**CS360**
**Lab 2**

# Python for POV-Ray Animations

The images you made in Lab 1 using POV-Ray are really 3-D *models* representing some sort of virtual reality. Since you used POV's Scene Description Language to define a three dimensional scene, rather than just the colors in a 2-D grid of pixels, it is easy to change this virtual reality. The compiler takes care of rendering your new view of reality. We shall use this to our advantage to create a movie of a scene.

Take a moment to watch a couple animations that others have created using POV-Ray:

- `http://www.irtc.org/`,
- `http://runevision.com/3d/anims/`

The idea is that you use POV-Ray to individually render each frame of the movie. All you need to make this work, is a way to script POV. Enter Python. Python is a fabulous language for scripting other applications and 'gluing' them together.

# Using Python to script a movie

## Part I: Camera Pan

Begin your lab by choosing a static scene in POV. It can be your scene from Lab 1 or you can use something from a tutorial or website somewhere (make sure you have permission and then give proper attribution to code obtained elsewhere), as long as you understand all the code it contains. This will be your starting code and will be the scene you'll animate. Let's call this file: `base.pov`

To get going, write a short Python program to control the entire process of generating a single image using this file. Your python program should be called `animate.py` and you should be able to create a single image by executing it from the command line like this:

```
python animate.py
```

The base POV file (`base.pov`), the details of the animation and the output image file name should be hardcoded in the python script. To have python open and read a text file you can do this:

```
fin = open( 'base.pov' )
sdl = fin.read() # Read the entire file into a string
fin.close()
# Now you have the entire POV source code in the string sdl
```

To write a new file using the string `sdl_new`:

```
outfile = 'tmp.pov'
fout = open( outfile, 'w' )
fout.write( sdl_new )
fout.close()
```

From within Python, you can invoke another program, as if from the command line, by executing the `os.system()` function. Here's the idea (in Python):

```
import os
#...
pov_cmd = 'C:\\POV-Ray\\v3.7\\bin\\pvengine.exe  +I %s +O%s -D -V +A +H600 +W800'
(make sure the path is the absolute path to the pvengine in POV-Ray on your computer)
cmd = pov_cmd % ('base.pov','base.png')
os.system(cmd)
```

Notice that we're using string formatting with the % operator: %s means format a string. Once you have this working, you now have great power. Python is much more expressive a language, in the sense of a traditional general purpose programming language, than POV-Ray. You can now use Python to change values, or add things to your `base.pov` scene, on-the-fly.

Your task for Part I of this lab will be to have the CAMERA pan around the center of your scene. See my example movie `movieA_med.avi`(2.1MB) or `movieA_high.avi`(12.2MB) from the class website. The camera pans all the way around the center of the scene, one time, and travels along a helical path. This is traveling in a circle while at the same time moving up at a constant rate. (How do you calculate this path? Use a little geometry and trigonometry, of course. sin(degree * (pi /180)) cos(degree * (pi /180)) ) You can easily accomplish this task by changing the location of the camera and possibly the `look_at` vector for each frame of the animation you create. Write a loop and generate one `.png` image every time through the loop. It is easiest if you give these images consecutive names, i.e. `tmp0001.png`, `tmp0002.png`, `tmp0003.png`, ....

After rendering a hundred or a thousand frames (take it easy at the beginning and only generate 10 or so) you can now make a movie out of them. There are a number of utilities that allow you to do this, for a number of different platforms. I chose `mencoder` which comes with the `mplayer` movie player (`http://www.mplayerhq.hu/design7/news.html`). If you choose to use this one, a possible command to give it is (within Python of course):

```
#
# Now make the movie
#
print 'Encoding movie'
os.system( 'mencoder.exe mf://tmp*.png -mf type=png:fps=25 -ovc lavc -lavcopts \
  vcodec=msmpeg4:vbitrate=2160000:keyint=5:vhq -o movie.avi ' )
```

(The \ at the end of the line is a continuation; you're not allowed to have a newline in the middle of a string in Python.) This will encode all the frames into the movie `movie.avi` using mpeg4 at 25 frames per second and a very good quality setting.

Feel free to use other utilities or even try to make an animated gif. The campus has Macromedia Flash 8 on the Terminal Server if you feel like trying to make a flash movie. If you absolutely cannot find a way to do it, send me your code, I'll run your Python code to generate the files and then encode the movie for you. Another possibility is Bink, from RAD Game Tools.

If you're having trouble watching the videos (codec problems), try VLC (`http://www.videolan.org/vlc/`).

### Part II: Moving objects and creating new ones

For the second half of the lab, you'll need to have your Python code create new objects in your scene and move them around, as the animation continues. You may either extend your code and movie from Part I, and make a more complicated movie, or start over and do this section without any camera panning. It's up to you what objects you create and move around, and exactly how you do it. What I'm looking for is further Python work to modify or add to your `base.pov` file. Explore further how Python works.

### Requirements for your Python code (Part II)

Since this lab is really about experiencing aspects of Python that make it different from a language like Java, I'm requiring that you use certain language features in the Python code you write.

1. For Part I you're allowed to just insert the CAMERA into your `base.pov` file directly from python. But for Part II you need to actually *change* what is in your `base.pov` file. Your python code needs to read in the input file and then search/find/match for the part you're wanting to change, change it, then go on to the next thing you're wanting to change.
2. You must write one or more modules (this is easy since all Python code must go in a module). Your module(s) must have functions and variables at module scope.
3. You must write a class and create one or more objects from it. This object must be an integral part of your program. i.e. use it as part of a well designed objected oriented program.
4. Your class must have instance data members as well as instance methods. It must also have members that are the analogue of `static` fields in Java. It must have at least one constructor.
5. You must use the built-in lists $[1, 2, 3]$, tuples $(1, 2, 3)$ and dictionaries $\{'Oregon' : 'Salem'\}$ in an integral way
6. At least one method or function must use 'keyword arguments'

## Restrictions

- You may not use this: `http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/205451` or anything like it. But feel free to read it for ideas.
- All the Python code you turn in must be your own. (But the POV base code can be from elsewhere as described above.)

- You may not use the built-in animation capabilities of POV-Ray.