# OPENGL – PART III

OpenGL is in a 3D graphics mode by default. When the program starts, the coordinate system is centered on the origin, or X=0, Y=0, Z=0. The positive X axis extends to the right of the center of the window, the positive Y axis extends upward from the center of the screen, and the positive Z axis points into the screen. This means that the window represents the XY plane where Z=0 and positive Z values are further into the screen.

**glClearColor(1.0, 0.0, 0.0, 1.0)**. The alpha component specifies transparency and should be left at 1.0 for opaque colors.

The **RenderSetup** function contains the first difference from previous programs. The first new function is **glMatrixMode(GL_PROJECTION)**. Previously, we've been working in the **GL_MODELVIEW** matrix and haven't known it! These two matrices describe the viewing state of the program. When drawing 2D models, we only needed the default viewing matrix, **GL_MODELVIEW**. In 3D, the **GL_PROJECTION** matrix must be modified so OpenGL can project, or rasterize, the 3D object onto a 2D window.

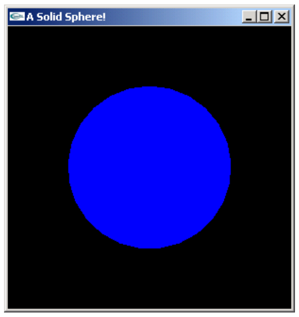
Visualizing a movie set is an easy way to understand the **GL_MODELVIEW** and **GL_PROJECTION** matrices. The geometric objects given to the renderer similar to props on a movie set. The **GL_MODELVIEW** matrix describes how the set is viewed from the camera (hence MODEL VIEW). In OpenGL, the camera begins at the origin and is pointed towards the +Z axis. Functions such as **glRotate*** and **glTranslate*** affect the **GL_MODELVIEW** matrix to rotate or move the camera view about the set. The **GL_PROJECTION** matrix describes how the scene is projected onto the computer screen (hence PROJECTION). This is similar to the lens in a projector.

The **gluPerspective** function can be used to adjust parameters such as the aspect ratio of the window (the ratio of heigth to width) or the field of view.

```
gluPerspective(90.0, 1.0, 1.0, 50.0);
```

The first two arguments of **gluPerspective** set the field of view to 90.0 degrees and the aspect ratio (window.width / window.height) to 1.0. The next two arguments specify the near Z clipping plane and the far Z clipping plane. In this case, the near Z clipping plane is 1.0. This means that all polygons greater than 1.0 units away from the camera can be seen, while polygons less than 1.0 units away cannot be seen. The far clipping plane is 50.0, and this means that all polygons more than 50.0 units away from the camera are not seen, while polygons closer than 50.0 units away can be seen.

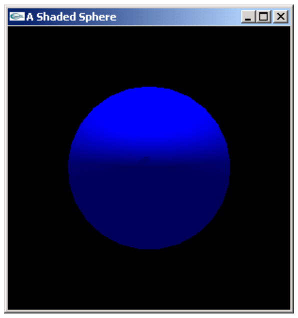
**Örnek 1 :** Mavi bir küre



```
#include <GL/glut.h>

void RenderSetup(void);
void DisplayCallback(void);

GLfloat blue[] = { 0.0, 0.0, 1.0, 1.0 };     /* blue */

void main(int argc, char **argv)
{
  /* Initialize the a window's properties and open the window */
  glutInit(&argc, argv);
  glutInitDisplayMode( GLUT_RGBA | GLUT_SINGLE );
  glutCreateWindow("A Solid Sphere!");

  /* Setup the rendering context for the scene */
  RenderSetup();

  /* Declare the display function and then enter the main loop */
  glutDisplayFunc( DisplayCallback );
  glutMainLoop();
 }

/* Set up the rendering context for the scene */
void RenderSetup(void)
{
  glClearColor(0.0, 0.0, 0.0, 1.0);

  /* Load a perspective matrix onto the projection stack */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(90.0, 1.0, 1.0, 50.0);
}

void DisplayCallback(void)
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  glClear(GL_COLOR_BUFFER_BIT);
  glTranslatef(0.0, 0.0, -10.0);

  glColor4fv(blue);
  glutSolidSphere(5.0, 25.0, 25.0);

  glFlush();
}
```

**Örnek 2 :** Aydınlatılmış Mavi bir küre

```
/*********************************************************************
 * shadesphere.c
 *    This program shows a blue sphere with lighting effects.
 **/

#include <GL/glut.h>

void RenderSetup(void);
void InitVariables(void);
void DisplayCallback(void);

/* Define a struct of two floats */
typedef struct { float X, Y; } vector2f;

vector2f window;          /* Window size */

GLfloat blue[] = { 0.0, 0.0, 1.0, 1.0 };     /* blue */

void main(int argc, char **argv)
{
  glutInit(&argc, argv);  /* Initialize the glut library */
  InitVariables();        /* Initialize program variables */

  /* Open a window for display */
  glutInitWindowSize(window.X, window.Y);
  glutInitDisplayMode( GLUT_RGBA | GLUT_SINGLE );
  glutCreateWindow("A Shaded Sphere");

  RenderSetup();    /* Setup the rendering context */

  /* Declare the display function and then enter the main loop */
  glutDisplayFunc( DisplayCallback );

  glutMainLoop();

}

/* Initialize program variables */
void InitVariables(void)
{
  /* Set the initial window size */
  window.X = 300;
  window.Y = 300;
}
```



```
/* Set up the rendering context for the scene */
void RenderSetup(void)
{
  GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat light_ambient[] = { .15, .15, .15, 1.0 };
  GLfloat light_pos[] = { 0.0, 100.0, 0.0 };

  glClearColor(0.0, 0.0, 0.0, 1.0);

  /* Load a perspective matrix onto the projection stack */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(90.0, 1.0, 1.0, 50.0);

  /* Enable lighting */
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
}

void DisplayCallback(void)
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glClear( GL_COLOR_BUFFER_BIT );

  /* Position the camera */
  glTranslatef(0.0, 0.0, -10.0);

  /* Set the color and material of the sphere, then draw */
  glColor4fv(blue);
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, blue);
  glutSolidSphere(5.0, 25.0, 25.0);
}
```

This program uses ambient and diffuse lighting to illuminate a blue sphere. The first two lines in **RenderSetup** define the color and intensity of the ambient and diffuse light. The variables are defined in (R, G, B, A) format. Usually, the ambient light is less intense than the diffuse light, because this makes highlights appear more strongly. The next line defines the position of the light in (X, Y, Z) format. The diffuse in this program is defined to be 100 units above the sphere.

Now that the variables have been set up, it is time to enable and initialize the light. The first function, **glEnable(GL_LIGHTING)**, enables lighting for the rest of the program. After this, the specific light the program will use is enabled with **glEnable(GL_LIGHT0)**. The OpenGL standard states that up to eight lights (numbered 0 to 7) can be enabled. Finally the ambient, diffuse, and position characteristics are set using the earlier defined variables.

In everyday life, you can look around and see many different types of materials. Metal shines with bright highlights in light, while cloth doesn't shine at all. How can these characteristics be put into an OpenGL program? The answer is material properties. Material properties dictate how a surface will interact with different types of light. In the example program, the ambient and diffuse characteristics are set to show blue when illuminated. Blue is also the color of the sphere, so setting the material properties to blue would make sense. Often, the material properties are set to the same value as the color of the object.

**Örnek 3 :** Güneş Sistemi Oluşturulması

## Building a Solar System

The program described in this section draws a simple solar system with a planet and a sun, both using the same sphere-drawing routine. To write this program, you need to use **glRotate***() for the revolution of the planet around the sun and for the rotation of the planet around its own axis. You also need **glTranslate***() to move the planet out to its orbit, away from the origin of the solar system. Remember that you can specify the desired size of the two spheres by supplying the appropriate arguments for the **glutWireSphere()** routine. To draw the solar system, you first want to set up a projection and a viewing transformation. For this example, **gluPerspective()** and **gluLookAt()** are used. Drawing the sun is straightforward, since it should be located at the origin of the grand, fixed coordinate system, which is where the sphere routine places it. Thus, drawing the sun doesn't require translation; you can use **glRotate***() to make the sun rotate about an arbitrary axis. To draw a planet rotating around the sun, as shown in Figure 3-24, requires several modeling transformations. The planet needs to rotate about its own axis once a day. And once a year, the planet completes one revolution around the sun.

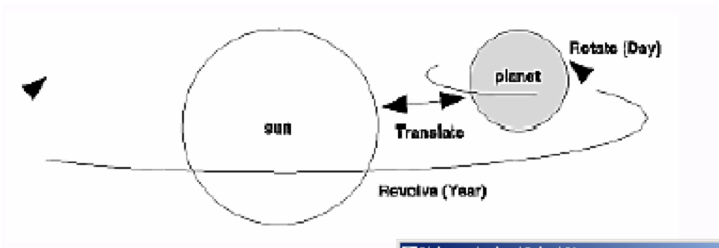
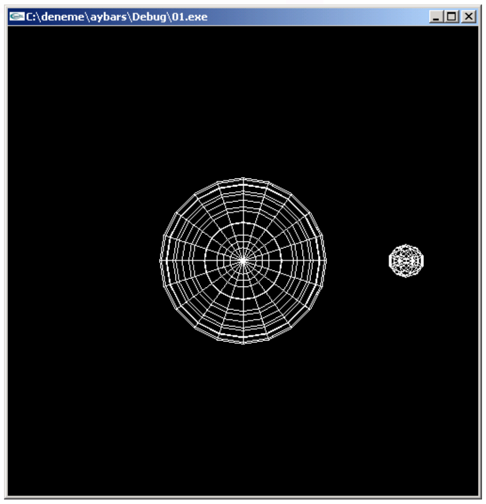


**Figure 3-24 :** Planet and Sun



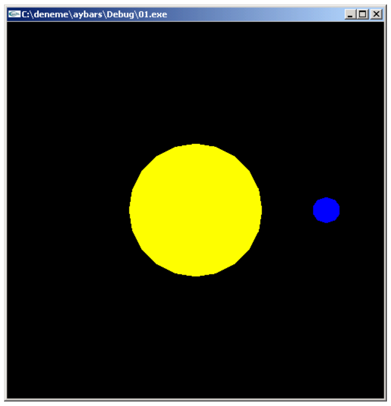
```
#include <GL/glut.h>
static int year = 0, day = 0;
void init(void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_FLAT);
}
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
glPushMatrix();
glutWireSphere(1.0, 20, 16); /* draw sun */
glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
glTranslatef (2.0, 0.0, 0.0);
glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
glutWireSphere(0.2, 10, 8); /* draw smaller planet */
glPopMatrix();
glutSwapBuffers();

}
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);
glLoadIdentity();
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

void keyboard (unsigned char key, int x, int y)
{
switch (key) {
case 'd':
day = (day + 10) % 360;
glutPostRedisplay();
break;
case 'D':
day = (day - 10) % 360;
glutPostRedisplay();
break;
case 'y':
year = (year + 5) % 360;
glutPostRedisplay();
break;
case 'Y':
year = (year - 5) % 360;
glutPostRedisplay();
break;
default:
break;
}
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMainLoop();
return 0;
}
```

In OpenGl, we set up a viewing system with the following three parameters: a view reference position (also called the viewpoint, or "eye" position) $\mathbf{P}_0$, a "look-at" position $\mathbf{P}_{ref}$, and the view-up vector **V** with the function

```
gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

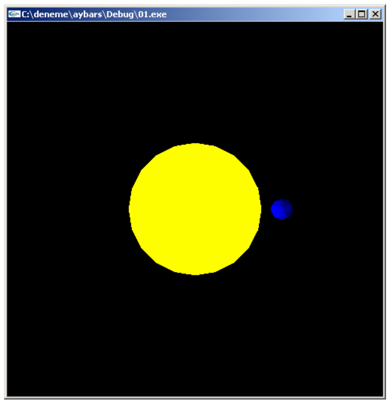
The direction of viewing is then given by the vector $\mathbf{P}_{\text{ref}}$ - $\mathbf{P}_0$, from the viewpoint to the look-at point. Default values are the world-coordinate origin (0, 0, 0) for the viewpoint, the negative z axis for the viewing direction, and the world-coordinate y direction for the view-up vector. A viewing matrix is set up for these parameters and is pre-multiplied by the current matrix. This function is in the Utility Library because it uses a combination of rotation and translation routines to produce the matrix for transforming from world to viewing coordinates.

**Örnek 4 :** Güneş Sistemi Oluşturulması : Katı Küreler



```
#include <GL/glut.h>
static int year = 0, day = 0;
void init(void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_SMOOTH);
//      glEnable (GL_CULL_FACE);
        glEnable (GL_DEPTH_TEST); //

}
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
glPushMatrix();

glColor3f (1.0, 1.0, 0.0); //
glutSolidSphere(1.0, 20, 16); /* draw sun */ //
glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);

glColor3f (0.0, 0.0, 1.0); //
glTranslatef (2.0, 0.0, 0.0);
glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
glutSolidSphere(0.2, 10, 8); /* draw smaller planet */ //
glPopMatrix();
glutSwapBuffers();

}
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);
glLoadIdentity();
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
switch (key) {
case 'd':
day = (day + 10) % 360;
glutPostRedisplay();
break;
case 'D':
day = (day - 10) % 360;
glutPostRedisplay();
break;
case 'y':
year = (year + 5) % 360;
glutPostRedisplay();
break;
case 'Y':
year = (year - 5) % 360;
glutPostRedisplay();
break;
default:
break;
}
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMainLoop();
return 0;
}
```

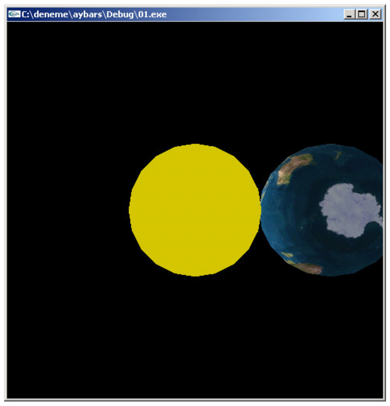
**Örnek 5 :** Güneş Sistemi Oluşturulması : Aydınlatmalı ve Kamera Hareketli



```
#include <GL/glut.h>
static int year = 0, day = 0;

float z=0.0;

void init(void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_FLAT);
//        glEnable (GL_CULL_FACE);
   glCullFace(GL_FRONT);
         glEnable (GL_DEPTH_TEST); //
}
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glPushMatrix();


glColor3f (1.0, 1.0, 0.0); //
glutSolidSphere(1.0, 20, 16); /* draw sun */ //

  GLfloat light_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
  GLfloat light_ambient[] = { .15, .15, .15, 1.0 };
  GLfloat light_pos[] = { 0.0, 0.0, -0.0 };
  /* Enable lighting */
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos);

glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);

GLfloat blue[] = { 0.0, 0.0, 1.0, 1.0 };     /* blue */
// glColor3f (0.0, 0.0, 1.0); //
  glColor4fv(blue);
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, blue);

glTranslatef (2.0, 0.0, 0.0);
glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
glutSolidSphere(0.2, 10, 8); /* draw smaller planet */ //

glDisable(GL_LIGHT0);
glDisable(GL_LIGHTING);
```

```
glPopMatrix();
glutSwapBuffers();

}
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, 2000.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
switch (key) {
case 'd':
day = (day + 10) % 360;
glutPostRedisplay();
break;
case 'D':
day = (day - 10) % 360;
glutPostRedisplay();
break;
case 'y':
year = (year + 5) % 360;
glutPostRedisplay();
break;
case 'Y':
year = (year - 5) % 360;
glutPostRedisplay();
break;
default:
break;
}
}



void arrow_keys(int a_keys, int x, int y)
{

  switch ( a_keys ) {
    case GLUT_KEY_UP : // When Up Arrow Is Pressed...
        glMatrixMode(GL_MODELVIEW);
         glLoadIdentity();

                    //                    glTranslatef(0.0f,0.0f,1.0f);
        z=z-1.0f;
        gluLookAt (0.0, 0.0, z+5.0, 0.0, 0.0, z+0.0, 0.0, 1.0, 0.0);

        glutPostRedisplay();
                    break;
    case GLUT_KEY_DOWN : // When Up Arrow Is Pressed...
        glMatrixMode(GL_MODELVIEW);
         glLoadIdentity();

                    //                glTranslatef(0.0f,0.0f,-1.0f);
        z=z+1.0f;
        gluLookAt (0.0, 0.0, z+5.0, 0.0, 0.0, z+0.0, 0.0, 1.0, 0.0);

        glutPostRedisplay();
                    break;

    default:
      break;
  }
}


int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
  glutSpecialFunc(arrow_keys);

glutMainLoop();
return 0;
}
```

**Örnek 6 :** Güneş Sistemi Oluşturulması : Aydınlatmalı, Kamera Hareketli, Smooth, Desen Kaplamalı ve Quadric'li.



```
#include <GL/glut.h>

#include "bitmap.h" ///


#define MAX_NO_TEXTURES 2

static int year = 0, day = 0;

float z=0.0;

/* Place for tex_ids */
GLuint        texture_id[MAX_NO_TEXTURES]; ///


///
/* ///////////////////////////////////////////////////////////////////////
  'load_texture' takes the file name, size and OpenGL texture format of an
  image.  This is used inside the function to load the image and set the
  mipmap level, texture parameters and environment settings.
  As it does this inside the function, it doesn't return anything.


  glGenTextures (1, texture_id);
  glBindTexture(GL_TEXTURE_2D, texture_id[0]);
  load_texture("earth_r.raw", 640, 320, 3, GL_RGB, GL_NEAREST);

  Notice how you don't need to repeat all those texture parameters?
*/ ///////////////////////////////////////////////////////////////////////
void load_texture ( char *file_name, int width, int height, int depth,
                    GLenum colour_type, GLenum filter_type )
{
  GLubyte *raw_bitmap ;
  FILE *file;

  if (( file = fopen(file_name, "rb"))==NULL )
  {
    printf ( "File Not Found : %s\n", file_name );
    exit   ( 1 );
  }

  raw_bitmap = (GLubyte *)malloc(width * height * depth * (sizeof(GLubyte)));

  if ( raw_bitmap == NULL )
  {
    printf ( "Cannot allocate memory for texture\n" );
    fclose ( file );
    exit   ( 1 );
  }

  fread  ( raw_bitmap , width * height * depth, 1 , file );
  fclose ( file);

  //  Set Filtering type
  glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
filter_type );
  glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
filter_type );

  //  Set Texture Evironment
  glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE );

  //  Build Mipmaps
  gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width, height, colour_type,
                GL_UNSIGNED_BYTE, raw_bitmap );

  //  Free up the array
  free ( raw_bitmap );
}


void init(void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_FLAT);
//         glEnable (GL_CULL_FACE);
  glCullFace(GL_FRONT);
         glEnable (GL_DEPTH_TEST); //


  glEnable ( GL_TEXTURE_2D );
  glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
  glGenTextures ( 2, texture_id );

  glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
  load_texture  ( "earth_r.raw", 640, 320, 3, GL_RGB, GL_NEAREST );

//   glBindTexture ( GL_TEXTURE_2D, texture_id[1]);
//   load_texture  ( "cl_lum.raw", 640, 320, 1, GL_LUMINANCE, GL_NEAREST );

}
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
glPushMatrix();


glColor3f (1.0, 1.0, 0.0); //
glutSolidSphere(1.0, 20, 16); /* draw sun */ //

  GLfloat light_diffuse[] = { 0.9, 0.9, 0.9, 1.0 }; ///
  GLfloat light_ambient[] = { .35, .35, .35, 1.0 }; ///
  GLfloat light_pos[] = { 0.0, 0.0, -0.0 }; ///
  /* Enable lighting */
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos);


glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);

GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };     /* blue */ ///
// glColor3f (0.0, 0.0, 1.0); //
  glColor4fv(white);
  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);

glTranslatef (2.0, 0.0, 0.0);
glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
// glutSolidSphere(0.2, 10, 8); /* draw smaller planet */ //

///
  glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
  GLUquadricObj*  Earth = gluNewQuadric ( );
  gluQuadricDrawStyle ( Earth, GLU_FILL   );
  gluQuadricNormals   ( Earth, GLU_SMOOTH );
  gluQuadricTexture   ( Earth, GL_TRUE    );
  gluSphere ( Earth, 1.0, 20, 20 );
  gluDeleteQuadric ( Earth );
///

//GLUquadricObj * Earth = 0;
//Earth =  gluNewQuadric();
//gluSphere(Earth, 1.0f, 15, 15);

glDisable(GL_LIGHT0);
glDisable(GL_LIGHTING);

glPopMatrix();
glutSwapBuffers();

}
```

```
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, 2000.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
switch (key) {
case 'd':
day = (day + 10) % 360;
glutPostRedisplay();
break;
case 'D':
day = (day - 10) % 360;
glutPostRedisplay();
break;
case 'y':
year = (year + 5) % 360;
glutPostRedisplay();
break;
case 'Y':
year = (year - 5) % 360;
glutPostRedisplay();
break;
case 's':
glShadeModel (GL_SMOOTH);
glutPostRedisplay();
break;
case 'S':
glShadeModel (GL_FLAT);
glutPostRedisplay();
break;

default:
break;
}
}



void arrow_keys(int a_keys, int x, int y)
{

  switch ( a_keys ) {
    case GLUT_KEY_UP : // When Up Arrow Is Pressed...
```

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

                    //                  glTranslatef(0.0f,0.0f,1.0f);
        z=z-1.0f;
        gluLookAt (0.0, 0.0, z+5.0, 0.0, 0.0, z+0.0, 0.0, 1.0, 0.0);

        glutPostRedisplay();
                    break;
    case GLUT_KEY_DOWN : // When Up Arrow Is Pressed...
        glMatrixMode(GL_MODELVIEW);
         glLoadIdentity();

                    //                  glTranslatef(0.0f,0.0f,-1.0f);
        z=z+1.0f;
        gluLookAt (0.0, 0.0, z+5.0, 0.0, 0.0, z+0.0, 0.0, 1.0, 0.0);

        glutPostRedisplay();
                    break;

    default:
      break;
  }
}


int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
  glutSpecialFunc(arrow_keys);

glutMainLoop();
return 0;
}
```