Lab 1: Introduction to Modeling in vPython

Goals

In this lab, you will learn:

- How to use VIDLE, a program editor for vPython computer programming language.
- How to structure a computer program for this lab.
- How to create, control, and name 3D objects such as spheres, boxes, arrows and cylinders.
- Learn how to create programming comments within VIDLE.
- Learn to use vectors in vPython.
- Learn how to print to the screen.
- Learn how to control the view in vPython.

Computer Program Overview

Throughout Phys 172, we will use the computer language **vPython** to write a series of computer programs that will help us model physical phenomena. A couple of things to remember about computer programs:

- A computer program is a set of instructions that tell the computer what to do.
- The instructions must be written in a very specific manner (called a *computer language*), such that the computer can read them.
- The computer language we will use to write our instructions is called vPython.
- The computer will follow the instructions sequentially until one of two things happen:
  - The end of the program is reached.
  - The computer encounters an error in the instructions. When this happens, the computer will print a red error message and the line in the instructions where the program stopped.

We will write our computer instructions (called *code*) in a vPython editor called **VIDLE**. VIDLE can be found on all ITaP computers. Both VIDLE and vPython are freeware, and can be downloaded to a personal computer from http://VPython.org and installed on either a PC or Mac. We recommend version 3.2, as this is most similar to the version running in the lab.

To access VIDLE from the lab or from an ITaP computer, go to start->course software->science->physics->PHYS172->VIDLE

Writing a Program: the first two lines

We will begin every one of our computer codes with the same two statements. These two statements must be entered ***exactly*** as shown below (VIDLE is case sensitive).

Enter the following two lines exactly into the VIDLE editor window:

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

The first statement (from *space underscore underscore* future *underscore underscore space* import *space* division) tells the computer to allow division to have a decimal answer, as the default setting in Python is an integer quotient. Thus, this line tells the computer to treat 5/2 as 2.5 rather than 2.

The second line (from *space* visual *space* import *space asterisk*) tells the computer to import the visual module of vPython, which allows us to create and manipulate 3D objects.

Saving a Program

Before we can run a vPython program, we must first save the program. To do this, go to the *File* menu of the VIDLE editor, and select *Save*. Select a location where you would like to save your program, and give it a name. **The name must end in .py.** This lets the computer know that the file is a python code, and allows the VIDLE editor to color the code meaningfully. The *.py* extention will not be added automatically, and so you must be sure to add it yourself. Save the file as *intro.py* and click *Save*.

3D Objects and Attributes

In this lab, we will create visual models using four different 3D shapes that are found in vPython: sphere, box, arrow, and cylinder.

To tell vPython to create a sphere, add the following to your code:

```
sphere()
```

Run the code by going to the *Run* menu and selecting ***Run Module***, or you can just press the **F5** key.

Two windows will pop up. The first one is the Python Shell window, where error messages will appear. The second window should contain a white sphere, which is filling the scene. In this window, our 3D models will be visualized. The center of the scene is located at position (0,0,0) by default, and the position of the sphere is (0,0,0) by default.

Let us now change the position of the sphere. To kill the program, close the visualizer window. However, **do not close the Python Shell window**, as it is important. Instead, rearrange the windows on your screen such that the Python Shell window is always visible.

Now change the line of code with the sphere() command to

```
sphere(pos=vector(-4,0,0), radius=0.5,
color=color.red)
```

1. **Before running the program, write down what you think this line of code will do.**

Now run the program (*Run module* in the *Run* menu, or hit the F5 key).

2. **Did the program behave as you expected? Why or Why not?**

You should now see that the sphere is in a different location, appears smaller than before, and is now a different color. These three different qualities of the object are called *attributes*, and will appear with some default setting unless otherwise specified. To change an object's attributes, you must label the attribute, and then specify what you want it to be with an equals sign. The order of the attributes does not matter as long as they are separated by a comma.

- The **Position Attribute** (specified by pos) is inherently a vector quantity. Therefore we must specify that the coordinates describing the position are a vector using the vector(x,y,z) command.

- The **Radius Attribute** (specified by radius) allows us to change the size of the sphere from its default setting of 1 to any specified value (in the same units as position.)

- The **Color Attribute** (specified by color) can be easily changed to one of four colors: white, black, red, magenta, yellow, green, blue, and cyan. To do this, just set the color attribute to color.xx, where xx is the desired color.

Kill the program (again by closing the visualizer widow, but not the Python Shell window), and let us add a box to our code.

Do this by adding the following line of code to our program below the line creating the sphere:

```
box()
```

We can specify the box's attributes in the same way that we specify the attributes of the sphere.

3. **What would you type to create a box with the following attributes?**

Position	Color
(1,2,0)	cyan

Run the program to see if the box had the specified attributes. Notice that the box is a perfect cube.

Boxes have an additional attribute, called the **Size Attribute**, which specifies the dimensions of the box as an ordered triplet: (x length, y length, z length). Although the size of the box is not a vector, vPython treats it as one, since the vector construct is already an ordered triplet. If the size

attribute is not changed, then the box defaults to a 1x1x1 cube (size=(1,1,1)). To create a box with the following attributes you need to add the Size Attribute.

Position	Color	Size
(1,2,0)	cyan	(2,4,1)

To add the Size Attribute, add the following to the line of code creating the box:
```
size=(2,4,1)
```
Run the program.

Notice that the left side of the box lies along the vertical, centerline of the viewer (the y-axis), and that the bottom side of the box lies along the horizontal centerline of the viewer (the x-axis.)

Remember that the position of the box describes the location of the *center* of the box, while the size of the box describes the *entire* lengths of the dimensions of the box. Therefore, the location of the left side of the box is given by the x-coordinate of the box's position *minus* half of the size of the box in the x dimension. Similarly, the right side of the box is given by the x-coordinate of the box's position *plus* one half of the size of the box in the x dimension.

4. How does one determine the location of the top and bottom sides of the box given the box's attributes?

CHECKPOINT 1: Raise your hand and ask your instructor to check your work. You may proceed while you wait.

Naming Objects

We have now created two objects, and in principle we could create many more. Therefore it is a good idea to name the objects. Naming the objects will also allow us to change their attributes later, which is vital for our modeling. Let us name the sphere Ringo and the box Tia.

To name an object, you type the name of the object ***immediately before*** the object to be named, followed by an equals sign:
```
ringo = sphere(pos=vector(-4,0,0), radius=0.5,
color=color.red)
tia = box(pos=vector(1,2,0), color=color.cyan,
size=(2,4,1))
```


You can now refer to the attributes of the objects by referring to the name of the object followed by a period, and then specifying the attribute.

Example:

To refer to the radius of the sphere, one may type:

```
ringo.radius
```

If we wished to change the radius of the sphere to 2, then we may type

```
ringo.radius=2
```

To change an attribute, for example, the x-position of tia:

1) `tia.pos.x=-2`

or

2) `tia.pos=vector(-2,2,0)`

These two methods have the same effect. However, we will use the second method most often.

Commenting the Code

We have thus far written a program consisting of the following four lines of code. Check to make sure that your program looks the same.

```
from __future__ import division
from visual import *

ringo = sphere(pos=vector(-4,0,0), radius=0.5, color=color.red)
tia = box(pos=vector(1,2,0), color=color.cyan, size=(2,4,1))
```

While this code is still rather simple, other codes will become much more complex. For this reason, we will be commenting our code, such that we know what each line of code is supposed to be doing. **You are required to comment each part of your code!**

To create a comment line, first type a ***pound sign, #,*** (also known as a *hash*), and then type your comment. The # tells the computer to ignore the rest of that line. VIDLE changes the colors of comment lines to red, to make them stand out.

A rule of thumb for commenting code: Write comments often and with sufficient detail, so that someone who has never seen your code before can understand what each line does.

- Add a line to the very top of the code (before the first two lines of code) specifying the name of the code, and what it does

  ```
  # intro.py
  # A code for introducing the features of vPython
  ```

- Add a comment to your code in the line *above* the creation of the sphere (our first object) to label that section of code as the place where we will create our objects:

  ```
  # Objects in Simulation
  ```

Your code should now look like this:

```
# intro.py
# A code for introducing the features of vPython

from __future__ import division
from visual import *

# Objects in Simulation
ringo = sphere(pos=vector(-4,0,0), radius=0.5, color=color.red)
tia = box(pos=vector(1,2,0), color=color.cyan, size=(2,4,1))
```

Scalar and Vector Quantities

Python, and by extension vPython, natively works with scalar quantities (a magnitude with no associated direction) and with vector quantities (a magnitude with an associated direction).

To create and define a **scalar** quantity: type the desired name of the quantity followed by and equals sign, and then the value of the scalar.

To create and define a **vector** quantity, specify the desired name of the vector followed by an equals sign, and then the vector(x,y,z) operator, which lets Python know that the quantity is a vector quantity. Be sure to break the vector into its components.

- To practice defining vector and scalar quantities, insert a new section of code *between* the line importing the visual module (from visual import *) and the comment labeling the section creating the objects (# Objects in Simulation). We will call this new section "Object Parameters":

```
# Object Parameters
ringo_size = 0.5
tia_location = vector(1,2,0)
```

We can now set other scalar and vector quantities equal to these two quantities. Change the radius attribute of the sphere "ringo" to equal *ringo_size*, and change the position attribute of the box "tia" to equal *tia.location*. Your code should now look like:

```
# intro.py
# A code for introducing the features of vPython

from __future__ import division
from visual import *

# Object Parameters
ringo_size = 0.5
tia_location = vector(1,2,0)
```

```
# Objects in Simulation
ringo = sphere(pos=vector(-4,0,0), radius=ringo_size,
color=color.red)
tia = box(pos=tia_location, color=color.cyan, size=(2,4,1))
```

Notice that we can specify attributes either with a numerical value, or with a variable of the same type (scalar or vector). We will be using this construct a lot.

Remember: Scalar quantities can only be equated with other scalar quantities, and vector quantities can only be equated with other vector quantities!

CHECKPOINT 2: Raise your hand and ask your instructor to check your work.

Arrows and Cylinders

The last two 3D objects that we will be using are arrows (arrow()) and cylinders (cylinder()). These two objects both have the **position (pos)** and **color (color) attributes** that spheres and boxes have. However, instead of describing the location of the center of the object, the position attribute describes the location of the *tail end* of the arrow and one of the ends of the cylinder.

To specify the location of the other end of the object (either the head of the arrow or the other end of the cylinder), these two objects have another attribute called **axis**. The axis of the object defines the location of the other end of the object *with respect to the position attribute*. In other words, the axis attribute specifies the three component dimensions of the arrow or cylinder.

To see this in action, let us create a new object (an arrow) called "point" with the following attributes:

Position	Axis	Color	Shaftwidth
(-2,0,0)	(2,0,0)	Green	0.2

The **shaftwidth attribute** specifies the width of the arrow.

Below the box called "tia", create the arrow with the given attributes by typing:

```
point=arrow(pos=vector(-2,0,0), axis=vector(2,0,0),
color=color.green, shaftwidth=0.2)
```

and run the program.

5. **Which way does the arrow point?**

6. **How would you change the attributes of the arrow such that the location of the endpoint of the arrow stays in the same place, but the arrow points in the opposite direction?**

7. **Try to change the direction of the arrow. Run the program to see if it worked.**

- To reverse the arrow, the location of the tail (specified by the pos attribute) needs to be located where the head was (specified by the sum of the pos and axis attributes) [new_pos = old_pos + old_axis = (-2,0,0) + (2,0,0) = (0,0,0)]
- The new head [new_pos + new_axis] needs to be located where the old tail was [old_pos = (-2,0,0) -> new_pos = (0,0,0) + new_axis = (-2,0,0) -> new_axis = (-2,0,0)]. Thus, the line creating the arrow should read:

```
point=arrow(pos=vector(0,0,0), axis=vector(-2,0,0),
color=color.green, shaftwidth=0.2)
```

Now let us place the arrow on the ball, and have it point three units down, and three units to the right. To do this, we could either change the position attribute of the arrow in the line creating the arrow, or we could do it on a new line as follows:

```
point.pos = ringo.pos
point.axis = vector(3,-3,0)
```

- Use a comment to label this section "Executions." Run the program.

When the program first starts up it will have the original position and axis attributes specified in the line creating the object. However, the computer moves through the instructions so quickly, that it appears to immediately change the position and axis of the arrow to those specified. The ability to change attributes of objects in later instructions to the computer will be vital in animating our models.

- Lastly, let us create a cylinder called "rod" with the following attributes:

Position	Axis	Color	Radius
(1,-3,2)	(3,3,-4)	yellow	0.2

- Be sure to place this object in the "Objects in Simulation" section of your code.

The cylinder has a radius attribute, which specifies the radius of the circular cross section of the cylinder. This is in contrast to the shaftwidth attribute of the arrow, which gives the total external dimension of the thickness of the arrow.

Your code should now look like:

```
# intro.py
```



```
# A code for introducing the features of vPython

from __future__ import division
from visual import *

# Object Parameters
ringo_size = 0.5
tia_location = vector(1,2,0)

# Objects in Simulation
ringo = sphere(pos=vector(-4,0,0), radius=ringo_size,
color=color.red)
tia = box(pos=tia_location, color=color.cyan,
size=(2,4,1))
point=arrow(pos=vector(0,0,0), axis=vector(-2,0,0),
color=color.green, shaftwidth=0.2)
rod = cylinder(pos=vector(1,-3,2), axis=vector(3,3,-4),
color=color.yellow, radius=0.2)

#Executions
point.pos = ringo.pos
point.axis = vector(3,-3,0)
```

The print statement

To have the computer let us know the value of something, we use the print command. Let us tell the computer to print off the position of the arrow "point". In the executions section of your code, add the line

```
print(ringo.pos)
```

Run the program. You should see that the position of the sphere has been outputted to the *Python Shell* window. Since it is a vector, it is shown inside of <> brackets.

However, this does not identify the quantity, and we may wish to print multiple quantities to the screen. Therefore, we can label the output in the *Python Shell* window by modifying the print command to

print("ringo.pos=", ringo.pos)

Run the program, and you will now see that the output in the *Python Shell* window is clearly labeled.

Controlling the Visualizer Window

Lastly, let us talk briefly about controlling the view in the visualizer window. If your program is not already running, run the program and navigate to the visualizer window. There are two things that we can do: rotate the view, and zoom in and out.

- To ***rotate*** the camera view, hold down the right mouse button and drag the mouse around (or two-finger click on a Macintosh notebook computer). You will see that the view rotates around the center of the view

- To ***zoom in*** **or** ***out***, hold down the left and right mouse buttons together (or hold down *option* and *left click* on a macintosh) and drag the mouse up and down. The scene should zoom in and out, while staying centered on the origin.

CHECKPOINT 3: Raise your hand and ask your instructor to check your work.

Debugging

When the computer encounters an error in the code, the program stops, and a red error message is printed to the *Python Shell* window. This message usually contains some information on the error, including where in the code the program was stopped. Often it is because there is a *syntax error*, where the required format was not followed, or a parenthesis near where the program stopped is missing.

Watch the following short video on debugging syntax errors for a brief overview:

http://www.youtube.com/watch?v=kqFdB0lbGa8&feature=plcp

Additionally, if you desire a review of creating 3D objects, watch the following short video:

http://www.youtube.com/watch?v=KbOyKOlWBrs&feature=plcp

Lastly, the following video provides a short overview on variable assignment:

http://www.youtube.com/watch?v=jLHS0ZvYE5Y&feature=plcp

