Anar Shikhaliyev

Physics-Based Motion Control System

**Description**: Implement a system to generate the motions of rigid bodies using physical principles. Include the effect of gravity and collision between objects. You may make some simple approximations for rotational dynamics. For collisions, reflect the velocity. Collision detection becomes easy if you consider a bounding sphere of the objects in the scene.

**Overview**

This Python script demonstrates a simple physics simulation using OpenGL to render a 3D scene with bouncing balls. The simulation includes gravitational effects, collisions with the ground, walls, and other balls, and elastic collisions between balls.

**Dependencies**

The script relies on the following Python libraries:

math: Provides mathematical functions.

OpenGL.GL: OpenGL bindings for rendering graphics.

OpenGL.GLU: OpenGL Utility Library for higher-level functionality.

OpenGL.GLUT: OpenGL Utility Toolkit for handling windows and user input.

**Ball Class**

The Ball class represents a bouncing ball with properties such as position, radius, color, velocity, and mass. It includes methods for drawing the ball (draw), updating its position based on physics laws (update), and handling collisions with other balls (reflect\_velocities).

**Simulation Scene**

The scene consists of a ground and walls forming a cube, and multiple balls with different initial conditions (position, radius, color, velocity, and mass). The walls act as boundaries, and the ground reflects the balls upon collision.

**Rendering**

OpenGL is used to render the 3D scene. The render\_scene function defines the appearance of the ground, walls, and roof. The render function orchestrates the rendering process and continuously updates the scene. The script employs a simple timer-based animation loop (timer function) to update the scene at a regular interval, creating the illusion of continuous motion.

Feel free to modify the initial conditions of the balls or experiment with different physics parameters to observe their effects on the simulation. Program will take inputs like below:

**Ball([0.0, 40.0, 0.0], 5.0, [1.0, 0.0, 0.0], [1.0, 2.0, 3.0], 5)**

The inputs order: **position, radius, color, velocity, mass.**