Course Project
(Solar System Simulation)

Name: Sally Kamel Soliman Armanyous

ID: 19015762

Problem Statement:

You are required to implement an application that simulate the solar system. Also, 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 (See figure 1 for more declaration). 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 flow:

- The code begins with some header includes and macro definitions.

```
#define ROWS 10  // Number of rows of asteroids.
// Globals.
static intptr_t font = (intptr_t)GLUT_BITMAP_8_BY_13; // Font selection.
static int width, height; // Size of the OpenGL window.
static float angle = 0.0; // Angle of the spacecraft.
static float xVal = 0, zVal = 0; // Co-ordinates of the spacecraft.
static int isCollision = 0; // Is there collision between the spacecraft and an asteroid?
static unsigned int spacecraft; // Display lists base index.
static int frameCount = 0; // Number of frames
static float Xangle = 0.0, Yangle = 0.0, Zangle = 0.0; // Angles to rotate scene.
static int isAnimate = 0; // Animated?
static int animationPeriod = 100; // Time interval between frames.
//spinning angle of each asteroid
static float sunSpin=0;
static float mercurySpin=0;
static float venusSpin=0;
static float earthSpin=0;
static float marsSpin=0;
static float jupiterSpin=0;
static float saturnSpin=0;
static float uranusSpin=0;
static float neptuneSpin=0;
static float moonSpin=0;
```

```
// rotaion angle of each asteroid aound the sun
static float mercuryRot=0;
static float venusRot=0;
static float earthRot=0;
static float marsRot=0;
static float jupiterRot=0;
static float saturnRot=0;
static float uranusRot=0;
static float neptuneRot=0;
static float moonRot=0;

GLfloat centerX = 0.0f;  // X-coordinate of the center point
GLfloat centerY = 0.0f;  // Y-coordinate of the center point
GLfloat centerZ = -40.0f; //z-coordinate of the center point
// Routine to draw a bitmap character string.
```

- The Asteroid class is defined, representing individual asteroids in the solar system. It has properties like position, radius, color, and spin angle. It also has a draw method to render the asteroid using OpenGL.

```
// Asteroid class.
class Asteroid
{
public:
    Asteroid();
    Asteroid(float x, float y, float z, float r, unsigned char colorR,
             unsigned char colorG, unsigned char colorB, float spinAngle);
    float getCenterX() { return centerX; }
    float getCenterY() { return centerY; }
    float getCenterZ() { return centerZ; }
    float getRadius() { return radius; }
    void setspinAngle(float ang){spinAngle=ang;}
    void draw();

public:
    float centerX, centerY, centerZ, radius, spinAngle;
    unsigned char color[3];
};
```

```
// Function to draw asteroid.
void Asteroid::draw()
{
    if (radius > 0.0) // If asteroid exists.
    {
        glPushMatrix();
        glTranslatef( x: centerX,  y: centerY,  z: centerZ);
        glRotatef( angle: spinAngle,  x: 0.0,  y: 1.0,  z: 0.0);
        glColor3ubv( v: color);
        glutSolidSphere(radius,  slices: (int)radius * 20,  stacks: (int)radius * 20);
        glPopMatrix();
    }
}
```

- Global variables are declared, including the size of the OpenGL window, spacecraft properties, animation variables, and spinning/rotation angles for each celestial body.
- The frameCounter function is a timer callback that counts the number of frames drawn every second. It outputs the frames per second (FPS) to the console.

```
// Routine to count the number of frames drawn every second.
void frameCounter(int value)
{
    if (value != 0) // No output the first time frameCounter() is called (from main()).
        std::cout << "FPS = " << frameCount << std::endl;
    frameCount = 0;
    glutTimerFunc( time: 1000,  callback: frameCounter,  value: 1);
}
```

- The setup function is an initialization routine. It sets up the display list for the spacecraft model (their color is set randomly but I will clearly specify them later in “drawscene” function while creating the lighting), initializes the array of asteroids, enables depth testing, and sets the background color.

```
void setup(void)
{
    spacecraft = glGenLists( range: 1);
    glNewList( list: spacecraft,  mode: GL_COMPILE);
    glPushMatrix();
    glRotatef( angle: 180.0,  x: 0.0,  y: 1.0,  z: 0.0); // To make the spacecraft point down the $z$-axis initially.
    glColor3f( red: 1.0,  green: 1.0,  blue: 1.0);
    glutWireCone( base: 5.0,  height: 10.0,  slices: 10,  stacks: 10);
    glPopMatrix();
    glEndList();
```

```
arrayAsteroids[0] = Asteroid( x: 0,  y: 0.0,  z: -40.0 ,  r: 10.0,
                              colorR: 235, colorG: 172, colorB: 63, sAngle: sunSpin);
arrayAsteroids[1] = Asteroid( x: 5,  y: 0.0,  z: -40.0 - 30.0,  r: 1.5,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: mercurySpin);
arrayAsteroids[2] = Asteroid( x: 10,  y: 0.0,  z: -40.0 - 30.0*2,  r: 3.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: venusSpin);
arrayAsteroids[3] = Asteroid( x: 15,  y: 0.0,  z: -40.0 - 30.0*3,  r: 4.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: earthSpin);
arrayAsteroids[4] = Asteroid( x: 20,  y: 0.0,  z: -40.0 - 30.0*4,  r: 2.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: marsSpin);
arrayAsteroids[5] = Asteroid( x: 25,  y: 0.0,  z: -40.0 - 30.0*5,  r: 6.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: jupiterSpin);
arrayAsteroids[6] = Asteroid( x: 30,  y: 0.0,  z: -40.0 - 30.0*6,  r: 5.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: saturnSpin);
arrayAsteroids[7] = Asteroid( x: 35,  y: 0.0,  z: -40.0 - 30.0*7,  r: 3.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: uranusSpin);
arrayAsteroids[8] = Asteroid( x: 40,  y: 0.0,  z: -40.0 - 30.0*8,  r: 3.0,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: neptuneSpin);
//moon
arrayAsteroids[9] = Asteroid( x: 21,  y: 0.0,  z: -40.0 - 30.0*3,  r: 1,
                              colorR: rand() % 256,  colorG: rand() % 256,  colorB: rand() % 256, sAngle: moonSpin);
glEnable( cap: GL_DEPTH_TEST);
glClearColor( red: 0.02,  green: 0.0888,  blue: 0.133333,  alpha: 0.0);

glutTimerFunc( time: 0,  callback: frameCounter,  value: 0); // Initial call of frameCounter().
}
```

- The init function sets up the lighting for the scene, specifically for the sun (arrayAsteroids[0]).

```
void init() {
    // Enable lighting
    glEnable( cap: GL_LIGHTING);
    glEnable( cap: GL_LIGHT0); // We'll use light source 0

    // Set up material properties of the sun (arrayAsteroids[0])
    GLfloat sunDiffuse[] = {  [0]: 1.0f,  [1]: 1.0f,  [2]: 1.0f,  [3]: 1.0f };

    glMaterialfv( face: GL_FRONT,  pname: GL_DIFFUSE,  params: sunDiffuse);
    glMaterialf( face: GL_FRONT,  pname: GL_SHININESS,  param: 128.0f);
}
```

- The animate function is a timer callback that handles the animation of the solar system. It updates the spinning angles of each asteroid around itself and updates the rotation angles of each asteroid around the sun (include the moon rotation angle around the earth).

```
void animate(int value)
{
    if (isAnimate)
    {//spinning of each asteroid around itself
        sunSpin += 0.5;
        if (sunSpin > 360.0) sunSpin -= 360.0;
        arrayAsteroids[0].setspinAngle( ang: sunSpin);

        mercurySpin += 2.0;
        if (mercurySpin > 360.0) mercurySpin -= 360.0;
        arrayAsteroids[1].setspinAngle( ang: mercurySpin);

        venusSpin += 1.0;
        if (venusSpin > 360.0) venusSpin -= 360.0;
        arrayAsteroids[2].setspinAngle( ang: venusSpin);

        earthSpin += 3.8;
        if (earthSpin > 360.0) earthSpin -= 360.0;
        arrayAsteroids[3].setspinAngle( ang: earthSpin);

        marsSpin += 4.0;
        if (marsSpin > 360.0) marsSpin -= 360.0;
        arrayAsteroids[4].setspinAngle( ang: marsSpin);

        jupiterSpin += 7.0;
        if (jupiterSpin > 360.0) jupiterSpin -= 360.0;
        arrayAsteroids[5].setspinAngle( ang: jupiterSpin);
```

```
arrayAsteroids[9].setspinAngle( ang: moonSpin);

//rotation around the sun
mercuryRot += 4.0;
if (mercuryRot > 360.0) mercuryRot -= 360.0;

venusRot += 2.5;
if (venusRot > 360.0) venusRot -= 360.0;

earthRot += 2.0;
if (earthRot > 360.0) earthRot -= 360.0;

marsRot += 1.0;
if (marsRot > 360.0) marsRot -= 360.0;

jupiterRot += 0.6;
if (jupiterRot > 360.0) jupiterRot -= 360.0;

saturnRot += 0.3;
if (saturnRot > 360.0) saturnRot -= 360.0;

uranusRot += 0.1;
if (uranusRot > 360.0) uranusRot -= 360.0;

neptuneRot += 0.08;
if (neptuneRot > 360.0) neptuneRot -= 360.0;
```

- The checkSpheresIntersection function checks if two spheres intersect given their positions and radus.
- The asteroidCraftCollision function checks if the spacecraft collides with an asteroid. It uses the bounding sphere of the spacecraft and checks for intersection with each asteroid.

```
// radius r1 and r2 intersect.
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));
}

// Function to check if the spacecraft collides with an asteroid when the center of the base
// of the craft is at (x, 0, z) and it is aligned at an angle a to to the -z direction.
// Collision detection is approximate as instead of the spacecraft we use a bounding sphere.
int asteroidCraftCollision(float x, float z, float a)
{
    int i, j;

    // Check for collision with each asteroid.
    for (i = 0; i<ROWS; i++)
        if (arrayAsteroids[i].getRadius() > 0) // If asteroid exists.
            if (checkSpheresIntersection( x1: x - 5 * sin( X: (M_PI / 180.0) * a),  y1: 0.0,
                                          z1: z - 5 * cos( X: (M_PI / 180.0) * a),  r1: 7.072,
                                          x2: arrayAsteroids[i].getCenterX(),  y2: arrayAsteroids[i].getCenterY(),
                                          z2: arrayAsteroids[i].getCenterZ(),  r2: arrayAsteroids[i].getRadius()))
                return 1;
    return 0;
}
```

- The drawScene function is the main drawing routine. It clears the color and depth buffers, applies rotations to the scene, and draws the celestial bodies (asteroids) using their properties and the draw method of the Asteroid class.
- In the interaction function when pressing on space the asteroids start their animation.

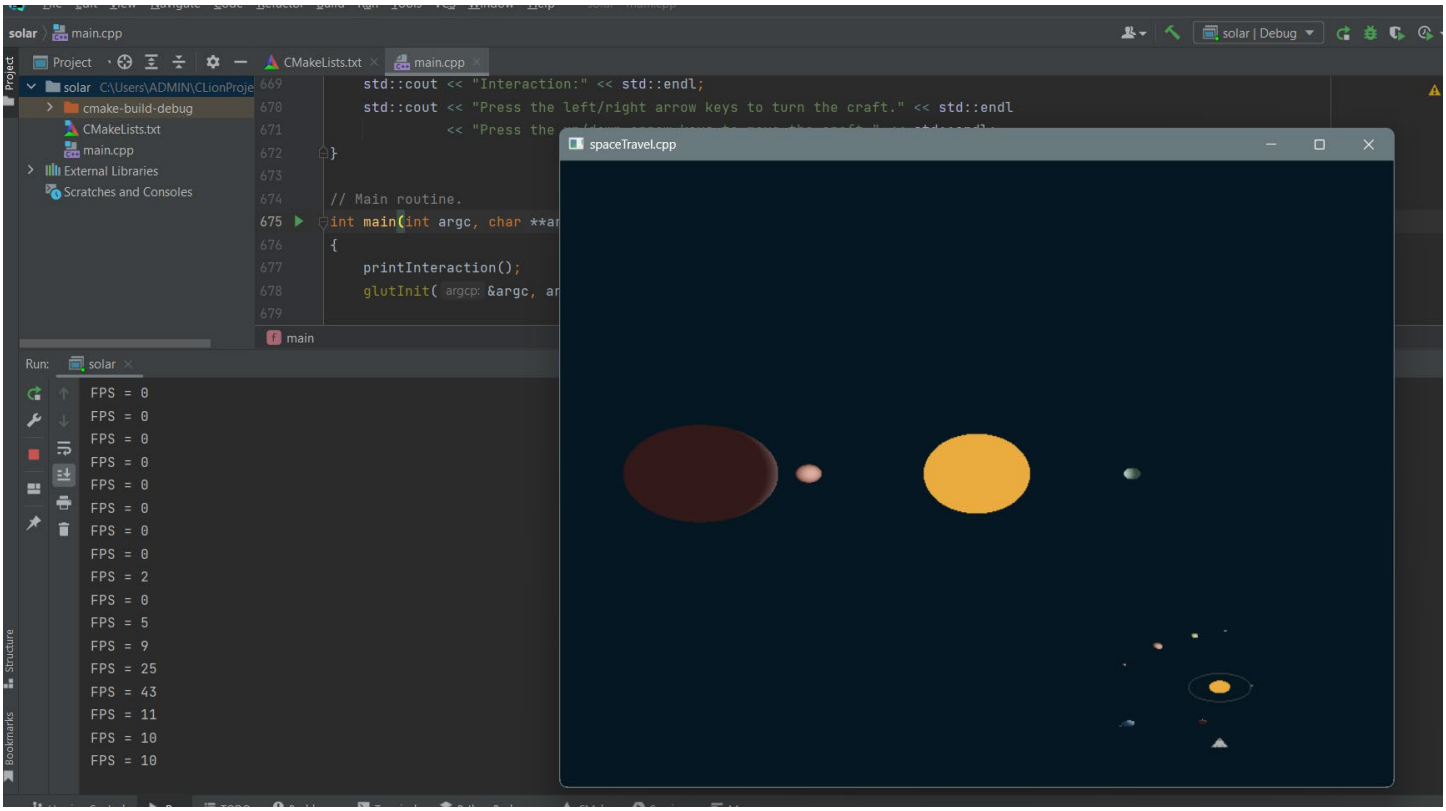
```
// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 27:
            exit( Code: 0);
            break;
        case ' ':
            if (isAnimate) isAnimate = 0;
            else
            {
                isAnimate = 1;
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

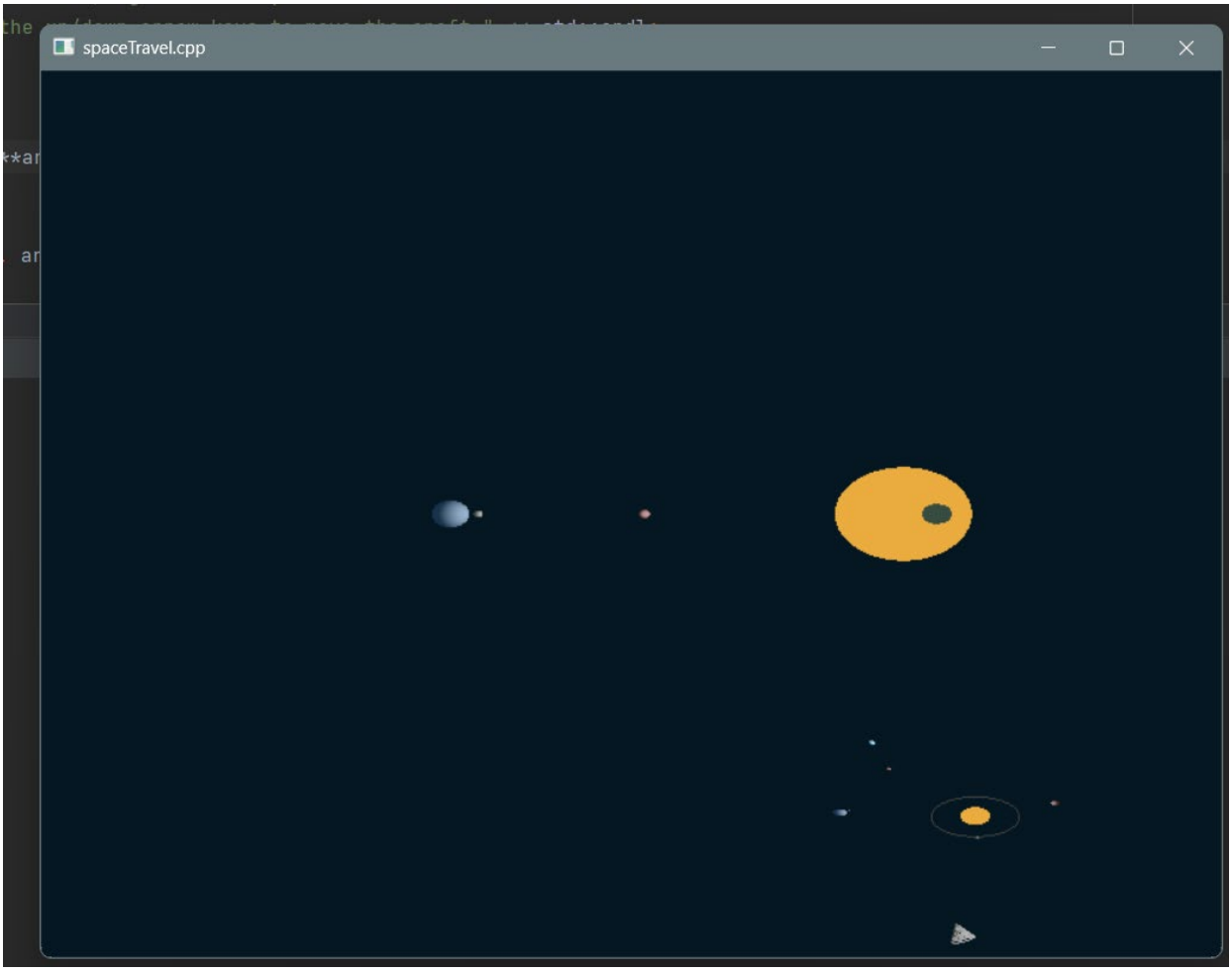
- Finally, the main function sets up the GLUT environment, creates the window, registers callback functions, and enters the GLUT event processing loop.

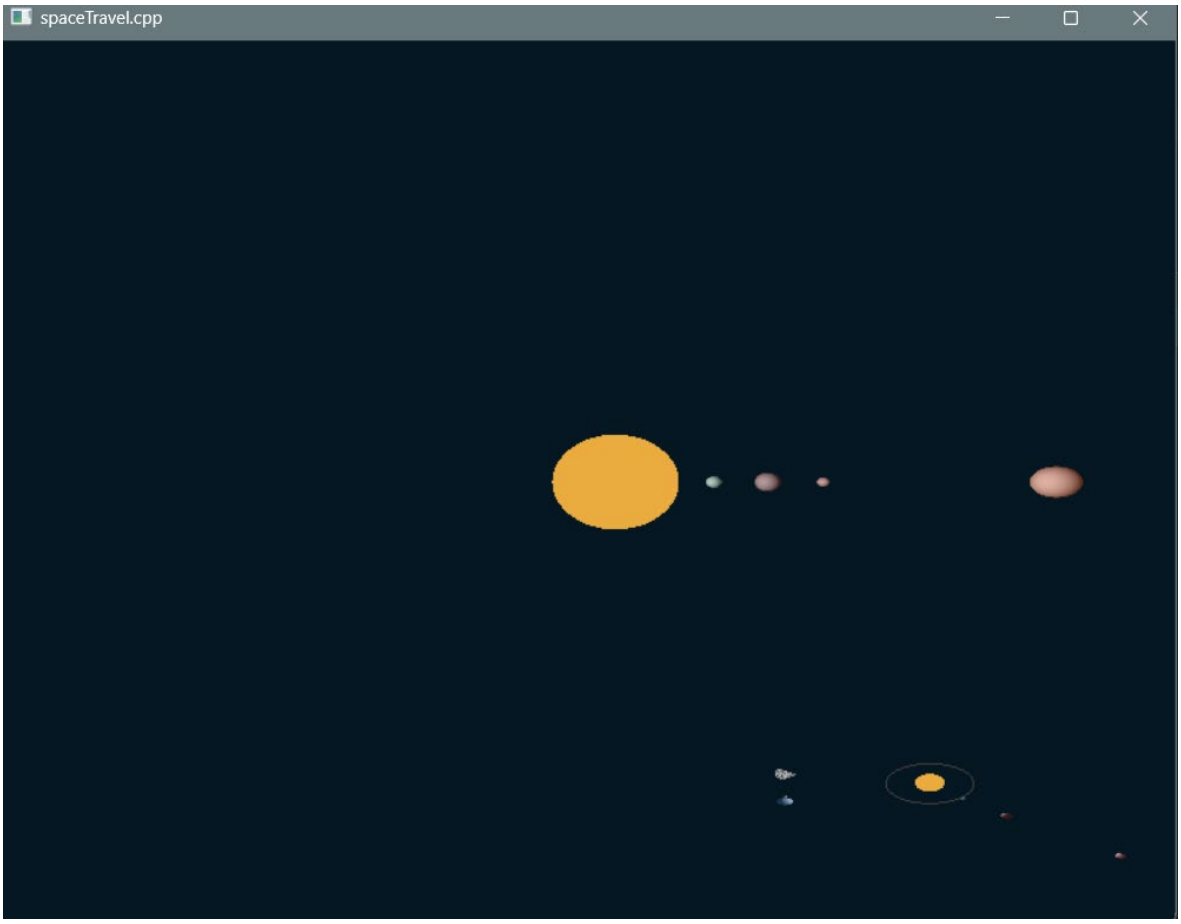
Overall, the code sets up a 3D scene with a spacecraft and multiple asteroids representing celestial bodies. It animates the rotation and spinning of the celestial bodies and provides collision detection between the spacecraft and the asteroids.

Screenshots of sample run:

I will include some screen shots but it will appear much better in the screen record that I will send.




spaceTravel.cpp


spaceTravel.cpp