

# Collision\_Detection

May 28, 2024

## 1 Simple Collision Detection

[File as PDF](#)

In particle simulations it is sometimes useful to detect collisions. This can be done for the sake of visualization or to really simulate physical collisions. Since we now know about classes. We will use this knowledge to implement some basic collision detection to our particle system. The method we use for collision detection in this code is a simplified model and is not entirely physically correct. It's a basic form of collision detection and resolution that checks if two objects are overlapping and then pushes them apart.

In a more physically accurate model, you would also consider factors like:

- **Elasticity:** When two objects collide, they don't just stop or move away from each other. They bounce off each other with a certain amount of energy depending on their elasticity.
- **Momentum Conservation:** In a real-world collision, the total momentum of the system (the sum of the momenta of the two colliding objects) is conserved before and after the collision.
- **Angular Momentum:** If the collision is not head-on (i.e., the objects don't hit each other directly in the center), it can cause rotation.
- **Friction:** This could also play a role in the collision, depending on the surfaces of the colliding objects.

For the moment we would like to skip these physical complications and just implement a simple version. Later in the notebook you will get a more physically correct version.

We actually split the task into a `solver class`, which is responsible for the time stepping through the animation and solving the collisions as well as `particle class` which represents the particle and holds all its properties.

The main ingredient for the solver is the `solve_collision`. This function looks for the overlap of two particles by calculating their distance. If the distance is smaller than the sum of their radii, then a collision occurs. To correct this collision, both particles involved in the collision are pushed back by the same amount  $\text{delta} * (\text{distance\_vec} / \text{dist})$ . While this is only done for a pair always, it might create trouble for other pairs. Yet we ignore all that. Also note that this is not according to the momentum conservation, as both particles are pushed along the connecting line by the same amount.

The class below is the particle class, which holds all details of the particle and updates the particles position according to acceleration and speed. It ignores, however, all the collision, which are

addressed by the Solver.

Below we just generate a list of particles that we can supply to the solver.

### **1.1 Initializing the simulation**

This creates a solver object and initializes a list of positions for drawing.

### **1.2 Function to run the simulation**

This holds our animation plotting with `ipywidgets`.

```
Canvas(height=400, width=400)
```