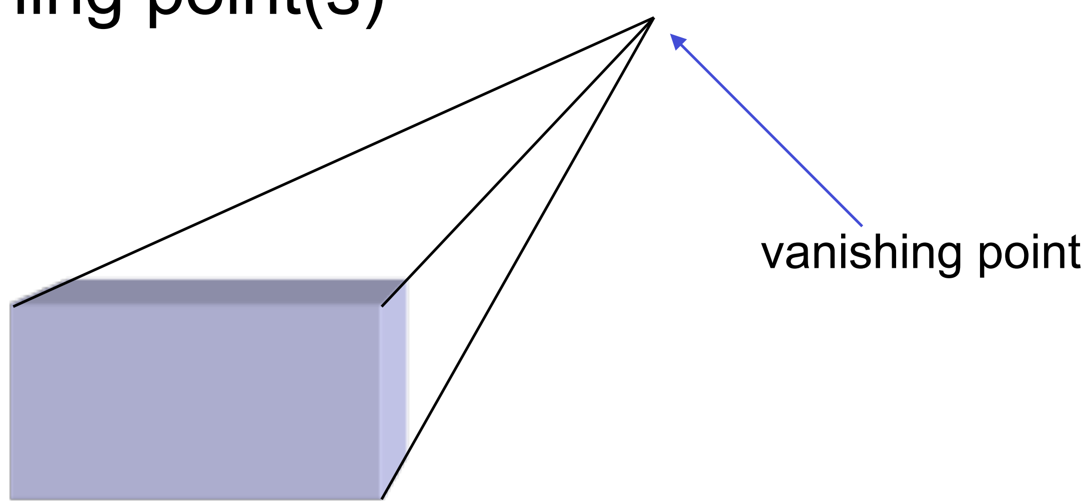
# Perspective Projections

Projectors coverge at center of projection



# Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)



# Three Point Perspective

- No principal face parallel to projection plane
- Three vanishing points for cube

# Two Point Perspective

- On principal direction parallel to projection plane
- Two vanishing points for cube

# One Point Perspective

- One principal face parallel to projection plane
- One vanishing point for cube

# Plus & Minus

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)

# Computer Viewing in OpenGL

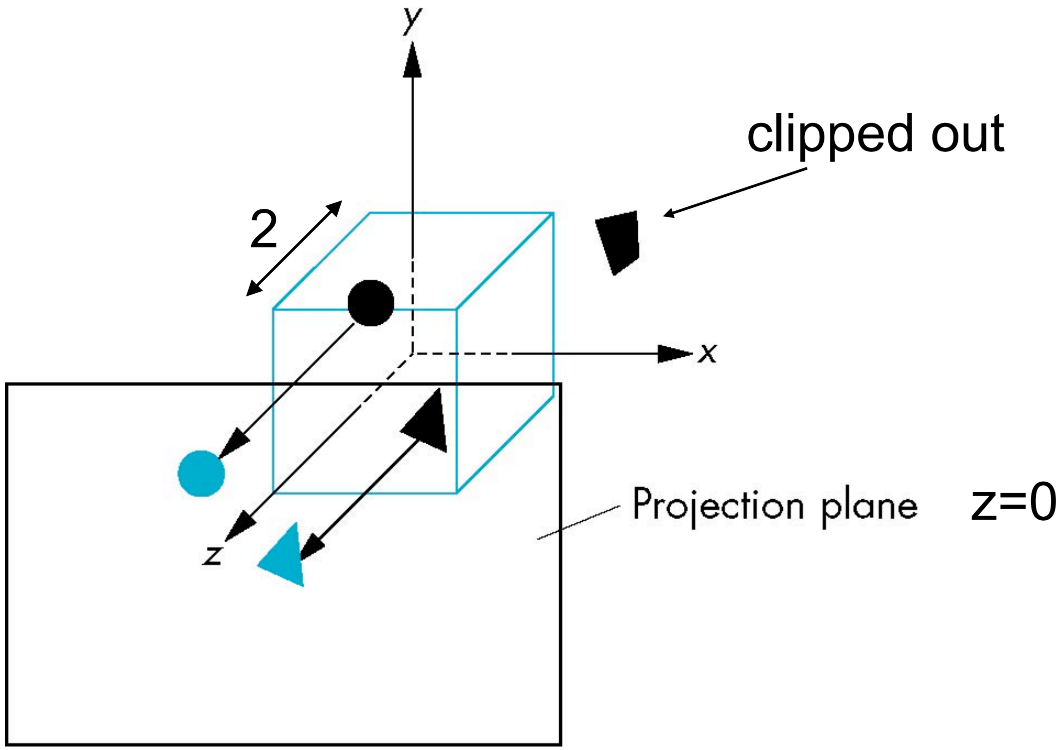
- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

# 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 Camera Frame

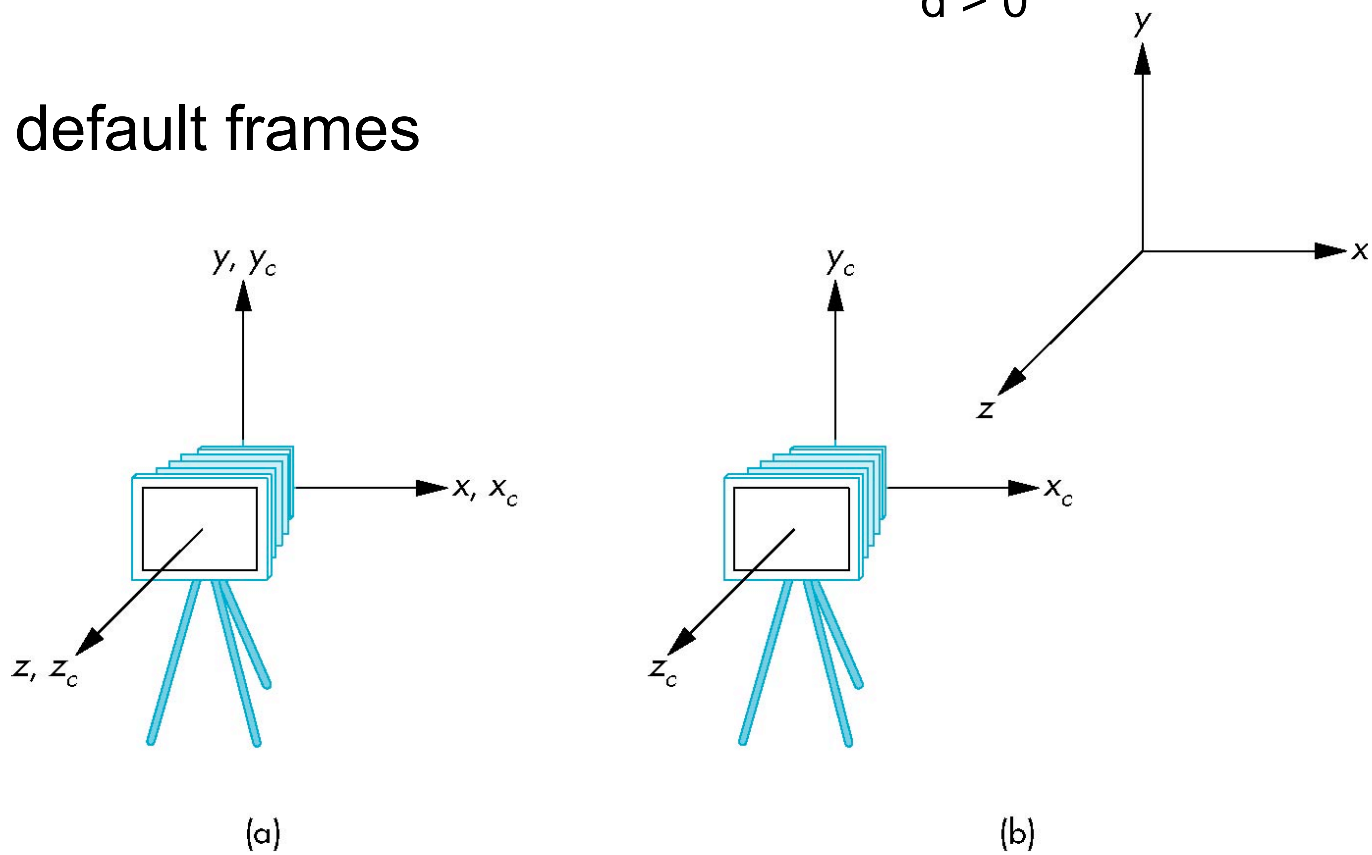
- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined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

  `-d > 0`

# Moving Camera Back from Origin

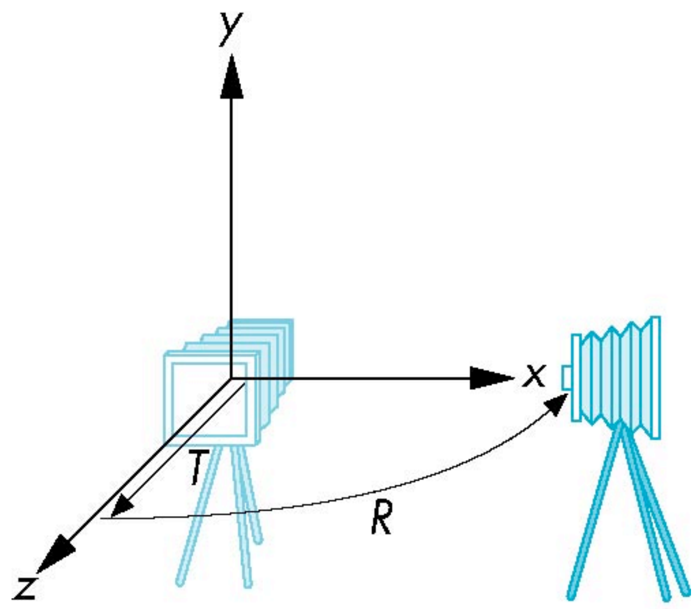
frames after translation by –d
d > 0

default frames



# Moving 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);
```

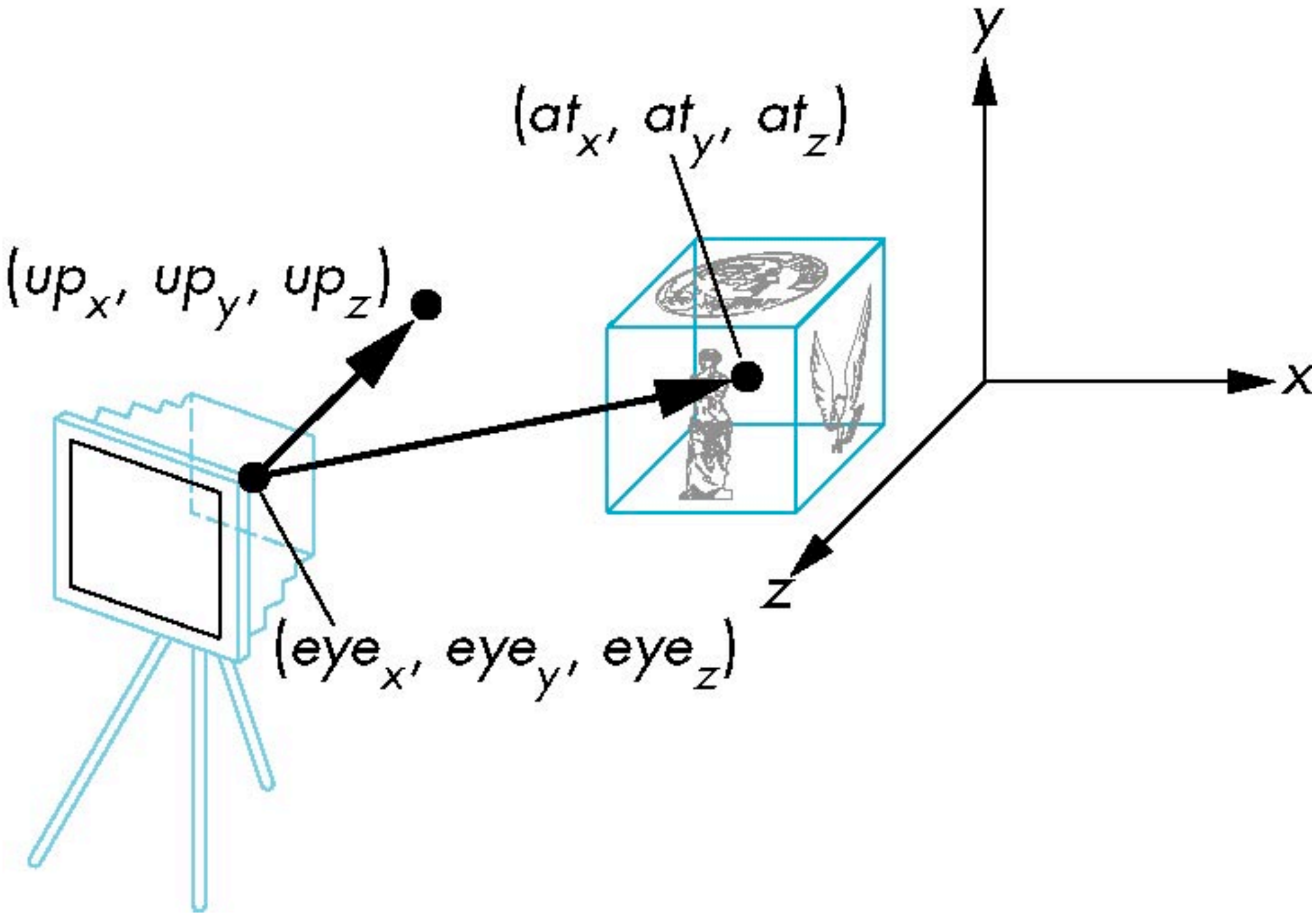
# The LookAt Function

- 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., 1.0. 0.0);
```

# gluLookAt

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



# Other Viewing API's

- 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
  - Elevation, azimuth, twist
  - 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

# 4x4 ModelView Matrix

default orthographic projection

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

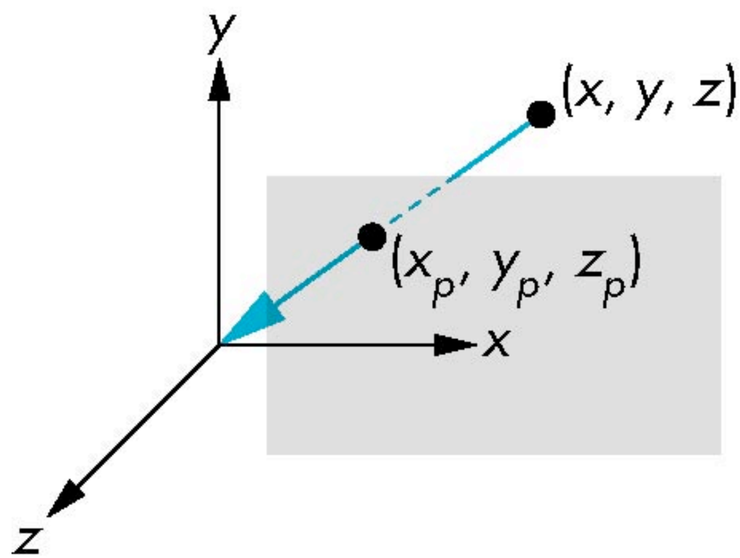
$$\mathbf{p}_p = \mathbf{Mp}$$

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

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

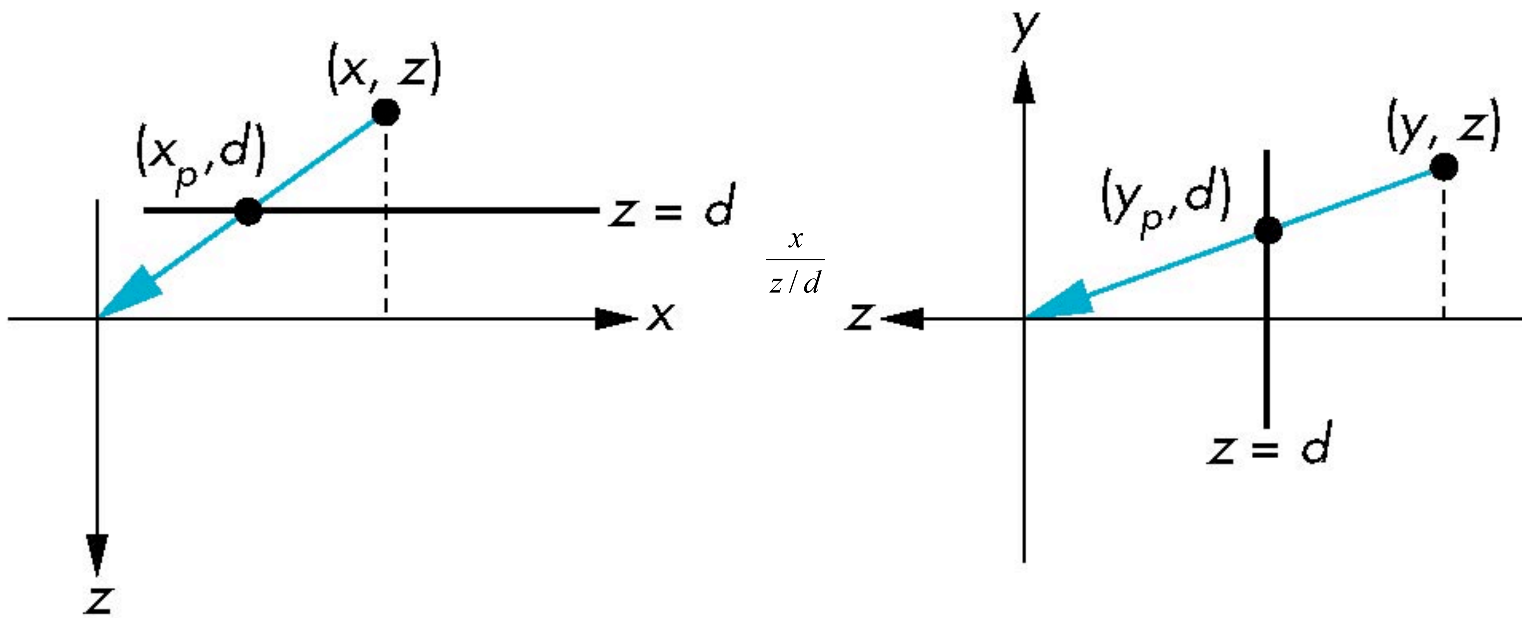
# Simple Perspective

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



# Perspective Equations

Consider top and side views




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

# 4x4 ModelView Matrix

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

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

# Perspective Division

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

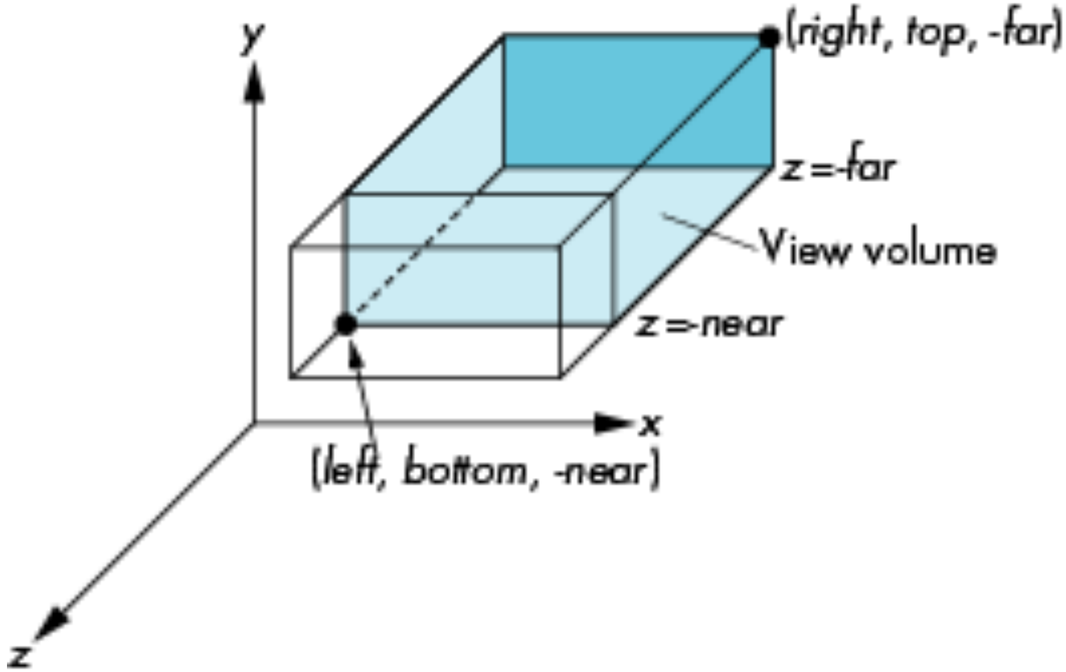
$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

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

# OpenGL Orthogonal Viewing

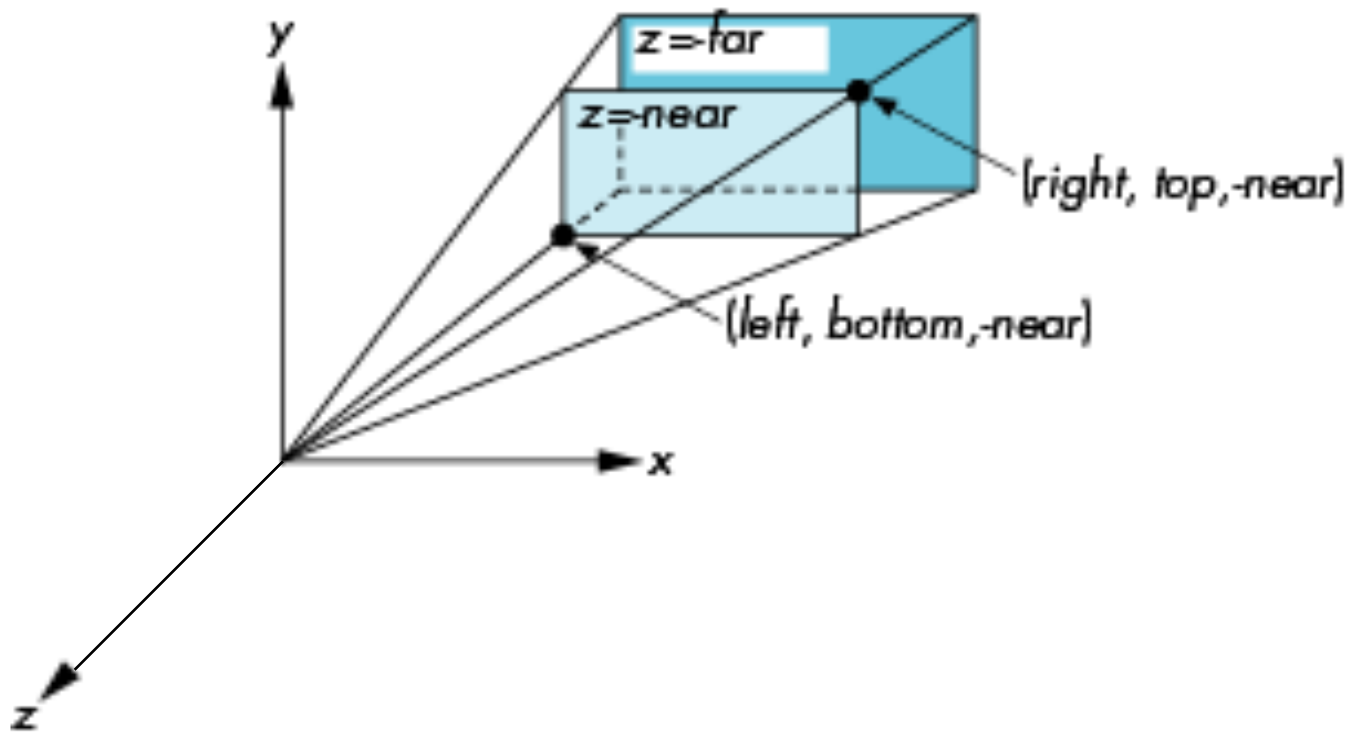
**`glOrtho(left,right,bottom,top,near,far)`**




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

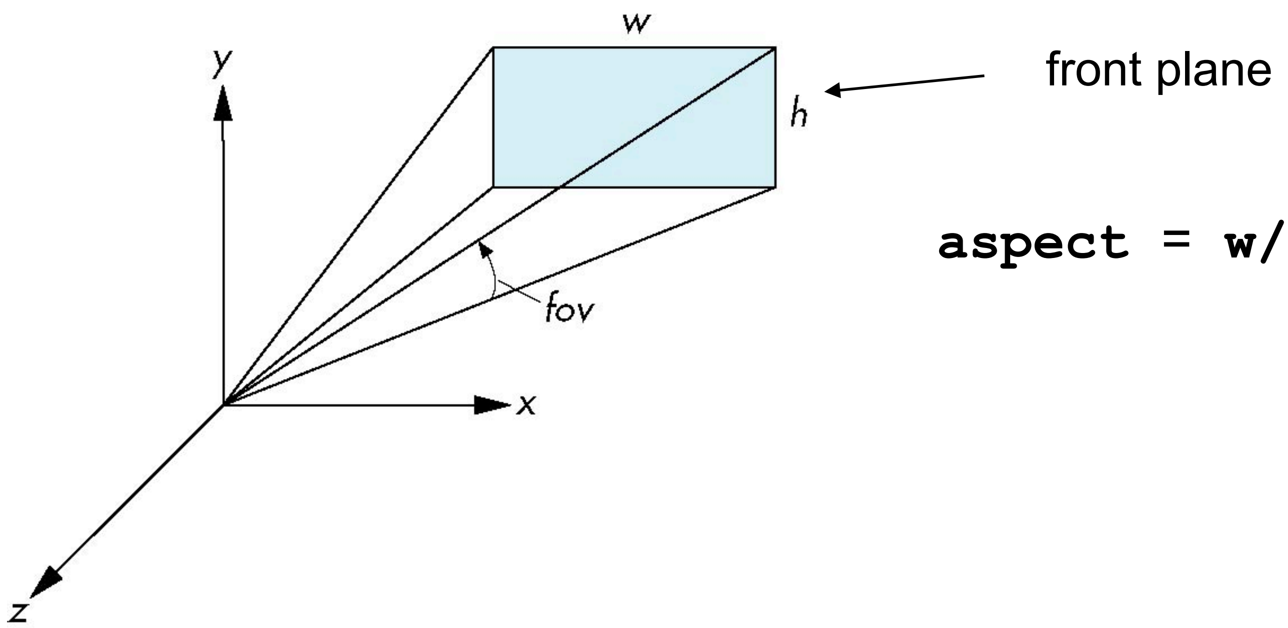
# OpenGL Perspective

**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




**`aspect = w/h`**