NUSRI Summer Programme 2016

RI3004A
3D Graphics Rendering

Lecture 5
Viewing

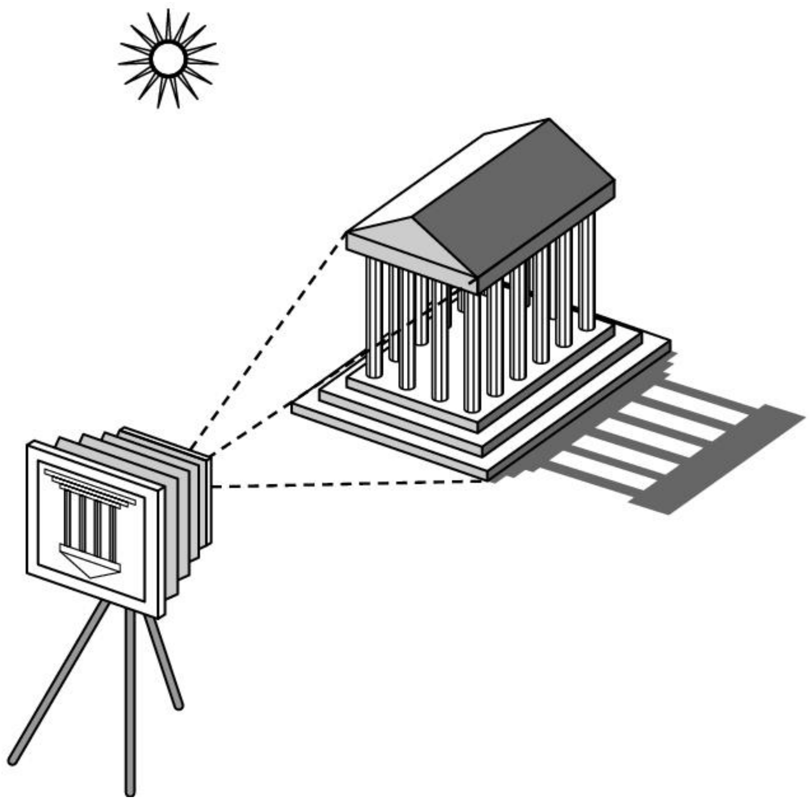
School *of* Computing
National University *of* Singapore

Objectives

- Specifying camera position and orientation
  - View transformation
- Projection
  - Orthographic
  - Perspective

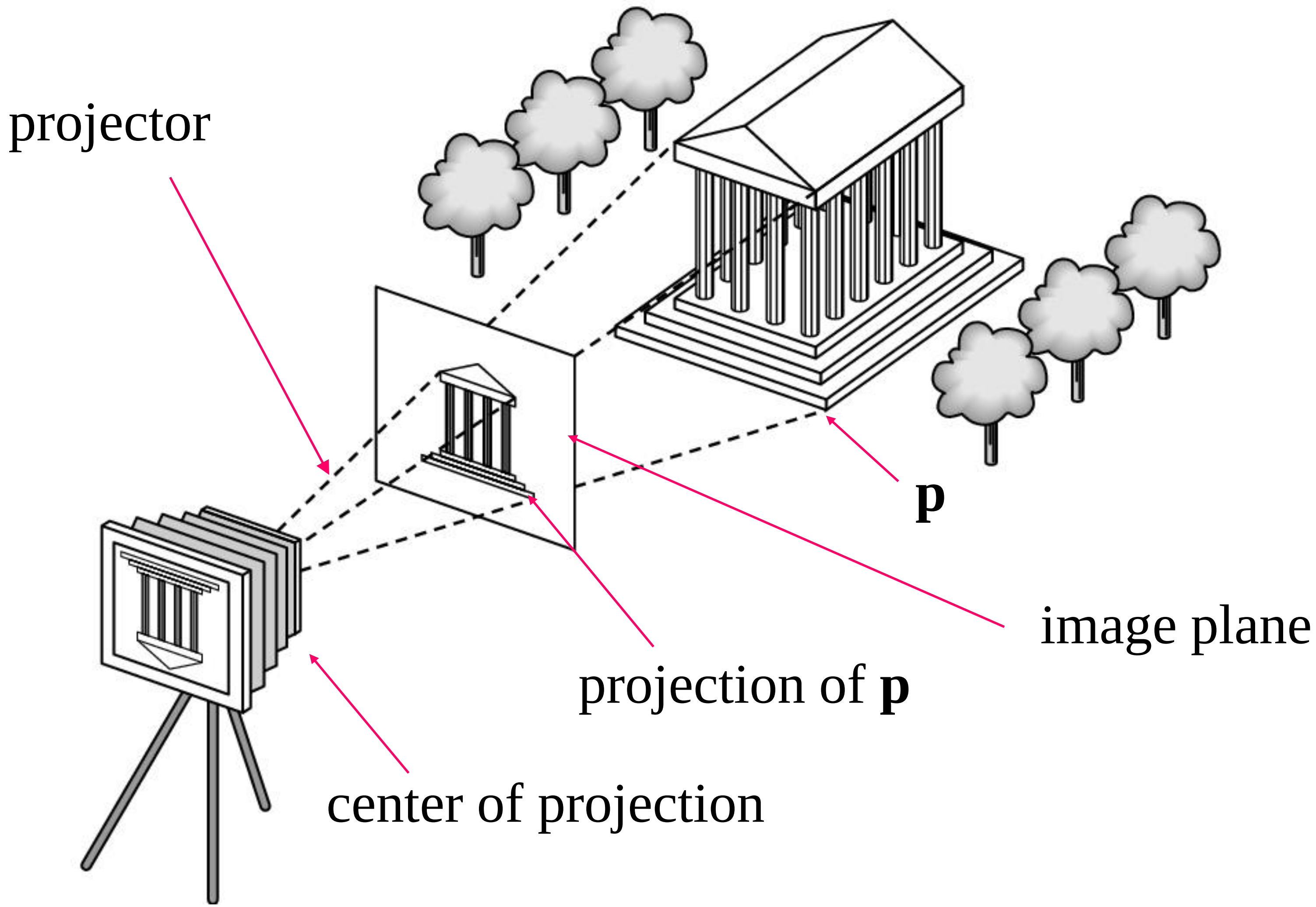
Elements of Image Formation

- Objects ✓
  - We have learned how to define objects' geometry and put them in the world coordinate frame
- Viewer
  - This lecture



- Light source(s)
- Materials

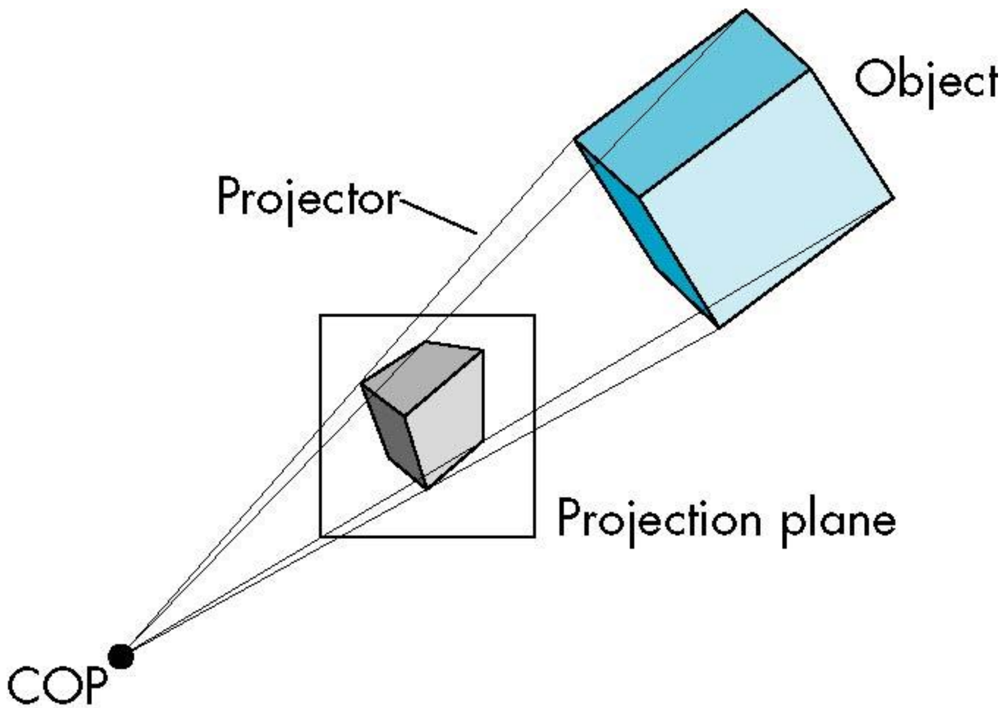
Synthetic Camera Model



Planar Geometric Projections

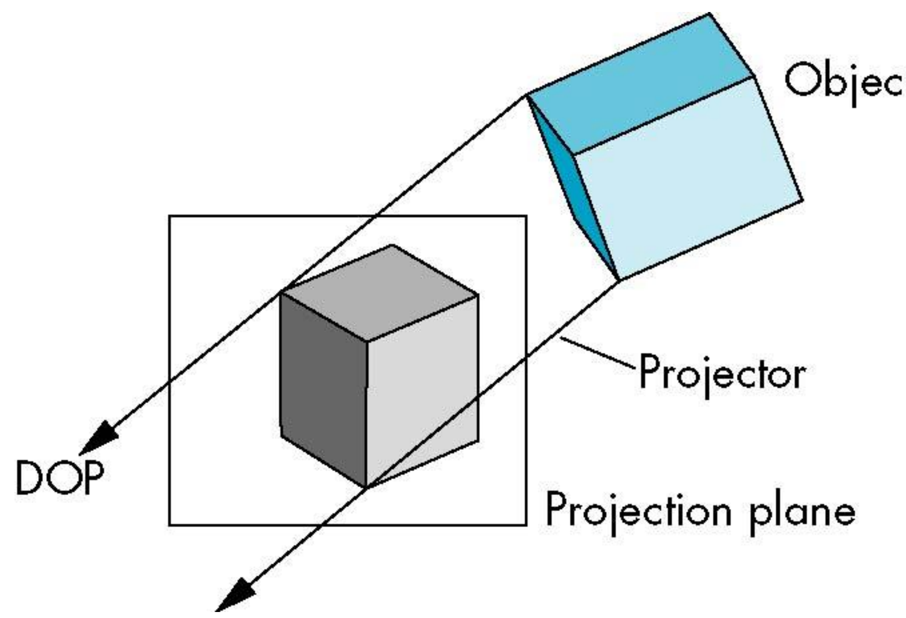
- Standard projections project onto a plane
- Projectors are lines that either
  - converge at a center of projection
  - are parallel
- Such projections preserve lines
  - but not necessarily angles
- Nonplanar projection surfaces are needed for applications such as map construction

Perspective vs Parallel



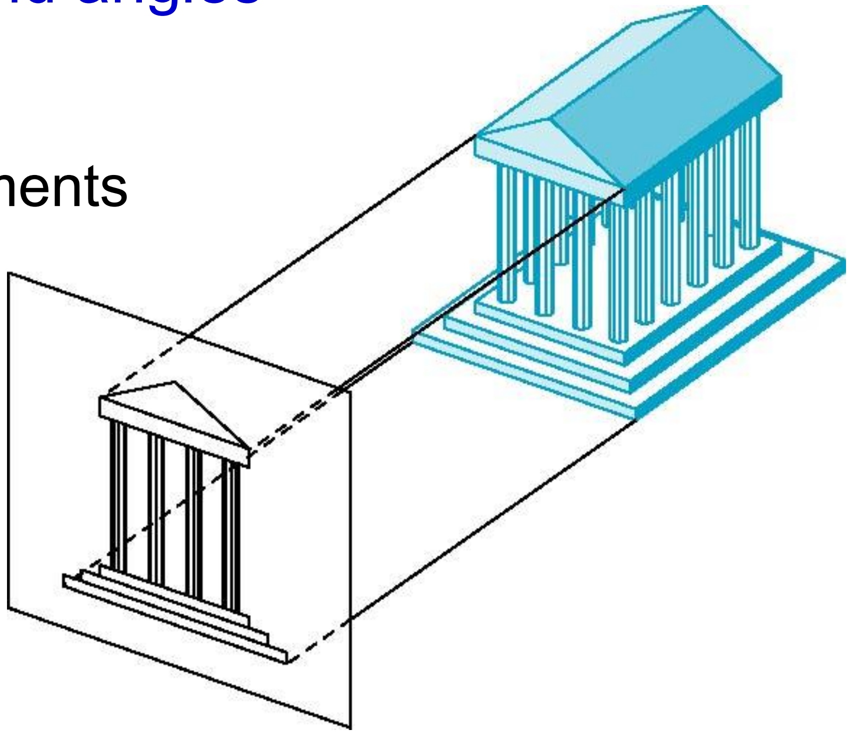
Perspective Projection

Parallel Projection



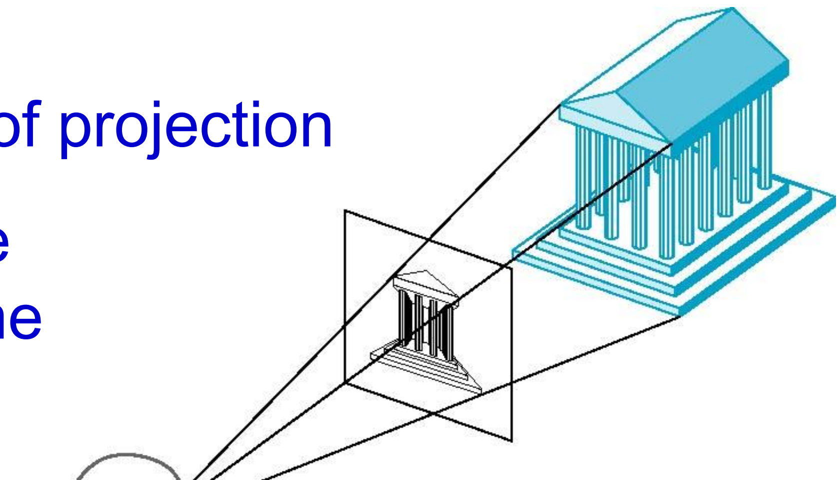
Orthographic Projection

- Special case of parallel projection
  - Projectors are orthogonal to projection surface
- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals



Perspective Projection

- Projectors converge at center of projection



- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks more realistic

- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)

- Angles preserved only in planes parallel to the projection plane

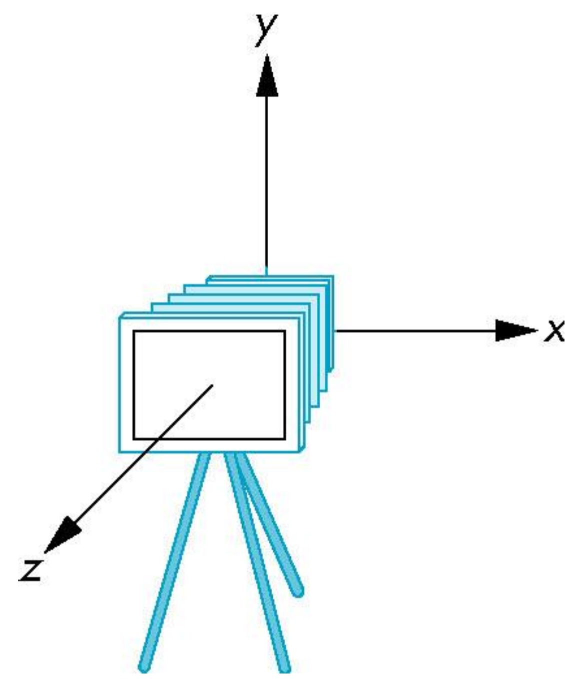
Computer Viewing

Computer Viewing

- 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
      - Perspective or orthographic
  - Clipping
    - Setting the view volume
      - Only some part of the world appears in viewport

The OpenGL Camera

- The camera has a local coordinate frame, called the *camera coordinate frame*
  - Camera is located at the origin
  - Looking in negative z direction
  - $+y$-axis is the "*up-vector*"
- All projections are w.r.t. the camera frame
- Initially the world and camera frames are the same
  - Default model-view matrix is an identity

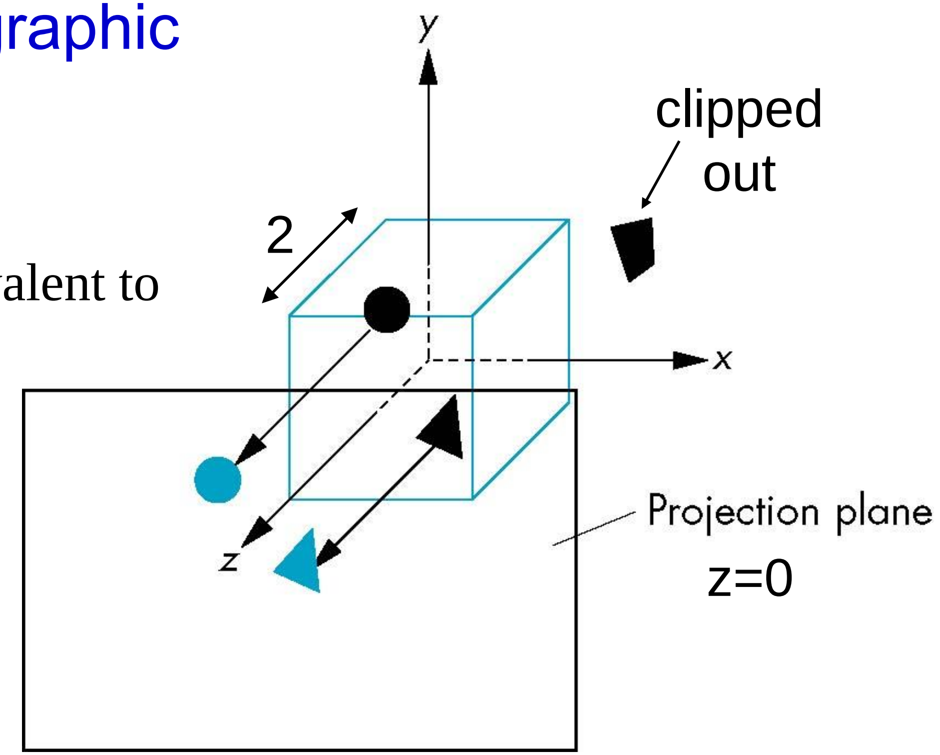


Default Projection

- 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 is orthographic

Note that the default projection is equivalent to

```
glOrtho( -1.0, 1.0,
         -1.0, 1.0,
          1.0, -1.0 );
```

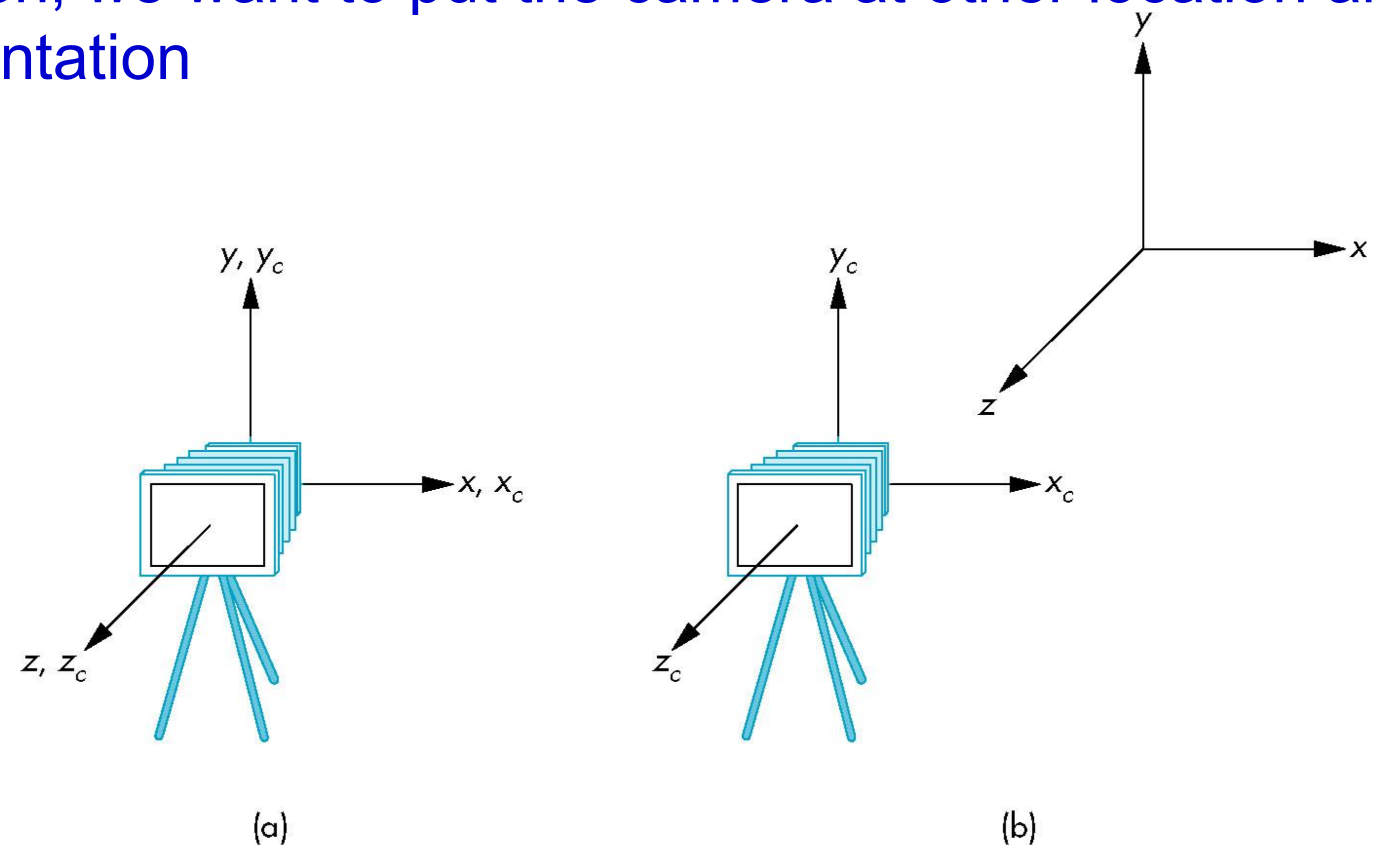


View Transformation

Positioning the Camera

Moving the Camera Frame

- By default, the camera coordinate coincides with the world coordinate frame
- Often, we want to put the camera at other location and orientation



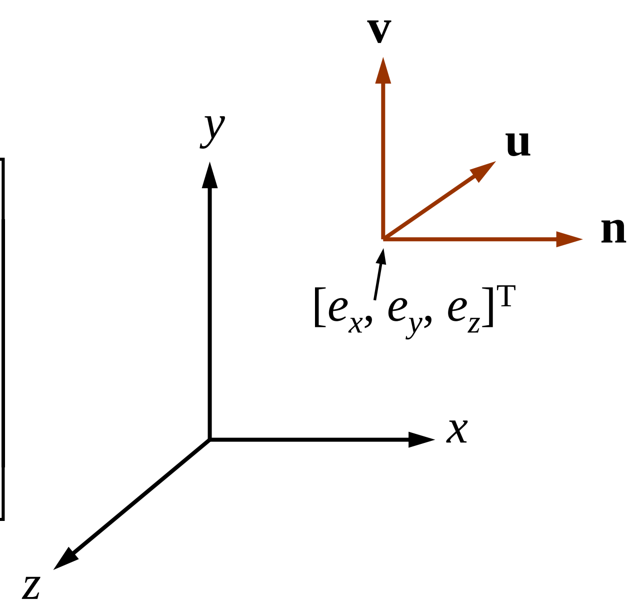
View Transformation

- In order to do projection and clipping later, all points in the world frame must be expressed w.r.t. the camera frame
  - This is called *view transformation*
  - Can be performed using a 4x4 matrix
  - It is made up of a translation first, then a rotation
    - $\mathbf{M}_{\text{view}} = \mathbf{R}\ \mathbf{T}$
  - The translation $\mathbf{T}$ moves the camera position back to the world origin
  - The rotation $\mathbf{R}$ rotates the axes of the camera frame to coincide with the corresponding axes of the world frame
  - Multiply all points in the world frame by $\mathbf{M}_{\text{view}}$ and they will be expressed w.r.t. to the camera frame

View Transformation

- Suppose the camera has been moved to the location $[e_x, e_y, e_z]^T$, and its x_c, y_c, z_c axes are the unit vectors $\mathbf{u}$, $\mathbf{v}$, $\mathbf{n}$, respectively, then

$$\mathbf{M}_{\text{view}} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- Note that $[e_x, e_y, e_z]^T$ and $\mathbf{u}$, $\mathbf{v}$, $\mathbf{n}$ are all specified w.r.t. to the world frame

View Transformation in OpenGL

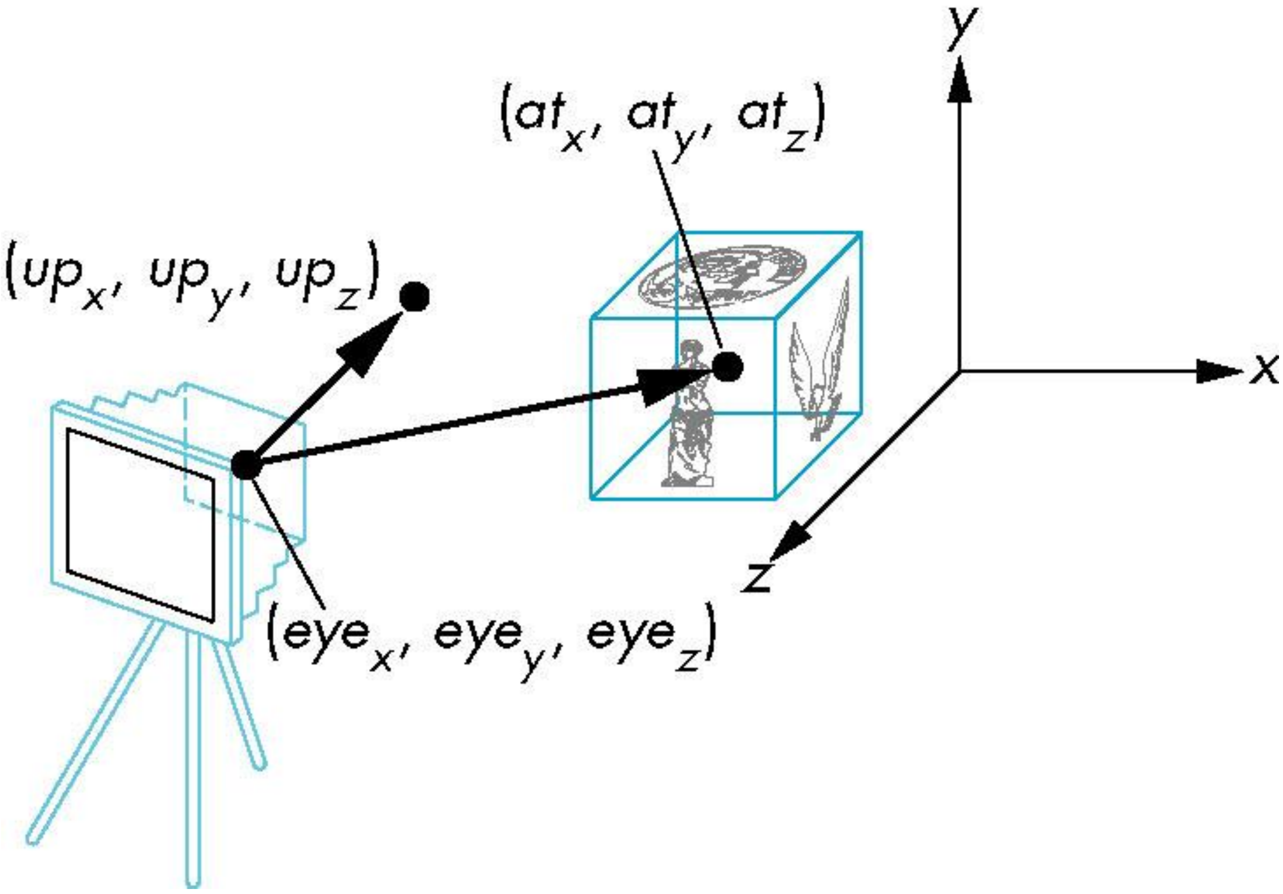
- In OpenGL, the view transformation matrix is normally the last transformation in the model-view matrix

```
glMatrixMode( GL_MODELVIEW ):
glLoadIdentity();
// specify view transformation matrix here;
...
```

- The GLU library contains the function **`gluLookAt()`** to form the required view transformation matrix through a simple interface
  - Conceptually, it positions the camera at the required location and orientation
  - Internally, it generates a view transformation matrix and post-multiplies it to the current model-view matrix

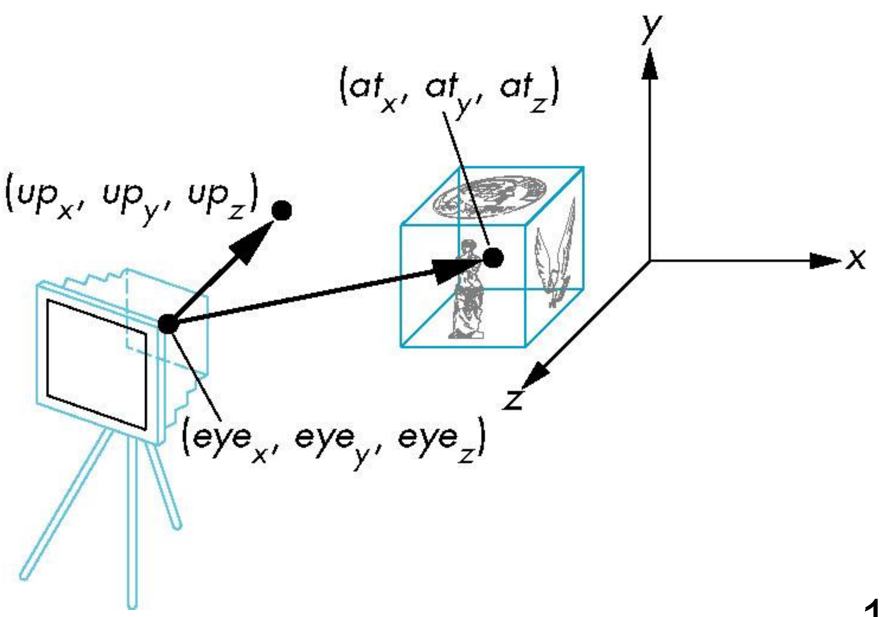
Using the `gluLookAt()` Function

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



The `gluLookAt()` Function

- Note that it does not directly specify the camera frame axes vectors **u**, **v**, **n**
- The "up-vector" may not be perpendicular to the view direction
- The vectors **u**, **v**, **n** can be derived as follows
  - **n** = normalize(**eye** – **at**)
  - **u** = normalize(**up**) × **n**
  - **v** = **n** × **u**



Projection

Defining the View Volume

OpenGL Projections

- In OpenGL, after a vertex is multiplied by the model-view matrix, it is then multiplied by the projection matrix
- The projection matrix is a 4x4 matrix that defines the type of projection
- The projection matrix can be specified by first defining a *view volume* (or *clipping volume*) in the camera frame
  - For orthographic projection, use **`glOrtho()`**
  - For perspective projection, use **`glFrustum()`**

OpenGL Projections

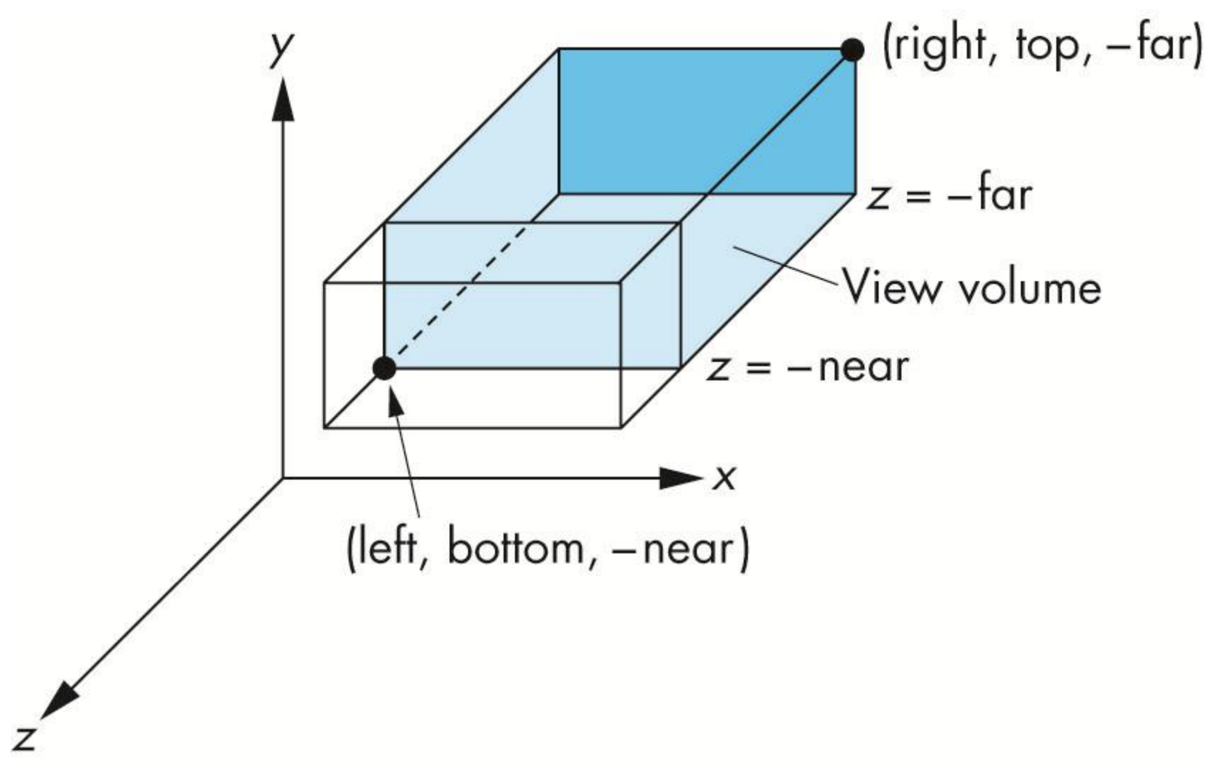
- A projection matrix is then computed such that it maps points in the view volume to a *canonical view volume*
  - The canonical view volume is the 2 x 2 x 2 cube defined by the planes $x = \pm 1$, $y = \pm 1$, $z = \pm 1$
  - Also called the *Normalized Device Coordinates* (NDC)
- The canonical view volume is then mapped to the viewport (*viewport transformation*)

Orthographic Projection

OpenGL Orthographic Projection

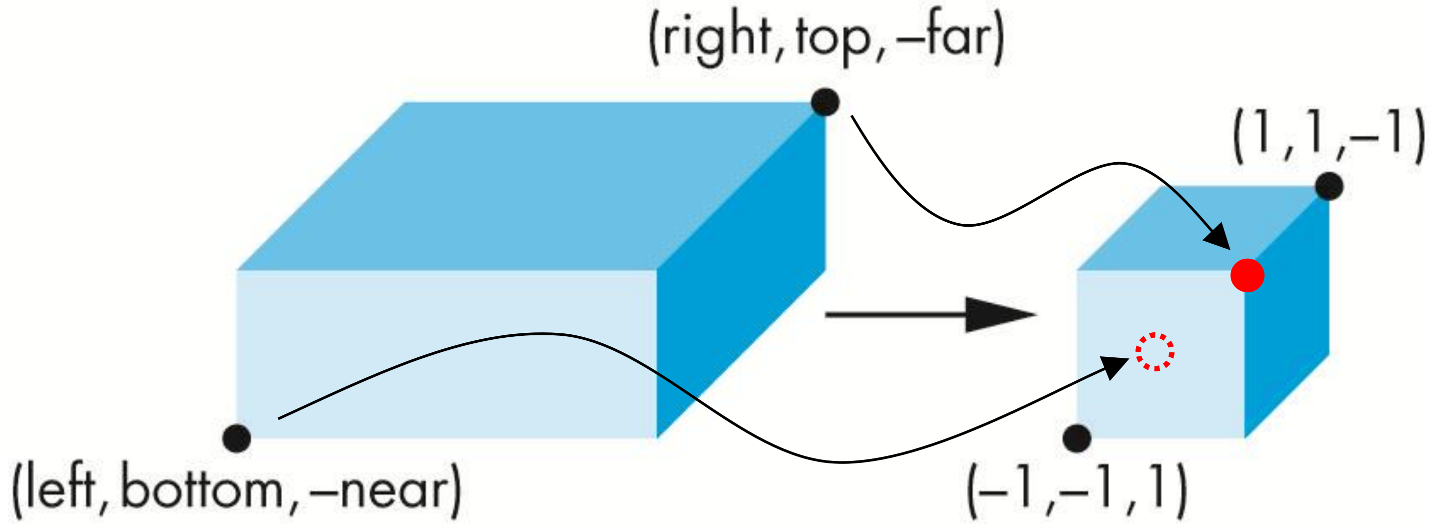
- Can be specified by defining a view volume (in the camera frame) using

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



OpenGL Orthographic Projection

- The `glOrtho()` function then generates a matrix that linearly maps the view volume to the canonical view volume, where
  - (left, bottom, –near) is mapped to (–1, –1, –1)
  - (right, top, – far) is mapped to (1, 1, 1)



Orthographic Projection Matrix

- The mapping can be found by
  - First, translating the view volume to the origin
  - Then, scaling the view volume to the size of the canonical view volume

$$\mathbf{M}_{\text{ortho}} = \mathbf{S}\left(\frac{2}{right-left}, \frac{2}{top-bottom}, \frac{2}{near-far}\right) \cdot \mathbf{T}\left(\frac{-(right+left)}{2}, \frac{-(top+bottom)}{2}, \frac{(far+near)}{2}\right)$$

  - Note that $z = -near$ is mapped to $z = -1$, and $z = -far$ to $z = +1$

Orthographic Projection Matrix

$$\mathbf{M}_{\mathrm{ortho}} = \begin{bmatrix} \dfrac{2}{right - left} & 0 & 0 & \dfrac{-(right + left)}{right - left} \\ 0 & \dfrac{2}{top - bottom} & 0 & \dfrac{-(top + bottom)}{top - bottom} \\ 0 & 0 & \dfrac{-2}{far - near} & \dfrac{-(far + near)}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewport Transformation

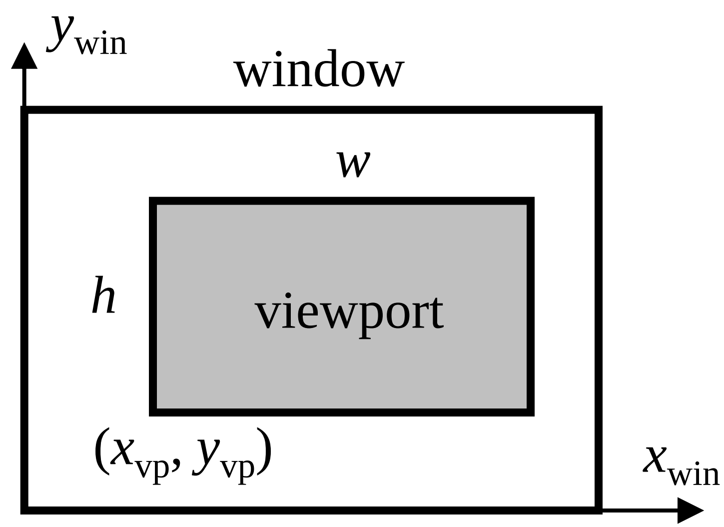
- The canonical view volume is then mapped to the viewport (from NDC to window coordinates)

$$\frac{x_{\mathrm{NDC}}-(-1)}{2}=\frac{x_{\mathrm{win}}-x_{\mathrm{vp}}}{w} \quad\Rightarrow\quad x_{\mathrm{win}}=x_{\mathrm{vp}}+\frac{w(x_{\mathrm{NDC}}+1)}{2}$$

$$\frac{y_{\mathrm{NDC}}-(-1)}{2}=\frac{y_{\mathrm{win}}-y_{\mathrm{vp}}}{h} \quad\Rightarrow\quad y_{\mathrm{win}}=y_{\mathrm{vp}}+\frac{h(y_{\mathrm{NDC}}+1)}{2}$$

$$z_{\mathrm{win}}=\frac{z_{\mathrm{NDC}}+1}{2}$$

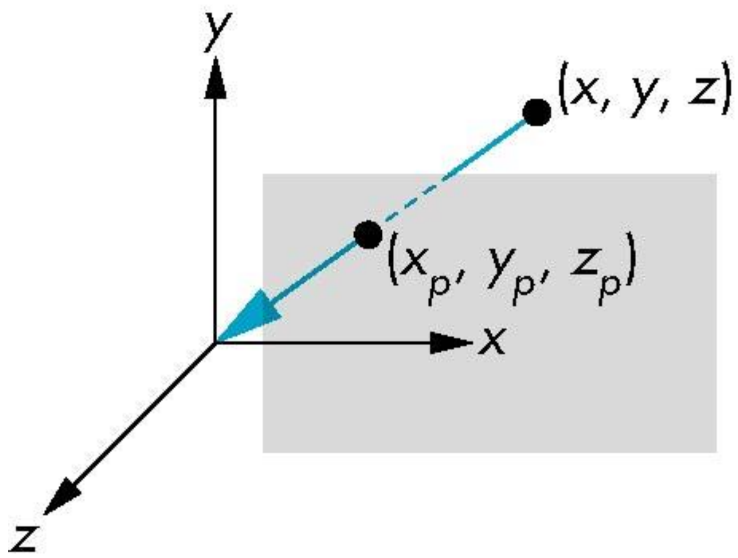
- By default, z_{win} is between 0 and 1
- It is needed for z-buffer hidden surface removal



Perspective Projection

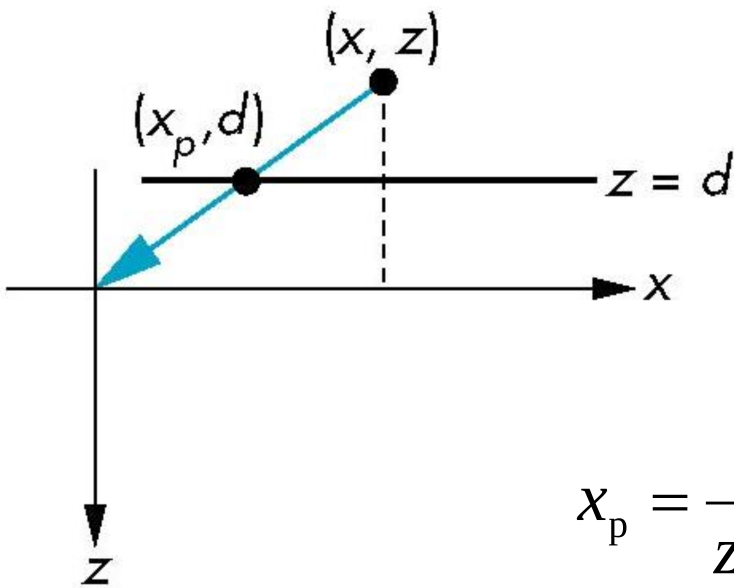
Simple Perspective Projection

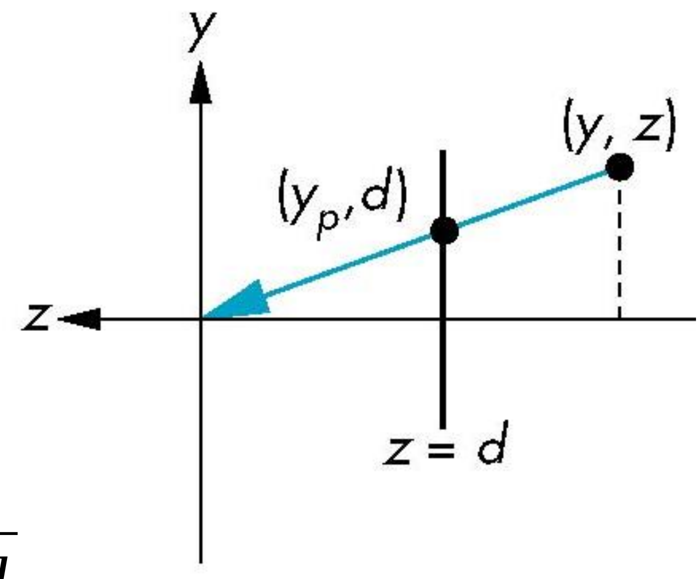
- Center of projection at the origin
- Projection plane is $z = d$, $d < 0$



Perspective Equations

- Consider top and side views





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

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

$$z_p = d$$

Using Matrix Multiplication

- Consider $\mathbf{p} = \mathbf{M}\mathbf{q}$ where

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

Perspective Division

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

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \xrightarrow{\text{perspective division}} \mathbf{p}' = \begin{bmatrix} \dfrac{x}{z/d} \\ \dfrac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

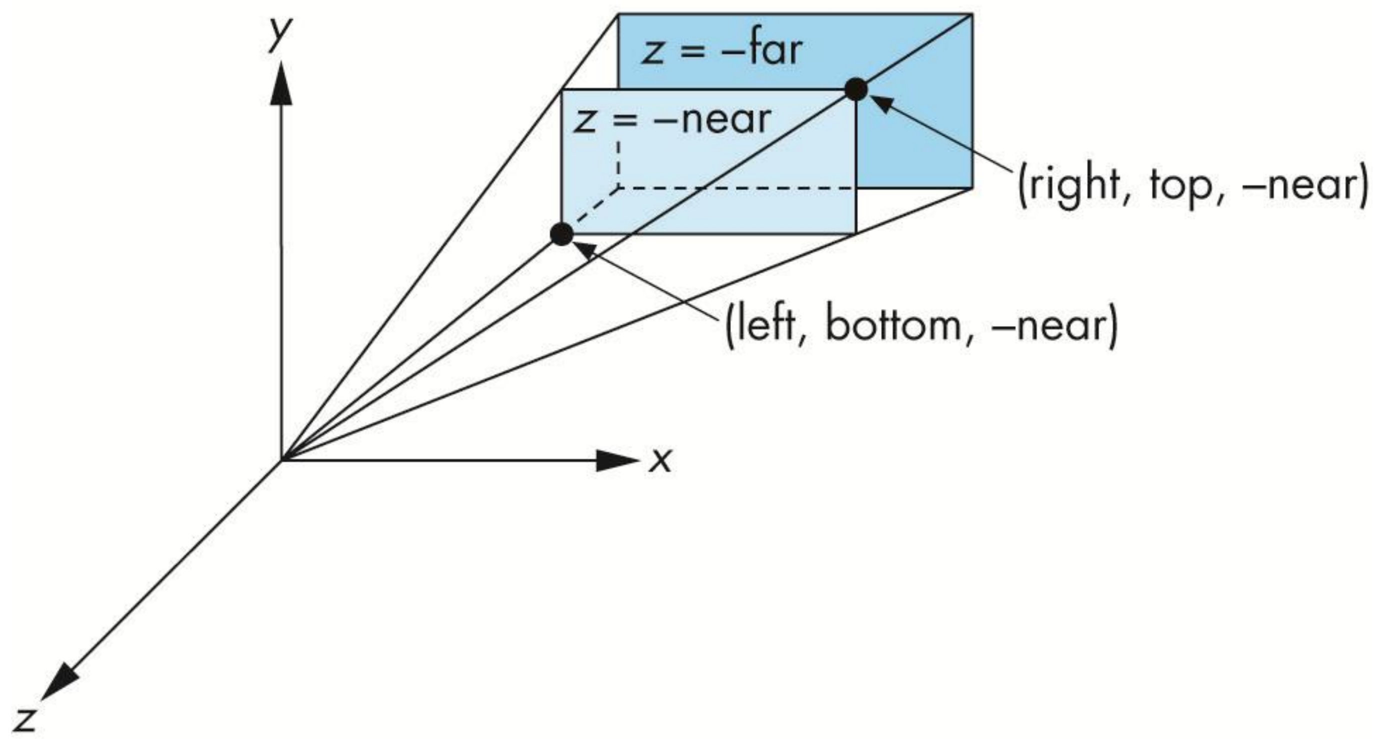
  the desired perspective equations

- This is one reason why 3D graphics API uses homogeneous coordinates

OpenGL Perspective Projection

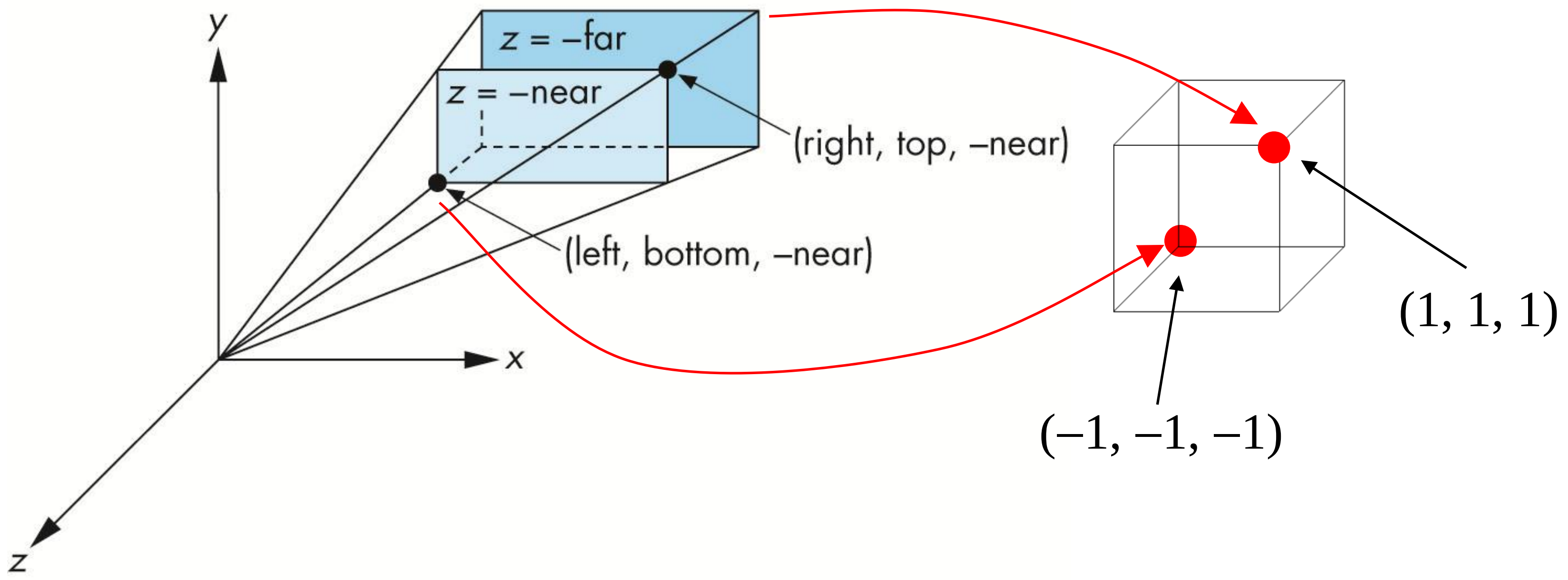
- Can be specified by defining a view volume (*view frustum*) in the camera frame using

```
glFrustum( left, right, bottom, top, near, far );
```

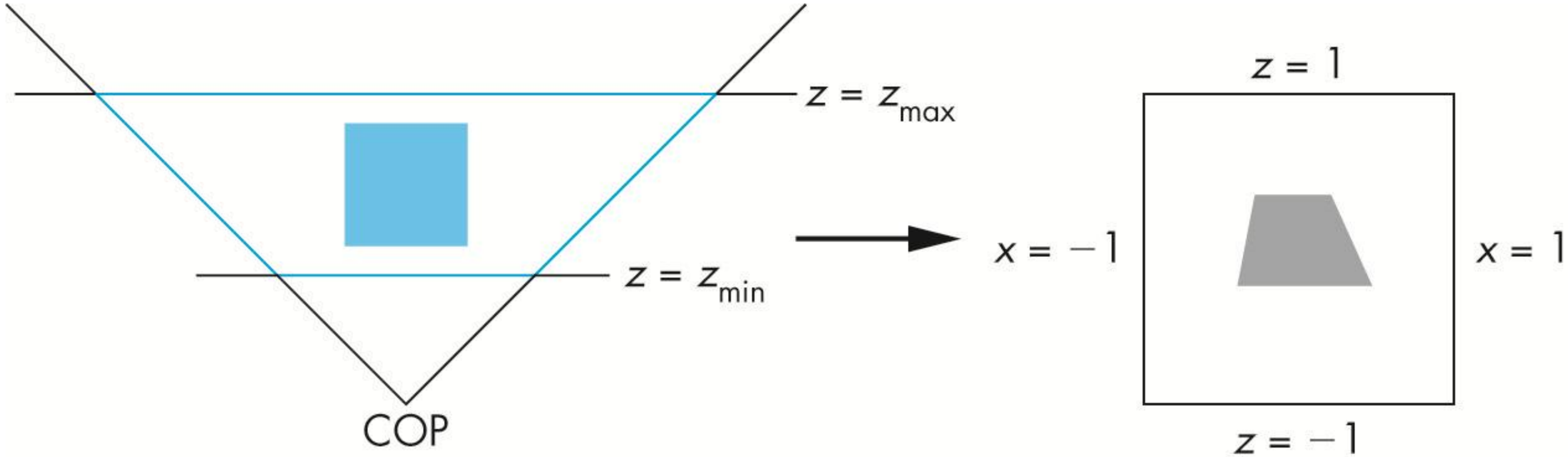


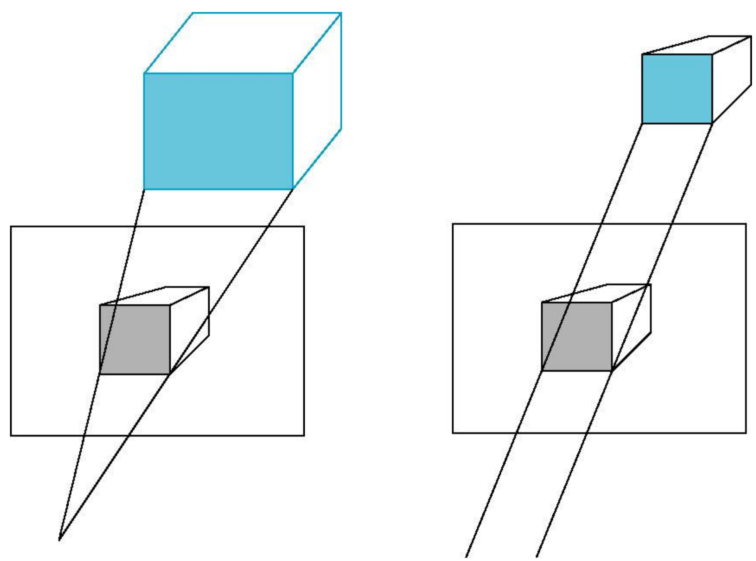
OpenGL Perspective Projection

- The **`glFrustum()`** function then generates a matrix that maps the view frustum to the canonical view volume, where



OpenGL Perspective Projection





(a) (b)

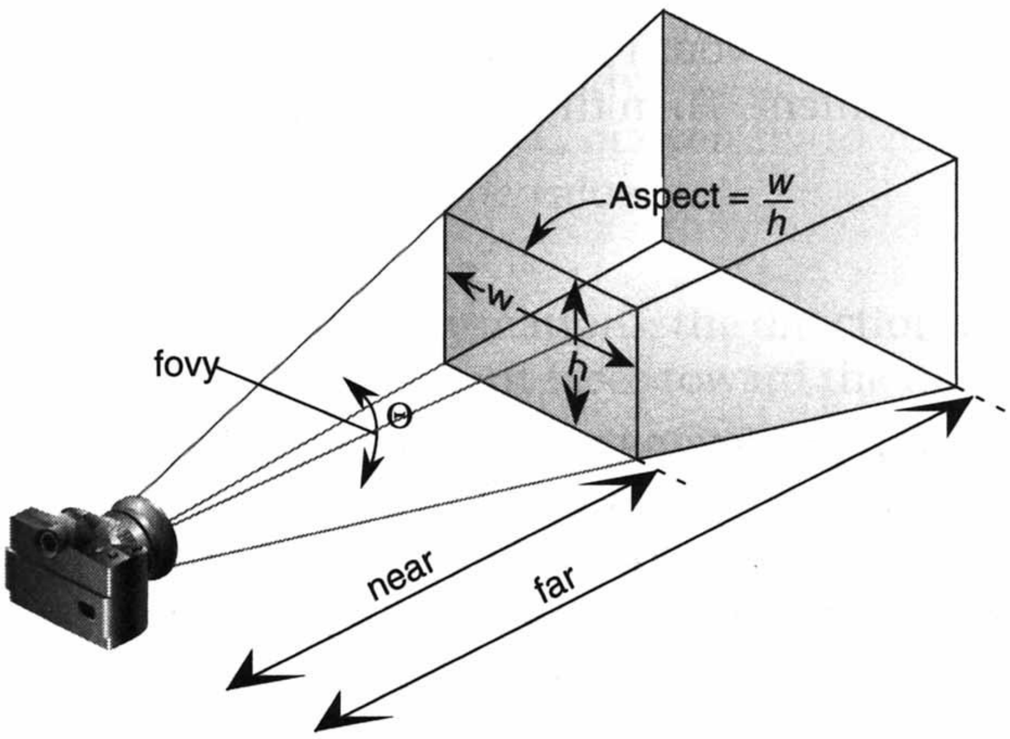
Perspective Projection Matrix

$$\mathbf{M}_{\text{persp}} = \begin{bmatrix} \dfrac{2 \cdot near}{right - left} & 0 & \dfrac{right + left}{right - left} & 0 \\ 0 & \dfrac{2}{top - bottom} & \dfrac{top + bottom}{top - bottomt} & 0 \\ 0 & 0 & \dfrac{-\left(far + near\right)}{far - near} & \dfrac{-2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

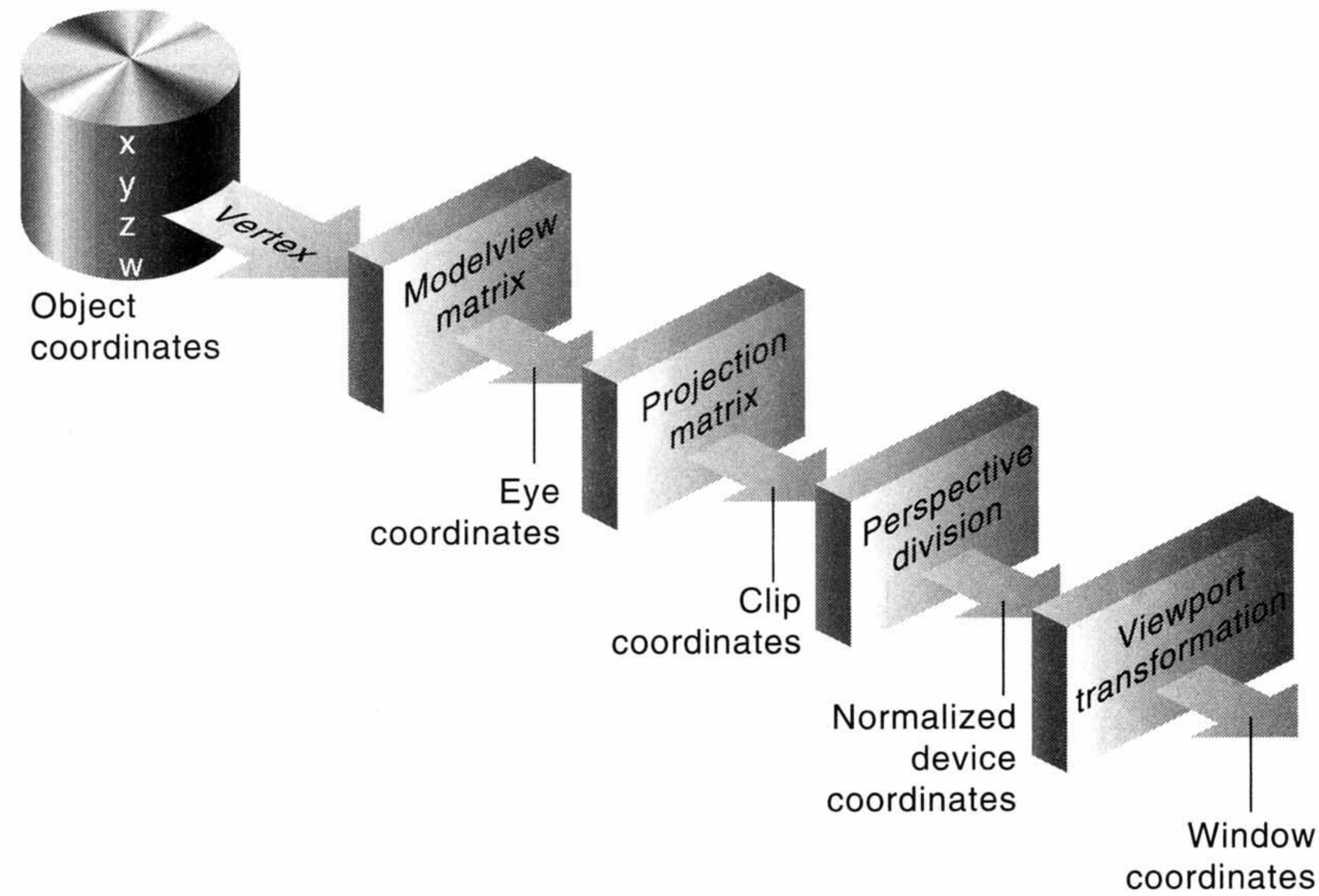
Perspective Projection Using Field of View

- The **`glFrustum()`** function allows non-symmetric view
- More often, we want to specify a symmetric view volume
- We can use

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



The OpenGL Transformation Stages



End of Lecture 5