

## LAB 01-4: Modeling Motion with VPython

In this activity, you will learn how to use a computer to model motion. You will write your program in a language called Python, using a python module called Visual. Together, we will refer to them as “VPython.” Note that you can accomplish the same goal with other programming languages; however, this language is especially easy to use for beginners and the graphics are 3-D which makes it both fun and instructional. Before you begin, write down the following data from your previous experiment analyzing the motion of a steel ball on a track.

|          |  |
|----------|--|
| velocity |  |
|----------|--|

### General Structure of a Program

In general, every program that models the motion of physical objects has two main parts:

1. **Before the loop:** The first part of the program tells the computer to:
  - (a) Create 3D objects.
  - (b) Give them initial positions and velocities.
  - (c) Define numerical values for constants we might need.
2. **The while loop:** The second part of the program, the loop, contains the lines that the computer reads to tell it how to increment the position of the objects over and over again, making them move on the screen. These lines tell the computer:

To learn how to model the motion of an object, we will write a program to model the motion of a ball moving with a constant velocity.

### Writing your program

It is helpful to start with a previous program, as opposed to starting one from scratch.

#### Start with a previous program

1. Open your previous program for creating vectors (see *Introduction to 3D computer modeling*).
2. Save this file with a new name like `ball-uniform-motion.py`.
3. Edit this program to only a ball of radius 0.05 m that is at the initial position  $\langle -1, 0, 0 \rangle$  m. Name the sphere `ball`.
4. Run your program to ensure that it works. It should ONLY the ball at the given position,  $\langle -1, 0, 0 \rangle$  m.
5. Add the line below to create a track that is at the origin and has a length of 3 m, a width of 0.1 m, and a height of 0.05 m. Note that the y-position is -0.075 m so that the ball appears to be on top of the track.

```
track=box(pos=vector(0,-0.075,0), size=(3,0.05,0.1), color=color.white)
```

6. Now, define the initial velocity of the ball (i.e. its velocity at  $t=0$  when the simulation begins). Let's assume for now that the ball is moving to the right at a speed of 0.3 m/s, similar to the slowly moving ball in the video analysis lab. Just as the position of the ball is referenced as `ball.pos`, let's define its velocity as `ball.v` which indicates that `v` is a property of the object named `ball`.

```
ball.v=vector(0.3,0,0)
```

Whenever you want to refer to the velocity of the ball, you must refer to `ball.v`.

### Define values for constants we might need

To make the cart move, we calculate its new position after each time interval  $\Delta t$  using the equation  $\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$ . This equation is valid to use only if the time interval is very small.

But what exactly is “very small”? It depends on the process. For example, if you are modeling the motion of Earth around Sun, one orbit takes 365 days. Thus, 1 hour is very small compared to 365 days.

For a ball rolling on a level track, it’ll only travel about 2 m and probably won’t travel at more than about 1 m/s for a total time elapsed of two seconds. Thus, the time interval  $\Delta t$  should be much smaller than about one or two seconds. We can try a hundredth of a second or a thousandth of a second perhaps. The advantage of using a smaller time interval is that the calculations are more accurate. The disadvantage is that more calculations are required and thus the animation may run very slowly. Of course, for uniform motion, the time interval is less important. But for non-uniform motion, it’s essential to use small time intervals.

7. For now, let’s use 1 hundredth of a second as the time interval,  $dt$ . Define a variable  $dt$  for the time interval.

```
dt=0.01
```

8. Also, let’s define the total time  $t$  that is a running clock. We will later add  $dt$  to  $t$  during each time interval in order to calculate the total time. The clock starts out at  $t = 0$ .

```
t=0
```

That completes the first part of the program which tells the computer to:

- (a) Create the 3D objects.
- (b) Give the ball an initial position and velocity.
- (c) Define variable names for the clock reading  $t$  and the time interval  $dt$ .

### Create a “while” loop to continuously calculate the position of the object.

We will now create a `while` loop. Each time the program runs through this loop, it will do two things:

- (a) Calculate the change in the ball’s position and use it to find a new position.
- (b) Calculate the total time by incrementing  $t$  by  $dt$ .
- (c) Repeat.

9. For now, let’s run the animation for 10.0 s. On a new line, begin the while statement. This tells the computer to repeat this instructions for as long as  $t \leq 10.0$  s.

```
while t < 10.0:
```

Make sure that the `while` statement ends with `:` because Python uses this to identify the beginning of a loop.

In class, you learned that the new position of the object after a short time interval  $\Delta t$  is

$$\vec{r}_f = \vec{r}_i + \vec{v}\Delta t \tag{1}$$

10. Let’s use this equation to calculate the new position of the ball after a time interval  $dt$ . Immediately after the `while` statement, type the following line. *Note that it must be indented.* I’ve included the `while` line for clarity.

```
while t < 10.0:
    ball.pos=ball.pos+ball.v*dt
```

Also note that I didn't define "final" or "initial" because the program literally takes the current position of the ball, adds its displacement, and then assigns the result to the position of the ball. In this way, it updates the ball's position.

11. Now let's increment the clock reading  $t$

```
t=t+dt
```

Note that it takes the clock reading  $t$ , adds the time step  $dt$ , and then assigns the result to the clock reading. Thus, through each pass of the loop, the program updates the clock reading.

12. Save and run the program.
13. The animation will likely run too fast. If so, use the `rate` statement to slow down the computer. Type the following line as the first line (indented of course) after the `while` statement. Again, I show the while statement for clarity.

```
while t < 10.0:
    rate(100);
```

14. Save and run the program.
15. You can make it run indefinitely (i.e. without stopping) by saying "while true" and in Python the number 1 is the same as "true" so you can write:

```
while 1:
```

Change the `while` statement to be `while 1: .`

16. Save and run the program.
17. Change the initial velocity of the ball to be  $\langle 0.3, 0.5, 0 \rangle$  m/s.

Before running the program, predict what you think the path of the ball will look like.

18. Save and run the program.
19. You can print the position and time data. In the last line of the while loop (indented of course), type the following:

```
print t, ball.pos;
```

20. Run the program and view the data that is printed. If you want to print just the x-position, change it to the following line:

```
print t, ball.pos.x;
```

Note that `ball.pos.x` refers to the x-position of the ball.

## Application

1. You have developed a *model* of uniform motion. Every model has:
  - (a) Assumptions
  - (b) Initial conditions

In this (idealized) model of a ball rolling with uniform motion on a track, what are the assumptions of your model? (There's really only one primary assumption.)

In this model, what are the initial position and the initial velocity of the ball? These are the initial conditions of the model.

2. Edit your simulation to make the ball move from the right side of the track to the left side of the track.
3. Edit your simulation to make the ball move from the top right corner of the window to the bottom left corner with uniform motion.
4. Start the ball on the left side of the track at  $x = -1.5$  m, give it an initial velocity of  $\langle 0.3, 0, 0 \rangle$  m/s, and edit the while loop so that it stops when the ball reaches the x-position of 2.0 m. Print the clock reading and position of the ball. At what time (i.e. clock reading) is the ball at  $x = 2.0$  m?
5. Answer the following question.

A steel ball on a track is at  $x = 1.25$  m at  $t = 0$  and moves with a constant velocity of  $\langle -0.6, 0, 0 \rangle$  m/s. What is the clock reading when the ball crosses the origin ( $x = 0$ )? Determine your answer both analytically (i.e. mathematically by hand) and numerically (i.e. computationally with a computer simulation). Compare your results obtained with the two methods.