1 Simulation

Physics-based simulations serve various different purposes. They might be used to verify a model of a system by programming the model and then running the program and checking that the output matches reality. Once a model has been verified, it might be used to investigate situations that might be hard to create or observe in reality. Physical chemistry, for example, might simulate a large set of possible molecules looking some particular property and then only test in the lab molecules that showed promise in the simulation. Similarly, simulated models of an airplane might be tested in a simulated wind-tunnel before any prototypes are built. In both cases, the software, computer power and time to run the simulations turns out to be cheaper than building the real object and testing it. On the other hand, models are only as good as the input data, so we probably don't want to fly a modeled airplane with no testing at all. Another use is to provide realistic visuals for computer gaming and movies (for example the huge battle with the orcs in the first *Lord of the Rings* movie).

So how does one go about modeling the real world inside a computer? What follows is a brief introduction to the underpinnings of the physics modules of simulation systems. I've chosen to use VPython because Python is a relatively easy computer language to learn and because VPython was created by a couple of physics professors for teaching students physics by having them build models using the physics they were learning. There is a lot of physics and geometry that VPython will take care of for us: knowing what spheres, cylinders, cones, ...look like; projecting 3D information into 2D with appropriate prospective so we see the depth that isn't in the computer screen; lighting the scene and shading it so that curves look curved.

Note: I'm interested in how to describe physics to a computer rather than programming per se, so we'll mostly be working by modifying existing examples. However, you will have to absorb some programming concepts (like loops) to understand what is happening. Those of you with some programming experience - particularly those of you currently in a python-based programming class - will recognise many friends. Those of you with no programming experience should as lots of questions. I will elaborate these instructions as needed.

2 Getting Acquainted with VPython

- Go to Math_Science Apps then to VPython-Py3.2 then to Examples.
- Double-click on bounce.py to open it. You should have two windows open in an application called VIDLE (this is what is known as an Interactive Development Environment): Untitled and bounce.py. You can close the Untitled window with the red button in the upper left corner of the window.
- Go to the Run menu and click Run Module. A third window should open showing a red ball falling and then bouncing off a blue rectangle.

2.1 Manipulating the 3D rendering

- option-drag (hold down option and click-then-drag on mousepad) - zoom in (up on mousepad) or out (down on mousepad)
- command-drag (hold down command and click-then-drag on mousepad) - change viewpoint

Close the window with the bouncing ball (by clicking the red radio button in the top left). Run it again, but this time, pay close attention to what happens right at the beginning of the simulation. VPython has an autozoom feature: it tries to zoom in as much as possible and still have all the objects visible in the window and the origin at the middle of the window.

2.2 a few notes about reading VPython code

- Everything in VPython is in cartesian coordinates (x, y, z).
- The default orientation of the viewing window is positive x to the right, positive y to the top of the screen, and positive z coming out of the screen.
- The notation `ball.velocity` means the velocity of the ball and can be used either to give it a value or to use its value. Other properties of objects work similarly: `object.property`.

0 Why did the camera need to zoom out after the first bounce? (What would make a ball bounce higher than its starting point? Do you see anything between the `from` and the `while` lines of the code that would create such a situation?)

0 Now that you've found the cause of the small zoom-out, modify that part of the program to drop the ball from rest. You will need to Save As your file and save it to Documents or Desktop in order to run it. When you have something that works, copy and paste the code into your answers.

0 Now modify the program so the ball is thrown upward to start. When you have something that works, copy and paste the code into your answers.

0 Just to try a little more exploring of the first few lines of the VPython code, add a stationary green ball in front (from the standard view) of the red ball. When you have something that works, copy and paste the code into your answers.

3 1D Kinematics

3.1 more notes on reading VPython code

- Line 9 of the bounce.py code says `while 1:` and then the lines below it are indented. What this means is that while 1 is 1 (i.e. forever) the program will repeatedly pass through the indented block of code, it will loop. In this case, this lets the ball bounce forever.

- There are three instances here where syntax similar to `ball.pos = ball.pos + ball.velocity*dt` occurs. A simple statement of similar form would be `n=n+1`. This is NOT the same as a mathematical equation $n = n + 1$ which has no solution. Instead, this is an assignment. It is shorthand for "let the new value of n be one more than the current value of n" or "increase n by one". Similarly, `ball.pos = ball.pos + ball.velocity*dt` means "let the ball's new position equal the ball's old position plus the ball's velocity multiplied by time".

0 Translate
`ball.velocity.y = -ball.velocity.y`
and
`ball.velocity.y = ball.velocity.y - 9.8*dt`
into English.

0 What physical condition is the statement `if ball.y < 1:` checking for?

0 Is the model of a bouncing ball in bounce.py complete? If you dropped a real ball on a real surface, would it behave exactly like the ball in the simulation? If not, how does the real ball's behavior differ?

0 Suppose you dropped a ball of wet clay. What should it do when it gets to the "floor"? Change the code so this happens. When your code works, paste it into your answers.

0 Clay probably won't maintain its shape when it hits so make your ball of clay go splat. Remember that a sphere is a special case of an ellipsoid (or in 2D a circle is a special case of an ellipse). If you define your ball as

`ball=ellipsoid(pos=(0,4,0), color=color.red, height=1, width=1, length=1)`

then when the ball hits you can reset `ball.height` to squash the ball. When your code works (no hovering, splatted balls of clay), paste it into your answers.

0 Suppose you dropped an under-inflated soccer ball. What should it do when it gets to the "floor"? Change the code so this happens. When your code works, paste it into your answers.

0 Suppose we go to a different planet with a different gravitational acceleration near its surface. Modify the code to model the ball dropping on this planet. When your code works, paste it into your answers.

4 2D Kinematics

Now we get steadily more adventurous.

0 Launch the ball at an angle. Do you need to change the code inside the while loop to accurately imitate the ball? Why or why not? Is there anything else you need to fix? When your code works, paste it into your answers.

0 Apply a constant acceleration along the x-axis. When your code works, paste it into your answers.

0 Apply a position-dependent acceleration along the x-axis. (You may choose what component(s) the acceleration depends on, but they should affect only the x-component of the acceleration.) The x-coordinate of the ball is

`ball.x` or `ball.position.x`

When your code works, paste it into your answers.

0 Apply a velocity-dependent acceleration along the x-axis. (You may choose what component(s) the acceleration depends on, but they should affect only the x-component of the acceleration.) When your code works, paste it into your answers.

If we let these last four program run, VPython will keep zooming out to keep the ball on screen and eventually we won't be able to see anything anymore because we will be too far away. One possible solution would be to have the camera automatically pan to keep the ball centered on screen (rather than keeping the origin centered). Another solution is to add some walls to bounce the ball back to the center of its little universe. The latter is more interesting physics, so we'll do that.

0 Where should the walls be? Add code that will draw the walls and code to make the ball bounce off them. Be careful now that you have to check whether the ball has hit either of two walls that you don't update twice when the ball isn't hitting a wall. Below are a couple of useful options.

1. Check for both walls at once by using a compound condition: `if (ball.x<-1 or ball.x>1):`
2. Check for one wall, if ball isn't hitting that wall, check other wall, if it isn't hitting that wall either, update velocity using acceleration: use `if` for your first check, `elif` (else if) for your second

check, and then `else` for what happens if the ball isn't hitting one of the walls.

When your code works, paste it into your answers.

Suppose you wanted to keep a trace of where the ball had been so you don't have to remember. We can do this by adding `make_trail=True, trail_type="points", interval=10, retain=50` to the initial creation of the ball so

```
ball = sphere(pos=(0,4,0), color=color.red)
```

becomes

```
ball = sphere(pos=(0,4,0), color=color.red, make_trail=True, trail_type="points", interval=10, retain=50)
```

If you don't like the density of trail points or how many of them are kept change, respectively, the interval (at 10 this leaves a dot at every 10th position) and the retain values. You can change the color of the trail separately from the object leaving the trail by adding a line

`ball.trail_object.color=color.orange` to your code.

5 3D kinematics

0 Create a closed room and set your ball bouncing in three dimensions with gravity and the bounces as the only forces affecting the ball. Do this for ideal collisions. When your code works, paste it into your answers.

0 Create a closed room and set your ball bouncing in three dimensions with gravity and the bounces as the only forces affecting the ball. Have your ball lose energy with each collision. When your code works, paste it into your answers.

0 Create a closed room and set your ball bouncing in three dimensions with gravity and the bounces as the only forces affecting the ball. Have your ball lose energy with each collision and add a small drag force that points in the opposite direction from the velocity and is proportional to the velocity. When your code works, paste it into your answers.

6 Waves

We can simulate two sinusoidal waves traveling down a string in opposite directions. Create a new file with the following code in it. Make sure the indentation comes out right when you cut and past.

```
from __future__ import print_function, division
from visual import *

###Make a string out of dx-long segments
dx = 0.05
g = curve(x=arange(-10,10,dx), color=color.green)

scene.autoscale =0   ## Don't zoom in and out
wlth = 2.0
omega = pi
t = 0.0
dt = 0.005

#### add traveling waves
```

```
while 1:
rate(50)
for pt in g.pos:
#print (pt)
pt[1]=2.5*cos(omega*t-2*pi*pt[0]/wlth)+2.5*cos(omega*t+2*pi*pt[0]/wlth)
t = t+dt
```

0 What does the variable `wlth` represent?

Uncomment (delete the `#` from the beginning of the line) the `print` statement inside the `while` loop. Run the program. You will probably want to pause it fairly soon to stop and investigate the output. 0 By examining what gets spit out by the print statement, figure out what the variable `pt` represents and what `pt[0]` and `pt[1]` represent. (Remember that we are in a cartesian coordinate system.)

Comment the print statement back out (unless you need it for debugging).

0 What part of the program makes one wave move right and the other move left?

0 Make the amplitudes of the left- and right-moving waves differ. What happens to the total wave? Watch the edges carefully. Does this make sense?

0 Make the frequencies of the left-and right-moving waves differ. What phenomenon is this?

0 The original file has the wave oscillating in the xy-plane (x is along the string). Change one of the wave to the xz-plane. When your code works, paste it into your answers. Look at it from different angles. Does it do what you expected? How is this related to vector addition?

7 Ray Tracing

You may find the arrow object useful in these exercises.
`pointer = arrow(pos=(0,2,1), axis=(5,0,0), shaftwidth=1)`
The axis part of the command tells what direction the light ray is pointing.

0 Go back to your 3D bouncing ball demo and assume that the walls are perfect mirrors. Create a rectangular solid, also perfectly shiny, somewhere inside (not touching any walls) your box. Start with a single ray of light pointing any direction you like and track its path around the volume of the box. When your code works, paste it into your answers. Does the box end up equally lit on all sides? (This is your light ray, not the lighting provided by VPython?)

0 Now let the walls and the box be imperfectly mirrored so that some light is absorbed each time the light strikes an object. How will you show this? When your code works, paste it into your answers. Does the box end up equally lit on all sides? (This is your light ray, not the lighting provided by VPython.)