Report: A1 Moving Circles

LEE SUNG JAE

2016314718

Mathematics Department

1. IDEAS

To randomize the location of balls, I saved the location and radius of each circle at an array. When it creates a new circle, it checked the length of the two centers and the sum of two radii. I allowed a new circle to be created only when the following inequality satisfies,

$$r1 + r2 <= \text{sqrt}((x1 - x2)^2 + (y1 - y2)^2) \quad \ldots (1)$$

where (x1,y1) is the location of the original circle center, (y1,y2) is the location of the new circle center, and the r1 and r2 are the radii of each circle. By this strategy, no two pairs of circles can be overlapped and make problems.

To implement movements of circles, I let circles hold velocity and theta values. Theta value means the direction of velocity. At every frame, the movement was recalculated in render() function by the following equation.

$$x` = x + \text{velocity} * \cos(\text{theta}), \quad y` = y + \text{velocity} * \sin(\text{theta}) \quad \ldots (2)$$

Sometimes, two circles try to intrude each other's area. Then in render() function, I used (1) to check if any two circles are overlapped in the next frame. If it is the case, I moved them back from each other by

$$(r1 + r2 - \text{sqrt}((x1 - x2)^2 + (y - y2)^2)) / 2 \quad \ldots (3)$$

to the direction opposite to touching each other. This direction vector is calculated by

$$((x1 - x2) / \text{sqrt}((x1 - x2)^2 + (y1 - y2)^2),$$

$$(y1 - y2) / \text{sqrt}((x1 - x2)^2 + (y1 - y2)^2)) \quad \ldots (4)$$

This vector is normalized. Hence, it can bed multiplied by (3). Also, when they collide, their velocity and theta values have to change. These values are updated by using the formula in Two-dimensional section of https://en.wikipedia.org/wiki/Elastic_collision as the professor recommended. I assumed that every circle has the same mass.

When a circle touches a wall, it must be bounced back with the same velocity but a different direction. The new direction depends on which wall a ball crashes, if it is a vertical wall,

$$\theta' = \pi - \theta \quad \ldots (5)$$

however, if it is a horizontal wall,

$$\theta' = 2\pi - \theta \quad \ldots (6)$$

2. Implementation

I started the A1 assignment on gl-02-circles blueprint. I mostly did not change the core codes like math library files and GLSL codes. I changed the codes of circle.h and main.cpp. When I randomize the attributes of circles, I used <ctime> and <cstdlib> libraries. When I implement codes, I made some mistakes and one of the hardest things to fix up was the new theta after collision of two balls. The Wikipedia article above only showed the vx and vy after collision. However, I had to make one velocity value and theta value by recalculating them. To calculate a new theta value, I tried arctan(vy / vx). I realized later that this only covers the arguments of $-\pi/2 \sim \pi/2$. However, I needed $-\pi \sim \pi$. Then, I realized that theta value is the angle from the vector(1,0). Hence, I used inner production and implemented it by arccos(vx / |v|). This had another problem. This returns the smaller angle between the velocity and the vector(1,0). So, the argument becomes $0 \sim \pi$. After I noticed this problem, I realized vy is not used in the formula. Therefore, I multiplied it by -1 when vy is negative. This worked well.

Another hardest thing was (4), namely, moving circles to the right position after collision. If two circles are not moved to the right position and still stay overlapped, it makes a huge problem like a wrong velocity, semi-permanent overlapping, and sudden acceleration. These are because when they are still overlapped after collision, it tries to recalculate the new theta and velocity but because they already got the new theta which goes away from each other, the new theta becomes the direction of contacting each other again. This calculation is done many times in short time and makes problems. When I made mistakes, I tried two circles going back from each other by 1. Using the new theta and 2. Using the original theta as the direction. However, both failed. They resulted in another collision. The only thing that worked was (4): Put them back to the state when they were touching each other. This strategy needs the normal vector of two centers of circles when they collide. This prevented any consistent overlapping after collision.