Your First Physics Simulation: The Bouncing Ball

I. PRELIMINARIES

In this exercise I want you to simulate a bouncing ball. We'll start with a perfectly elastic ball traveling in one dimension with no air resistance, and we'll make things more complicated (or "interesting") as we go along.

In class on Wednesday we talked about how Newton's Second Law is really a differential equation. For motion in the vertical, or y direction we have

$$\frac{dp_y}{dt} = (F_{\text{net}})_y, \tag{1}$$

which for constant mass objects simplifies to

$$\frac{dv_y}{dt} = \frac{1}{m}(F_{\text{net}})_y. \tag{2}$$

For small time intervals Δt this relationship can be approximated

$$\frac{\Delta v_y}{\Delta t} \simeq \frac{1}{m}(F_{\text{net}})_y, \tag{3}$$

which can be rearranged to give

$$\Delta v_y \simeq \frac{1}{m}(F_{\text{net}})_y \Delta t, \tag{4}$$

or

$$v_{2_y} \simeq v_{1_y} + \frac{1}{m}(F_{\text{net}})_y \Delta t. \tag{5}$$

In the special case with no air resistance $(F_{\text{net}})_y = -mg$, so Eq. (5) simplifies to

$$v_{2_y} \simeq v_{1_y} - g\Delta t. \tag{6}$$

Exercise: You should convince yourself that there is a similar approximation that can be used to update the position:

$$y_2 \simeq y_1 + v_{1_y}\Delta t. \tag{7}$$

Before you start doing any programming you should use Eqs. (5) and (7) to fill in the entries of the following table. This gives the position and velocity for a ball dropped from rest at a height of 4 m. When doing your calculations use a time step of $\Delta t = 0.1$.

t	$y(t)$	$v_y(t)$	F_{net}/m
0.0	4.0	0.0	-9.8
0.1			
0.2			
0.3			

II. COMPUTER EXERCISES

- Open your working program that had a red ball between blue plates. You should save it with a new name because we're going to modify things. Run your program and make sure it works. Notice that the field of view changes as the objects move. The default behavior of VPython is to adjust the field of view automatically so that the displayed objects fill the screen. If you want to fix the field of view you can add the following line to your program immediately after the lines in which you first define the ball and the walls:

  ```
  scene.autoscale = 0
  ```

  This turns the autoscaling feature off.

- Last time you defined a variable `v` that corresponded to a velocity. Today you will see that the vector velocity is an *attribute* of the `ball` *object* that you defined; VPython keeps track of the velocity for you. Delete the line from your program that looks like

  ```
  v = 1
  ```

  and replace it with the line

  ```
  ball.velocity = vector(1,0,0)
  ```

  Then delete the line that looks like

  ```
  ball.pos.x = ball.pos.x + v*dt
  ```

  and replace it with the line

  ```
  ball.pos = ball.pos + ball.velocity*dt
  ```

  You should run your program and see that nothing has changed.

  NOTE: In this program we have now updated the **vector** position of the ball. We

could have updated just the x-component of the position with the line

```
ball.pos.x = ball.pos.x + ball.velocity.x*dt
```

but it's nice to be able to do all three components in one vector operation.

- Now modify your program so that you have a single blue plate oriented horizontally, and a single *stationary* red ball. The plate will serve as the floor on which your ball will bounce. Put the floor at position $(0, 0, 0)$. Give the red ball a radius of 0.5 and put it at position $(0, 4, 0)$ so that it is directly above the center of your floor. Run your program to make sure that the floor and the ball are displayed correctly.

- Modify your program so that the position and velocity are updated according to Newton's Second Law in every pass through the `while` loop. Remember that everything that is indented after the `while` statement is "inside" the loop.

  - Check that the motion of the falling ball "looks" correct when you run your program. (Is it obvious that the ball is accelerating?)

  - Check that your program is actually calculating the correct numbers for each time step. To do this you can change the value of `dt` to 0.1 and check to see if your program's values agree with those you entered in the table earlier in this exercise. You will need a `print` statement to see the values calculated by your program, and you may want to slow down the `rate` so that it's easier to stop the execution of your program at an appropriate place.

- Make your ball bounce; it should bounce indefinitely.

- Make your ball do something "more interesting." For example,

  - Give it an initial velocity with an x component. Make it bounce if it hits the floor, but continue down indefinitely if it misses.

  - Put in air resistance.

  - Make the ball *inelastic*, i.e., make it loose some energy on every bounce.

  - Make the ball change colors when it bounces.

  - ...