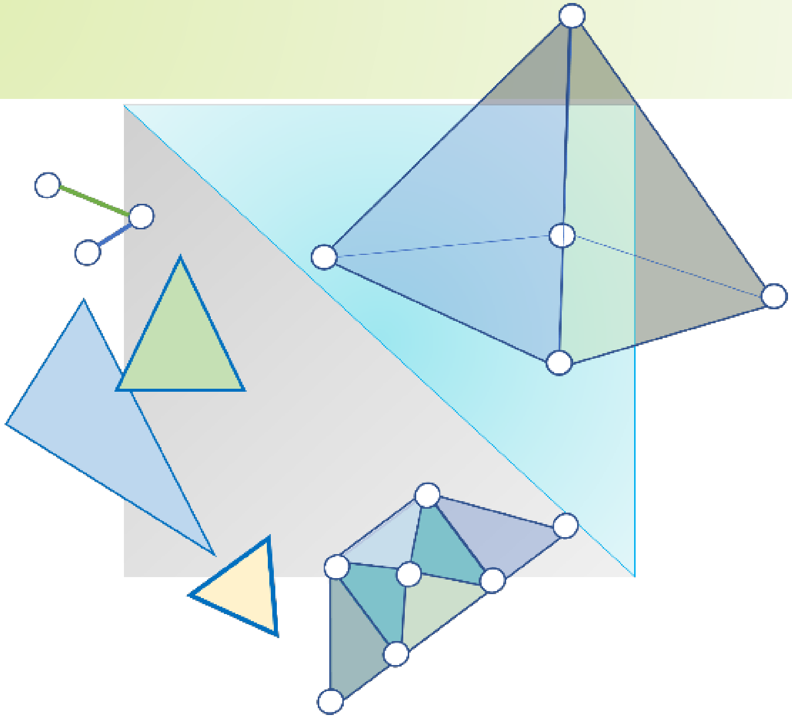
CITS3003 Graphics & Animation



Lecture 12:
Computer Viewing

Objectives

- Introduce OpenGL viewing functions
  - Learn how to place the camera
- Introduce the mathematics of projection
  - Learn how to define orthographic and perspective projection
  - gluLookAt(), glOrtho(), glFrustum(), gluPerspective() and their mat.h counterparts
- Introduce glMatrixMode()

Computer Viewing

- There are three aspects of the viewing process, all of which are implemented in the pipeline,

1. Positioning the camera
   - Setting the model-view matrix
2. Selecting a lens
   - Setting the projection matrix
3. Clipping
   - Setting the view volume

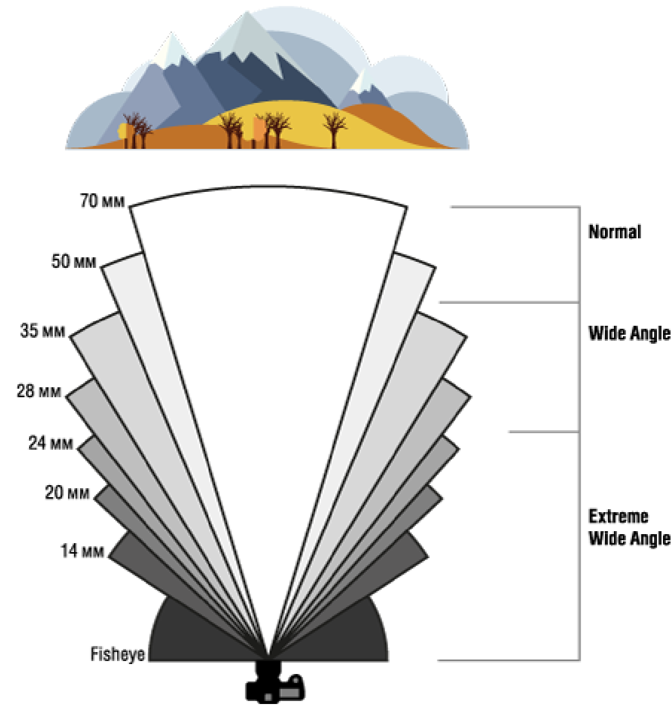
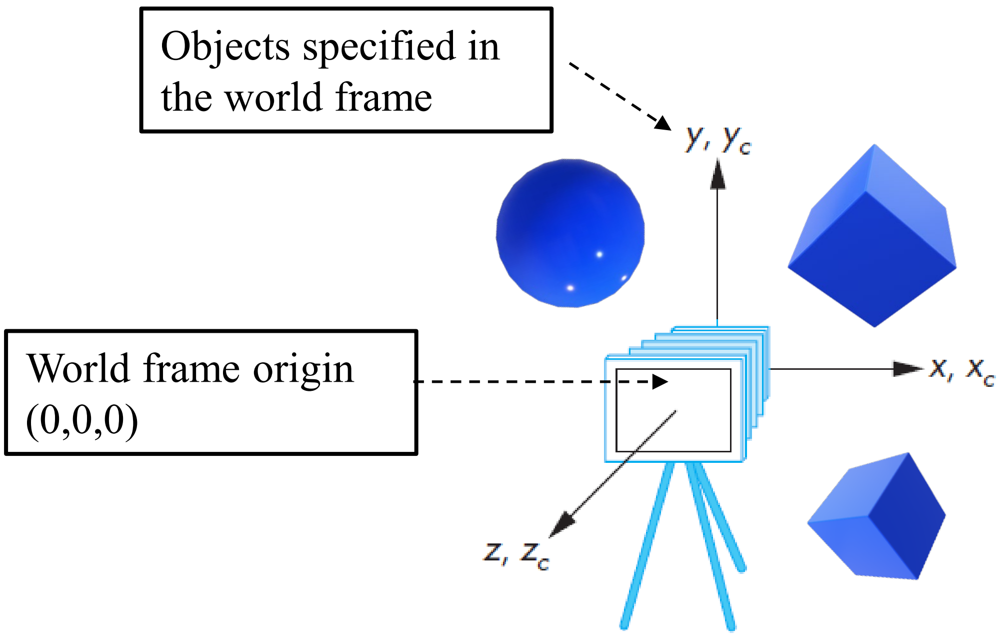


Image credits : https://www.bhphotovideo.com/explora/photography/tips-and-solutions/faq-wide-angle-lenses

The OpenGL Camera

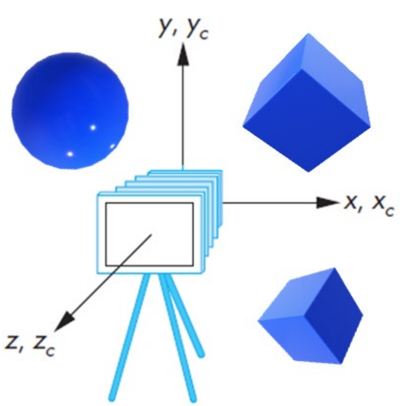
- In OpenGL, initially the object and camera frames are the same
  - The default model-view matrix is an identity
- The camera is located at the origin and points in the negative z direction



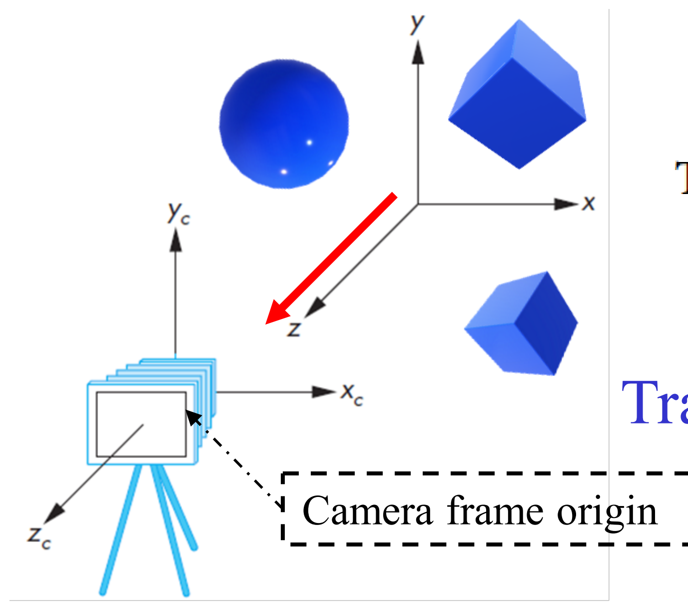
Moving the Camera Frame

default frames

frames after translation by d
where $d > 0$



Default Frames



Translate the camera in
+z direction

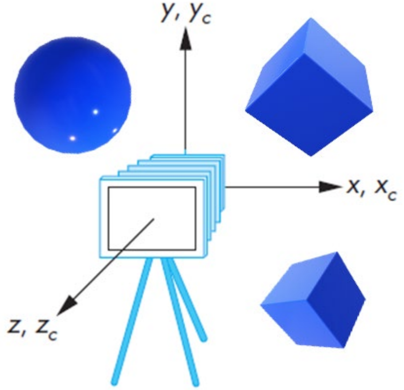
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate(0.0,0.0,-d);

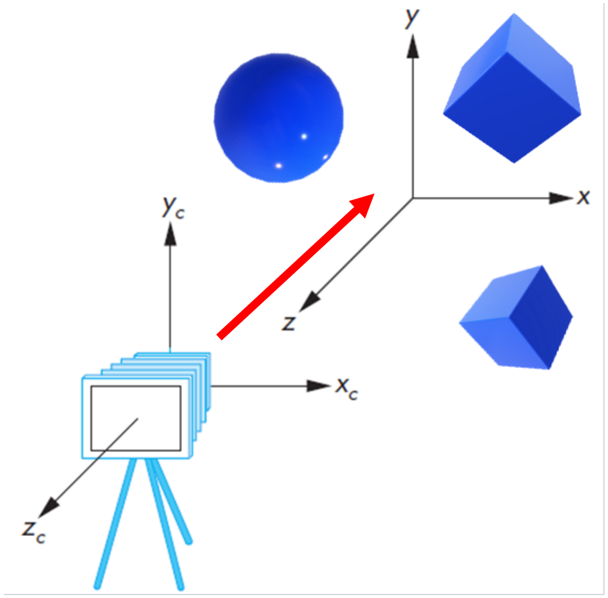
Moving the Camera Frame

We can move the objects in the $-z$ direction

- **Moving the world frame**



Default Frames

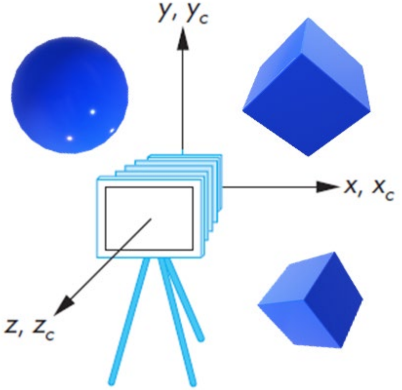


Translate the objects in -z direction

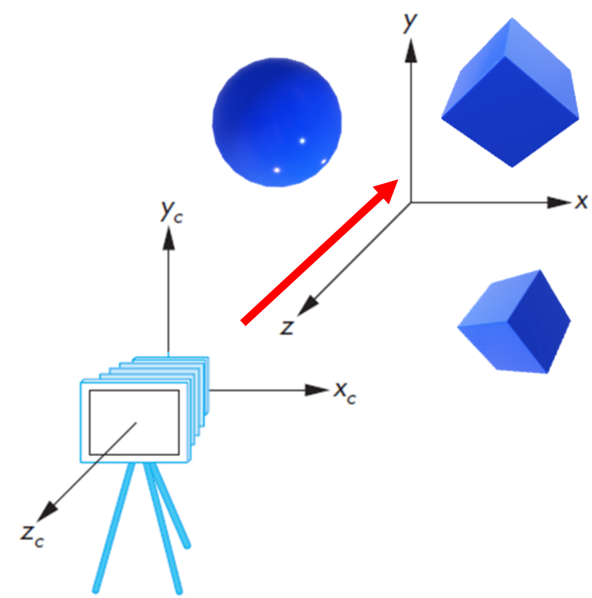
Moving the Camera Frame

- If we want to visualize objects that have both positive and negative z –values we can either
  - Move the objects in the negative z direction
    - Translate the world frame
  - Move the camera in the positive z direction
    - Translate the camera frame

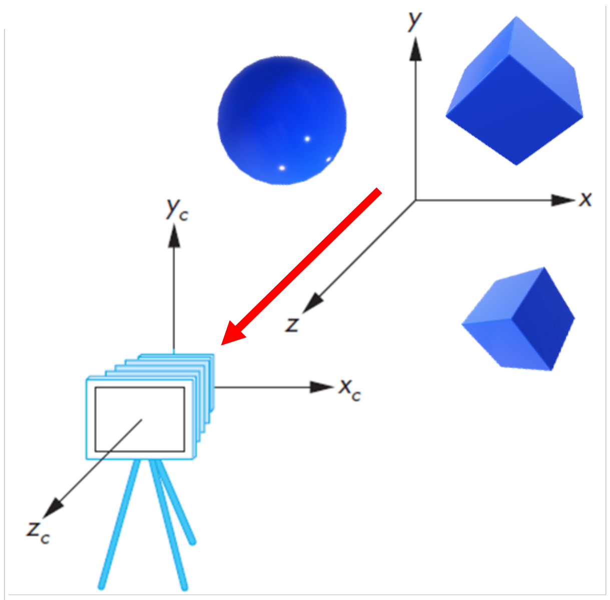
Both of these views are equivalent and are determined by the model-view matrix



Default Frames



Translate the objects in -z direction



Translate the camera in +z direction

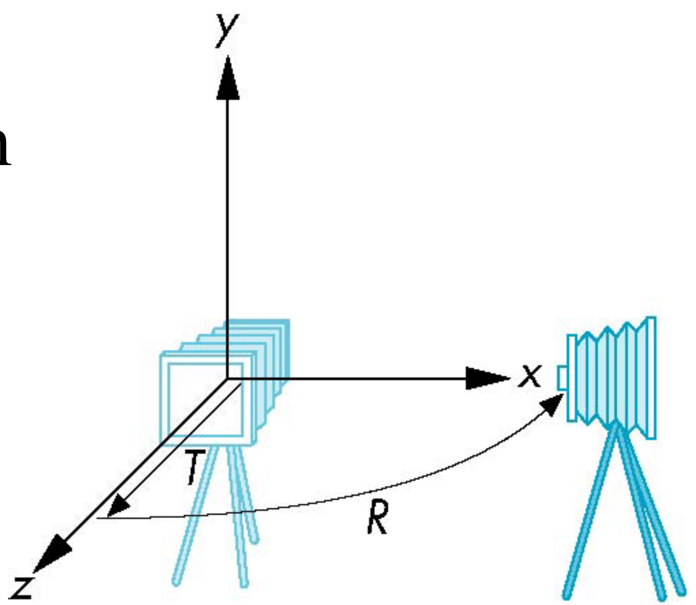
Moving the Camera

We can move the camera to any desired position by a sequence of rotations and translations

Example: side view at the $+x$ axis looking towards the origin

1. Rotate the camera
2. Move it away from origin

Model-view matrix $M = TR$



Moving the Camera – OpenGL code

- Remember that the last transformation specified is first to be applied

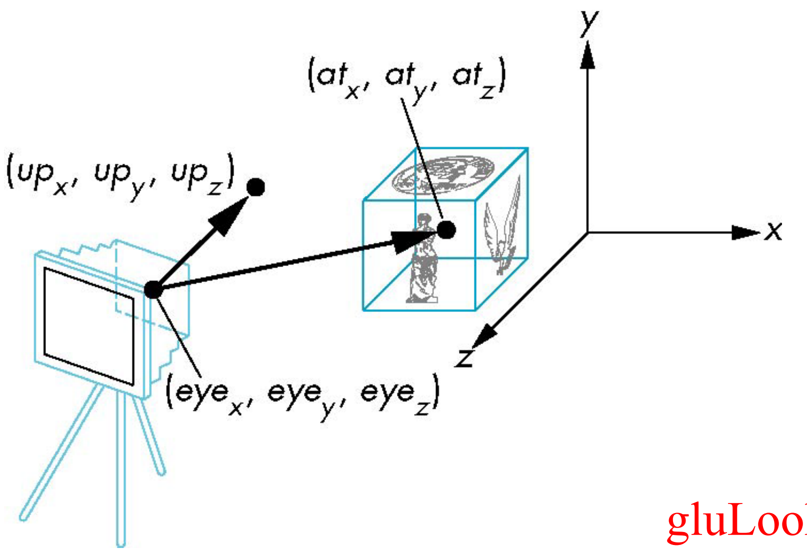
```
// Using mat.h

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

The LookAt() Function

- The GLU library contains the function gluLookAt which can be used to form the required model-view matrix.

void gluLookAt(eyeX, eyeY, eyeZ, centreX, centreY, centreZ, upX, upY, upZ)

- We need to define the eye (camera) position, the centre (fixation point), and an up direction. All are of type GLdouble.



Programmer defines:

- **eye** position
- LookAtpoint **(at)** and
- **Up**vector (***Up***direction usually (0,1,0))

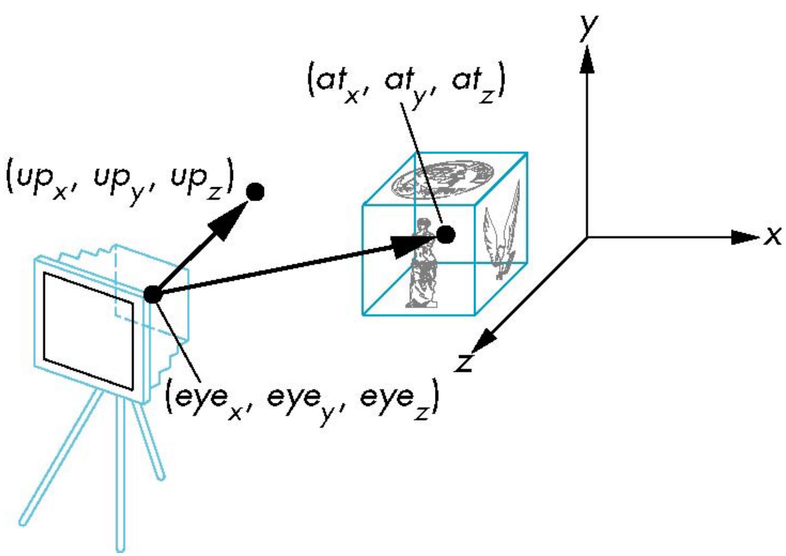
gluLookAt deprecated!

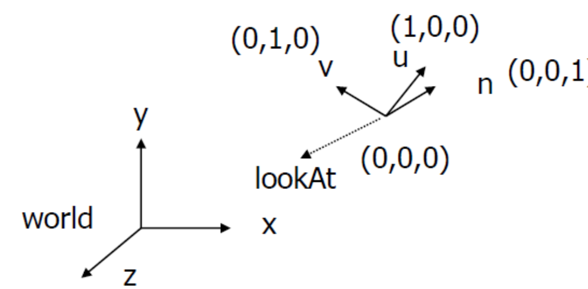
The LookAt() Function

- Alternatively, we can use LookAt() defined in mat.h
  - The function returns a mat4 matrix.
  - Can concatenate with modeling transformations
- Example:

Type: GLfloat

```
mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
```





The LookAt() Function:

- Forms camera (u,v,n) frame
  - **n** away from the view volume,
  - **v** is the cross product of n and up vector,
  - **u** at right angles to both **n** and **v**
- Compose matrix to transform coordinates (object to camera)

Other Camera Viewing Controls

- The LookAt() function is only for positioning the camera

- Other ways to specify camera position are:
  - Yaw, pitch, roll (angles)
  - Elevation, azimuth, twist (angles)

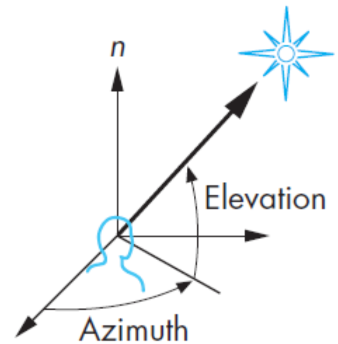
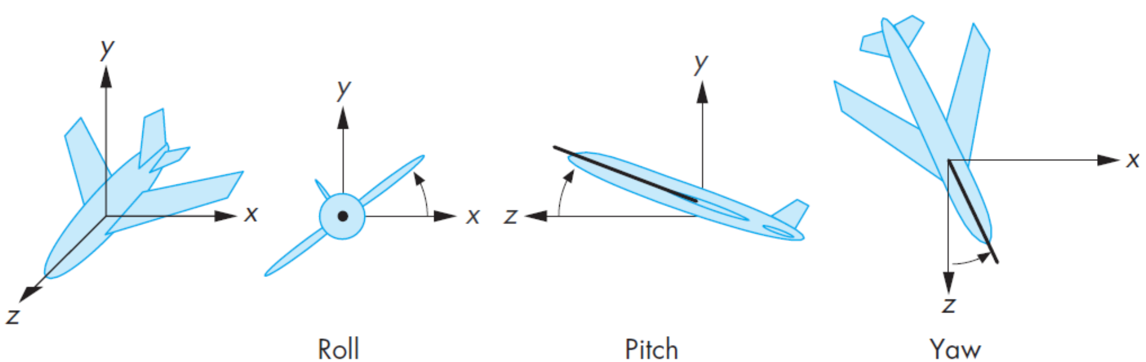
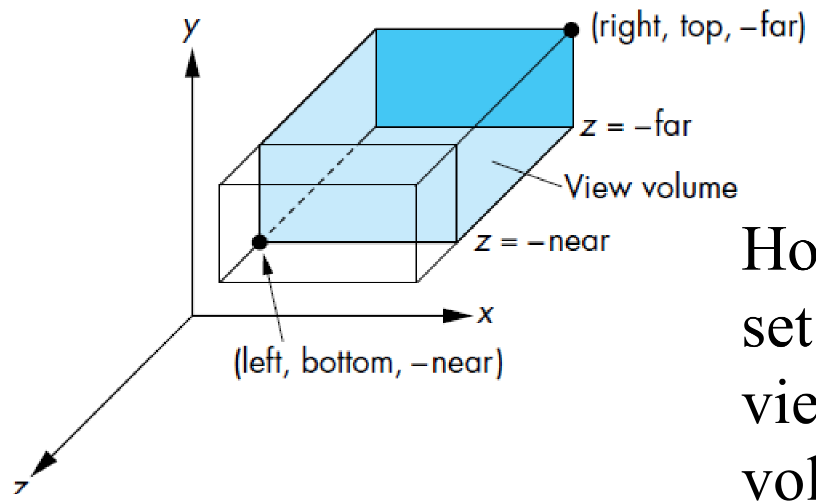


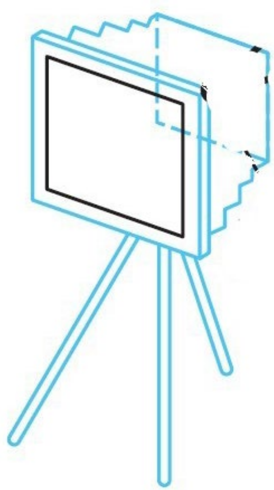
FIGURE 4.20 Elevation and azimuth.



3D Viewing and View Volume

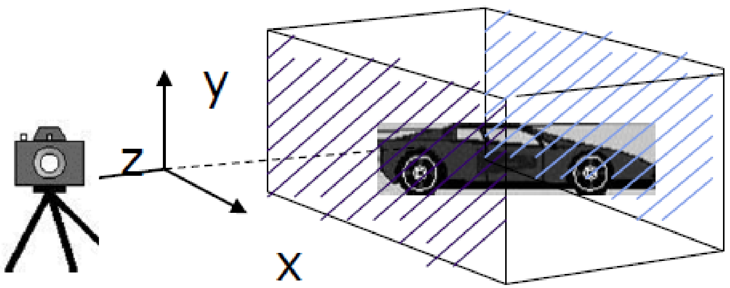


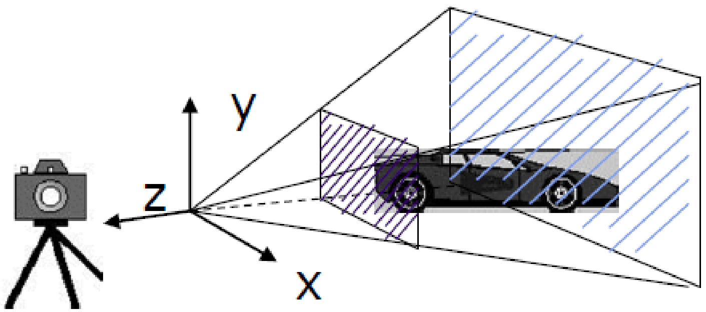
How do we set the viewing volume?



Previously we set the camera position

Different View Volumes





Orthogonal View Volume   *Perspective View Volume*

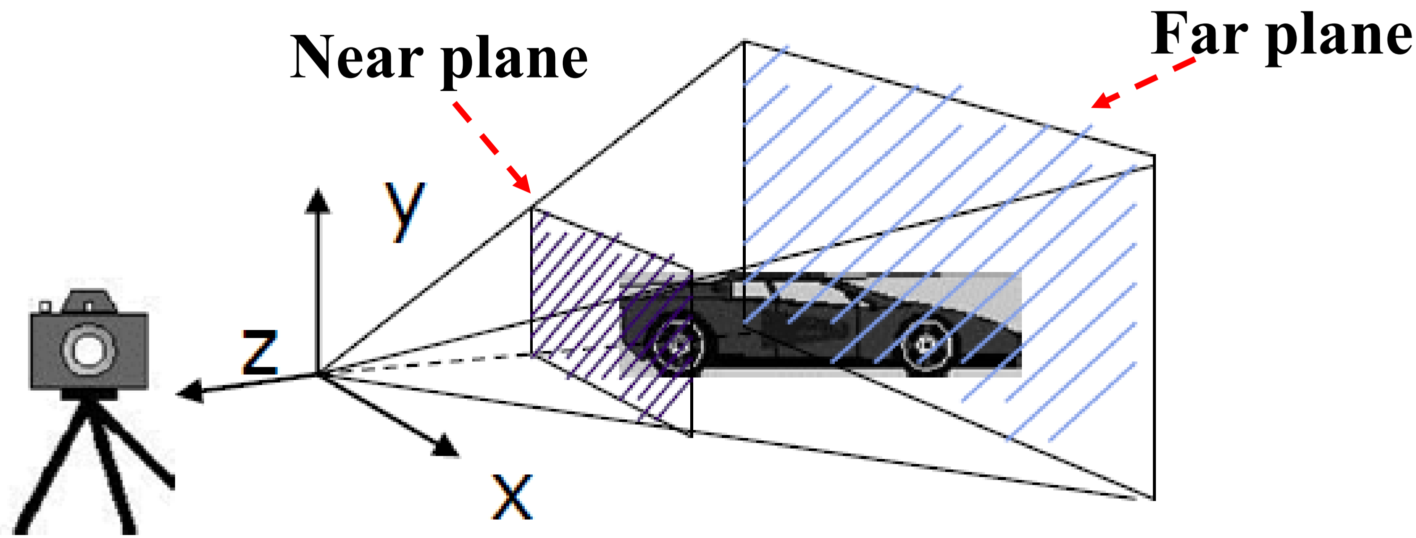
Different view volume leads to different look

View volume parameters:

- Projection: Perspective, orthographic etc.,
- Near and far clipping planes- only the objects b/w near and far planes appear on the image
- Field of view – determines how much of the world is captured in the picture
- Aspect ratio- w/h of the near plane

Viewing Frustrum

Near plane + far plane + field of view = **Viewing Frustum**

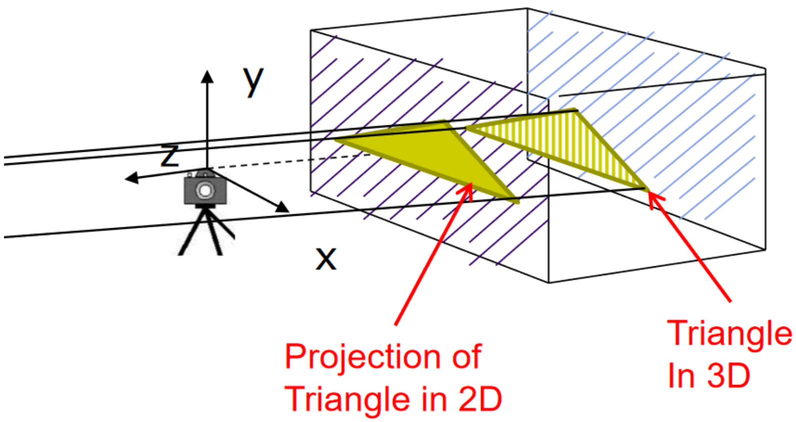


Objects outside the viewing frustrum are clipped

Default Orthographic Projection

- The default projection in the eye (camera) frame is orthogonal

How to find the orthographic projection of a 3D object on a projection plane?



- Draw parallel lines from each object vertex to the projection plane.
- The projection center is at infinite
- Use (x,y) coordinates, just drop z coordinates

In orthographic projection, the projection lines are parallel to each other and perpendicular to the projection plane. Because there is no convergence of light rays in orthographic projection, the concept of focal length is not applicable

Image Credits: Emmanuel Agu, WPI, CS 543

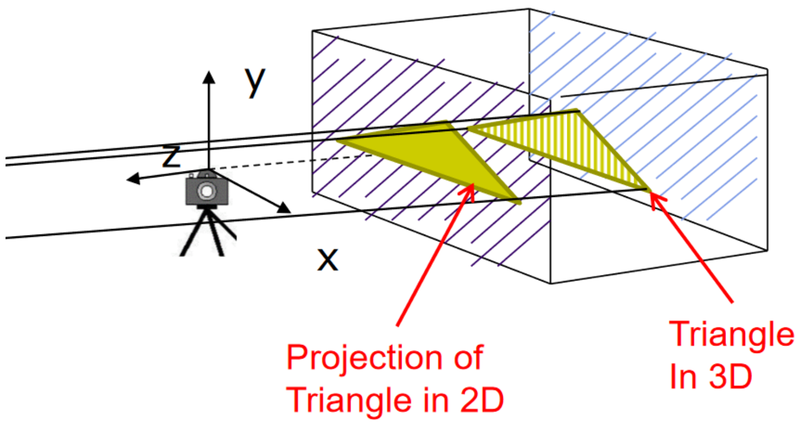
Default Orthographic Projection

- The default projection in the eye (camera) frame is orthogonal
- For a point $\mathbf{p} = (x, y, z, 1)^{\mathrm{T}}$ within the default view volume, it is projected to $\mathbf{p}_p = (x_p, y_p, z_p, w_p)^{\mathrm{T}}$, where

$$x_p = x,\quad y_p = y,\quad z_p = 0, w_p = 1$$

- i.e., we can define
  - $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

  and we can then write $\mathbf{P}_p = \mathbf{Mp}$

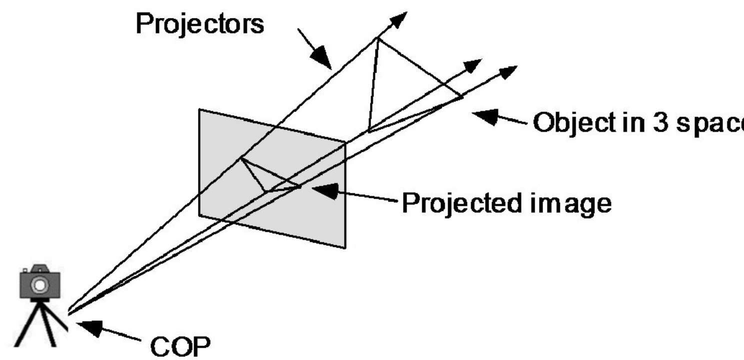


- In practice, we can let $\mathbf{M} = \mathbf{I}$ and then set z to 0

Image Credits: Emmanuel Agu, WPI, CS 543

Simple Perspective

In perspective projection, the camera's focal length d is finite

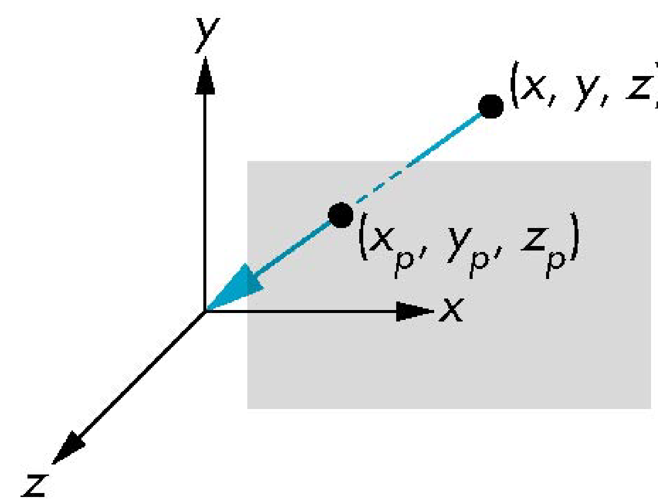


A simple perspective projection:

Center of projection is at the origin

Projection plane $z = d$, where $d < 0$

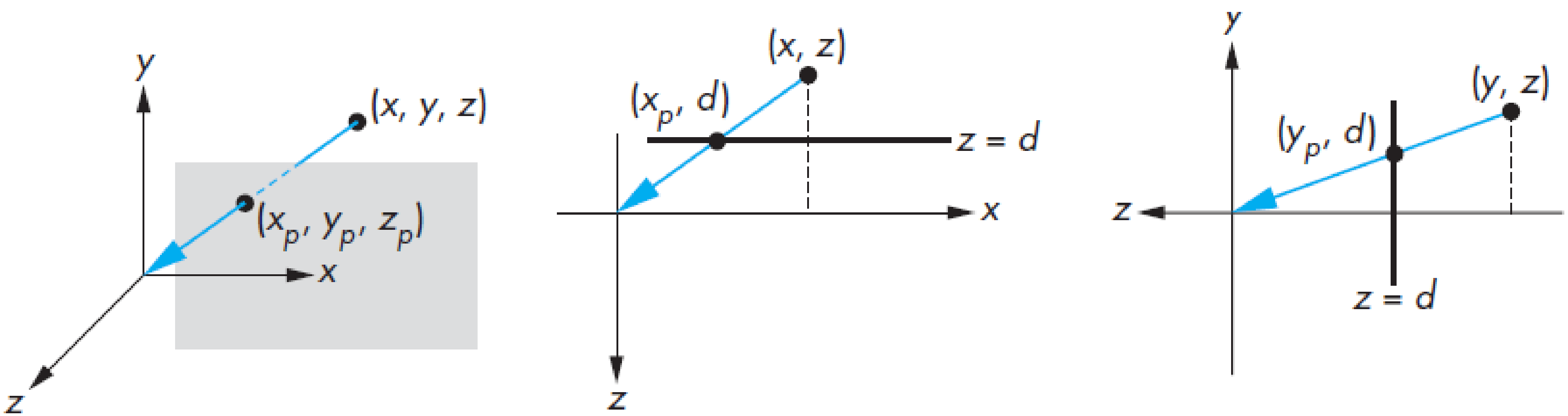
How to find the perspective projection of a 3D object on a projection plane?



- Draw line from object to projection center
- Calculate where each intersects projection plane

Simple Perspective (cont.)

Consider the top and side views



(a) (b) (top view) (c) (side view)

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

i.e., $x_p = \frac{x}{z/d}$

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

i.e., $y_p = \frac{y}{z/d}$

$z_p = d$

Recall: the OpenGL synthetic camera model in an earlier lecture

Simple Perspective (cont.)

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

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

$$\Rightarrow \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

In OpenGL, this is the w term

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

Perspective Division

- Since $w = z/d \neq 1$, so we must divide by w to return back to our three-dimensional space.

- This ***perspective division*** yields

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

which are the desired perspective equations, as on slide 20.

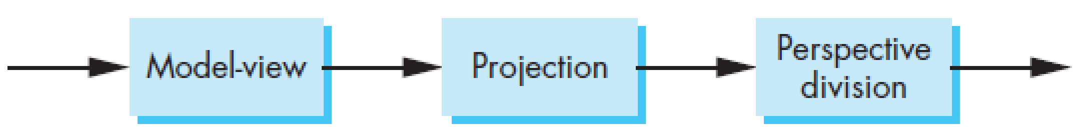


FIGURE 4.33 Projection pipeline.

Orthogonal Viewing

- The OpenGL orthogonal viewing function is:

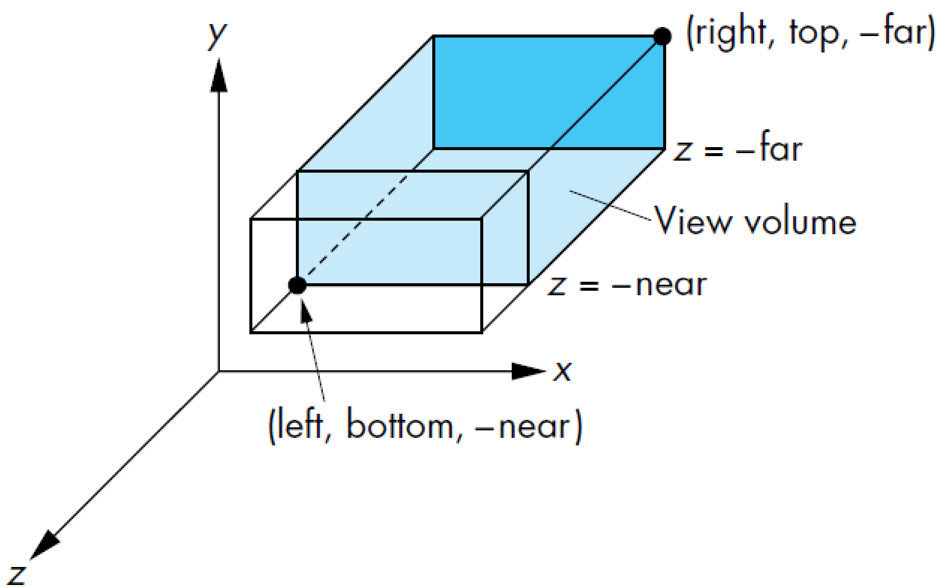
  void glOrtho(left, right, bottom, top, near, far)

- Alternatively, we can use Ortho() defined in mat.h:

```
mat4 Ortho(left,right,bottom,top,near,far)
```

Type: GLdouble

Type: GLfloat

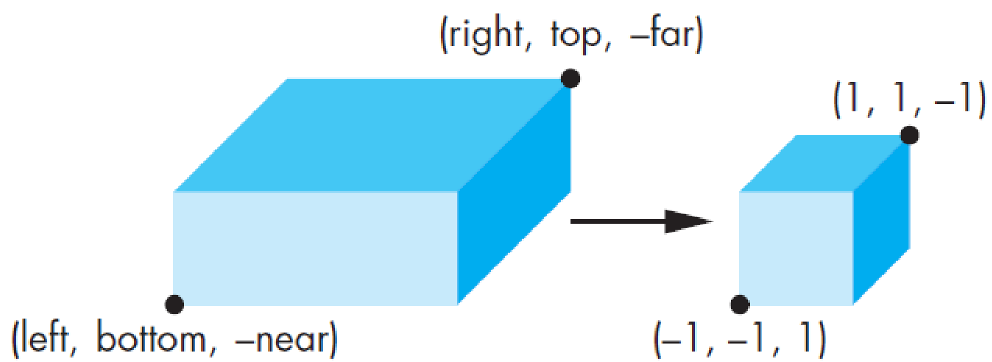


near and **far** are measured from camera

Orthogonal Normalization

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

normalization ⇒ find transformation to convert specified clipping volume to default



Orthogonal Matrix

- Two steps
  - Move center to origin

    T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))
  - Scale to have sides of length 2

    S(2/(left-right),2/(top-bottom),2/(near-far))

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

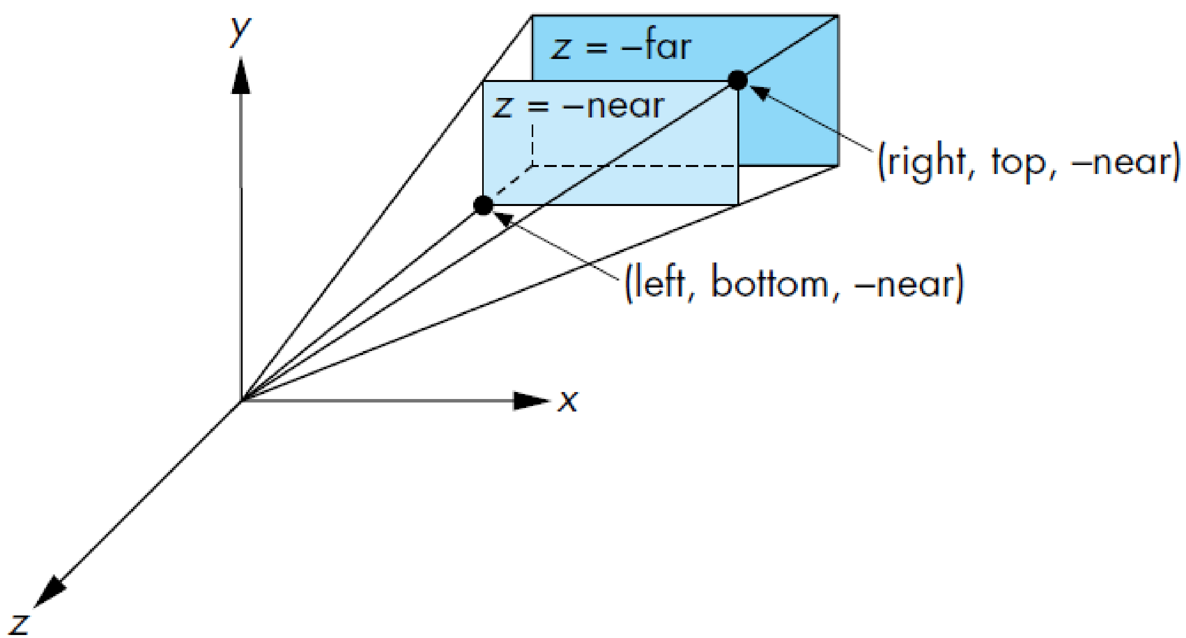
Perspective Viewing

- To define a perspective transformation matrix for the camera, we can use

```
mat4 Frustum(left,right,bottom,top,near,far)
```

defined in mat.h:

All are of type GLfloat

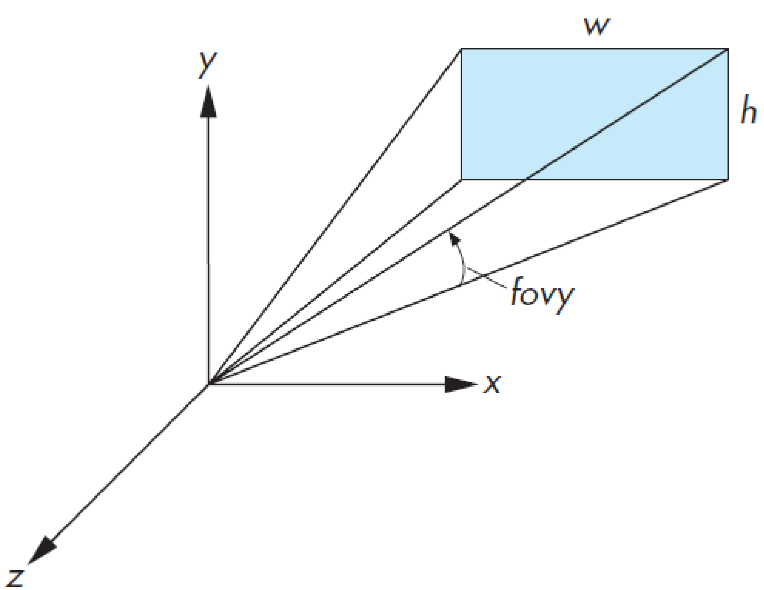


Perspective Viewing with "Field of View"

- Another way to get perspective projection is:

  `mat4 Perspective(fovy, aspect, near, far)`

  which often provides a better interface

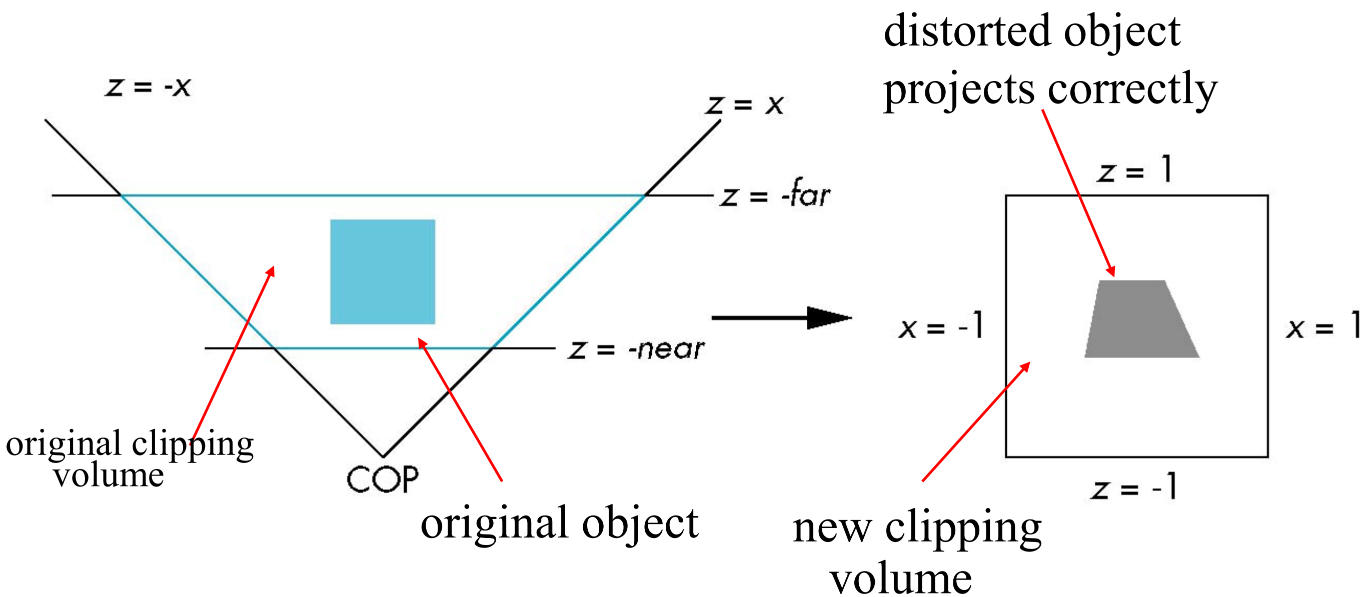


All are of type GLfloat

Note:
`aspect = w/h`
`fovy` is an angle in degrees

The angle *fovy* is the angle between the top and bottom planes of the clipping volume.

Perspective Normalization



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

projection matrix corresponding to Frustum(left,right,bottom,top,near,far)

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

projection matrix corresponding to Persective(fovy, aspect, near, far)

The Complete Viewing Pipeline

- Model (orient individual objects)
- View (orient the camera OR the entire world)
- Projection

$$P * V * M_i * O_i$$

- There is one projection, one camera but there could be many objects O_i and hence M_i where $i = 1,2,3, \dots n$

The Complete Viewing Pipeline

- Model (orient individual objects)
- View (orient the camera OR the entire world)
- Projection

$$P * V * M_i * O_i$$

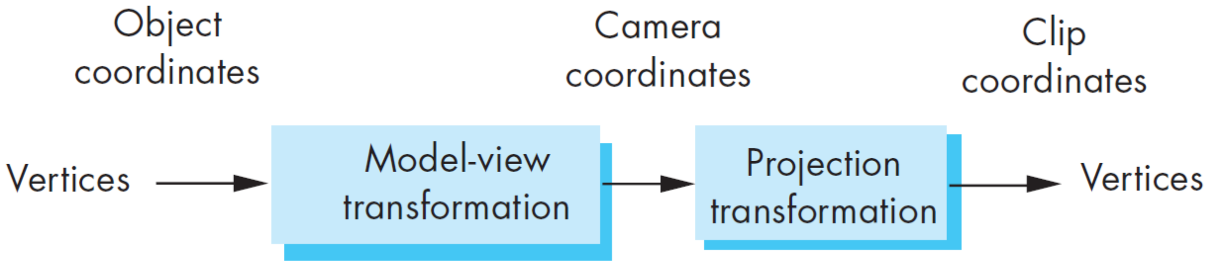


FIGURE 4.11 Viewing transformations.

- The model-view matrix will take vertices in object coordinates and convert them to a representation in camera coordinates.
- The projection matrix will both carry out the desired projection—either orthogonal or perspective—and convert a viewing volume specified in camera coordinates to fit inside the viewing cube in clip coordinates.

gluLookAt(), glOrtho(), glFrustum(), and gluPerspective()

- Did you notice that…

- The "gl" and "glu" versions have no return arguments

- Whereas the mat.h versions LootAt(), Ortho(), Frustum() and Perspective() return 4x4 matrices of type mat4

glMatrixMode()

- Recall that OpenGL is a state machine

  Legacy OpenGL maintains several matrices for transforming points in 3D space

- glMatrixMode() defines the current matrix
  - GL_MODELVIEW
  - GL_PROJECTION
  - GL_TEXTURE
  - GL_COLOR

- glGet(GL_MATRIX_MODE) will return the current matrix mode

glMatrixMode()

- When you define MODELVIEW with gluLookAt()

  - OR

- When you define PROJECTION with glOrtho(), glFrustum(), or gluPerspective()

- The current matrix is multiplied by the new matrix

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity() /*clear the matrix*/
glFrustrum(-1.0, -1.0, -1.0, 1.5. 20.0)
```

Further Reading

"Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL" by Edward Angel and Dave Shreiner, 6^{th} Ed, 2012

- Secs. 4.1. Classical and Computer Viewing; 4.1.2. Orthographic Projections; 4.1.5 Perspective Viewing
- Sec. 4.2. Viewing with a Computer
- Sec. 4.3.1. Positioning of the Camera Frame; 4.3.3. The Look-At Function
- Sec. 4.4.1. Orthographic Projections; 4.4.2. Parallel Viewing with OpenGL; 4.4.4. Orthogonal-Projection Matrices; (optional) 4.4.6 An Interactive Viewer
- Secs. 4.5. – 4.7. Projections – Perspective-Projection Matrices