Projection Transformations in OpenGL

- **Selecting the projection matrix**

  - Use *glMatrixMode(GL_PROJECTION)* to select the *Projection* matrix

  - In practice, we define a projection transformation before drawing any object

- **Specifying the projection matrix**

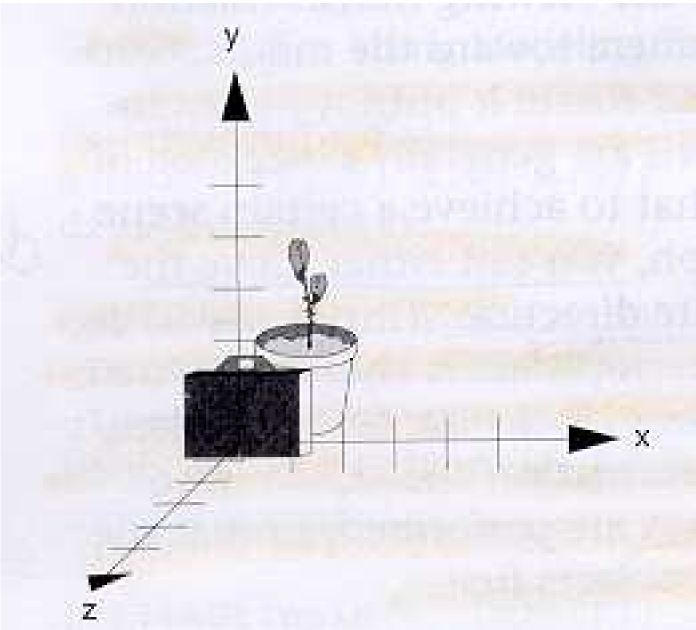
  - Specify the camera's coordinate system (VCS)

  - Specify the view volume

- **Default position of the camera**

  - The camera is positioned at the origin and is looking in the negative *z* direction

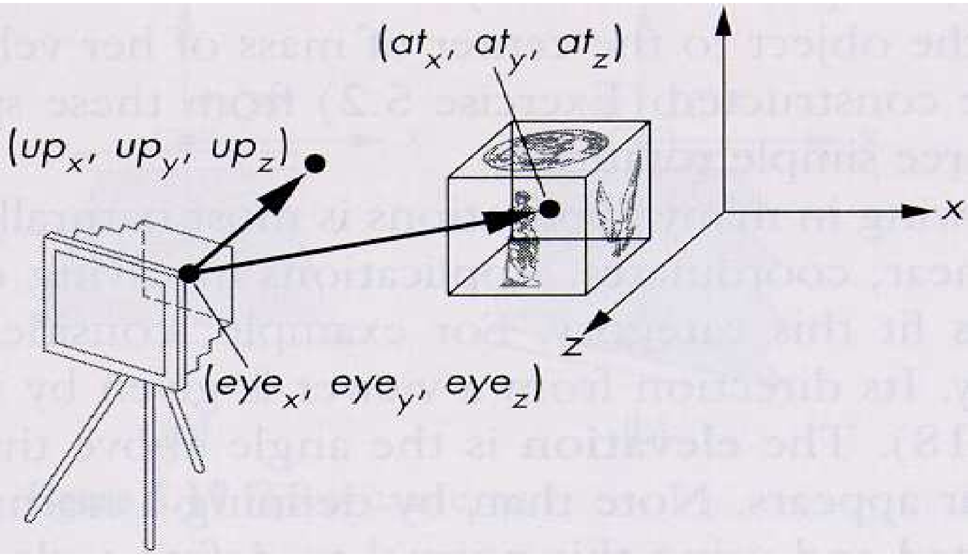
  - The COP is at the origin



- **Changing the position of the camera**

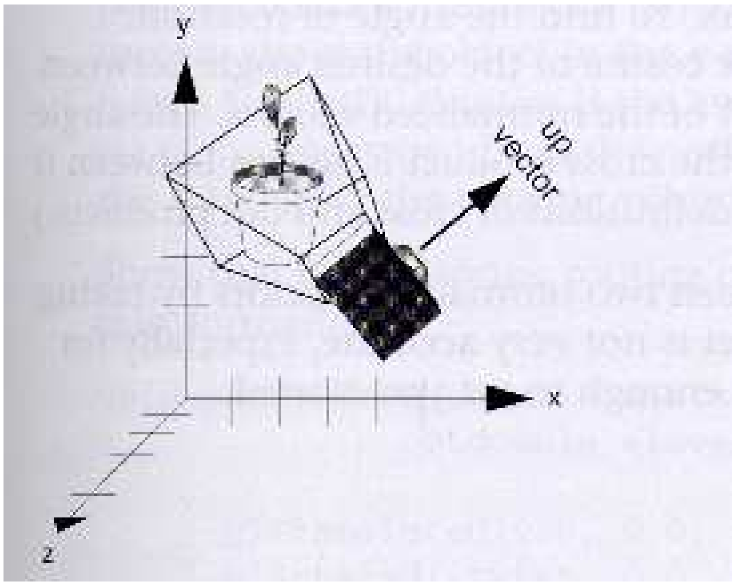
- Use the function

$$gluLookAt(eye_x, eye_y, eye_y, at_x, at_y, at_z,\ view-up_x, view-up_y, view-up_z)$$



- Default position: *gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, -100.0, 0.0, 1.0, 0.0)*

- New position: *gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0)*



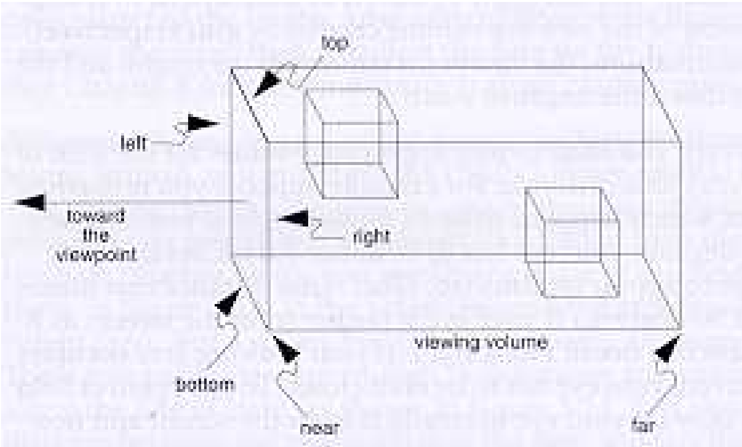
- *gluLookAt()* creates a matrix (*view orientation matrix*) and postmultiplies it with the current transformation matrix (CTM)

-- --

- **Specifying the view volume: orthographic projection**

  - Use the function *glOrtho(left, right, bottom, top, near, far)*

  - Both *near* and *far* can be positive or negative



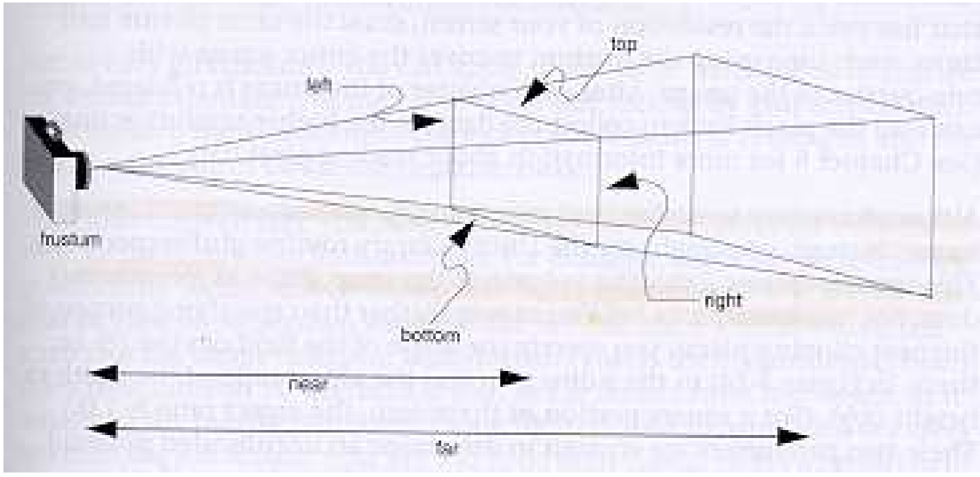
  - *glOrtho()* creates a projection matrix and postmultiplies it with the current transformation matrix (CTM)

- **Specifying the view volume: perspective projection**

  - There are two ways: using *glFrustum()* or *gluPerspective()*

  - <u>First way:</u> *glFrustum(left, right, bottom, top, near, far)*

  - *near* and *far* should always be positive



- The visible z values lie in the range $-far \le z \le -near$

- The frustum does not have to be symmetric

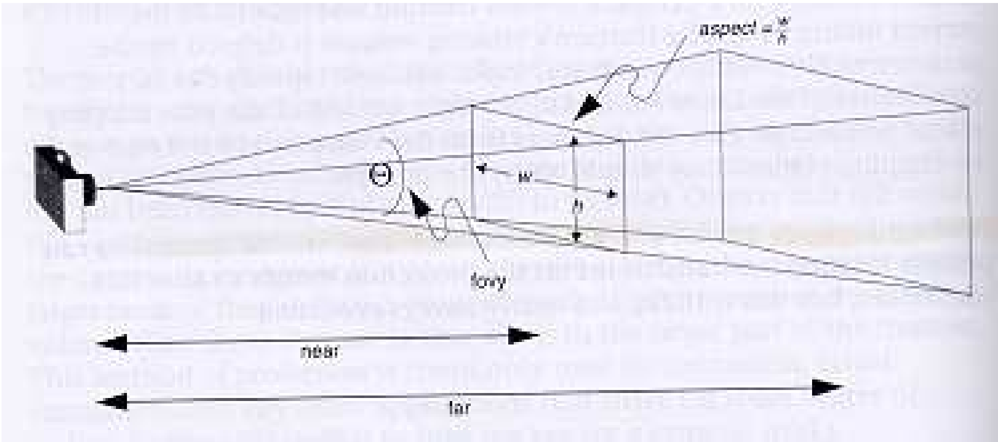
- *glFrustum()* creates a projection matrix and postmultiplies it with the current transformation matrix (CTM)

- <u>Second way:</u> *gluPerspective(θ, aspect, near, far)*

- $0 < \theta < 180$, $h = 2\ near\ tan(\theta/2)$, $aspect = width/height$, $w = aspect\ h$

- *near* and *far* are the same like in *glFrustum()*

- The frustum is symmetric in both the x- and y-axes



- *gluPerspective()* creates a projection matrix and postmultiplies it with the current transformation matrix (CTM)

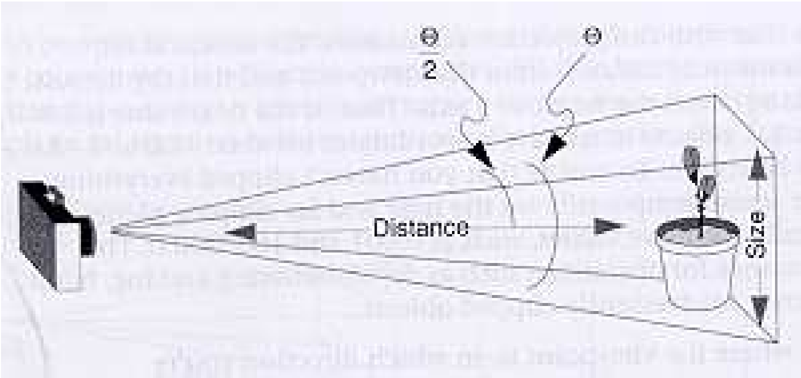
- **Choosing the angle θ**

  - Making θ smaller without moving the model simulates "zooming-in"

  - Making θ larger without moving the model simulates "zooming-out"

  - Suppose we know the size of the object and its distance from the viewpoint

$$tan(\theta/2) = \frac{s/2}{d} \text{ or } \theta = 2\tan^{-1}(\frac{s}{2d})$$



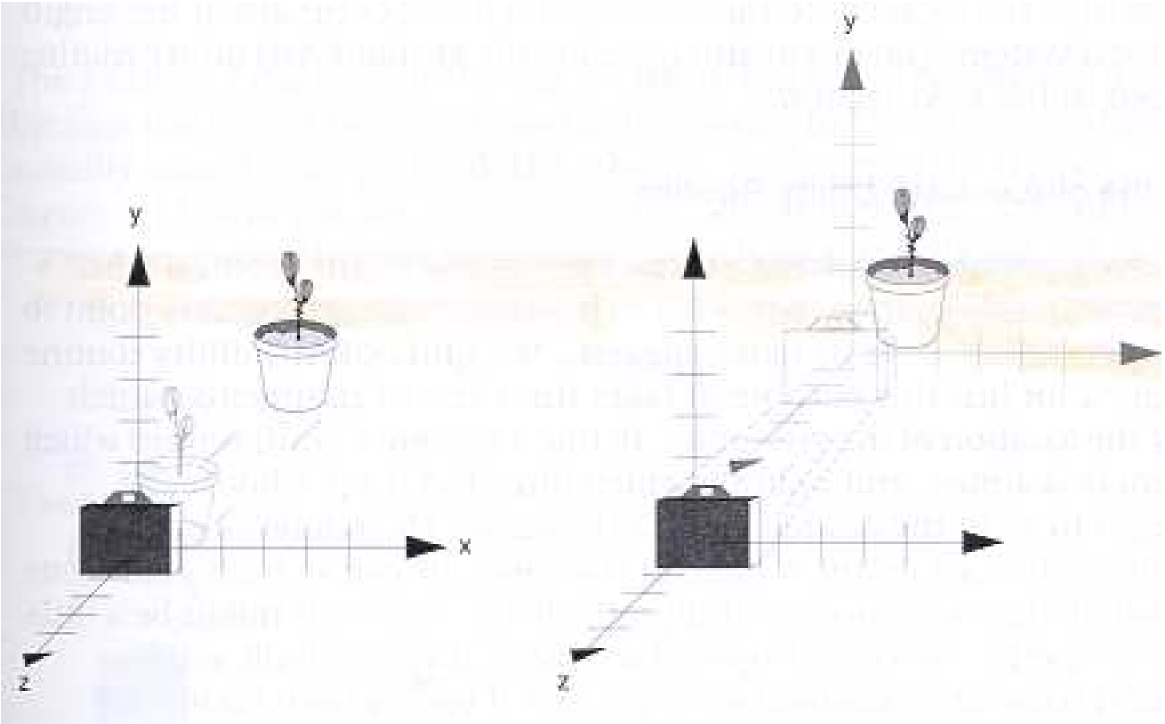
- **Hint: specify the view volume in the reshape function**

```
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor(1.0, 1.0, 1.0);
 glLoadIdentity();
 gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
 glScalef(1.0, 2.0, 1.0);
 glutWireCube(1.0);
}

void reshape(int w, int h)
{
 glViewport(0, 0, (GLsizei)w, (GLsizei)h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 20.0);
 glMatrixMode(GL_MODELVIEW);
}
```

- **Duality in the camera and object transformations**

  - Moving the camera in one direction is equivalent to moving the object(s) in the opposite direction



  - The code below is equivalent to the code above ...

```
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glColor(1.0, 1.0, 1.0);
 glLoadIdentity();
 glTranslate(0.0, 0.0, -5.0);
 glScalef(1.0, 2.0, 1.0);
 glutWireCube(1.0);
}
```

- **Oblique projections in OpenGL**

  - OpenGL does to support oblique projections explicitly

  - To implement oblique projections, we must first apply a shear transformation on the object(s) and then apply orthographic projection