Kingdom of Saudi Arabia
Ministry of Higher Education
College of computer
Department of Computer Science

المملكة العربية السعودية
وزارة التعليم العالي
كلية الحاسب
قسم علوم الحاسب

Computer Graphics

Solar System Project

Dr.Adel Mohammed

Main Function:

```
#define _CRT_SECURE_NO_DEPRECATE
#include <GL/freeglut.h>
#include <math.h>
#include <stdio.h>

//Set the width and height of the Window
#define Windowwidth  1024
#define Windowheight 1024
// Here the Distance can be treated as a unit. For example, the distance between
// mercurio and sun is 1*Distance, and between earth and sun is 3*Distance
#define Distance 100000000
#define Eradius 16000000
const GLfloat Pi = 3.1415926536f;
static int day = 0;
GLdouble Angle = 80.0;
GLdouble aix_x = 0.0, aix_y = 2000000000, aix_z = 2000000000;



//The ratio of radius0.4:1:1:0.5:11:9:4:3
//mercurios,venus,earth,mars,jupiter,saturn,uranus,neptune
GLuint tbg,tsun,tearth,tmercu,tven,tmars,tjup,tsat,tura,tnep;

//Loading texture files, which can also be found at
http://www.cppblog.com/doing5552/archive/2009/01/08/71532.aspx
int power_of_two(int n)
{
   if (n <= 0)
      return 0;
   return (n & (n - 1)) == 0;
}
GLuint LoadTexture(const char* filename)
{

   GLint width, height, total_bytes;
   GLubyte* pixels = 0;
   GLuint  texture_ID = 0;


   FILE* pFile = fopen(filename, "rb");
   if (pFile == 0)
      return 0;

   fseek(pFile, 0x0012, SEEK_SET);
   fread(&width, 4, 1, pFile);
   fread(&height, 4, 1, pFile);
   fseek(pFile, 54, SEEK_SET);


   {
      GLint line_bytes = width * 3;
      while (line_bytes % 4 != 0)
         ++line_bytes;
      total_bytes = line_bytes * height;
   }
```

```
pixels = (GLubyte*)malloc(total_bytes);
if (pixels == 0)
{
   fclose(pFile);
   return 0;
}


if (fread(pixels, total_bytes, 1, pFile) <= 0)
{
   free(pixels);
   fclose(pFile);
   return 0;
}


// The is for the compatibility with old version of opengl
{
   GLint max;
   glGetIntegerv(GL_MAX_TEXTURE_SIZE, &max);
   if (!power_of_two(width)
      || !power_of_two(height)
      || width > max
      || height > max)
   {
      const GLint new_width = 512;
      const GLint new_height = 512; // set the picture to 512*512
      GLint new_line_bytes, new_total_bytes;
      GLubyte* new_pixels = 0;


      new_line_bytes = new_width * 3;
      while (new_line_bytes % 4 != 0)
         ++new_line_bytes;
      new_total_bytes = new_line_bytes * new_height;


      new_pixels = (GLubyte*)malloc(new_total_bytes);
      if (new_pixels == 0)
      {
         free(pixels);
         fclose(pFile);
         return 0;
      }


      gluScaleImage(GL_RGB,
         width, height, GL_UNSIGNED_BYTE, pixels,
         new_width, new_height, GL_UNSIGNED_BYTE, new_pixels);


      free(pixels);
      pixels = new_pixels;
      width = new_width;
      height = new_height;
   }
}
```

```
    glGenTextures(1, &texture_ID);
    if (texture_ID == 0)
    {
        free(pixels);
        fclose(pFile);
        return 0;
    }

    glBindTexture(GL_TEXTURE_2D, texture_ID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
        GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);
    free(pixels);
    return texture_ID;

}

//Initilize to load files
void init_LoadallTexture()
{
    tsun = LoadTexture("pictures/sol.bmp");
    tbg = LoadTexture("pictures/bg.bmp");
    tearth = LoadTexture("pictures/terra.bmp");
    tmercu = LoadTexture("pictures/mercurio.bmp");
    tven = LoadTexture("pictures/venus.bmp");
    tmars = LoadTexture("pictures/marte.bmp");
    tjup = LoadTexture("pictures/jupiter.bmp");
    tsat = LoadTexture("pictures/saturno.bmp");
    tura = LoadTexture("pictures/urano.bmp");
    tnep = LoadTexture("pictures/neptuno.bmp");
}


void get_bg()
{

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Set up the orthographic projection
    glPushMatrix();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();



    glPopMatrix();
}
void drawOrbit(double radius, int numVertices = 100)
{
    glLineWidth(5.0); // Set line width for thicker orbits
    glColor3f(1.0f, 1.0f, 1.0f); // Set color for orbits
```

```
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < numVertices; ++i)
    {
        double angle = i * 2 * Pi / numVertices;
        double x = radius * cos(angle);
        double y = radius * sin(angle);
        glVertex3d(x, y, 0.0);
    }
    glEnd();
}



void get_sun()
{
    glPushMatrix();
    glRotatef(day / 25.0 * 360, 0.0, 0.0, -1.0);
    {
        GLfloat sun_light_position[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat sun_light_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat sun_light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
        GLfloat sun_light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

        glLightfv(GL_LIGHT0, GL_POSITION, sun_light_position);
        glLightfv(GL_LIGHT0, GL_AMBIENT, sun_light_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, sun_light_diffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, sun_light_specular);

        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHTING);

    }
    {
        GLfloat sun_mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat sun_mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat sun_mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat sun_mat_emission[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat sun_mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, sun_mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, sun_mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, sun_mat_specular);
        glMaterialfv(GL_FRONT, GL_EMISSION, sun_mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, sun_mat_shininess);
    }


    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);
    glBindTexture(GL_TEXTURE_2D, tsun);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 69600000, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();
}
void get_mercu()
```

```
{

   glPushMatrix();
   glTranslatef(2, 0.0f, 0.0f);
   glRotatef(2, 0.0f, 0.0f, -1.0f);
   glPopMatrix();
   glPushMatrix();
   glRotatef(day / 87.7 * 360, 0.0f, 0.0f, -1.0f);
   glTranslatef( Distance, 0.0f, 0.0f);
   {
      GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
      GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
      GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
      GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
      GLfloat mat_shininess = 0.0f;

      glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
      glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
      glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
      glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

   }
   GLUquadricObj* sphere = NULL;
   sphere = gluNewQuadric();
   glEnable(GL_TEXTURE_2D);
   gluQuadricDrawStyle(sphere, GLU_FILL);

   glBindTexture(GL_TEXTURE_2D, tmercu);
   gluQuadricTexture(sphere, GL_TRUE);
   gluQuadricNormals(sphere, GLU_SMOOTH);
   gluSphere(sphere, 0.4*Eradius, 100, 100);
   glDisable(GL_TEXTURE_2D);
   glPopMatrix();

}
void get_ven()
{

   glPushMatrix();
   glTranslatef(2, 0.0f, 0.0f);
   glRotatef(1.5, 0.0f, 0.0f, -1.0f);
   glPopMatrix();
   glPushMatrix();
   glRotatef(day / 224.7 * 360, 0.0f, 0.0f, -1.0f);
   glTranslatef(2 * Distance, 0.0f, 0.0f);
   {
      GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
      GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
      GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
      GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
      GLfloat mat_shininess = 0.0f;

      glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
      glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
      glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
      glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

   }
```

```
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tven);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere,  Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();

}
void get_earth()
{

    glPushMatrix();
    glRotatef(day / 360.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(3 * Distance, 0.0f, 0.0f);

    {
       GLfloat earth_mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
       GLfloat earth_mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
       GLfloat earth_mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
       GLfloat earth_mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
       GLfloat earth_mat_shininess = 0.0f;

       glMaterialfv(GL_FRONT, GL_AMBIENT, earth_mat_ambient);
       glMaterialfv(GL_FRONT, GL_DIFFUSE, earth_mat_diffuse);
       glMaterialfv(GL_FRONT, GL_SPECULAR, earth_mat_specular);
       glMaterialfv(GL_FRONT, GL_EMISSION, earth_mat_emission);
       glMaterialf(GL_FRONT, GL_SHININESS, earth_mat_shininess);
       /glutSolidSphere(15945000, 20, 20);/

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tearth);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 15945000, 100, 100);
    glDisable(GL_TEXTURE_2D);
    //glDisable(GL_LIGHT0);
    //glDisable(GL_LIGHTING);
    glPopMatrix();
}
void get_mars()
{

    glPushMatrix();
    glTranslatef(2, 0.0f, 0.0f);
    glRotatef(1.5, 0.0f, 0.0f, -1.0f);
    glPopMatrix();
    glPushMatrix();
    glRotatef(day / 687.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(4 * Distance, 0.0f, 0.0f);
    {
```

```
        GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tmars);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 0.5*Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();

}
void get_jup()
{
    glPushMatrix();
    glTranslatef(2, 0.0f, 0.0f);
    glRotatef(180, 0.0f, 0.0f, -1.0f);
    glPopMatrix();
    glPushMatrix();
    glRotatef(day / 4337.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(6.5 * Distance, 0.0f, 0.0f);

    {
        GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tjup);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 11*Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
```

```
    glPopMatrix();

}
void get_sat()
{
    glPushMatrix();
    glTranslatef(2, 0.0f, 0.0f);
    glRotatef(180, 0.0f, 0.0f, -1.0f);
    glPopMatrix();
    glPushMatrix();
    glRotatef(day / 10000.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(10.5 * Distance, 0.0f, 0.0f);

    {
        GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tsat);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 9*Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();

}
void get_ura()
{
    glPushMatrix();
    glTranslatef(2, 0.0f, 0.0f);
    glRotatef(180, 0.0f, 0.0f, -1.0f);
    glPopMatrix();
    glPushMatrix();
    glRotatef(day / 20000.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(13 * Distance, 0.0f, 0.0f);

    {
        GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
```

```
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tura);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 4*Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();

}
void get_nep()
{
    glPushMatrix();
    glTranslatef(2, 0.0f, 0.0f);
    glRotatef(120, 0.0f, 0.0f, -1.0f);
    glPopMatrix();
    glPushMatrix();
    glRotatef(day / 164.0 * 360, 0.0f, 0.0f, -1.0f);
    glTranslatef(15 * Distance, 0.0f, 0.0f);

    {
        GLfloat mat_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        GLfloat mat_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_emission[] = { 0.0f, 0.0f, 0.0f, 1.0f };
        GLfloat mat_shininess = 0.0f;

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

    }
    GLUquadricObj* sphere = NULL;
    sphere = gluNewQuadric();
    glEnable(GL_TEXTURE_2D);
    gluQuadricDrawStyle(sphere, GLU_FILL);

    glBindTexture(GL_TEXTURE_2D, tnep);
    gluQuadricTexture(sphere, GL_TRUE);
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluSphere(sphere, 3*Eradius, 100, 100);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();

}

//Display Function
void myDisplay(void)

{
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    get_bg();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(Angle, 1, 1, 10000000000);
      glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(aix_x, aix_y, aix_z, 0, 0, 0, 0, 0, 1);

    // Draw orbits
    glColor3f(0.5f, 0.5f, 0.5f); // Set color for orbits
    drawOrbit(Distance);
    drawOrbit(2 * Distance);
    drawOrbit(3 * Distance);
    drawOrbit(4 * Distance);
    drawOrbit(6.5 * Distance);
    drawOrbit(10.5 * Distance);
    drawOrbit(13 * Distance);
    drawOrbit(15 * Distance);

    get_sun();
    get_mercu();
    get_ven();
    get_earth();
    get_mars();
    get_jup();
    get_sat();
    get_ura();
    get_nep();

    glFlush();
    glutSwapBuffers();

}

//Here the timing function is used to control the speed of frame rate
void timerProc(int id)
{
    ++day;
    glutPostRedisplay();
    glutTimerFunc(50, timerProc, 1);//The first parameter is depend on your own device
}

//Control the view position and angle
void mykeyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'W':
    case 'w':
        aix_y += 100000000;
        aix_z -= 100000000;
        break;
    case 'S':
    case 's':
        aix_y -= 100000000;
        aix_z += 100000000;
        break;
    case 'A':
    case 'a':
```

```
        aix_x -= 100000000;
        aix_z += 100000000;
        break;
    case 'D':
    case 'd':
        aix_x += 100000000;
        aix_z -= 100000000;
        break;


    case 'J':
    case 'j':
        Angle -= 5.0;
        break;
    case 'L':
    case 'l':
        Angle += 5.0;
        break;
    }
    glutPostRedisplay();

}
int main(int argc, char* argv[])

{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(Windowwidth, Windowheight);

    glutCreateWindow("Solar System");
    init_LoadallTexture();
    glutDisplayFunc(&myDisplay);
    glutKeyboardFunc(mykeyboard);
    glutTimerFunc(50, timerProc, 1);
    glutMainLoop();

    return 0;


}
```

This code is an OpenGL program that simulates the solar system using textures and basic shapes to represent the planets and the sun. Here's a breakdown of the methods used and what they do:

Libraries Included:

- `<GL/freeglut.h>`: OpenGL Utility Toolkit for creating and managing windows.

- `<math.h>`: C math library for mathematical functions.

- `<stdio.h>`: Standard input-output library for basic input and output operations.

Constants and Variables:

- Definitions for window dimensions, distances, and planet radii.

- Various constants and variables used for simulation.

Texture Loading:

- `power_of_two(int n)`: Checks if a number is a power of two.

- `LoadTexture(const char* filename)`: Loads a texture from a file and sets OpenGL parameters.

Initialization:

- `init_LoadallTexture()`: Loads all the required textures for planets and the background.

Functions for Drawing Celestial Bodies:

- `get_bg()`: Draws the background using a loaded texture.

- `get_sun()`, `get_mercu()`, `get_ven()`, `get_earth()`, `get_mars()`, `get_jup()`, `get_sat()`, `get_ura()`, `get_nep()`: Functions for drawing each planet with its respective texture and position. They set the material properties and draw the planets as spheres.

Display Function:

- `myDisplay(void)`: Clears the screen, sets the projection and modelview matrices, sets the view position, and calls functions to draw the celestial bodies.

Timing and Redisplay:

- `timerProc(int id)`: Controls the timing of the animation by updating the days and triggering a display update.

- `glutPostRedisplay()`: Requests a display update.

- `glutTimerFunc()`: Sets a timer function to control frame rate.

Keyboard Control:

- `mykeyboard(unsigned char key, int x, int y)`: Handles keyboard input to control the view position and angle

Certainly! In the provided code, the `mykeyboard` function responds to specific key presses and triggers actions related to camera movement or object transformations. Here's a breakdown of the cases within the function:

- 'W' or 'w': This case handles moving the viewpoint or object upward or forward. Typically, this action involves incrementing the Y-coordinate or moving along the forward direction in the scene.
- 'S' or 's': These keys control the downward or backward movement of the viewpoint or object. It usually involves decrementing the Y-coordinate or moving opposite to the forward direction.
- 'A' or 'a': This case manages the movement to the left. It could involve decrementing the Xcoordinate or shifting towards the left side in the scene.
- 'D' or 'd': Similar to 'A'/'a' but for moving to the right. It typically involves incrementing the Xcoordinate or shifting towards the right side in the scene.

- 'J' or 'j': This case is associated with rotating or adjusting the angle to the left. It might change the viewing angle or rotation angle of the object to the left side in the scene.
- 'L' or 'l': Conversely, this case is for rotating or adjusting the angle to the right. It might modify the viewing angle or rotation angle of the object to the right side in the scene.

Each case handles specific user inputs and triggers corresponding actions related to movement or transformation within the scene. Depending on the context of the application, these actions could involve updating camera positions, changing object orientations, or altering the viewing perspective to navigate or interact with the rendered elements.

Main Function:

- Initializes GLUT, creates the window, sets display function, keyboard function, timer function, and enters the main loop.

This program essentially creates a visual representation of the solar system using OpenGL, textures, and basic shapes to mimic the planets' orbits and rotations around the sun.

Animation:

Rotations are implemented using OpenGL's transformation functions like `glRotatef()` and `glTranslatef()`. These functions manipulate the current matrix to perform rotations or translations, affecting how objects are drawn in the scene. Here's how rotations are implemented for the planets:

1. `glRotatef()` Function:
- This function is used to apply rotations to objects. It takes parameters specifying the angle of rotation and the axis around which the rotation occurs.

- For example, in functions like `get_earth()`, `get_mars()`, etc., the `glRotatef()` function is used to rotate the planets around their respective axes. The function modifies the current matrix, causing the subsequent drawing operations to reflect the applied rotation.

2. `glTranslatef()` Function:
- This function translates (moves) the coordinate system by a specified amount along the x, y, and z axes.

- In functions like `get_mercu()`, `get_ven()`, etc., `glTranslatef()` is used to position the planets at their appropriate distances from the sun. It moves the planets along their orbits around the sun.

3. Matrix Stacks and `glPushMatrix()`, `glPopMatrix()`:
- `glPushMatrix()` and `glPopMatrix()` are used to save and restore the current matrix state, allowing hierarchical transformations.

- Each planet's rendering function often begins with `glPushMatrix()` to save the current transformation state, applies necessary transformations (such as rotations and translations), draws the planet, and then uses `glPopMatrix()` to revert to the previous state. This helps isolate transformations to specific objects without affecting others.

By using combinations of these functions within the rendering functions for each celestial body, the code achieves the intended rotations and translations to simulate the orbits and rotations of the planets within the solar system.

Light and Shadow:

Lighting and shadows are implemented using OpenGL's lighting functionalities and material properties. Here's how they're achieved:

1. Lighting Setup:
- Lighting is enabled using `glEnable(GL_LIGHTING)` and specific lights are enabled (in this case, `GL_LIGHT0`).

- `glLightfv()` sets different lighting parameters for a specific light source (like position, ambient, diffuse, specular, etc.).

- For example, in the `get_sun()` function, lighting parameters like position, ambient, diffuse, and specular properties are set for the sun to simulate its illumination effects.

2. Material Properties:
- Material properties define how an object reacts to light. These properties are set using `glMaterialfv()` to specify ambient, diffuse, specular, emission, and shininess properties.

- Each planet's rendering function (`get_earth()`, `get_mars()`, etc.) sets its material properties to create different surface appearances and responses to light.

3. Shading and Shadows:
- The shading model used in this code seems to be Phong shading (`GLU_SMOOTH`), which interpolates normals across the surface to achieve smooth shading effects.

- The use of material properties such as `GL_SPECULAR` and `GL_SHININESS` contributes to creating highlights and shininess on the surfaces of the planets.

- However, the provided code doesn't explicitly implement shadow generation or casting. Generating shadows in OpenGL typically involves more advanced techniques like shadow mapping or shadow volumes, which are not present in this code snippet.

Overall, the lighting effects and material properties are set up to simulate illumination, surface appearance, and specular highlights on the celestial bodies within the solar system visualization. However, the implementation doesn't include complex shadow generation algorithms for casting shadows between objects.