

# PHY307, Science and Computers I

**Lab #10, October 23, 2002**  
**(due by end of lab TODAY)**

## Forging fractals, Fiddling with Frames

Summary of what you will do:

You will continue (from Lab #9) the process of generating fractals and computing their dimension.

### LAB REPORT

1. Get set up. Start a lab report entitled “*MyLastNameLab10.doc*”, where *MyLastName* is your actual last name. That way you can e-mail it as an attachment. You will also have graph paper plots that you will submit.

### FRAMES

In VPython, the **frame** object can be used to group objects into a single object. **frames** have **pos** and **axis** attributes that give their orientation to the frame in which they are defined. Rotations and translations of the frame move the associated objects as a rigid object. This is programmed by (a) creating a frame using the **frame()** function and (b) assigning or attaching objects to the **frame**. What you learn about frames here will be used to study natural looking fractals.

For example, the following creates a frame **f** and associates two spheres and a cube with that frame:

```
from visual import *
scene.autoscale = 0
f = frame()
cylinder(frame = f, radius=0.3, pos = (-2,0,0), axis=(4,0,0), color=color.yellow)
sphere(frame = f, pos=(-2,0,0), color=color.red)
sphere(frame = f, pos=(2,0,0), color=color.red)
```

Check that when you run this, nothing spectacular happens (**frames** are invisible.) But now add this loop at the end of the above:

```
while 1:
    rate(40)
    f.rotate(angle=0.05, axis=(0,0,1))
    f.pos = f.pos + vector(0.01,0.01,0)
```

2. RECORD WHAT YOU SEE WHEN you run this program.
3. You can also chain frames together: a frame can belong to a frame. You can make lists of frames. Note this about lists: you can refer to a member of a list by giving its position in the list: the first item is at position [0], the second item is at position [1], the third item is at position [2], etc. Negative numbers count from the end: position [-1] gives the last item in the list, position [-2] is next to last, etc. Try out the following examples (you should do this and understand this, but you don't *need* to put it into your report):

```

alist = [5, 9, 43, 27,25]
print alist[0]
print alist[1]
print alist[4]
print alist[-1]
from visual import *
boxlist = [box(), box(pos=(1,1,1)), box(pos=(3,3,3))]
boxlist[1].rotate(angle = 2, axis=(0,0,1)) ## WHAT DOES THIS DO?

```

4. Now use frames and lists together. Make a program by
  - a. **importing visual.**
  - b. turning off **scene.autoscale**
  - c. starting a list named **framelist** with a single frame (default values).
  - d. Does your program work so far? Check it. Note that frames are invisible and you are simply seeing that you have made no errors so far.
  - e. start a **for** loop – step through the numbers from 0 to 15. In this loop:
    1. set **prev\_frame** to be the last item of **framelist**. Run your program to make sure there is no error.
    2. create **new\_frame**, a **frame** that is a subframe of **prev\_frame** (that is, **frame = prev\_frame** should be an argument to the **frame()** function), has an **axis** attribute of **(0.9,0.,0.1)**, and a **pos** attribute of **(0, 0.5, 0)**.  
**new\_frame = frame(frame=prev\_frame,axis=(0.9,0,0.1),pos=(0,0.5,0)**  
 Run your program again.
    3. **append new\_frame** to **framelist**. Run your program again.
    4. Within the loop, still, create a **box** that has **new\_frame** as its **frame** and has a **height** of 0.2.
  - f. RUN YOUR PROGRAM – TAKE A SNAPSHOT and EXPLAIN WHAT YOU SEE.
5. Dynamically modify frames. Add a **while 1:** loop at the end of your program that repeats 50 times a second and rotates the *first* **frame** in **framelist** by an **angle** of 0.02 about the *y*-axis (**axis=(0,1,0)**). You can rotate an object in VPython by using the function **rotate** which belongs to an object. If **b** is a **box**, you can rotate an angle of 0.1 about the *z*-axis using the statement **b.rotate(angle=0.1, axis=(0,0,1)**. Do this, then, for your frames. EXPLAIN WHAT YOU SEE WHEN ROTATING THE 1<sup>st</sup> FRAME AND REPEAT THIS FOR A ROTATION OF THE 5<sup>th</sup> FRAME.

## ONLY A PROGRAM CAN MAKE VIRTUAL TREES

This is likely to be your homework, but if you have time, complete in class.

1. Upload the files **fractreeWEB.py** and **fractreemodule.py** from the *Codes* page at the PHY307 pages. Save these to your home directory. You will only modify **fractreeWEB.py**. The function **fractree** takes some arguments, draws a tree according to those arguments, and returns a real number that is the total surface area of the branches of the tree.
2. Run **fractreeWEB.py**. RECORD WHAT YOU SEE; INCLUDE A SNAPSHOT.
3. Change the number of generations and run again. RECORD WHAT YOU SEE, INCLUDE A SNAPSHOT.
4. Now make a forest of trees on a single scene. Each tree is to have a different **numgenerations**, from 0 to 7, say. Do this by writing a **for** loop that will vary **numgenerations**. Vary the  $x$ -location of the frame **f** for each tree (set the  $x$ -component to be dependent on **numgenerations**, similar to making a row of spheres in the midterm or a row of cubes in the first homework), so that the trees will be visually separate. RECORD WHAT YOU SEE, INCLUDE A SNAPSHOT.
5. Note that you also have data on the surface area of the tree as a function of scale. This is in the output/text screen. PLOT THIS DATA (SURFACE AREA VS. SCALE) AND ESTIMATE THE FRACTAL DIMENSION OF THE TREE. This is similar to plotting number of triangles vs. scale in earlier work.
6. Finally, *make your own tree type* and estimate its fractal dimension. You can do this by taking off or adding a subbranch to **axislist** and **poslist**. These two lists give the information on how to generate subbranches. You can, in addition, modify **shrink** if you like. INCLUDE SNAPSHOTS AND PLOT THE DATA TO ESTIMATE THE DIMENSION.