Computer Graphics

Instructor: Sungkil Lee

Assignment 1: Moving Circles

2021312661

안준성(Joon Seoung An)

소프트웨어학과 (Software)

1. Algorithms

(1) Algorithm for Keyboard Interaction

First, there is a keyboard interaction that needs to be implemented for this assignment. The actions are as follows:

- Exit the program: Press ESC or q.
- Help: Press F1 or h to check which command to execute.
- Adjust the number of circles: Use the + and - keys.
- Toggle wireframe: Press w.
- Reinitialize circles: Press r to reposition the circles.

This keyboard interaction is implemented in main.cpp by defining various keys in a keyboard function and setting up event callbacks using glfwSetKeyCallback(). This way, when the program runs and a key is pressed, the corresponding event is processed.

(2) Create Circles

The process of creating circles is handled by the create_circles() function in circle.h. Circles are created sequentially based on a count, each with a random position, radius, color, and speed. During the creation process, an algorithm is used to avoid collisions between circles; this will be explained later. A triangular approximation with 24 tessellations was used to generate circles.

(3) Movement

Next, the movement of the circles is implemented. The update function in circle.h uses the time difference (dt) passed from main.cpp to update the movement. The circle's position is updated by adding the product of dt, its velocity, and VELOCITY_SCALE.

(4) Collision Detection 1 - Initial Collision

Initial collision detection refers to avoiding collisions among circles at the moment of initialization. First, the radius is set by taking the square root of the total number of circles to be drawn. Circles are then generated at random positions; if a newly generated circle collides with an existing one, it is discarded and recreated to avoid initial collisions.

(5) Collision Detection 2 - Wall Collision

Wall collisions are managed in main.cpp using two global variables. In the update() function, the aspect ratio is calculated from the window size (x and y dimensions), and x_bound and y_bound are determined based on whether the width or height is larger. These values are then passed to the render() function when updating each circle.

Within the update function in circle.h, conditional statements check for collisions with the top, bottom, left, and right walls, and adjust the circle’s position and velocity accordingly.

(6) Collision Detection 3 - Collision Between Circles

Collisions between circles are also handled in main.cpp by passing the entire set of circles as an argument when updating each circle in the render() function.

In circle.h, the update function iterates through all circles and computes a collision impact as a floating point value. If this value is greater than 0, it indicates that a collision has occurred, and the circle's velocity is adjusted.

The process involves first calculating the normal vector between the two circles. Next, the relative speed difference is computed. The dot product of the relative velocity and the normal vector is then calculated to obtain a scalar value. If this scalar value is less than 0, it confirms that a collision has occurred, and the velocities of the colliding circles are updated accordingly. This code is implemented with reference to the Elastic Collision section in the assignment handout.

2. Data structures

Below is a detailed explanation of the circle struct in *circle.h*. The struct contains the following values:

1. **pos:** 2D position
2. **velocity:** The circle's speed
3. **radius:** The radius of the circle
4. **theta:** Rotation angle (not used in this context)
5. **color:** The circle's color in RGBA values
6. **model_matrix:** The matrix M in the TRS (Translation, Rotation, Scaling) model
7. **lvoc:** A value that stores the level of collision

The circle struct also includes the following functions:

1. **update:** A function that updates each circle, implementing movement and collision detection.
2. **collide:** A function that returns the collision impact as a floating-point value.

3. Discussions

The implementation successfully met all the requirements outlined in Section 4. Initially, simply adjusting the velocity based on the collision impact caused the speed to gradually increase over time. However, by controlling the velocity using the dot product of the relative velocity and the normal vector, the issue was resolved.

4. Check requirments

(1) Initialize a window size whose aspect ratio (width/height) is 16/9 (e.g.,1280×720). As long as the window is not a square and circles are not ellipses, it is alright.

(2) The program has more than 16 solid circles.

(3) Your program can change the number of circles up to 32 using '+/-' (or arrow) keys.

(4) The initialization of circles should use random radii, colors, and locations.

(5) The initialization of circles uses random velocities. Your circles should not be significantly slow.

(6) The circles' movement speeds should be consistent across different computers; use a time difference between consecutive frames to define the movement of circles.

(7) The initial positions of circles avoid overlaps and collisions with the other circles and walls.

(8) When a circle meet other circles or side walls, most of them (more than 90% of circles) can avoid collisions well.

5. Example