Assignment 4: Physics simulation: bouncing spheres

EC602 Design by Software

Fall 2021

Contents

1 Introduction

This problem involves predicting the movement and collisions of a large number of objects moving in a three-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 and the numerical capabilities of python.

1.2 Group Size

For this assignment, the maximum group size is 3.

1.3 Due Date

The assignment is due 2021-10-18 at 11:59:59

1.4 Submission Link

You can submit here:

Assignment Four Submission Page

1.5 Points

This assignment is worth 10 points.

2 Bouncing Spheres

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

The program will be called "spheres.py"

2.1 Starting scenario

The objects are specified by entering the following information: the mass, radius, x/y/z position, x/y/z velocity and name of each sphere.

All the values are in SI units: kg, m, and m/s.

The location information (x/y/z coordinates in meters) and velocity (x/y/z coordinates, in meters/second) is the initial values for time $t = 0$.

2.2 Motion model

The objects travel in three dimensional space. 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.

We ignore any gravitational or frictional forces.

2.2.1 Limits of space

In this simulation, the spheres are all contained within a spherical container, called “the universe”.

When objects hit the edge of the universe, they bounce back into the universe without loss of energy as if they are bouncing off of a wall with effectively infinite mass.

The radius of the containing sphere is specified on the command line, and the center of the containing sphere is the origin, i.e. (0,0,0).

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 spheres.

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”.

They apply equally well to three dimensions.

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 collisions occur simultaneously, they should be processed pair-wise until no more collisions are pending.

These idealized assumption leads to possibly unrealistic behaviors.

Your program should model this behavior correctly.

3 Program requirements

3.1 Input format

The input will be from *stdin*

Here is an example input:

```
20 1  0 0 0    0 0 1 one
2  5  0 1 100  0 0 0 two
```

Sphere `one` has mass 20, radius 1. Its initial position is (0,0,0) and its initial velocity is (0,0,1)

Sphere `two` has mass 2, radius 5. Its initial position is (0,1,100) and its initial velocity is (0,0,0)

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

The number of spheres can be 0, 1, 2, or more (there is no limit).

3.2 Command Line arguments

The program must be run with two command line arguments, like this:

```
python spheres.py 120  14.5
```

which means the container is 120 m radius and the time at which to conclude the simulation is 14.5 seconds.

3.2.1 Numerical ranges

All mass, radius, positions can be arbitrary floating point numbers.

The collision limit is a positive integer.

3.3 Program behavior

3.3.1 Input Phase

The program starts off by prompting for the starting scenario:

```
Please enter the mass, radius, x/y/z position, x/y/z velocity
and name of each sphere
When complete, use EOF / Ctrl-D to stop entering
```

3.3.2 Output phase

The program, once it knows about the initial conditions, will first print out the initial situation, including the universe's radius, and the status of each sphere.

It then reports on

- each collision (sphere to sphere)
- each reflection (sphere to container)

After each collision or reflection, the condition of all spheres is reported, including the new velocities.

When there are no more events, the program should exit with return code 0.

Each floating point number should be printed using the `:g` format specifier, as in `f"{x:g}"`

3.3.3 Input and Output Example

Consider this scenario with three spheres:

```
20 1  0 0 0    0 0 1 one
2  5  0 1 100  0 0 0 two
3  1  2 -1 -2    0 0 0 three
```

The output of

```
python spheres.py 120 105
```

when this is the input will be:

```
Here are the initial conditions.
universe radius 120.0
end simulation 105
one m=20 R=1 p=(0,0,0) v=(0,0,1)
two m=2 R=5 p=(0,1,100) v=(0,0,0)
three m=3 R=1 p=(2,-1,-2) v=(0,0,0)
energy: 10
momentum: (0,0,20)

Here are the events.

time of event: 94.0839
colliding one two
one m=20 R=1 p=(0,0,94.0839) v=(0,-0.0298792,0.823232)
two m=2 R=5 p=(0,1,100) v=(0,0.298792,1.76768)
three m=3 R=1 p=(2,-1,-2) v=(0,0,0)
energy: 10
momentum: (0,0,20)

time of event: 102.539
reflecting two
one m=20 R=1 p=(0,-0.252632,101.044) v=(0,-0.0298792,0.823232)
two m=2 R=5 p=(0,3.52632,114.946) v=(0,0.189874,-1.78267)
three m=3 R=1 p=(2,-1,-2) v=(0,0,0)
energy: 10
momentum: (0,-0.217836,12.8993)
```

3.4 Notes

The ID is a single contiguous string in any format.

The objects are never in an initially overlapping or colliding state. Therefore, 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

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

As long as the correct return code is produced, any output to stderr is allowable, but nothing should be printed to stdout (except for the initial instructions.)

4 Files for Download

The following ZIP file contains a suite of input files and output files.

- examples_spheres.zip

If the input scenario is "three.txt", then

```
python 120 2 < three.txt  > three_120_2_myid.txt
```

creates the output "answer" file "three_120_2_myid.txt"

Here is a script that will run all the scenarios covering in the above ZIP file:

- make_spheres_results.py

Please read the documentation of that script before running it.