3-D Programing in VPython

Modeling a ball thrown in the air:

1. Creating a simple vpython program. In the editor, type:

```
from visual import *
```

This tells python to include the visual modules. You can also include "from __future__ import division" – this uses a better method for division than the standard python.

On the next line type :

```
sphere()
```

This calls the sphere function with no parameters and makes a default sphere. Run your program by pressing F5 or going to the "Run" menu. It will ask you to save.

2. Create ball and ground.

```
ball=sphere(pos=(0,0.1, 0), radius=0.1, color=color.red)
floor=box(pos=(0,0,0), size=(1,0.05,1),
color=color.green)
```

Run again to make sure there are no errors. You should see a ball and ground.

2. Input initial conditions

```
#set up initial conditions
ball.velocity=vector(0,5,0)
ball.mass=0.25
ball.p=ball.velocity*ball.mass
g=vector(0,-9.8,0)
Fnet=g*ball.mass
dt = 0.001
t = 0
```

Note: "#" means the following text is a comment. Here "g" is the gravitational field and is a vector.

3. The while loop

```
while t<4:
     rate(300)
     ball.pos=ball.pos + (ball.p/ball.mass)*dt
     ball.p = ball.p +Fnet*dt
     t = t + dt
```

run the program. NOTE: the stuff indented after the while loop is included in the loop.

4. Bouncing on the floor. You need to tell the program what to do if the ball hits the floor. Something like: “if the ball is lower than the floor, the momentum vector should change directions”

```
if ball.pos.y < (floor.pos.y + ball.radius):
     ball.p = -ball.p
```

5. Adding air resistance: Air resistance can be calculated as: $\vec{F}_{air} = -\frac{1}{2}\rho CAv^2\hat{v}$. So all that needs to be changed is to add an Fnet calculation in the while loop (this changes with speed)

in initial conditions, add:

```
c=.5
rho = 1.2
A = pi*ball.radius**2
Fnet = (g*ball.mass -
      .5*rho*c*A*mag(ball.p/ball.mass)**2*ball.p/mag(ball.p)
)
```

6. Adding a graph:
First, add the following to your beginning of the program:

```
from visual.graph import *
```

Then, before the while loop add:

```
posgraph = gcurve(color=color.green)
```

and in the while loop, add:

```
posgraph.plot(pos=(t, ball.pos.y))
```

this plots t and the y-component of the position of the ball.

7. Printing data:
To print the time and y position for each instance, simply add the following inside the while loop:

```
print t, "\t", ball.pos.y
```

The "\t" puts a tab between time and y position. This will print the values to the output window

8. Shooting at an angle.
Really, the only thing you need to change is the initial velocity. But you also need to change the bounce to `ball.p.y=-ball.p.y` (so only the y-momentum of the ball changes)

9. Adding a trail:
Before the while loop, add something like:

```
trail  = curve(color=color.white)
```

In the while loop, add:

```
trail.append(pos=ball.pos)
```

10. Arrows:
Arrows can be added to represent vectors. The arrow function has the following two important attributes: pos and axis. Pos is the position (vector) of the base. Axis is the vector from the base to the tip. To put a vector representing the momentum of the ball, first create the vector before the while loop:

```
pvector = arrow(pos=balll.pos, axis=ball.p)
```

If you leave it at this, it will not update, so you should also put this in the while loop:

```
pvector.pos=ball.pos
pvector.axis=ball.p
```

This gives a HUGE vector, you may want to scale it by saying scale=0.25 and `pvector.axis=scale*ball.p`