

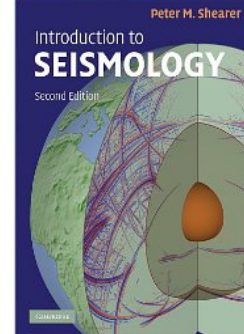
---

## L15 – POV-Ray - Part 1

---

### 1. What is POV-Ray?

**POV-Ray** stands for the **Persistence of Vision Raytracer**. POV-Ray belongs to a class of programs called ray tracers. For you seismologists this concept should be quite familiar. This has nothing to do with ray tracing seismic ray paths, but is essentially the same thing. Here we are creating an image by ray tracing light paths from an object to an observer or camera. You can generate some really incredible images using this tool. The cover art for Peter Shearer's new seismology text was designed using POV-Ray by Gunnar Jahnke (of LMU, Munich). For some really nice examples of what you can do check out the POV-Ray Hall of Fame:



<http://hof.povray.org>

In this class we will use POV-Ray on the Windows side of the computers. However, POV-Ray can be run on almost any system. The reason is that POV-Ray is a very basic program that doesn't include any graphics. It can produce images, but relies on other software available on any computer system to view these images. In this sense POV-Ray is similar to the Generic Mapping Tools (GMT). To create an image in POV-Ray we create a text file in the **POV scene description language** and then we render this image in POV-Ray. Just as in GMT we will be writing out our scenes in a text file which tells POV-Ray where the camera, lights, and objects are.

### 2. Getting Started

To launch POV-Ray: **Start > Programs > POV-Ray for Windows**

Now let's create a new text file to describe our scene:

**File > New File**

Here is a simple example you can type in:

```
// This is a simple sphere

// first, the camera position
camera {
  location <0,8,-15>
  look_at <1,0,5>
}

// now add some light
light_source {
  <5,10,-15>
  color rgb <1,1,1>
}
```

```

plane { // the floor
  y, 0 // along the x-z plane (y is the normal vector)
  texture {
    pigment { color <0,0,1> } // checkered pattern
    normal { ripples 20 }
  }
}

sphere {
  <1,3,-5>, 3
  pigment {
    marble
    turbulence 1
    color_map {
      [0.0 color <1,0,0>]
      [0.25 color <0,0,1>]
      [1.0 color <0,1,0>]
    }
  }
  scale 3
}
finish { reflection 0.2
refraction 0.8
ior 1.5
phong 1 }
}

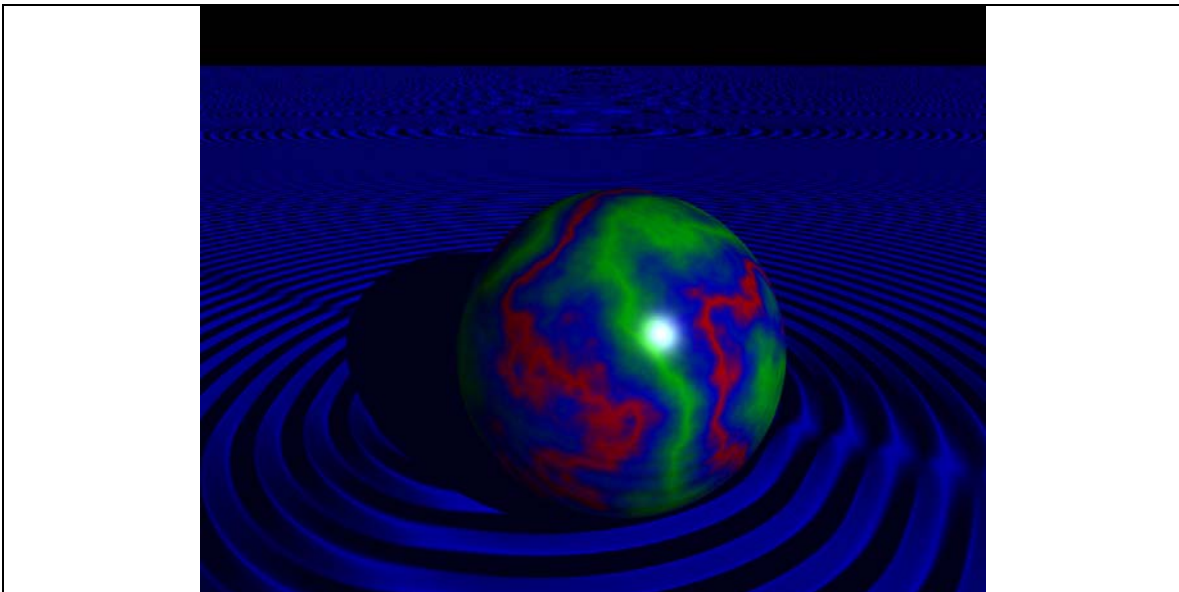
```

First we need to save the file: **File > Save As...** Let's just save the file name as **sphere.pov**.

And now we can render the image by hitting the Run button:



The following image should be rendered. Note that POV-Ray will save the image as **sphere.bmp** in the same location as you saved the **sphere.pov** file.



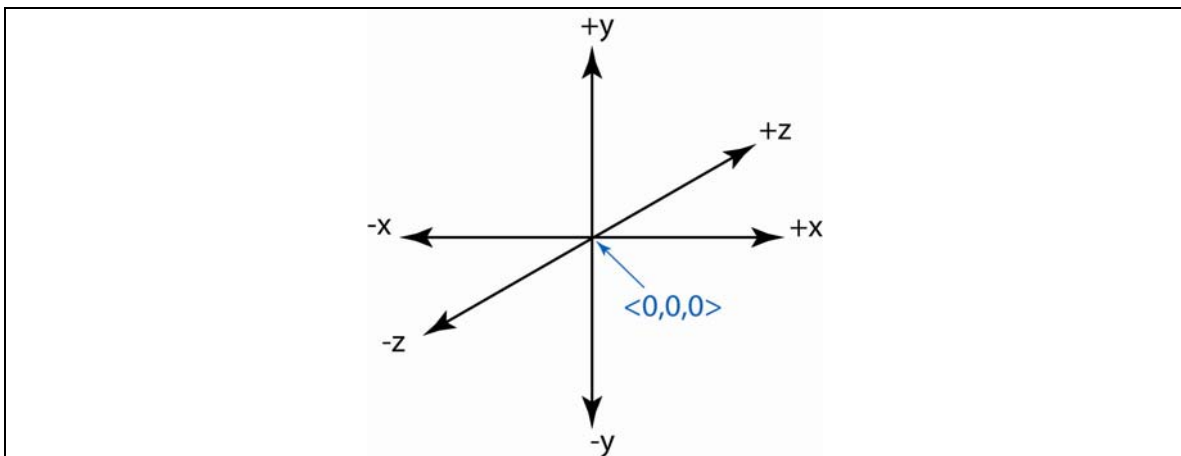
Note that we can change the size of the rendered image by going to **Render > Edit Settings/Render** and selecting a different size (e.g.,  $1024 \times 768$ ).

Initialization files (.INI files) contain information such as the resolution of the rendered image. If you need a resolution that is not included in the **QUICKRES.INI** file you can change them by going to: **Tools > Edit resolution INI file**.

### 3. Camera Angle

The example in the previous section shows a few key points about what is necessary in a POV-Ray file: (1) camera location, (2) lighting location, and (3) some objects to display (in this case a plane and a sphere).

To understand where to place the camera we need to first understand the POV-Ray coordinate system. POV-Ray uses a simple Cartesian coordinate system that looks like this:



The two easiest attributes to use of the **camera** object is as shown in the first example. Namely the **location** and **look\_at** attributes. In our above example we used:

```
camera {
  location <0,8,-15>
  look_at <1,0,5>
}
```

Where the location just gives us the **x**-, **y**-, and **z**- coordinates of where the camera is located (**x = 0, y = 8, z = -15**), and we stated we wanted it to look at the position **x = 1, y = 0, and z = 5**. We could make a change and say we want to look directly at the origin by changing the **look\_at** attribute:

```
look_at <0,0,0>
```

It's more fun now to change the **location** attribute. Try a few changes, for example:

```
location <0,10,-50>
location <0,5,-10>
```

## 4. Lighting

Adding lighting to our scene is very similar to that of how we describe the camera position. The simplest light source is just a point light source.

```
light_source {
  <5,10,-15>
  color rgb <1,1,1>
}
```

Here we first specify that we want the light placed at position  $x = 5$ ,  $y = 10$ , and  $z = -15$ . We next state we want to use a white light. We use a **RGB vector** to describe the color of the light, but note that in GMT our RGB colors ranged from 0 to 255, in POV-Ray our colors range from 0 to 1. So, the vector  $\langle 1,1,1 \rangle$  would be the same as  $255/255/255$  in GMT. In general it's best to use a white light source, but you are not restricted to it.

To change our light source so that it is looking directly down on our object try a position:  
 $\langle 0,15,0 \rangle$

You can also add additional light sources. E.g., just add another line group:

```
light_source {
  <-20,0,0>
  color rgb <0.5,1,1>
}
```

## 5. Simple POV-Ray Objects

Defining a camera position and light source are great, but they serve no purpose unless you have something to look at. The simplest thing one can do as a new comer to POV-Ray is learn how to manipulate some of the basic POV-Ray shapes. In general primitive POV-Ray objects are described as:

```
Object_Name {
  Object_Parameters
  Some_Simple_Attribute
  Some_Complicated_Attribute {
    Some_Attribute
  }
}
```

### 5.1 Sphere

We already saw an example of a sphere above. The basic form of drawing a sphere is as follows:

```
sphere {
  <center>, radius
}
```

But, we should add some color to it with the pigment attribute. An example is:

```
sphere {
  <0,0,0>, 3
  pigment {
    color rgb <0, 0, 1>
  }
}
```

To make sure we understand how these objects work let's create a simple pov-ray file that we can play with:



```
// Make Some Simple Objects

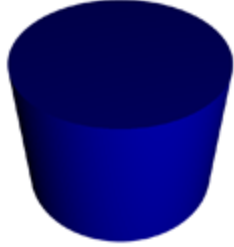




camera {
  location <0,10,-10>
  look_at <0,0,0>
}

light_source {
  <5,10,-15>
  color rgb <1,1,1>
}

// Simple Sphere
sphere {
  <0,0,0>, 3
  pigment {
    color rgb <0, 0, 1>
  }
}
```

There are several classes of basic objects we can use in POV-Ray. The next table shows the basic syntax for some of the most important.

Object	Syntax	Example	
Box	<pre>box {   &lt;corner-1&gt;,   &lt;corner-2&gt; }</pre>	<pre>box {   &lt;0,0,0&gt;, &lt;3,3,3&gt;   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	
Cone	<pre>cone {   &lt;center-1&gt;,   radius-1   &lt;center-2&gt;,   radius-2 }</pre>	<pre>cone {   &lt;0,0,0&gt;, 3   &lt;0,4,0&gt;, 0   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	

Object	Syntax	Example	
Cylinder	<pre>cylinder {   &lt;center-1&gt;,   &lt;center-2&gt;,   radius }</pre>	<pre>cylinder {   &lt;0,0,0&gt;, &lt;0,4,0&gt;,   3   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	
Disc	<pre>disc {   &lt;center&gt;,   &lt;normal&gt;,   radius   [, hole radius] }</pre>	<pre>disc {   &lt;0,0,0&gt;,   &lt;1,1,0&gt;,   3, 2   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	
Plane	<pre>plane {   &lt;normal&gt;,   offset }</pre>	<pre>plane {   &lt;0,1,0&gt;, 1   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	Just a plane surface!
Sphere	<pre>sphere {   &lt;center&gt;, radius }</pre>	<pre>sphere {   &lt;0,0,0&gt;, 3   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	
Torus	<pre>torus {   major radius,   minor radius }</pre>	<pre>torus {   3, 0.5   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	
Triangle	<pre>triangle {   &lt;corner-1&gt;,   &lt;corner-2&gt;,   &lt;corner-3&gt; }</pre>	<pre>triangle {   &lt;0,0,0&gt;,   &lt;1,0,0&gt;,   &lt;0.5,1,0&gt;   pigment {     color rgb &lt;0,0,1&gt;   } }</pre>	

## 6. Finish

An objects finish describes how the objects interact with light. For example, how much light they reflect. To play with the finish attribute let's look at our simple sphere example again. We can make our object shine a bit by having our light source hit the sphere and giving it the phong finish:

```

camera {
  location <0,10,-10>
  look_at <0,0,0>
}

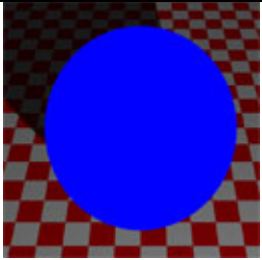
light_source {
  <5,5,-15>
  color rgb <1,1,1>
}

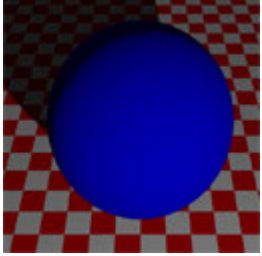
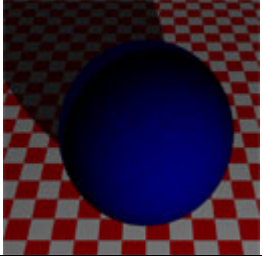
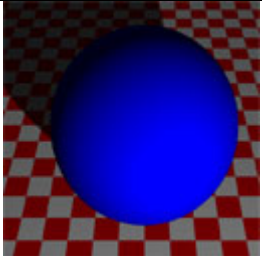
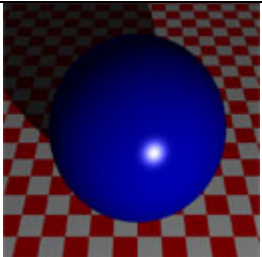
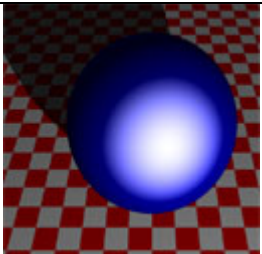
// create a checker board surface
plane {
  <0,1,0>, -5
  pigment { checker color <1,0,0> color <1,1,1>}
}

// Simple Sphere
sphere {
  <0,0,0>, 3
  pigment { color rgb <0, 0, 1>}
  finish { phong 0.8}
}

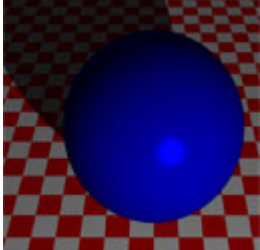
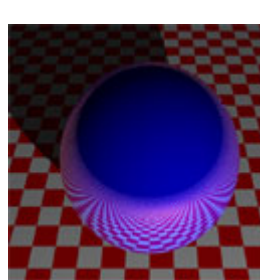
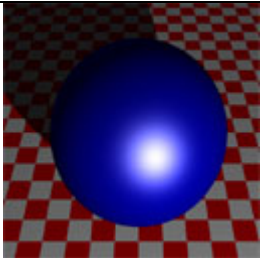
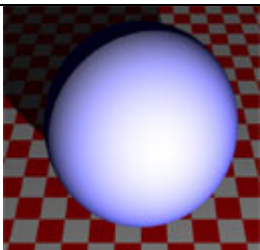
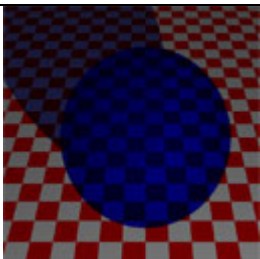
```

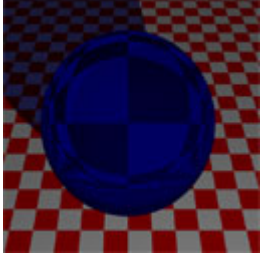
The next table describes the primary ways we can apply finish to our objects.

Finish	Description	Example	
<b>ambient</b>	<p>How much of the lighting comes from ambient light (i.e., light that bounces off other objects).</p> <p><b>Range {0.0 to 1.0}</b></p> <p>0.0 means that objects that are not directly lit will be black.</p> <p>Higher values will make an object appear to glow.</p>	<pre> finish {   ambient 1.0 } </pre>	

<p><b>brilliance</b></p>	<p>How much of the lighting from direct light sources will bounce off of the object.</p> <p><b>Range {0.0 to ??}</b></p> <p>Large numbers can make an object more metallic looking. Numbers less than 1.0 make the object look softer.</p>	<pre>finish {   brilliance 0.5 }</pre>	
<p><b>crand</b></p>	<p>Used to make an object appear to have a rough surface.</p> <p><b>Range: {0.0 to 1.0}</b></p> <p>The larger the number the rougher the surface.</p>	<pre>finish {   crand 0.5 }</pre>	
<p><b>diffuse</b></p>	<p>Similar to the ambient keyword, but how much of the lighting will come from diffuse (direct) light sources.</p> <p><b>Range: {0.0 to 1.0}</b></p>	<pre>finish {   diffuse 1.0 }</pre>	
<p><b>phong</b></p>	<p>Create a highlight on the object.</p> <p><b>Range: {0.0 to 1.0}</b>. The larger the number the brighter the highlight.</p>	<pre>finish {   phong 1.0 }</pre>	
<p><b>phong_size</b></p>	<p>Describes size of phong highlight.</p> <p><b>Range: {1.0 to 250.0}</b></p> <p>The larger the number the smaller (tighter) the highlight</p>	<pre>finish {   phong 1.0   phong_size 1 }</pre>	



<p><b>metallic</b></p>	<p>Only works in conjunction with phong and specular finishes. Highlight on object takes on color associated with object and not just on the light color.</p>	<pre>finish {   phong 1.0   metallic} </pre>	
<p><b>reflection</b></p>	<p>Using reflection will give the surface a mirrored finish.</p> <p><b>Range: {0.0 to 1.0}.</b></p> <p>A value of 0.0 turns reflection totally off. The larger the number the more reflective the surface is.</p>	<pre>finish {   reflection 1.0 } </pre>	
<p><b>specular</b></p>	<p>This is similar to phong, in that it is used to create a highlight on the object. This is purportedly more realistic than phong.</p> <p><b>Range: {0.0 to 1.0}.</b></p> <p>The larger the number the brighter the highlight.</p>	<pre>finish {   specular 1.0 } </pre>	
<p><b>roughness</b></p>	<p>This controls the size of the highlight used in the specular command.</p> <p><b>Range: {0.0005 to 1.0}</b></p> <p>The smaller the number the smoother the object is.</p>	<pre>finish {   specular 1.0   roughness 1.0 } </pre>	
<p><b>refraction</b></p>	<p>Refraction only works if your objects are partly transparent. This can be changed in the pigment statement. For example:</p> <pre>pigment {   color rgbf   &lt;0,0,1,0.8&gt; } </pre> <p>Will partially transparent objects light can now refract through them.</p>	<pre>finish {   refraction 1 } </pre>	

	<p><b>Refraction = 0</b> means to turn <b>off</b> refraction.</p> <p><b>Refraction = 1</b> means to turn <b>on</b> refraction.</p>		
<b>ior</b>	<p>Stands for <b>index of refraction</b>. By default the <code>ior = 1.0</code> which means no refraction (speed of light is the same inside and outside of the object). This allows us to change lights index of refraction for an object</p>	<pre>finish {   refraction 1   ior 1.5 }</pre>	

Now that we know all about finish, just for fun try out the following:

```
// Simple Sphere
sphere {
  <0,0,0>, 3
  pigment { color rgbf <1, 1, 1, 0.8> }
  finish {
    reflection 0.1
    refraction 1.0
    ior 1.5
    phong 0.8
  }
}
```

For even more fun start adding in texture statements:

```
// Simple Sphere
sphere {
  <0,0,0>, 3
  texture {
    pigment { color rgbf <1, 1, 1, 0.8> }
    normal { bumps 1/2 scale 1/6 }
    finish {
      reflection 0.1
      refraction 1.0
      ior 1.5
      phong 0.8
    }
  }
}
```

## 7. Image Overlays

In our next lecture we will go over the different uses of the pigment command in detail. But, for now let's look at one of the most useful keywords: **image\_map**. On the course web page the material for this lecture contains a file called: **Earth Surface, Clouds & Ocean**. Download this file into the directory where you are keeping your **.pov** files. Unzip these files and note that these are just **.png** images of the Earth. Now, see how simple it is to overlay this on a sphere in POV-Ray using the **pigment** command:

```
// Earth Sphere
sphere {
  <0,0,0>, 3
  pigment {image_map {png "./Earth/03_Earth_Land.png"
    map_type 1 interpolate 2 }}
}
```

Of course, we don't necessarily need to restrict ourselves to a spherical Earth...

```
box {
  <-3,-3,-3>, <3,3,3>
  pigment {image_map {png "./Earth/03_Earth_Land.png"
    map_type 1 interpolate 2 }}
}
```

The most important modifier is **map\_type**:

- **map\_type = 0**: This is a planar mapping. We will describe it more below.
- **map\_type = 1**: This is the spherical mapping. The image is wrapped around the origin.
- **map\_type = 2**: This is a cylindrical mapping. The image is wrapped around the y-axis.
- **map\_type = 5**: This is a toroidal mapping. Fantastic if you ever find the need to wrap an image on a donut.

For plotting maps onto spheres we will typically want to use the **map\_type 1** modifier. But it is often useful to paste planar images onto surface.

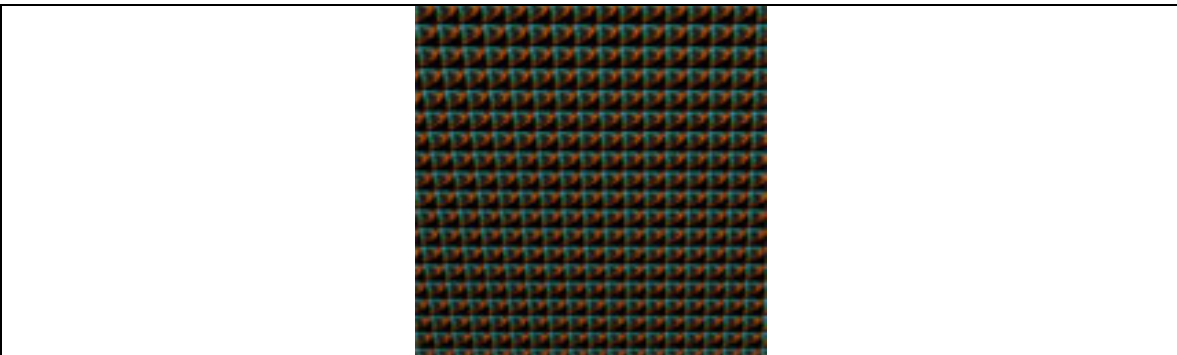
For example the following image of a nebula is taken from the Hubble telescope:



This image is of size  $1024 \times 831$  pixels. It might make a really nice background image. So, let's create a plane surface in the x-y plane and image\_map this onto it:

```
plane {
  <0,0,1>, 50
  pigment {image_map
    {png "./Hubble/04_Hubble.png" map_type 0 interpolate 2 }
  }
}
```

What you get is the next image. It's not too useful. The problem is that the normal image map takes any image no matter what it's size and maps it onto the interval from 0-1 (for both axes). So, not only does the image turn out small it is also now stretched. It also repeats the image over and over.



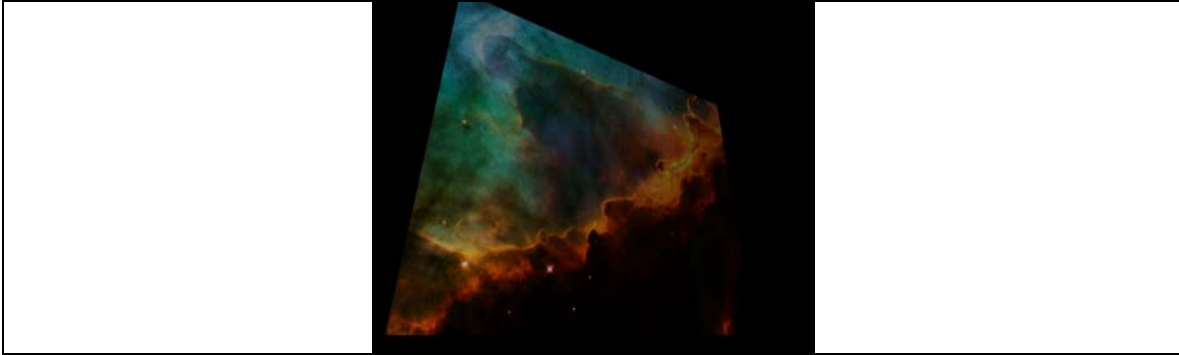
The key is to start scaling the image. Since the image wider than it is tall we can use different scales for the different axes. For example,

```
camera {
  location <0,0,-40>
  look_at <20,18,-5>
}

light_source {
  <5,5,-15>
  color rgb <1,1,1>
}

plane {
  <0,0,1>, 50
  pigment {image_map {png "./Hubble/04_Hubble.png" map_type 0
    interpolate 2 once}
    scale <123,100,0>
  }
}
```

Using the **once** key word will turn off wrapping the image. The image below shows that we can build up perspective views of our image maps for nice effects.



## 8. Translation and Rotation

If you play around with image mapping onto spherical objects POV-Ray only seems to behave well if you place the sphere at the plot origin. That isn't very useful if you want multiple objects. Luckily, we have the **translate** function.

Translate basically just tells us by how much in the x-, y-, and z- directions to move our object by. E.g., for our example of the Earth we can move it over in the positive x-direction by:

```
camera {
  location <0,0,-10>
  look_at <0,0,0>
}

light_source {
  <5,5,-15>
  color rgb <1,1,1>
}

sphere {
  <0,0,0>, 3
  pigment {image_map {png "./Earth/03_Earth_Land.png"
  map_type 1 interpolate 2 }}
  translate <5,0,0>
}
```

Rotate is really useful for our image maps mapped to spheres. We could change our look\_at parameter for our camera. But, it would be easier just to rotate the sphere. The **rotate** command looks like:

```
rotate <x angle, y angle, z angle>
```

Which tells us how many degrees to rotate our object around one of the principal axes. For example, change our earth plot to:

```
sphere {  
  <0,0,0>, 3  
  pigment {image_map {png "../Earth/03_Earth_Land.png"  
    map_type 1 interpolate 2 }}  
  rotate <30,180,0>  
}
```

## 9. Homework

1) On the course web page there is a large tar file called Solar System Objects. This file contains surface maps of many of the bodies in our solar system. This is my collection of image maps, and is not complete, but about as complete and as current as you will find anywhere I think. Use these images to do something creative in POV-Ray that utilizes at least 2 objects. Print out a copy of your image and bring to the next class. We will vote on the best image with the winner taking home a prize! Here is an example image I created in about 15 minutes. Now, good luck and have fun!

