1 Inelastic Collisions

Up until now, your program has solved for the ideal case of no wind drag and no energy losses at collision. In order to add a little more realism to the simulation, we will also let some of the energy be lost during the collision between the circle and the wall. Energy is lost in real collisions due to non-elastic deformation of the circle and/or wall surface, which depends mostly on the velocity of the circle. Hence, when the circle hits a surface, we will calculate the rebounded velocity $\vec{u}_b$ as

$$\vec{u}_b = (1 - c_d)\vec{u}_s$$

(1)

where $\vec{u}_s$ is the velocity at the surface of the wall after accounting for the change in direction due to the bounce and $c_d$ is a loss coefficient between 0 and 1. This is the simplest model for an inelastic collision. More physically-based models require knowledge of the coefficient of restitution of the collision, which depends on the velocity and the material properties of the circle and wall, and is usually found through experiment.

When we add inelasticity to the collisions, the circle will eventually come to rest on the floor of the domain. When this happens, it is likely at some point that a small round-off error will allow the ball to just pass through the floor. Once that happens, it will continue to fall and will be lost. In order to protect against this, you should have a check that sets the position of the bottom of the ball to zero if it goes below zero at the end of a time step calculation.

2 Interacting Circles

Now that you have a general code to account for a collision, it should be fairly easy to reuse that code to handle arbitrary other obstacles and collisions. Remember, whenever the circle changes direction in the vertical due to a collision (e.g., floor, ceiling, other circles altering the vertical component of the velocity), the circle accelerates down and decelerates up. Also, whenever the
circle encounters a horizontal surface, the vertical velocity does not change direction, and the collision is simpler.

When you have multiple walls or objects, you should also realize that a circle may interact with more than one wall or object within one time step. Make sure that your code accounts for all interactions within each time step; otherwise, your circles may pass through walls or each other.

3 Assignment

Before you begin this assignment, make sure that your code for Programming Assignment #3a is working properly and solving all parts of the problem. To complete Laboratory #3, also do the following parts:

1. Update your collision solver to include inelastic collisions following the approach in Section 1. Test your code using the initial conditions from Assignment #3a with $c_d = 0.2$. Let your code run until the circle comes to rest.

2. Generalize your collision function and allow for including a ceiling in the domain. This surface should behave similarly to the ground, but reflect the circle down. Test your code with the same initial conditions as before, but with a ceiling at twice the initial height of the circle. Include inelastic collisions with $c_d = 0.2$. Does your circle come to rest at the same $x$–position as in the previous step? Why or why not?

3. Add one additional collision by selecting one of the following options or your own option. Be sure to reuse your collision code and account for all collisions within each time step. You may reduce the time step as needed to make the code easier to implement (i.e., the fewer the collisions per time step, the easier.).

   • Add walls on both sides of the circle so that it bounces within a closed box. For this part, note that the circle may hit more than one wall at each time step, and that a circle may hit two walls at the same time if it hits a corner. Can you find a condition where the circle escapes (passes through a wall that was not accounted for)? Where should you add a check in your code to test for this condition?

   • Keep the floor and ceiling, but add one more circle to the domain. Select initial conditions for each circle that ensure that they will collide at least once. Demonstrate your working code for one test condition with colliding circles. Is it possible for the circles to collide with each other more than once in this domain? If so, does your code work? How could you test for that?

   • Remove the ceiling, but include three or more circles that may bounce and interact with each other. Demonstrate your working code with a minimum of one collision between each of the circles. Could you generalize your code to $n$ circles? What would you have to check for to include that?

Test your code by setting $c_d = 0$ and plotting the magnitude of the velocity of the first ball as a function of time. What is the right answer? Include this test and plot in your Memorandum.
4. For the text of your memorandum, give an introduction, describe any problems your code is encountering, and provide all plots required in Assignment #3a. Repeat those plots for the inelastic collisions for item 1, above. Also, demonstrate the trajectory of the circle(s) for items 2 and 3, above (plot $x$ versus $y$ for the circle position for the whole simulation). Be sure you have answered all questions posed in the Assignment sections of Laboratories 3a and 3b. Include your final version of the code in an Appendix.