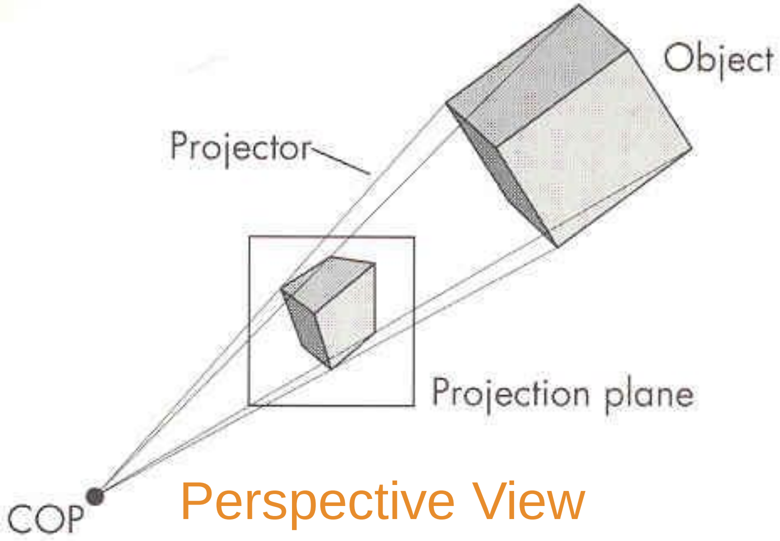
Viewing

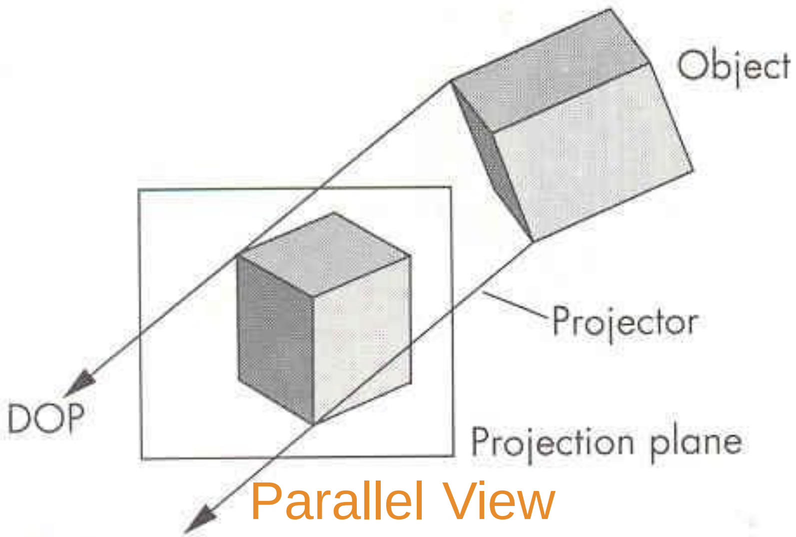
7TH WEEK, 2021

Fundamental Types of Viewing

- Perspective views
  - Finite <u>COP</u> (center of projection)
- Parallel views
  - COP at infinity
  - <u>DOP</u> (direction of projection)

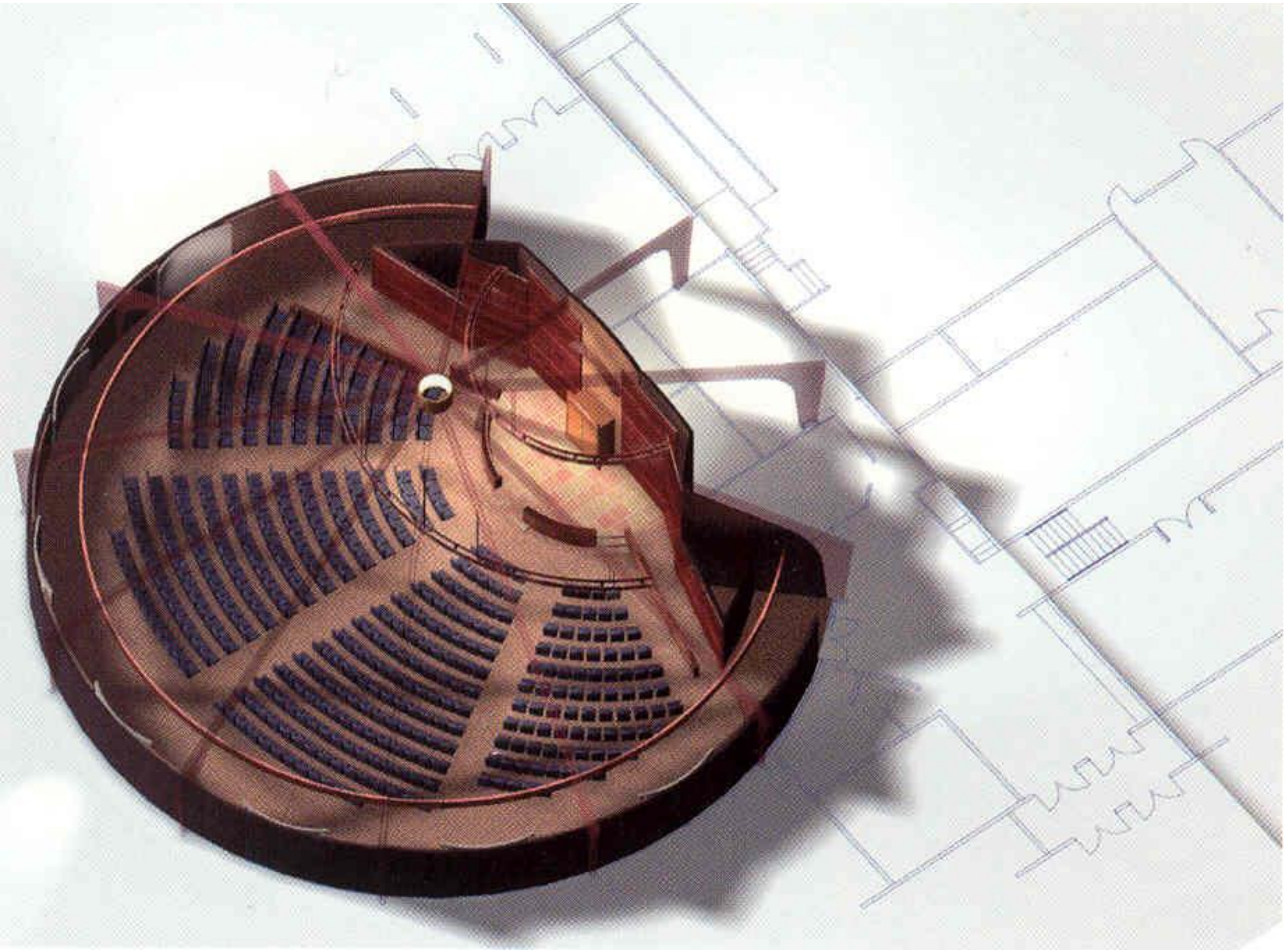


Perspective View



Parallel View

Parallel View



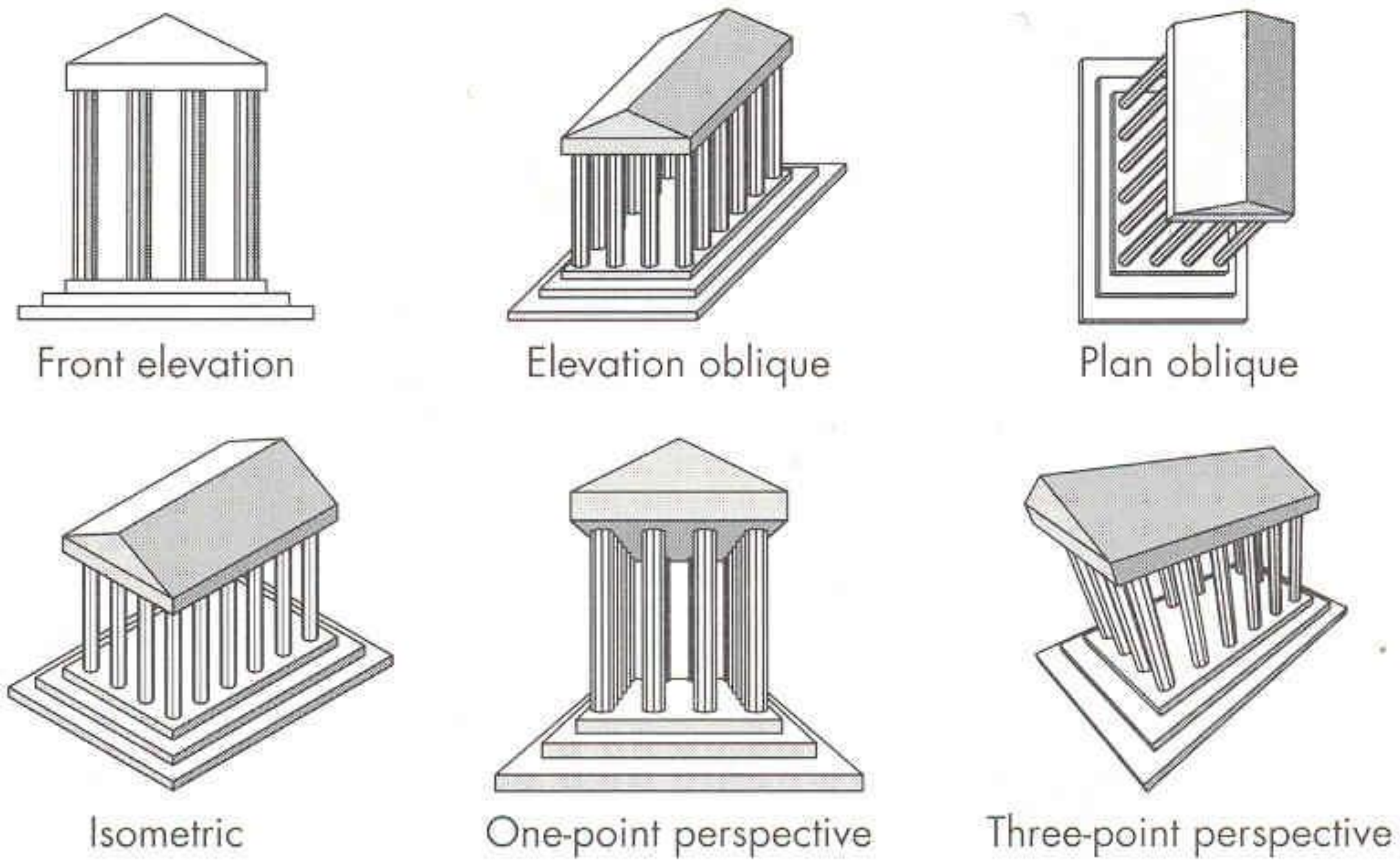
Perspective View

Classical Viewing

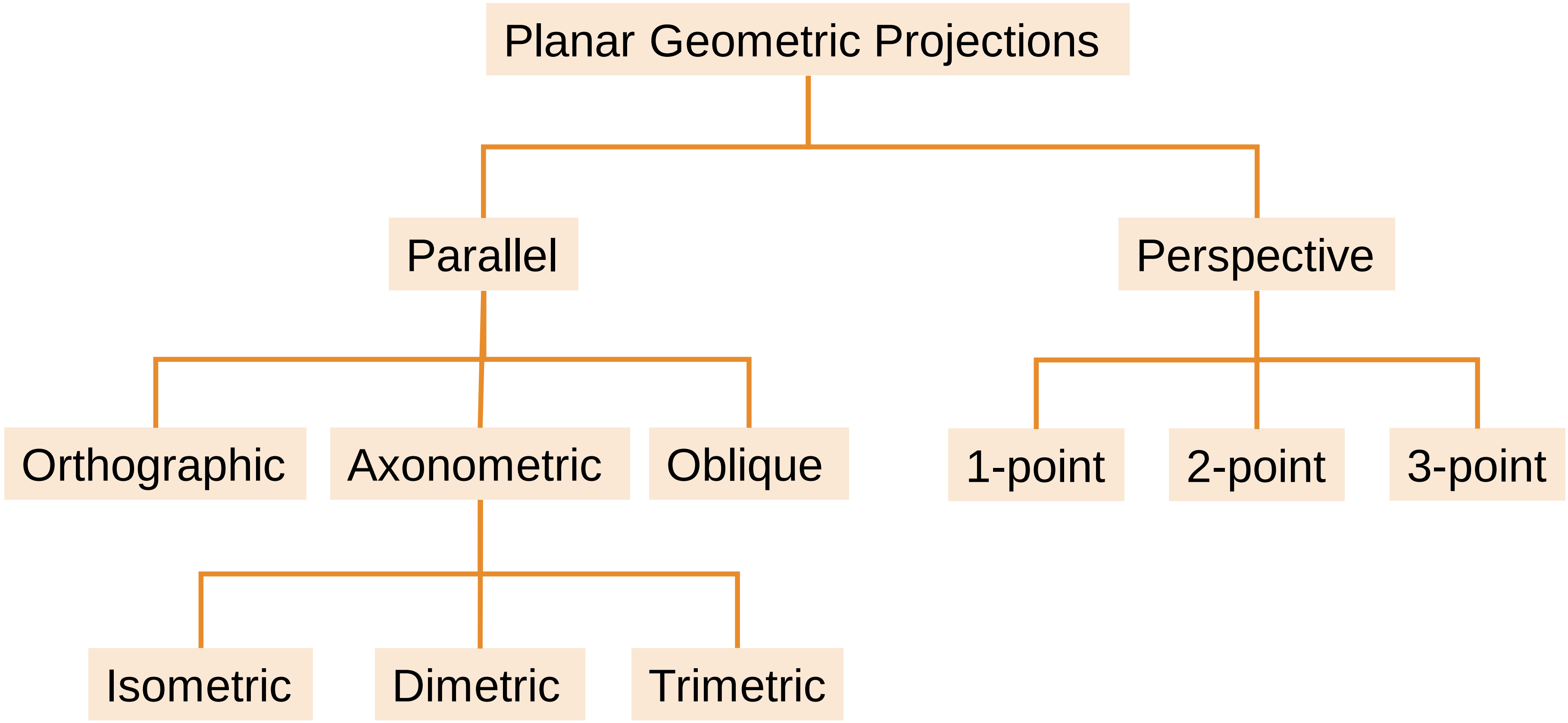
- Viewing requires three basic elements
  - One or more <u>objects</u>
  - A <u>viewer</u> with a projection plane
  - <u>Projectors</u> that go from the object(s) to the projection plane

- Classical views are based on the relationship among three elements
  - Specific relationship between the objects and the viewer

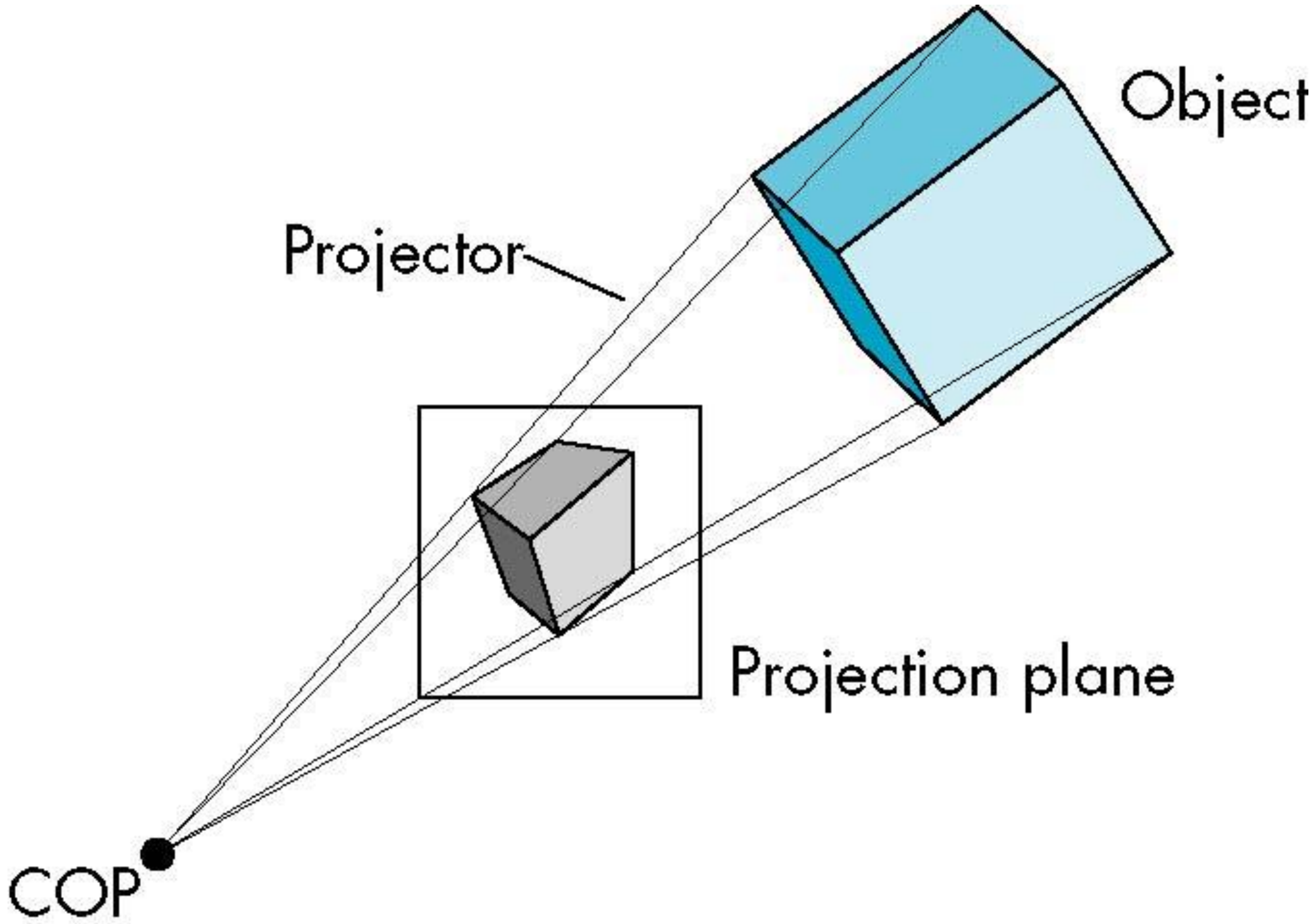
Classical Viewing



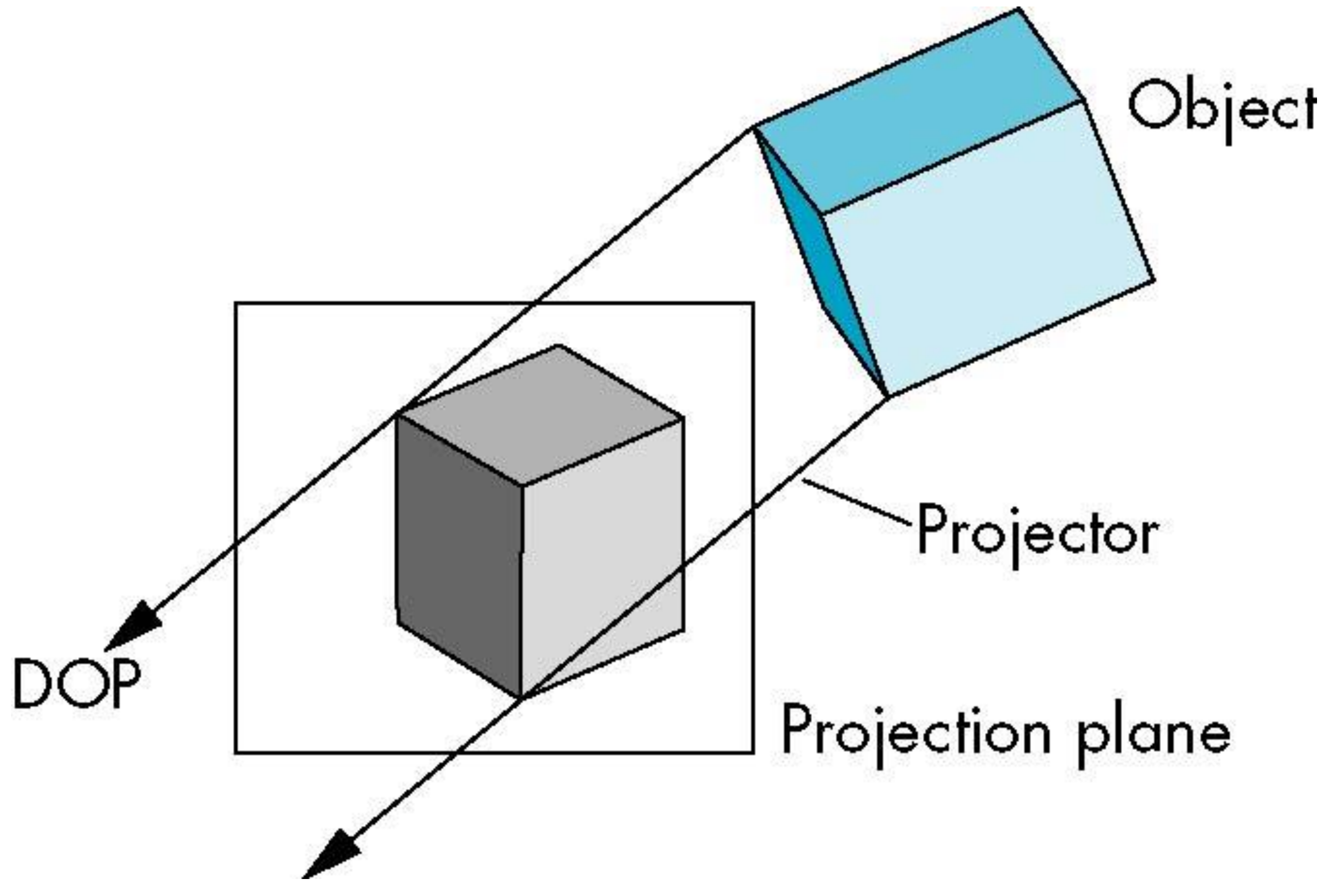
Taxonomy of Planar Geometric Projections



Perspective Projection

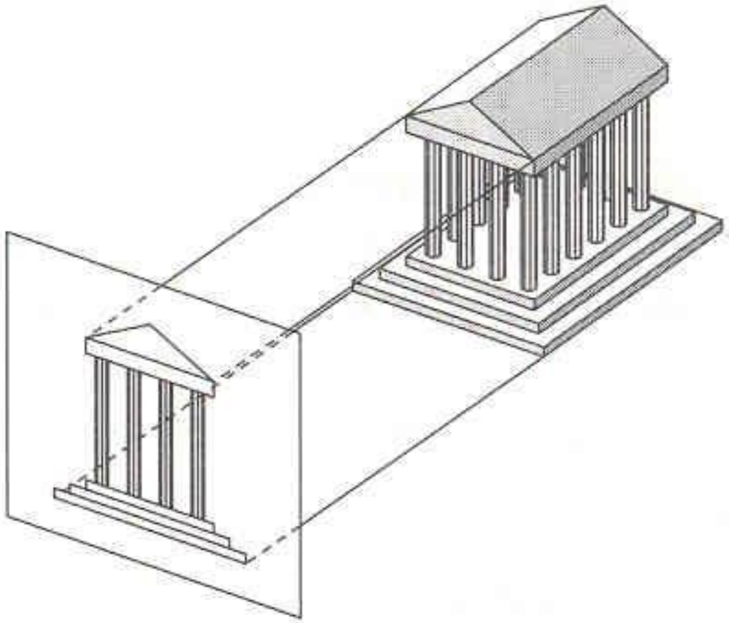


Parallel Projection

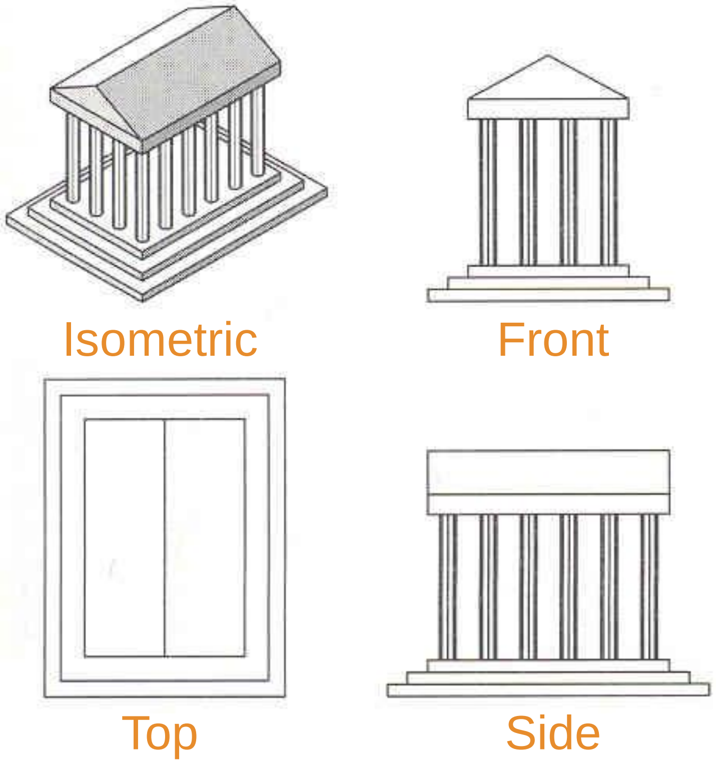


Multiview Orthographic Projections

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



Orthographic Projections



Temple and Three Multi-view Orthographic Projections

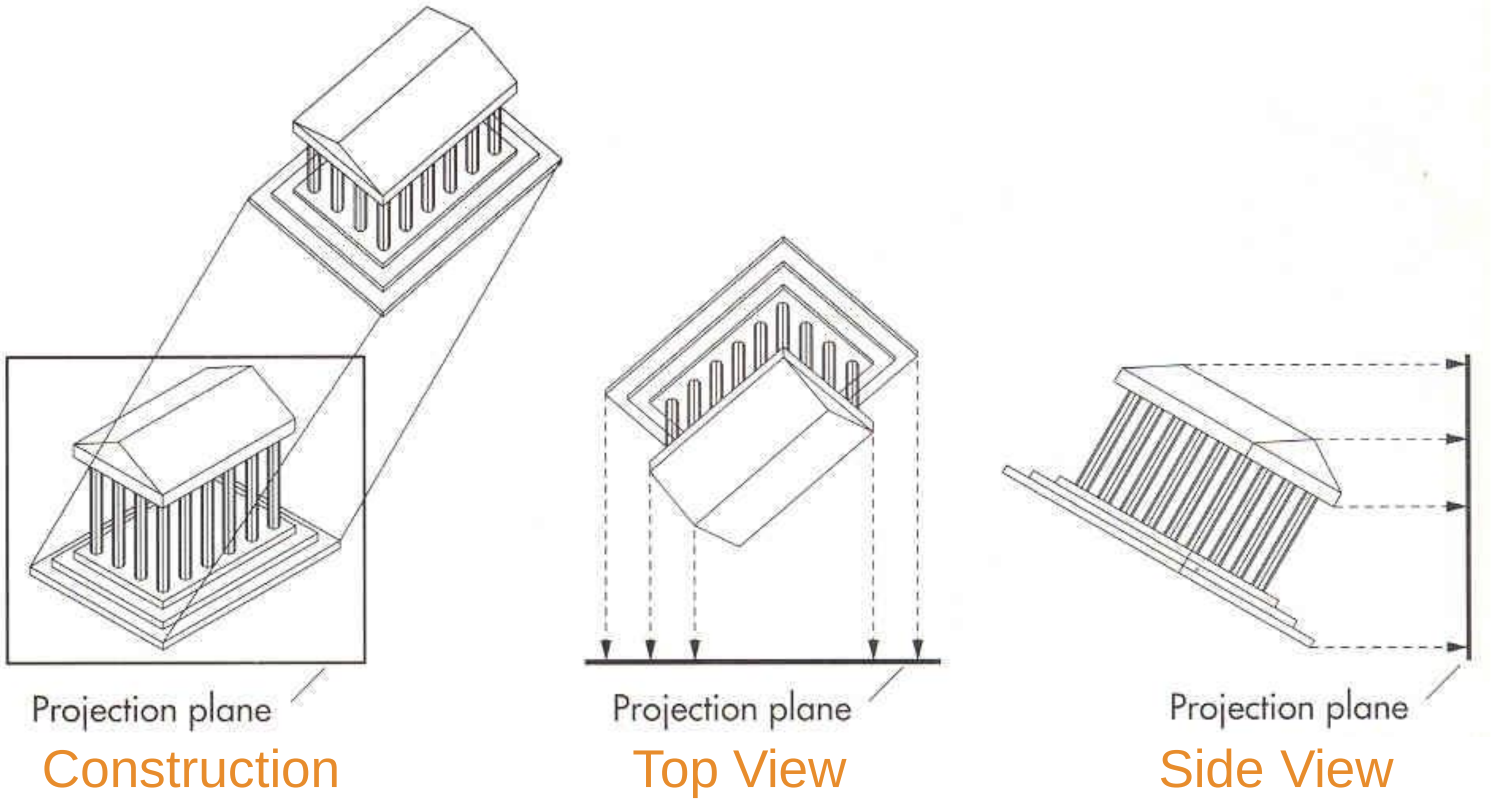
Advantages and Disadvantages

- Preserving 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

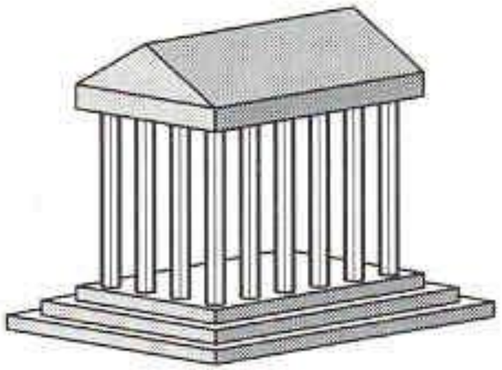
Axonometric Projections

- Projection plane can have any orientation with respect to the object
  - Projectors are still orthogonal to the projection planes

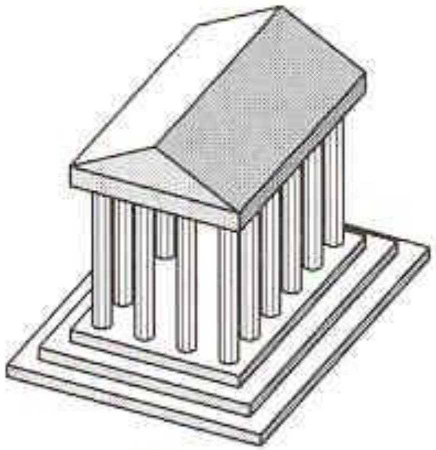


Axonometric Projections

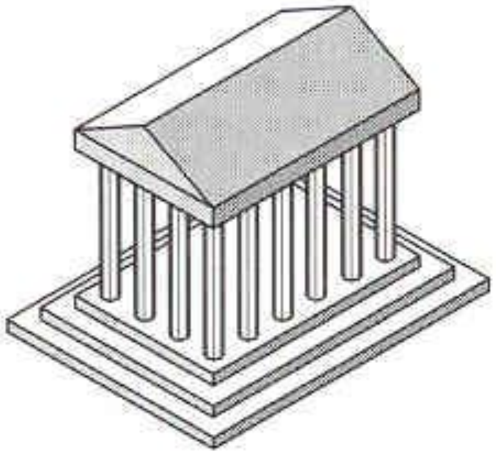
- Preserving parallel lines but not angles
  - Isometric – projection plane is placed symmetrically with respect to the three principal faces
  - Dimetric – two of principal faces
  - Trimetric – general case



Dimetric



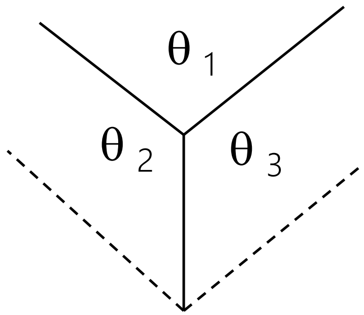
Trimetric



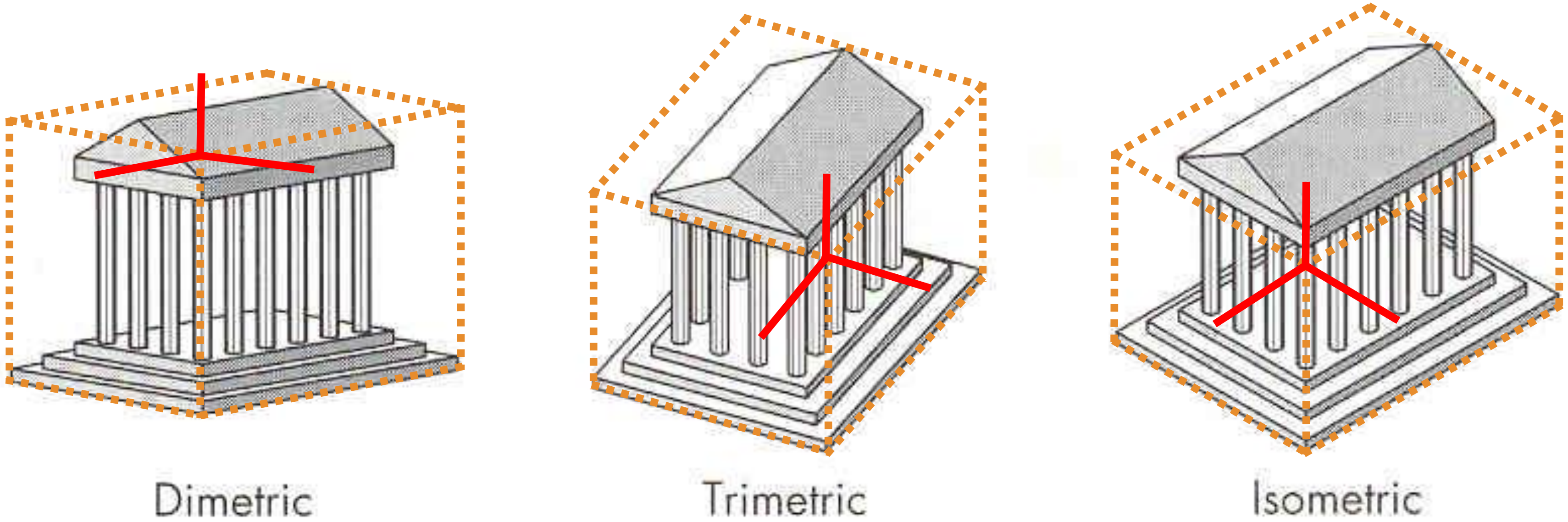
Isometric

Axonometric Projections

- Preserving parallel lines but not angles



How many angles of a corner of a projected cube are the same?



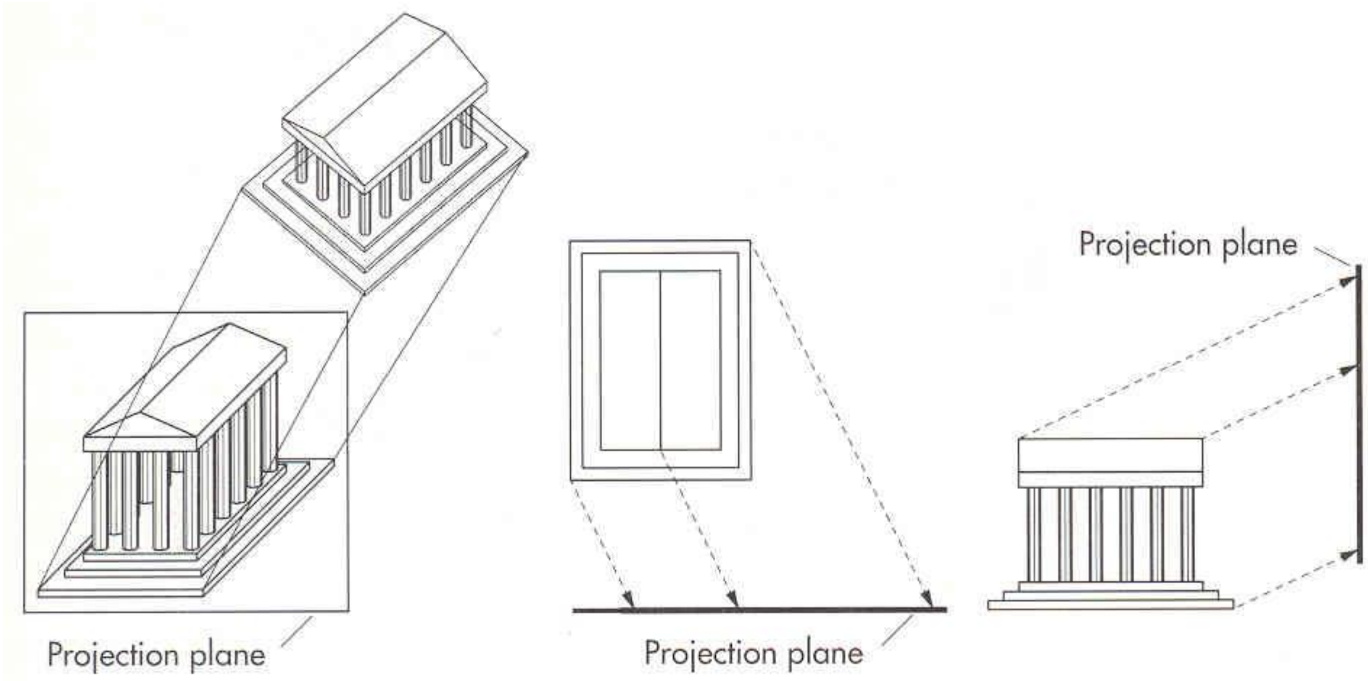
Two

None

Three

Oblique Projections

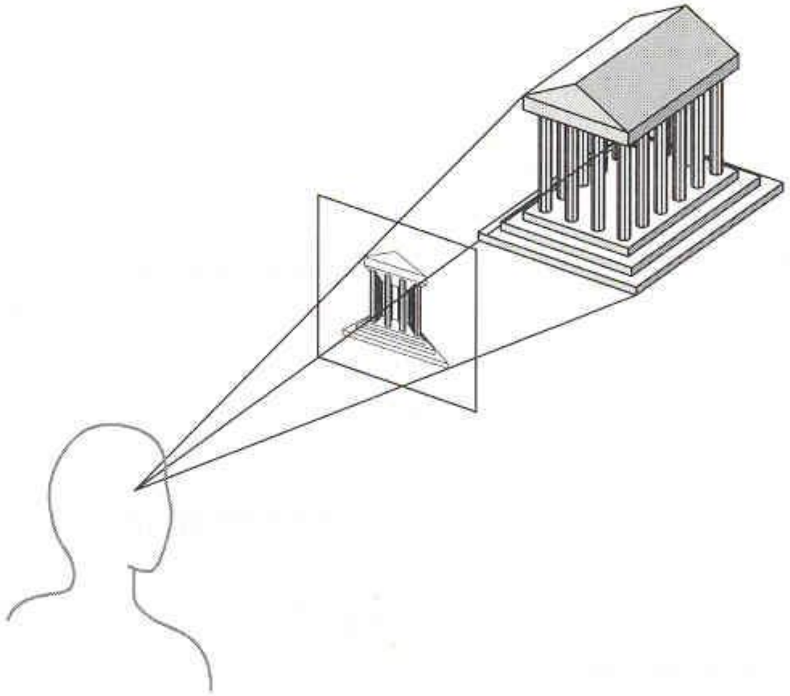
- Projectors can make an arbitrary angle with the projection plane
  - Preserving angels in planes parallel to the projection plane



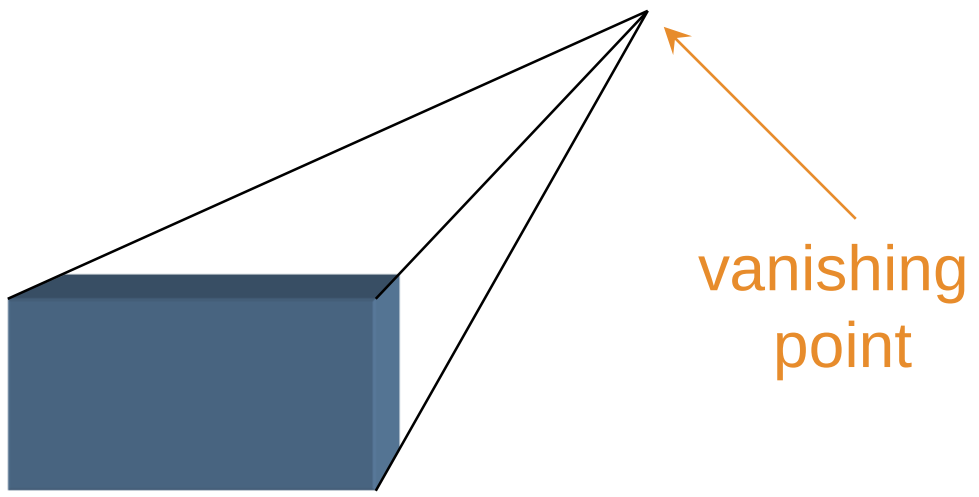
Construction Top View Side View

Perspective Projections

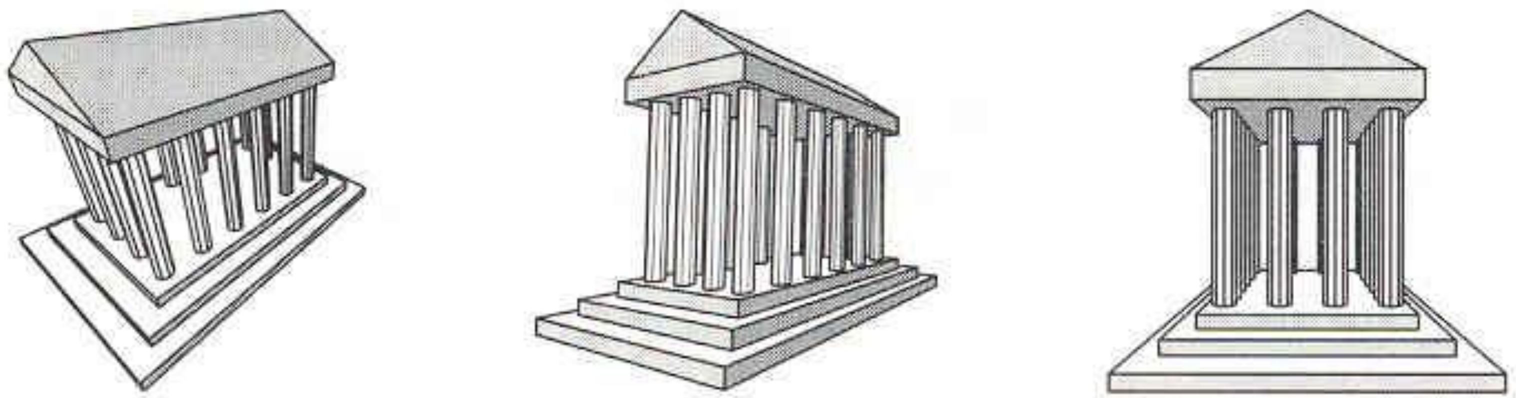
- Projectors converge at the center of projection



Perspective Projections



- One-, two-, and three-point perspectives
  - **Vanishing points** – parallel lines (not parallel to the projection plane) on the object converge at a single point in the projection



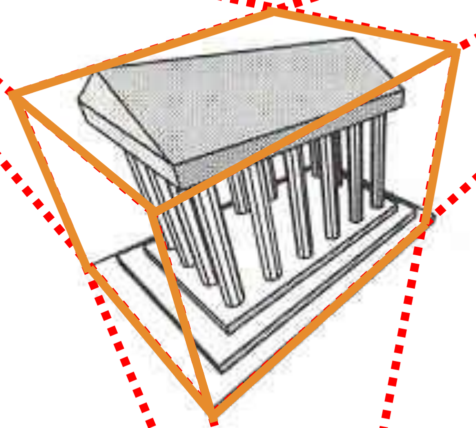
Three-Point Perspective

Two-Point Perspective

One-Point Perspective

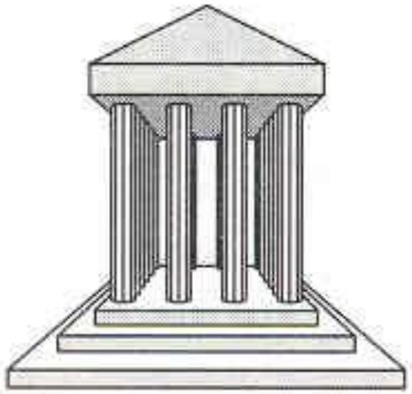
Perspective Projections

- One-, two-, and three-point perspectives
  - **Vanishing points** – parallel lines (not parallel to the projection plane) on the object converge at a single point in the projection



Three-Point Perspective

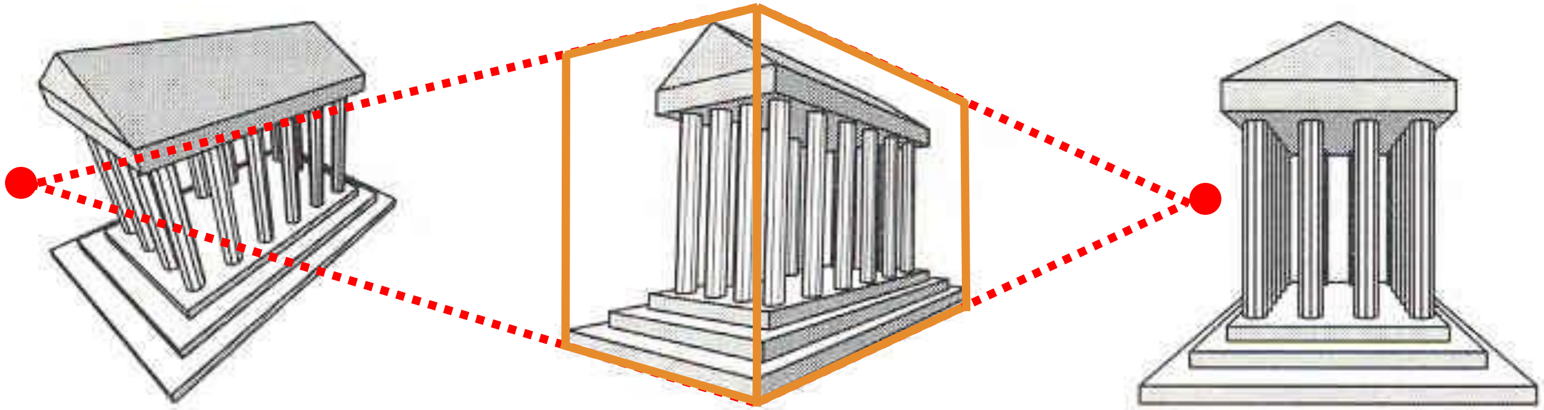
Two-Point Perspective



One-Point Perspective

Perspective Projections

- One-, two-, and three-point perspectives
  - **Vanishing points** – parallel lines (not parallel to the projection plane) on the object converge at a single point in the projection



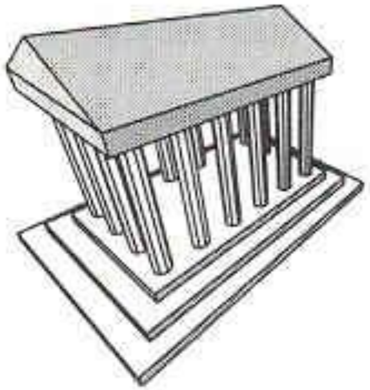
Three-Point Perspective

Two-Point Perspective

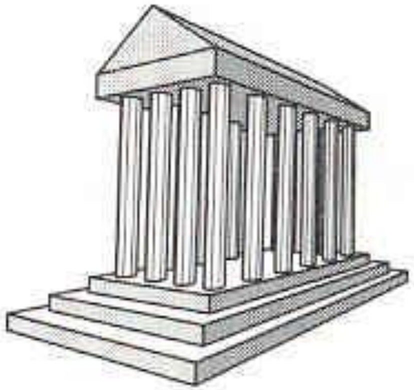
One-Point Perspective

Perspective Projections

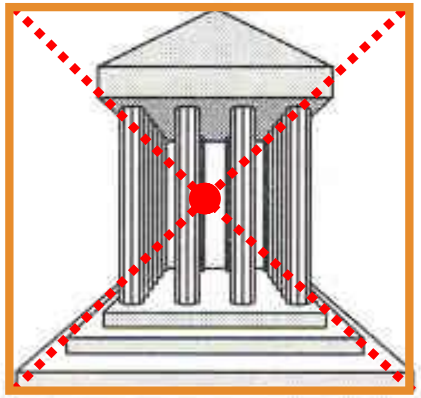
- One-, two-, and three-point perspectives
  - **Vanishing points** – parallel lines (not parallel to the projection plane) on the object converge at a single point in the projection



Three-Point Perspective



Two-Point Perspective



One-Point Perspective

Advantages and Disadvantages

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

Computing Viewing

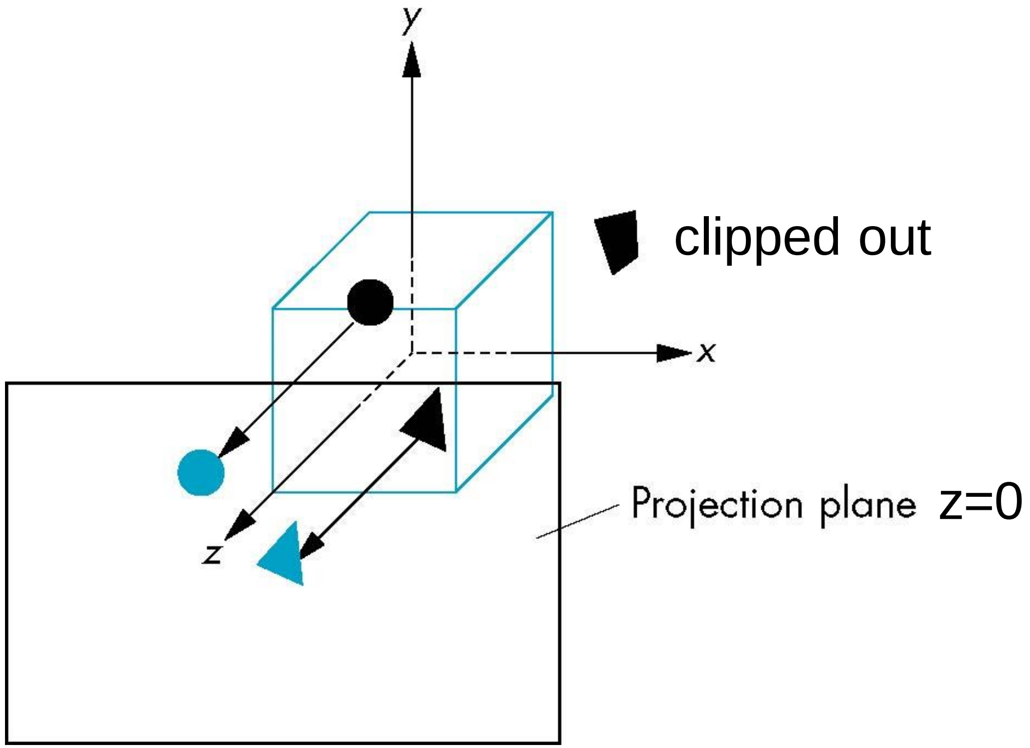
- Three aspects of the viewing process, all of which are implemented in the pipeline
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

The WebGL Camera

- In WebGL, 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
- WebGL also specifies a default view volume
  - A cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

Default Projection

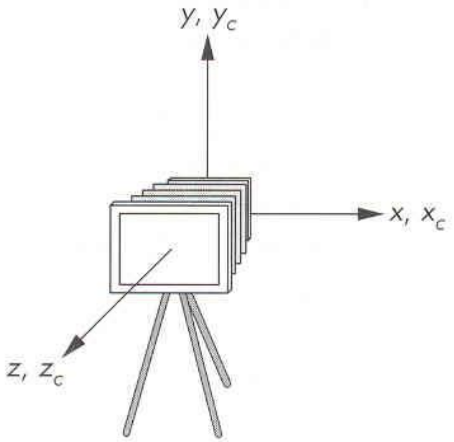
- Default projection is orthographic



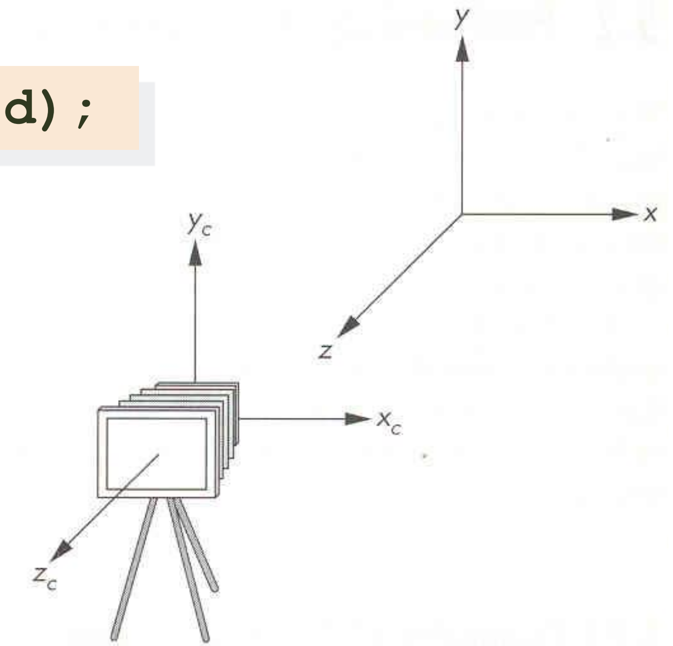
Positioning of the Camera

- Moving the camera away from the objects
  - Translate the camera frame in the positive z-direction
  - Translate the world frame in the negative z-direction

```
Translate(0.0, 0.0, -d);
```



Initial Configuration

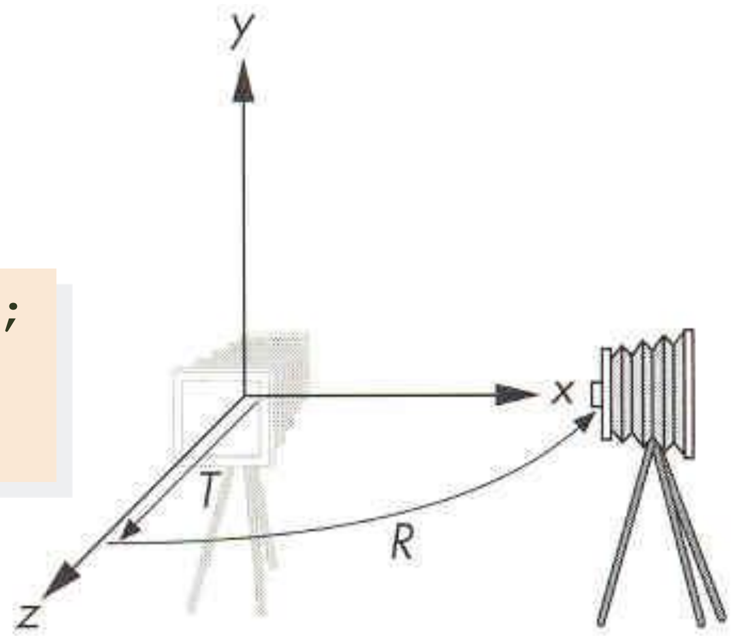


After Change in the Model-View Matrix

Positioning of the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
  - Ex) looking at the same object from the positive x axis
    - Rotate the camera
    - Move it away from the origin
    - Model-view matrix C=TR

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



Look-At Function

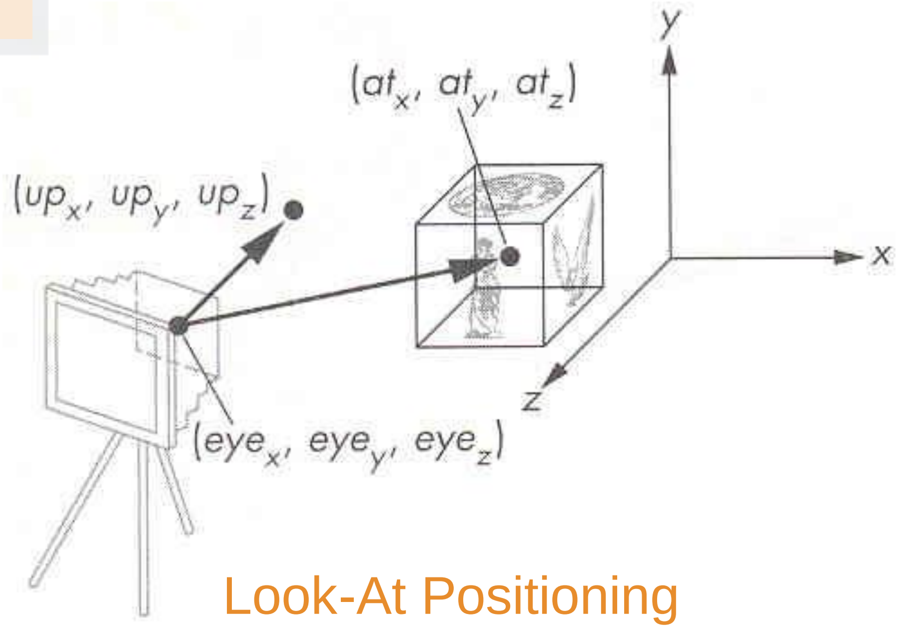
- The GLU library contains the function
  - To form the required <u>model-view</u> matrix
- Replaced by LookAt() in MV.js

```
LookAt(eye, at, up);
```

  - Can concatenate with modeling transformations
  - Ex)

```
var eye = vec3(1.0, 1.0, 1.0);
var at = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);

var mv = LookAt(eye, at, up);
```



Look-At Positioning

Projections and Normalization

- The default projection in the eye (camera) frame

  ➔ Orthographic

- For points within the default view volume:

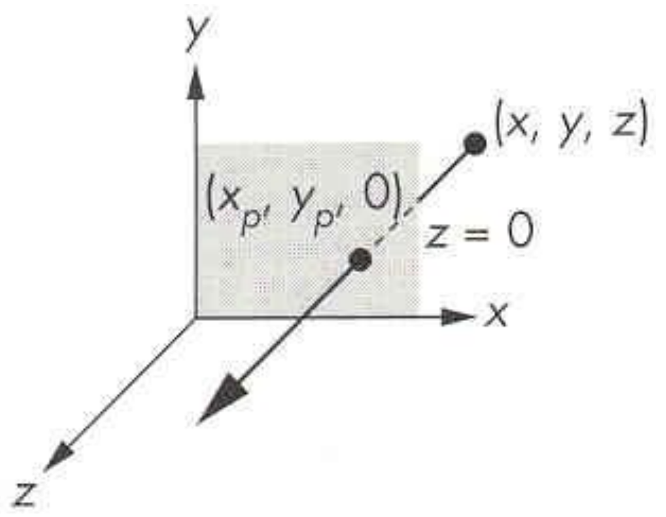
$$x_p = x, \quad y_p = y, \quad z_p = 0$$

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

Simple Orthogonal Projections

- Projectors are perpendicular to the view plane

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

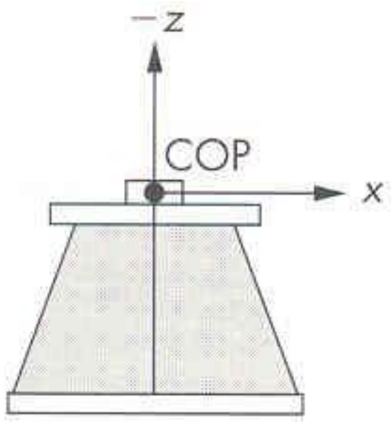


- Orthographic projection matrix

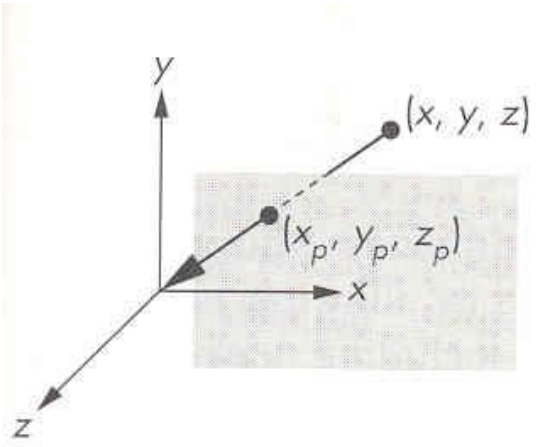
$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

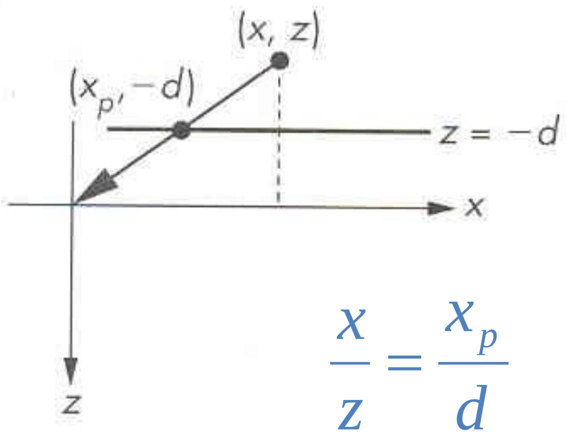
Simple Perspective Projections

- Simple camera
  - Projection plane is orthogonal to z axis
  - Projection plane in front of COP

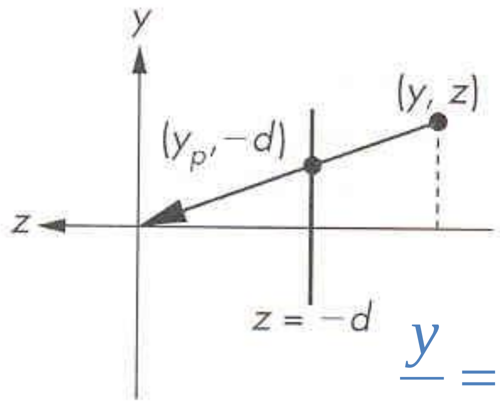


$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d}, \quad z_p = d$$





$$\frac{x}{z} = \frac{x_p}{d}$$



$$\frac{y}{z} = \frac{y_p}{d}$$

Three-Dimensional View

Top View

Side View

Simple Perspective Projections

- Homogeneous coordinates

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \Rightarrow \mathbf{p} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ \frac{z}{z/d} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

- Perspective projection matrix

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

Projection Pipeline

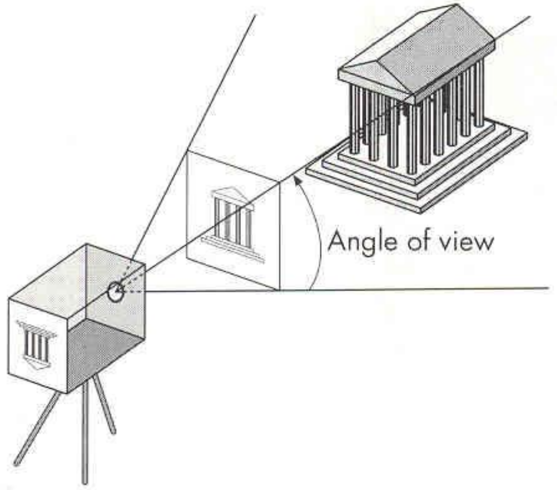
Perspective Division

- If $w \neq 1$, we must divide by w to return from homogeneous coordinates
- The **perspective division** yields the desired perspective equations

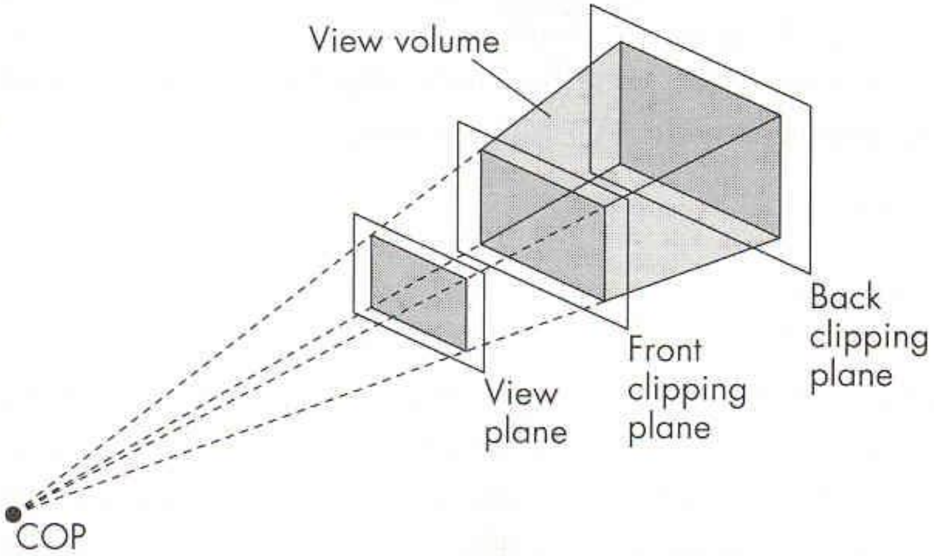
$$
\begin{aligned}
x_p &= x \\
y_p &= y \\
z_p &= z \\
w_p &= z/d
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
x_p &= \frac{x}{z/d} \\
y_p &= \frac{y}{z/d} \\
z_p &= d \\
w_p &= 1
\end{aligned}
$$

Projections in OpenGL

- Angle of view
  - Only objects that fit within the angle of view of the camera appear in the image

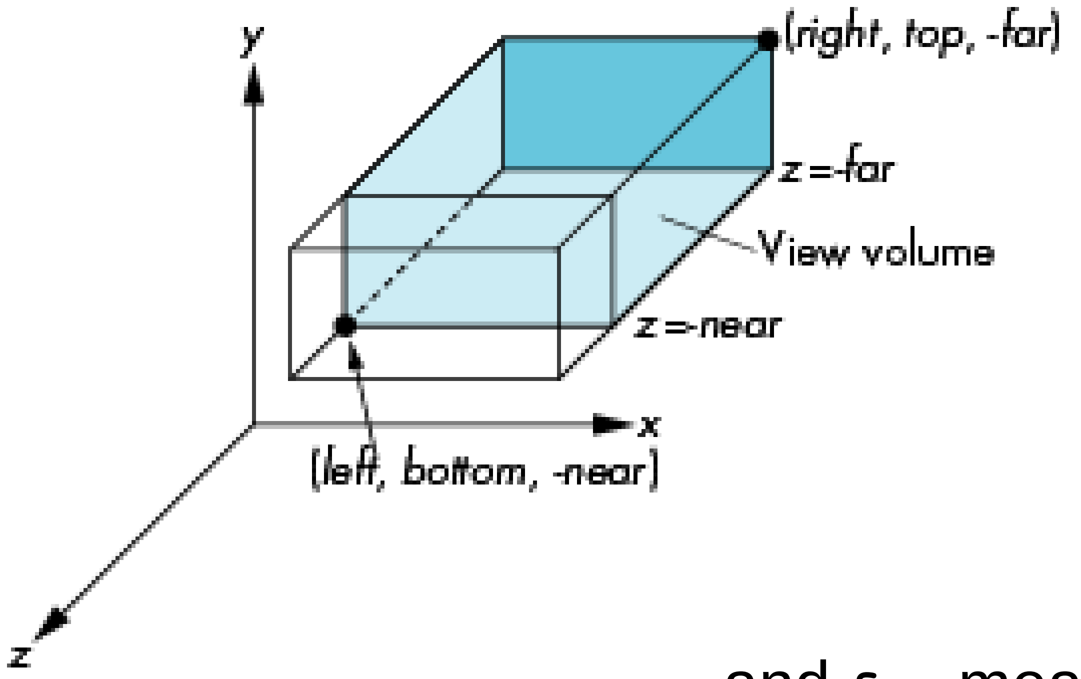


- View volume
  - Being clipped out of scene
  - **Frustum**: truncated pyramid



WebGL Orthographic Viewing

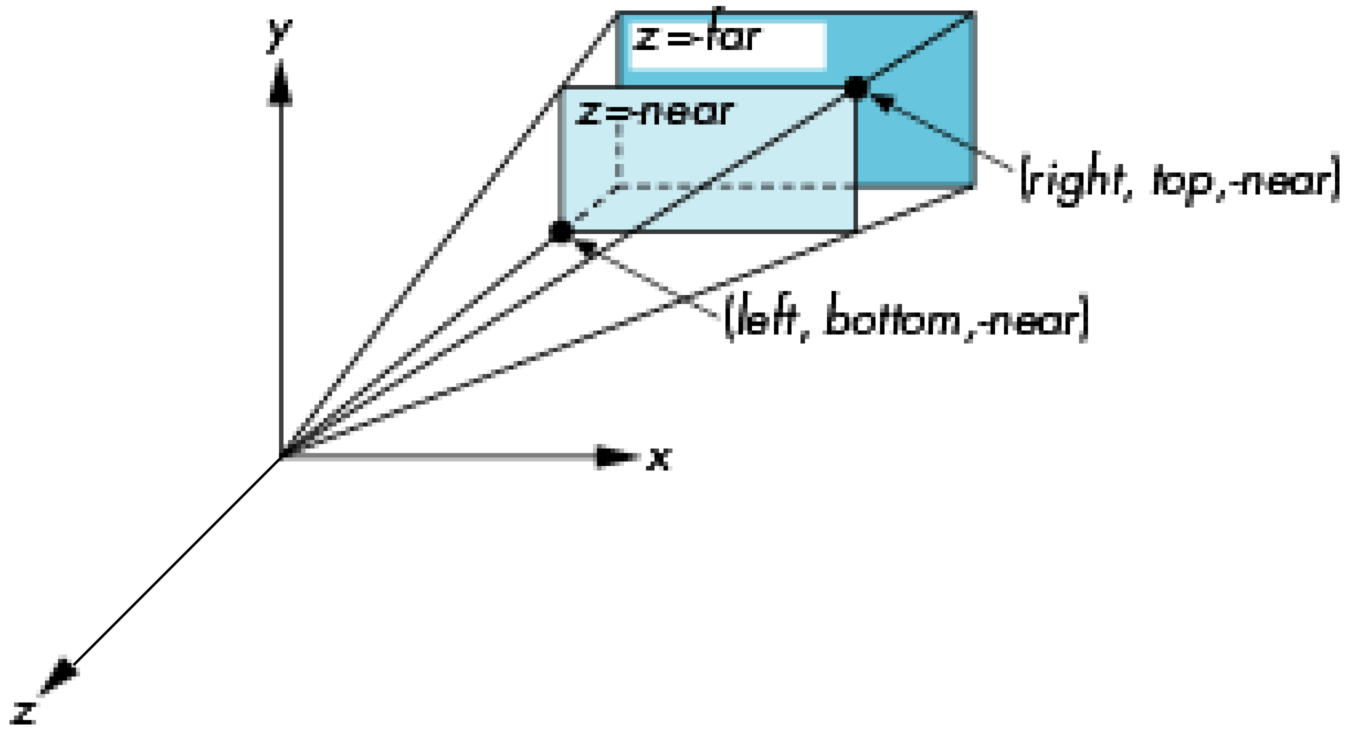
```
ortho(left,right,bottom,top,near,far)
```



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

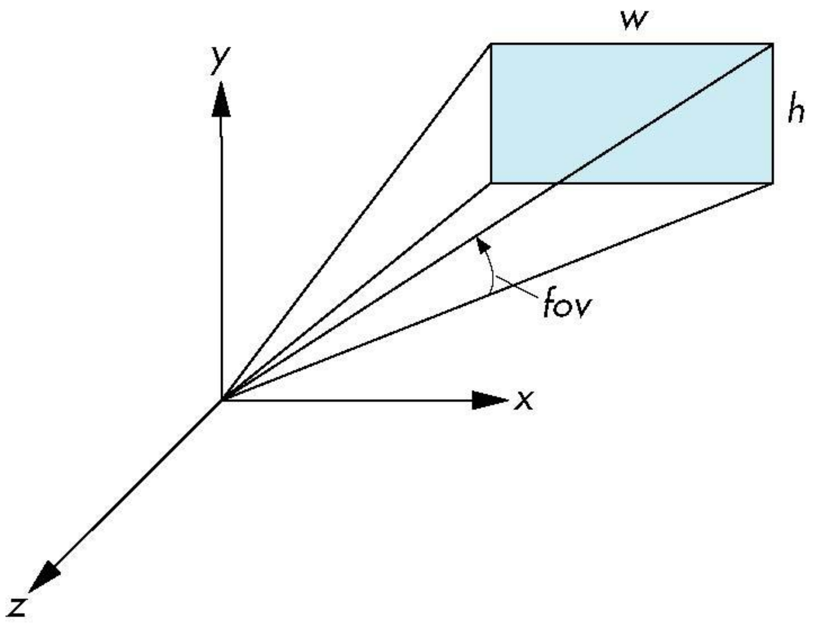
WebGL Perspective Viewing

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



Using Field of View

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

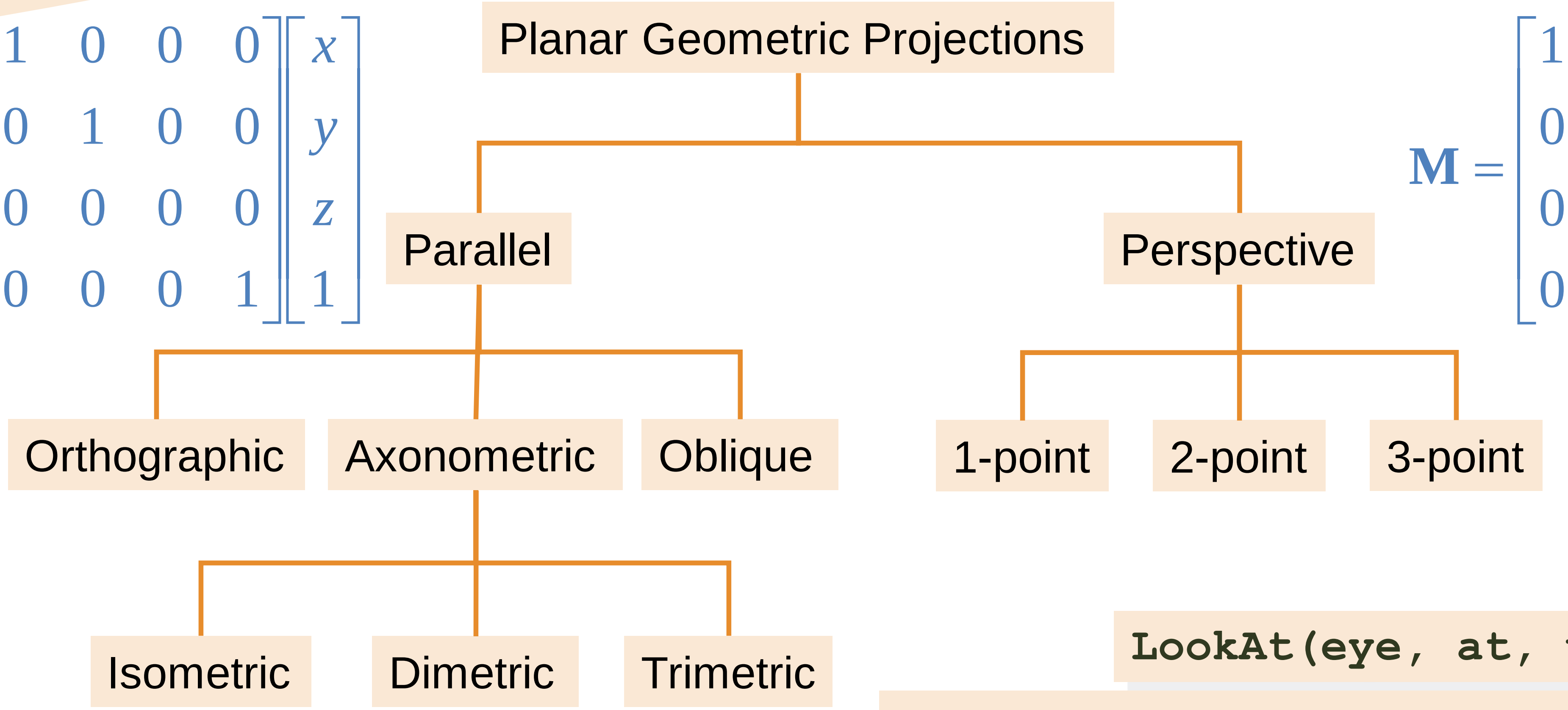


front plane

`aspect` = **`w/h`**

Summary

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



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

```
LookAt(eye, at, up);
ortho(left,right,bottom,top,near,far)
frustum(left,right,bottom,top,near,far)
perspective(fovy, aspect, near, far)
```

수고하셨습니다