Assignment 6: Physics simulation: collision

EC602 Design by Software

Fall 2018

Contents

1 Introduction

This problem involves predicting and movement and collisions of a large number of objects moving in a two-dimensional space.

The problem is relevant, for example, to air-traffic control, self-driving cars, or game simulations (asteroids, curling, pool)

The objects undergo perfect elastic collision.

1.1 Assignment Goals

The assignment goals are to help you learn about physical modeling for a simulator or game.

1.2 Group Size

For this assignment, the maximum group size is 3.

1.3 Due Date

This assignment is due 2018-11-8 at 23:59:59

1.4 Submission Link

You can submit here: assignment 6 submit link

1.5 Points

This assignment is worth 10 points (5 for C++, 5 for python)

2 Collision Detector

Your task is to determine collisions between moving objects given a starting scenario.

The program will be called “collision.cpp” or “collision.py” (one for C++, one for python)

2.1 Starting scenario

The objects are specified in a file which gives an ID, a location at time $t = 0$ (x/y coordinates in meters) and velocity (x/y coordinates, in meters/second).

Each object is a circle (disc) with a radius of 5 meters and a mass of 10 kg.

2.2 Motion model

The objects travel on a two-dimensional surface called the "field". Each object travels in a straight line at a constant velocity. Each object will continue travelling in this direction forever, unless it collides with another object.

2.3 Collision model

Two objects collide at the moment the distance between them becomes equal to the *collision distance*. For this problem, the collision distance will be the sum of the radii of the colliding stones.

2.3.1 Elastic Collisions

When two objects collide, they undergo elastic collision.

If two objects with positions $\mathbf{r}_1$ and $\mathbf{r}_2$ collide, the new velocities will be

$$\mathbf{v}_1' = \mathbf{v}_1 - \frac{2m_2}{m_1 + m_2} \frac{(\mathbf{v}_1 - \mathbf{v}_2) \cdot (\mathbf{r}_1 - \mathbf{r}_2)}{(\mathbf{r}_1 - \mathbf{r}_2) \cdot (\mathbf{r}_1 - \mathbf{r}_2)} (\mathbf{r}_1 - \mathbf{r}_2)$$

$$\mathbf{v}_2' = \mathbf{v}_2 - \frac{2m_1}{m_1 + m_2} \frac{(\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{r}_2 - \mathbf{r}_1)}{(\mathbf{r}_2 - \mathbf{r}_1) \cdot (\mathbf{r}_2 - \mathbf{r}_1)} (\mathbf{r}_2 - \mathbf{r}_1)$$

In the above equation $\cdot$ represents the dot product. If $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d)$ then

$$\mathbf{u} \cdot \mathbf{v} = ac + bd$$

These equations are from https://en.wikipedia.org/wiki/Elastic_collision in section "Two-dimensional collision with two moving objects".

2.3.2 Collision prediction

Let the position of object i at time t be denoted as $\mathbf{r}_i(t)$ and its radius R_i.

For any pair of objects i and j, they collide at the moment that

$$|\mathbf{r}_i(t) - \mathbf{r}_j(t)| = R_i + R_j$$

This is most easily calculated by squaring, so we get

$$|\mathbf{r}_i(t) - \mathbf{r}_j(t)|^2 = (\mathbf{r}_i(t) - \mathbf{r}_j(t)) \cdot (\mathbf{r}_i(t) - \mathbf{r}_j(t)) = (R_i + R_j)^2$$

Since the velocities are constant, you can write

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

where $\mathbf{p}_i$ is the position of the center of object i at time $t = 0$ and $\mathbf{v}_i$ is the velocity of object i.

The equation can be solved for t, and it turns out that it is of the form

$$at^2 + bt + c = 0$$

This is the quadratic equation, and it has 0, 1, or 2 real solutions.

It is an exercise for you to determine what these mean as applied to the physics of the situation.

For the collision between i and j to be a real collision, it turns out that

$$(\mathbf{r}_i - \mathbf{r}_j) \cdot (\mathbf{v}_i - \mathbf{v}_j) < 0$$

is a necessary condition. This means "these objects are approaching each other."

Time did not exist before $t = 0$, so if two objects would have collided in the past, we ignore this event. It never happened.

2.3.3 Collision Modeling

We assume that the object collisions occur instantly. If multiple collsions occur simultaneously, they should be processed pair-wise until no more collisions are pending.

These idealized assumption leads to possibly unrealistic behaviors.

For example, in the following situation

```
thrown -50 0 10 0
lower  18 -9 0  0
upper  18  9 0  0
```

there are two possible outcomes, `thrown` could end up moving left and slightly down, or left and slightly up. This depends on whether `thrown` interacts with `lower` or `upper` first.

Your program should model this behavior correctly.

3 Program requirements

3.1 Input format

The input will be from *stdin*

Each line of input specifies one of the objects. It specifies the object ID, the location at time $t = 0$ (x/y coordinates in meters) and the initial velocity (x/y components, in meters/second).

When the program encounters end-of-file (EOF, or ctrl-D), it means that all objects have been entered and the simulation should begin.

Here is an example of the format for three objects:

```
2MU133 -34.94 -69.13 0.468 -0.9
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
```

The first object has a license plate or ID number 2MU133.

Its initial position is (-34.94,-69.13) and its initial velocity is 0.468 m/s in the x-direction and -0.9 m/s in the y-direction.

This is the first part of the file called random10.coordinates

3.2 Program configuration

Your program should print out a report for each future time specified on the command line, in order of increasing time.

For example

```
collision 10 50
```

This program will print out a positional report for times 10 seconds and 50 seconds.

```
collision 10 50 <random10.coordinates >random10.results
```

This saves the results into a text file called `random10.results`

3.3 Output format

Your program should print the time followed by the object ID, location, and velocity of each object, like this:

```
10
2MU133 -34.94 -69.13 0.468 -0.9
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
50
2MU133 -34.94 -69.13 0.468 -0.9
0WI913 -43.08 92.12 -0.811 -0.958
6UP738  2.97 -66.25 -0.077 0.074
```

Not The locations and velocities should be calculated and printed with 8 digits of precision (in the example shown, the numbers are exact.)

The objects should be output in the same order they were specified in the input.

If a collision or collisions occur at the same time as a report time, all collisions should be effected prior to the velocity and position report.

If there are no objects, the times should be printed.

3.4 Notes

- The ID is a single contiguous string in any format.
- the objects are never in an initially overlapping or colliding state.
- you may assume that every object is at least the collision distance away at time $t = 0$ (i.e. touching is ok but not overlapping), and the first collision occurs at $t > 0$.

3.5 Error handling

3.5.1 Bad input

If the input format has any problems (too many fields on one line, invalid numbers, etc) the program should exit with return value 1.

3.5.2 Command line problems

If any value on the command line is not a number, the program should exit with return value 2.

The following are examples of things that are not numbers:

- “alpha”

- “a 1”
- “120q”

The following are examples of things that are numbers:

- “+23”
- “-23”
- “3.4e+8”
- “45”
- “45.501”

Negative time values should be ignored.

If there are no valid values on the command line, the program should exit with return value 2.

3.6 Output when return code

As long as the correct return code is produced, any output to stderr is allowable, but nothing should be printed to stdout.

4 The assignment requirements

You should implement the program in C++ `collision.cpp` AND in python, `collision.py`

4.1 Visualization tools

The following are python scripts for visualizing the problem:

- visualize takes an input situation and draws the positions and velocities
- visualize_motion takes the output of `collision` programs and creates a movie using matplotlib.