Introduction to VPython

This tutorial will guide you through the basics of programming in VPython.

VPython is a programming language that allows you to easily make 3-D graphics and animations. We will use it extensively in this course to model physical systems. First we will introduce how to create simple 3-D objects. Then we will use VPython to explore vectors and vector operations in 3-D.

On the screen desktop there should be an icon called "IDLE for Python" (if not, ask your instructor where to find IDLE). Double click the IDLE icon. This starts IDLE, which is the editing environment for VPython.

1. Starting a program

- **Enter the following line of code in the IDLE editor window.**

```
from visual import *
```

Every VPython program begins with this line. This line 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.**

2. Creating a sphere

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

```
sphere()
```

This line 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.

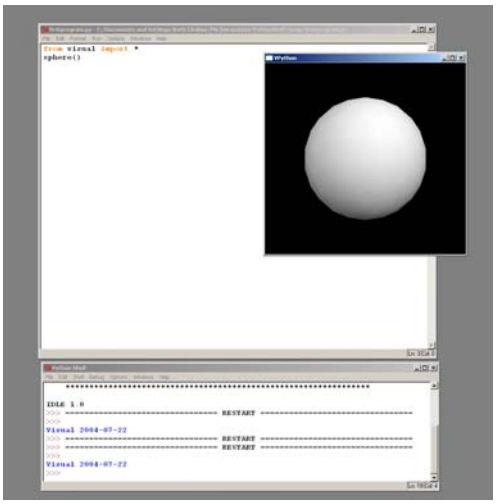
- **Hold down both buttons and move the mouse up and down to make the camera move closer or farther away from the center of the scene.**
- **Hold down the right mouse button alone and move the mouse to make the camera "revolve" around the scene, while always looking at the center.**

When you first run the program, the coordinate system has the positive x direction to the right, the positive y direction pointing up, 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

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 the graphics window. Do not close the Python Shell window.**



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

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

```
from bisual import *
```

- **Run the program.**

Notice you get a 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 (in this case "ImportError: No module named bisual").

- **Correct the error in the first line.**

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.

5. Attributes

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.**

The sphere-creation statement gives the sphere object three "attributes":

1.) *pos*: the position vector of the center of the sphere, relative to the origin at the center of the screen
2.) *radius*: the sphere's radius
3.) *color*: the sphere's color. Color values are written as "color.xxx", where xxx could be red, blue, green, cyan, magenta, yellow, orange, black, or white.

- **Change the last line to read:**

```
sphere(pos=vector(2,4,0), radius=0.20, color=color.white)
```

Note the changes in the sphere's position, radius, and color.

Experiment with different values for the attributes of the sphere. Try giving the sphere other position vectors. Try giving it different values for "radius." Run the program each time you make a change to see the results. When you are done, reset the line to how it appears above (that is, pos=vector(2,4,0), and radius=0.20).

6. 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" could 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 < 2, 4, 0 > 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).

7. Creating a second object

- **We can tell the program to create additional objects. Type the following on a new line, then run the program:**

```
sphere(pos=vector(-3,-1,0), radius=0.15, color=color.green)
```

You should now see the original white sphere and a new green sphere. In later exercises, the white sphere will represent a baseball and the green sphere will represent a tennis ball. (The radii are exaggerated for visibility.)

8. Arrows

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

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

```
arrow(pos=vector(2,-3,0), axis=vector(3,4,0), color=color.cyan)
```

This line tells VPython to create an arrow object with 3 attributes:
1.) *pos*: the position vector of the *tail* of the arrow. In this case, the tail of the arrow is at the position < 2, -3 , 0 > m.
2.) *axis*: the components of the arrow vector; that is, the vector measured from the tail to the tip of the arrow. In this case, the arrow vector is < 3, 4, 0 > m.
3.) *color*: the arrow's color.

To demonstrate the difference between "pos" and "axis," let's make a second arrow with a different "pos" but same "axis."

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

```
arrow(pos=vector(3,2,0), axis=vector(3,4,0), color=color.red)
```

Note the red arrow starts at a different point than the cyan arrow, but has the same magnitude and direction. This is because they have the same "axis," but different values of "pos."

Question: What position would you give a sphere so that it would appear at the *tip* of the red arrow? Discuss this with your partner. Then check the answer at the end of this tutorial.

9. Scaling an arrow's axis

Since the axis of an arrow is a vector, we can perform scalar multiplication on it.

- **Modify the axis of the red arrow by changing the last line of the program to the following:**

```
arrow(pos=vector(3,2,0), axis=-0.5*vector(3,4,0), color=color.red)
```

Run the program. The axis of the red arrow is now equal to -0.5 times the axis of the cyan arrow. This means that the red arrow now points in the opposite direction of the cyan arrow and is half as long. Multiplying an axis vector by a scalar will change the length of the arrow, because it changes the magnitude of the axis vector. The arrow will point in the same direction if the scalar is positive, and in the opposite direction if the scalar is negative.

10. Comments (lines ignored by the computer)

For the next section, we will only need one arrow. Let's make VPython ignore one of the "arrow" lines in the program.

- **Change the second to last line (the *cyan* arrow) to the following:**

```
#arrow(pos=vector(2,-3,0), axis=vector(3,4,0), 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 line will be skipped when the program is run.

- **Run the program. There should now only be one arrow on the screen.**

11. Arrows and position vectors

We can use arrows to represent position vectors and relative position vectors. Remember that a relative position vector that starts at a position $\vec{A}$ and ends at a position $\vec{B}$ can be found by "final minus initial," or $\vec{B}-\vec{A}$. Do the following exercise:

We want to make an arrow represent the relative position vector of the tennis ball with respect to the baseball. That is, the arrow's tail should be at the position of the baseball (the *white* sphere), and the tip should be at the position of the tennis ball (the *green* sphere).

- **What would be the "pos" of this arrow, whose tail is on the baseball (the white sphere)?**
- **What would be the "axis" of this arrow, so that the tip is on the tennis ball (the green sphere)?**
- **Using these values of "pos" and "axis", change the last line of the program to make the red arrow point from the white baseball to the green tennis ball.**
- **Run the program.**
- Self check**: Examine the 3D display carefully. If the red arrow does not point from the white baseball to the green tennis ball, correct your program.**

12. Naming objects and using object attributes

- **Now change the position of the tennis ball (the second, green sphere in the program)--imagine it now has a z-component, so that the line would now be:**

```
sphere(pos=vector(-3,-1,3.5), radius=0.15, color=color.green)
```

- **Run the program.**

- **Note that the relative position arrow still points in its original direction. We want this arrow to always point toward the tennis ball, no matter what position we give the tennis ball. To do this, we will have to refer to the tennis ball's position symbolically. But first, since there is more than one sphere and we need to tell them apart, we need to give the objects names.**

- **Give names to the spheres by changing the "sphere" lines of the program to the following:**

```
baseball = sphere(pos=vector(2,4,0), radius=0.20, color=color.white)
tennisball = sphere(pos=vector(-3,-1,3.5), radius=0.15, color=color.green)
```

We've now given names to the spheres. We can use these names later in the program to refer to each sphere individually. Furthermore, we can specifically refer to the attributes of each object by writing, for example, "tennisball.pos" to refer to the tennis ball's position attribute, or "baseball.color" to refer to the baseball'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.

Let's also give a name to the arrow.

- **Edit the last line of the program (the red arrow) to the following, to give the arrow a name:**

```
bt = arrow(pos=vector(3,2,0), axis=-0.5*vector(3,4,0), color=color.red)
```

Since we can refer to the attributes of objects symbolically, we want to write symbolic expressions for the "axis" and "pos" of the arrow "bt". The expressions should use general attribute names in symbolic form, like "tennisball.pos" and "baseball.pos", not specific numerical vector values such as vector(-3,-1,0). This way, if the positions of the tennis ball or baseball are changed, the arrow will still point from baseball to tennis ball.

- In symbols (letters, not numbers), what should be the "pos" of the red arrow that points from the baseball to the tennis ball? **Make sure that your expression doesn't contain any numbers.**

- In symbols (letters, not numbers), what should be the "axis" of the red arrow that points from the baseball to the tennis ball? (Remember that a relative position vector that starts at position $\vec{A}$ and ends at position $\vec{B}$ can be found by "final minus initial," or $\vec{B} - \vec{A}$.). **Make sure that your expression doesn't contain any numbers.**

- Change the last line of the program so that the arrow statement uses these symbolic expressions for "pos" and "axis".

- Run the program. Examine the 3D display closely to make sure that the red arrow still points from the baseball to the tennis ball. If it doesn't, correct your program, still using no numbers.

- Change the "pos" of the baseball to (-4, -2, 5). Change the "pos" of the tennis ball to (3, 1, -2). Run the program. Examine the 3D display closely to make sure that the red arrow still points from the baseball to the tennis ball. If it doesn't, correct your program, still using no numbers.

CHECKPOINT 1: Ask an instructor to check your program for credit.
You can read ahead while you're waiting to be checked off.

13. Creating a static model

Be sure you have saved your old program, vectors.py.

Start a new program by going to the "File" menu and selecting "New window." Again, the first line to type in this new window is:

```
from visual import *
```

From the "File" menu, select "Save." Browse to a location where you can save the file, and give it the name "planets.py". YOU MUST TYPE the ".py" file extension; IDLE will NOT automatically add it.

The program you will write makes a model of the Sun and various planets. The distances are given in scientific notation. In VPython, to write numbers in scientific notation, use the letter "e"; for example, the number 2×10^7 is written as **`2e7`** in a VPython program.

Create a model of the Sun and three of the inner planets: Mercury, Venus, and Earth. The distances from the Sun to each of the planets are given by the following:

Mercury: 5.8×10^{10} m from the sun
Venus: 1.1×10^{11} m from the sun
Earth: 1.5×10^{11} m from the sun

The inner planets all orbit the sun in roughly the same plane, so place them in the x-y plane. Place the Sun at the origin, Mercury at $< 5.8\times10^{10}$, 0, 0 >, Venus at $< -1.1\times10^{11}$, 0, 0 >, and Earth at $< 0, 1.5\times10^{11}$, 0 >.

If you use the real radii of the Sun and the planets in your model, they will be too small for you to see in the empty vastness of the Solar System! So use these values:

Radius of Sun: 7.0×10^9 m
Radius of Mercury: 4×10^9 m
Radius of Venus: 6.0×10^9 m
Radius of Earth: 6.4×10^9 m

The radius of the Sun in this program is ten times larger than the real radius, while the radii of the planets in this program are about 1000 times larger than the real radii.

Give names to the objects: Sun, Mercury, Venus, and Earth, so that you can refer to their attributes.

Finally create two arrows ***using symbolic values*** for the "pos" and "axis" attributes (***no numerical data***):

1.) Create an arrow representing the relative position of Mercury with respect to the Earth. That is, the arrow's tail should be on Earth, and the arrow's tip should be on Mercury. Give the arrow a name, "a1".
2.) Imagine that a space probe is on its way to Venus, and that it is currently halfway between Earth and Venus. Create a relative position vector that points from the Earth to the current position of the probe. Give the arrow a name, "a2".

Remember: Do not use numerical data to specify the arrow attributes.

CHECKPOINT 2: Ask an instructor to check your program for credit.
You can read ahead while you're waiting to be checked off.

14. Loops and motion

Another programming concept we will use in the course is a loop. A loop is a set of instructions in a program that are repeated over and over until some condition is met. There are several ways to create a loop, but usually in this course we will use the "while" statement to make loops.

Let's try using a loop to repeatedly add to a quantity and print out the current value of the quantity.

- **Add the following statement at the end of your planets program:**

```
step = 0
```

This tells the program to create a variable called “step” and assign it the value of 0.

- **On the next line, type:**

```
while step<100:
```

- **Press the “Enter” key. Notice that the cursor is now indented on the next line. (If it’s not indented, check to see if you typed the colon at the end of the “while” line. If not, go back and add the colon, then press “Enter” again.)**

The “while” statement tells the computer to repeat certain instructions while a certain condition is true. The lines that will be repeated are the ones that are indented after the “while” statement. In this case, the loop will continue as long as the variable “step” is less than 100.

- **On the next (indented) line, type:**

```
   step = step+1
```

In algebra, “step = step+1” would be an incorrect statement, but in VPython as in most programming languages, the equals sign means something different than it does in algebra. In VPython, the equals sign is used for *assignment*, not equality. That is, the line *assigns* the variable step a *new* value, which is the *current* value of step plus 1. This means that the first time through the loop, the computer adds the current value of step, which is 0, to 1, giving the value 1, and then assigns step this new value of 1. The next time through the loop, the computer again adds 1 to step, making step equal to 2, and so on: 3, 4, 5, ….98, 99, 100.

- **To show this, on the next line (still indented), type:**

```
   print step
```

The last four lines you typed should now look like this:

```
step = 0
while step<100:
   step = step+1
   print step
```

- **Run the program.**

In the text output window, you should see a list of numbers from 1 to 100 in increments of 1. The first number,1, is the value of step at the end of the first time through the loop. Before each execution of the loop, the computer compares the current value of step to 100, and if it is less than 100, it executes the loop again. After the 100th time, the value of step is now 100. When the computer goes back to the “while” statement for the next repetition, it finds the statement “step<100” is now false, since 100 is not less than itself. Because the condition is false, the computer does not do any more executions of the loop.

To go back to writing statements that are not repeated in a loop, simply unindent by pressing the “Backspace” key.

- **Type the following on a new, unindented, line:**

```
print "End of program, step=", step
```

Now the last five lines in your program should look like this:

```
step = 0
while step<100:
   step = step+1
   print step
print "End of program, step=", step
```

- **Run the program.**

You’ll now see the sequence of numbers printed, followed by the text “End of program., step=100” (the while loop ended because step had become equal to 100). The line that prints this text is not in the loop, so the text prints only once, after the loop is done executing.

Now we'll make Mercury move. It will move in a very unphysical way, but later you will learn how to program the actual motion of stars and planets.

- **Before the start of your while loop, insert this statement:**

```
deltar = vector(1e9,0,0)
```

This defines a vector increment $\Delta\vec{r}$ of the position of Mercury. We'll continually add this small vector "displacement" to the position of Mercury, which will make it move across the screen. The variable deltar is a vector, just like Earth.pos or a1.axis, but it is purely calculational and there is no visible display such as a sphere or an arrow associated with deltar.

- **Inside your while loop (indented), insert this statement (assuming your sphere is named Mercury):**

```
Mercury.pos = Mercury.pos+deltar
```

- **Run the program.**

You should see Mercury move across the screen, because you are continually updating its position by adding a small vector displacement to its current position. The planet may move quite fast, depending on how fast your computer is.

- **To slow your program down, add the following statement inside your while loop (indented):**

```
rate(20)
```

This statement says, "Don't do more than 20 loop iterations per second, even if the computer is fast enough to do more." Since you're taking 100 steps in your program, Mercury will now take 5 seconds to move across the screen.

- **Make the arrow "a1" point from the Earth to Mercury at all times, while Mercury moves. Insert an appropriate statement into the while loop (indented) to update the arrow "a1" so that its tail remains on the Earth but its tip is always on Mercury. Don't create any new arrows; just update attributes of your existing arrow "a1".**

Your Mercury has been moving to the right, in the +x direction.

- **Finally, change deltar in such a way that Mercury moves northeast, that is, up and to the right at a 45 degree angle to the horizontal. The tip of the arrow pointing from the Earth to Mercury must remain on Mercury during the motion.**

CHECKPOINT 3: Ask an instructor to check your program for credit.
You can read ahead while you're waiting to be checked off.

15. Turn in your program to WebAssign

There is a WebAssign assignment where you turn in your final program, planets.py. *When you turn in a program to WebAssign, be sure to follow the instructions given there, which may sometimes ask you to change some of the parameters in your program.* Each person that worked on the program should turn it in. Take turns logging into WebAssign and uploading the program. If you are not yet registered in WebAssign, make sure that you keep a copy of the program to submit later.

16. Using VPython outside of class

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

17. Programming help

There is an on-line reference manual for VPython. In the text editor (IDLE), on the Help menu choose "Visual". The first link, "Reference manual", gives detailed information on spheres, arrows, etc. In the text editor (IDLE), on the Help menu choose "Python Docs" to obtain detailed information on the Python programming language upon which VPython is based. We will use only a small subset of Python's extensive capabilities.

Answer to question in tutorial:
The tip of the red arrow is located at *pos* + *axis* = < 3, 2, 0 > + < 3, 4, 0 > = < 6, 6, 0 >.