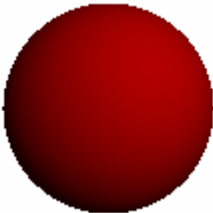

L16 – Povray - Part 2

1. Pigment

In the previous lecture we have briefly introduced pigments. This section should tell you pretty much everything you need to know about them.

Solid Colors

Our previous examples showed us how to make objects solid colors. To remind us, let's draw a sphere and make it red:

<pre> camera { location <0,0,-10> look_at <0,0,0> } light_source { <5,5,-15> color rgb <1,1,1> } background { color rgb <1,1,1> } sphere { <0,0,0>, 4 pigment {color rgb <1,0,0>} } </pre>	
---	---

We just declared our pigment to be a color with the statement `color rgb`.

Color Maps

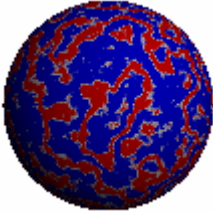
Many of the pigment options that we can use are best used when combined with a color map. The concept of a color map in POV-Ray is quite similar to that in GMT. Only we usually define the range of colors to be between 0 and 1 as many of the pigment options act on the 0-1 range of color maps. Below is an example of how to define a color map in POV-Ray.

```

// Red-to-Blue through White Color Map
color_map {
  [0.0 color rgb <1,0,0>]
  [0.3 color rgb <1,0,0>]
  [0.5 color rgb <1,1,1>]
  [0.7 color rgb <0,0,1>]
  [1.0 color rgb <0,0,1>]
}

```

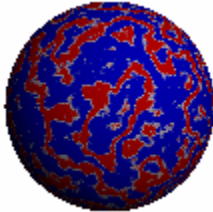
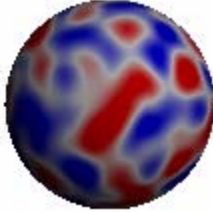
As an example of how to use this we can replace the sphere command we used in the above example by:


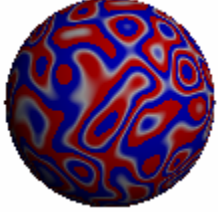
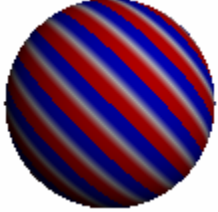
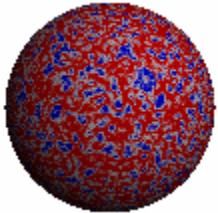


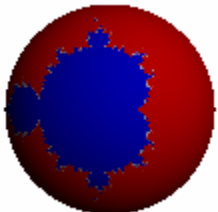
<pre>#declare Red_Blue = color_map { [0.0 color rgb <1,0,0>] [0.3 color rgb <1,0,0>] [0.5 color rgb <1,1,1>] [0.7 color rgb <0,0,1>] [1.0 color rgb <0,0,1>] } sphere { <0,0,0>, 4 pigment { agate color_map {Red_Blue} } }</pre>	
--	--


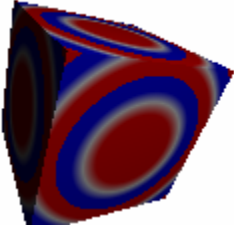
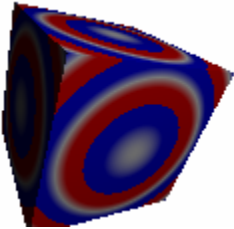

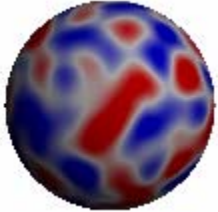

There are three main points to be made from the above example:

- We used the special pigment **agate** that gives a swirly *agate-like* appearance to our sphere.
- The colors that the **agate** function used were defined by the **color map**.
- We can use **#declare** to give names to specific objects (in this case a color map) so we don't have to re-type them over and over if we want to use them more than once.

The next table shows the different ways that we can affect the color table.

Pigment	Syntax	Example	
Agate	<pre>pigment { agate }</pre>	<pre>pigment { agate color_map {Red_Blue} }</pre>	
Bozo	<pre>pigment { bozo }</pre>	<pre>pigment { bozo color_map {Red_Blue} }</pre>	

Checker	<pre>pigment { checker color color-a color color-b }</pre>	<pre>pigment { checker color rgb <1,0,0> color rgb <1,1,1> }</pre>	
Frequency	<p>Frequency controls how many times the color map is used over the 0.0 to 1.0 range. E.g., Frequency 2 will cause the color map to repeat itself 2 times.</p>	<pre>pigment { bozo color_map {Red_Blue} frequency 5 }</pre>	
Gradient	<pre>pigment { gradient <x,y,z> }</pre> <p>The vector in this command is normal to the direction of the gradient.</p>	<pre>pigment { gradient <1,1,0> color_map {Red_Blue} }</pre>	
Granite	<pre>pigment { granite }</pre>	<pre>pigment { granite color_map {Red_Blue} frequency 5 }</pre>	
Hexagon	<pre>pigment { hexagon color color-a color color-b color color-c }</pre>	<pre>pigment { hexagon color rgb <1,0,0> color rgb <1,1,1> color rgb <0,0,1> }</pre>	
Leopard	<pre>pigment { leopard }</pre>	<pre>pigment { leopard color_map {Red_Blue} frequency 2 }</pre>	
Mandel	<p>Creates a pattern that looks like the Mandelbrot set. The number after mandel states how many iterations to calculate in making the pattern.</p>	<pre>pigment {mandel 50 color_map {Red_Blue} scale 2.5 }</pre>	

<p>Marble</p>	<pre>pigment { marble }</pre>	<pre>pigment {marble color_map {Red_Blue} }</pre>	
<p>Onion</p>	<pre>pigment { onion }</pre> <p>Note that the onion command doesn't work well with the sphere (the sphere would be only one color)</p>	<pre>box { <-3,-3,-3>, <3,3,3> pigment { onion color_map {Red_Blue} } rotate <-30,-30,0> }</pre>	
<p>Phase</p>	<p>Offset's the phase of the color map.</p> <p>Range: 0.0 to 1.0</p>	<pre>box { <-3,-3,-3>, <3,3,3> pigment { onion color_map {Red_Blue} phase 0.5 } rotate <-30,-30,0> }</pre>	
<p>Radial</p>	<pre>pigment { radial }</pre>	<pre>pigment { radial color_map {Red_Blue} } rotate <0,60,0></pre>	
<p>Spotted</p>	<pre>pigment { spotted }</pre>	<pre>pigment {spotted color_map {Red_Blue} }</pre>	
<p>Wood</p>	<pre>pigment { wood }</pre> <p>If you really want wood, then better not use the Red_Blud color table.</p>	<pre>pigment {wood color_map {Red_Blue} }</pre>	

We can also modify our pigment commands with **turbulence**. What does turbulence do? You guessed it. It stirs things up a bit. To see its use consider the next two examples:

<pre> pigment {wood color_map {Red_Blue} turbulence 0.2 } </pre>	
<pre> pigment {wood color_map {Red_Blue} turbulence <0.2,0,0> } </pre>	

Here we applied the **turbulence** command to our wood example from above. The first example applied equal amounts of turbulence (defined in the range from 0.0 to 1.0) in all directions. The second example only applied turbulence in the x-direction.

Now, perhaps you can see why we waited until the 2nd lecture to give a run down on the basics of the pigment command. Namely, it took almost 5 pages to do it.

2. Height Fields

Height fields are one of the most useful objects for visualizing real data in POV-Ray. To demonstrate how to use height fields let's look back at an example of visualizing elevation data. **Lecture #7** discussed making GMT **.grd** files from Digital Elevation Model (DEM) data. In the material for the current lecture I include a **.grd** file generated for the Zion National Park area. Download that file now as we will use it to generate a height field in POV-Ray.

With GMT and ImageMagick we can make a plot of the **Zion.grd** file and then convert this image to a **.gif** file. The following script shows an example of how to do this.

```

#!/bin/csh

# Set input/output
set Gridfile = Zion.grd           #name of input .grd file to use
set output   = Zion_Height.gif    #name of output file. Must contain
                                  # .gif extension

# Set map boundaries
set xmin = 300000
set xmax = 339995
set ymin = 4100005
set ymax = 4160000

```

```

# Make color palette table to utilize the maximum range of elevations
# Note that image should be gray scale.  Lowest elevations should
# be colored black, ranging to highest elevations white
gmtset COLOR_BACKGROUND 0/0/0
gmtset PAGE_COLOR 0/0/0
set cscale = `grdinfo -T100 $Gridfile`
makecpt -Cgray -M -Z $cscale >! height.cpt

# Generate the postscript file
grdimage $Gridfile -R${xmin}/${xmax}/${ymin}/${ymax} -Jx1:250000 \
  -Cheight.cpt -Qs -E300 -P >! Height.ps

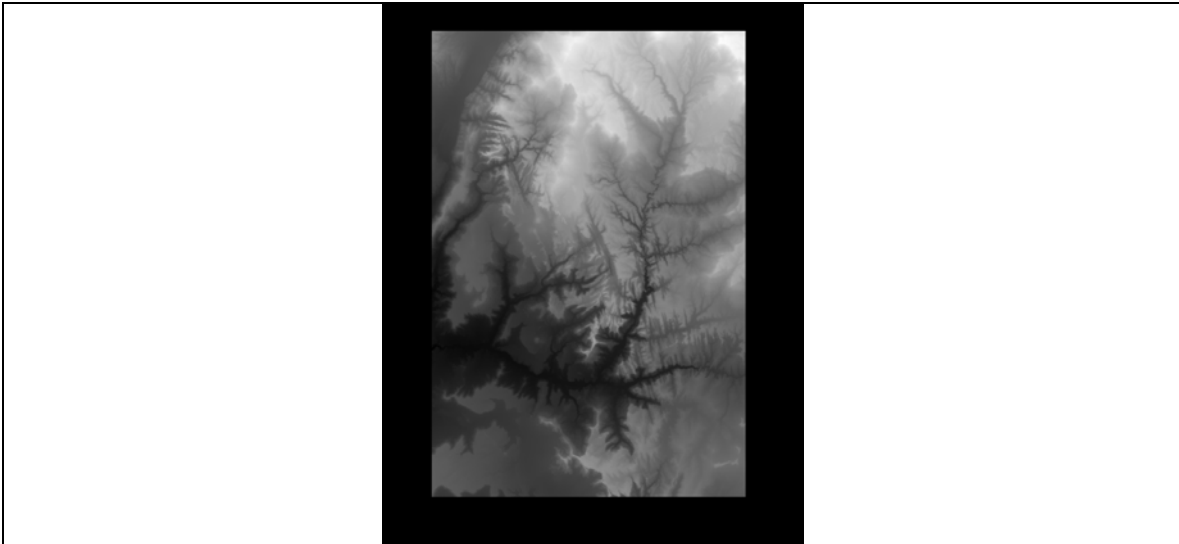
rm height.cpt

# Make .gif file for use with POV-Ray height fields
convert Height.ps $output

# Show the .gif height field
rm Height.ps
display $output

```

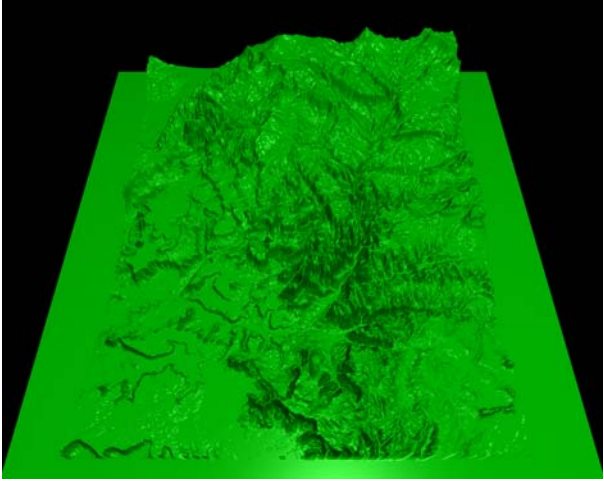
Running this script will generate a **.gif** image of Zion National Park that looks like this:



The most important points about this process are:

- To make a height field in POV-Ray we need a **.gif** image. Other formats are also supported but the **.gif** file format is simple to deal with.
- The **.gif** format allows for a range in gray from 0-255, hence only 256 distinct elevations are allowable. Hence, we wish to maximize our color palette table to include just the range of elevations in the **.grd** file.
- The **.gif** file should be in gray scale. Noting that the minimum elevations should be color coded black and ranging up to white for the maximum elevations.

Now that we have a [.gif](#) file as created in the above example it is simple to visualize this in POV-Ray. The following script shows the simplest (no-frills) way to do this:

<pre> camera { location <0,5,0> look_at <0,0,3> } light_source { <1,10,0> color rgb <1,1,1> } height_field { gif "Zion_Height.gif" smooth pigment {color rgb <0,1,0>} } finish {phong 0.4} scale <6,0,6> translate <-3,0,0> } </pre>	
--	--

An important note to make is that with POV-Ray we are not limited in where we can place our camera location whereas in GMT our camera location is constrained to be off at infinity. So, for example, we can zoom in to very unique views.

In addition, one can now play with all of POV-Ray's unique finishes, textures, pigments and so on to create very realistic views, including the addition of clouds and other objects. It is up to you as an artist at this point.

As another point, almost anything can be turned into a height field. For example, open up Adobe Photoshop or ImageReady and create a black and white file with some text. For example, I created the following [.gif](#) file where I just typed my name:





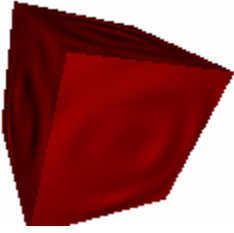
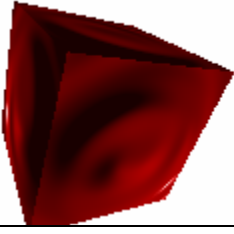
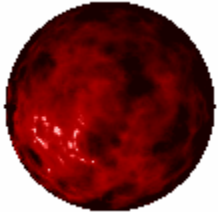
Note that if saving a [.gif](#) file in Photoshop or ImageReady you need to set **Reduction** to **Grayscale** or it might not work properly. Now we can use this as a height field in POV-Ray:



As a final note, what you do with this type of height field object is pretty much up to your creativity. Once you get used to playing around with POV-Ray you stick these height fields on objects and do all kinds of interesting things (see the next section on bump maps)!

3. Normals

The normal attribute modifies the normal surface vectors to give an appearance of bumpiness. The next table shows the basic ways this can be done.

Normal	Syntax	Example	
Bumps	<pre>normal {bumps bump_size}</pre> <p>bump_size can range from 0.0 (no bumps) to 1.0 (maximum size)</p>	<pre>sphere { <0,0,0>, 4 pigment {color rgb <1,0,0>} normal {bumps 1.0} finish {phong 0.8} }</pre>	
Dents	<pre>normal {dents dent_size}</pre> <p>dent_size can range from 0.0 (no dents) to 1.0 (maximum size)</p>	<pre>normal {dents 1.0}</pre>	
Ripples	<pre>normal {ripples size}</pre> <p>size takes the standard input range from 0.0 to 1.0.</p>	<pre>box { <-3,-3,-3>, <3,3,3> pigment {color rgb <1,0,0>} normal {ripples 1.0} rotate <-30,-30,0> finish {phong 0.8} }</pre>	
Waves	<pre>normal {waves size}</pre> <p>size takes the standard input range from 0.0 to 1.0.</p>	<pre>normal {waves 1.0}</pre>	
Wrinkles	<pre>normal {wrinkles size}</pre> <p>size takes the standard input range from 0.0 to 1.0.</p>	<pre>normal {wrinkles 1.0}</pre>	

Bump Maps

One of the most interesting way to affect the normals is to use the **bump_map** attribute. This is like a cross between an **image_map** and a **height_field**, allowing us to extrude image information onto an object. The next example shows how to do this simply with the **.gif** image of my name that we generated earlier.

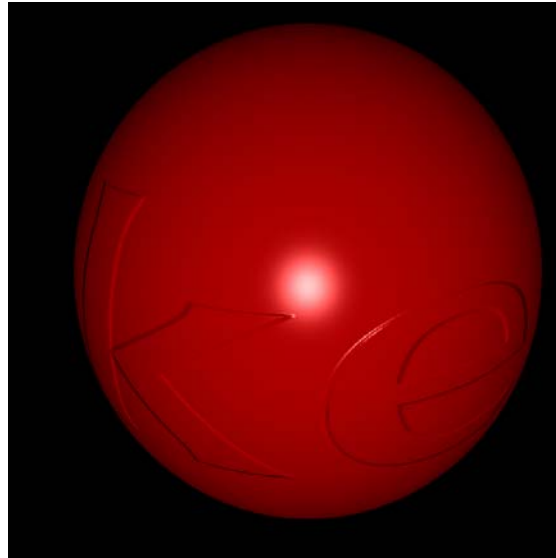
```

camera {
  location <0,1,-10>
  look_at <0,0,0>
}

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

sphere {
  <0,0,0>, 3
  pigment {color rgb <1,0,0>}
  normal{
    bump_map {
      gif "Mike_1.gif"
      bump_size 1
      map_type 1
      interpolate 2
    }
  }
  finish {phong 0.8}
  rotate <-20,0,0>
}

```



4. Homework

(1) Generate a height field image of some region of Utah (e.g., Antelope Island, Twin Peaks Wilderness, King's Peak area, etc.). Generate two images: (a) an aerial overview and (b) a zoomed in view to a specific region of interest. Use the various pigment, finish, and normals modifiers we have discussed thus far to generate the image. As an example, below is an image I created looking at the Wasatch Front (Mt. Olympus is in the center of the frame).

