illiMath04 Notes – June 25, 2004 – Dr. Peter Brinkmann

William Baker

June 25, 2004

1 Using Interactive Python Prompt While Running a Distributed Scene Graph Program

There are two different ways to run a python graphics program:

- At the OS prompt type: "python foo.py"
- Enter Python and then type from foo import *

There are important differences between these two ways of running a Python program. If you use the first option, and start the program from the OS prompt, then you will enter the program's main loop. This results in the program running normally including animation. However, in this mode the Python command prompt will be unavailble while the program is running. If the second option is used, then the Python interactive command prompt will be availible.

When the second option is used (first entering Python and then importing the program), the interactive Python command prompt stays availble. However, the program's animation will not run, and any sort of mouse or keyboard interation there might normally be will be unavailible. Even with these restrictions, it is still possible to control the program by calling its functions manually, and this can prove to be very useful for testing purposes. See example code below for the blobbyman demo:

1.1 Code Example: Using the Python command with the Blobbyman (blobby.py)

After starting the Blobbyman program, the Blobbyman appears in a window, but the mouse and keyboard cannot manipulate him like they normally can. The code segments below are a few examples of commands that could be run to manipulate Blobbyman from the Python prompt.

To Move Blobbyman's Head:

```
>>> dgTransform(TID['neck'],T['neck']*ar_rotationMatrix('x',3.14/4)
```

To get wand data:

```
>>> ifo.getObjectMatrix()
```

Note: "ifo" is an interface object that gets data from input devices

2 Using the time Module (to Slow Animation)

Using the "time" module it is possible to slow animation, or pause the program. The example give by Dr. Brinkman used the bouncing ball demo (bounce.py). Two ways of slowing animation were given, and are briefly explained below.

Option 1: Slowing animation does not affect the magnitude of changes in the scene at each step of the animation (Regardless of the time in sleep(), objects in the scene move the same number of steps and the same distance).

```
from time import sleep
for i in range (100):
    sc.update()
    sleep(0.05)
```

Option 2: Couple the sleep time and animation step size, so that the animation speed is linked to the distance that objects in a scene move at each new frame of animation.

```
tt = 0.05            # Variable to hold time step
for i in range(100)
   sc.update(arVector3(0,-9.81,0)/.305, tt)  #arVector(...) included
                                             #because of order of
                                             #parameters, see below
   sleep(tt)
```

3 How the order of parameters affects calling functions

If you specify the names of the parameter that you are sending in to a function, order does not matter. However, if you send in one parameter without a name, Python will replace the first parameter's default value.

4 Using Threads to create an interactive mode for the Master/Slave framework

4.1 Problem

The problem with the Master/Slave framework is that to get anything to work you must use the GLUT mainloop (or whichever graphics main loop you like).

While inside this loop you get to see everything you should, and to control your program as you programmed it to be controlled. However, this is an all or nothing situation. Once the main loop is entered, all control using the interactive Python prompt is lost.

4.2 Solution

The solution to this proble is to run a Python interpreter in a seperate thread. This will allow for Python interactive command line control and allow the main loop to excecute and run the program.

4.3 Code

```
from threading import Thread

def interact():
   while 1:
      s=raw-input('### ')
      try:
         exec s in globals()
      except Exception, e:
         print e
Thread(target=interact).start()
```

5 Interactive Python Tips

5.1 Loading *.py files into Python Interactive Mode

There is a difference between the following two lines:

```
import foo
```

```
from foo import *
```

The first line loads the module foo into Python, but it doesn't load the names of variables and functions from foo into the global namespace. The second line loads foo and all of its methods into the global namespace.

To illustrate the difference between these lines, assume there there is a function bar() in foo. The syntax used to call bar() depends on which of the two methods above was used to load the module foo.

Using the first method we would type:

```
>>> foo.bar()
```

Using the second method we would type:

```
>>> bar()
```

5.2 The dir() commmand

The function dir() returns a listing of all the functions and variables currently availible for use. dir() can accept as a parameter the name of a module so that you can see all of the functions and variables within a module that has been loaded into Python (but not into the global namespace).

Example:

```
Python 2.2.3 (#1, Oct 15 2003, 23:33:35)
n 2.2.3 (#1, Oct 15 2003, 23:33:35)
[GCC 3.3.1 20030930 (Red Hat Linux 3.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> dir()
['__builtins__', '__doc__', '__name__']
>>> dir(math)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'math' is not defined
>>> import math
>>> dir()
['__builtins__', '__doc__', '__name__', 'math']
>>> dir(math)
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh',
>>> from math import *
>>> dir()
['__builtins__', '__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cos
>>>
```

5.3 The help() and type() commands

Two other built in commands in Python are help() and type(). These two can be used together to help you learn what built in functions are availble for objects you might have in your program.

help() takes as a parameter a type and then returns information about that type.

type() takes a variable and returns its type.

These can be used together so that all you would need to do to learn about one of your variables is help(type(variable name)).

Example:

```
>>> help(str)
...help on strings
>>> foo="hello world"
>>> type(foo)
<type 'str'>
>>> help(type(foo))
...help on strings
```

```
>>>
```

5.4 The __name__ variable

Python has a built in variable __name__. This variable is assigned a value based on how a program is run. If a program is run from the command line:

```
[..]$ python foo.py
```

Then __name__ equals __main__. However, if the program foo is loaded into Python in interactive mode, then __name__ will equal something else:

```
>>> import blobby
...
...
...
>>> blobby.__name__
'blobby'
>>>
```

This is useful because inside a program you can check if __name__ equals __main__, and if it does then you know it was called from the OS prompt.