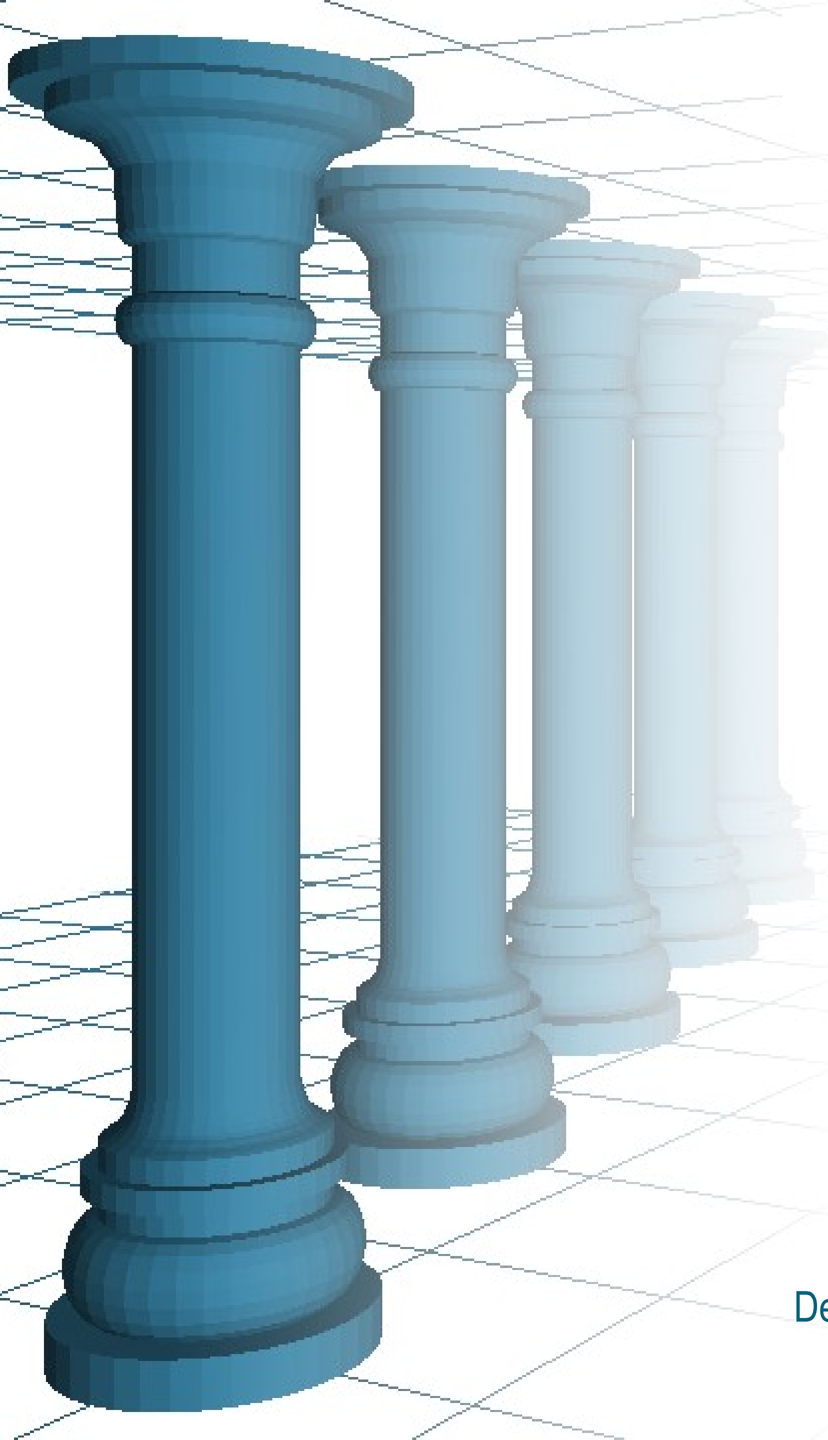


COSC363 Computer Graphics

6

Viewing and Projection

There's more to a scene than meets the eye

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

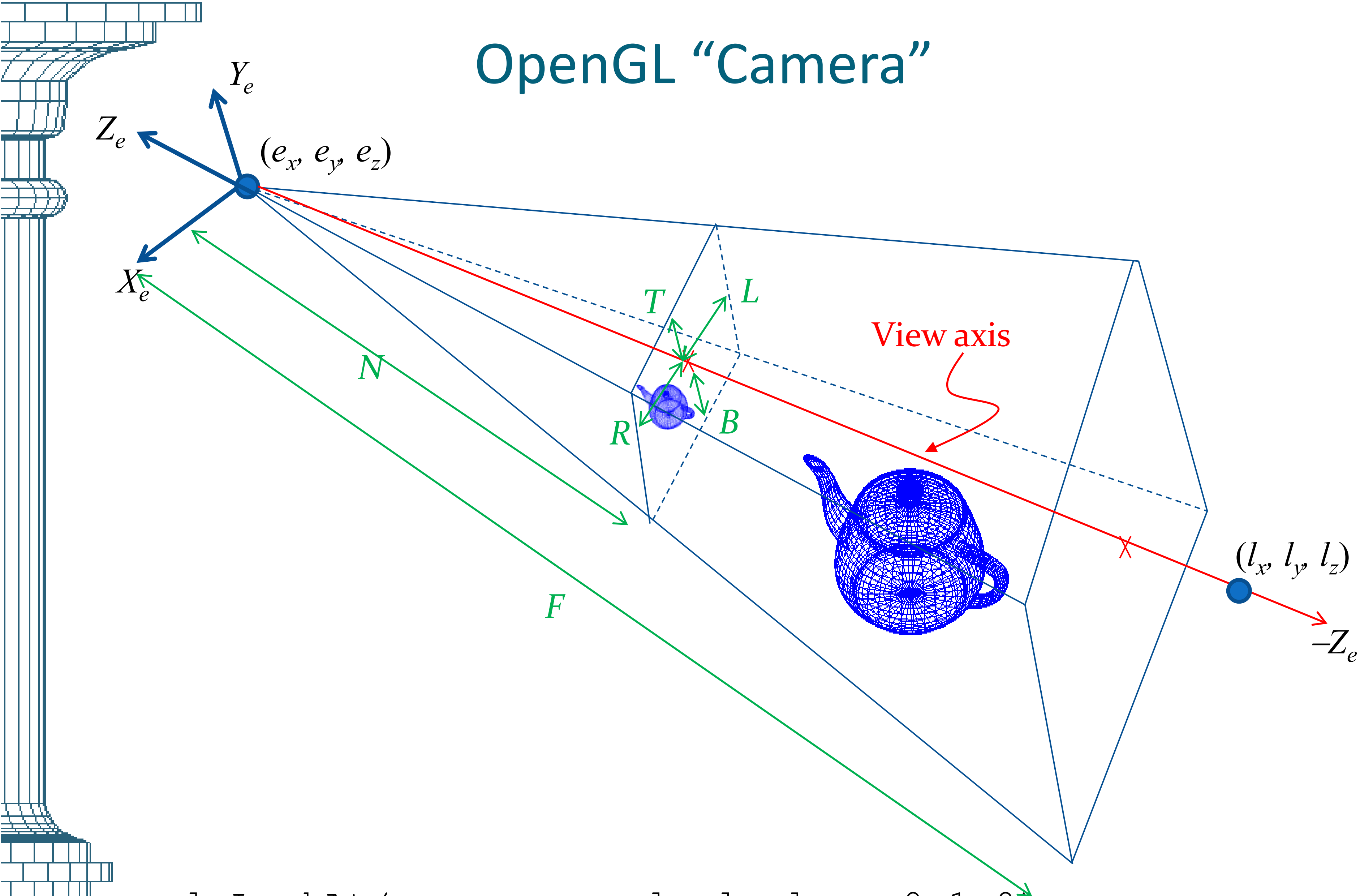
Camera View and Projection

- Camera View
  - Depends on the camera's position and orientation
  - Specified by a view transformation matrix ***V*** generated by the function `gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz);`
  - OpenGL transformations cannot be used to change the parameters defined using `gluLookAt()`.
- Camera Projection
  - Depends on the field of view and focal length.
  - Specified by a projection matrix ***P*** generated by

    `glFrustum(L, R, B, T, N, F);`

    or, `gluPerspective(fov, ar, N, F)`

OpenGL "Camera"

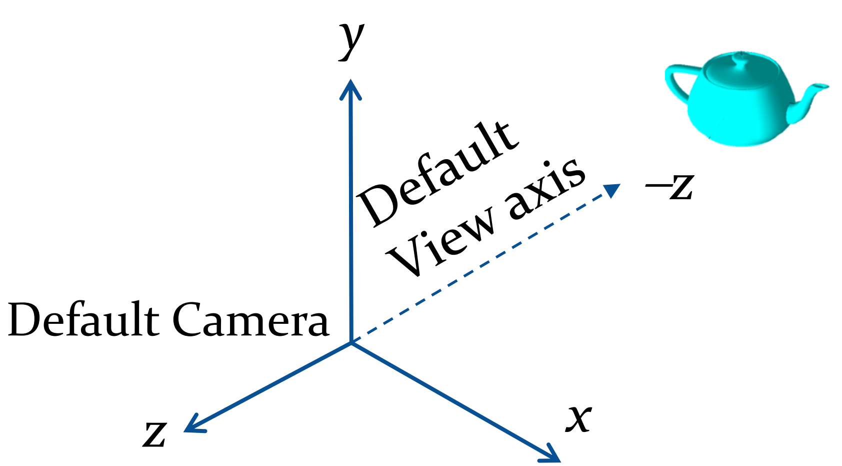



```
gluLookAt(ex,ey,ez,  lx,ly,lz,  0,1,0);
glFrustum(L, R, B, T, N, F);
```

Camera View

If gluLookAt() represents a transformation from the world coordinate space to the camera-centered coordinate system, where the camera is at the origin, and the view axis is along the negative *z* direction.

If gluLooAt() function is not used, then the view transformation is an identity transformation. This corresponds to the **default camera view**, where the camera is at the origin, looking towards the **negative *z*-axis of the world space**.



Camera Modes

A camera can be placed in a scene and transformed in different ways:

- Free-camera: The user can control the position and orientation of the camera irrespective of other object transformations in the scene.
- Camera attached to an object, eg. First Person View (see next slide)
- Fly-by camera: The camera transformed along a predefined path, usually without any user interaction.

Free Camera Example

- The camera moves parallel to the floor plane (*xz* plane)
- The current position of the camera is (e_x, e_y, e_z) (e_y = constant).
- The current view+motion direction is given by angle θ.

User controls:

Turn left: Decrement θ

Turn right: Increment θ

Move forward by step *d*:

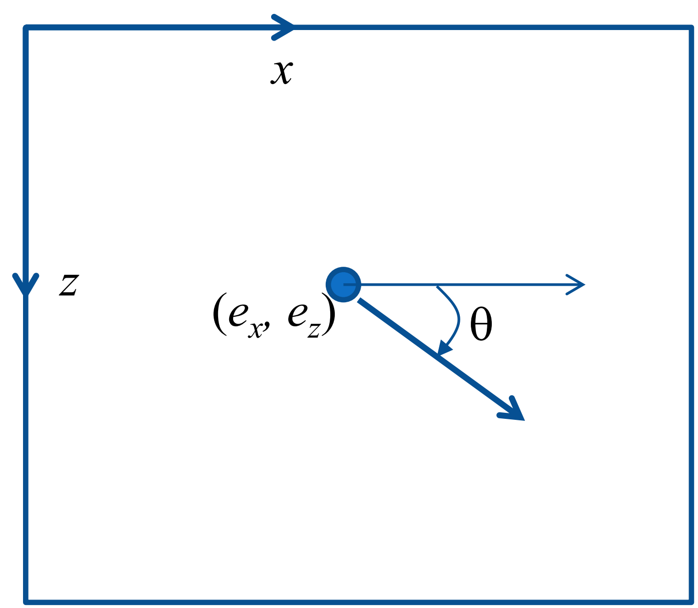
$$e_x = e_x + d\cos\theta$$
$$e_z = e_z + d\sin\theta$$

Move backward by step *d*:

$$e_x = e_x - d\cos\theta$$
$$e_z = e_z - d\sin\theta$$

Look point:

$$l_x = e_x + \cos\theta$$
$$l_y = e_y$$
$$l_z = e_z + \sin\theta$$



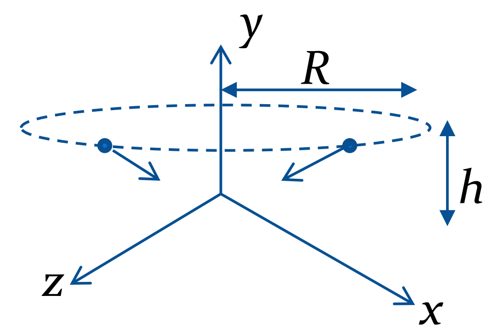
View from top

Fly-by Camera Examples

- Camera moves along a circular path parallel to the floor plane, always looking at the origin.

$$e_x = R\cos\theta, \qquad 0 \le \theta \le 2\pi$$
$$e_y = h$$
$$e_z = R\sin\theta$$

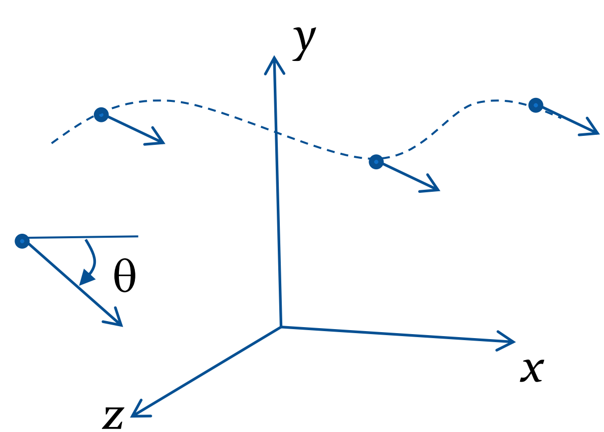
$$(l_x, l_y, l_z) = (0, 0, 0)$$



- Camera moves along a trajectory with a constant view direction

$$(e_x, e_y, e_z) = (X(t),\ Y(t),\ Z(t))$$

$$l_x = e_x + \cos\theta$$
$$l_y = e_y - \sin\theta$$
$$l_z = 0$$



First Person View

- First Person View (FPV): The view of the scene from the primary object/character being controlled. In a game, it is the view from the player’s eye level.

- Second Person View provides a view from the second most important object or character (eg. target), and is rarely used.

Third-Person View

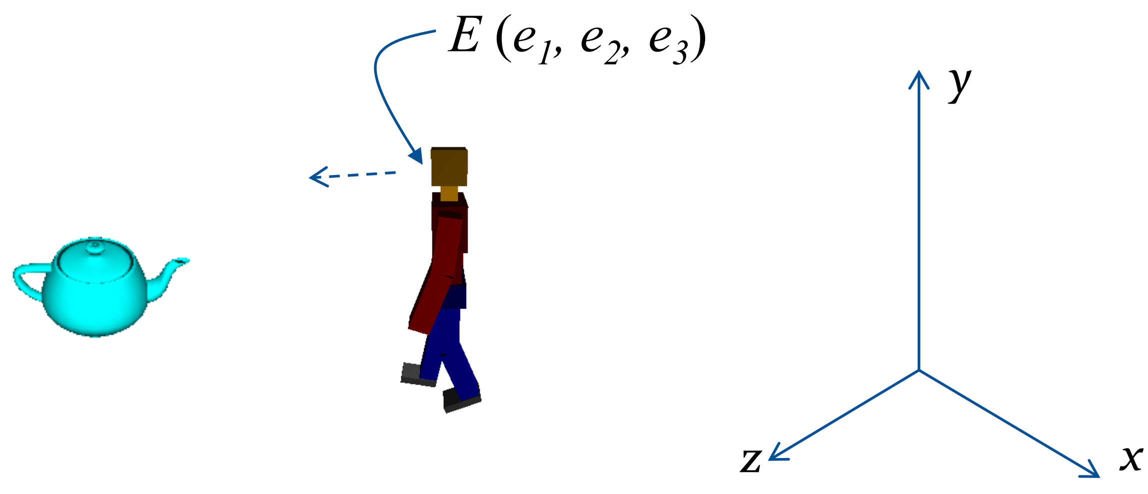
Third Person View: A view of the scene from a different perspective (eg. a general observer). This camera mode could either be a "free-camera" or dependent on other transformations (eg. locked view).

Camera Modes

Creating First Person Views (Method 1)

Keep track of the object's position and orientation in world coordinate space, and update the camera position and the look vector.

- You will need to compute the object's pose every frame, and reposition the camera on the transformed object.
- <u>Note</u>: You cannot get the transformed vertex coordinates from OpenGL, you will have to compute them separately.



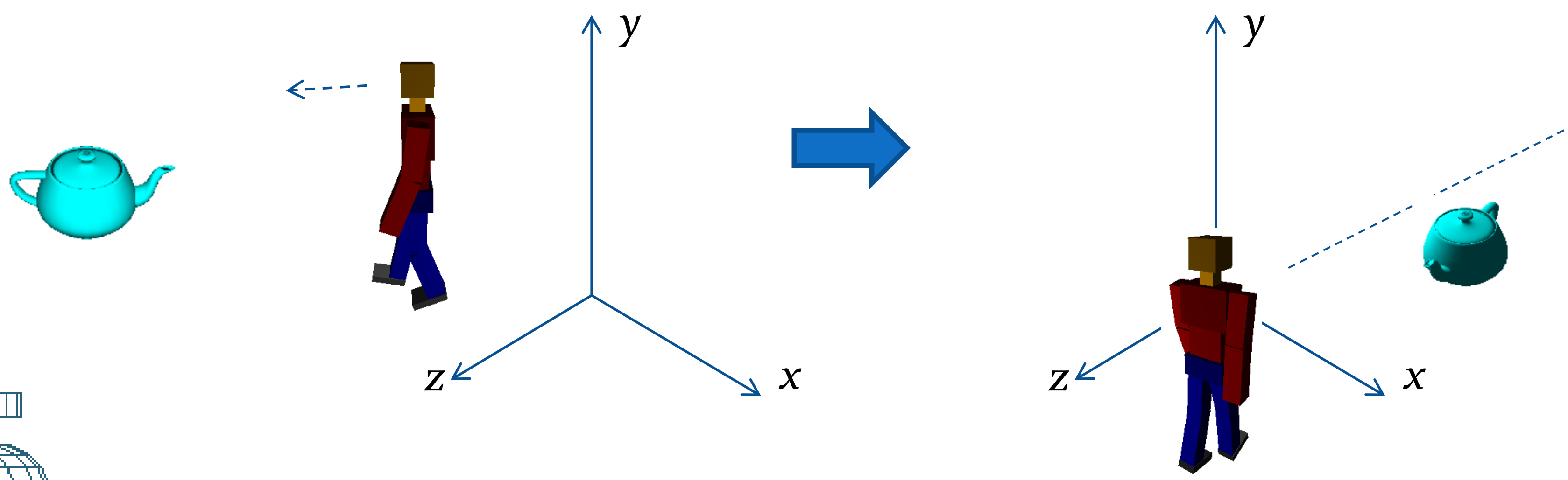
Creating First Person Views (Method 1)

```
void display()
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  ...  // compute camera parameters here
  ...  // by using character transformation
     gluLookAt (ex,ey,ez, lx,ly,lz, 0,1,0);
     ...   //common transforms
     glPushMatrix();
        //character transform
        drawCharacter();  //user defined
     glPopMatrix();

     glPushMatrix();
        //Teapot transform
        glutSolidTeapot(1);
     glPopMatrix();
```

Creating First Person Views (Method 2)

- This method does not use gluLookAt(...) which requires the transformed coordinates of a point on the character.
- Instead, the enitre scene is inverse-transformed so that the character goes back to the origin, looking towards the –z axis.



Creating First Person Views (Method 2)

```
void display()
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  ...  // Inverse of character transformation

     ...   //common scene transforms
     glPushMatrix();
        //character transform
        drawCharacter();  //user defined
     glPopMatrix();

     glPushMatrix();
        //Teapot transform
        glutSolidTeapot(1);
     glPopMatrix();
```

Character Transformation Example

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
  glRotatef(180, 0, 1, 0);    //Look towards -z
  glRotatef(-theta, 0, 1, 0);
  glTranslatef(-tx, -ty, -tz);

     ...   //common scene transforms
     glPushMatrix();
        glTranslatef(tx, ty, tz);
        glRotatef(theta, 0, 1, 0);
        drawCharacter();  //user defined
     glPopMatrix();

     ...  //other objects in the scene
```

Inverse of

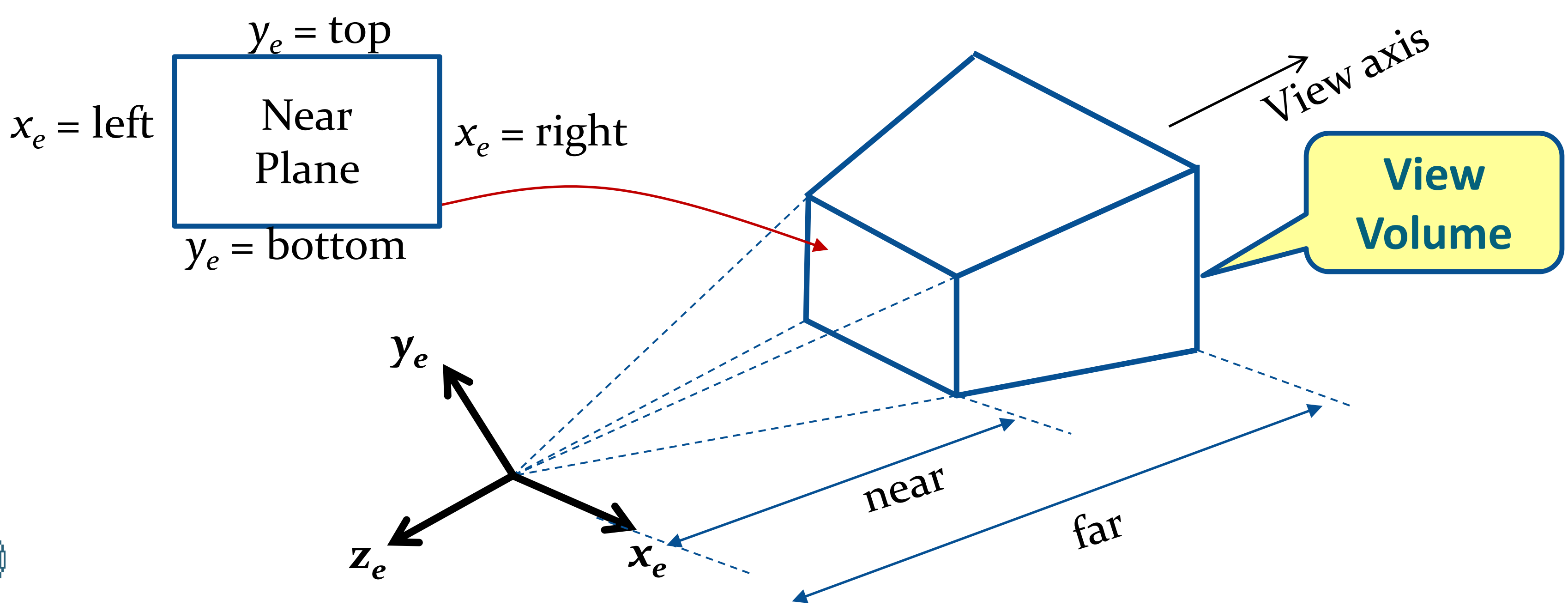
View Volumes

- The view transformation only transforms the world coordinates of points into the camera's coordinate frame.
- We need to specify "how much" the camera actually sees. That is, we require a view volume that contains the part of the scene that is visible to the camera. In other words, the view volume acts as a **clipping volume**.
- We further require a projection model to simulate the way in which the 3D scene is viewed.
- The view volume is attached to the camera and is always defined in the camera-coordinate space. Therefore, all points inside the view volume are represented using **eye coordinates** (slide 3).

Perspective View Volume

- The perspective view volume is defined by a frustum that has its vertex at the eye position. The near-plane acts as the plane of projection.
- OpenGL function:

```
    glFrustum(left, right, bottom, top, near, far);
Eg:  glFrustum(-10, 10, -8, 8, 10, 100);
```

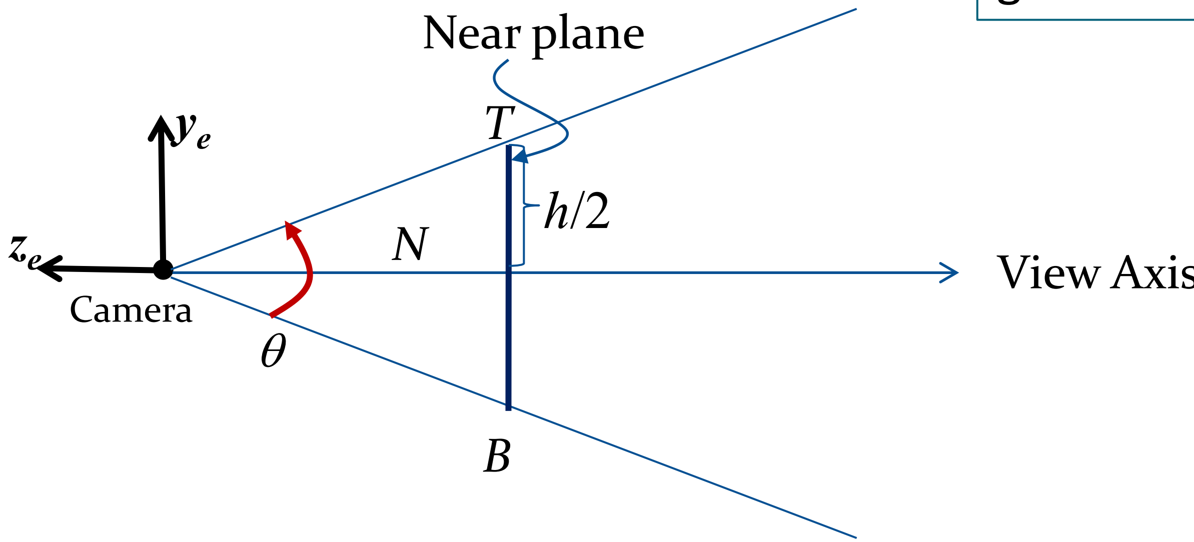


Perspective View

- The field of view of the view frustum is a useful parameter that can be conveniently adjusted to cover a region in front of the camera.

glFrustum(L, R, B, T, N, F)

$$h = T - B$$



- Field of view along the *y*-axis of the eye-coordinate space fov = θ.

$$\tan\left(\frac{\theta}{2}\right) = \frac{h}{2N}$$

gluPerspective

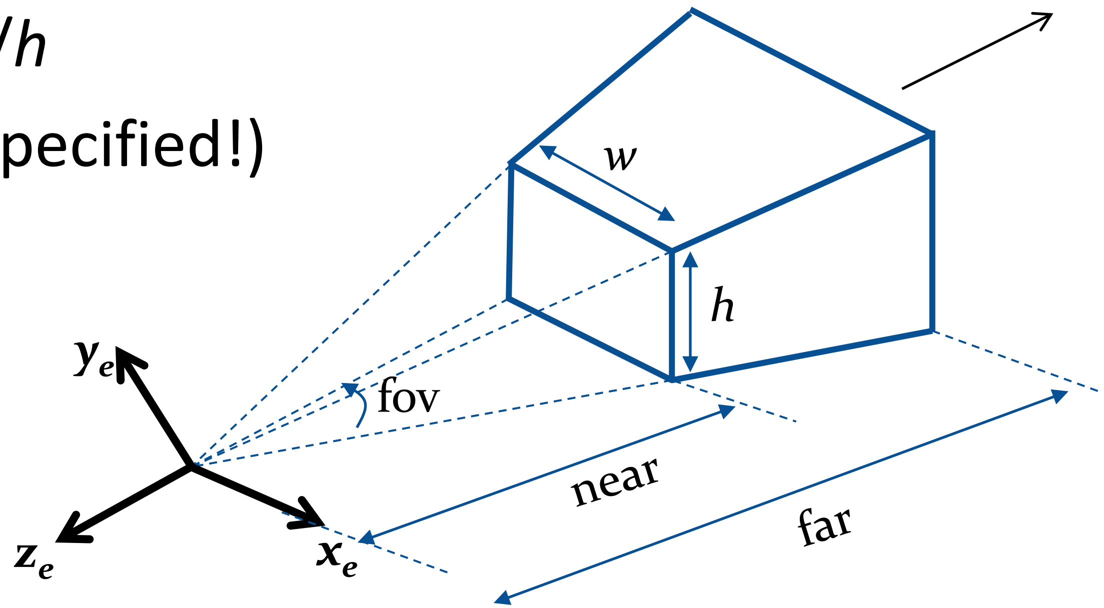
- The GLU library provides another function for perspective transformation in the form

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

- In this case, the view axis passes through the centre of the near plane.
- Aspect Ratio $a = w/h$

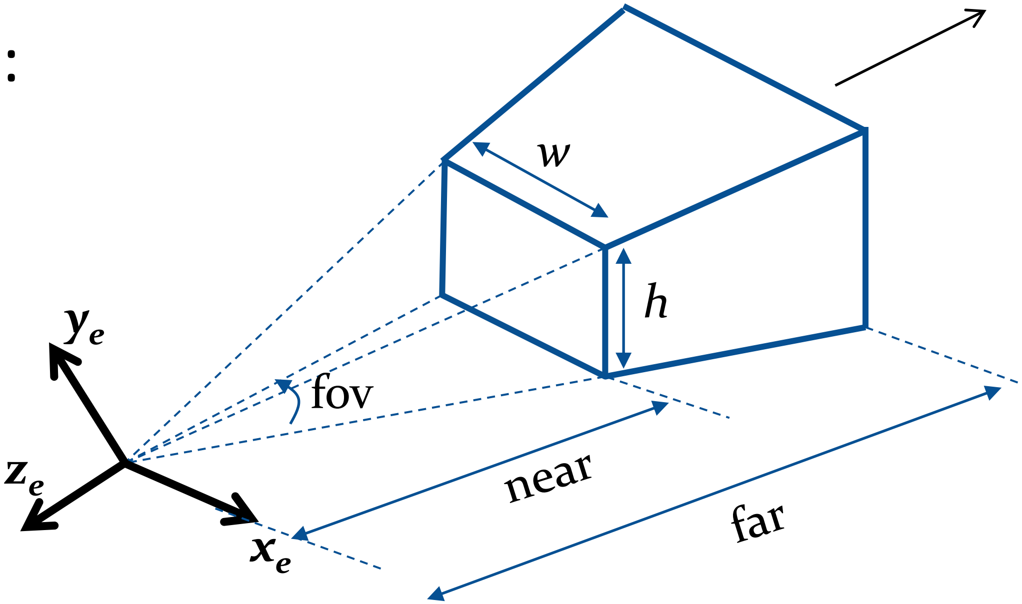
(Note: *w*, *h* are not specified!)

- fov = θ



gluPerspective vs. glFrustum

(θ, a, N, F) $\rightarrow$ (L,R,B,T,N,F):



$$h = 2N\tan\left(\frac{\theta}{2}\right)$$ (Slide 20)

$w = a.\ h$

$L = -w/2 \qquad R = w/2$

$B = -h/2 \qquad T = h/2,$

Note: $L + R = 0, \quad B + T = 0$

- (L,R,B,T,N,F) $\rightarrow$ (θ, a, N, F) :

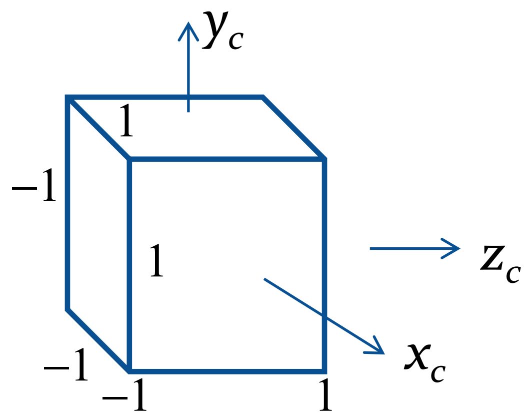
$w = R - L\ , \qquad h = T - B$

$a = w/h$

$$\theta = 2\tan^{-1}\left(\frac{h}{2N}\right)$$

The Canonical View Volume

- All view volumes are mapped to a canonical view volume which is an axis-aligned cube with sides at a distance of 1 unit from the centre.
- The coordinates of a point inside the canonical view volume are called clip coordinates.
- The canonical view volume facilitates clipping of the primitives with its sides.
- A point is visible only if it has clip coordinates between -1 and +1.

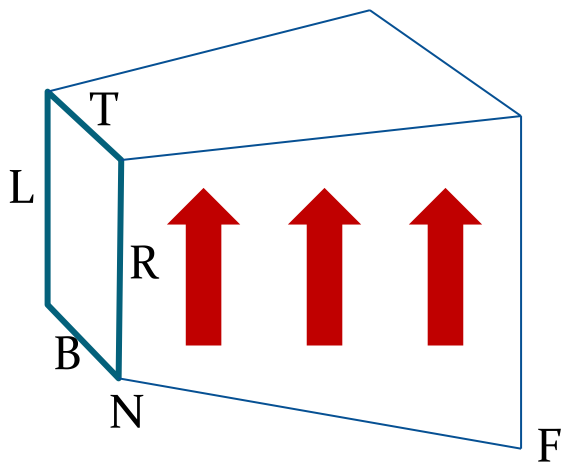


Clip Coordinate Axes

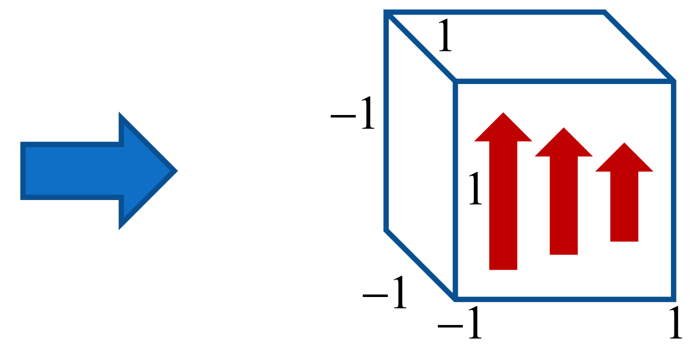
Left handed system

glFrustum(L,R,B,T,N,F)

- The function `glFrustum(...)` transforms points inside the perspective view volume into points inside the canonical view volume, where the coordinates have the range [-1, 1].



Eye coordinates

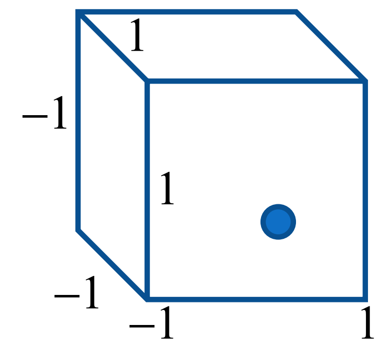


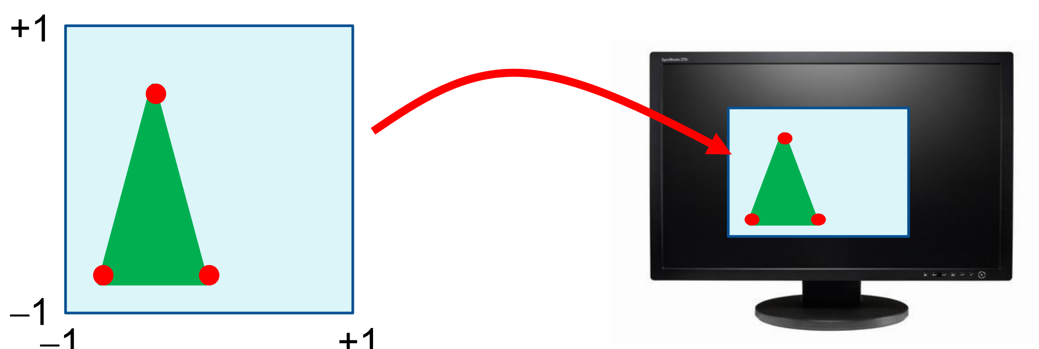
Clip coordinates

Clip Coordinates

Suppose a point has clip coordinates (x_c, y_c, z_c).

- The z_c value is called the point's **pseudo-depth**. It has a value between -1 and +1.
- The pseudo-depth is converted into a depth buffer value in the range [0, 1] using the equation $z_{\text{depth}} = (z_c + 1)/2$
- If the point passes the **depth test**, then its clip coordinates (x_c, y_c) are mapped to the display viewport.





Clip Coordinates

An Overview of Transformations