# Perspective Projection

OpenGL

# Learning Outcomes

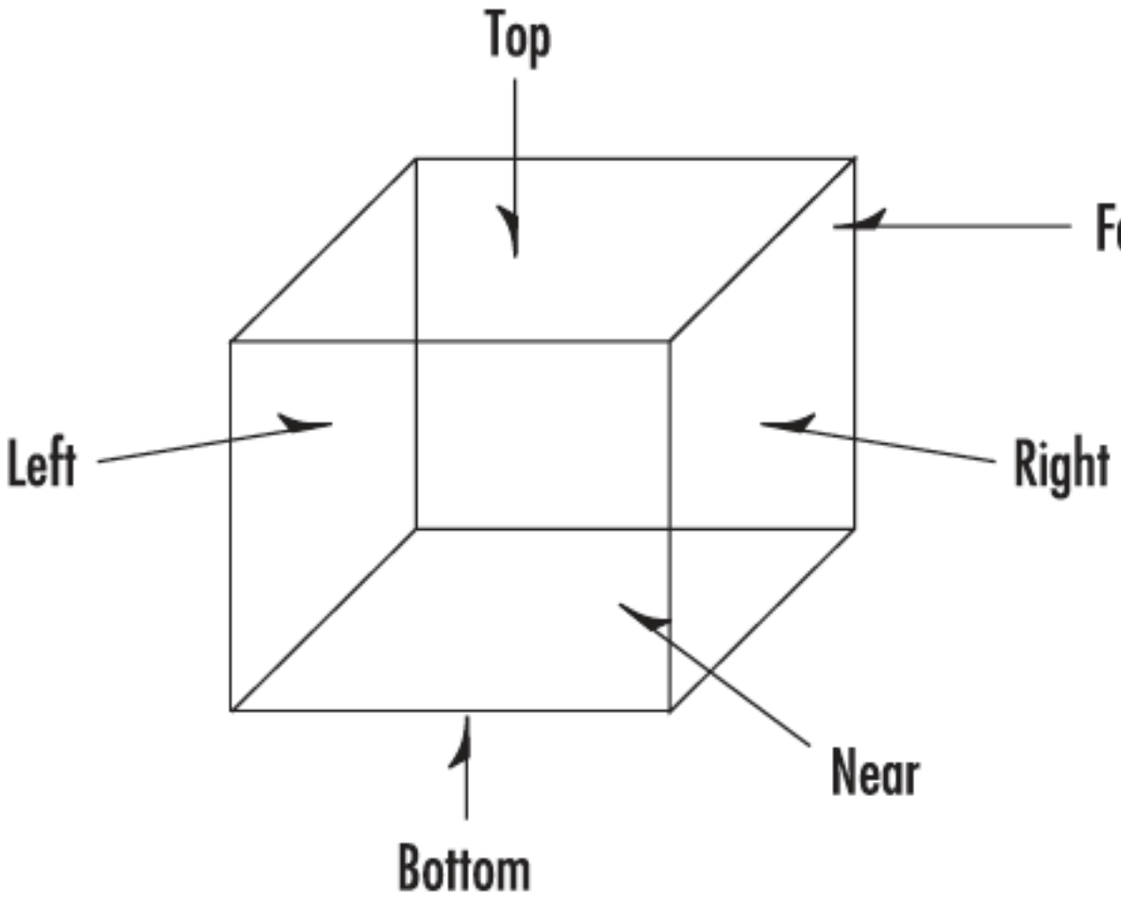
- Understand the Perspective Projection, and be able to compare it to the Orthographic projections

- Understand frustum, field of view angle, aspect ratio and near and far clipping planes in this context.

- Review again simple animations using rotate and translate

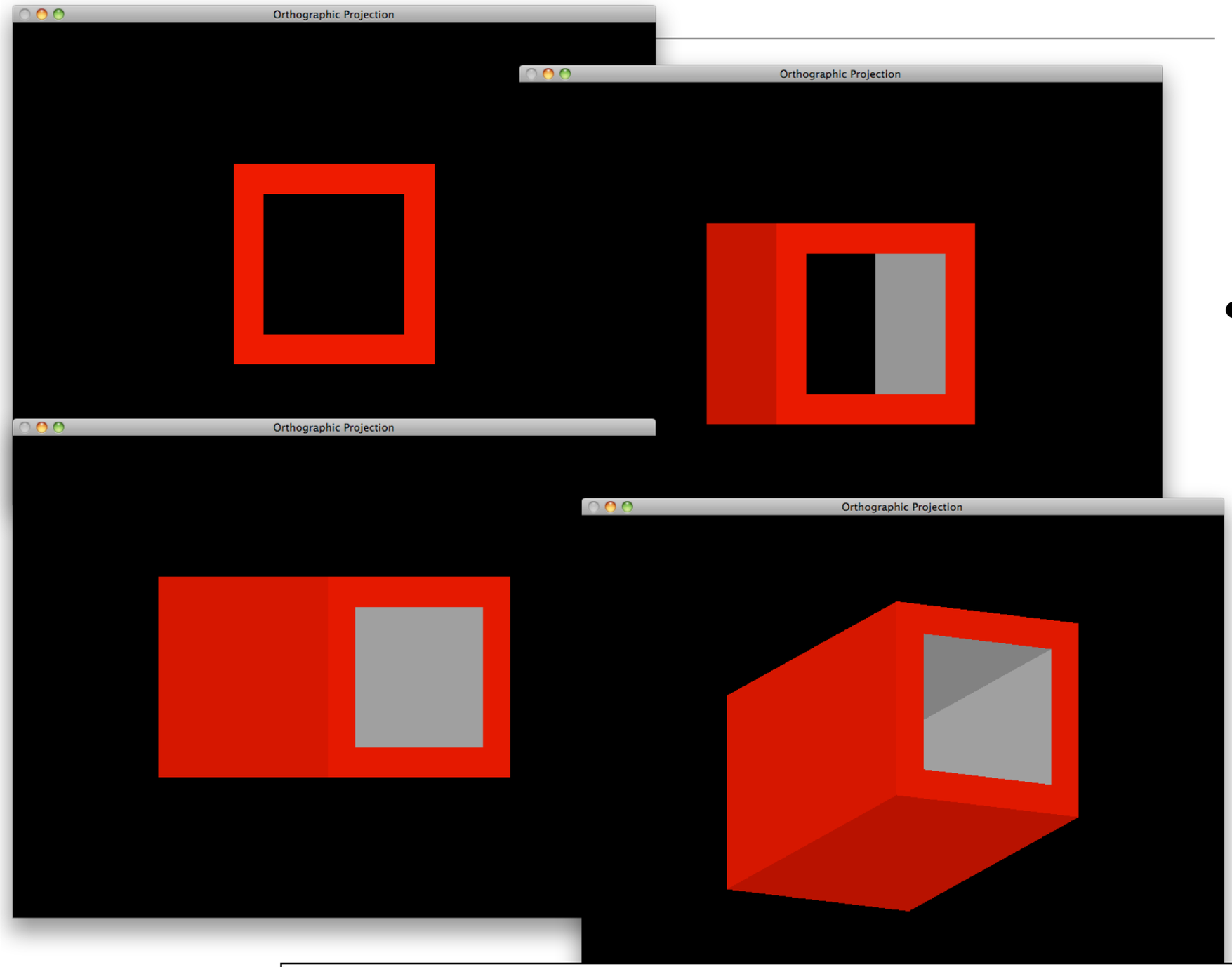
- Review a simple method of moving a camera around a scene

# Orthographic Projections




- This projection by specifying a square or rectangular viewing volume. Anything outside this volume is not drawn.
- Furthermore,all objects that have the same dimensions appear the same size, regardless of whether they are far away or nearby.
- This type of projection  is most often used in architectural design, computer-aided design (CAD), or 2D graphs.
- Specify the viewing volume in an orthographic projection by specifying the far, near, left, right, top, and bottom clipping planes.
- Objects and figures that you place within this viewing volume are then projected (taking into account their orientation) to a 2D image that appears on your screen

# glOrtho Projections

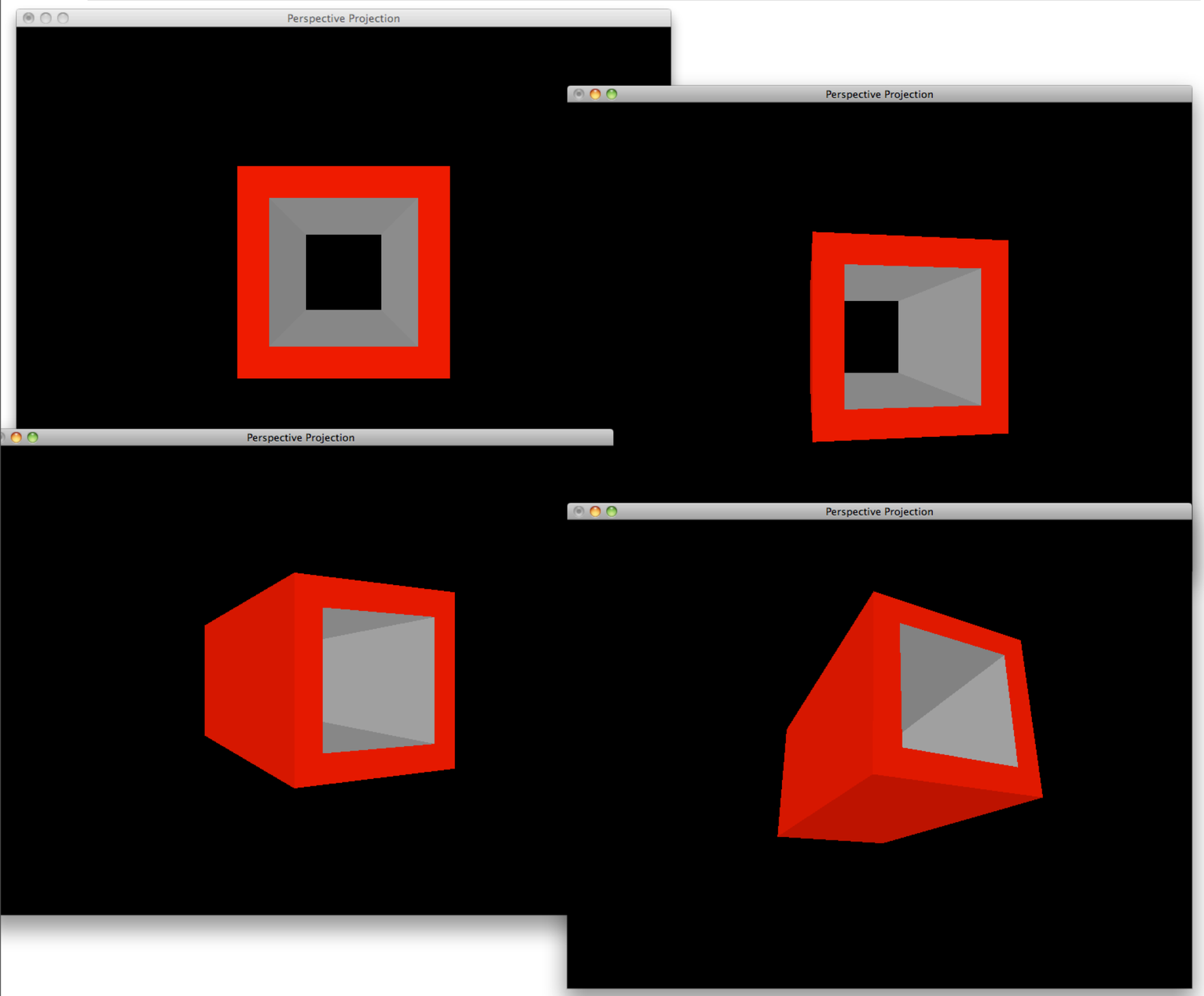



- A parallel viewing volume specified by the function glOrtho, setting the near and far, left and right, and top and bottom clipping coordinates.

- Because the tube does not converge in the distance, this is not an entirely accurate view of how such a tube appears in real life. To add some perspective, we must use a perspective projection.

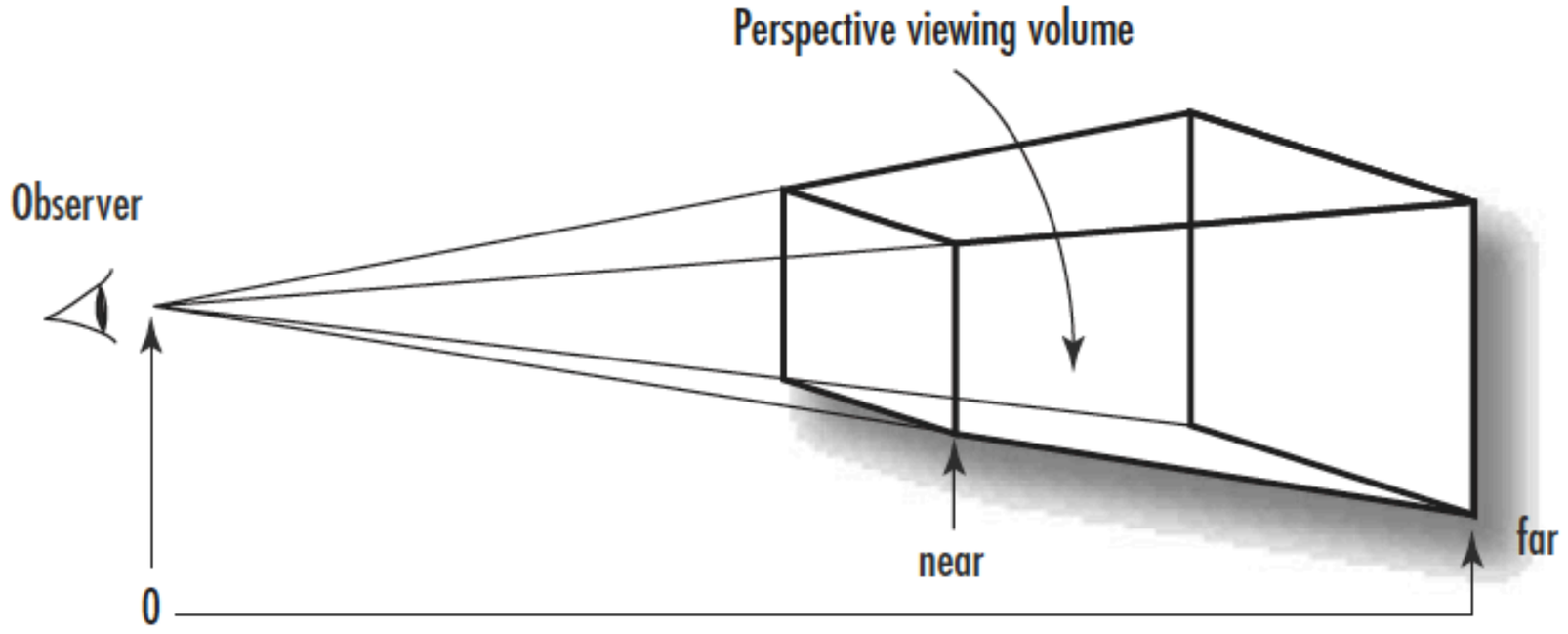
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho (-100.0f, 100.0f, -100.0f, 100.0f, -200.0f, 200.0f);
```

# Perspective Projections




- A perspective projection performs perspective division to shorten and shrink objects that are farther away from the viewer.
- The width of the back of the viewing volume does not have the same measurements as the front of the viewing volume after being projected to the screen.
- Thus, an object of the same logical dimensions appears larger at the front of the viewing volume than if it were drawn at the back of the viewing volume.

# Fustrum

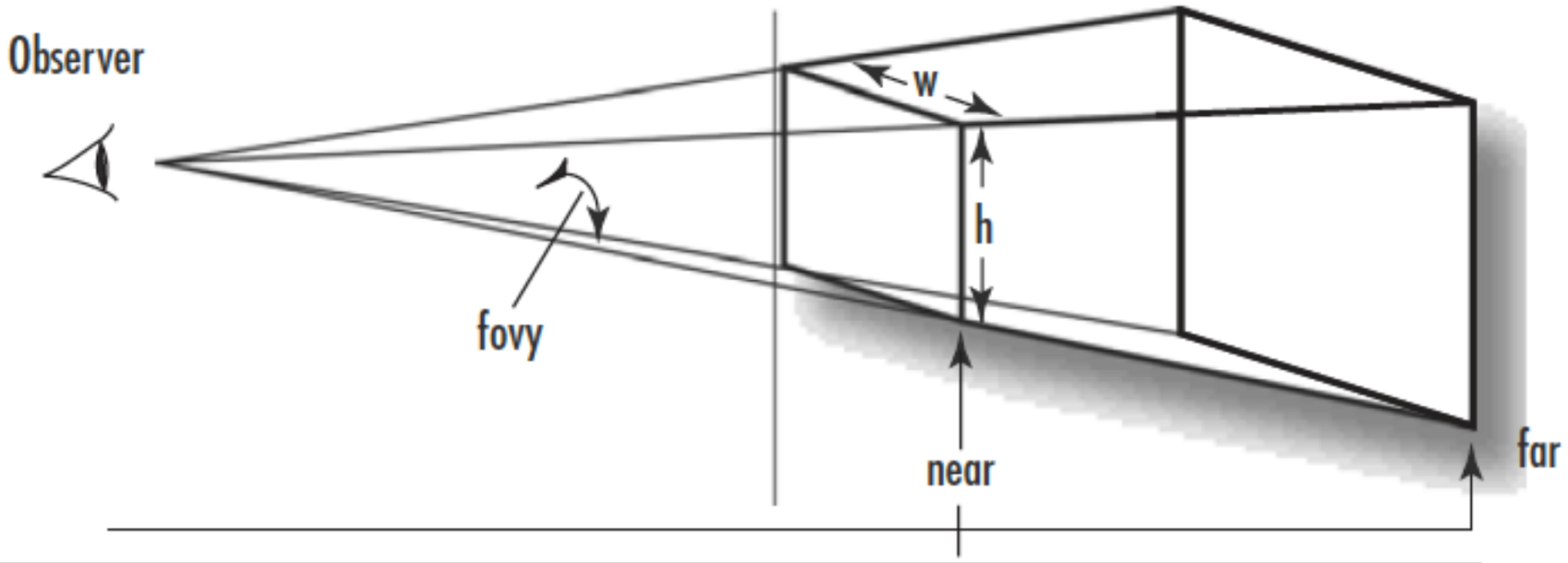



- Frustum: a truncated section of a pyramid viewed from the narrow end to the broad end.

- Define a frustum with the function glFrustum - specifying the coordinates and distances between the front and back clipping planes.

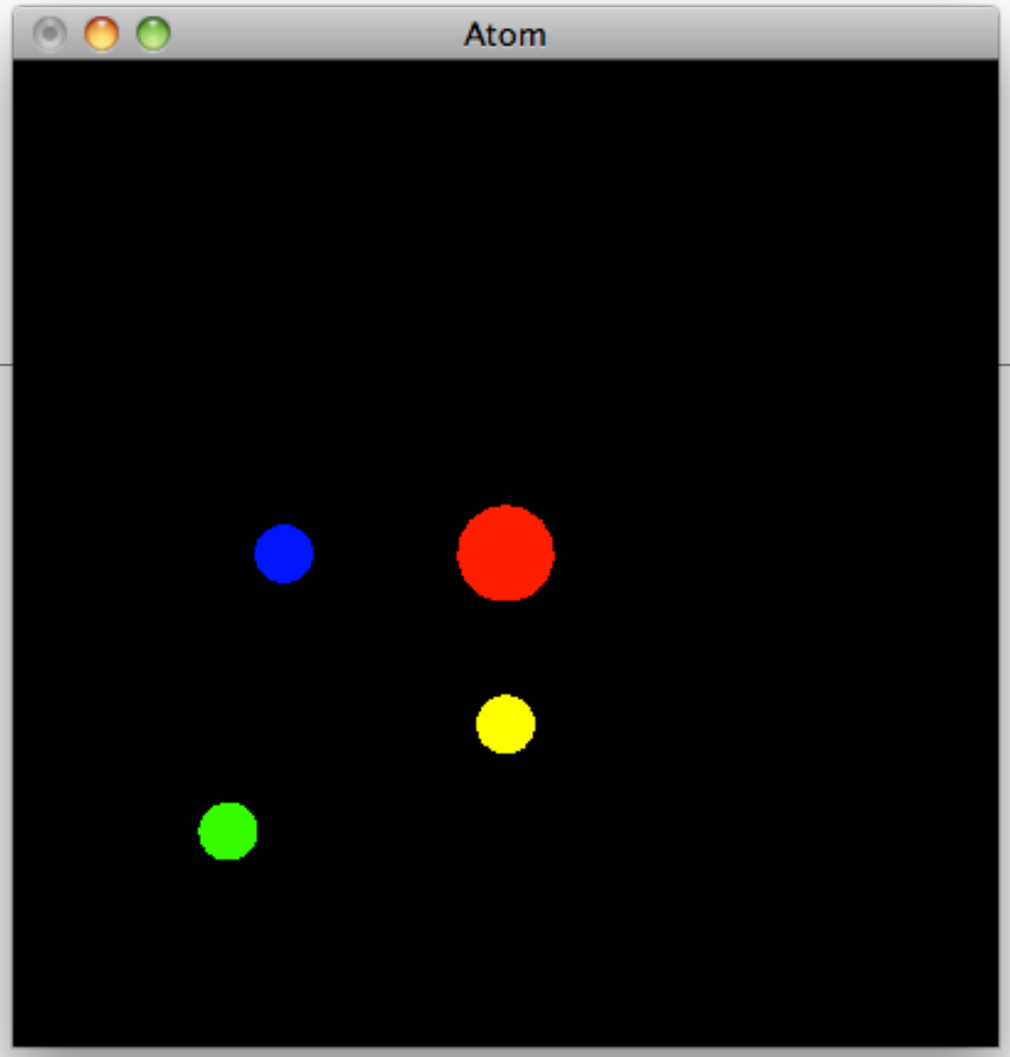
- However, glFrustum is not as intuitive about setting up your projection to get the desired effects, and is typically used for more specialized purposes (for example, stereo, tiles, asymmetric view volumes).

# gluPerspective



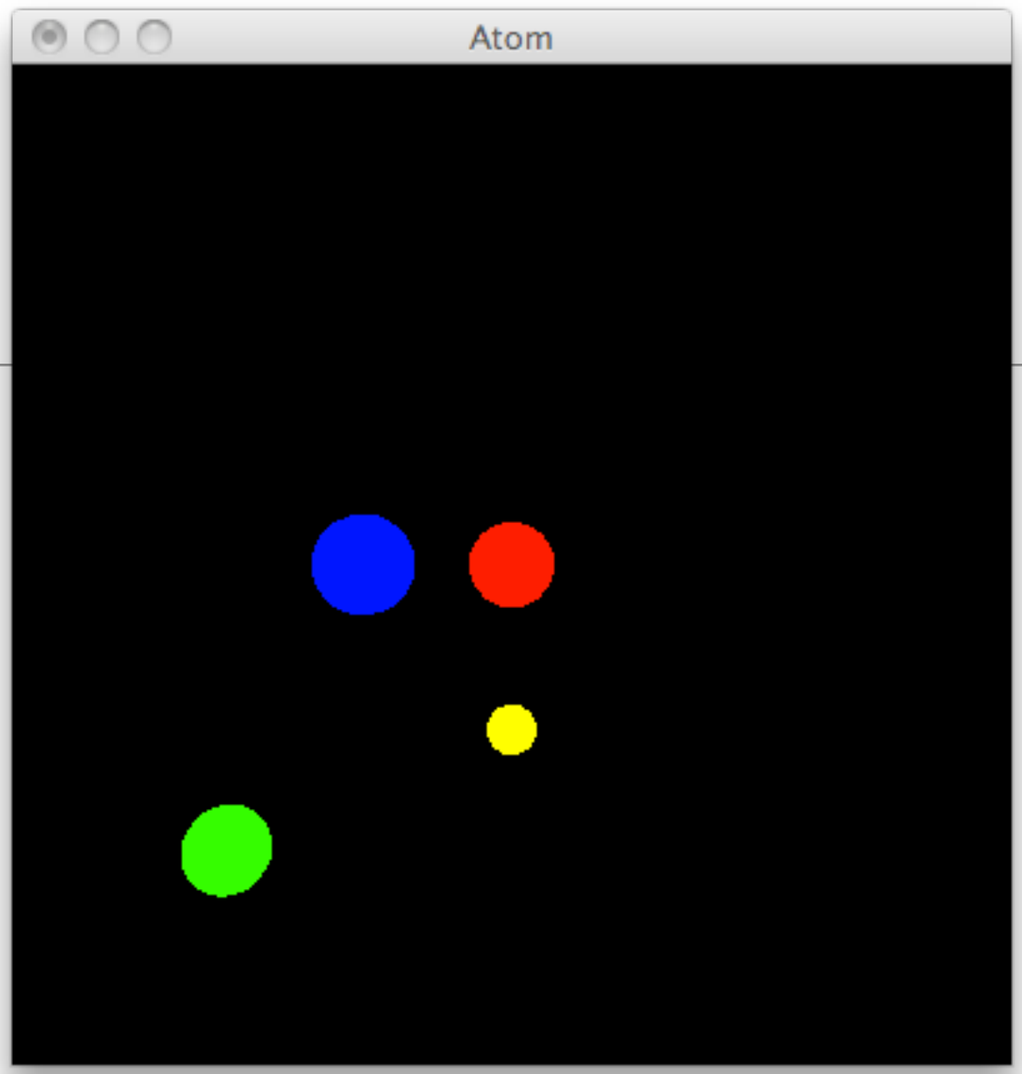

```
void gluPerspective(GLdouble fovy, GLdouble aspect,GLdouble zNear, GLdouble zFar);
```

- Parameters :
  - field-of-view angle (fovy)
  - aspect ratio of the width to height
  - distances to the near and far clipping planes
- aspect ratio is calculated by dividing the width (w) by the height(h) of the window or viewport.




- Orthographic

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho (-100.0f, 100.0f,
         -100.0f, 100.0f,
         -200.0f, 200.0f);

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

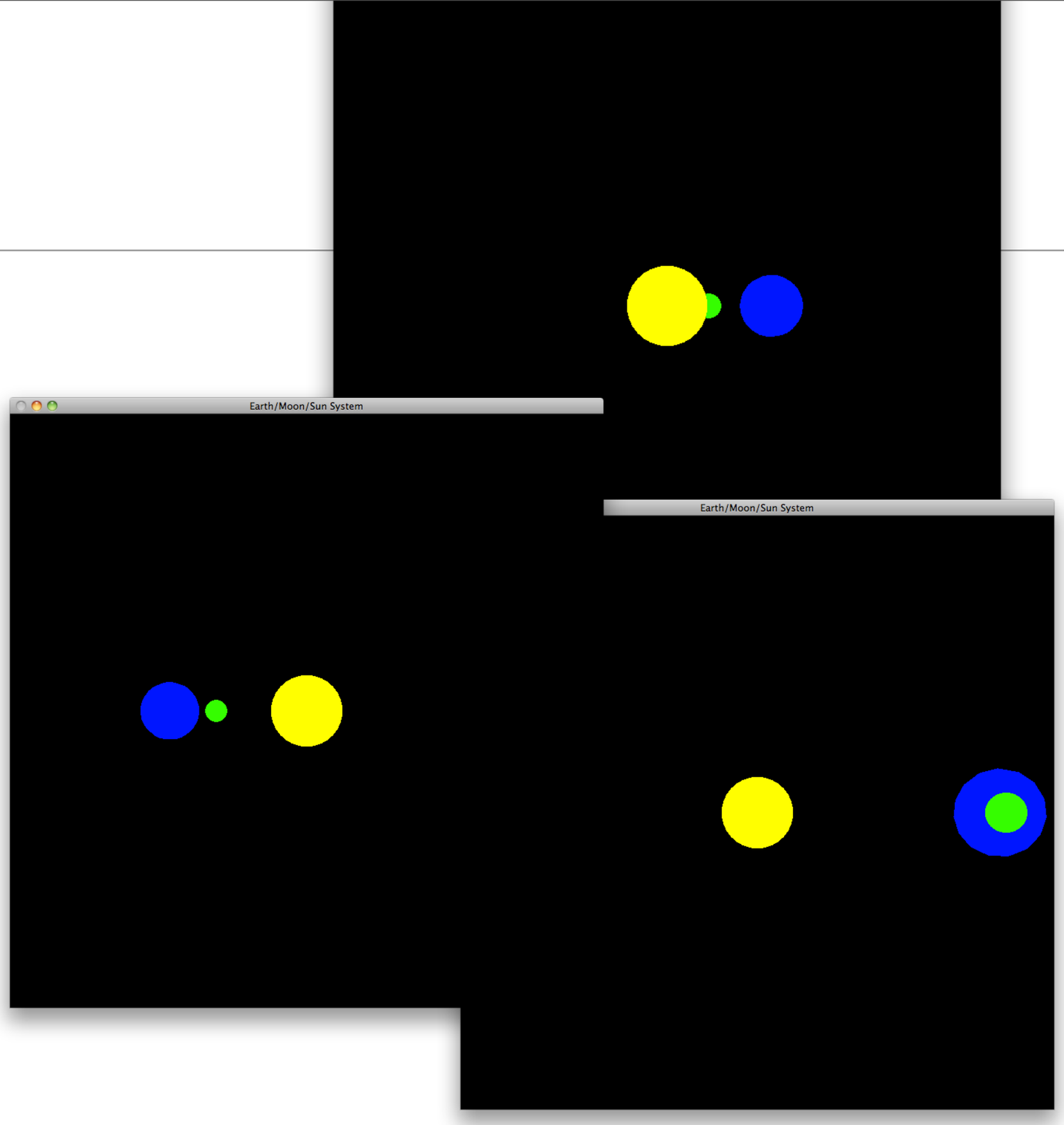



- Perspective

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, 1, 50.0, 400.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -200.0f);
```

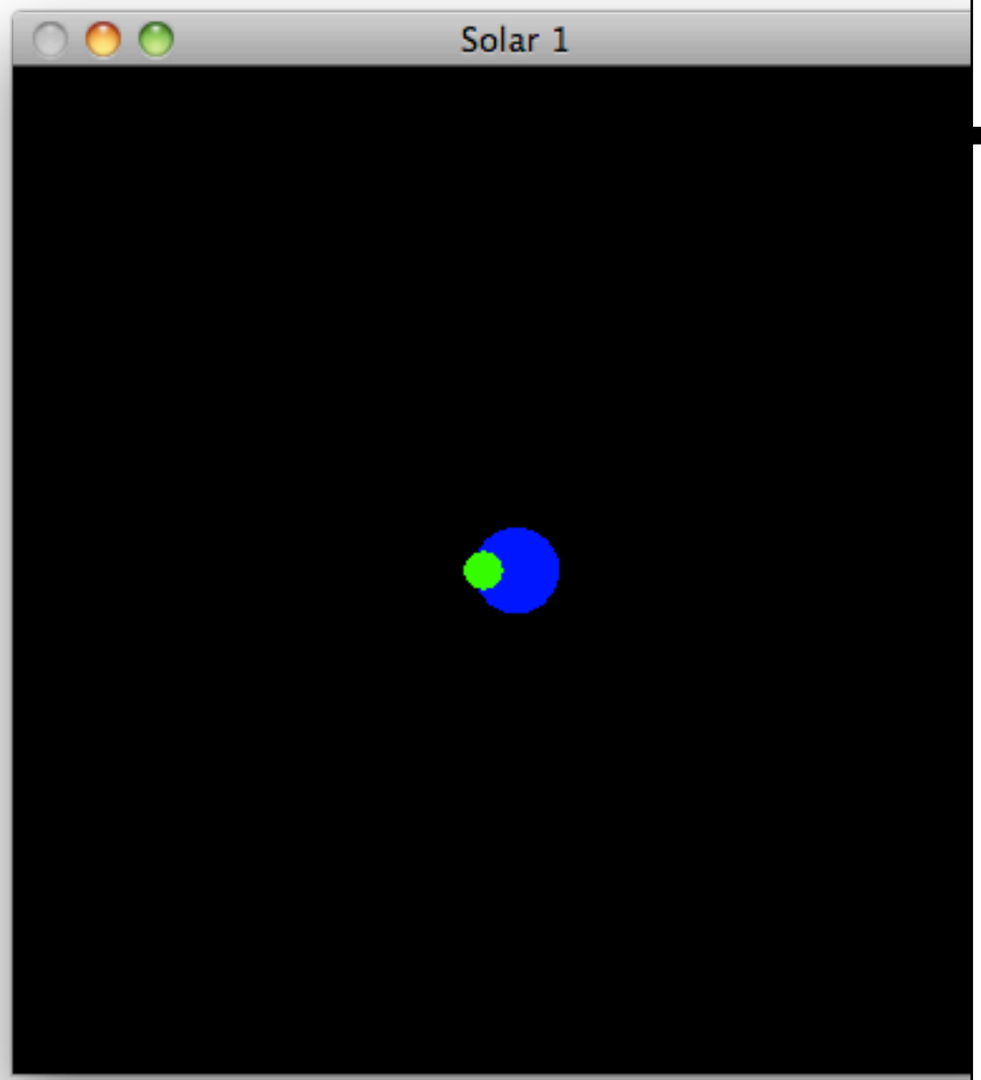
# Solar System

- Moon (green) orbits Earth (blue)
- Earth orbits Sun (Yellow)



```
void renderSolarSystem(void)
{
  static int moonRot = 0;
  static int earthRot = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
    Vector3(0.0f, 0.0f, -300.0f).translate();
    Color::Yellow.render();
    glutSolidSphere(15.0f, 30, 17);
    renderEarthMoon(earthRot, moonRot);
  glPopMatrix();

  moonRot  = (moonRot + 10) % 360;
  earthRot = (earthRot + 5) % 360;

  glutSwapBuffers();
}
```

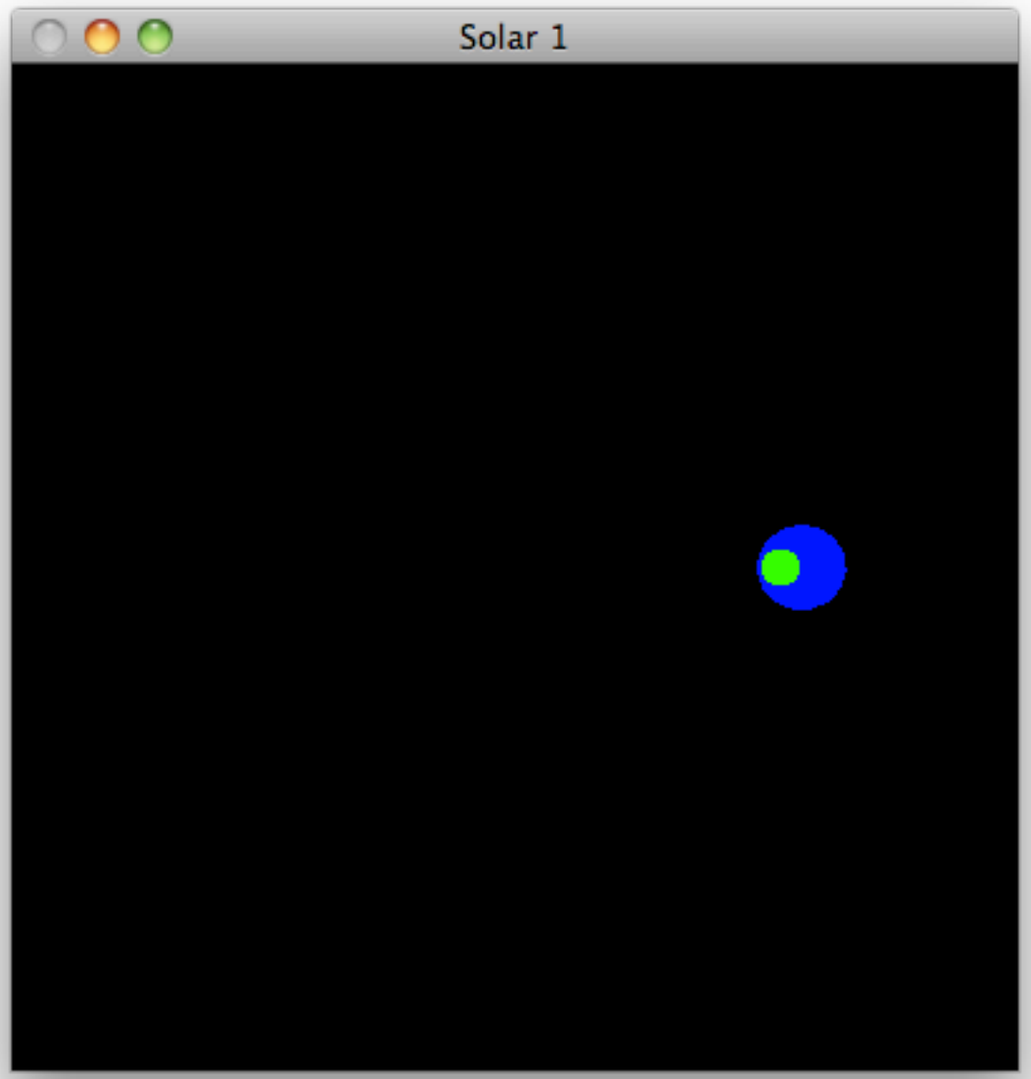
# renderEarthMoon




```
void renderEarthMoon(int moonAngle)
{
  glPushMatrix();
    Color::Blue.render();
    glutSolidSphere(15, 15, 15);

    Color::Green.render();
    Vector3::UnitY.rotate(moonAngle);
    Vector3(30.0f, 0.0f, 0.0f).translate();
    glutSolidSphere(6.0f, 30, 17);
  glPopMatrix();
}
```

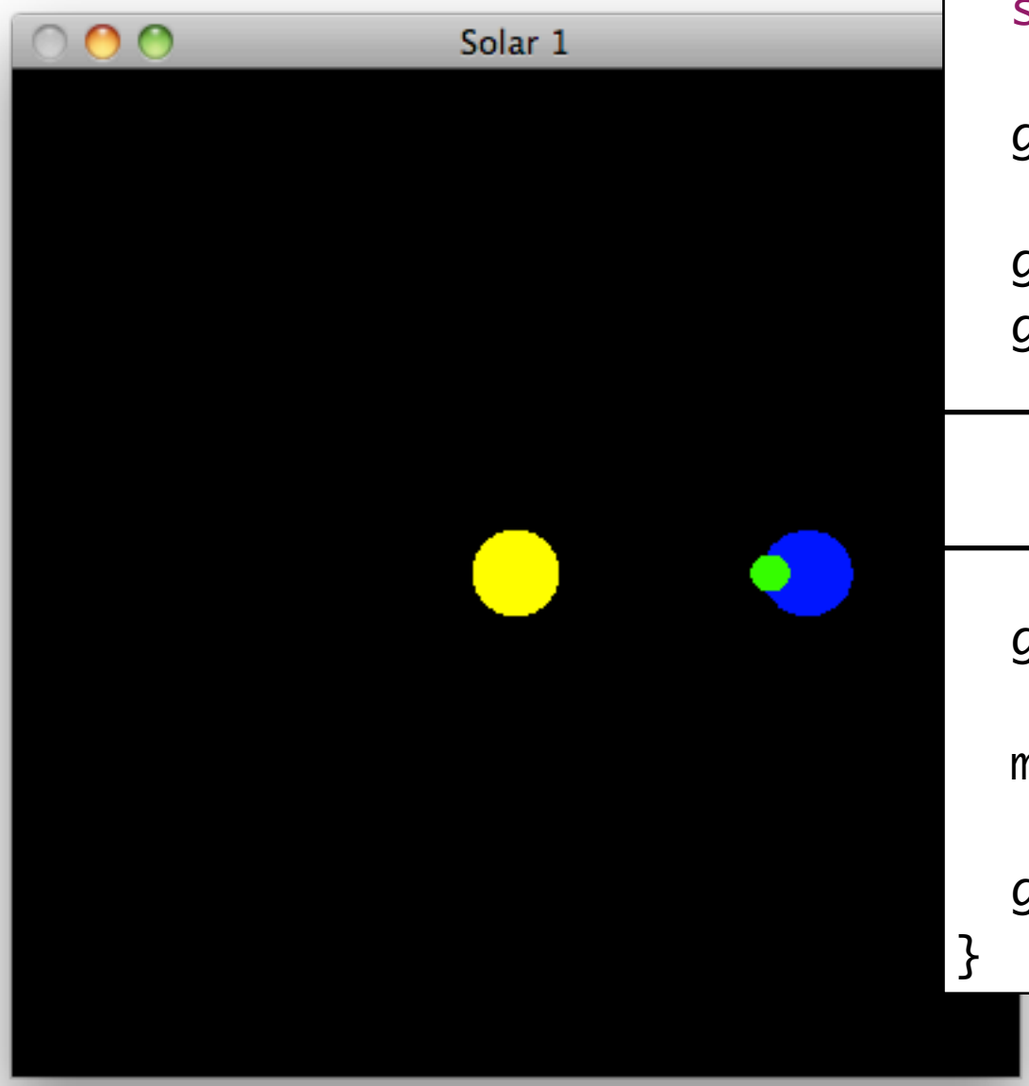
```
void renderSolarSystem(void)
{
  static int moonRot = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
    Vector3(0.0f, 0.0f, -300.0f).translate();
    renderEarthMoon(moonRot);
  glPopMatrix();

  moonRot  = (moonRot + 10) % 360;

  glutSwapBuffers();
}
```

# Move Earth/Moon




```
void renderEarthMoon(int earthAngle, int moonAngle)
{
  glPushMatrix();
    Vector3(100,0,0).translate();
    Color::Blue.render();
    glutSolidSphere(15, 15, 15);

    Color::Green.render();
    Vector3::UnitY.rotate(moonAngle);
    Vector3(30.0f, 0.0f, 0.0f).translate();
    glutSolidSphere(6.0f, 30, 17);
  glPopMatrix();
}
```

# Introduce Sun

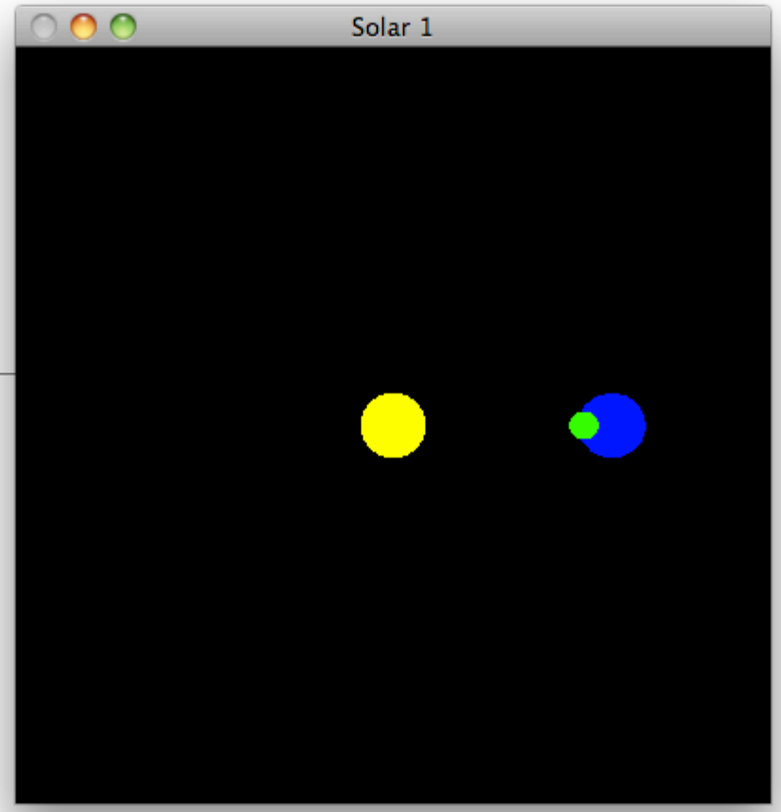



```
void renderSolarSystem(void)
{
  static int moonRot = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
    Vector3(0.0f, 0.0f, -300.0f).translate();
    Color::Yellow.render();
    glutSolidSphere(15.0f, 30, 17);
    renderEarthMoon(earthRot, moonRot);
  glPopMatrix();

  moonRot  = (moonRot + 10) % 360;

  glutSwapBuffers();
}
```

# Paramaterise Earth Rotation

```
void renderEarthMoon(int earthAngle, int moonAngle)
{
  glPushMatrix();
    Vector3::UnitY.rotate(earthAngle);
    Vector3(100,0,0).translate();
    Color::Blue.render();
    glutSolidSphere(15, 15, 15);

    Color::Green.render();
    Vector3::UnitY.rotate(moonAngle);
    Vector3(30.0f, 0.0f, 0.0f).translate();
    glutSolidSphere(6.0f, 30, 17);
  glPopMatrix();
}
```

# Animate




```
void renderSolarSystem(void)
{
  static int moonRot = 0;
  static int earthRot = 0;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
    Vector3(0.0f, 0.0f, -300.0f).translate();
    Color::Yellow.render();
    glutSolidSphere(15.0f, 30, 17);
    renderEarthMoon(earthRot, moonRot);
  glPopMatrix();

  earthRot = (earthRot + 5) % 360;
  moonRot  = (moonRot + 10) % 360;

  glutSwapBuffers();
}
```

```
void renderEarthMoon(int earthAngle, int
moonAngle)
{
  glPushMatrix();
    Vector3::UnitY.rotate(earthAngle);
    Vector3(100,0,0).translate();
    Color::Blue.render();
    glutSolidSphere(15, 15, 15);

    Color::Green.render();
    Vector3::UnitY.rotate(moonAngle);
    Vector3(30.0f, 0.0f, 0.0f).translate();
    glutSolidSphere(6.0f, 30, 17);
  glPopMatrix();
}
```

# Special Keys & Camera Movement

```
void specialKeys(int key, int x, int y)
{
  int up=0, down=0;
  int left=0, right=0;
  int in=0, out=0;

  up    = (key == GLUT_KEY_UP)?       -5 : 0;
  down  = (key == GLUT_KEY_DOWN)?      5 : 0;
  left  = (key == GLUT_KEY_LEFT)?      5 : 0;
  right = (key == GLUT_KEY_RIGHT)?    -5 : 0;
  in    = (key == 9)?         5 : 0;   // tab
  out   = (key == 32)?        -5 : 0;  // space

  glTranslatef(left+right, up+down, in+out);
  glutPostRedisplay();
}
```