VPYTHON: 3D PROGRAMMING FOR ORDINARY MORTALS

Bruce Sherwood, Ruth Chabay, *North Carolina State University, Raleigh, North Carolina, USA*

Abstract

VPython is a programming environment that enables even novices to write programs that produce navigable real-time 3D animations. One to two hours of carefully crafted instruction is sufficient to bring novice students to the point of being able to do serious computer modeling (http://www.matterandinteractions.org). VPython is based on the Python programming language (http://python.org) which has a large user community. Like Python, VPython is open source freeware available for Windows, Linux, and Macintosh (http://vpython.org). Workshop participants will learn to write 3D programs.

1. Python and VPython

VPython is a programming environment that enables even novices to write programs that produce navigable real-time 3D animations. The new version 5 supports transparency, local as well as distant lights, and material displays such as wood or marble on all platforms, and for the first time runs as a native application on the Mac. Students in introductory physics courses use VPython to write programs to model physical systems and to visualize electric and magnetic fields (http://www.matterandinteractions.org; Chabay 2007; Chabay 2008). Educators use VPython to create lecture-demonstration programs. Researchers use VPython to model systems and to visualize data in 3D.

One to two hours of carefully crafted instruction is sufficient to bring novice students to the point of being able to do serious computer modeling. VPython is based on the Python programming language (http://python.org) which has a large user community. Like Python, VPython is open source freeware available for Windows, Linux, and Macintosh (http://vpython.org). A set of VPython computational modeling activities for the introductory physics course taken by engineering and science students is available at http://www.compadre.org/psrc/items/detail.cfm?ID=5692. Fig. 1 shows an example of a VPython program that models the 3D motion of a mass hanging from a spring, written by a student in an introductory mechanics course.

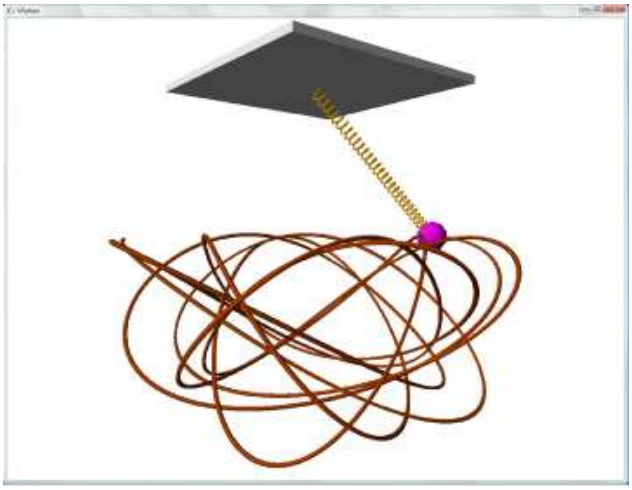

Figure 1: A student VPython program written in an introductory mechanics course

Python is a clean language in which it is easy to do simple things right from the start, with little excess overhead. For example, the complete Python version of the classic “Hello World” program in Python looks like this, with no extraneous distracters:

```
print “Hello World”
```

Nor is there the distraction of extraneous setup or make procedures. After installing Python, one can invoke the built-in IDLE integrated development environment, type the one-line program shown above, and press F5 to see it run. The simplicity and immediacy of this environment has led to some introductory programming courses using Python as a first language.

While it is very easy to do simple things in Python, it also supports sophisticated object-oriented programming, and there is an unusually gentle slope leading to creating and using classes. There is a sizable community using Python for scientific programming (http://scipy.org), and a significant fraction of Google programming is done in Python.

VPython is the combination of Python plus an extension module called "visual". VPython builds on the Python tradition of cleanliness and simplicity by making it possible to do simple things in 3D very easily. To display a sphere on the screen, all that one needs to do is to invoke the visual module and create a sphere:

```
from visual import *
sphere()
```

Enter this text into an IDLE edit window and press F5. You will see a 3D sphere, and using the mouse you can rotate the camera around the scene and zoom in and out.

There are many useful defaults built into VPython. The simple statement "sphere()" triggers the creation of a graphics window containing an OpenGL context, and the statement produces by default a white sphere centered at the origin, of radius 1, with the "camera" automatically positioned to look toward the origin from a distance which makes the sphere fill the window. Defaults can be overridden. For example, the following statement creates an orange sphere with the appearance of wood, with radius 0.5 and 30% opacity, with its center located at x=2, y=-1, z=1:

```
sphere(pos=(2,-1,1), color=color.orange, radius=0.5, opacity=0.3, material=materials.wood)
```

The default autoscaling means that a program showing the orbit of the Earth around the Sun, with a scale of 1e11 m, and a program showing alpha scattering off a gold nucleus (Rutherford scattering) with a scale of 1e-13 m both make displays in which the scene automatically fills the graphics window.

The inventory of graphics objects currently supported by VPython includes sphere, arrow, box, cone, convex, curve (an array of connected points), cylinder, ellipsoid, faces (an array of triangles), helix, label (to display text), points (an array of disconnected points), pyramid, ring, and frame (for grouping objects together). One can create distant and local lights. There are statements for creating autoscaling graphs, controls (buttons, toggles, sliders, menus), and a platform-independent file dialog box. VPython supports handling of mouse and keyboard events. A single statement suffices to make the 3D scene truly three-dimensional, to be viewed with red-cyan glasses or, with two projectors aimed at a metallic screen, with polarized 3D glasses.

Of great importance for scientific work is support for vectors and vector operations such as sum, difference, dot product, cross product, magnitude, etc. Students writing VPython programs to model physical systems such as gravitational orbits write coordinate-free vector statements to calculate forces, update momenta, and update positions. This is the first environment within which students have had an opportunity to use and experience vectors as powerful unitary entities as opposed to mere sines and cosines.

2. Animations

Here is a complete program that calculates and displays the orbit of the Earth orbiting the Sun (the radii of Sun and Earth have been exaggerated for display purposes):

```
from visual import *
G = 6.7e-11
Sun = sphere(pos=(0,0,0), color=color.yellow, radius=2e10)
Earth = sphere(pos=(1.5e11,0,0), color=color.cyan, radius=6e9)
Sun.m = 2e30
Earth.m = 6e24
Earth.p = Earth.m*vector(0,3e4,0)
dt = 100
while True:
    r = Earth.pos - Sun.pos
    rmag = mag(r)
    rhat = r/rmag
    Fmag = G*Sun.m*Earth.m/rmag**2
    Fnet = -Fmag*rhat
    Earth.p = Earth.p + Fnet*dt
    Earth.pos = Earth.pos + (Earth.p/Earth.m)*dt
```

After entering these statements into an IDLE editor window, press F5 and an animated orbit appears with the Earth circling the Sun. With the mouse one can view the scene from different angles and different distances.

The program starts by importing all of the visual capabilities, including the standard math library. The Sun and Earth are represented by out-size spheres so as to be visible in the nearly empty Solar System, and they are positioned at an appropriate distance apart. The Sun's mass is specified, as are the Earth's mass and initial momentum. A time step for the numerical integration is chosen to be 100 seconds. The statement "while True:" means that the indented code under the while statement will run forever, until the user closes the graphics window.

The statement "r = Earth.pos - Sun.pos" is a vector subtraction of the current position of the Earth minus the position of the Sun, so that r represents a vector pointing from the Sun to the Earth. The magnitude of r and the corresponding unit vector ("rhat") are calculated and used to calculate the magnitude of the gravitational force and the vector force, which is used to update the momentum. Then the position of the Earth is updated.

Pedagogically it is beneficial that a student who writes such a program is working in a coordinate-free way (except for setting initial conditions), thinking of and using vectors as powerful objects rather than as mere sines and cosines. It appears that students gain a much more sophisticated concept of "vector" as a result of such computational modeling experiences.

Simple additions to this program include appending points to a curve object to display the trajectory, and updating arrow objects to display force and momentum vectors.

3. How does it work?

It is surprising that in the while loop of the orbit program shown above there seem to be no graphics output statements that would produce an animation, only computational statements to predict the motion. The key is that every time through the loop the pos (position) attribute of the sphere object representing the Earth is updated. Approximately 30 times per second, these computations are interrupted by a separate "rendering" thread which creates a new image based on the current attributes of the existing objects. The rendering thread clears a region of memory and for each existing object (in this case, two spheres) uses OpenGL to add that object to the image, *using the current attributes of those objects*. The effect is that each time the rendering thread creates an image of the Sun and Earth, the Earth is in a new position, and as long as the rendering thread can run about 30 times every second, the animation will look smooth to the human viewer. When completely rendered, the scene is handed to the graphics card to display the image on the screen, and the rendering thread returns control to the computational thread. The rendering process also takes into account mouse moves for positioning the camera, which makes the scene navigable.

The effect is to make navigable 3D animations a side effect of computations, thereby making feasible the creation of sophisticated displays by programming novices. This was a key invention of David Scherer in the spring of 2000, when he created VPython while an undergraduate student at Carnegie Mellon University.

4. Acknowledgements

These developments were supported in part by the National Science Foundation through Grant Nos. DUE-0320608, DUE-0237132, and DUE-0618504.

References

Chabay R and Sherwood B (2007) Matter & Interactions, John Wiley & Sons, Hoboken NJ, USA

Chabay R and Sherwood B (2008) Computational physics in the introductory calculus-based course, American Journal of Physics, 76(4&5), 307-313