COP 290 - Assignment 1
Screensaver App

Akshay Kumar Gupta	Barun Patra	J. Shikhar Murty
2013CS50275	2013CS10773	2013EE10462

1 Overall Design

A modular design pattern is followed with two main classes named `Ball` and `Box`. `Ball` defines the properties of a ball while `Box` defines the area in which the balls are confined. The number of balls n is accepted as user input and n balls of different sizes moving at different velocities are simulated in a box. The user has the option to change the speeds of individual balls. A multi-threaded paradigm is used in which one thread is used to control one ball. The n threads communicate with each other to correctly handle ball collision. Barrier synchronization has been used to used to synchronize the threads but a switch to one-to-one communication has been planned.

2 Components

1. Class `Ball`

   The `Ball` Class defines the properties of the ball. It has the following members:

   - `Centre` – This is a `Vector` of the centre coordinates of the ball. It defines the coordinates of the centre of the ball. [`Vector` is a user-defined class for a mathematical 3-dimensional vector.]
   - `Velocity` – This is a `Vector` which defines the velocity of the ball.
   - `Radius` – The radius of the ball.
   - `Mass` – Mass of the ball which is proportional to the radius of the ball.
   - `Red, Green, Blue` – Data members which together define the color of the ball.
   - `initBall()` – Member function which initializes the centre, velocity, radius and color of the ball uniformly at random in a specified range.

2. Class `Box`

   The `Box` class defines the properties of the box in which balls are confined. It has the following members:

   - `Width` - The width of the box.
   - `Height` - The height of the box.
   - `Depth` - The depth of the box.

3. GUI

   The GUI is handled by the following functions:

   - `display()` – It is responsible for drawing the balls and the box.
   - `mouseInput()` – It detects mouse input from the user and acts accordingly. Mouse input is used to select a particular ball and to press a button to increase/decrease its speed (More in the *Variable Ball Speed* section).
   - `keyboardInput()` – It detects keyboard input from the user and acts accordingly. It can be used as an alternative to the GUI button to increase/decrease ball speed (More in the *Variable Ball Speed* section).

4. Physics

   The main components where physics is handled is:

   - ballCollisionHandler(): This function accepts two Balls, say ball 1 and ball 2, which have collided and updates their velocities. Velocities are computed very efficiently because costly trigonometric computations are not carried out. This is because the position and velocity of a ball are objects of a class called `Vector`. This class provides functions for a mathematical vector namely `add`, `subtract`, `modulus`, `dotProduct`, `crossProduct`, `normalize` and `scale`. Using these functions, the new velocities are computed using the equations:

$$\hat{n} = \frac{\vec{r_1} - \vec{r_2}}{||\vec{r_1} - \vec{r_2}||}$$

$$w_1 = \frac{m_2 e(u_2 - u_1) + m_1 u_1 + m_2 u_2}{m_1 + m_2}$$

$$w_2 = \frac{m_1 e(u_1 - u_2) + m_1 u_1 + m_2 u_2}{m_1 + m_2}$$

$$\vec{v_1} = \vec{u_1} - (\vec{u_1} \cdot \hat{n})\hat{n} + w_1 \hat{n}$$

$$\vec{v_2} = \vec{u_2} - (\vec{u_2} \cdot \hat{n})\hat{n} + w_2 \hat{n}$$

Moreover, the collision handler also has support for inelastic collisions. So the user can pass an extra command line argument specifying the coefficient of restitution e for collisions.

Computing the new velocity in the event of a collision of a ball with the wall is taken care of in the relevant thread function itself.

3 Inter-Component Interaction

Sub-component interaction is carried out in the following scenarios:

- Ball to Wall collision: Collision of a wall with a ball is detected when:

  $$|Pos_{wall} - Pos_{centre}| \leq radius$$

  The appropriate changes are made to the ball velocity according to the elasticity (=1 by default).

- Ball to Ball Collision: Collision of a ball is detected when

  $$||\vec{r_1} - \vec{r_2}|| \leq radius_1 + radius_2$$

  where $\vec{r_1}$ and $\vec{r_2}$ denote the centers of the two balls.
  The new velocities are computed according to the equations mentioned above.

- Physics and GUI: Every time the new positions of the balls are calculated the `display()` function is called which draws the balls.

4 Multi-Threading

There are n threads for n balls, wherein each thread is responsible for controlling a single ball. Currently barrier synchronization is used to synchronize the threads. Firstly, the ball positions are updated. Then wall collisions are detected and updated. Then ball collisions are detected and handled inside a mutex lock to prevent race conditions.

5 Variable Ball Speeds

The user can select a ball with a left mouse click which will make the ball white in color. The ball's speed can then be increased or decreased between a pre-defined maximum or minimum by pressing the relevant button or through keyboard input, whichever is preferred by the user. The ball's velocity which is maintained in the `Ball` object will be updated accordingly. The model for movement and collisions ensures that the physics does not break. When the next ball is selected the previously selected ball will return to its original color and the new ball will become white.

6 Unit Testing

Extensive unit testing will be carried out on the following components:

- Physics: We will pass different ball positions and velocities to the
  - Wall collision function
  - Ball collision function

  and check the new ball positions and velocities using assert statements. We will check all the different possible corner cases that may arise. We will check that the increase and decrease in ball speeds occur correctly.
- GUI: We will check that the balls are created with random positions, speed and color. We will visually check that balls and box are rendered correctly.
- Threads: We will check that all updates to variables which can potentially be accessed by multiple threads are done inside a mutex lock. We will also check that cases where threads/mutexes fail to initialize are handled correctly.

7 Planned Improvements

We intend to add the following features:

- Slider for varying ball speed: A slider that will move up and down to show the speed of the selected ball.
- Quad Tree: Currently our model is $\mathcal{O}(n^2)$ for collision detection and handling. We plan to implement a quad tree to track ball positions so that unnecessary checks for collision do not take place. This can potentially improve the running time by a large amount.
- M threads for N balls: We plan to add support for controlling n balls using m threads where $m < n$.
- One-to-one thread communication: Currently we are using barrier synchronization model for thread synchronization. We intend to switch to a one-to-one thread communication model as suggested.