Graphics Project

Name / Ahmed Abdallah Aboeleid

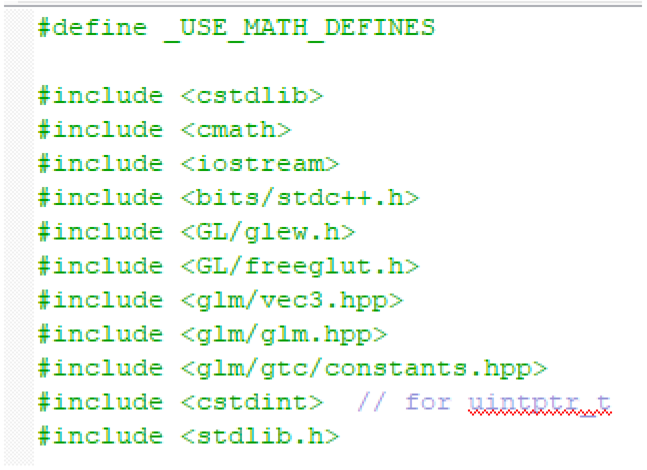
ID / 19015274

Problem Statement:

- You are required to implement an application that simulate the solar system
- you should enable user from controlling space-craft to explore the solar system.
- You are required to use two view ports: One for space-craft and the other for the whole solar system
- In simulation you need to handle:
  • Instantiation of Sun and 8 planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune)
  • Solar system animation (spinning and rotation of planets around Sun and Moon around Earth) • Space-craft movement.
  • Lighting and emission.
- You are free to choose the proper implementation of:
  • Planet sizes (make sure it is sensible)
  • Planet colors or (textures) (make sure it is sensible)
  • Mouse and keyboard interaction (make sure it gives good user experience)

Code:

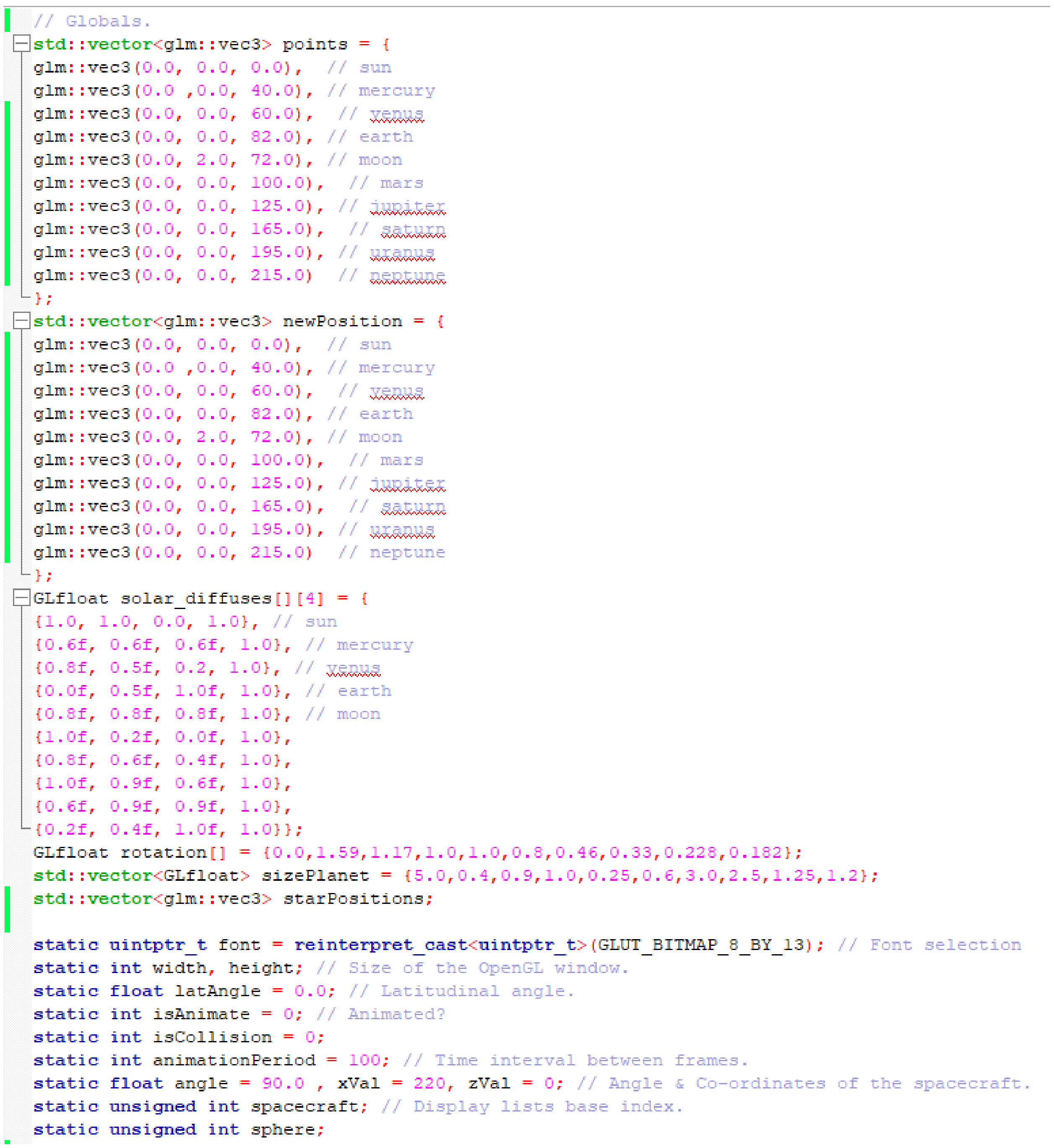
⇨ Include all library I will use



```
#define _USE_MATH_DEFINES

#include <cstdlib>
#include <cmath>
#include <iostream>
#include <bits/stdc++.h>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <glm/vec3.hpp>
#include <glm/glm.hpp>
#include <glm/gtc/constants.hpp>
#include <cstdint>  // for uintptr_t
#include <stdlib.h>
```

⇨ Define global variables:

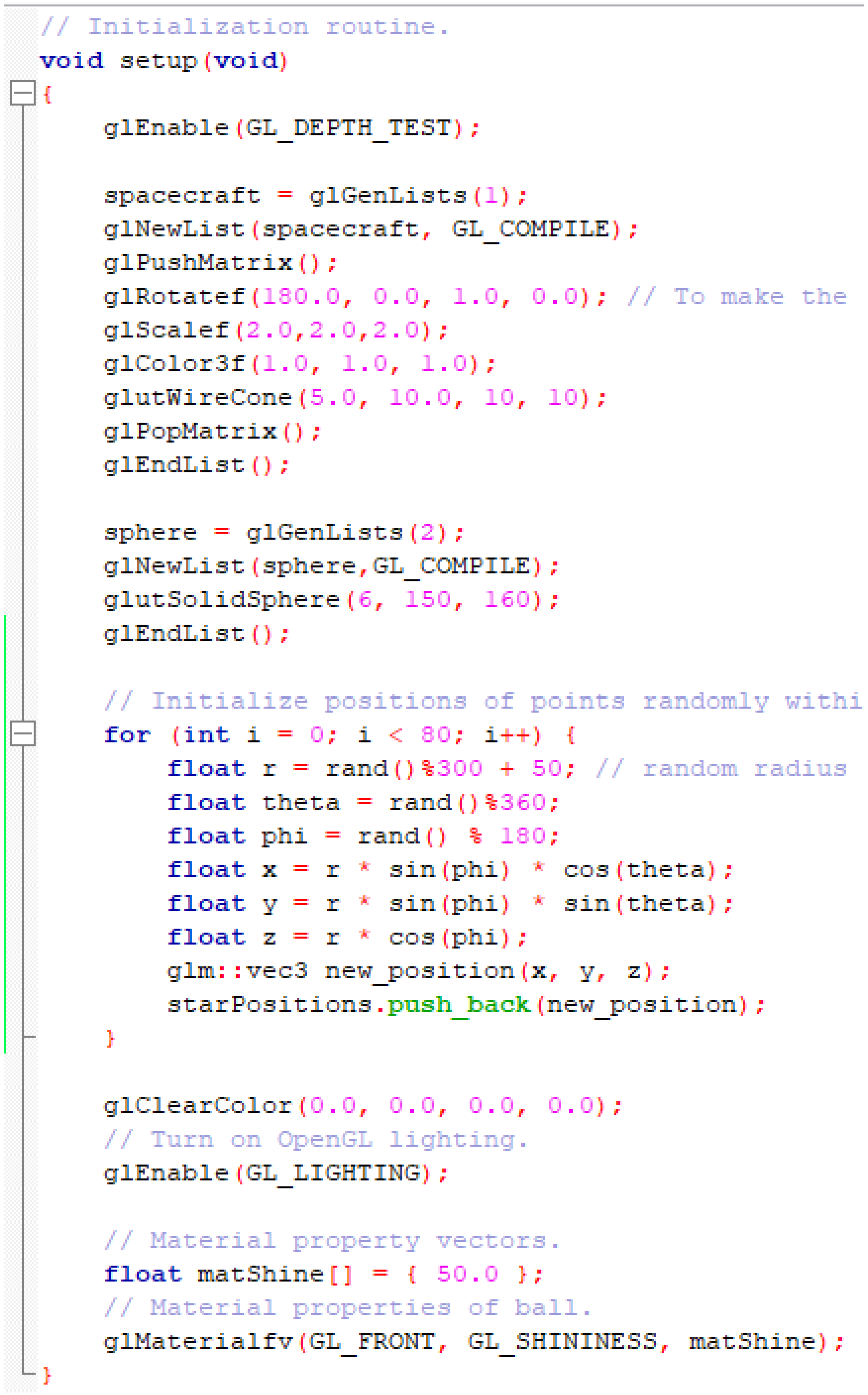
- Vector points represent values of translation of each planet in solar system
- Vector new-positions represent values of position of each planet during it's rotation around sun
- 2d array diffuse represent diffuse of each planet which will use in lightening
- Array rotation represent value of attribute which multiplied by latangle to make rotations of planets around sun more realistic
- Vector sizeplanet represent value of scale which will use to scala each planet.
- After that begin to define global variable which wil use in following functions



```cpp
// Globals.
std::vector<glm::vec3> points = {
glm::vec3(0.0, 0.0, 0.0),  // sun
glm::vec3(0.0 ,0.0, 40.0), // mercury
glm::vec3(0.0, 0.0, 60.0),  // venus
glm::vec3(0.0, 0.0, 82.0), // earth
glm::vec3(0.0, 2.0, 72.0), // moon
glm::vec3(0.0, 0.0, 100.0),  // mars
glm::vec3(0.0, 0.0, 125.0), // jupiter
glm::vec3(0.0, 0.0, 165.0),  // saturn
glm::vec3(0.0, 0.0, 195.0), // uranus
glm::vec3(0.0, 0.0, 215.0)  // neptune
};
std::vector<glm::vec3> newPosition = {
glm::vec3(0.0, 0.0, 0.0),  // sun
glm::vec3(0.0 ,0.0, 40.0), // mercury
glm::vec3(0.0, 0.0, 60.0),  // venus
glm::vec3(0.0, 0.0, 82.0), // earth
glm::vec3(0.0, 2.0, 72.0), // moon
glm::vec3(0.0, 0.0, 100.0),  // mars
glm::vec3(0.0, 0.0, 125.0), // jupiter
glm::vec3(0.0, 0.0, 165.0),  // saturn
glm::vec3(0.0, 0.0, 195.0), // uranus
glm::vec3(0.0, 0.0, 215.0)  // neptune
};
GLfloat solar_diffuses[][4] = {
{1.0, 1.0, 0.0, 1.0}, // sun
{0.6f, 0.6f, 0.6f, 1.0}, // mercury
{0.8f, 0.5f, 0.2, 1.0}, // venus
{0.0f, 0.5f, 1.0f, 1.0}, // earth
{0.8f, 0.8f, 0.8f, 1.0}, // moon
{1.0f, 0.2f, 0.0f, 1.0},
{0.8f, 0.6f, 0.4f, 1.0},
{1.0f, 0.9f, 0.6f, 1.0},
{0.6f, 0.9f, 0.9f, 1.0},
{0.2f, 0.4f, 1.0f, 1.0}};
GLfloat rotation[] = {0.0,1.59,1.17,1.0,1.0,0.8,0.46,0.33,0.228,0.182};
std::vector<GLfloat> sizePlanet = {5.0,0.4,0.9,1.0,0.25,0.6,3.0,2.5,1.25,1.2};
std::vector<glm::vec3> starPositions;

static uintptr_t font = reinterpret_cast<uintptr_t>(GLUT_BITMAP_8_BY_13); // Font selection
static int width, height; // Size of the OpenGL window.
static float latAngle = 0.0; // Latitudinal angle.
static int isAnimate = 0; // Animated?
static int isCollision = 0;
static int animationPeriod = 100; // Time interval between frames.
static float angle = 90.0 , xVal = 220, zVal = 0; // Angle & Co-ordinates of the spacecraft.
static unsigned int spacecraft; // Display lists base index.
static unsigned int sphere;
```

⇨ Setup function:

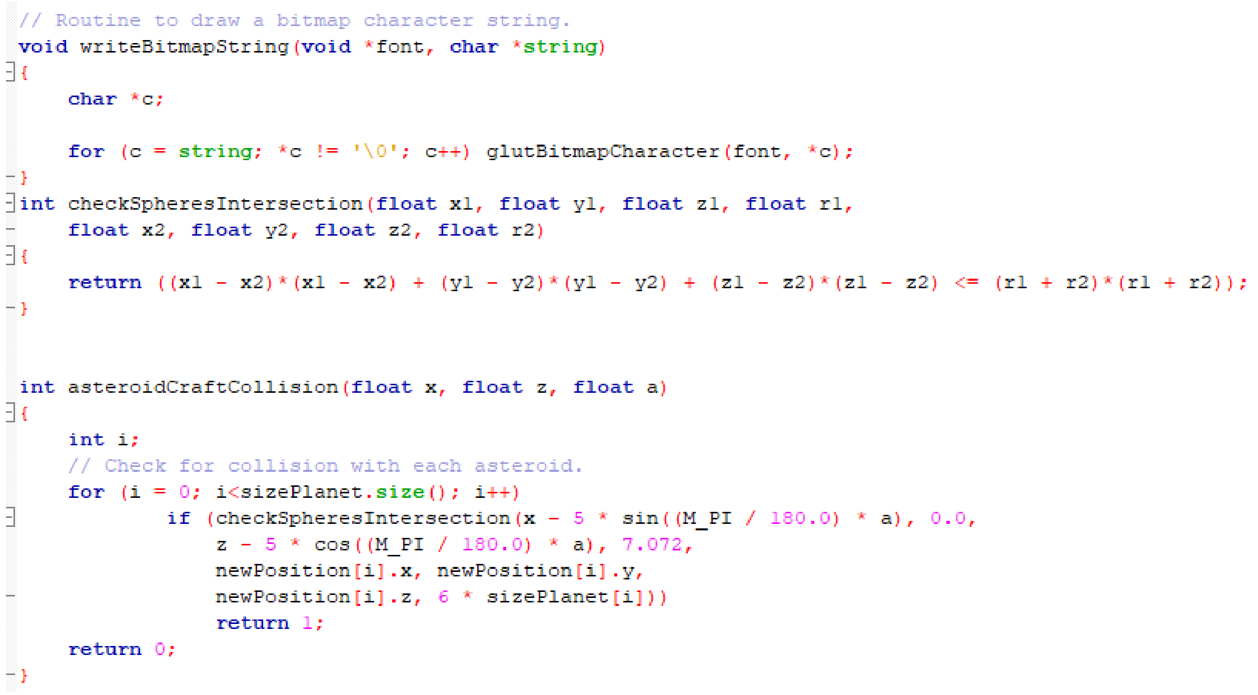
Create 2 lists which will use to create points at random positions which represent stars draw sphere “planet” and draw spacecraft , then setup lightening



```
// Initialization routine.
void setup(void)
{
    glEnable(GL_DEPTH_TEST);

    spacecraft = glGenLists(1);
    glNewList(spacecraft, GL_COMPILE);
    glPushMatrix();
    glRotatef(180.0, 0.0, 1.0, 0.0); // To make the
    glScalef(2.0,2.0,2.0);
    glColor3f(1.0, 1.0, 1.0);
    glutWireCone(5.0, 10.0, 10, 10);
    glPopMatrix();
    glEndList();

    sphere = glGenLists(2);
    glNewList(sphere,GL_COMPILE);
    glutSolidSphere(6, 150, 160);
    glEndList();

    // Initialize positions of points randomly withi
    for (int i = 0; i < 80; i++) {
        float r = rand()%300 + 50; // random radius
        float theta = rand()%360;
        float phi = rand() % 180;
        float x = r * sin(phi) * cos(theta);
        float y = r * sin(phi) * sin(theta);
        float z = r * cos(phi);
        glm::vec3 new_position(x, y, z);
        starPositions.push_back(new_position);
    }

    glClearColor(0.0, 0.0, 0.0, 0.0);
    // Turn on OpenGL lighting.
    glEnable(GL_LIGHTING);

    // Material property vectors.
    float matShine[] = { 50.0 };
    // Material properties of ball.
    glMaterialfv(GL_FRONT, GL_SHININESS, matShine);
}
```

➩ Handle collision:

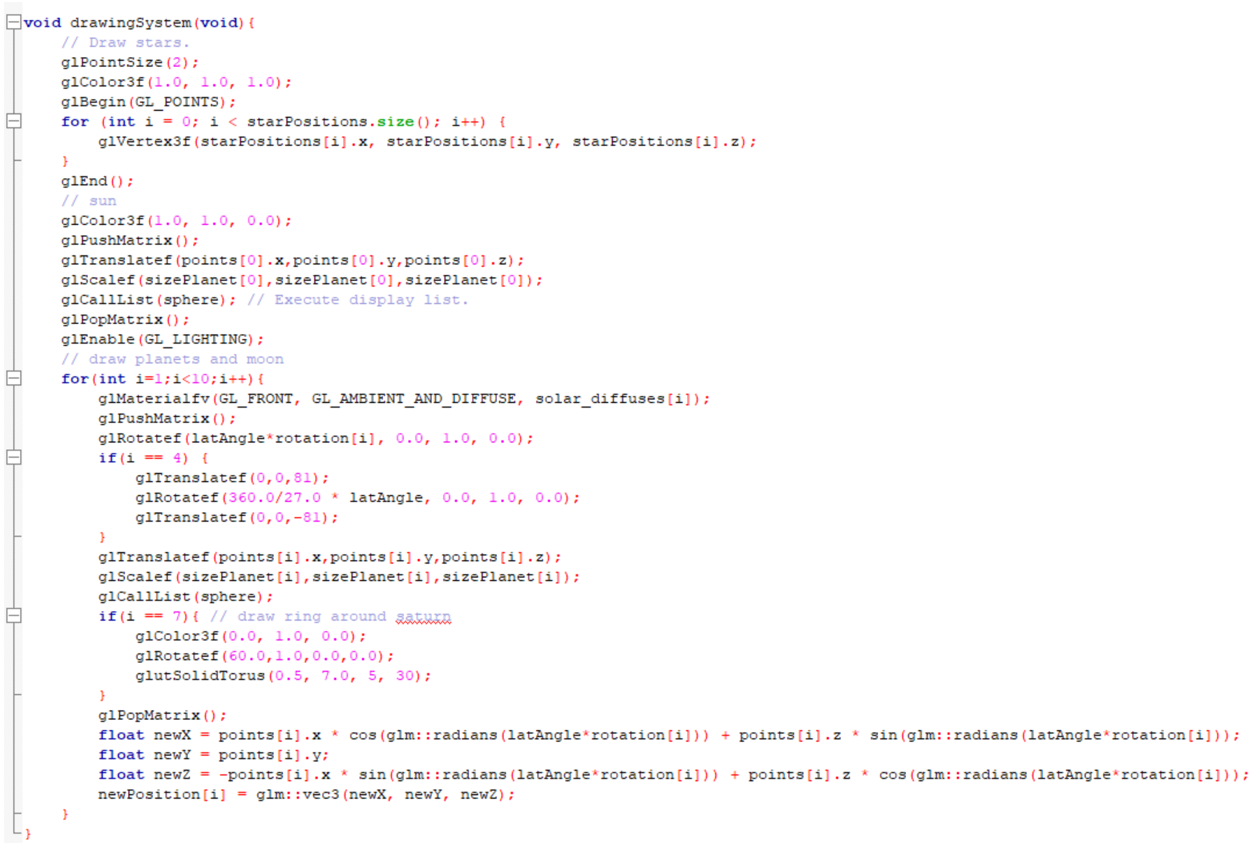
- Function “writeBitmapString” used to print message when collision occur
- Function “checkSpheresIntersection” used to calculate distance between spacecraft and planet
- Function “asteroidCraftCollision” used to check if there is collision between spacecraft and planet by passing position of each planet and spacecraft to function “checkSpheresIntersection”



```
// Routine to draw a bitmap character string.
void writeBitmapString(void *font, char *string)
{
    char *c;

    for (c = string; *c != '\0'; c++) glutBitmapCharacter(font, *c);
}
int checkSpheresIntersection(float x1, float y1, float z1, float r1,
    float x2, float y2, float z2, float r2)
{
    return ((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2) + (z1 - z2)*(z1 - z2) <= (r1 + r2)*(r1 + r2));
}


int asteroidCraftCollision(float x, float z, float a)
{
    int i;
    // Check for collision with each asteroid.
    for (i = 0; i<sizePlanet.size(); i++)
            if (checkSpheresIntersection(x - 5 * sin((M_PI / 180.0) * a), 0.0,
                z - 5 * cos((M_PI / 180.0) * a), 7.072,
                newPosition[i].x, newPosition[i].y,
                newPosition[i].z, 6 * sizePlanet[i]))
                return 1;
    return 0;
}
```

⇨ Drawing planet:

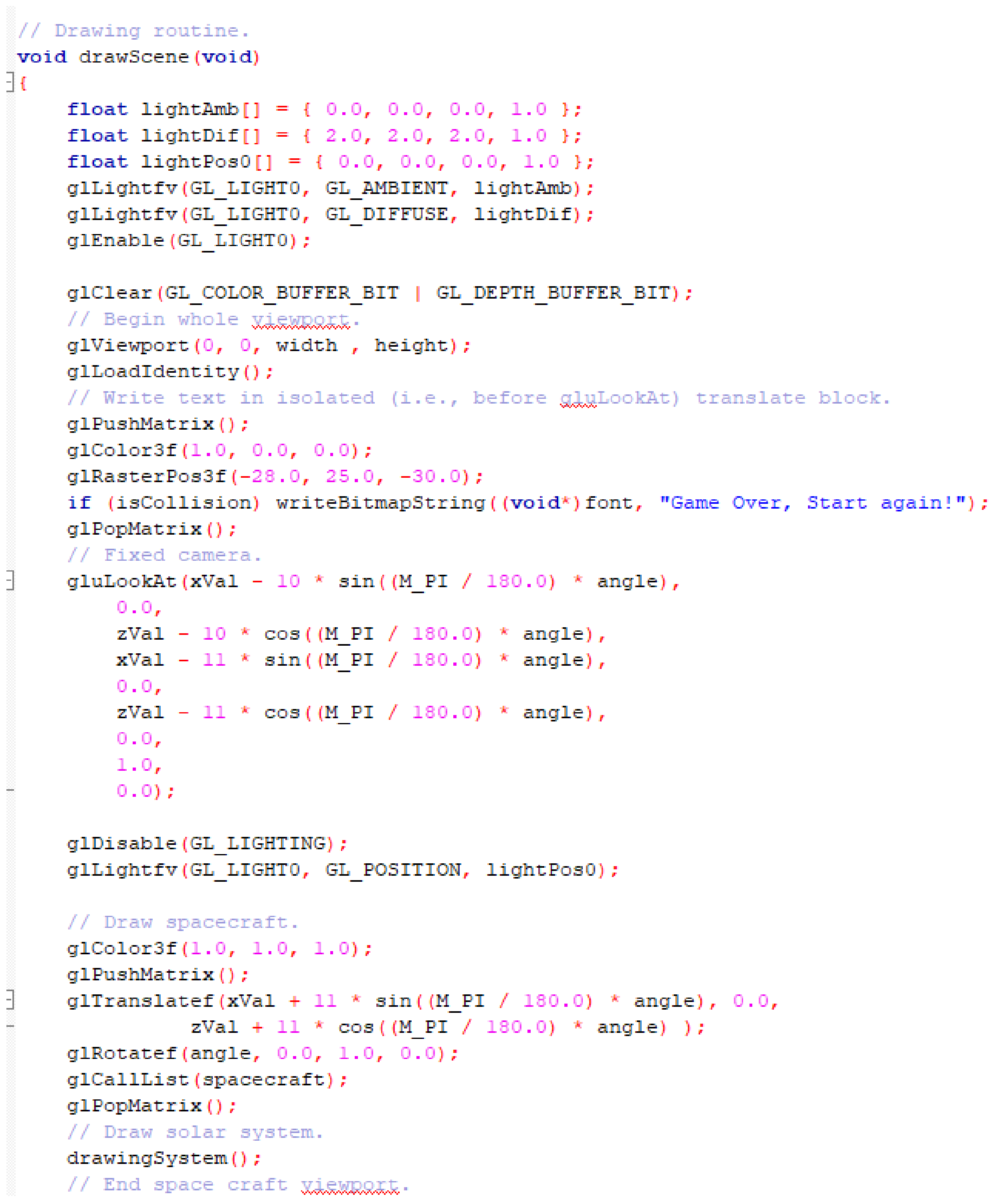
- Draw sun in first then enable lighting and begin draw planet by calling list of sphere and handle special cases of drawing moon and torus around Saturn
- Before drawing each planet , change material of light to be suitable for planet.



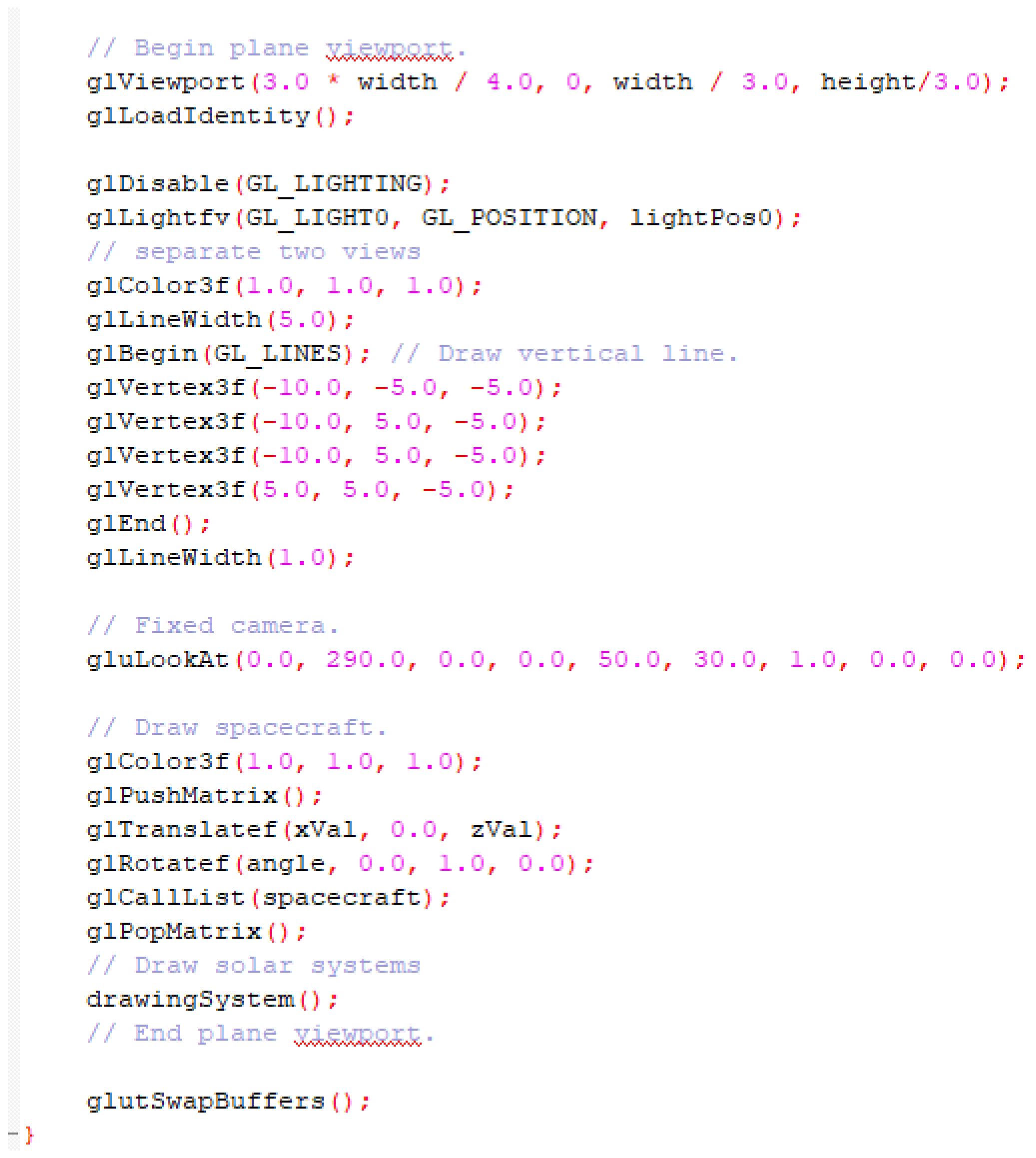
```
void drawingSystem(void){
    // Draw stars.
    glPointSize(2);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POINTS);
    for (int i = 0; i < starPositions.size(); i++) {
        glVertex3f(starPositions[i].x, starPositions[i].y, starPositions[i].z);
    }
    glEnd();
    // sun
    glColor3f(1.0, 1.0, 0.0);
    glPushMatrix();
    glTranslatef(points[0].x,points[0].y,points[0].z);
    glScalef(sizePlanet[0],sizePlanet[0],sizePlanet[0]);
    glCallList(sphere); // Execute display list.
    glPopMatrix();
    glEnable(GL_LIGHTING);
    // draw planets and moon
    for(int i=1;i<10;i++){
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, solar_diffuses[i]);
        glPushMatrix();
        glRotatef(latAngle*rotation[i], 0.0, 1.0, 0.0);
        if(i == 4) {
            glTranslatef(0,0,81);
            glRotatef(360.0/27.0 * latAngle, 0.0, 1.0, 0.0);
            glTranslatef(0,0,-81);
        }
        glTranslatef(points[i].x,points[i].y,points[i].z);
        glScalef(sizePlanet[i],sizePlanet[i],sizePlanet[i]);
        glCallList(sphere);
        if(i == 7){ // draw ring around saturn
            glColor3f(0.0, 1.0, 0.0);
            glRotatef(60.0,1.0,0.0,0.0);
            glutSolidTorus(0.5, 7.0, 5, 30);
        }
        glPopMatrix();
        float newX = points[i].x * cos(glm::radians(latAngle*rotation[i])) + points[i].z * sin(glm::radians(latAngle*rotation[i]));
        float newY = points[i].y;
        float newZ = -points[i].x * sin(glm::radians(latAngle*rotation[i])) + points[i].z * cos(glm::radians(latAngle*rotation[i]));
        newPosition[i] = glm::vec3(newX, newY, newZ);
    }
}
```

⇨ Draw Scene:

This code is a drawing routine for a game that features a spacecraft navigating through a solar system. The routine sets up the lighting and viewport for the game, draws the spacecraft, and draws the solar system. It also includes a collision detection feature where if there is a collision between the spacecraft and an object in the solar system, a "Game Over" message is displayed. The camera view is fixed and the spacecraft is rotated based on an angle variable. The routine uses OpenGL functions to draw lines, text, and shapes, and also uses trigonometry to calculate the position of the spacecraft in relation to the camera. Overall, this routine provides the visual elements for the game



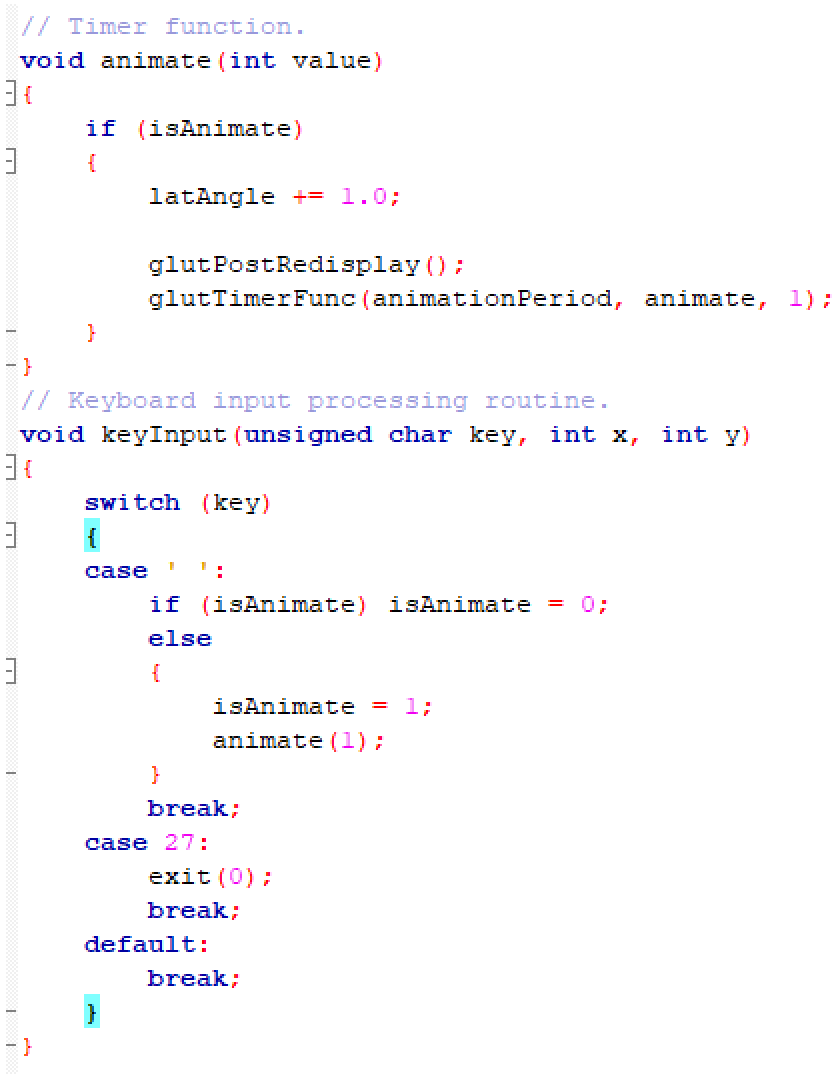
```
// Drawing routine.
void drawScene(void)
{
    float lightAmb[] = { 0.0, 0.0, 0.0, 1.0 };
    float lightDif[] = { 2.0, 2.0, 2.0, 1.0 };
    float lightPos0[] = { 0.0, 0.0, 0.0, 1.0 };
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDif);
    glEnable(GL_LIGHT0);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Begin whole viewport.
    glViewport(0, 0, width , height);
    glLoadIdentity();
    // Write text in isolated (i.e., before gluLookAt) translate block.
    glPushMatrix();
    glColor3f(1.0, 0.0, 0.0);
    glRasterPos3f(-28.0, 25.0, -30.0);
    if (isCollision) writeBitmapString((void*)font, "Game Over, Start again!");
    glPopMatrix();
    // Fixed camera.
    gluLookAt(xVal - 10 * sin((M_PI / 180.0) * angle),
        0.0,
        zVal - 10 * cos((M_PI / 180.0) * angle),
        xVal - 11 * sin((M_PI / 180.0) * angle),
        0.0,
        zVal - 11 * cos((M_PI / 180.0) * angle),
        0.0,
        1.0,
        0.0);

    glDisable(GL_LIGHTING);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);

    // Draw spacecraft.
    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();
    glTranslatef(xVal + 11 * sin((M_PI / 180.0) * angle), 0.0,
              zVal + 11 * cos((M_PI / 180.0) * angle) );
    glRotatef(angle, 0.0, 1.0, 0.0);
    glCallList(spacecraft);
    glPopMatrix();
    // Draw solar system.
    drawingSystem();
    // End space craft viewport.
```



```
    // Begin plane viewport.
    glViewport(3.0 * width / 4.0, 0, width / 3.0, height/3.0);
    glLoadIdentity();

    glDisable(GL_LIGHTING);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
    // separate two views
    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(5.0);
    glBegin(GL_LINES); // Draw vertical line.
    glVertex3f(-10.0, -5.0, -5.0);
    glVertex3f(-10.0, 5.0, -5.0);
    glVertex3f(-10.0, 5.0, -5.0);
    glVertex3f(5.0, 5.0, -5.0);
    glEnd();
    glLineWidth(1.0);

    // Fixed camera.
    gluLookAt(0.0, 290.0, 0.0, 0.0, 50.0, 30.0, 1.0, 0.0, 0.0);

    // Draw spacecraft.
    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();
    glTranslatef(xVal, 0.0, zVal);
    glRotatef(angle, 0.0, 1.0, 0.0);
    glCallList(spacecraft);
    glPopMatrix();
    // Draw solar systems
    drawingSystem();
    // End plane viewport.

    glutSwapBuffers();
}
```

⇨ Animate & handle input:

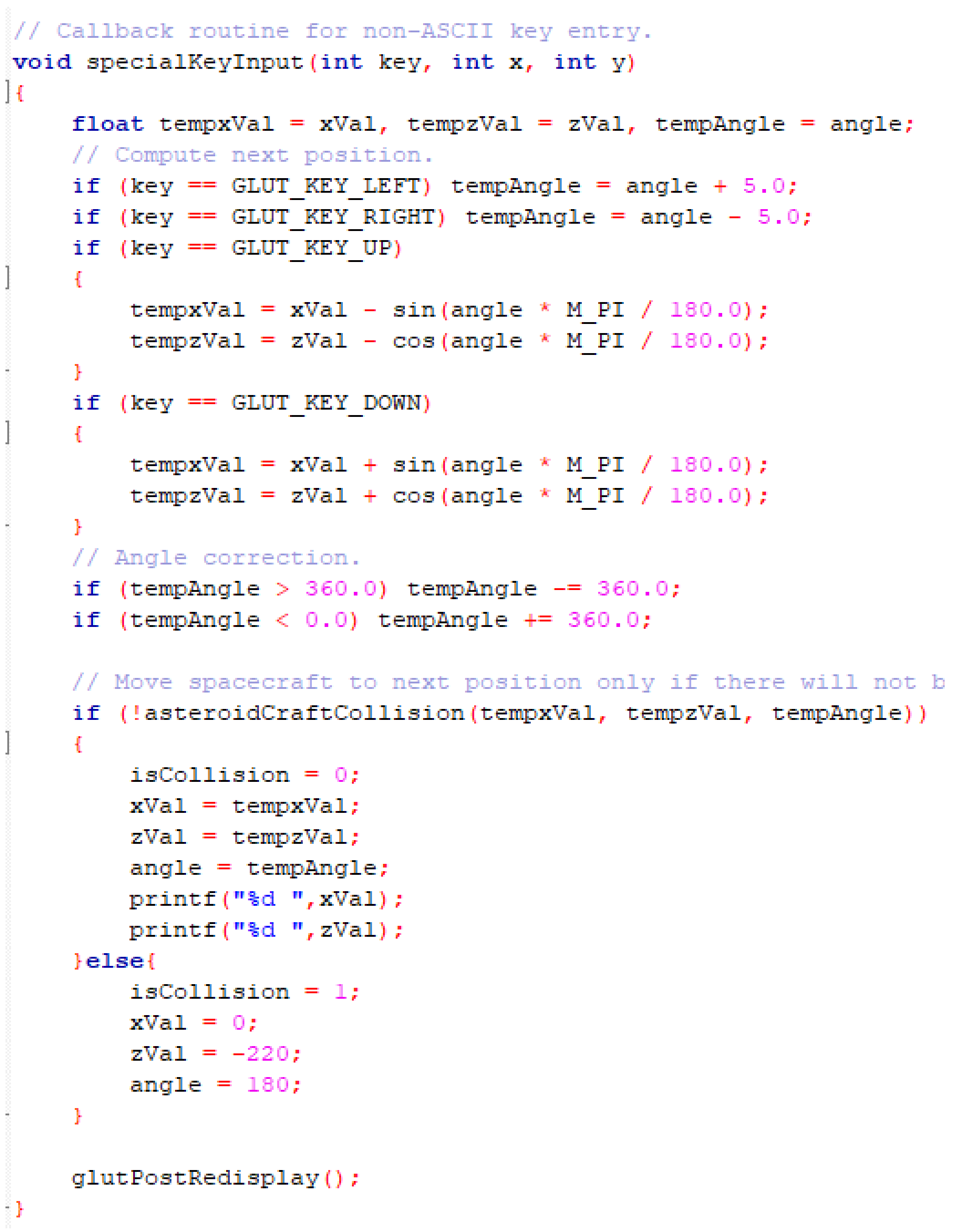
Call animate to change angle of rotation and draw scene again and call back every 100 ms. And handle input by pressing “space” to begin simulation.



```
// Timer function.
void animate(int value)
{
    if (isAnimate)
    {
        latAngle += 1.0;

        glutPostRedisplay();
        glutTimerFunc(animationPeriod, animate, 1);
    }
}
// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
    switch (key)
    {
    case ' ':
        if (isAnimate) isAnimate = 0;
        else
        {
            isAnimate = 1;
            animate(1);
        }
        break;
    case 27:
        exit(0);
        break;
    default:
        break;
    }
}
```

⇨ Handle moving of spacecraft:

Handle moving by arrows in keyboard and prevent spacecraft from moving if there is collision , it return spacecraft to initial position like it loss game and restart it.



```
// Callback routine for non-ASCII key entry.
void specialKeyInput(int key, int x, int y)
{
    float tempxVal = xVal, tempzVal = zVal, tempAngle = angle;
    // Compute next position.
    if (key == GLUT_KEY_LEFT) tempAngle = angle + 5.0;
    if (key == GLUT_KEY_RIGHT) tempAngle = angle - 5.0;
    if (key == GLUT_KEY_UP)
    {
        tempxVal = xVal - sin(angle * M_PI / 180.0);
        tempzVal = zVal - cos(angle * M_PI / 180.0);
    }
    if (key == GLUT_KEY_DOWN)
    {
        tempxVal = xVal + sin(angle * M_PI / 180.0);
        tempzVal = zVal + cos(angle * M_PI / 180.0);
    }
    // Angle correction.
    if (tempAngle > 360.0) tempAngle -= 360.0;
    if (tempAngle < 0.0) tempAngle += 360.0;

    // Move spacecraft to next position only if there will not b
    if (!asteroidCraftCollision(tempxVal, tempzVal, tempAngle))
    {
        isCollision = 0;
        xVal = tempxVal;
        zVal = tempzVal;
        angle = tempAngle;
        printf("%d ",xVal);
        printf("%d ",zVal);
    }else{
        isCollision = 1;
        xVal = 0;
        zVal = -220;
        angle = 180;
    }

    glutPostRedisplay();
}
```

⇨ Interaction with user:

```
// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
    std::cout << "Interaction:" << std::endl;
    std::cout << "Press space to begin rotation." << std::endl
        << "Press the left/right arrow keys to turn the craft." << std::endl
        << "Press the up/down arrow keys to move the craft." << std::endl;
}
```

Screenshots: