Viewing and Projections

Lecture 12

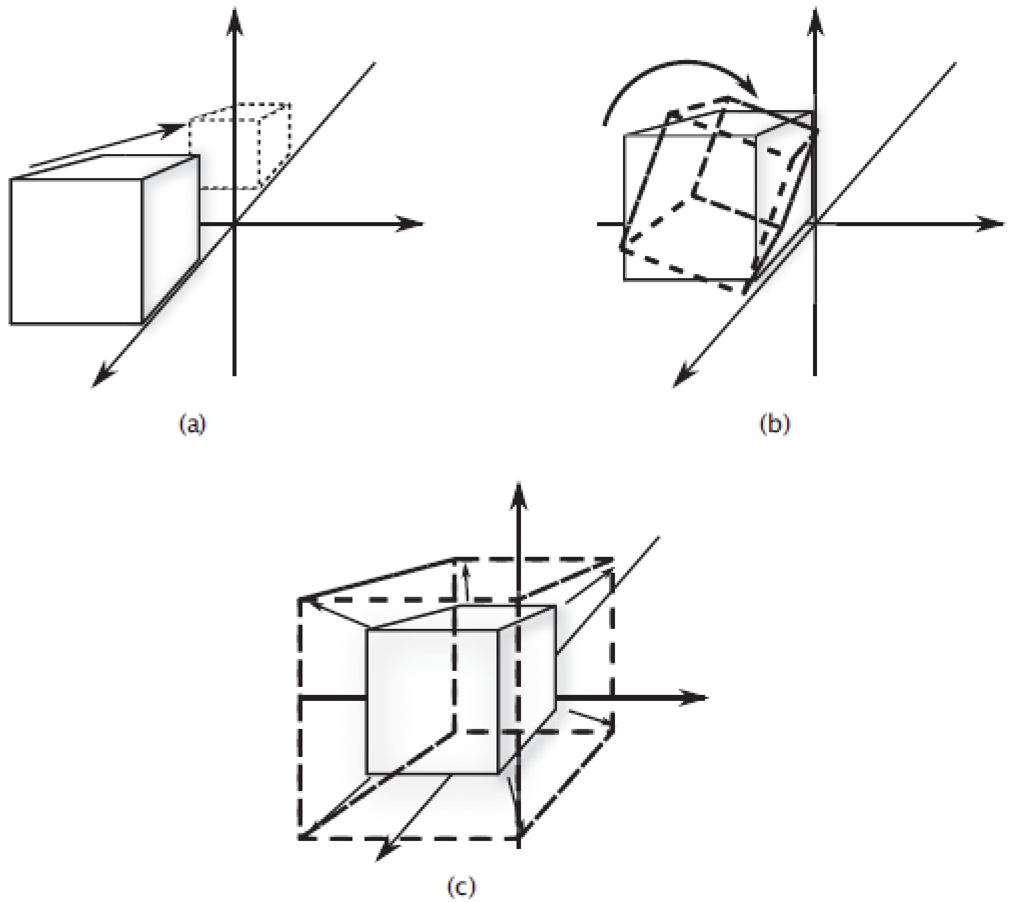
Outline

- OpenGL transformations
- Viewing and Projection
- Placing the camera
- Orthographic Frustum
- Perspective Frustum
- Sun/Earth/Moon Example

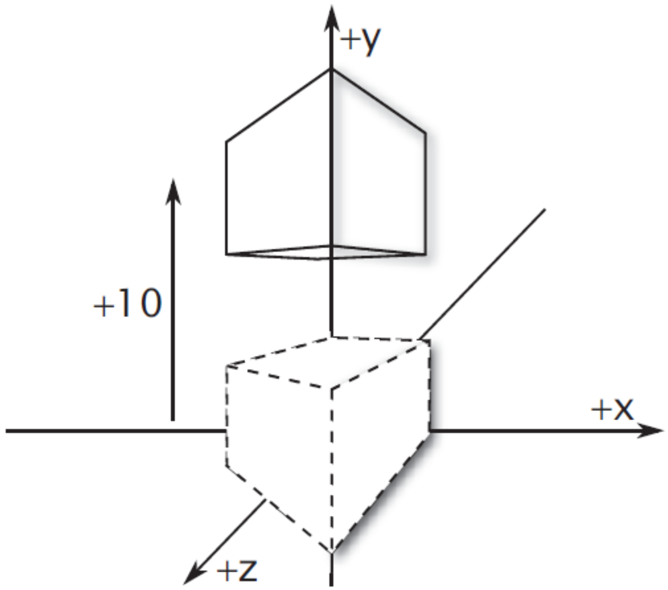
OpenGL transformations

Transformation	Use
Viewing	Specifies the location of the viewer or camera
Modeling	Moves objects around the scene
Modelview	Describes the duality of viewing and modeling transformations
Projection	Clips and sizes the viewing volume
Viewport	Scales the final output to the window

Modeling transformations



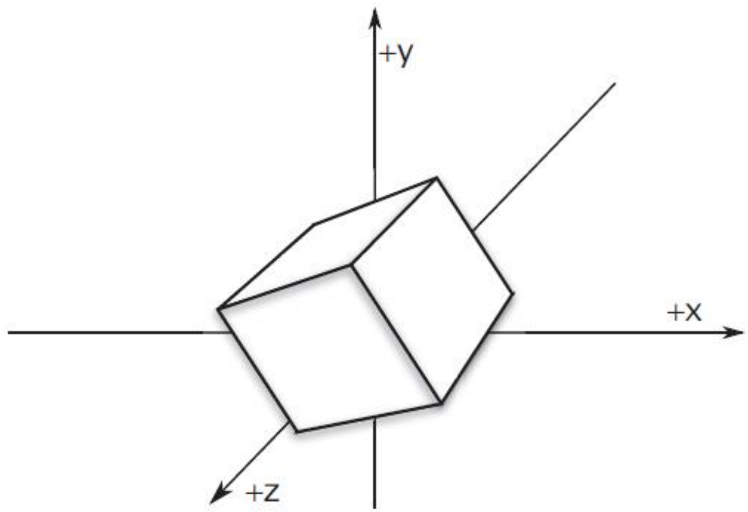
$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & t_x \\ 0.0 & 1.0 & 0.0 & t_y \\ 0.0 & 0.0 & 1.0 & t_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$



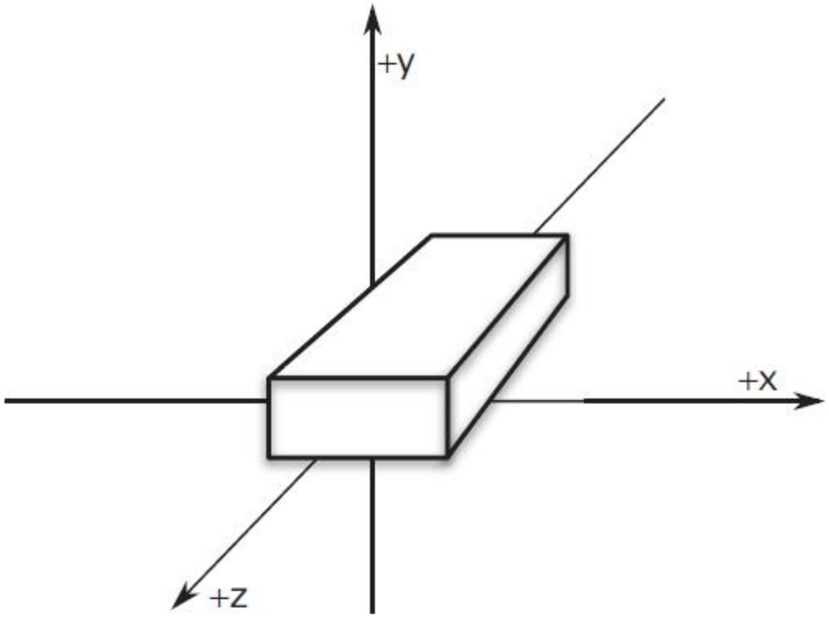
$$R_x(\theta) = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & \cos\theta & \sin\theta & 0.0 \\ 0.0 & -\sin\theta & \cos\theta & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0.0 & -\sin\theta & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ \sin\theta & 0.0 & \cos\theta & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0.0 & 0.0 \\ \sin\theta & \cos\theta & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$R_z(\psi)\,R_y(\theta)\,R_x(\phi) = \begin{bmatrix} c_\theta c_\psi & c_\phi s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi - c_\phi s_\theta c_\psi & 0.0 \\ -c_\theta s_\psi & c_\phi c_\psi - s_\phi s_\theta s_\psi & s_\phi c_\psi + c_\phi s_\theta s_\psi & 0.0 \\ s_\theta & -s_\phi c_\theta & c_\phi c_\theta & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$



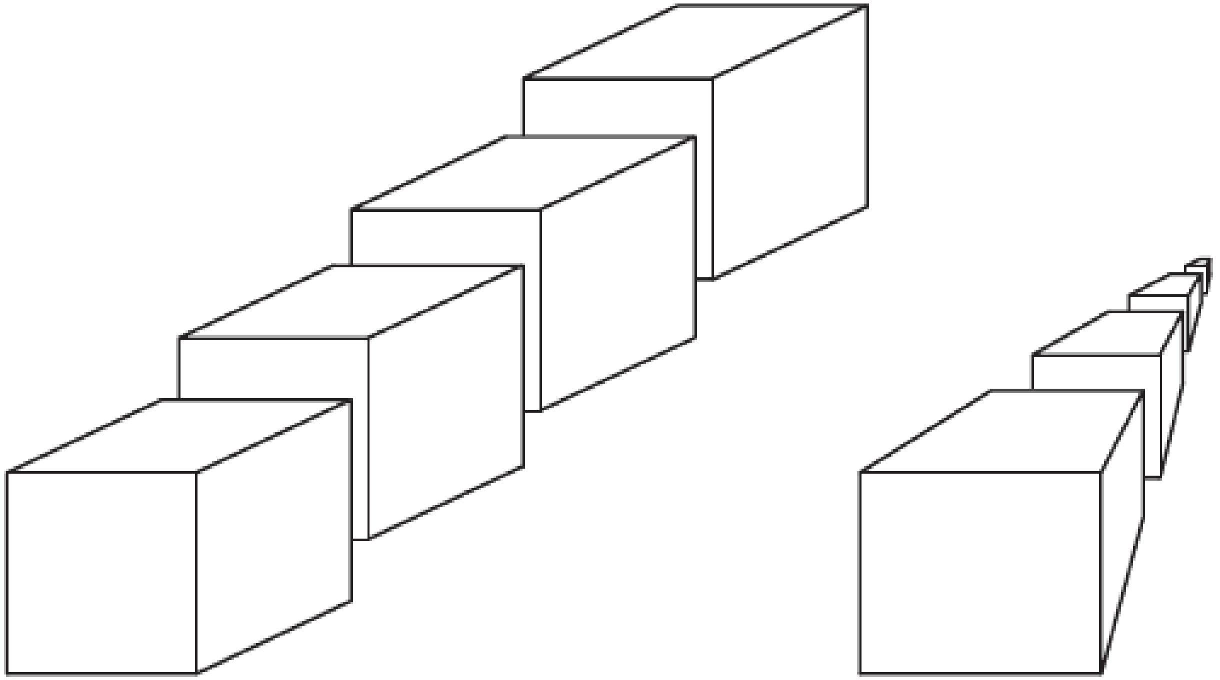
$$\begin{bmatrix} s_x & 0.0 & 0.0 & 0.0 \\ 0.0 & s_y & 0.0 & 0.0 \\ 0.0 & 0.0 & s_z & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$



Projection transformations

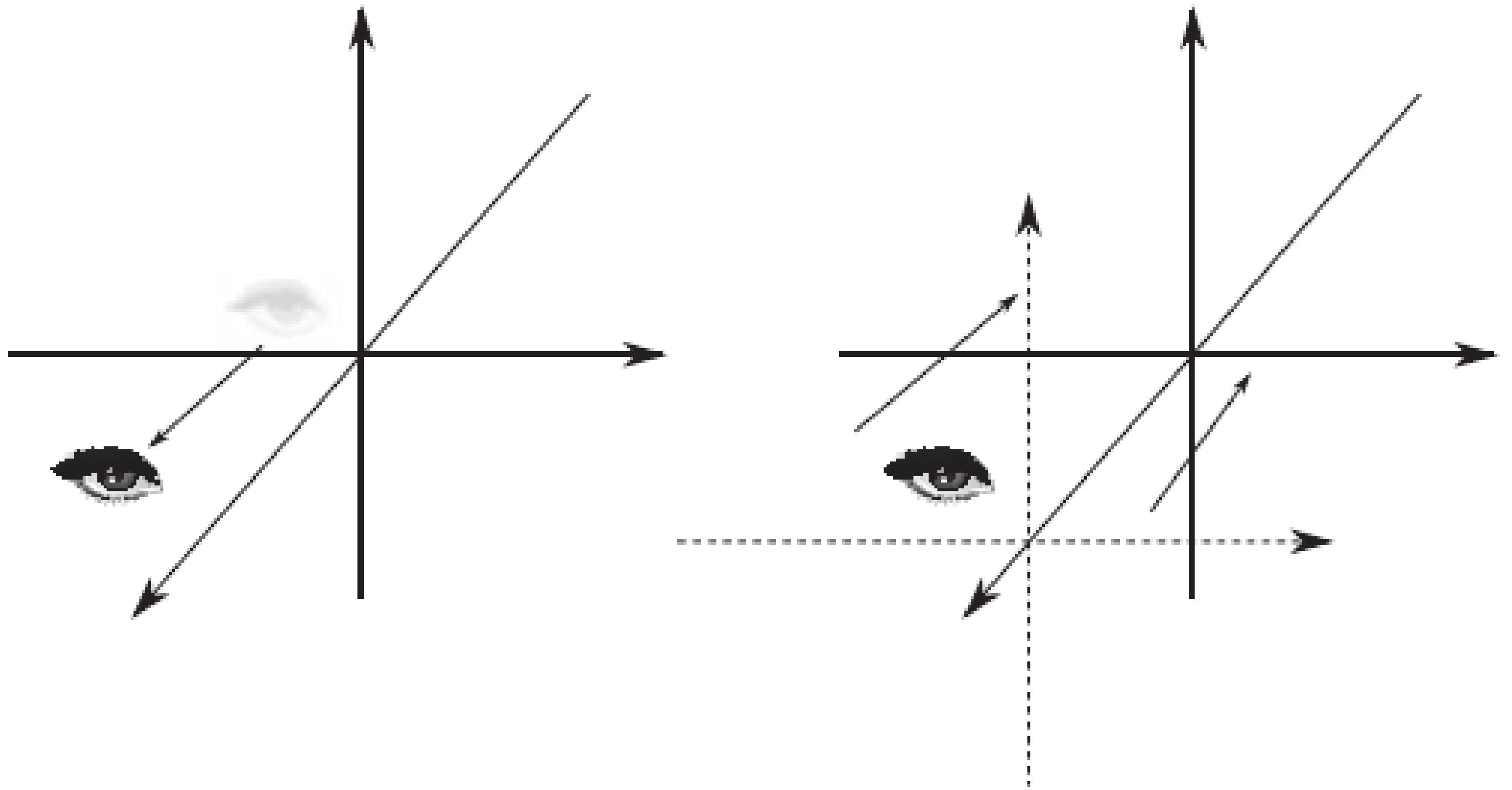
- Projection transformation is applied to the final modelview orientation.
- This projection actually **defines the viewing volume** and **establishes clipping planes**
- More specifically, the projection transformation specifies how a finished scene is translated to the final image on the screen.
- There're two types of projections: *orthographic* and *perspective*

Orthographic vs perspective projection



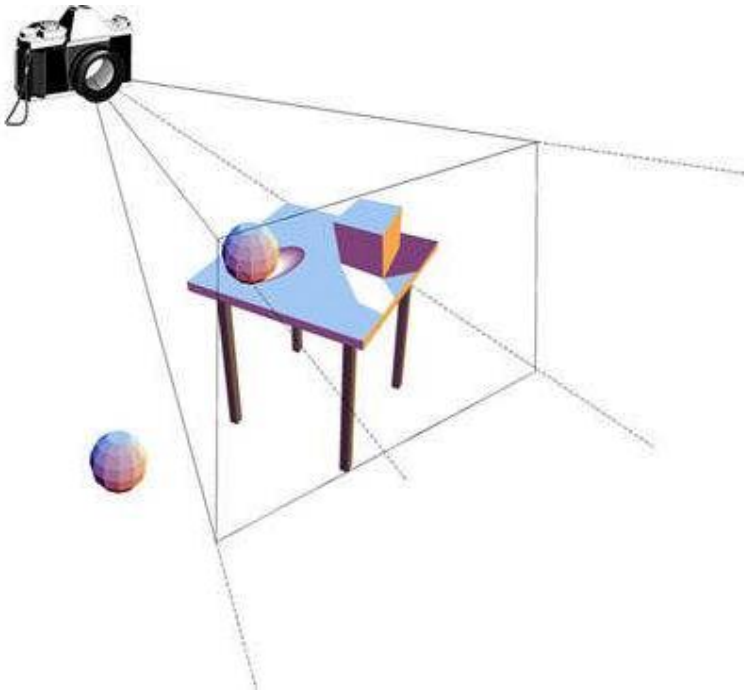
Perspective

View coordinates: camera or eye coordinates



Viewing and Projection

- In OpenGL we distinguish between:
  - **Viewing**: placing the camera
  - **Projection**: describing the viewing frustum of the camera (and thereby the projection transformation)
  - **Perspective divide**: computing homogeneous points



Perspective matrix

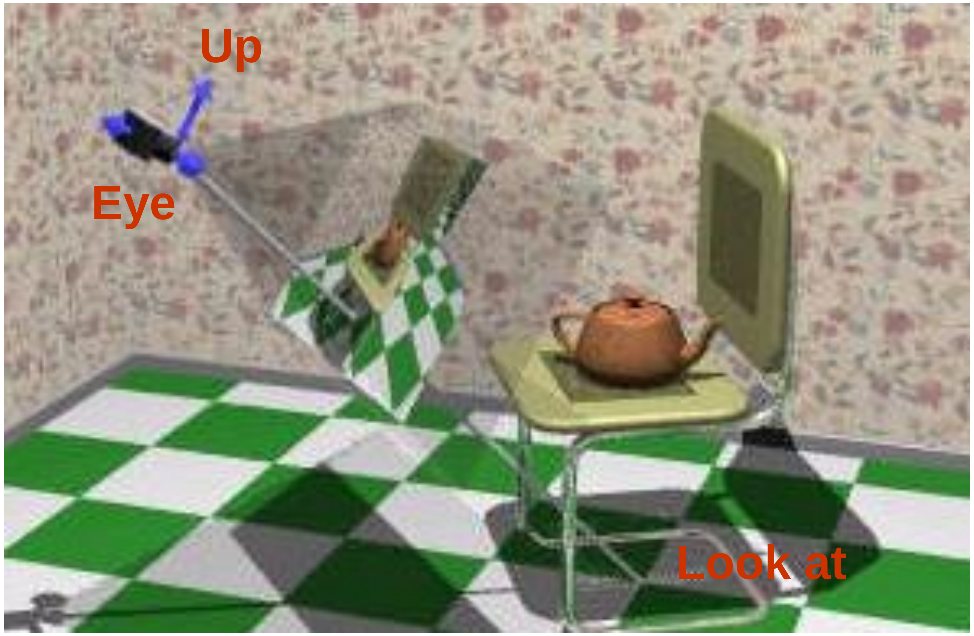
$$\begin{bmatrix} \frac{2\cdot near}{right-left} & 0.0 & \frac{right+left}{right-left} & 0.0 \\ 0.0 & \frac{2\cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0.0 \\ 0.0 & 0.0 & \frac{near+far}{near-far} & \frac{2\cdot near\cdot far}{near-far} \\ 0.0 & 0.0 & -1.0 & 0.0 \end{bmatrix}$$

Orthographic matrix

$$\begin{bmatrix} \frac{2}{right-left} & 0.0 & 0.0 & \frac{left+right}{left-right} \\ 0.0 & \frac{2}{top-bottom} & 0.0 & \frac{bottom+top}{bottom-top} \\ 0.0 & 0.0 & \frac{2}{near-far} & \frac{near+far}{far-near} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Placing the camera

- Identify the **eye point** where the camera is located
- Identify the **look-at point** that we wish to appear in the center of our view
- Identify an **up-vector** that we wish to be oriented upwards in our final image



gluLookAt()

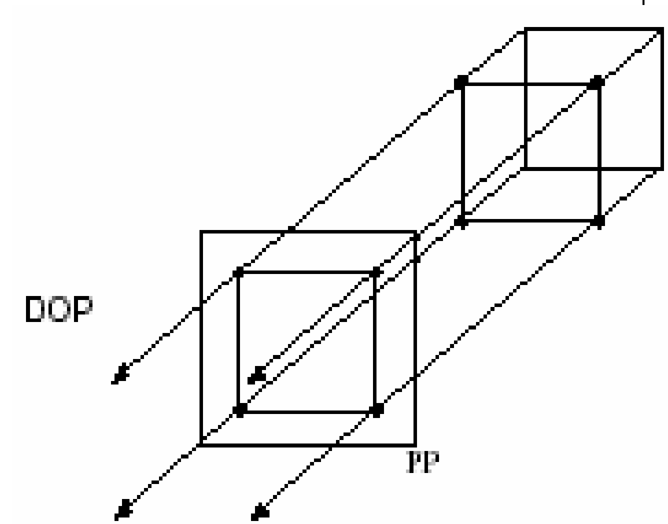
- OpenGL provides a very helpful utility function that implements the look-at viewing specification:

```
gluLookAt ( eyex, eyey, eyez,  // eye point
            atx,  aty,  atz,   // lookat point
            upx,  upy,  upz ); // up vector
```

- These parameters are expressed in world coordinates

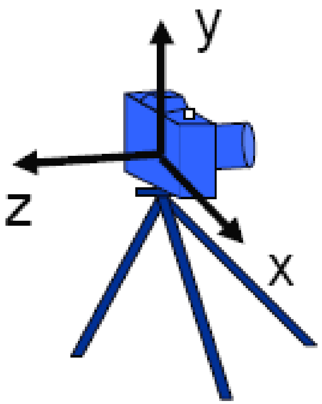
Parallel Projections

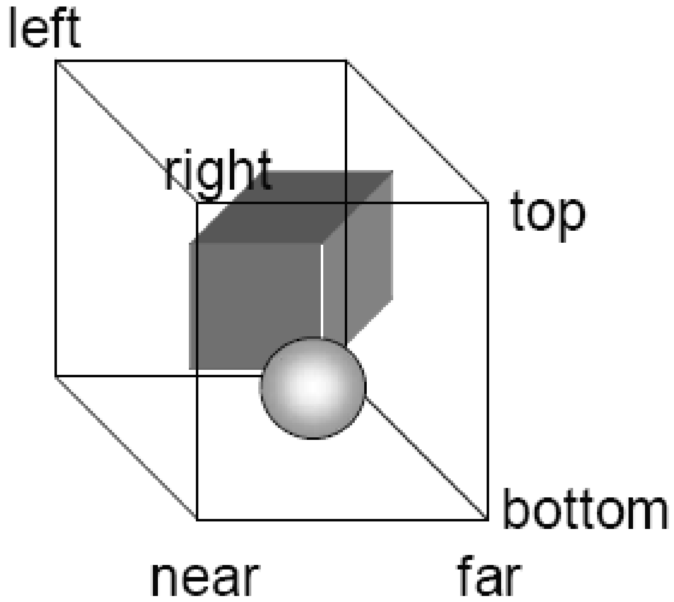
- The simplest form of parallel projection is simply along lines parallel to the z-axis onto the xy-plane
- This form of projection is called **orthographic**



Orthographic Frustum

- The user specifies the orthographic viewing frustum by specifying min and max x/y coordinates
- It is necessary to indicated a range of distances along the z-axis by specifying near and far planes.





Orthographic Projection in OpenGL

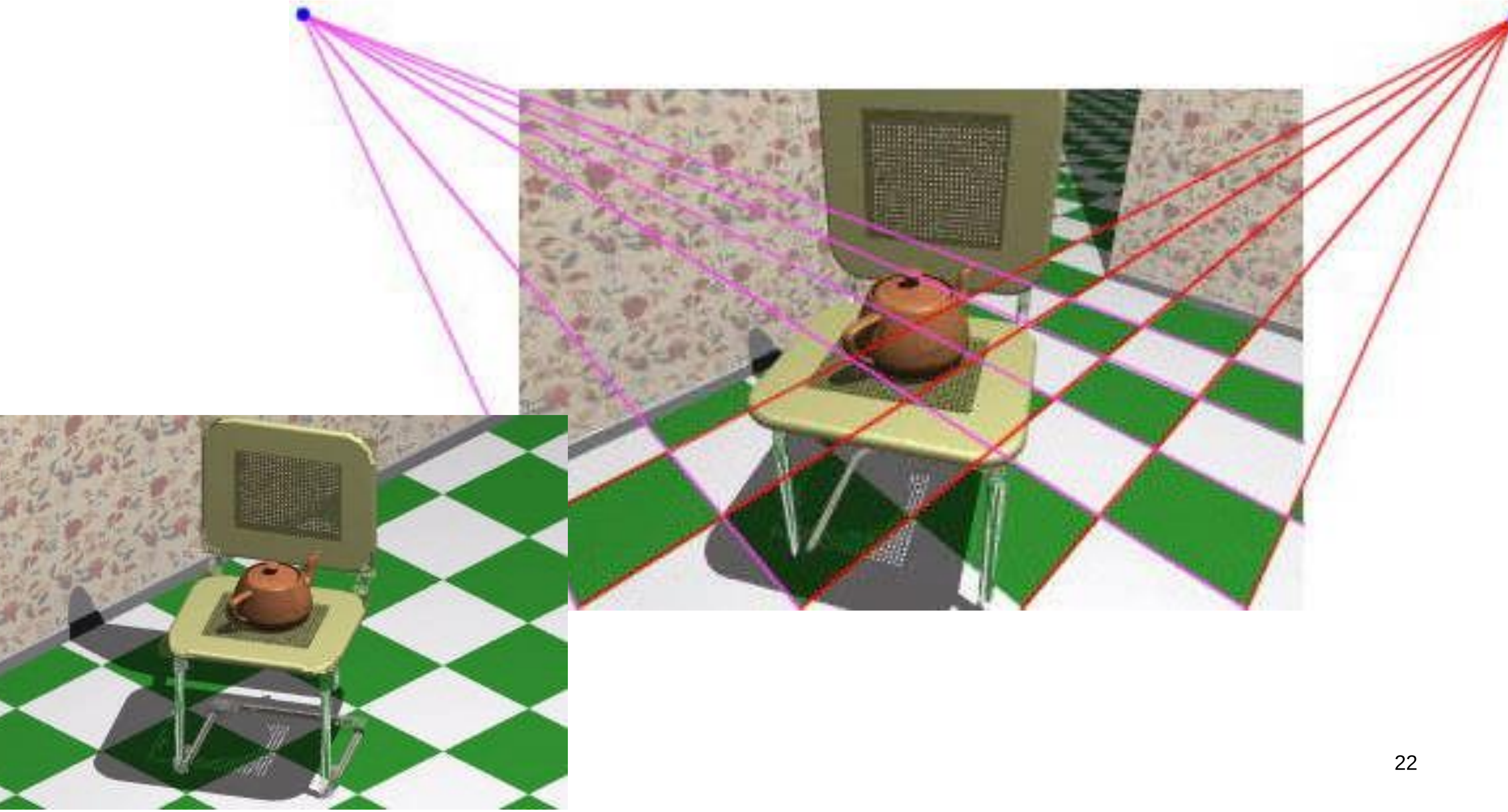
- **glOrtho(left, right, bottom, top, near, far);**

- And the 2D version (another GL utility function):
  - **gluOrtho2D(left, right, bottom, top);**

Properties of Parallel Projections

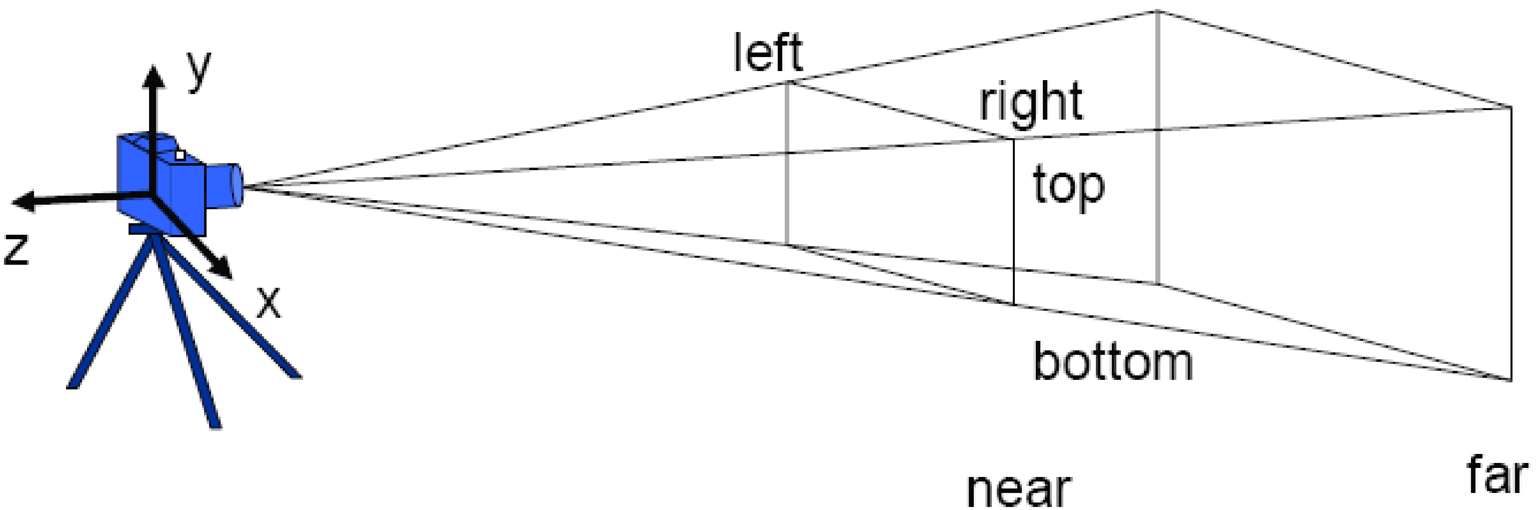
- Not realistic looking
- Good for exact measurements
  - A kind of affine transformation
  - Parallel lines remain parallel
  - Ratios are preserved
  - Angles (in general) not preserved
- Most often used in CAD, architectural drawings, etc., where taking exact measurement is important

Perspective Projections



Perspective Viewing Frustum

- Just as in the orthographic case, we specify a perspective viewing frustum
- Values for left, right, top, and bottom are specified at the near depth.



Perspective Viewing Frustum

- OpenGL provides a function to set up this perspective transformation:

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

- There is also a simpler OpenGL utility function:

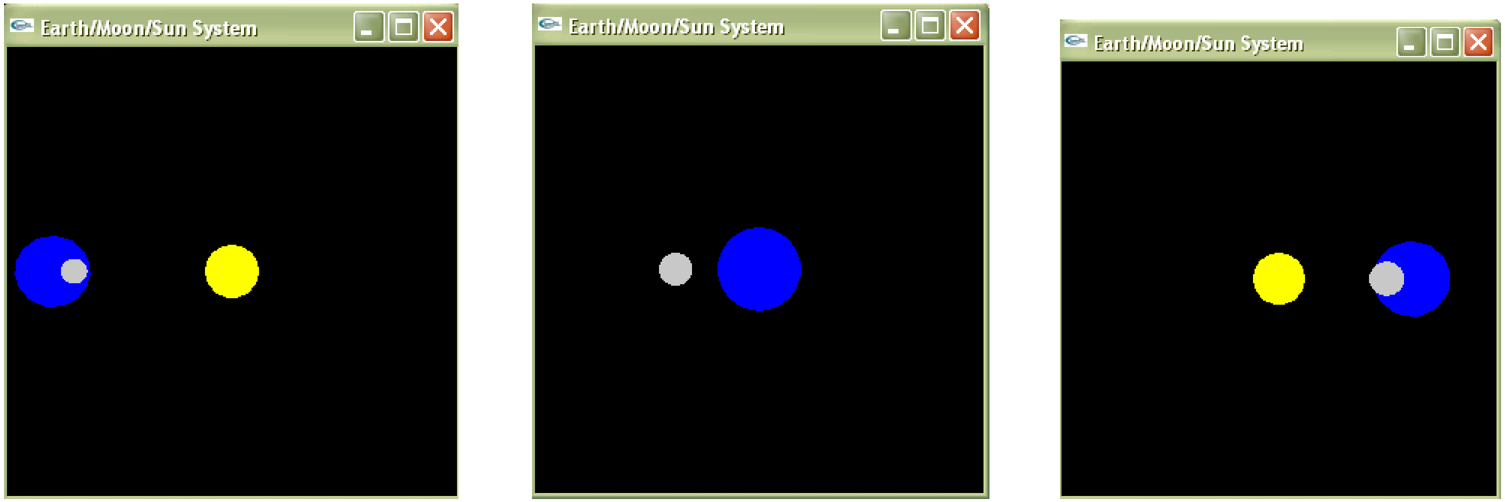
  gluPerspective(fov, aspect, near, far);

  - fov = vertical field of view in degrees
  - aspect = image width / height at near depth

Properties of Perspective Projections

- The perspective projection is an example of a projective transformation
- Here are some properties of projective transformations:
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
- One of the advantages of perspective projection is that size varies inversely with distance – looks realistic
- A disadvantage is that we can't judge distances as exactly as we can with parallel projections

Sun/Earth/Moon System




```
#include <gl/glut.h>
GLfloat whiteLight[]={0.2, 0.2, 0.2, 1.0};
GLfloat sourceLight[]={0.8, 0.8, 0.8, 1.0};
GLfloat lightPos[]={0.0, 0.0, 0.0, 1.0};
```

```
void display(void) {

    static float fMoonRot=0.0;
    static float fEarthRot=0.0;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

        //Translate the whole scene out and into view
        glTranslatef(0.0, 0.0, -300.0);

        //Sun
        glColor3ub(255, 255,0);
        glutSolidSphere(15.0, 15, 15);

        //Move the light after we draw the sun!
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

        //Rotate coordinate system
        glRotatef(fEarthRot, 0.0, 1.0, 0.0);
```

```
        //Draw the Earth
        glColor3ub(0, 0, 255);
        glTranslatef(105.0, 0.0, 0.0);
        glutSolidSphere(15.0, 15, 15);

        //Rotate from Earth based coordinates and draw Moon
        glColor3ub(200, 200, 200);
        glRotatef(fMoonRot, 0.0, 1.0, 0.0);
        glTranslatef(30.0, 0.0, 0.0);
        fMoonRot += 15.0;
        if (fMoonRot > 360.0)
                fMoonRot=0.0;
        glutSolidSphere(6.0, 15, 15);
    glPopMatrix();

    fEarthRot += 15.0;
    if (fEarthRot > 360.0)
        fEarthRot=0.0;

    glutSwapBuffers();
}
```

```
void setup() {
    glEnable(GL_DEPTH_TEST); //hidden surface removal
    glFrontFace(GL_CCW); //counter clock-wise polygons face out
    glEnable(GL_CULL_FACE); //Do not calculate inside of jet

    glEnable(GL_LIGHTING);

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, whiteLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, sourceLight);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glClearColor(0.0, 0.0, 0.0, 1.0);
}
```

```
void timer(int value) {
    glutPostRedisplay();
    glutTimerFunc(100, timer, 1);
}
void resize(int w, int h) {
    GLfloat fAspect;
    if (h==0) h=1;

    glViewport(0, 0, w, h);
    fAspect=(GLfloat) w/(GLfloat) h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(45.0, fAspect, 1.0, 425.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
int main(int argc, char* argv[]) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE |
                              GLUT_RGB | GLUT_DEPTH);
   glutCreateWindow("Earth/Moon/Sun System");
   glutReshapeFunc(resize);
   glutDisplayFunc(display);
   glutTimerFunc(500, timer, 1);
   setup();
   glutMainLoop();

   return 0;
}
```