Lab1/CW1: Introduction to 3D Modeling

September 20, 2017

1 Introduction to 3D Computational Modeling

1.1 Objective

In this course you will construct computational models to predict and display the motion of interacting objects in 3D.
You will use a programming environment called VPython, which consists of the widely-used programming language Python plus a 3D graphics module called `vpython` which makes it possible to do vector algebra and to visualize vector quantities in 3D.

After completing this activity you should be able to:

- Use VIDLE, the interactive editor for VPython
- Write a simple VPython program
- Create 3D objects such as spheres and arrows
- Use vectors in VPython

This file is adapted from the Lab VP01 materials from Matter & Interactions 4e

1.2 What is a Computer Program?

You have seen short computer programs, written in VPython, in Chapter 1 of the *Matter & Interactions* textbook. The important things to understand about a computer program are:

- A computer program consists of a sequence of instructions that describe a calculation.
- The computer carries out the instructions one by one, in the order in which they appear, and stops when it reaches the end.
- Each instruction must be entered exactly correctly, as if it were an instruction to your calculator.
- If the computer encounters an error in an instruction (such as a typing error), it will stop running and print an error message.

1.3 Your First Program: Creating 3D Objects

These instructions assume that you are using a computer on which VPython has been installed.

- Watch the first VPython instructional video VPython Instructional Videos: 1. 3D Objects demonstrating how to easily create 3D objects in VPython.

- Make sure you start with the following two lines, which must go at the beginning of every VPython program to ensure compatibility:

```
from __future__ import division, print_function
from visual import *
```

The first line is needed because there are different versions of Python, which is continuously improved and upgraded. The statement `from __future__ import division, print_function` tells the Python language to treat 1/2 as 0.5, and makes the new form of print statements work on older versions of Python.

You don't need this statement if you are using Python 3.0 or later, but it doesn't hurt, because it is simply ignored by later versions of Python. Note that there are *two* underscores immediately before and after the word `future`.

The second line tells the program to use the 3D module (called `visual`). The asterisk means, **Add to Python all of the features available in the visual module**.

Note: `from XXX import *` is not a recommended practice. However, since we are newbies, let's just do it for now.

1.3.1 Example 1

- Follow the instructions in the video, create 3D objects

```
from __future__ import division, print_function
from visual import *
scene_eg1=display(title='First Program') # Create a scene
```

1.3.2 Zooming and Rotating

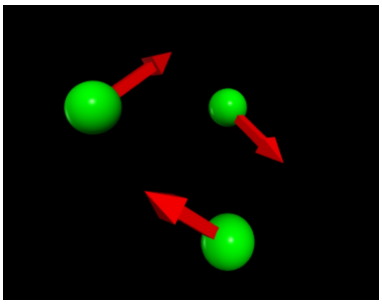
By default the origin $\langle 0, 0, 0 \rangle$ is at the center of the scene, and the **camera** (that is, your point of view) is looking directly at the origin.

When you first run the program, the coordinate system has the $+x$ direction to the right, the $+y$ direction pointing up, and the $+z$ direction coming out of the screen toward you. This is the same convention used in the * Matter & Interactions* textbook.

- On a two-button mouse, hold down both mouse buttons and move the mouse up and down to make the camera move closer or farther away from the center of the scene. (On a one-button mouse, hold down the **ALT/Option** key and the mouse button.)
- On a two-button mouse, hold down the right mouse button alone and move the mouse to make the camera **revolve** around the scene, while always looking at the center. (On a one-button mouse, hold down **CTRL** and the mouse button.)

1.3.3 Exercise 1

Write a program to create the following scene



1.4 Naming Objects and Variables

- Watch the second VPython instructional video VPython Instructional Videos: 2. Variable Assignment demonstrating how to name objects and varibles so you can refer to them later in the program.

1.5 Print function

You can use the `print` function to print out the information. For example,

```
print(ball.pos)
```

will print out the position of an object called `ball`.

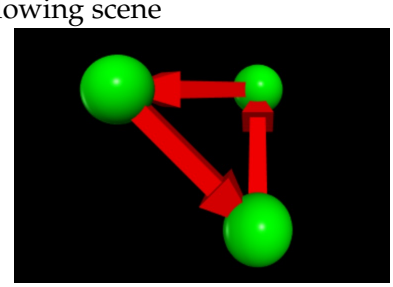
1.5.1 Example2

Modifying this code following the instructions in the video

```
from __future__ import division, print_function
from visual import *
scene_eg2=display(title='Naming Objects and Variables') # Create a scene
sphere(pos=vector(-1,0,0),radius=0.25, color=color.red)
sphere(pos=vector(1,1,0),radius=0.15, color=color.orange)
arrow(pos=vector(0,0,0),axis=vector(0,1,0),color=color.yellow)
```

1.6 Exercise 2

Write a program to create the following scene



1.7 Finding and Fixing Errors

Everyone makes errors when programming–you may already have made one or more errors in the process of creating your first program. During the course of your work with VPython you will need to find and fix two different kinds of errors:

- Syntax Errors
- Math, Physics, or Logic Errors

Both kinds of errors are similar to errors you may make when working pencil and paper problems using a calculator. Errors made entering expressions into your calculator are syntax errors. Using an inappropriate or incorrect equation, or trying to divide a scalar by a vector, is a physics or math error.

1.7.1 Syntax Errors

Common syntax errors include

- Typos and spelling errors
- Missing commas, colons, equal signs, etc.
- Unmatched parentheses

1.8 Example 3

- Watch the VPython instructional video VPython Instructional Videos: A. Debugging Syntax Errors which illustrates some common errors.
- Fix the syntax errors in the following program

```
from vpython import *
scene_eg3=display(title='Syntax Errors') # Create a scene
Bob=sphere(pos=vector(-1,-1,-1) radius=0.4, color=color.red)
Lucy=sphere(pos=vector(1,0,0, radius=0.4, color=color.green)
arrow(pos=bob.pos, axis=vector(1,-1,0), color=color.magenta)
```

1.9 Other Things to Notice

1.9.1 Autoscaling and Units

Change the position of one of the objects in your program so it is three times farther from the origin (for example, if a sphere's position is $\langle 6, 0, 0\rangle$, move it to $\langle 18, 0, 0\rangle$). Can you still see all the spheres when you run the program?

VPython automatically *zooms* the camera in or out so that all objects appear in the window. This behavior is called *autoscaling*. Usually this is helpful, but occasionally we may want to turn off autoscaling.

**What units do you think VPython is using?

Since VPython automatically moves the camera to try to keep all objects in the display window, it can handle any consistent set of units. We will always use SI units in our physics programs.

1.9.2 Comments

Comment lines start with a # (pound sign). A comment line can be a note to yourself, such as:

```
# objects created in the following lines
```

Or a comment can be used to remove a line of code temporarily, without erasing it. You can also put a comment at the end of a line: `sphere() # it's round.`

1.10 Example 4

Comment out all but one arrow in the following program. For the remaining arrow: Change something in the arrow's code such that the arrow is half as long and points in the opposite direction, with its tail remaining on the same sphere.

```
scene_eg4 = display(title='Using comments') #create another scene
Bob=sphere(pos=vector(-1,-1,-1), radius=0.4, color=color.red)
Lucy=sphere(pos=vector(1,0,0), radius=0.4, color=color.green)
arrow(pos=Bob.pos, axis=vector(1,-1,0), color=color.magenta)
```