PHYSICS 113 – Recitation Assignment 2

Name

Recitation Assignment # 2

The Three Body Problem

Oct. 4, 2006

Important Note:
You must put your name on all assignments you turn in, failure to do so will result in a loss of points. Put a comment at the beginning of all your code like this:

```
# Travis Hoppe - Rec. Assignment #2 - Oct. 4 2006
```

In today's assignment, you won't really do any new physics, but you will learn a bit of new programming. In particular, we will be using the:

```
star=[]
```

command.

What is this? Essentially, this tells the computer that you are going to make a list of things (stars, in our case), and then, when you refer to them later in your code, you can access them using (for example):

```
star[0].m=msun
```

Rather than give everything a name like you did last time ("earth", "jupiter", "sun"), you'll be creating a bunch of stars and numbering them. Note: the first star has number 0.

How do you create a new star? Well, last time you wrote:

```
earth=sphere(pos=(....),mass=...)
```

This time, you'll type:

```
star.append(sphere(pos=(...),mas=...))
```

Let's see how this all works.

As last time, when you reach a checkbox, ask Travis to come over and check your work.

1. Log in, relax, etc.
2. Go into your contemporary1 directory, and use emacs to create a new file called "prog2.py".
3. Start your program with the following:

```
from visual import *

star=[]


#define a few standard units
AU=1.5e11   #1 ‘‘Astronomical Unit’’ is the distance from the earth to
            #the sun
msun=2e30
G=6.67e-11
ve=30000.
yr=3.15e7

# Set some things for the screen
scene.range=5*AU
scene.autoscale=0

# put in the first star
star.append(sphere(pos=vector(-1*AU,0,0),color=color.red,radius=0.1*AU))
```

Note how much cleaner your code looks when you define positions, for example, in terms of astronomical units

4. Set the velocity of the first star to be ve in the y-direction, and the mass to be $1M_\odot$.

□ 5. Create a second star at $< 0.5AU, 0, 0 >$. Color it yellow. Give it a mass of $2M_\odot$, and a velocity of $< 0, -0.5\ ve, 0 >$. All stars should have the same radius.

6. Now we start adding the physics. Set the initial time to $t = 0$, and make each timestep equal to $0.001 * yr$. Use the "while" statement we used last time, and run the code for 20 years. During each step, make sure you set t=t+dt.

7. Write a statement which computes the force (as opposed to the acceleration) on each particle. As a reminder, the force relation is:

$$F_i = \frac{GM_iM_j}{r_{ij}^3}(\vec{r}_j - \vec{r}_i) \tag{1}$$

where i ,and j, represent the star you are computing the force on and the star which is causing the force, respectively.

Note: You will need to compute star[0].F as well as star[1].F.

8. After the force calculation add a line (each) updating the velocity (as you did for the earth in last week's exercise).

□ 9. After the velocity update, add a line (each) updating the positions. You should find that the stars orbit each other.

10. Let's add a trail. Right above where you "create" your stars, add the line:

```
c=[]
```

Then, after each star is defined, add the line:

```
c.append(curve(color=color.red,pos=star[0].pos))
```

For star "0", and similarly for star "1" (which should be done in yellow).

Then, after you have updated the position of your stars in the while loop, add the line:

```
c[0].append(star[0].pos)
```

which will create a red trail following star[0] around.

□ 11. Add a third star. This one should be blue (and have a blue trail). It should have a mass of $0.5M_{\odot}$, and start from a position $< -10*AU, 0.1*AU, 0 >$ and with a velocity $< 0.1*ve, 0, 0 >$.

Make sure you compute the gravitational interactions between the 3rd star and the first two.

12. Spend a little while playing around with some parameters. Change the speed (slightly) at which you start the 3rd star, or change the "height" (y position) where it starts. See how the other stars respond.

13. Now, you should have a working code, and since I'm about to have you make some big changes, I don't want you to ruin your old code. Use the "save-as" button in emacs to save your file as "prog2a.py". That way, the old version will be nice and safe, and you'll be editing the new version.

14. Now, one of the things you may have noticed is that you have to type the same things again, and again, and again (1 for each particle). This is very time consuming. We will use loops to make things simpler.

Before your while loop, write the following:

```
nstars=3
```

15. In the loop, write:

```
while (t < 20*yr) :
rate(200)
# Calculate everybody's acceleration
for i in range(nstars):
star[i].F=vector(0,0,0)
for j in range(nstars):
if (i != j) :
    star[i].F=star[i].F+ --- put the force calculation in here
```

The "force calculation" should simply be the force on "i" from "j". You've already done this, except for [1] on [0], for example.

In this case, what you are doing is a "for loop." It's saying:

(a) Take a variable called i and set it to 0.

(b) Do everything underneath the `for` statement.

(c) Add 1 to i (so the after the first time around, it takes the value 1).

(d) Go to the top of the loop.

(e) Check – is i larger than nstars? If so, start doing whatever command is outside the loop. If not, go to step (b).

16. Check to make sure your code still works!

□ 17. In addition to the force calculations, you can replace the bit at the bottom of the "while" loop where you update the positions and velocities with a "for" loop.

Note: the loop we did for force is a double loop (loop over the particle you are computing the force for *and* loop over the particle providing the force). This time, there will only be a single loop.