VPython Review

OBJECTIVES

In this lab, you will:

- Learn the basic structure of a computer program
- Create objects in VPython
- Display vector quantities in 3D
- Animate the motion of 3D objects
- Construct a graph in VPython

In this lab, we will review the use of VPython by using the concept of electric fields. From last semester, recall that VPython is a simple, programming language which can:

Display objects
Display vectors representing physical quantities
Animate object motion
Simulate physical interactions
Create graphs
Output results from calculations.

1) The 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. The syntax of the instructions must be exact. If the computer encounters an error in an instruction, it will stop running and print a red error message.

A typical program has five sections:

- Setup statements
- Definitions of constants
- Setting initial conditions
- Creation of objects
- Calculations to predict motion or move objects (these may be repeated using loops**)**

2) Opening and Setting Up VPython

VPython is installed on all these lab computers. It is also on all ITAP computers.

a) Open the Python editor, IDLE, through Start -> All Programs -> Course Software -> science -> phys -> Python 2.4 -> IDLE for VPython. This procedure may be different in your lab. Your TA will tell you if it is different.

You can also open programs by right clicking on them and selecting "Edit with IDLE," but this will cause your IDLE window to close if you close the display window, so avoid this method.

b) Enter the following lines of code in the IDLE editor window.

```
from __future__ import division
from visual import *
from visual.graph import *
```

Every VPython program begins with the first two lines, the third line is optional. The first line tells the Python language to treat 1/2 as 0.5. Without the first statement, the Python programming language does integer division with truncation i.e. 1/2 is zero! The second line tells the program to use the 3D module (called "visual"), which allows us to easily draw many 3D objects. The third line allows VPython to output graphs. You will not need this line in all your programs, but there is no harm in including it.
If you are using a graph (you will be in this first program), you may wish to include the following after the above three lines.

```
scene = display(x=0, y=0, width=600, height = 600)
graph = gdisplay(x=600, y=0, width=400, height=300)
```

This positions the display and graph window so that they do not block each other.

Now VPython is ready to receive instructions.

3) Commenting

You can use comments to make your code easier to read. A comment is a statement that you can see on the screen but the computer will not interpret it as an instruction. VPython knows words are comments if they are preceded by a pound sign (#). The # may be at the start of a line in order to comment out the entire line, or it may be midline in order to comment out a description of what the line of code does.

a) Type the following.

```
# Define Constants

# Set Initial Values

# Create Objects

# Calculations
```

These are the titles of the four sections of your code. We will not tell you where to comment your code, but you should at least comment the titles of each section. Some programs have comments after each line of code!

4) Constants

You will need to use physical constants in many of the VPython calculation. Rather than entering the number several times, it is easier to give the physical constant a name, i.e. set a series of memorable letters equal to the number. Then, for example, when you need to enter the charge of an electron you can type "-e" instead of "-1.6e-19".

We will enter two constants in this first program. Enter them in the define constants section.

b) Enter the charge of a proton and give it the name "e" by typing:

```
e = 1.6e-19
```

c) Enter the value of $\frac{1}{4\pi\epsilon_0}$. Name that value "oofpez."

```
oofpez = 9e9
```

You will need to add to the list of constants as the semester continues.

Notice you did not give the values units. Like your handheld calculator, VPython does not use units. It is your responsibility to keep track of the units of the numbers you put into to VPython, and the units of the numbers VPython calculates for you. To make your programming easier, you should always use SI units.

5) Creating Objects

You do not need any initial values yet so you can skip to create objects section. This program will have three objects, a positively charged atom, an arrow representing the electric field caused by this atom at an observation location, and a graph of distance vs. electric field. You should place these lines of code in the create objects section.

a) Create a sphere to represent a proton by typing:

```
atom=sphere(pos=vector(-10e-10,0,0), radius=1e-10, color=color.red)
```

This creates a sphere named "atom" with three characteristics. It is located at (-10 x 10^{-10}, 0, 0) m, it has a radius of 1 x 10^{-10} m (the approximate radius of atoms), and it is red.

b) Give the atom the equivalent charge of 10 excess protons.

```
atom.q = 10*e
```

c) Create an arrow by typing:

```
Earrow=arrow(pos=(0,5e-10,0), axis=(0,0,0), color=color.orange)
```

This creates an arrow named "Earrow" with its tail at (0, 5 x 10^{-10}, 0) m (the observation location), with zero magnitude (axis is set to be (0,0,0)), and which is orange.

d) Create a graph by typing:

```
Egraph=gcurve(color=color.cyan)
```

This creates a graph named "Egraph" which will have a cyan line.

e) Run the code by hitting F5.

You should see the sphere. There is no arrow because it has zero magnitude.

You can rotate the view of the sphere by holding down the right click button and moving the mouse. You can also zoom in an out by holding down both the right and left click buttons and moving the mouse.

6) Calculations

Do the following in the calculations section. Now you will calculate the electric field at an observation location. You will do this with Coulomb's law.

$$\vec{E}_{net} = \frac{1}{4\pi\varepsilon_0}\frac{q}{r^2}\hat{r}$$

First you will need to find the r vector. You can make VPython do the calculations for you.

a) Define the r vector by typing:

```
r = Earrow.pos – atom.pos
```

This line tells VPython to define the letter r as a vector pointing from the position of your atom to your observation location (the position of the tail of the Earrow arrow). The ".pos" after the object name tells VPython you are referring to the position of that object. You can use this syntax with most object attributes, e.g. you can type "Earrow.axis" to denote the axis of the Earrow arrow.

b) Now find the magnitude of r. You can do this by taking the square root of the sum of the squares, or by typing:

```
rmag = mag(r)
```

This line finds the magnitude of r with "mag(r)" then sets it equal to "magr."

c) Find the r hat vector by typing:

```
rhat = r / rmag
```

This divides the vector by its magnitude to find its unit vector.

d) Now use Coulomb's law to find the electric field by typing:

```
E = oofpez * (atom.q / rmag**2) * rhat
```

VPython uses "**" to denote powers instead of the carrot (^) you may be used to.

e) With the above equation and given values, calculate the vector electric field E.

f) Have VPython display the E field by typing:

```
print ("E =", E, "N/C")
```

This will make VPython print the words "E =," followed by the value of the electric field, followed by "N/C" (the electric field's units). Does this value match with what you calculated above?

7) Arrows and Scale Factors

Now you will make the arrow to represent the E field.

a) Type the following after the electric field calculation.

```
Earrow.axis = E
```

This changes the value of the axis attribute off the Earrow to be equal to the E vector.

b) Run the code (F5).

You will now see that your sphere has disappeared off the screen and only the arrow is present. This is because the magnitude of the electric field is about 20 orders of magnitude larger than the radius of the sphere (just considering the number VPython uses). These two quantities, however, do not have the same units, thus there is no sense in comparing them. This frees you to scale down the magnitude of the electric field arrow so that you can see both the atom and Earrow.

To do this scaling you will define a **scale factor**. You can then multiply the axis of the arrow by the scale factor to reduce its length. Provided we use the same scale factor each time we are dealing with electric fields, this trick will not corrupt the physical relations in our simulation.

To decide an appropriate value of a scale factor, consider that we want the length of the E field vector to be about the same order of magnitude as the atom's radius. So

$$\frac{\text{atom.radius}}{\left|\vec{E}\right|} = \text{scalefactor}$$

Do not enter this as code. Instead

c) Perform the above calculation and place a line of code, after the definitions of constants, defining the scale factor to be the number you just calculated.

```
scalefactor = 8.68e-21
```

The scale factor needs to be a number and not defined in terms of other variables. Those variables may change while the program is running causing different arrows to have different scale factors, and that would corrupt our physical interpretation.

d) Change the line of code you entered in Step 7. a. to:

```
Earrow.axis = scalefactor * E
```

When you run the program you should now see both the atom and the arrow. If you think the arrow is too small or large you can adjust the scale factor until it is visually pleasing.

8) Particle Motion

In order to make the particle move across the screen you will need to use a loop. A loop is an instruction that tells VPython to perform a series of tasks repeatedly until the program meets some criterion. You will create a clock in your program which "ticks" (steps forward one time interval) each time VPython performs the series of tasks in the loop. This clock will create a form of time which you can use to create motion.

a) Set the clock to zero by typing the following in your initial values section.

```
t = 0
```

b) In your constants section, define a time interval, how much time passes every time the clock "ticks," with the following line.

```
deltat = 1e-13
```

c) For the atom to move it needs a velocity. In your create objects section, under the atom.q line, type:

```
atom.v = vector(1,0,0)
```

This gives the atom a velocity of 1 m/s in the positive x direction.

The choice of time interval is a tradeoff between how accurate you want your program to be and how fast you want it to run. In our case the particle will move one thousandth of its radius during each time interval, this seems reasonable for the purposes of visualization. If you feel your program is going too quickly, you can always shrink the time interval, however, if you feel your program is running too slowly you can only extend the time interval up to a certain point before you start affecting the accuracy of the program.

None of the code you just entered will cause the particle to move. These steps are simply defining the tools you will use to make it move.

d) Enter the following as the first line in your calculations section.

```
while atom.pos.x < 1e-9:
```

This is an instruction for VPython to keep carrying out the instructions until the criterion that the x-coordinate of the atom's position is greater than 1×10^{-9} m (a reasonable distance to watch our atom travel). VPython knows which instructions to keep carrying out because those instructions come immediately after the above line of code and are all indented. Once VPython encounters code which is not indented, it goes back to the start of the loop, checks if the while statement is still true, and begins to perform the tasks in the loop again. You may also note that you are using the dot-syntax twice to not just denote the position of the atom, but even more specifically, its x-coordinate.

e) Highlight the reminder of the program (everything following the line of code you just entered) and click "Format" then click "Indent Region".

Remember, if VPython encounters code which is not indented it will know that it has reached the end of the loop and return to the top.

f) At the bottom of your code enter the following lines. Make sure they are indented!

```
atom.pos=atom.pos+deltat*atom.v
Egraph.plot(pos=(rmag,mag(E)))
t=t+deltat
```

The first line is the position update formula. This line instructs VPython to take the old position of the atom, add the distance the atom has traveled during that time interval, and make the resulting value the new position of the atom.

The second line tells VPython to put a point on the graph corresponding to the magnitude of the electric field given the current magnitude of the r vector.

The third line updates the time. This last line is not strictly necessary because the program does not make use of the quantity t, but many future programs will.

g) At the bottom of your create objects section enter the following line of code.

```
scene.autoscale=0
```

The line turns off auto scaling. This is a feature which causes VPython to rescale the display window in order to keep objects in view or give a closer view, however it can make it difficult to understand what is happening. You should only enter this line after you have created all the objects you want to see.

h) Finally before you run the program place a # sign in front of the print statement.

Having print statements in the loop will drastically slow down your program.

You should now see the atom glide across your screen and the arrow change as the electric field changes at the observation location. You should also see a graph which corresponds to the r^{-2} relation in Coulomb's law.

9) Play

Experiment with the program. Try changing colors, the atom's velocity, what the graph displays, the length of a tick, the condition in the loop, etc.

Have Fun!!!

Checkpoint: Ask an instructor to check your work for credit.