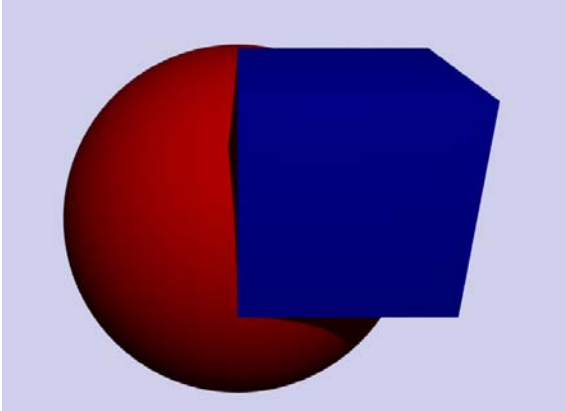

L17 – Povray - Part 3

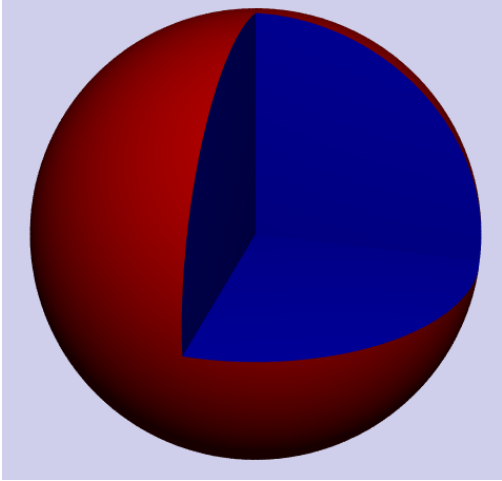
1. Constructive Solid Geometry

Thus far we have only considered some very simple shapes that we have created in POV-Ray (e.g., spheres, triangles). We can however create much more interesting objects by combining some of these simple shapes. This is what we term **constructive solid geometry**. As an example, you may be wondering how the cross-section of the Earth shown in Lecture 15 was drawn? The answer you will shortly see is quite simple.

The following table outlines the basic operations we may perform using a sphere and a box. First let's imagine the simple case where I create a sphere and a box that are overlapping:

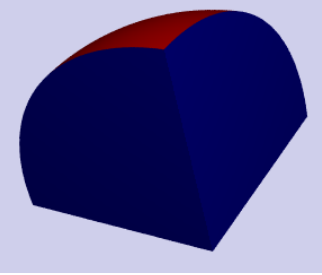
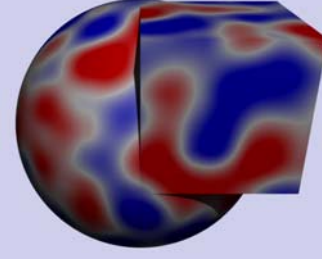
<pre> camera { location <0,5,-10> look_at <0,0,0> } light_source { <1,10,-10> color rgb <1,1,1> } background { color rgb <0.812,0.812,0.922> } sphere { <0,0,0>, 3 pigment {color rgb <1,0,0>} } box { <0,0,0> <3,3,-3> pigment {color rgb <0,0,1>} finish {phong 0.8} } </pre>	
---	---

Now, I could subtract the box away from the sphere using the difference command:

<pre> difference { sphere { <0,0,0>, 3 pigment {color rgb <1,0,0>} } box { <0,0,0> <3,3,-3> pigment {color rgb <0,0,1>} finish {phong 0.8} } rotate <0,15,0> } </pre>	
--	--

Voila, now I have a cross-section I can be proud of! Well, almost – we can now think of how do we drape imagery onto the cross section!

Other options are:

<p>Intersection</p>	<pre> intersection { sphere { <0,0,0>, 3 pigment {color rgb <1,0,0>} } box { <0,0,0> <3,3,-3> pigment {color rgb <0,0,1>} finish {phong 0.8} } rotate <20,210,0> } </pre>	
<p>Merge</p>	<pre> merge { sphere { <0,0,0>, 3 } box { <0,0,0> <3,3,-3> } pigment { bozo color_map {Red_Blue}} } </pre>	

In addition to these basic types of options another important option is the **union** command. With the union command we can group objects together and then apply pigments, translations, rotations, etc. onto the group of objects at once. Consider the next example where we create an object composed of three spheres:

```

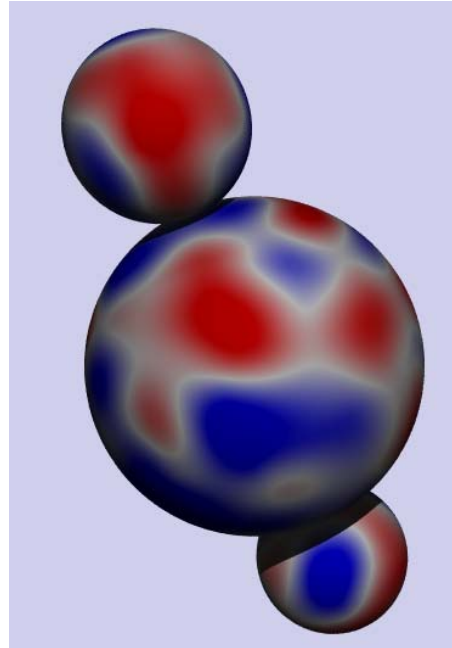
#declare TripleSphere =
union {
  // sphere #1
  sphere {
    <0,0,0>, 2
  }

  // sphere #2
  sphere {
    <0,3,0>, 1
  }

  // sphere #3
  sphere {
    <0,-3,0>, 1
  }
} // end union

// Display the object here!
object {TripleSphere
  pigment {bozo color_map{Red_Blue}}
  rotate <0,0,20>
}

```



Note that now we use **object** to display the TripleSphere object that we declared. Now we can use just a single pigment and rotate command to act on the entire object!

One can create incredibly complex objects using such constructive geometry. However, it is often easiest to draw it out on graph paper first.

2. A CSG Example

Let's take a look at what we've learned thus far in this class and put some things together into a nice cross-sectional image of the Earth. Here, we will build up a complex image by putting several small pieces together in a somewhat cookbook fashion.

2.1 - Declaring Lights and Camera Angle

It is customary to start out by declaring a camera object and light source. Let's start out as follows:

```

// Mantle Cross-Section with Image overlay
#include "textures.inc"

// ----- LIGHTS/CAMERA -----//
camera {
  location <0,6000,-15000>
  look_at <0,0,3>
}

```

```

light_source {
  <1,12000,-10000>
  color rgb <1,1,1>
}

// background
background { color rgb <0,0,0>}
// ----- END LIGHTS/CAMERA -----//

```

Note that on the second line I use the statement: `#include "textures.inc"`. There are some nice pre-defined textures and pigments that come with the standard POV-Ray download. On the class webpage there are several reference sheets that describe what these are. We include this statement here because we will use one of them in this example.

2.2 - The Inner Core

Let's start out by generating a metallic looking inner core. We can do this as follows:

```

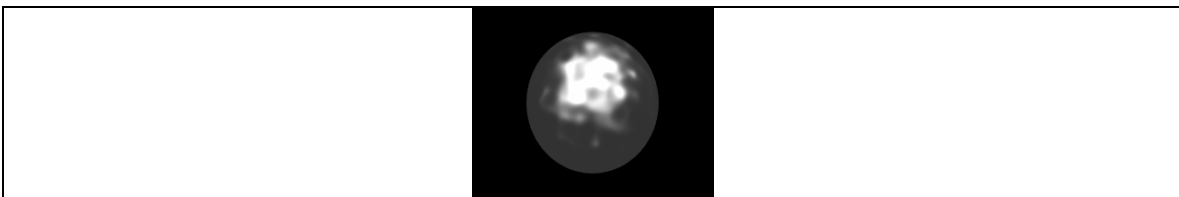
// ----- INNER CORE -----//
#declare Inner_Core =
sphere {
  <0,0,0>, 1221.5
  texture {Chrome_Metal
    normal {bumps scale 250}
    finish {phong 1.0}
  }
}
// ----- END INNER CORE -----//

```

Note that we scaled our core size to the actual size. Also, we used the `Chrome_Metal` texture from `textures.inc`. Since we declared the inner core as its own object to view it, we need to add a statement to the end of our code like:

```
object {Inner_Core}
```

At this point we should see the following when rendering our image:



2.3 - The Outer Core

Obviously, let's now add the outer core to the mix!

```
// ----- OUTER CORE -----//
#declare OC_Color =
color_map {
[0.0 color rgb <0.8,0.8,0.6>]
[0.5 color rgb <0.8,0.75,0.6>]
[1.0 color rgb <0.8,0.7,0.6>]
}

#declare Outer_Core =
difference {
  sphere {
    <0,0,0>, 3480.0
  }

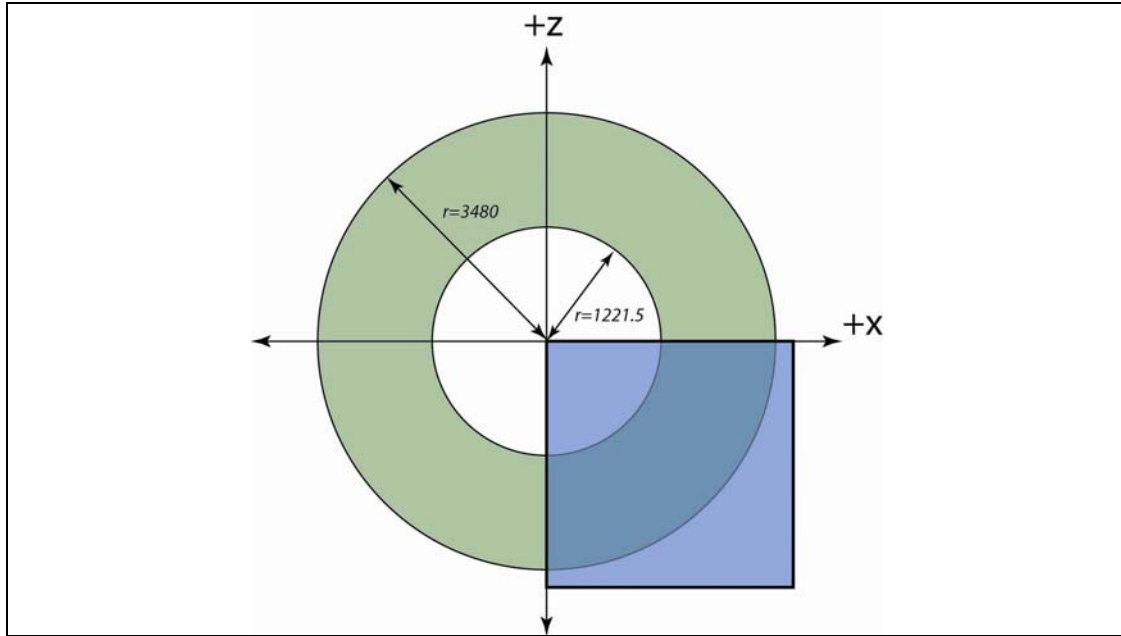
  sphere {
    <0,0,0>, 1221.5
  }

  box {
    <0,-7000,0>, <7000,7000,-7000>
  }

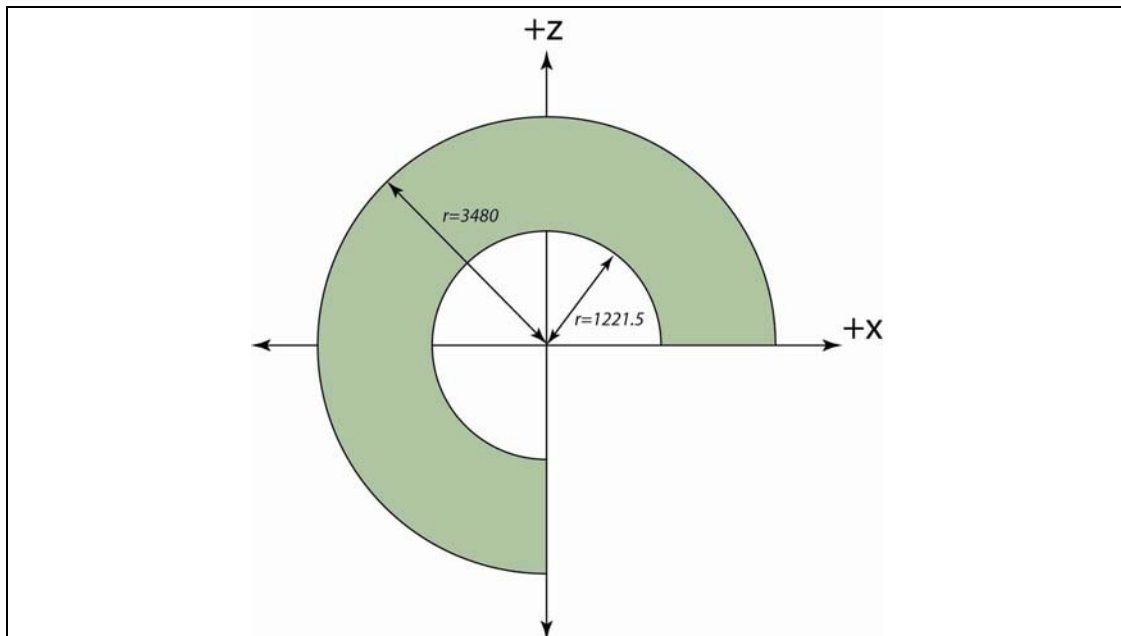
  texture {
    pigment {wood color_map {OC_Color} scale 100 turbulence 0.3}
    finish {phong 0.8}
  } // end texture
} // end difference
// ----- END OUTER CORE -----//
```

This is a little more complicated of a declaration. Let's look at what we did here:

- First, we declared a new color map that we will use to color our outer core. We defined this as **OC_Color**.
- We made the outer core by taking the difference between two spheres: (1) A sphere with radius of the Outer Core subtracted from (2) A sphere with a radius of the Inner Core. Next we subtracted off a box to generate a cross-sectional view. In plan view, we took the difference between the three following objects:



Which when differenced leaves us with a single object:



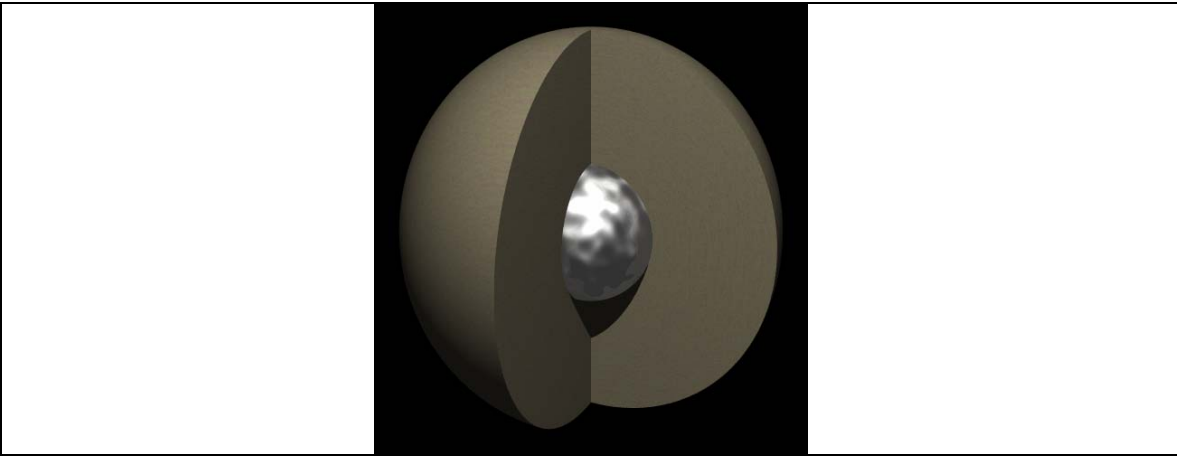
- With all of the objects combined into one, we can use a single texture statement to color it all. Here we used a **wood** pigment with a touch of **turbulence** to try and emulate a feel of mixing or stirring of the liquid outer core.

In order to view this image we now create a new object that we call Earth that is the union of the Inner and Outer Cores. Then we can rotate the object as a whole to get some better views.

```
#declare Earth =
union {
  object {Inner_Core}
  object {Outer_Core}
}

object {Earth
  rotate <0,25,0>
}
```

The resulting image at this point should look like (Render this yourself to see the fine turbulent detail in the outer core):



2.4 - The Crust and Earth Surface

It would seem to make more sense to do the mantle now. But, for this example the mantle will be a little trickier. Whereas, addition of the crust will be similar to what we did in creating the outer core, except that now we will add the Earth Surface image as a **pigment** for the outermost layer.

```
// ----- CRUST AND SURFACE -----//
#declare Crust =
difference {

  sphere {
    <0,0,0>, 6371.0
    pigment {image_map {png "./Earth/03_Earth_Land.png"}
      map_type 1 interpolate 2}}
    finish {ambient 0 diffuse 1}
  }

  sphere {
    <0,0,0>, 6331.0
    pigment {color rgb <1,0.8,0.6>}
  }
}
```

```

box {
  <0,-7000,0>, <7000,7000,-7000>
  pigment {color rgb <1,0.8,0.6>}
  finish {phong 0.8}
}

} // end difference
// ----- END CRUST -----//

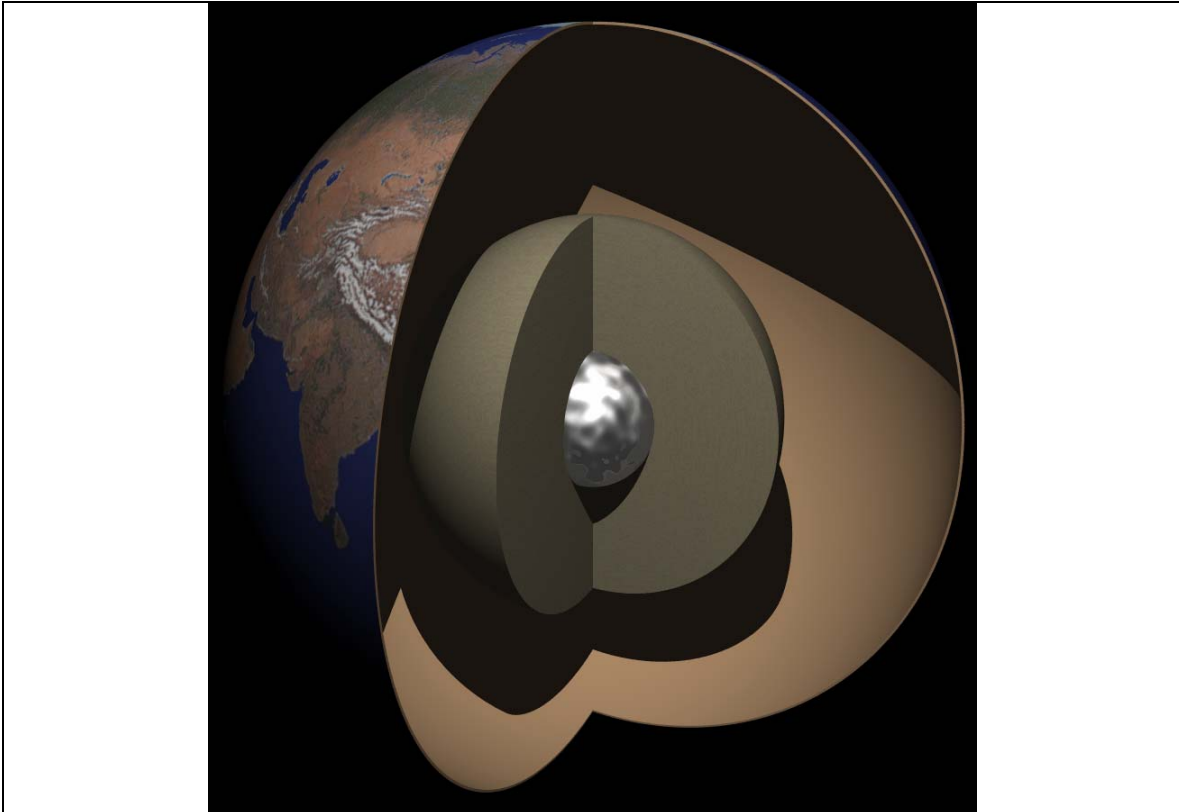
```

Don't forget we now need to declare our Earth as:

```

#declare Earth =
union {
  object {Inner_Core}
  object {Outer_Core}
  object {Crust}
}

```



Doing this we can see that we have a nice thin shell of crust now encompassing the Core. At this point Neal Adams would be pleased – a perfect rendition of his hollow Earth theory! Nonetheless, we are scientists and know better. Hence, here comes the mantle!

2.5 - The Mantle

We are going to do this in two separate pieces. The first part will have a solid color on the left-hand side of the cross-section. Here is how we would create the solid part:

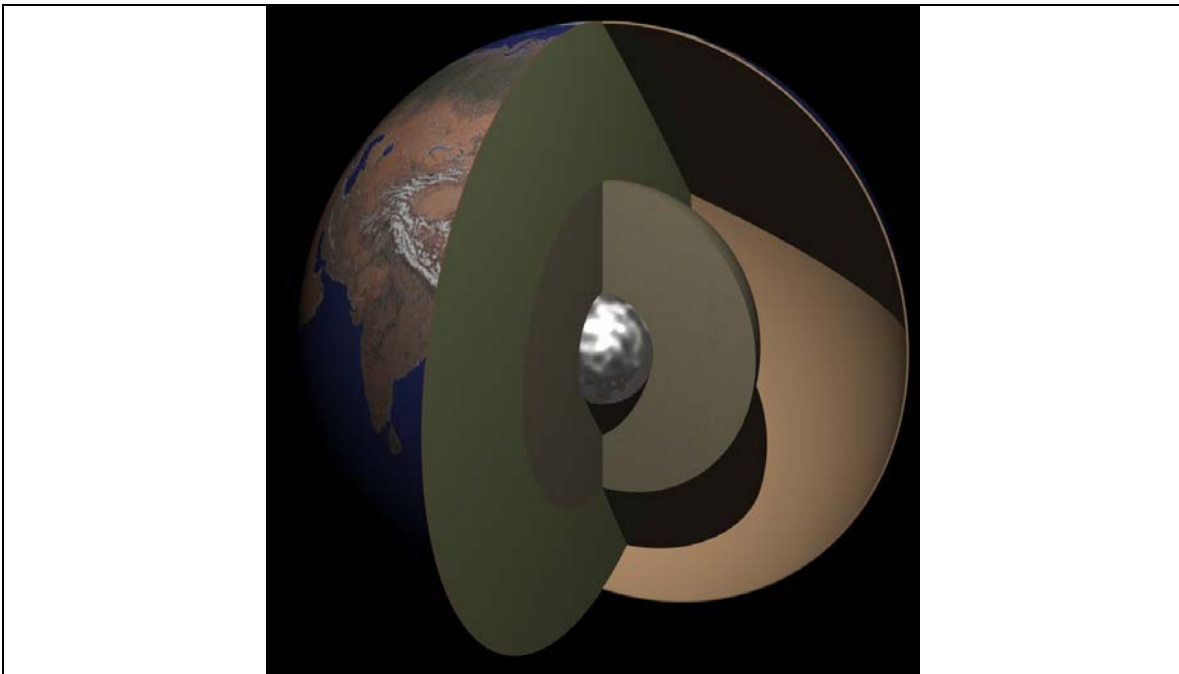
```
// ----- SOLID MANTLE -----//
#declare Mantle_Solid =
difference {
  sphere {
    <0,0,0>, 6331.0
  }

  sphere {
    <0,0,0>, 3480.0
  }

  box {
    <0,-6371,6371>, <6371,6371,-6371>
  }

  texture {
    pigment {color rgb <0.82,0.87,0.611>}
    finish {phong 0.8}
  }
} // end difference
// ----- END SOLID MANTLE -----//
```

If you just rendered this you would now get an image like:



Now, on the right-hand side of the cross-section let's drape some imagery. On the website material for this lecture there is an image file: [mantle_cartoon.png](#). We can drape this just into the mantle portion:

```
// ----- MANTLE IMAGE -----//
#declare MantleImage =
difference {

  sphere {
    <0,0,0>, 6331.0
    pigment {color rgb <0.82,0.87,0.611>}
  }

  sphere {
    <0,0,0>, 3480.0
    pigment {color rgb <0.82,0.87,0.611>}
  }

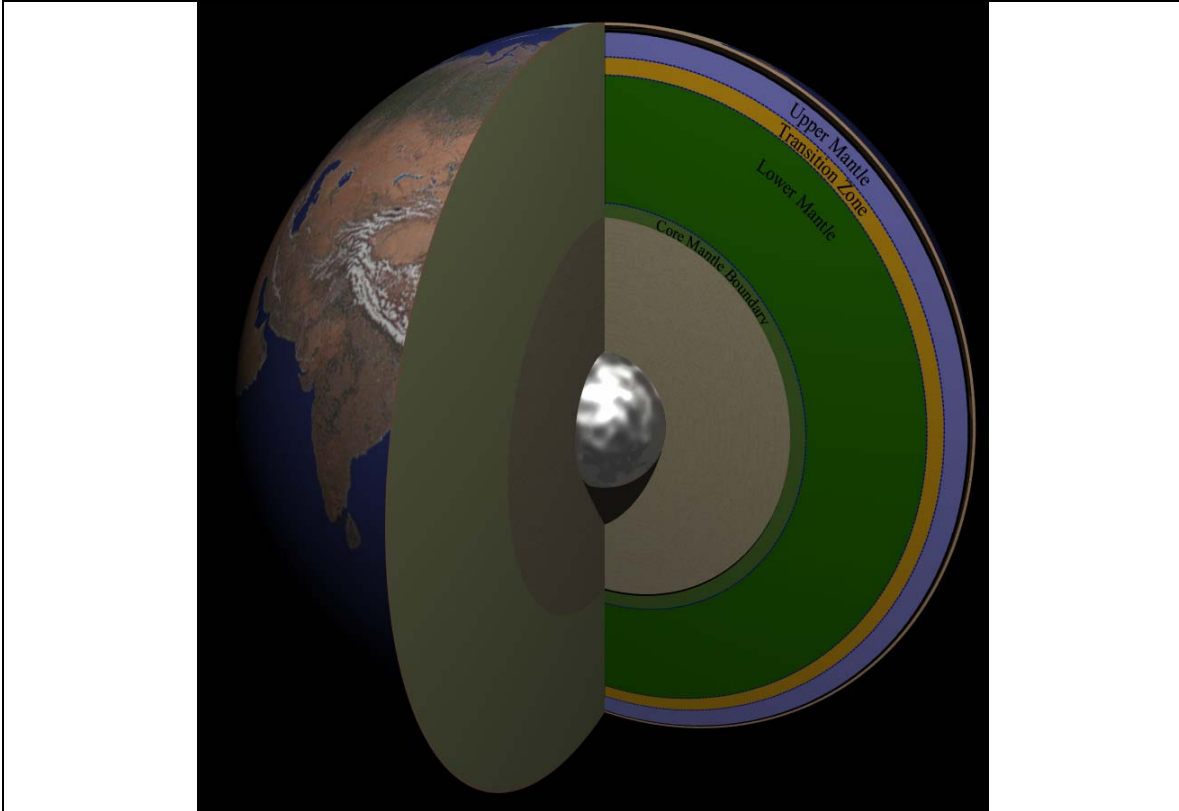
  box {
    <0,-6371,0>, <6371,6371,-6371>

    translate <0,6371,0>
    pigment {image_map {png "./mantle_cartoon.png"
      map_type 0 interpolate 2}
      scale <6331,6331*2,1> rotate<0,0,0>}
    translate <0,-6371,0>
  } // end box
} // end difference
// ----- END MANTLE IMAGE -----//
```

Combining all of our elements:

```
#declare Earth =
union {
  object {Inner_Core}
  object {Outer_Core}
  object {Mantle_Solid}
  object {MantleImage}
  object {Crust}
}
```

We now have the following image:



2.6 - Final Touches

At this point there isn't much else to do. But, one can always play around more and more. For example, adding lights, changing lights, etc. Here, I am going to add some information on our crustal layer so that we also have a sea surface and clouds. Below is some rather complicated looking code (but not too bad if you just look at it line by line) that adds in seas and clouds. Here these lines replace the former line that just added the sphere overlain by the Earth Land file. Some of this was modified from material I found on an old website by Constantine Thomas. I can no-longer find the original site, so I apologize to the original author of this union script for not linking to his site.

```
// seas, land surface, and clouds
union {

    sphere {
        <0,0,0>, 6371.0
        pigment {image_map {png "./Earth/03_Earth_Seas.png"
            map_type 1 interpolate 2 }}
        finish {ambient 0 diffuse 1 specular 0.5 roughness 0.01}
    }
}
```

```

sphere {
<0,0,0>, 6381.0
pigment {image_map {png "./Earth/03_Earth_Land.png"
  map_type 1 interpolate 2 transmit 2, 1.0 }}
finish {ambient 0 diffuse 1}
}

#declare Indexes = 256; // number of entries in the color map
#declare T = 5.5; // controls how fast the clouds become
// opaque towards the center
// T=1 -> linear
// T<1 -> less transparency on the edges
// T>1 -> more transparency on the edges

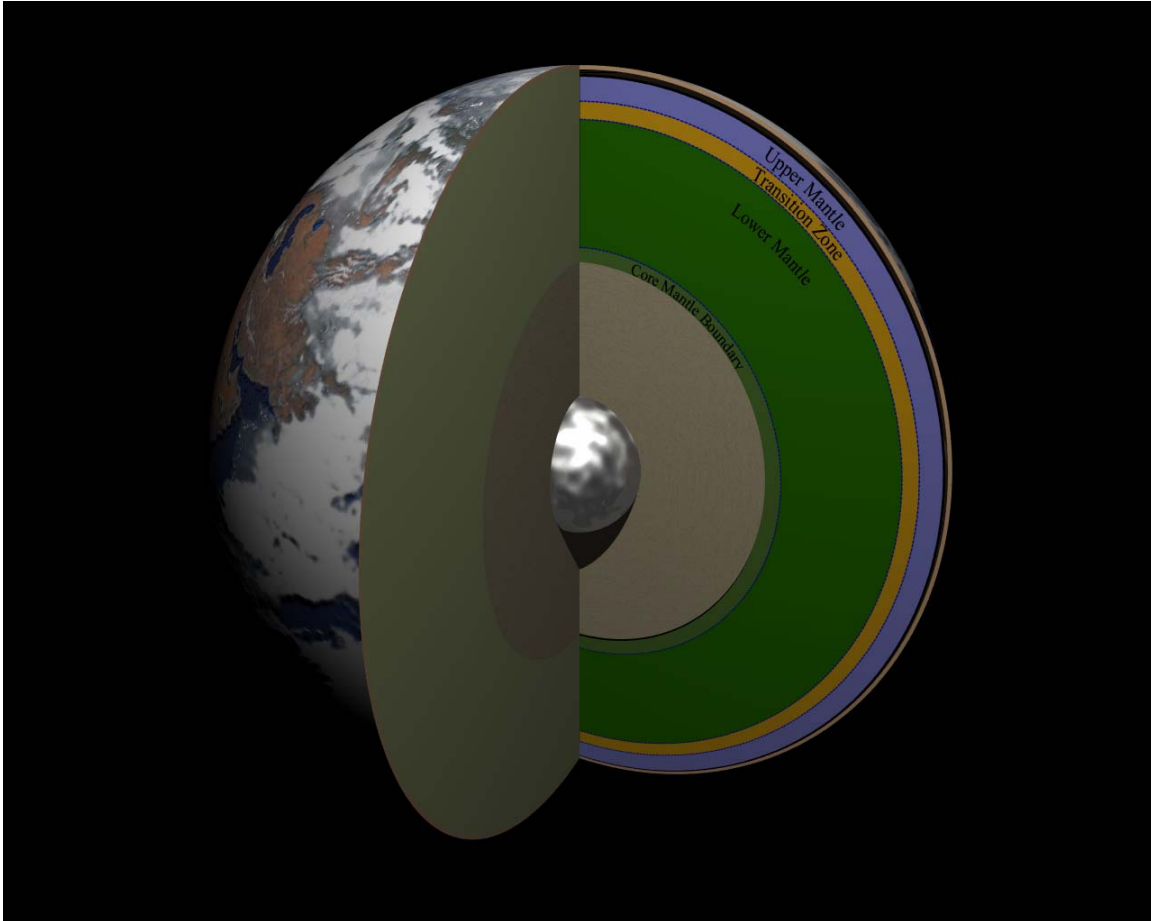
sphere {
<0,0,0>, 6391.0
pigment{image_map {png "./Earth/03_Earth_Clouds.png"
  map_type 1 interpolate 2
  #declare n=0;
  #while (n < Indexes)
  transmit n,1-pow(n/(Indexes-1),T)
  #declare n=n+1 ;
  #end
}
}

normal {bump_map {png "./Earth/03_Cloud_BumpMap.png"
  map_type 1 interpolate 2 bump_size 0.5}}
finish {ambient 0 diffuse 1}
}

} // end union

```

The final rendered image is shown below.



There are many, many other useful things we can do with POV-Ray. One important thing I have not discussed is volume rendering. Perhaps in the future I will post a lecture on this topic as well.

4. Homework

There is no assigned homework for this lecture. Just have fun playing with POV-Ray.