ADVENTURES IN 3D MODELING USING VPYTHON

Russell Herman and Gabriel Lugo
University of North Carolina Wilmington
hermanr@uncw.edu and lugo@uncw.edu

In the summer of 2008 we taught mathematical modeling to a group of high school students as part of a month long program called Summer Ventures in Science and Mathematics (http://www.summerventures.org/). The idea was to quickly introduce the students to programming and creating simulations which not only aided in visualization, but also could be used to gather data for further analysis. The students would then explore a research topic armed with only a week introduction to mathematical modeling and software tools. In this paper we describe the process, the main software package, VPython, and some of the student projects that resulted[1].

Mathematical Modeling

We began the course by defining and highlighting key aspects of mathematical modeling and providing some interesting examples. We made the case that modeling is the process of representing a real-world phenomenon as a set of mathematical equations and is used in fields such as the natural sciences, engineering, social sciences, mathematics, and economics. Models can be classified in many ways: linear or nonlinear, deterministic or stochastic, continuous or discrete. The methods used are typically a combination of numerical computation, symbolic analysis, and visualization. The types of mathematics used ranges from high school through advanced mathematics (Algebra, Trigonometry, Graphing Functions, Calculus, Linear Algebra, Differential Equations).

Examples were provided throughout the program. These consisted of physics problems (projectile motion, harmonic motion, waves, the physics of sports, and planetary motion), problems in mathematics (complex numbers, random numbers, Fourier analysis, and fractals), and biological models (population dynamics and epidemic models). We also explored the programming of additional problems such as percolation, Monte Carlo methods and cellular automata.

Mathematical Background

The background of the students was that typical of bright high school students from around North Carolina. Only one had seen some calculus, most had basic algebra and geometry, and several have taken trigonometry. So, we needed to think about how get students to solve systems of differential equations without using calculus or physics. Since the end result would be programming a discretization of such continuous models, we only needed simple tools, such as slopes of secant lines, to express rates of change.

[1] Further details can be found at http://people.uncw.edu/hermanr/SVSM.htm.

The afternoon of the first day we worked on the second modeling problem – exponential population models. We introduced population models as a way to introduce the analysis of data and doing an exponential fit to the data based on a derivation of the solution to a discrete (difference) population model. With a little prodding, we lead students through the derivation of a discrete model and the continuous solution as a limit (done in Maple) as schematically shown in Figure 1. Then we used MS Excel to test the model with actual U.S. population data. Afterwards, the logistic model was introduced and explored for several populations in the U.S. and other countries.

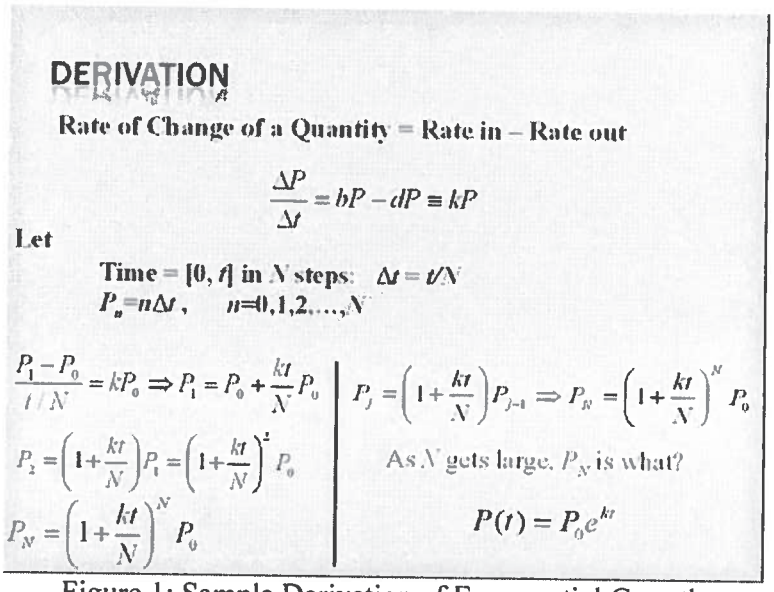


Figure 1: Sample Derivation of Exponential Growth

The second day students were introduced to programming with the goal of modeling projectile motion based upon an approximate solution of $\frac{d^2\mathbf{r}}{dt^2} = \text{constant}$, which can be rewritten as a system of two first order differential equations: $\frac{d\mathbf{r}}{dt} = \mathbf{v}, \frac{d\mathbf{v}}{dt} = \text{constant}$. Of course, they did not know that this is what they were doing. Instead, we lead them through rates of change using difference quotients, starting from motion with uniform velocities in one dimension, $v_{ave} = \frac{\Delta x}{\Delta t} = \frac{x_{new} - x_{old}}{\Delta t}$. After a simple rewriting, one has the beginning of the prediction of the motion of an object, like a ball, as $x_{new} = x_{old} + v_{ave}\Delta t$.

Introduction of VPython for 3D Simulations

The next step in the process was to program and visualize the motion. For this we introduced the students to just enough programming to simulate the motion. We needed a simple to use, cheap, and attention grabbing programming environment. We chose to introduce them to VPython, an open source, Python-based, modeling program.

At the VPython site, http://vpython.org/, the authors call it "3D Programming for Ordinary Mortals". Well, with a name like that, why not try it on students with practically no background in physics, calculus, or programming? VPython was co-authored by David Scherer, David Andersen, Jonathan Brandmeyer, Ruth Chabay, Ari Heitner, Ian Peters, and Bruce Sherwood for introductory physics instruction. It was first developed while some of the authors were at Carnegie Mellon and later used to teach physics at NC State University. It has recently been updated to Version 5 based upon community feedback. However, we had used the previous VPython 3.0 version.

It is relatively easy to install. The students installed it within a few minutes of using it. One first installs Python and then VPython. Then, run the IDLE icon. In this editor, one first types

from visual import *

On the next line type

sphere()

Then save the file (**CTRL-S**) as test.py and run it (**F5**). This brings up a sphere. Such objects like a sphere, cylinder, arrow, helix, can easily be created, resized, and moved. This makes doing simple 3D graphics straightforward. In fact, the students were making complicated objects in minutes. It is easy to change the object attributes. For the sphere, one needs only type **sphere(radius=0.5,color=color.red)** to make the sphere red and change its radius.

One can even give objects names, **ball = sphere(radius=0.5,color=color.red)**. This allows one to move the object, ball, by changing its position, which is given using vectors:

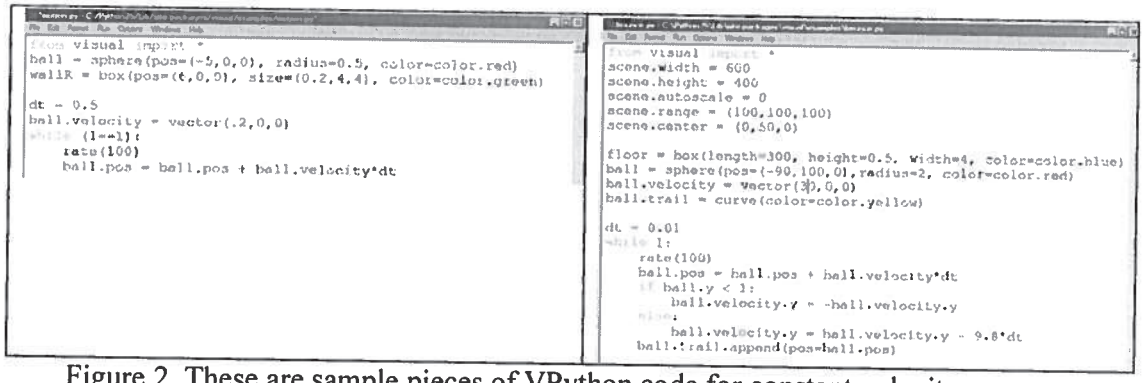
ball = sphere(pos=(0,2,0),radius=0.5,color=color.red)
ball.pos = (1,2,3)

Now we can program the ball to move on its own using the earlier result for changing position, $x_{new} = x_{old} + v_{ave}\Delta t$. First, we introduce the concept in programming that the equal sign means "replace". Letting x = ball.pos and v_{ave} = ball.velocity, the new ball position is given in terms of the old position by

ball.pos = ball.pos + ball.velocity*dt

However, we need to carry this out for several time steps. So, we then introduced looping using the while statement and a few other subtleties which we will not go into here. A sample of the code is shown in Figure 2. We then wanted to have the ball bounce from a wall. It turns out that the ball in the code on the left will not "see" the wall and passes right through it. If the ball moves too far to the right, then we can reverse its direction. This requires an additional programming structure, conditional statements. So, we add in

the loop **If ball.x > wallR.x**. Then what? We need to reverse direction: **ball.velocity = -ball.velocity**.



```
from visual import *
ball = sphere(pos=(-9,0,0), radius=0.5, color=color.red)
wallR = box(pos=(6,0,0), size=(0.2,4,4), color=color.green)

dt = 0.5
ball.velocity = vector(.2,0,0)
while (1==1):
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
```

```
from visual import *
scene.width = 600
scene.height = 400
scene.autoscale = 0
scene.range = (100,100,100)
scene.center = (0,50,0)

floor = box(length=300, height=0.5, width=4, color=color.blue)
ball = sphere(pos=(-90,100,0),radius=2, color=color.red)
ball.velocity = vector(30,0,0)
ball.trail = curve(color=color.yellow)

dt = 0.01
while 1:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
    ball.trail.append(pos=ball.pos)
```

Figure 2. These are sample pieces of VPython code for constant velocity and constant accelerated motion as described above.

We then add acceleration. We model dropping a ball that bounces on the floor. Here we introduce free fall and average velocity. This leads to adding velocity corrections.

new velocity = old velocity + a*dt
ball.velocity = ball.velocity + ball.acceleration*dt

Thus, we have simple model for solving a system of first order equations. Of course, a little numerical analysis is needed to say something about the accuracy. We later discuss this in class and show that other Python packages can be imported alongside VPython in order to do the equivalent of what expensive programs like MATLAB can do. We added packages useful for scientific computing such as NumPy, SciPy, Matplotlib and SymPy.

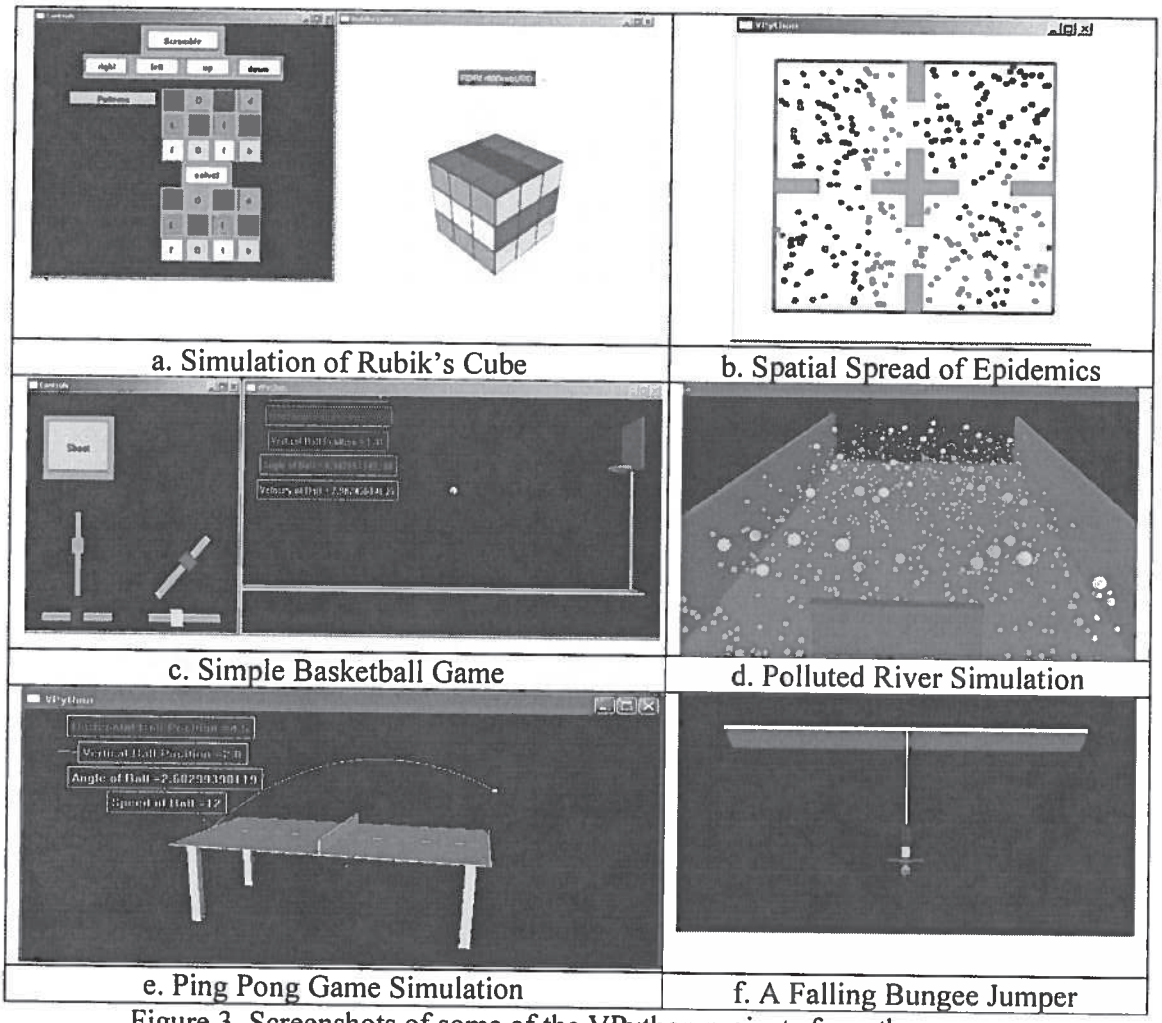
In the meantime, students were encouraged to add more walls and enclose the moving ball and allow for motion in 3D. They added more balls and soon were exploring other examples provided on the Internet or that came with the software. We learned how to create a mass on a spring and set it into motion. Once we had simple simulations, the object positions and velocities could be saved and later analyzed in other programs, such as MS Excel. It was now up to the students to begin to think about a research project.

Student Projects

For the rest of the week the students were exposed to other models using VPython, Python, and computer algebra systems in a variety of disciplines. They then were formed into groups and selected research topics. Some of the possible topics and the student presentations are currently at http://people.uncw.edu/hermanr/SVSM.htm.

Two groups explored projectile motion with drag and lift: ping pong ball and golf ball flight. One group looked into modeling river flow with the insertion and diffusion of pollutants. Another group modeled bungee jumping. Two other groups looked into epidem-

ics. One group strictly used Maple and MS Excel to analyze the bird flu data using a coupled system of bird and human populations. The other group did a sophisticated analysis of the spatial spread of epidemics in VPython, titled "Stochastic Spatial Dynamics of Epidemic Models". Finally, one pair of students was so obsessed with Rubik's cube, that they wanted to understand how to solve the cube. So, they learned about permutation groups, starting simple with colored balls, progressing to the rigid motions of a triangle, and then a subgroup of the permutation of the faces of Rubik's cube. Based upon this, they modeled simulations of Rubik's cube in VPython and worked out the subgroup and verified the table entries. Some screenshots of these projects are shown below.



a. Simulation of Rubik's Cube | b. Spatial Spread of Epidemics

c. Simple Basketball Game | d. Polluted River Simulation

e. Ping Pong Game Simulation | f. A Falling Bungee Jumper

Figure 3. Screenshots of some of the VPython projects from the course.

Conclusion

We found it was easy to use VPython to get the students exploring complicated systems in a relatively short period of time. This four week introduction to 3D Modeling and visualization can also be used throughout the undergraduate mathematics curriculum.