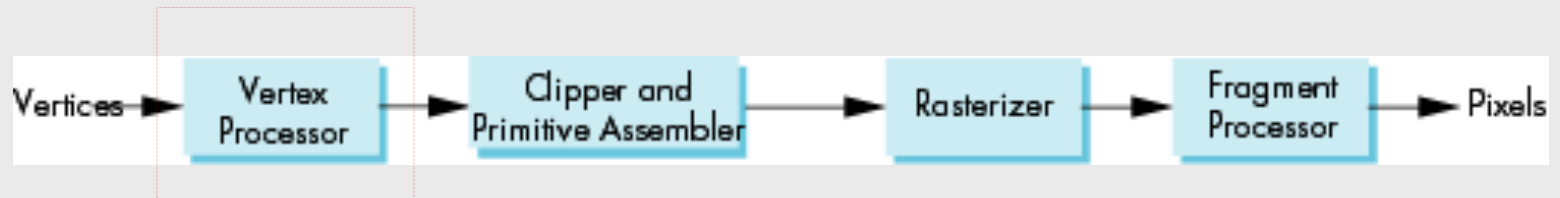


Comp 410/510

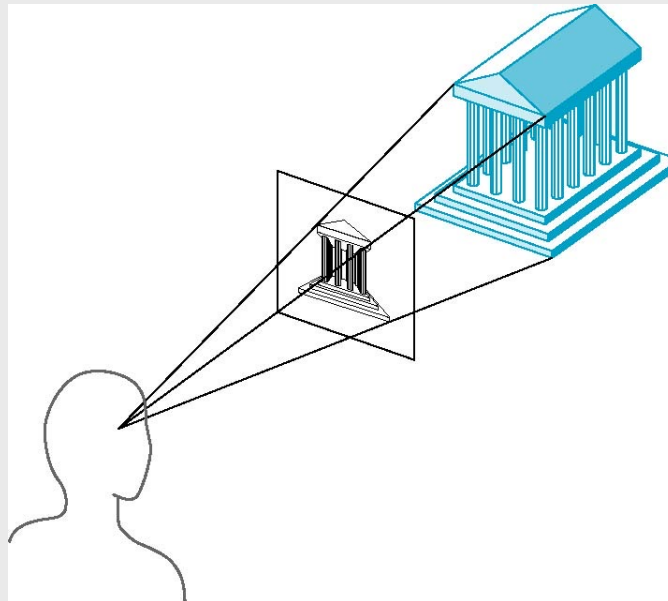
Computer Graphics
Spring 2023

Viewing & Projections



Viewing

- Viewing requires three basic elements
 - One or more objects
 - A viewer with a projection surface
 - Projectors that go from the object(s) to the projection surface
- Viewing is based on the relationship among these elements
 - The viewer picks up the object and orients it depending on how one would like to see it

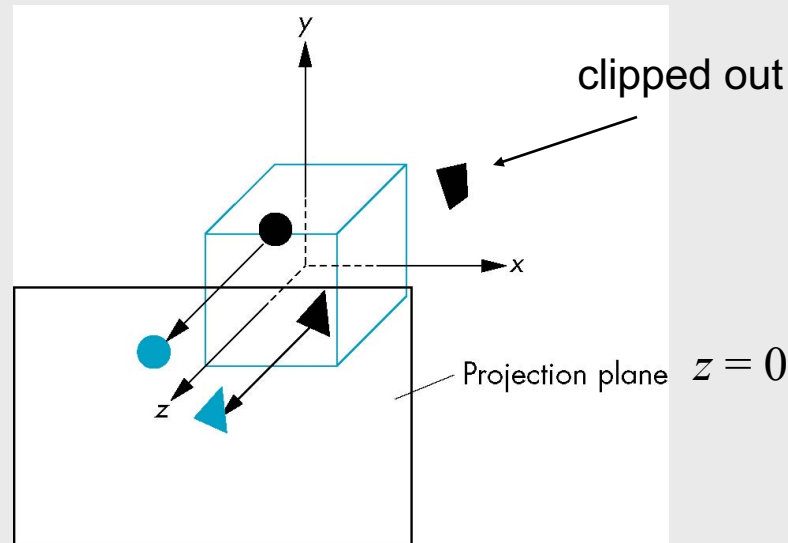


Computer Viewing

- There are three aspects of the computer 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

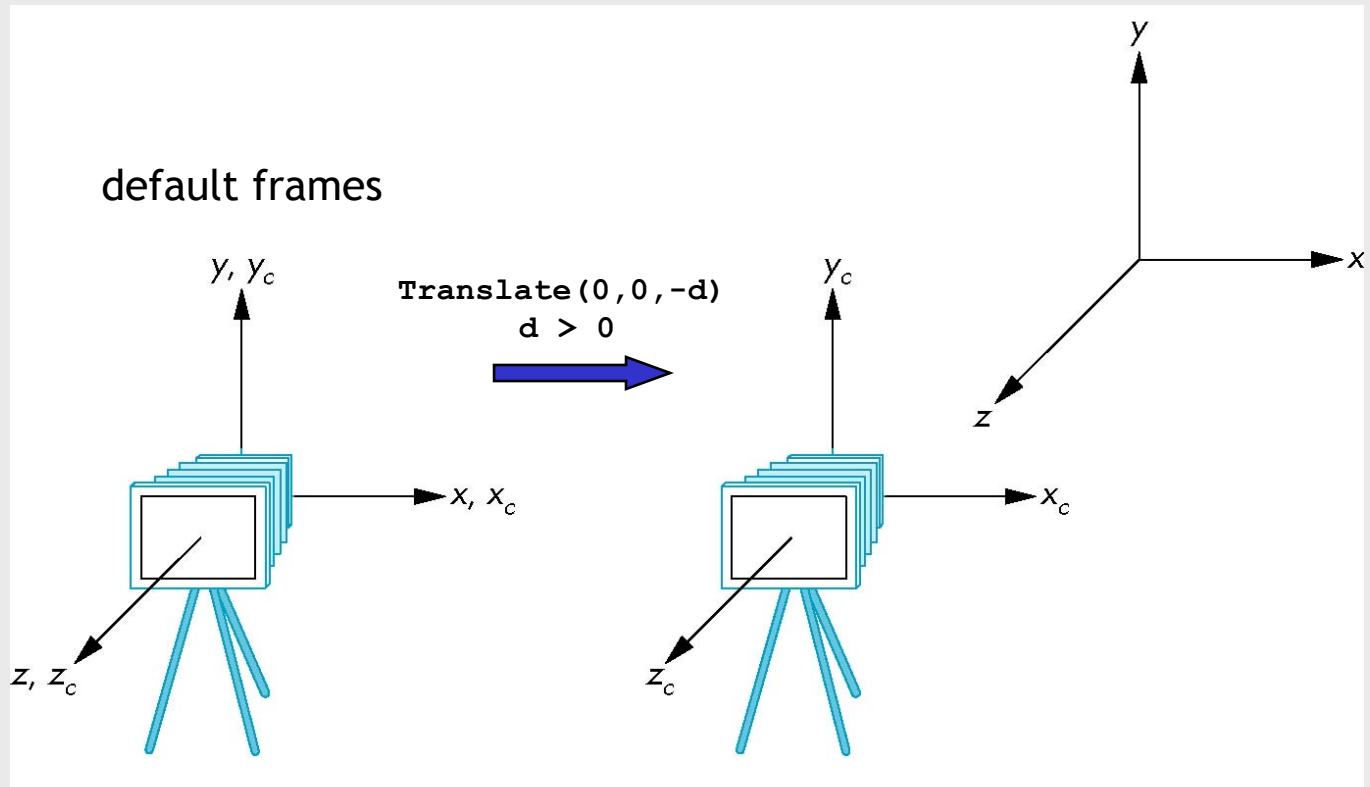
- In OpenGL, initially the world and camera frames are the same
 - Default model-view matrix is an identity matrix
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
 - Default projection is **orthographic**



Moving the Camera Frame

- If we want to visualize objects 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 can be specified by the model-view matrix with
 - e.g., `Translate(0.0,0.0,-d)` where $d > 0$

Moving Camera back from Origin



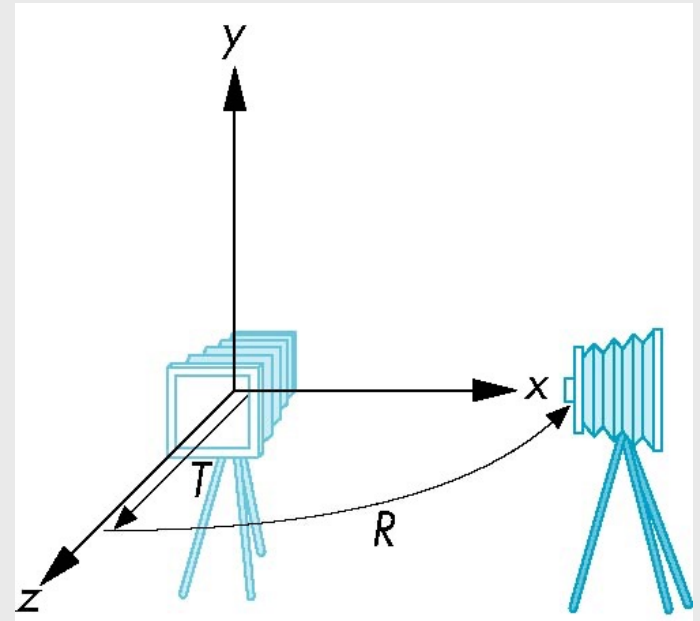
Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: Side view
 - Move first the camera away from origin (T)
 - Then rotate it (R)
 - For modelview, just **inverse** this transformation
 - Model-view: $M = T^{-1} R^{-1}$
- OpenGL code:

```
// Using mat.h
```

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

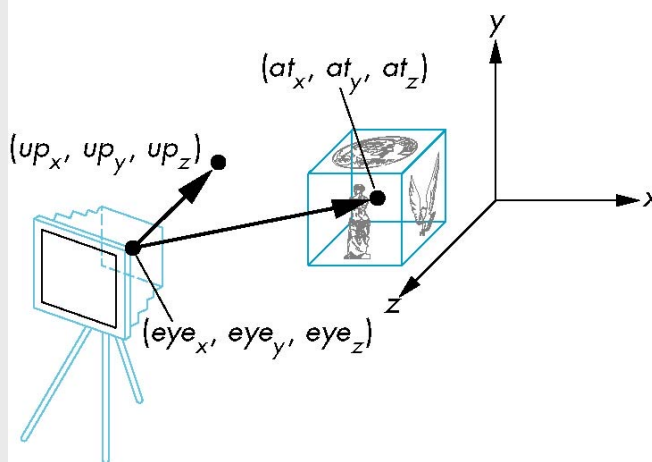
Remember that last transformation specified is first to be applied!



The LookAt Function

- The old GLU library contained the function `gluLookAt` to form the required modelview matrix through a simple interface (now deprecated)
- Can use instead the user supplied `LookAt()` in `mat.h`
 - Can concatenate with modeling transformations

```
mat4 LookAt(vec4 eye, vec4 at, vec4 up)
```

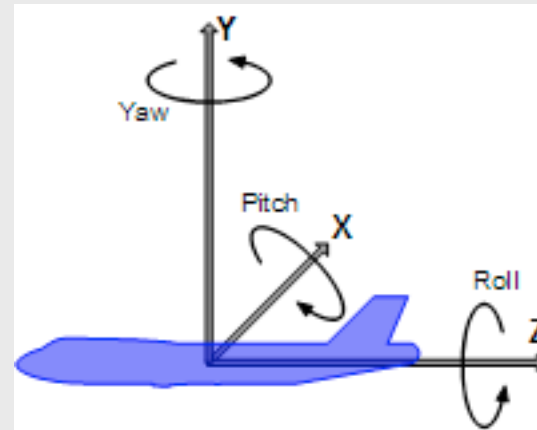
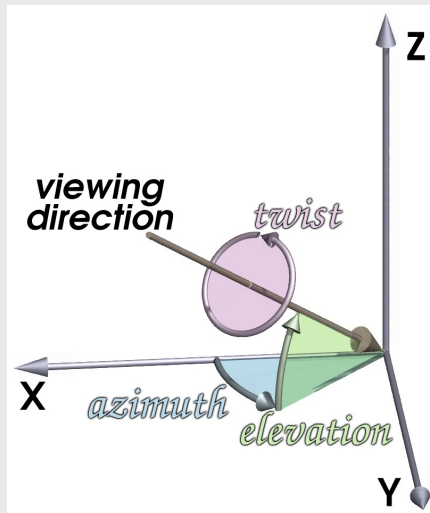


Projection plane \perp $(eye - at)$

Note the need for setting an **up** direction

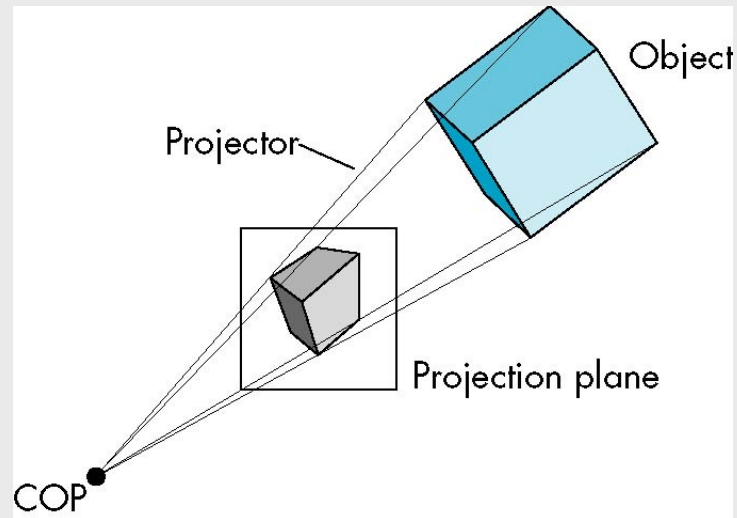
Other Viewing APIs

- The LookAt function is only one of many possible APIs for positioning the camera
- Others include
 - View reference point, view plane normal, view up (PHIGS, GKS-3D)
 - Yaw, pitch, roll
 - Azimuth, elevation, twist

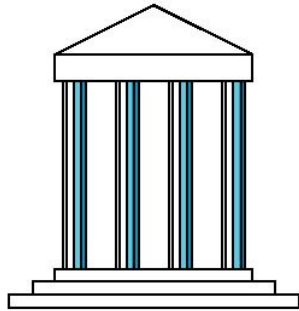


Planar Geometric Projections

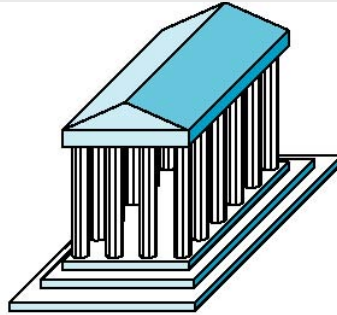
- Standard projections project onto a plane
- Projectors are lines that either
 - converge at a center of projection or
 - are parallel
- Such projections preserve lines
 - but not necessarily angles
- Non-planar projections are needed for applications such as map construction



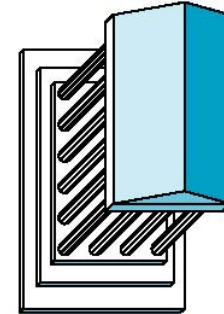
Classical Projections



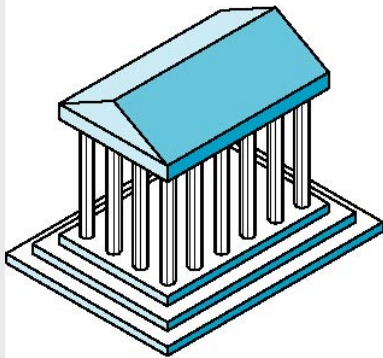
Front elevation



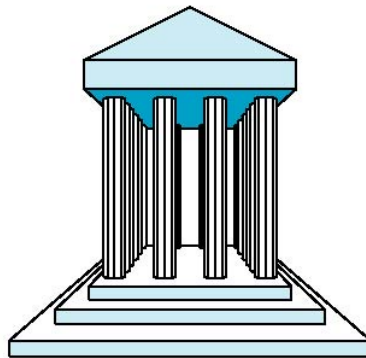
Elevation oblique



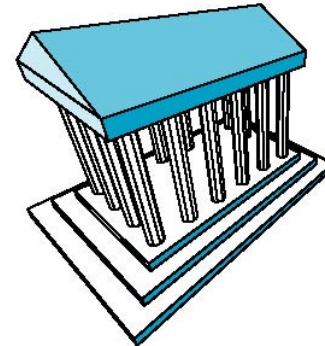
Plan oblique



Isometric



One-point perspective



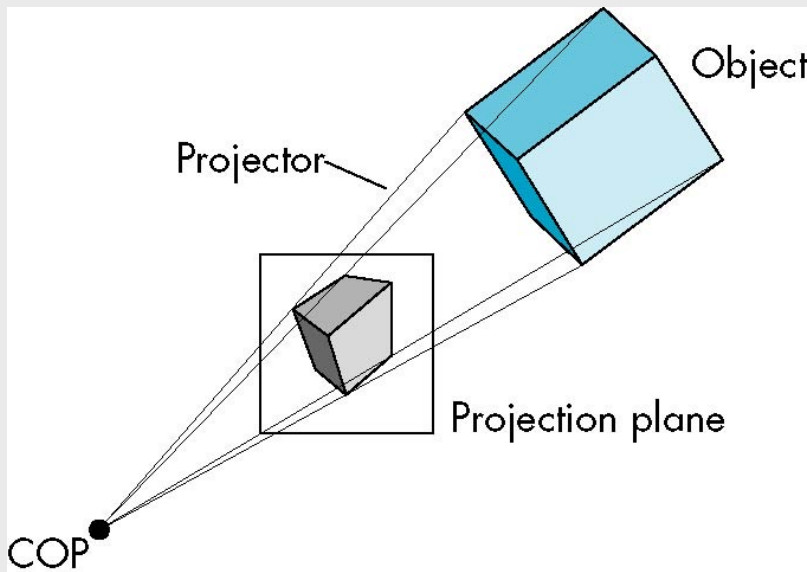
Three-point perspective

In classical viewing, each object is assumed to be constructed from **flat principal faces**.

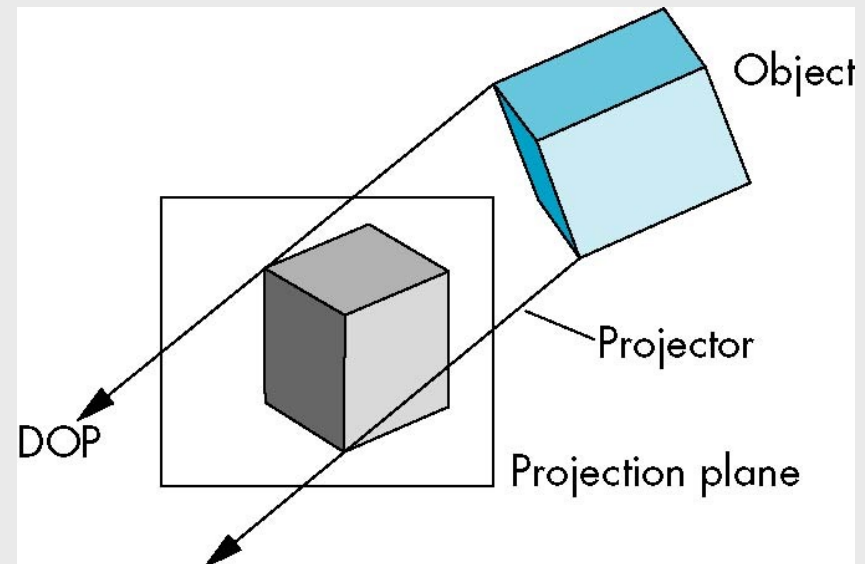
Examples: Buildings, polyhedra, manufactured objects

Perspective vs Parallel

- Fundamental distinction is between **parallel** and **perspective** projections, even though parallel viewing is mathematically the limit of perspective projection
- Any other distinction between different projections is a matter of how the projection plane is oriented with respect to the object, i.e., to the principal face(s).

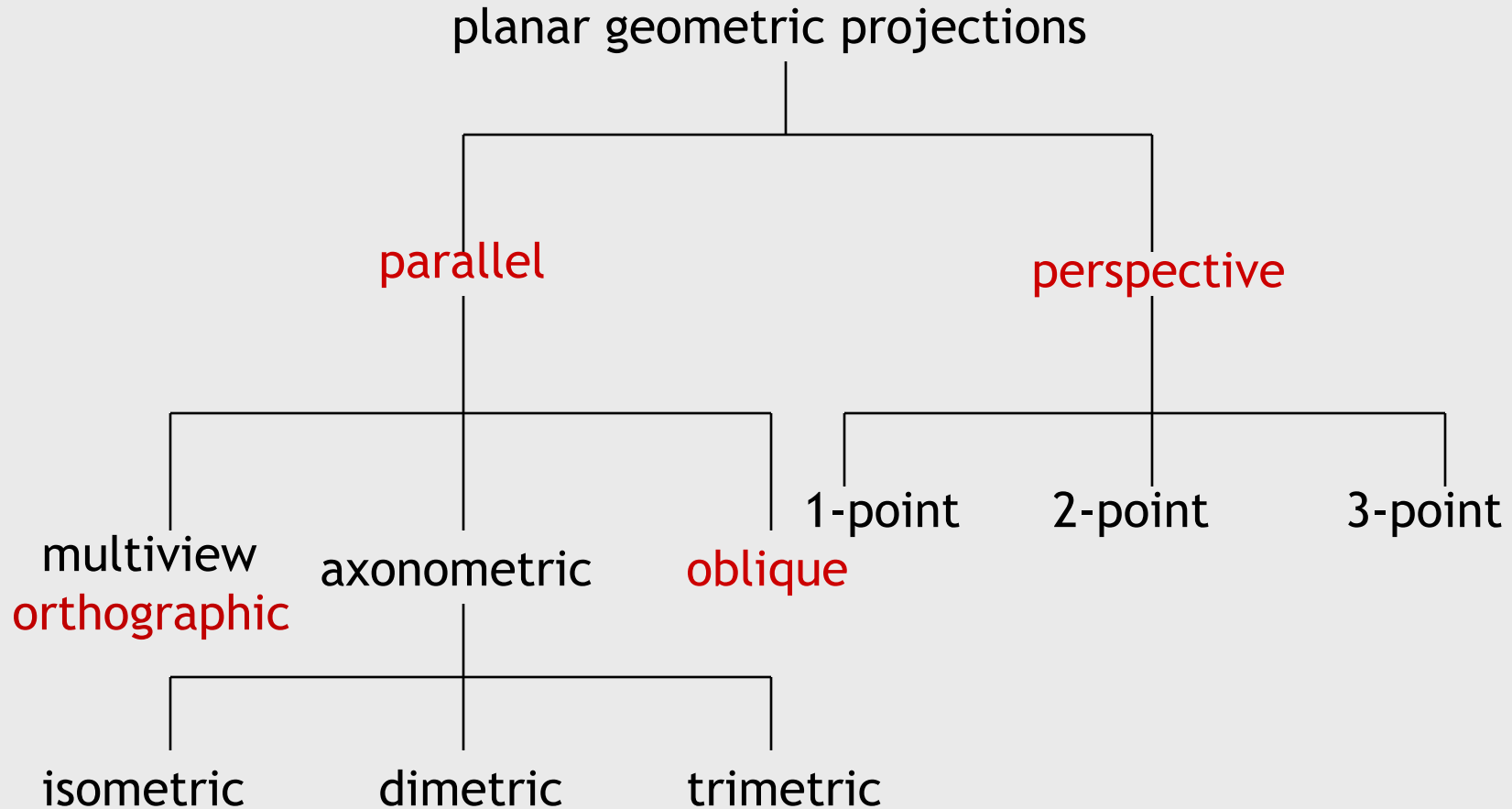


perspective



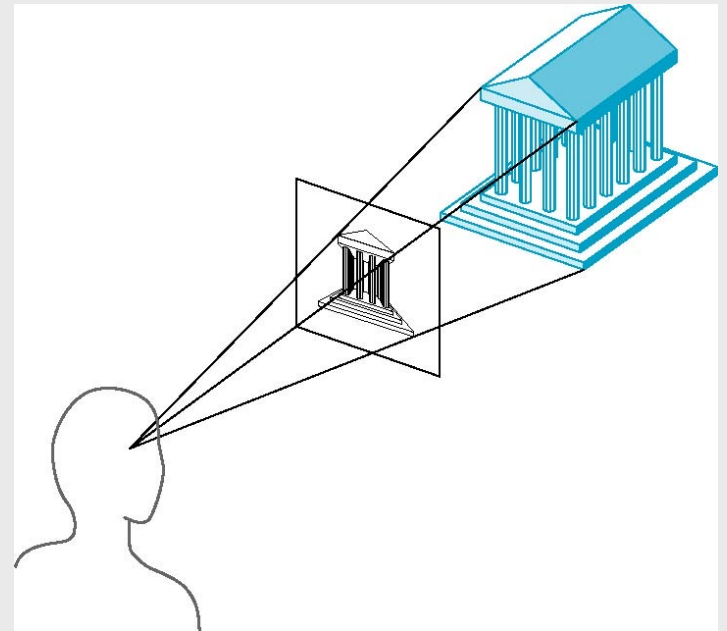
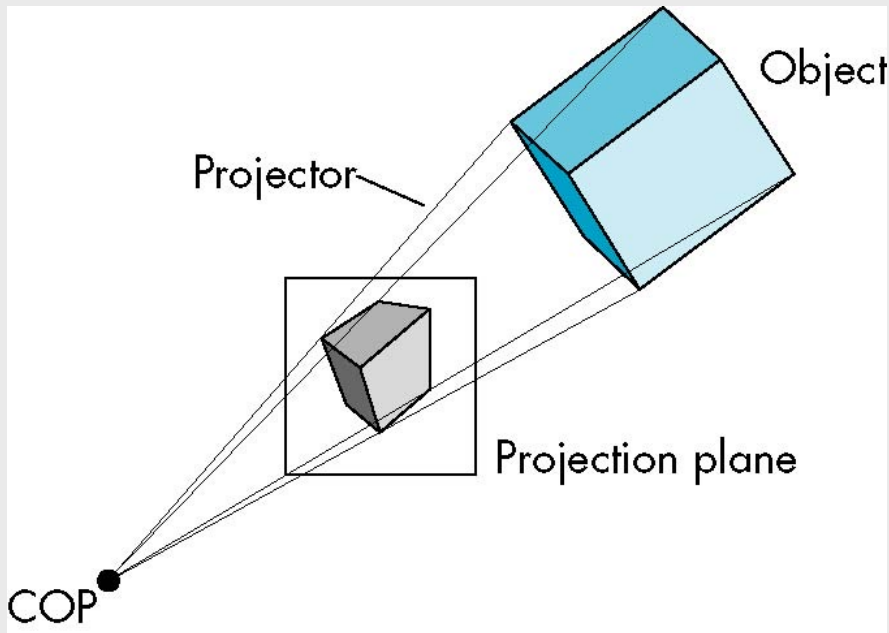
parallel

Taxonomy of Planar Geometric Projections



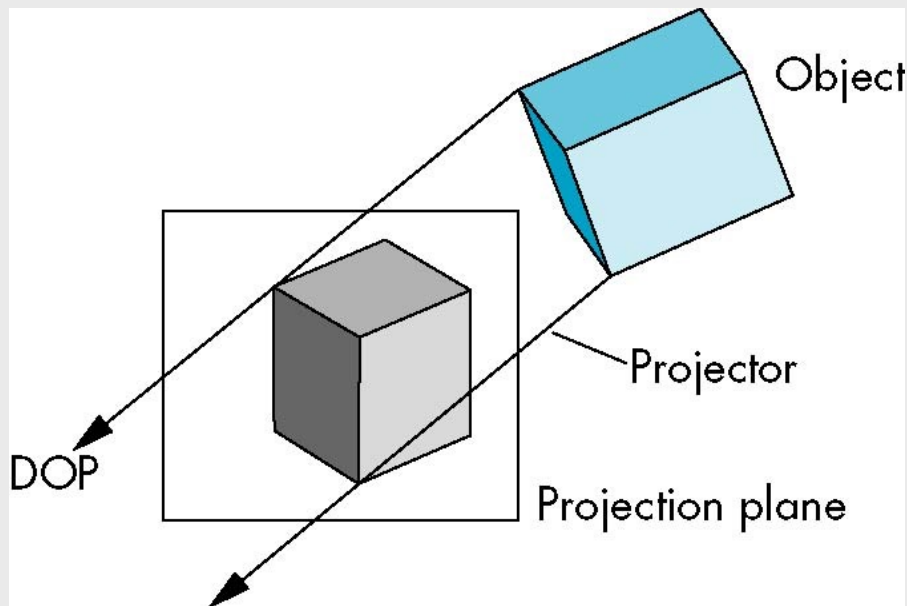
Perspective Projection

Projectors converge at the center of projection:



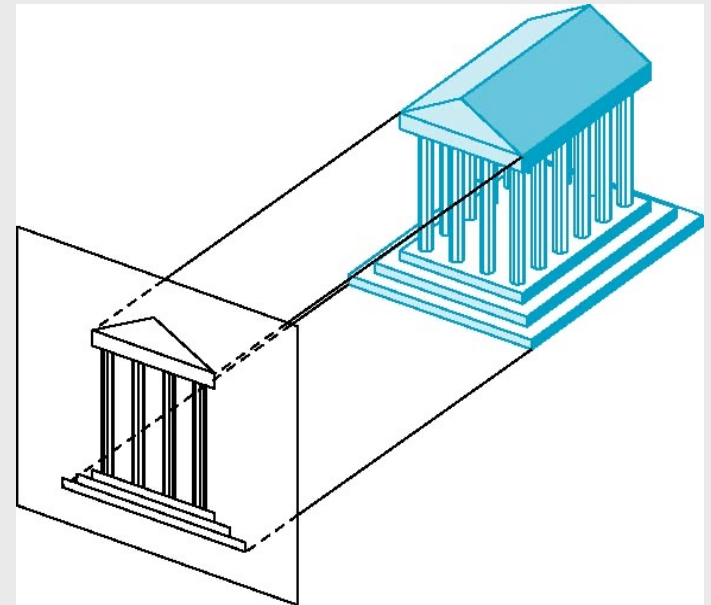
Parallel Projection

General Parallel Projection



Projectors are **parallel** to each other

Orthographic Projection

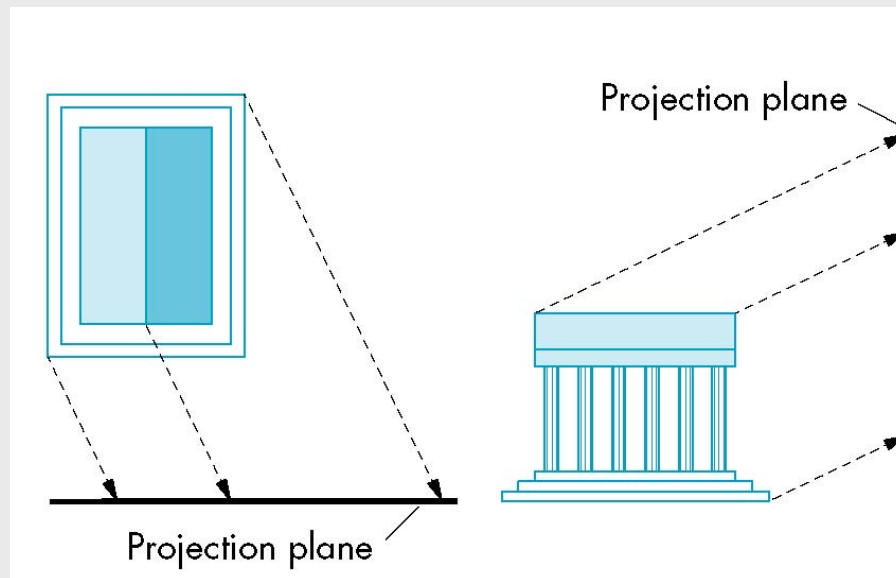


Projectors are **orthogonal** to projection surface

Remark: Orthographic (or orthogonal) projection is a special case of parallel projection.

Oblique Projection

- Special case of parallel projection
- Angles in faces parallel to projection plane are preserved while we can still see “around” side.



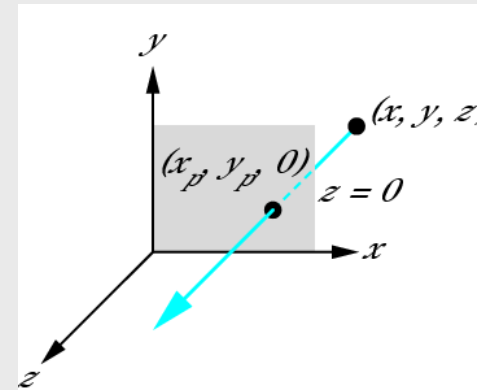
Projections and Normalization

- The default projection is orthographic
- For points within the default view volume

$$x_p = x$$

$$y_p = y$$

$$z_p = 0$$



- Most graphics systems use **view normalization**
 - All views are converted to the default orthographic view by using normalization transformations
 - Allows use of the same pipeline for all types of viewing
 - We will see view normalization next time
- First let's see how to write projections in homogeneous coordinates

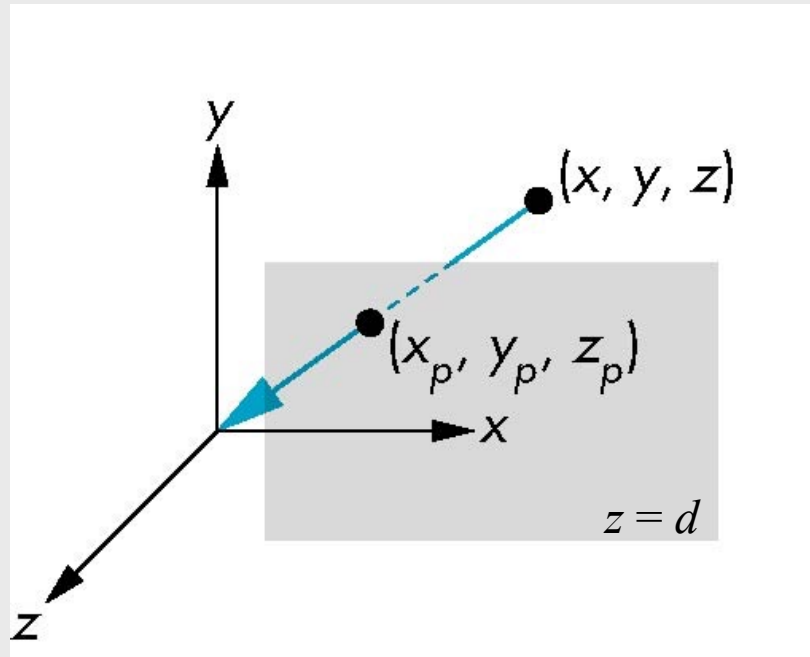
Default Orthographic Projection

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{l} x_p = x \\ y_p = y \\ z_p = 0 \\ w_p = 1 \end{array}$$

$$\mathbf{p}_p = \mathbf{M}_{\text{orth}} \mathbf{p}$$

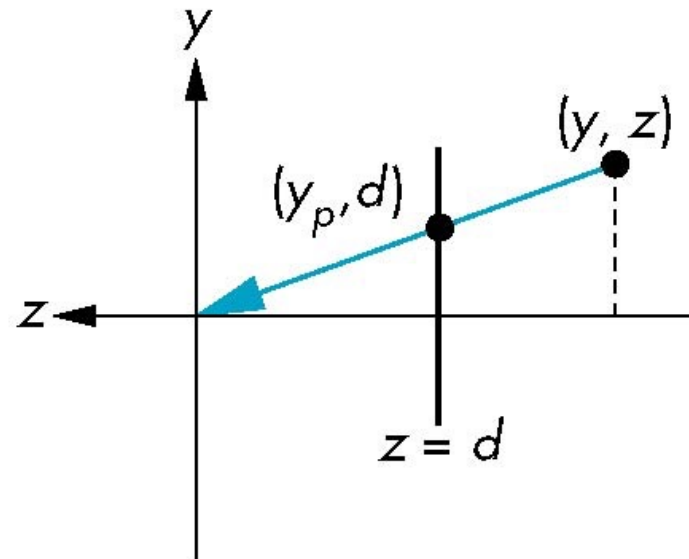
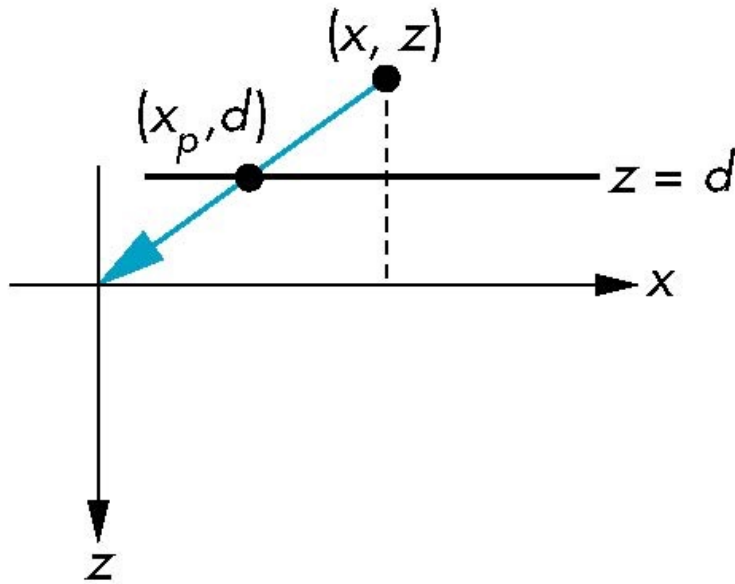
Simple Perspective Projection

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



Perspective Equations

Consider top and side views:



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

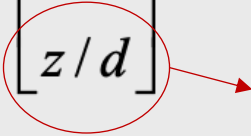
$$y_p = \frac{y}{z/d}$$

$$z_p = d$$

Simple Perspective Projection Matrix

Consider $\mathbf{p}'_p = \mathbf{M}_{\text{persp}} \mathbf{p}$ where $\mathbf{M}_{\text{persp}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

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

 not necessarily 1; needs perspective division

Perspective Division

- In the previous slide, $w \neq 1$, so we must divide each coordinate by w to return from homogeneous coordinates
- This **perspective division** yields

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

the desired perspective equations

OpenGL Projection

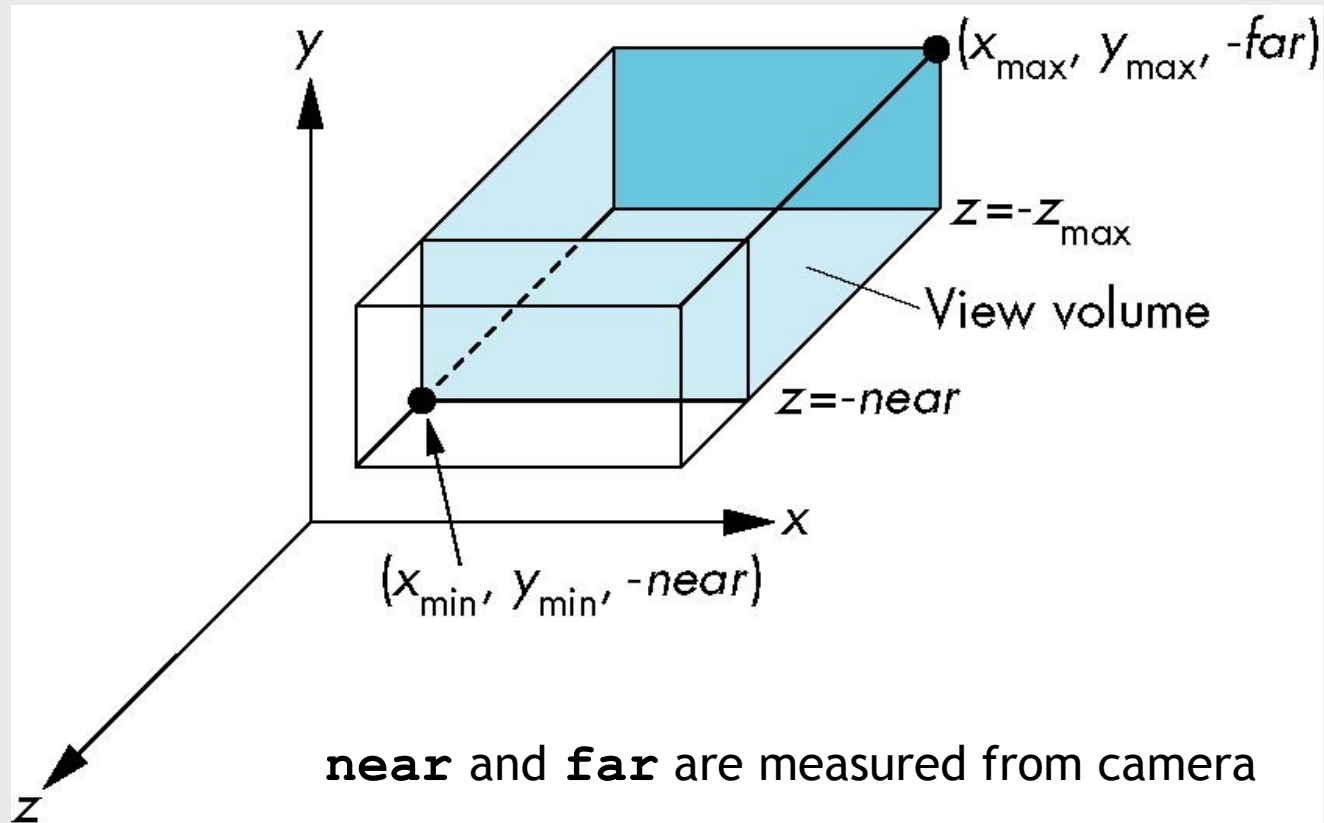
We specify the projection type and the clipping volume with `mat.h` functions that are equivalent to deprecated OpenGL functions:

- `Ortho()`
- `Frustum()`
- `Perspective()`

OpenGL Orthogonal Viewing

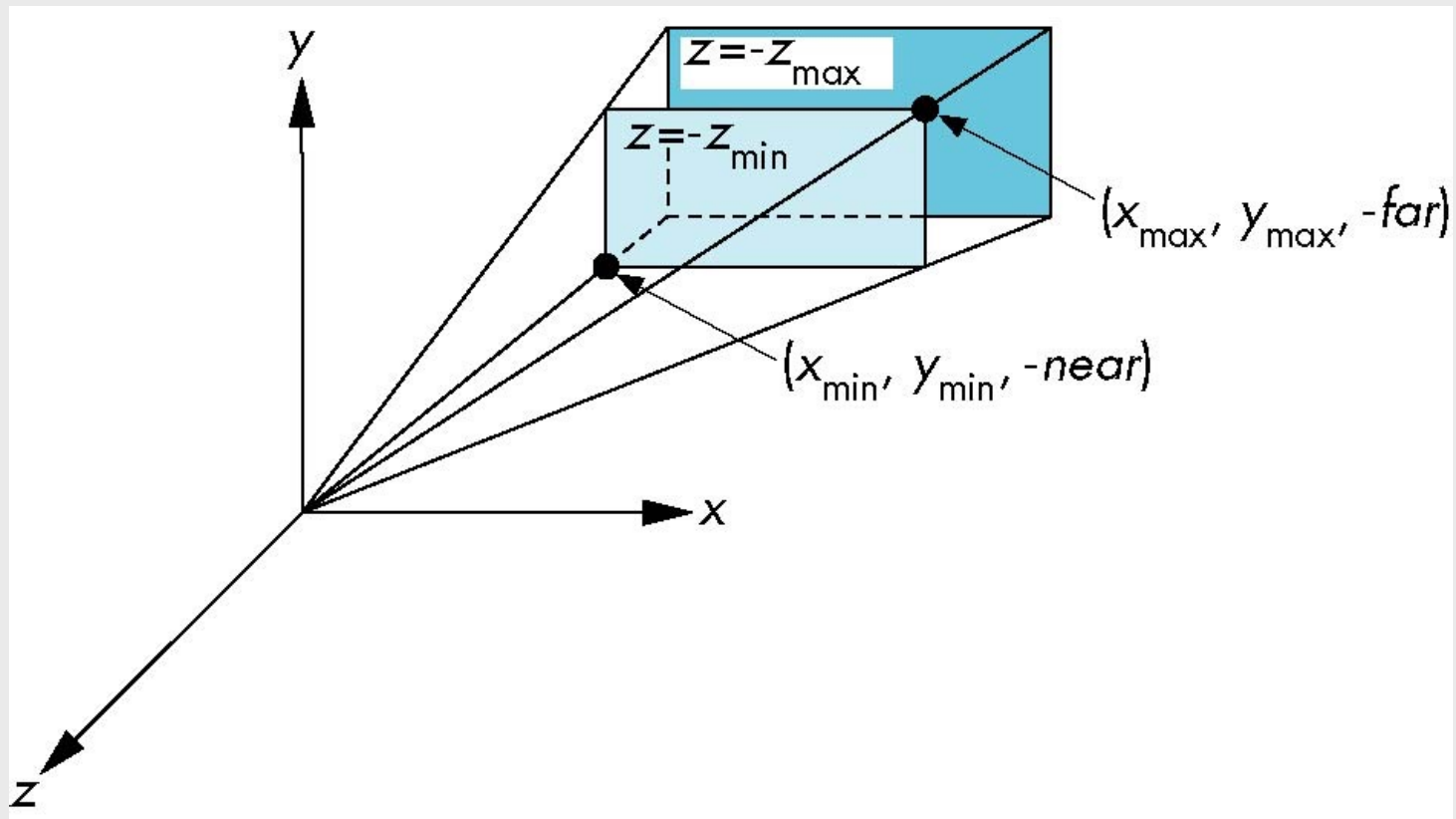
`Ortho(xmin, xmax, ymin, ymax, near, far)`

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



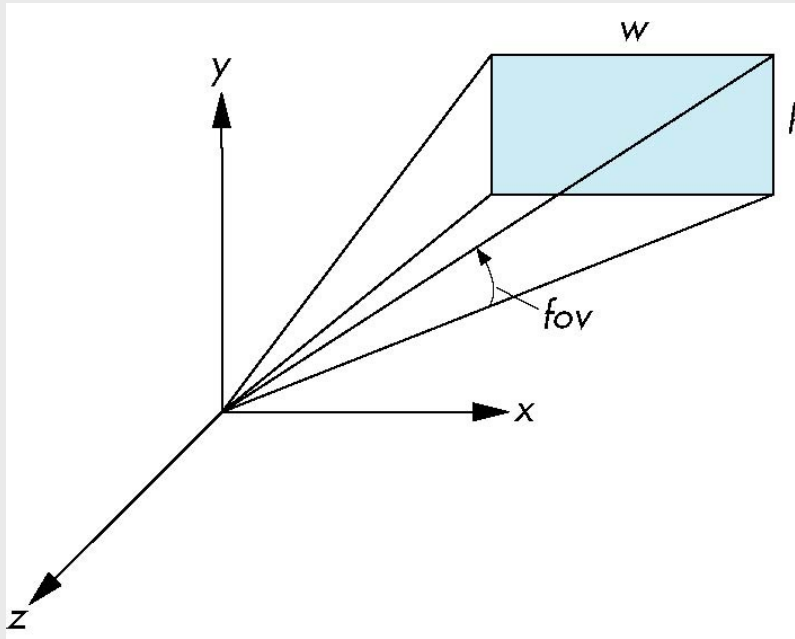
OpenGL Perspective

`Frustum(xmin, xmax, ymin, ymax, near, far)`



Using Field of View

- With `Frustum()`, it might be difficult to get the desired view
- `Perspective(fov, aspect, near, far)` provides a better interface



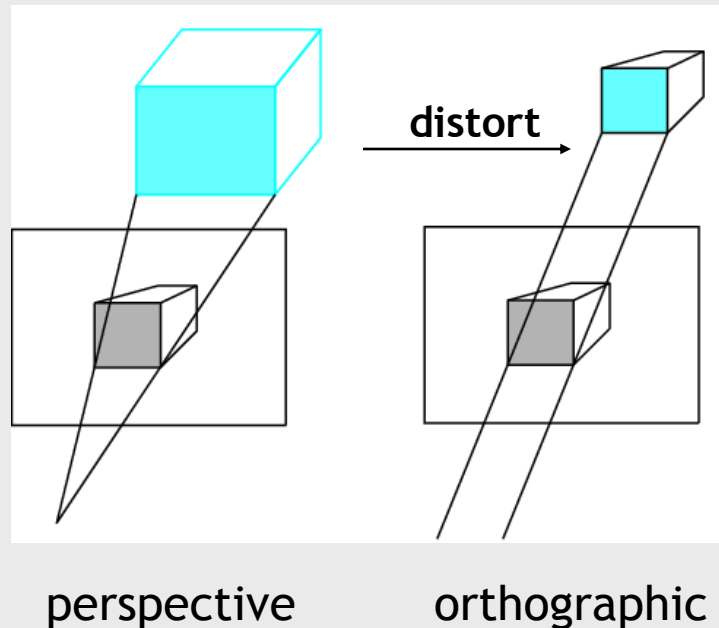
$$\text{aspect} = w/h$$

Objectives

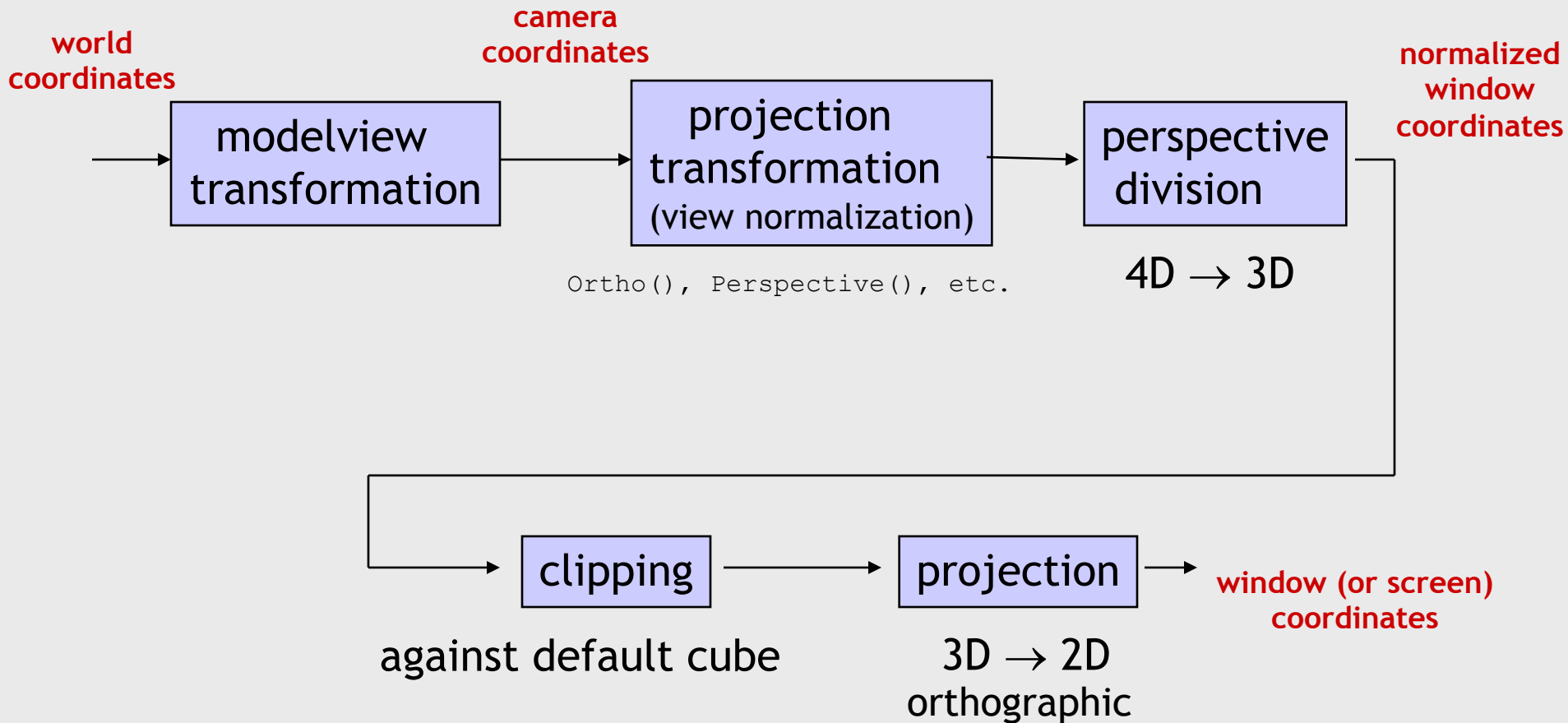
- Learn how projection is implemented in the pipeline
- Introduce projection normalization
- Implementation of `Ortho()`, `Perspective()`, `Frustum()`

Projection-Normalization

- Rather than deriving a different projection matrix for each type of projection, we can convert all projections to orthographic projection with the default view volume
- This strategy allows a single pipeline for all types of projections as well as efficient clipping



Pipeline View



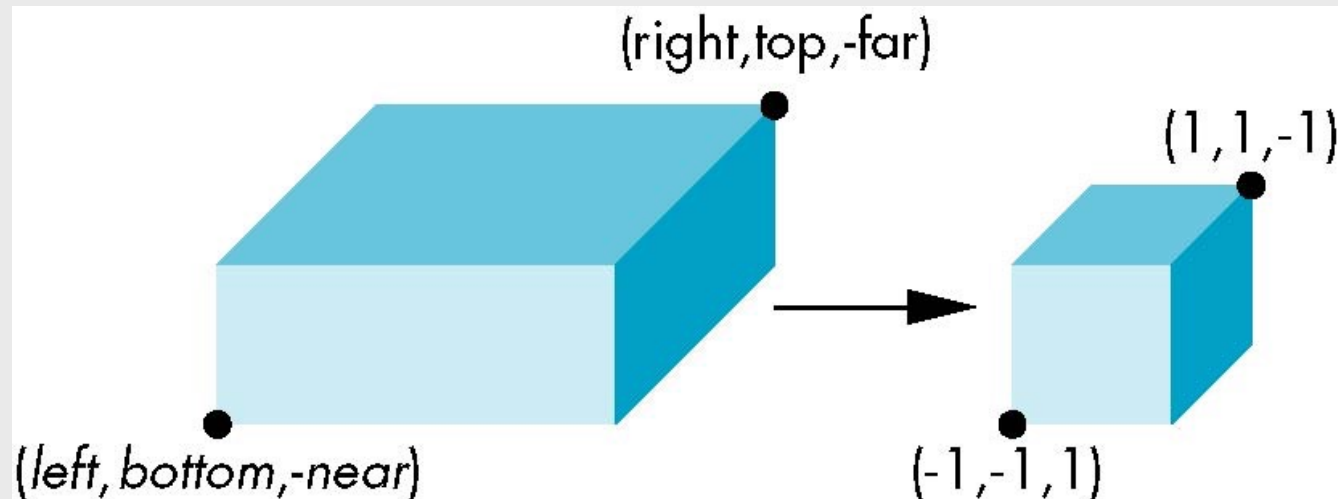
- We stay in four-dimensional homogeneous coordinates through both modelview and projection-normalization transformations
 - Both these transformations are nonsingular
 - Default to identity matrices (orthographic view)

Normalization of Orthographic Projection

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

returns 4x4 normalization matrix N

$N \Rightarrow$ the transformation that converts
specified clipping volume to default



Normalization Transformation

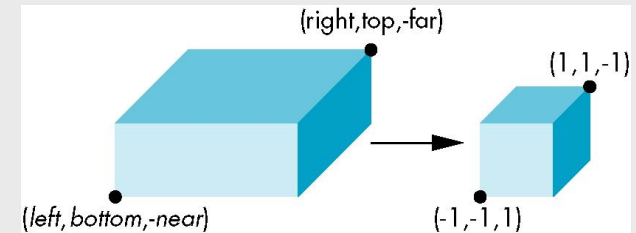
- Two steps

- Move center to origin

$$T \left(-\frac{left+right}{2}, -\frac{bottom+top}{2}, \frac{near+far}{2} \right)$$

- Scale to have sides of length 2

$$S \left(\frac{2}{right-left}, \frac{2}{top-bottom}, \frac{2}{near-far} \right)$$



$$N = ST = \begin{bmatrix} \frac{-2}{left - right} & 0 & 0 & -\frac{left + right}{left - right} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{near - far} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that this transforms $z=-near$ to $z=-1$ and $z=-far$ to $z=1$.

Final Projection

- Set $z = 0$
- Equivalent to homogeneous coordinate transformation

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

- Hence, general orthographic projection in 4D is

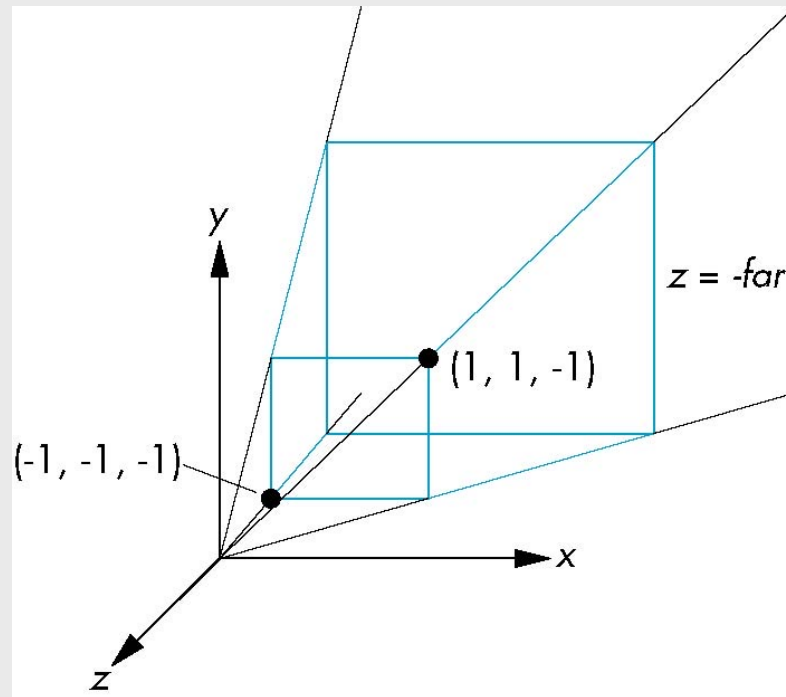
$$\mathbf{P} = \mathbf{M}_{\text{orth}} \mathbf{ST}$$

Clipping is performed **between** transformations \mathbf{M}_{orth} and \mathbf{ST}

Normalization of Perspective Projection

Consider a **simple** perspective projection

- with COP at the origin,
- the projection plane at $z = -1$, and
- 90 degree field of view determined by the planes $x = \pm z$ and $y = \pm z$

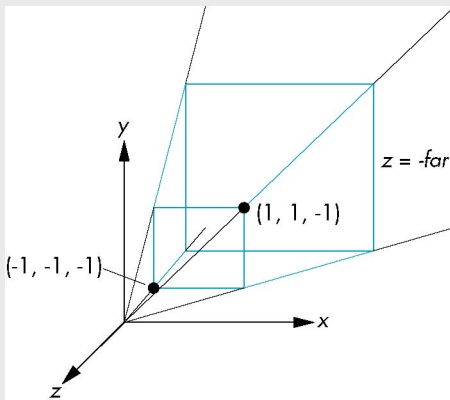


Simple Perspective Projection Matrix

Simple perspective projection matrix with $d = -1$:

$$\mathbf{M}_{\text{persp}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note that this matrix is independent of the far clipping plane



$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -x/z \\ -y/z \\ -1 \\ 1 \end{bmatrix}$$

Normalization Transformation

Consider view normalization with:

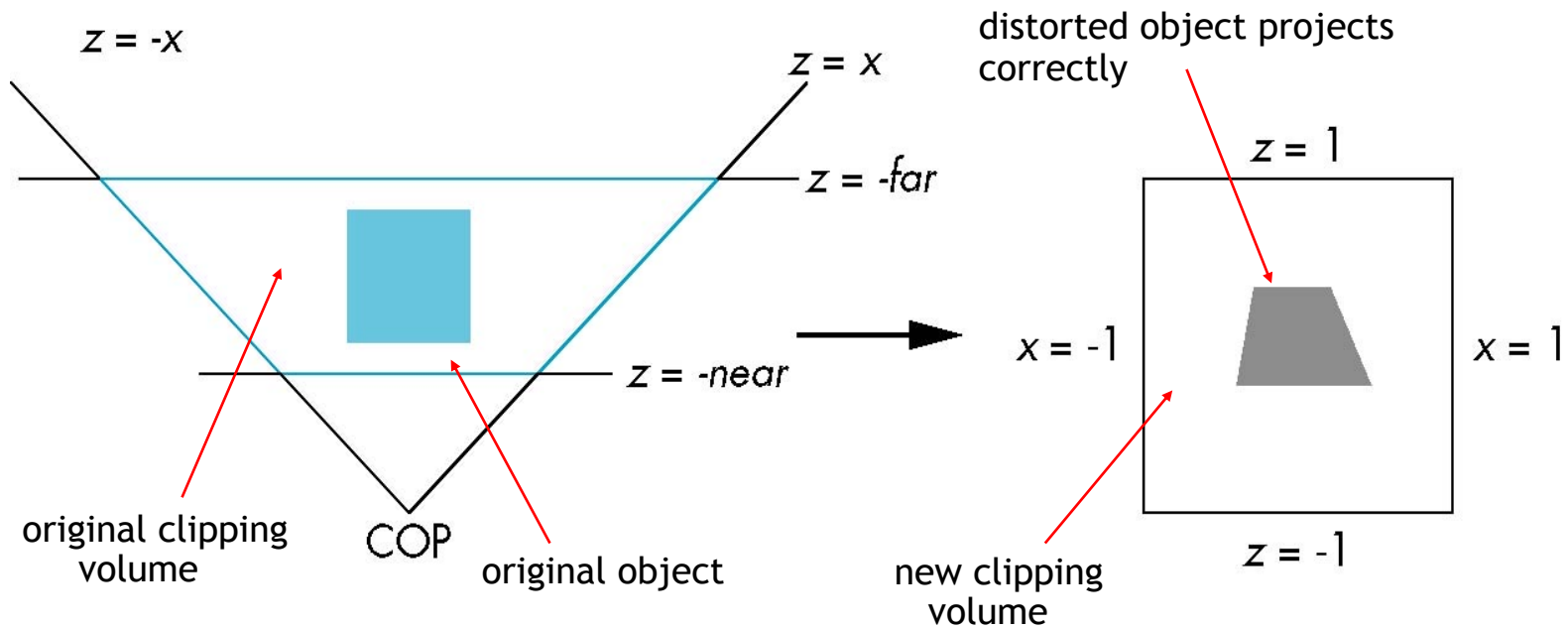
$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$M_{\text{orth}} N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$\nearrow M_{\text{persp}}$
Simple perspective
projection matrix
(except **one entry** which
is not significant)

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -x/z \\ -y/z \\ 0 \\ 1 \end{bmatrix}$$

Perspective View Normalization



Effect of N

$$Np = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

After perspective division, the point $(x, y, z, 1)$ goes to

$$x'' = -x/z$$

$$y'' = -y/z$$

$$z'' = -(\alpha + \beta/z)$$

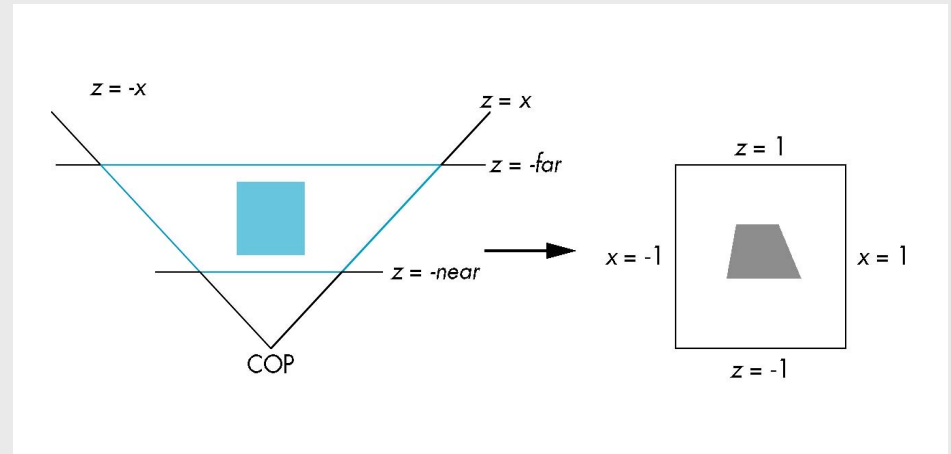
How to choose α and β to get the default clipping volume?

Picking α and β

If we pick

$$\alpha = - \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2\text{near} * \text{far}}{\text{near} - \text{far}}$$



$$x'' = -x/z$$

$$y'' = -y/z$$

$$z'' = -(\alpha + \beta/z)$$



near plane is mapped to $z = -1$

far plane is mapped to $z = 1$

and the sides are mapped to $x = \pm 1, y = \pm 1$

Hence the new clipping volume becomes the default clipping volume.

View Normalization & Hidden-Surface Removal

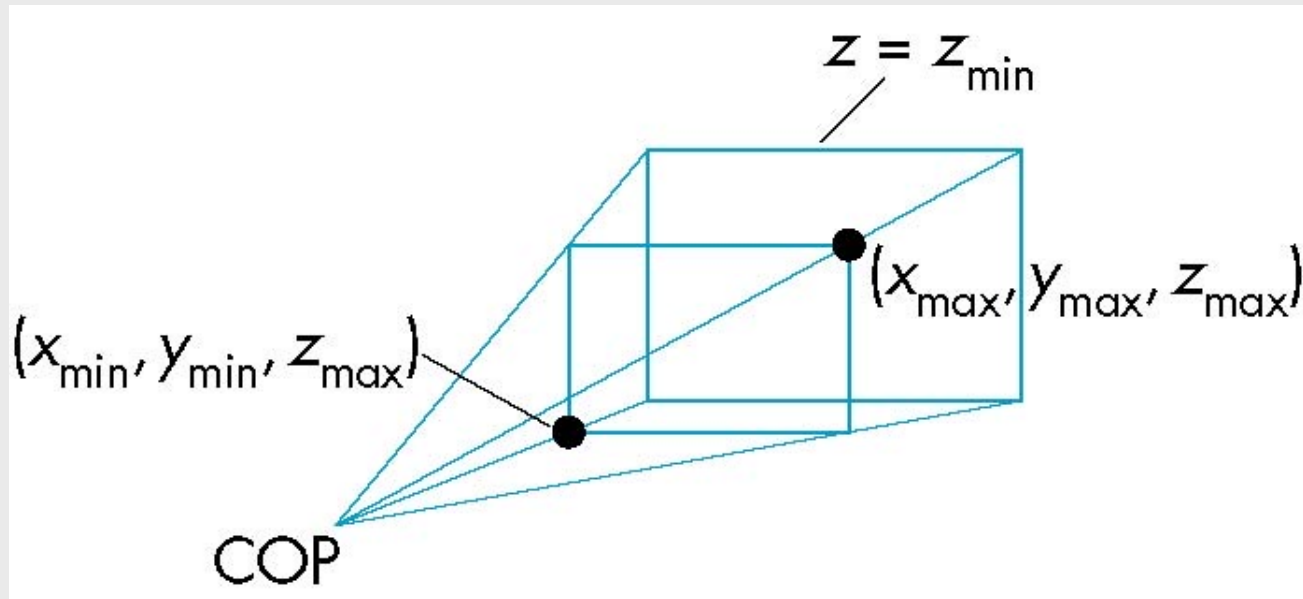
- Although our selection of the form of the perspective matrix may appear somewhat arbitrary, it is such that if $z_1 > z_2$ in the original clipping volume, then so is for the transformed points: $z_1'' > z_2''$

$$\begin{aligned}x'' &= -x/z \\y'' &= -y/z \\z'' &= -(\alpha + \beta/z)\end{aligned}$$

- Thus hidden surface removal works when we first apply the normalization transformation.
- However, the formula $z'' = -(\alpha + \beta/z)$ implies that the distances are distorted by the normalization, which can cause numerical problems, especially if the near distance is small, i.e., when $z \approx 0$.
- So when using perspective projection, never and ever set the near plane close to origin

OpenGL Perspective (mat.h)

- **Frustum()** allows for an unsymmetric viewing frustum
- Although **Perspective()** does not



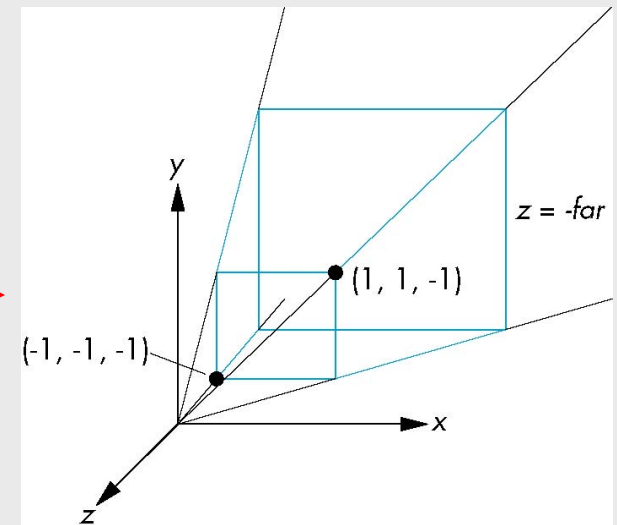
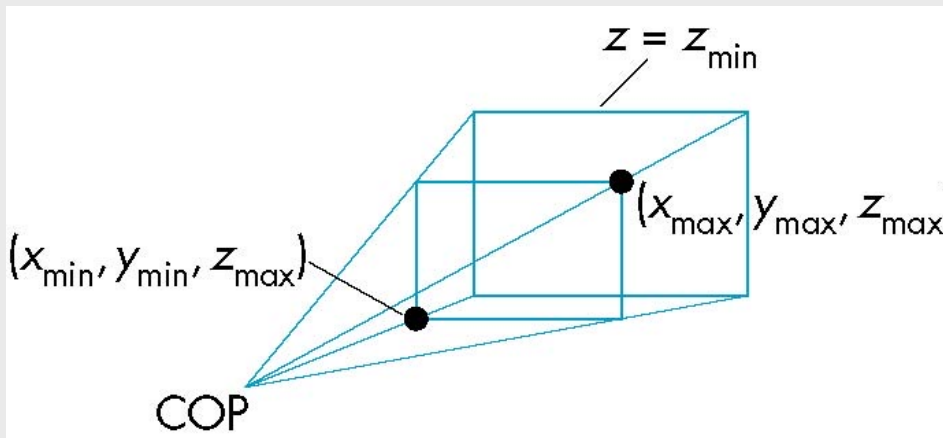
OpenGL Perspective Matrix

- The normalization in `Frustum()` requires an initial **shear** to form a right viewing pyramid, followed by a **scaling** to get the normalized perspective volume.
- Finally, the perspective matrix needs a final simple orthographic transformation

$$P = M_{\text{orth}} NSH$$

our previously defined
perspective normalization matrix

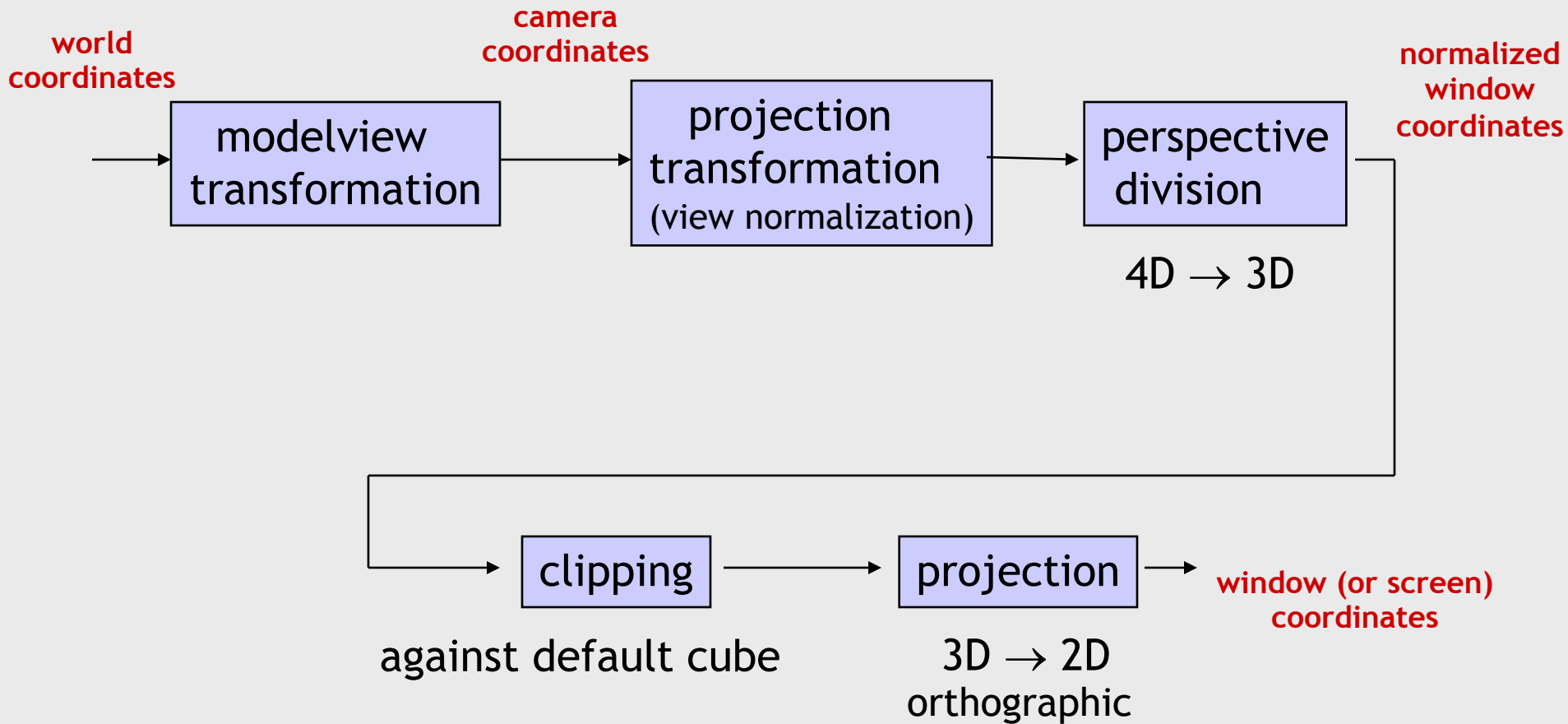
shear and scale



Why do we need view normalization?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing
- We delay final projection until the end as long as possible to retain 3D information needed for hidden-surface removal and shading
- We simplify clipping

Pipeline View



OpenGL and Shadows

- OpenGL, as a local illumination graphics API, does not support shadows.
- Yet we can explicitly create shadows while rendering by using projections.

Projections and Shadows

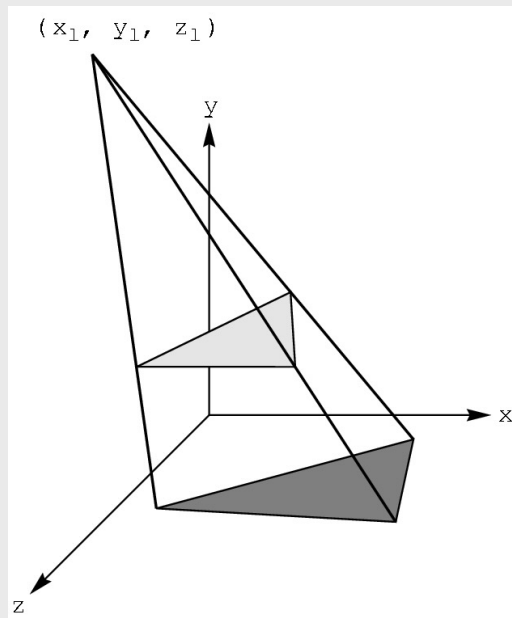
Suppose the shadow plane is $y = 0$ with a single point light source (x_l, y_l, z_l) .

To draw shadows, render the entire scene with modelview $T^{-1} M_{\text{persp}} T$:

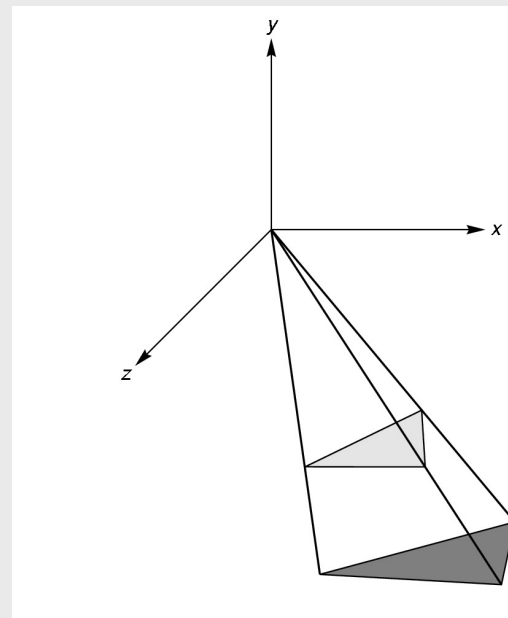
- T : Translate as if light source moves to camera origin
- M_{persp} : Perspective projection onto shadow plane
- T^{-1} : Undo the translation

$$M_{\text{persp}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1/y_l & 0 & 0 \end{bmatrix}$$

Shadow plane:
 $y = 0$



T

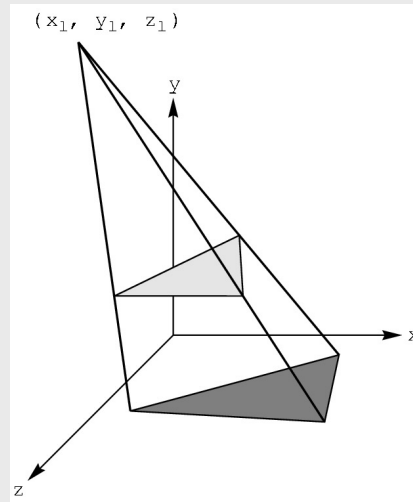


Shadow plane:
 $y = -y_l$

Algorithm to create shadows

Render the same polygon(s) twice:

- the first time as usual with current modelview matrix M
- the second time as a shadow with an altered modelview matrix that transforms the vertices with $T^{-1} M_{\text{persp}} T M$.



See “Projections and Shadows” section in your textbook.

See also the code on the course website.