

# PHY307, Science and Computers I

## Lab #7, September 26, 2002 (due by end of lab Thursday)

### Cultivating Colorful Continuous Closed Curves using Coordinates

#### Summary of what you will do:

You will add another VPython object to your repertoire: the 3D curve. The **curve** object is good for plotting out paths in space and for plotting 3D data, like the weather data that Lorenz looked at in his simulations. We will continue this lab on Thursday in Part 2, looking at chaos using these curves. Save your work for today to hand in with Part 2 on Thursday.

#### FIRST, CURVES

1. Log in and start a report for Lab #7 – this is Part 1 – Part 2 will be done on Thursday.
2. In a fashion reminiscent of, but not identical to **gcurves** (the curves in plots), a **curve** object can be either created all at once *or* it can be grown and modified. First, you will draw a curve all at once.
  - a. Make a parabola all at once (you don't need to type in the comments):

```
from visual import *

xdata = arange(-1,1,0.01)      # Make array of 200
                                # numbers, -1 to 1.
ydata = xdata * xdata          # Makes an array of
                                # squares of items
                                # in zlist.
zdata = xdata * 0              # An array of zeros

curvea = curve(x = xdata, y = ydata, z = zdata)
```

SAVE A PICTURE of what you create. [NOTE: try **radius=0.02** as an argument to **curve**.]

- b. Make multiple curves. Instead of a single assignment, try this:

```
from visual import *

xdata = arange(-1,1,0.01)    # -1 to 1.
ydata = xdata * xdata        # Squares
zdata = xdata * 0

for zoffset in arange(-2,2,0.2)
    curve(x=xdata,y=ydata,z=zdata+zoffset)
```

SAVE A PICTURE of what you create AND EXPLAIN why this program does what it does.

3. Now, *grow* a curve. Like the **list** type of variable, the **curve** type allows you to append items – in this case a point position - to the end. (Save as **helix1.py** .)

```
from visual import
```

```
mycurve = curve(radius=0.02)
```

```
for t in arange(0,10,0.02):
```

```
    rate(40)
```

```
    helixpoint = (cos(t*3),sin(t*3),t)
```

```
    pointcolor = (1-t/10., t/10., 1)
```

```
    mycurve.append(pos=(cos(t*3),sin(t*3),t), color=pointcolor)
```

Run this and DESCRIBE what you see. Include a SNAPSHOT.

Now for some physics. Take the following code from the Codes section of the course page (**orbittype.py**):

```
## Oct. 1, 2002, A. Middleton
## orbittype.py
## This code simulates the motion of an object subject to a force
## from another object that is fixed at the origin (0,0,0).

from visual import *
scene.autoscale = 0

forcetype = -2

## draw object at origin in a suggestive yellow color
sphere(radius=0.2, color=color.yellow)

## this will be the flying object, suggestively named and
## colored
earth = sphere(radius = 0.1, color=color.blue, pos = (1,0,0))
## Give it a nonzero velocity, so that it will not plummet
earth.vel = vector(0,0.8,0)

dt = 0.01

while 1:
    rate(80)
    earth.pos = earth.pos + dt * earth.vel
    earth.vel = earth.vel + dt * (-earth.pos*mag(earth.pos)**forcetype)
    mycurve.append(pos=earth.pos)
```

Run this code – what do you see?

Add an empty **curve** named **mycurve**, before the **while** loop. In the **while** loop, append the position of the earth to the end of the **curve**.

Run this again – what do you see now?

Set **forcetype** to be **-1**. Does the trace of the orbit look different?

Changing **forcetype** changes how gravity depends on the separation of two objects. Some **forcetype** values give closed repeating curves, which is what we are used to in the planets.