

Introduction to 3D computer modeling – Drawing vectors*

OBJECTIVES

In this course you will construct computer models to:

- Visualize motion in 3D
- Visualize vector quantities like position, momentum, and force in 3D
- Do calculations based on fundamental principles to predict the motion of interacting objects
- Animate the predicted motions in 3D

To do this you will use a 3D programming environment called VPython.

In this lab you will learn:

- How to use IDLE, the interactive editor for VPython
- How to structure a simple computer program
- How to create 3D objects such as spheres and arrows
- How to use vectors in VPython

TIME

You should finish this part of the lab in 50 minutes.

OVERVIEW OF A COMPUTER PROGRAM

A computer program consists of a sequence of instructions.

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 will print a red error message.

A typical program has four sections:

- Setup statements
- Definitions of constants
- Creation of objects
- Calculations to predict motion or move objects (these may be repeated)

I. Using IDLE to create a program

Find “IDLE” or maybe “VIDLE” on the desktop or in PROGRAMS and open it. This is a text editor where you will write and run your programs.

1. Starting a program: Setup statements

Enter the following two statements in the IDLE editor window:

```
from __future__ import division
from visual import *
```

Every VPython program begins with these setup statements.

The first statement (from *space* underscore underscore *future* underscore underscore *space* import *space* *) tells the Python language to treat 1/2 as 0.5. Without the first statement, the Python programming language does integer division with truncation and 1/2 is zero!

The second statement tells the program to use the 3D module (called “visual”).

Before we write any more, let’s save the program:

- **In the IDLE editor, from the “File” menu, select “Save.” Browse to a location where you can save the file, and give it the name “vectors.py”. YOU MUST TYPE the “.py” file extension --IDLE will NOT automatically add it. Without the “.py” file extension IDLE won’t colorize your program statements in a helpful way.**

* This activity is based on a similar activity written by Ruth Chabay and Bruce Sherwood.

2. Creating an object

In the next section of your program you will create 3D objects:

- Now tell VPython to make a sphere. On the next line, type:

```
sphere ()
```

This statement tells the computer to create a sphere object. Run the program by pressing F5 on the keyboard. Two new windows appear in addition to the editing window. One of them is the 3-D graphics window, which now contains a sphere.

3. The 3-D graphics scene

By default the sphere is at the center of the scene, and the “camera” (that is, your point of view) is looking directly at the center.

- Hold down both mouse buttons and move the mouse forward and backward to make the camera move closer or farther away from the center of the scene. (On a Mac, hold down the option key while moving the mouse forward and backward.)
- 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 Mac, in order to rotate the view, hold down the Command key while you click and drag the mouse.)

When you first run the program, the coordinate system has the positive x direction to the right, the positive y direction pointing up, toward the top edge of the screen, and the positive z direction coming out of the screen toward you. You can then rotate the camera view to make these axes point in other directions.

4. The Python Shell window is important -- Error messages appear here

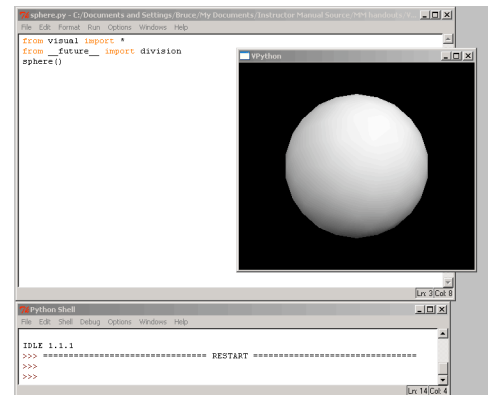
IMPORTANT: Arrange the windows on your screen so the Shell window is always visible.

DO NOT CLOSE THE SHELL WINDOW.

KILL the program by closing only the graphic display window.

The second new window that opened when you ran the program is the Python Shell window. If you include lines in the program that tell the computer to print text, the text will appear in this window.

- Use the mouse to make the Python Shell window smaller, and move it to the lower part of the screen. Keep it open when you are writing and running programs so you can easily spot error messages, which appear in this window.
- Make your edit window small enough that you can see both the edit window and the Python Shell window at all times.
- Do not expand the edit window to fill the whole screen. You will lose important information if you do!
- To kill the program, close only the graphics window. Do not close the Python Shell window.



5. Error messages: Making and fixing an error

To see an example of an error message, let's try making a spelling mistake:

- Change the third statement of the program to the following:

```
spherical ()
```

- Run the program.

There is no function or object in VPython called `spherical()`. As a result, you get an error message in red text in the Python Shell window. The message gives the filename, the line where the error occurred, and a description of the error. For example:

```
Traceback (most recent call last):
  File "/Users/atitus/Documents/courses/PHY221/presentations/chapter-
01/lab/vpython/vectors.py", line 4, in <module>
    spherical()
NameError: name 'spherical' is not defined
```

Read error messages from the bottom up: The bottom line contains the information about the location of the error.

- **Correct the error in the program.**

Whenever your program fails to run properly, look for a red error message in the Python Shell window.

Even if you don't understand the error message, it is important to be able to see it, in order to find out that there is an error in your code. This helps you distinguish between a typing or coding mistake, and a program that runs correctly but does something other than what you intended.

6. Changing "attributes" (position, size, color, shape, etc.) of an object:

Now let's give the sphere a different position in space and a radius.

- **Change the last line of the program to the following:**

```
sphere(pos=vector(-5,2,+3), radius=0.40, color=color.red)
```

- **Run the program. Experiment with other changes to `pos`, `radius`, and `color`, running the program each time you change something. Find answers to the following questions:**

QUESTIONS TO ANSWER ABOUT SPHERES:

What does changing the `pos` attribute of a sphere do?

What does changing the `radius` attribute of a sphere do?

What does changing the `color` attribute of a sphere do? What colors can you use? (Note: you can try `color=(1,0.5,0)` for example. The numbers stand for RGB (Red, Green, Blue) and can have values between 0 and 1. Can you make an orange sphere?

7. Autoscaling and units

VPython automatically *zooms* the camera in or out so that all objects appear in the window. Because of this *autoscaling*, the numbers for the `pos` and `radius` can be in any consistent set of units, like meters, centimeters, inches, etc. For example, this could represent a sphere with a radius 0.20 m and a position vector of $\langle 2, 4, 0 \rangle$ m. In this course we will always use SI units in our programs ("Systeme International", the system of units based on meters, kilograms, and seconds).

8. Creating an arrow object

We often use `arrow` objects in VPython to depict vector quantities. We will next add arrows to our program.

- **Type the following on a new line, then run the program:**

```
arrow(pos=vector(0,0,0), axis=vector(5,0,0), color=color.white, shaftwidth=0.1)
```

- **This arrow can be used to visualize the x-axis, for example.**
- **Experiment with other changes to `pos`, `axis`, and `color`, running the program each time you change something. Change one thing at a time so you can tell what effect your changes have. For example, try making a second arrow with the same `pos` and a different color, but in the +y direction. Also, try pointing the arrow in the +z direction.**
- **Find answers to the following questions:**

QUESTIONS TO ANSWER ABOUT ARROWS:

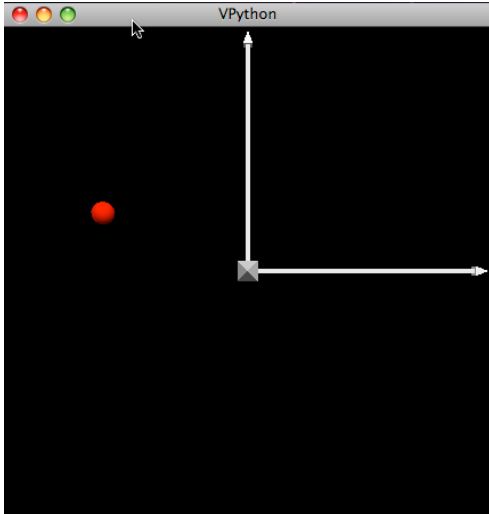
Is the `pos` of an arrow the location of its tail, its tip, its middle, or somewhere else?

Is the `axis` of an arrow the position of its tail, the position of its tip, or the position of the tip relative to the tail (that is, a vector pointing from tip to tail)?

- **Create a Cartesian coordinate system by creating three arrows that point in the +x, +y, and +z direction.**

Just copy and paste the line above and change its `axis` to point in the +y direction. Repeat for a z-axis. Your program should produce something like the picture below, showing both the sphere and the three arrows for the Cartesian coordinate system. Rotate it around and view it from various perspectives.

- **Create an arrow that represents the position of the sphere and points from the origin to the location of the sphere.**



9. Comment lines (lines ignored by the computer)

Comment lines start with a # (pound sign).

A comment line can be a note to yourself, such as:

```
# units are meters
```

Or a comment can be used to remove a line of code temporarily, without erasing it.

- **Put a # at the beginning of the line creating your arrow for the position vector of the object, and run the program:**

```
#arrow(pos=vector(0,0,0), axis=vector(-5,2,3), color=color.cyan)
```

Note the pound sign at the beginning of the line. The pound sign lets VPython know that anything after it is just a comment, not actual instructions. The statement will be skipped when the program is run, so the arrow will not be created.

- **Run the program. Explain what you see.**
- **Uncomment that line and run the program again. The arrow now appears.**

10. Naming objects; Using object names and attributes

We will show the position vector for a tennis ball that automatically changes if the tennis ball is moved.

- **Clean up your program so it contains only the following objects:**

A Cartesian coordinate system with x, y, and z axes that are each 5 m long.

A green sphere (representing a tennis ball) at location $\langle -3, 1, 2 \rangle$, with radius 0.2.

An arrow representing the position of the tennis ball that points from the origin to the position of the tennis ball.

- **Run your program and verify that it looks as expected.**
- **Change the initial position of the tennis ball to $\langle 0, 2, -3 \rangle$**

- **Run the program.**
- **Note that the arrow still points in its original direction. We want this arrow to always point toward the tennis ball, no matter what initial position we give the tennis ball. To do this, we will have to refer to the tennis ball’s position symbolically, by its name. But first, we must give the tennis ball a name.**
- **Give a name to the spheres by changing the “sphere” statement in the program to the following:**

```
tennisball = sphere(pos=vector(0,2,-3), radius=0.2, color=color.green)
```

We’ve now given a name to the sphere. We can use this name later in the program to refer to the sphere. Furthermore, we can specifically refer to the attributes of the sphere by writing, for example, `tennisball.pos` to refer to the tennis ball’s position attribute, or `tennisball.color` to refer to the tennis ball’s color attribute. To see how this works, do the following exercise.

- **Start a new line at the end of your program and type:**

```
print tennisball.pos
```

- **Run the program.**
- **Look at the text output window. The printed vector should be the same as the tennis ball’s position.**
- **Run the program to see if the printed position matches the position of the tennis ball given in the program.**

Let’s change the axis of the arrow to be the position of the tennis ball.

- **Modify the arrow so that its axis refers to the tennis ball’s position symbolically, not numerically.**
- **Run the program to see if the arrow points from the origin to the tennis ball.**
- **Change the position of the tennis ball to something like (-2,-4,1) and run the program. The arrow should point to the tennis ball regardless of the position of the tennis ball.**

11. Multiplying by a scalar: Scaling an arrow’s axis

Since position of a sphere is a vector, we can perform scalar multiplication on it. Start with the tennis ball at (2,0,0):

```
tennisball = sphere(pos=vector(2,0,0), radius=0.2, color=color.green)
```

- **Modify the position of the tennis ball by changing the statement to the following, and note what happens:**

```
tennisball = sphere(pos=2*vector(2,0,0), radius=0.2, color=color.green)
```

QUESTIONS TO ANSWER ABOUT SCALING ARROWS:

To move the tennis ball three times further on the x-axis, what scalar would you multiply its position by?

To move the tennis ball to the other side of the origin by multiplying by a scalar factor, what factor should you use?

12. Turn in your program on WebAssign

There is a WebAssign assignment where you should turn in your final program. When you turn in a program to WebAssign, be sure to follow the instructions given there, which will ask you to change some of the parameters in your program.

13. Using VPython outside of class

You can download VPython from <http://vpython.org> and install it on your own computer.

14. Reference manual and programming help

There is an on-line reference manual for VPython. In the text editor (IDLE), on the Help menu choose “Visual.”