Liz Rogers
Math 198, Spring 2005

d3s.py

My final project was to convert 3spheres.py, a pocket program in VPython, into d3s.py, coded in Syzygy language, so the program can run in the CUBE.

Guido van Rossem created Python, an object-oriented language, in 1990. Programming in Python included a graphics library to create 2D graphics, but was a bit complicated to both learn and use. David Shearer, a sophomore at Carnegie Mellon University, then created a 3D graphics module for Python, which allowed users to easily create 3D objects and place them in 3D space. This module, called VPython, doesn't force programmers to deal with display management; instead, they can concentrate on the actual mathematical computations behind the movement.

These aspects of VPython are what enable even beginner computer programmers to make simple 3D pocket programs if they know how to figure out the math aspect of the program. After learning to use VPython by completing the VPython Introduction, reading other simple programs such as bounce.py and bresenham.py, and creating different objects in learning.py to teach myself how to use simple VPython language, I was able to complete small pocket programs without any other computer programming background.

3spheres.py is one of these pocket programs. After completing the VPython Introduction, I decided to modify it to include 3 bouncing balls (allowing for collisions) and add both arrows and trails. First, the balls and walls are created with language like the following:

```
ballA = sphere(pos=(-5,0,0), radius=0.5, color=color.red)

wallR = box(pos=(6,0,0), size=(0.2,15,15), color=color.yellow)
```

The change in time (`dt`) was set at `0.02`. I also created a list for the balls so that later in the program I could check each ball's x, y, and z position and set the boundaries for their bouncing; these boundaries were visualized with walls. I created the arrows and the trails with language like the following (arrows are built into the VPython graphic library, and trails are members of balls):

```
bvA = arrow(pos=ballA.pos, axis=ballA.velocity, color=color.green)

ballA.trail = curve(color=bvA.color)
```

Finally, I created an infinite while loop (`while 1==1`). In this loop, all the balls' positions were constantly updated:

```
ball.pos = ball.pos + ball.velocity*dt
```

The balls' positions were then checked against the positions of the walls (and the front boundary, which was `z = 6`) with if-then statements where if the ball reached a wall, the velocity was reversed. Also, the position of each ball was checked against the other two balls; if two balls were in the same position (they collided), the velocities of both balls were reversed; these checks were done with language like the following:

```
if ball.x > wallR.x:

        ball.velocity.x = -ball.velocity.x

if ballA.pos == ballB.pos:

        ballA.velocity = -ballA.velocity

        ballB.velocity = -ballB.velocity
```

Finally, each arrow's position was set at the corresponding ball's position, and each arrow's axis was set at the corresponding ball's velocity vector. The trails for the balls were set at the corresponding ball's position; these were done with language such as the following:

```
bvA.pos = ballA.pos

bvA.axis = ballA.velocity

ballA.trail.append(pos=ballA.pos)
```

I didn't get a chance to develop my knowledge of Python and VPython as much as I could have in the course since I became ill and left the semester early, so instead creating a new program for my final project, I decided to convert 3spheres.py into Syzygy.

I had to create objects (both balls and walls) with language such as the following in Syzygy (which corresponds to the VPython version of the code in example):

```
tidA = dgTransform("balltransform", "basetransform",
  ar_translationMatrix(0,4,-0) )

mm=ar_scaleMatrix(0.5,0.5,0.5)

dgTransform("scaleball","balltransform",mm )

dgMaterial("ballcolor","scaleball",arVector3(1,0,0))

ballA=arSphereMesh(20)

ballA.setSectionSkip(2)

ballA.attachMesh("ballA","ballcolor")
```

"tidA" allows the position to become a variable so it can be constantly updated (done later in the program with a function called "updateball"). dgTransform is defined as the following:

```
dgTransform("basetransform",foo, ar_translationMatrix(0,2,-5) )
```

foo is defined as the following:

```
ff=arDistSceneGraphFramework();

foo=ff.getNavNodeName()
```

updateball used the following code to update each ball's position (which corresponds to the VPython version of the code in example):

```
for number in range(0,2):

        pos[number] = pos[number] + vel[number]*dt
```

updateball also called another function I created, wallbounce, to check if each ball was in the same position as the wall (collides with any of the walls) and wallbounce also reversed the ball's velocity if it was found to collide with a wall. The code looked something like the following (which corresponds to the VPython version of the code in example):

```
def wallbounce(ball-pos, ball-vel):

       if ball-pos[0] > wallR-pos[0]:

               ball-vel[0] = -ball-vel[0]

wallbounce(ballA-pos, ballA-vel)
```

updateball called another small function, ballbounce, to check if each ball was in the same position as any other ball. If two balls collided (were in the same position), ballbounce reversed both balls' velocities. That code looked something like the following (which corresponds to the VPython version of the code in example):

```
def ballbounce(ball-pos, ball-vel, ball2-pos, ball2-vel):
        if ball-pos == ball2-pos:
                ball-vel = -ball-vel
                ball2-vel = -ball2-vel
ballbounce(ballA-pos, ballA-vel, ballB-pos, ballB-vel)
```

Finally, updateball used dgTransform again to update the ball's positions:

```
dgTransform(tid, ar_translationMatrix(pos[0],pos[1],pos[2]))
```

Not all aspects of 3spheres.py transferred into Syzygy; the arrows could not be created as the balls and walls were, and though trails were a member of the "ball" object in VPython, Syzygy spheres weren't classes with built-in members. Thus, d3s.py does not use either arrows or trails.

Though d3s.py doesn't have all of the features of 3spheres.py, these two programs can serve as an example of simple pocket programs and their conversions into Syzygy, just as bounce.py and dbo.py do. It also goes to prove that even beginners with very little (or no) previous exposure to computer programming in any language can learn to make 3D pocket programs with VPython.