

Scientific Programming in Python
Project
6 October 2009

The Physics of Colliding Balls: Part 1

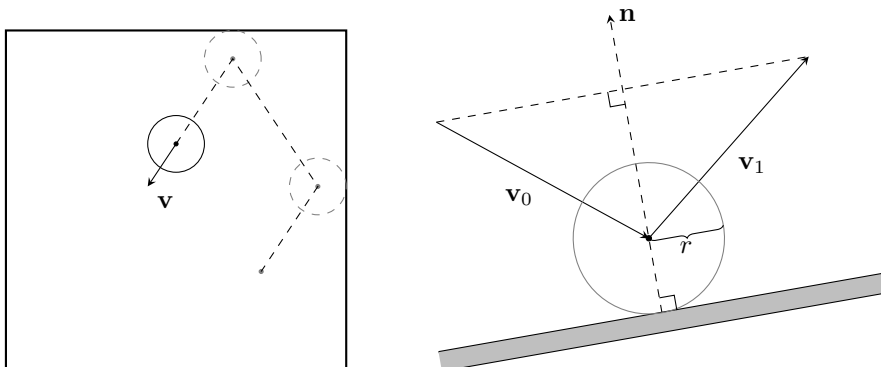
In this project we will simulate the motion of solid balls with known mass and size. We can use this to demonstrate simple things like the quadratic curve followed by a ball falling under gravity, slightly more complex things like balls undergoing elastic collisions, and complex things like an ideal gas. In this first part of the project we will model a single ball inside a box. We will use discrete time steps—i.e. we will update the state of the ball once per time interval Δt . This gives the discrete update equation for the position \mathbf{x} , velocity \mathbf{v} , and acceleration \mathbf{a} of the ball.

$$\begin{aligned}\mathbf{x}(t + \Delta t) &= \mathbf{x}(t) + \mathbf{v}(t)\Delta t \\ \mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \mathbf{a}(t)\Delta t\end{aligned}$$

We will start by having no acceleration, $\mathbf{a}(t) = \mathbf{0}$, but can also simulate gravity with $\mathbf{a}(t) = \mathbf{g}$.

Colliding with a wall

A ball bouncing off a wall is simple a reflection around the tangent to the wall.



By dividing the velocity into two components—tangent to the plane and normal to the plane of the wall, we see that the tangent component remains unchanged while the normal component changes sign. This can be expressed in the equation

$$\mathbf{v}_1 = \mathbf{v}_0 - 2\mathbf{n}(\mathbf{v}_0 \cdot \mathbf{n})$$

where \mathbf{v}_1 is the ball velocity after the bounce, \mathbf{v}_0 is the ball velocity before the bounce, and \mathbf{n} is a unit vector normal to the wall.

Tasks

1. Create a ball object that contains all of the state and properties of the ball that we are interested in. These might include position, velocity, mass, radius.
2. Define the dimensions of the box that contains your ball. Initially this can be a square box that is aligned with the coordinate axes, but you can extend this to more general shapes.

3. Update the state of the ball for a discrete timestep Δt , check if the ball has collided with a wall and handle the collision if necessary.
4. Visualise the system using the `visual` library.
5. Check that your implementation is correct by calculating the kinetic energy of the ball. It should remain constant.

Additional tasks

- Add gravity so that the path of the ball is a quadratic curve rather than a straight line.
- Add friction in the form of energy loss during a collision.