Solar System Simulation using OpenGL

This project simulates a solar system, showcasing the orbital motion of planets around a central sun. Using OpenGL, it creates a dynamic and visually engaging 3D representation of planetary motion, incorporating real-time animations and customizable features.

Features

Rotation Angles

The simulation uses rotation angles to control the motion of celestial objects:

- **Global Variables**:

C++ code:

```
GLfloat angle = 0.0, angle1 = 0.0, angle2 = 0.0;

GLfloat planetAngles[8] = {0.0, ...};
```

  - angle: Tracks the rotation of the central torus, representing a conceptual celestial element.
  - planetAngles[8]: An array storing the rotation angles for each planet, determining their orbital positions.

This approach ensures a smooth and consistent animation as the angles update during each frame.

---

display() Function

The display() function is the core rendering routine responsible for drawing all objects in the scene. It includes:

- **Clearing the Frame Buffer**:

C++ code

```
glClear(GL_COLOR_BUFFER_BIT);
```

This clears the previous frame, ensuring no remnants of the last image remain.

- **Resetting Transformations**:

C++ code

```
glLoadIdentity();
```

This resets transformations to a default state, preventing transformations from accumulating across frames.

- **Drawing the Sun**:

C++ code:

```
glColor3f(1.0, 1.0, 0.0); // Yellow color
glutSolidSphere(1.0, 20, 20);
```

The sun is represented as a static yellow sphere at the center of the scene.

Planets

The simulation renders eight planets, each with unique properties defined by arrays:

1. **planetDistances[]**: Defines the orbital radii of planets, controlling how far they are from the sun.
2. **planetSizes[]**: Determines the size of each planet, visually distinguishing between smaller and larger celestial bodies.
3. **planetSpeeds[]**: Specifies the orbital speed of each planet, with closer planets moving faster to mimic realistic physics.
4. **planetColors[][3]**: Stores RGB color values for each planet, giving them distinctive appearances.

Planet Rendering Loop:

C++ code:

```
for (int i = 0; i < 8; i++) {
  glPushMatrix(); // Save the current matrix state.
  glRotatef(planetAngles[i], 0.0, 0.0, 1.0); // Rotate around the Z-axis.
  glTranslatef(planetDistances[i], 0.0, 0.0); // Move to orbital position.
  glColor3fv(planetColors[i]); // Set the planet's color.
  glutSolidSphere(planetSizes[i], 20, 20); // Render the planet as a sphere.
  glPopMatrix(); // Restore the previous matrix state.
}
```

This loop calculates and renders each planet's position, size, and color dynamically.

Central Torus

A torus (3D ring) is rendered at the center to add a decorative element. It rotates continuously to enhance the visual appeal of the simulation.

Torus Code:

C++ code :

```
glutWireTorus(0.3, 1.2, 20, 20);
```

- The inner radius is 0.3, and the outer radius is 1.2. It consists of 20 segments for both the horizontal and vertical divisions.

---

init() Function

This function sets up the environment for rendering:

- **Background Color**:

C++ code:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
```

Sets the background color to black, simulating the darkness of space.

- **Projection Matrix**:

C++ code:

```
glOrtho(-20.0, 20.0, -20.0, 20.0, -20.0, 20.0);
```

Creates an orthographic 3D viewing volume, defining the visible coordinate range.

These initial settings establish the context for rendering the solar system in an organized and visually accurate space.

Animation

The animation logic updates rotation angles continuously, ensuring fluid motion for the torus and planets:

- **Torus Rotation**:

C++ code:

```
angle += 0.005;
```

This gradually increases the torus's rotation angle to create a spinning effect.

- **Planetary Motion**:

C++ code:

```
for (int i = 0; i < 8; i++) {
  planetAngles[i] += planetSpeeds[i];
  if (planetAngles[i] >= 360.0) planetAngles[i] -= 360.0;
}
```

Each planet's angle increases based on its speed. Once an angle exceeds 360 degrees, it resets to zero, maintaining a circular orbit.

How to Run

1. **Install OpenGL and GLUT**: Ensure you have the necessary libraries installed for OpenGL development.
2. **Compile the Code**: Use a C++ compiler to compile the source code.
3. **Execute the Program**: Run the compiled executable to launch the simulation:

Code Overview

Key OpenGL functions used:

- **Rendering**: glutSolidSphere, glutWireTorus.
- **Transformations**: glRotatef, glTranslatef.
- **Matrix Management**: glPushMatrix, glPopMatrix, glLoadIdentity.
- **Buffer Management**: glutSwapBuffers (smooth frame rendering).

Future Enhancements

This project can be expanded in the following ways:

- Add textures to the sun and planets for more realism.
- Simulate moon orbits and additional celestial objects like comets or asteroids.
- Introduce camera controls to navigate through the solar system interactively.
- Implement realistic physics-based orbital mechanics.

Screenshot