CMSC 23700 **Introduction to Computer Graphics** **Project 1**
Winter 2014 **January 13**

A simple OpenGL animation

Due: Monday, January 27 at 10pm

1 Summary

This project is the first part of a two-part project, in which you will implement a number of standard rendering techniques. The goals of this project are to get your feet wet with simple graphics programming and to give you some quick feedback on the course submission and grading policies.

In this part of the project, you will implement two simple shader programs; one that does flat shading without lighting and one that implements per-pixel shading with a directional light.

2 Description

Your task in this project is to implement an interactive viewer of a simple 3D animation. The animation is of a $2 \times 2 \times 2$ box centered at the origin, which contains one or more bouncing spheres.

The viewer is located at $\langle 0, 0, -3 \rangle$ (this is called the *camera* or *eye* position) and is looking in the direction of the positive Z-axis. We recommend a field of view of around 65 degrees. Your program is responsible for displaying the animation and for allowing the user to change its state using the keyboard.

2.1 The geometric objects

There are two kinds of objects in the scene that you will render:

1. The box is $2 \times 2 \times 2$ units in size and is centered at $\langle 0, 0, 0 \rangle$. You should render five sides: back ($z = 1$), left ($x = -1$), right ($x = 1$), bottom ($y = -1$), and top ($y = 1$). You should render the sides as a pair of triangles (you can use `GL_TRIANGLE_FAN`).

2. The balls have radius $r = 0.05$. You can construct a triangle mesh to render for the sphere using the `Sphere` class in the provided library code. When adding a ball to the mix, pick a random location, initial velocity, and color. The initial speed (length of the velocity vector) should be 1.

2.2 Shading

Your program should support two shading/lighting models. The first is flat shading without lights; *i.e.*, the pixels of an object are rendered as the object's color without any lighting calculations. The second model is per-pixel lighting with ambient and directional lighting. The ambient light should have intensity $\langle 0.25, 0.25, 0.25 \rangle$, and the directional light should have intensity $\langle 1, 1, 1 \rangle$ and direction $\langle -1, -1, 0 \rangle$ (*i.e.*, the light is coming from the upper left toward the lower right).

2.3 User interface

Your viewer should support the following keyboard commands:

`l`	toggle the lighting mode between no lighting and directional lighting.
`+`	add a ball to the simulation.
`-`	remove a ball from the simulation.
`q`	quit the viewer.

You can limit the number of balls to 100. Your application should also handle resizing the viewport.

3 Sample code

To get you started, we will seed your repository with sample code. The code will be organized into a directory called `prom-1` with three subdirectories: `build`, `data`, and `src`. The `build` directory contains a Makefile for compiling your project. You should put your shader source files in the `data` directory (in future projects, this directory will hold texture data and geometric models). Lastly, your source code belongs in the `src` directory.

4 Hints

We recommend the staging your implementation effort in the following steps:

1. Start by getting the box to display with no lighting and getting features such as resizing and quiting to work.
2. Animate a single bouncing ball and handle ball-wall collisions.
3. Finally, add multiple balls and ball-to-ball collisions.
4. Add directional lighting.

4.1 Representing the balls

The position of a ball can be represented by three pieces of information: a time t_0, an initial position $\mathbf{q}$, and a velocity vector $\mathbf{v}$. Given these vlaues, the ball's position can be defined as a function of time

$$\mathbf{p}(t) = \mathbf{q} + (t - t_0)\mathbf{v} \quad \text{for } t \geq t_0$$

In addition, you will want to record the ball's color information and the time of its next collision.

4.2 Event queue

To manage the simulation, you will need to maintain a priority queue of events ordered by time. There are two types of events: ball-wall collisions and ball-ball collisions. A given ball will continue moving in its current direction until it hits something, so the only time you need to update the state of the system is when an event occurs. Because the objects and their motions are simple, it is possible to accurately predict what the next event to affect a given ball is likely to be.

If we are going to render the scene at time t, then we should first process all of the events that occur at or before t. To process a wall collision, we compute the ball's new velocity ($\mathbf{v}$), set its initial time value (t_0) to the collision time, and set its initial position ($\mathbf{q}$) to the collision position. We then compute the ball's next collision by testing it against the walls and other balls. Note that if the ball is going to collide with some other ball, you may have to recompute other collisions. For example, say that ball B was scheduled to hit ball C, but we now discover that ball A will hit B first. This means that we must recompute C's next collision (and so on).

For simplicity, we will assume that a ball is involved in only one kind of collision at a time (*i.e.*, no ball-ball-wall or three-ball collisions).

4.3 Ball-wall collisions

Given a ball with position

$$\mathbf{p}(t) = \mathbf{q} + t\mathbf{v}$$

the time of intersection with the plane defining a given wall can be determined by projecting out the appropriate component of the velocity vector $\mathbf{v}$. For example, if $\mathbf{v}_x > 0$,[1] then the ball will hit the right side of the box ($x = 1$) at time t satisfying

$$\begin{aligned} (1 - r) &= \mathbf{p}(t)_x \\ &= \mathbf{q}_x + t\mathbf{v}_x \end{aligned}$$

thus the time of intersection with the right wall is

$$t = \frac{1 - r - \mathbf{q}_x}{\mathbf{v}_x}$$

If $\mathbf{v}_x < 0$, then we check it against the left wall ($x = -1$ plane). For each ball, we must test it against at most three planes and then take the minimum time as the time of collision.

When a ball hits a wall, its new velocity vector will be a reflection of its current vector off the plane of the wall.[2] Figure 1 illustrates this situation.

4.4 Ball-ball collisions

Consider two balls, whose positions are specified as

$$\begin{aligned} \mathbf{p}_1(t) &= \mathbf{q}_1 + t\mathbf{v}_1 \\ \mathbf{p}_2(t) &= \mathbf{q}_2 + t\mathbf{v}_2 \end{aligned}$$

[1] In practice, we need to test for $\mathbf{v}_x > \epsilon$, where ϵ is a small positive number (*e.g.*, 10^{-6}).

[2] Question 1 of Homework 1 addresses the computation of the reflection vector.

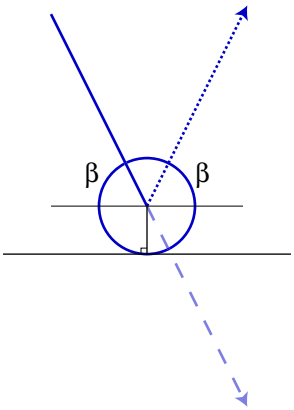

Figure 1: Ball-to-wall collision

The distance between the centers of these two balls is

$$d(t) = \sqrt{(\mathbf{p}_1(t) - \mathbf{p}_2(t))^2}$$

$$\begin{aligned} d(t)^2 &= (\mathbf{p}_1(t) - \mathbf{p}_2(t))^2 \\ &= ((\mathbf{q}_1 + t\mathbf{v}_1) - (\mathbf{q}_2 + t\mathbf{v}_2))^2 \\ &= ((\mathbf{q}_1 - \mathbf{q}_2) + t(\mathbf{v}_1 - \mathbf{v}_2))^2 \\ &= (\mathbf{q} + t\mathbf{v})^2 \\ &= \mathbf{q}^2 + 2(\mathbf{q} \cdot \mathbf{v})t + \mathbf{v}^2 t^2 \end{aligned}$$

where $\mathbf{q} = (\mathbf{q}_1 - \mathbf{q}_2)$ and $\mathbf{v} = (\mathbf{v}_1 - \mathbf{v}_2)$. When $d(t) \leq 2r$, then the balls have collided, so we can detect the point of collision by finding the roots of

$$\mathbf{v}^2 t^2 + 2(\mathbf{q} \cdot \mathbf{v})t + (\mathbf{q}^2 - 4r^2)$$

If there are no real roots or both roots are negative, then there is no collision, if both roots are positive, then the smaller one is the time of collision, and if one root is positive and the other negative, then the balls are currently overlapping (which should not happen in your simulation).

When two balls collide, their direction vector should change as follows. First compute the plane at the point of contact that is tangent to both balls. Then, the new velocity vector is the reflection of the old vector off of the tangent plane. Figure 2 illustrates this situation. If a ball is moving away from the tangent plane, then its velocity vector should not change.

5 Submission

We have set up an **svn** repository for each student on the `phoenixforge.cs.uchicago.edu` server. We will collect the projects at 10:00pm on Friday January 24th from the repositories, so make sure that you have committed your final version before then.

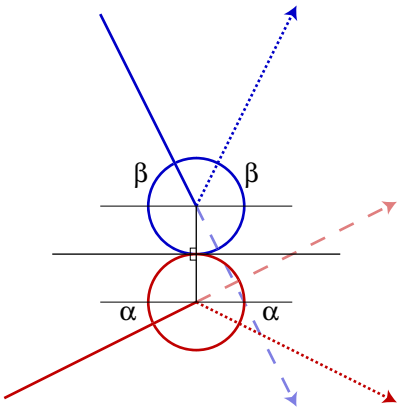

Figure 2: Ball-to-ball collision

History

2014-01-19 Add missing values for lighting.

2014-01-19 Corrected description of the ball's initial velocity so that it agrees with the sample code and updated the due date.

2014-01-12 Original version.