LAB 4 — COLOUR AND TEXTURE

16/10/2016
GOALS FOR THE LAB

The goal of this lab is to get used to using colours and textures in POV-Ray. Create a scene similar to the one below.

- A yellow sphere
- An orange sphere
- A brown sphere
- A box with an image_map texture (any image of your choice)
- A plane with checkerboard texture
STARTING CODE

Suggested camera and lighting for this lecture (feel free to change this if you’re comfortable enough)

```plaintext
light_source
{
  <4, 6, -8> //right up and back a bit
  rgb<1,1,1>
}

camera
{
  location <0, 6, -8> //up a bit and back a bit
  look_at <0,0,0> //facing the centre of the world
}
```
For this lab let's always use the following sphere definition (sphere starts off centred at origin <0,0,0> of radius 1):

- we will only change the pigment
- and move the sphere using transforms (translate)

```plaintext
sphere
{
  <0,0,0> 1
  pigment
  {
    rgb<1, 0.6, 0>
  }
  translate<-1.5,1,0>
}
```
PIGMENT COLOURS

We need to know how to combine red, green and blue,

This may be apparent from the spectral density functions. (In practice this can be worked out with some experience or trial-and-error)

**Yellow**

- “Lots of green and lots of red”

  \[
  \text{rgb} < 1, 1, 0>
  \]

**Orange**

- “Some green, lots of red”

  Try to work this out yourself

**Brown**

- “Some blue, a bit more green, and even a bit more red”

  Try to work this out yourself
For the cube, try to use a standard box as follows and change pigment and translate the box.

```plaintext
box
{
    <-1, -1, -1>, <1, 1, 1>
    pigment
    {
        image_map
        {
            jpeg "rocks.jpg"
        }
    }
    translate <1.5, 0, 0>
}
```

Instead of an rgb vector, load an image file to assign colours:

- Get any image file (e.g. from the web)
- This needs to be in one of the following formats: gif, tga, iff, ppm, pgm, png, jpeg, tiff, sys
- Save this in the same folder as the POV file you are working with

Specify the filetype and file name in an `image_map` structure within the `pigment` block.

- File name
- File type
PROCEDURAL TEXTURE

For this use a simple “ground plane” and apply a texture to it

Direction of “NORMAL” to plane  Distance from origin

plane
{
  <0,1,0>, -1
pigment
{

???
}
}

Now you need to specify the procedural map using one of pov-ray’s procedural functions

♫ e.g. checker, marble, wood, agate
♫ You can also specify some colours
♫ See:
  http://library.thinkquest.org/3285/language/pigment.html
ADDITIONAL HELPER NOTES
The `image_map` object lets you load an image file to use as a texture.

```plaintext
sphere
{
  <0,0,0>, 2
pigment
{
  image_map
  {
    tga //filetype
    "mytexture.tga" //filename
  }
}
}
```
RECAP: PLANES IN POV-RAY

plane
{
  <0, 1, 0>, 4
  pigment { rgb <0.5, 0.5, 0.5> }
}

Direction of normal (N.B. Length of vector doesn’t matter here)

Distance from origin (along normal direction)

The normal to a surface is a line that is orthogonal (90 degrees) to it. Kind of like the pin on a thumb-tack.

Other Objects: http://www.povray.org/documentation/view/3.6.1/273/
**IMAGE MAP**

```
sphere
{
  <0,0,0>, 2
}

pigment
{
  image_map
  {
    jpeg "texture.jpg"
  }
}
```

**Note:**
- By default, the image is scaled to a square 1x1 in dimension, projected orthogonally on the x-y plane, and repeats itself.
- You can apply transforms to the texture to change this.
Here, the texture has been scaled by 2 along X and Y directions so that the texture covers more of the sphere.
PROCEDURAL TEXTURES (IN POV-RAY)

- checker
- marble (???)
- wood (???)
- agate
Some built-in procedural texture patterns, e.g., granite, agate, marble, etc., are available in POVRay. We just specify the texture by name. And then provide a color map (or palette) that the function uses to generate patterns.

For more types, see the POVRAY Online tutorial [http://library.thinkquest.org/3285/language/pigment.html](http://library.thinkquest.org/3285/language/pigment.html)
EXAMPLE: CHECKER

Simple example of a procedural texture that takes two colours and creates a pattern.

```
pigment
{
  checker
  Blue
  White
}
pigment
{
  checker
  pigment { agate }
  pigment { Black }
}
```
Note that you can actually modify the texture i.e. move, stretch, rotate.

- Achieved by putting a transform inside the pigment object
- Quite useful in practice.

Some advanced tasks to try:

- Try transforming the `image_map`
- Try using other procedural maps like `wood` and `marble`. Can you resize and deform the procedural map?
- Can you change the colours used in the procedural map?
MODIFYING TEXTURES

Turbulence

Color map

Scaling (non-uniform)
You are not required to use all of these notes for today’s lab but they may be useful for future reference.
Some built-in procedural texture patterns e.g. granite, agate, marble etc.. are available in POVRay. We just need to specify the texture by name. And then we can provide a colour map that the function uses to generate patterns.

For more types, see the POVRAY Online tutorial: http://library.thinkquest.org/3285/language/pigment.html

```plaintext
sphere
{
    <0, 0, 0> 1.5

pigment
{
    agate //the texture name
    color_map //swatch of colours
    {
        [0.00 color rgb <0.75, 0, 0>]
        [0.33 color rgb <0.75, 0.75, 0>]
        [0.50 color rgb <0, 0.75, 0>]
        [0.66 color rgb <0, 0, 0.75>]
        [1.00 color rgb <0.75, 0, 0>]
    }
}
```
Colours in the colour map are used by povray to generate the actual texture

- Think of this as a palette or swatch
- However the colours are stored in a certain position (somewhat like defining a gradient pattern in some image editors)

Defining colour maps

- Specify a list of colours (as many as you want)
- Attach them somewhere between a position of 0.0 and 1.0
- The intermediate values are interpolated (appropriate values in between are computed)
**COLOR_MAP**

Colours in the colour map are used by povray to generate the actual texture

- Think of this as a palette or swatch
- However the colours are stored in a certain position (somewhat like defining a gradient pattern in some image editors)

Defining colour maps

- Specify a list of colours (as many as you want)
- Attach them somewhere between a position of 0.0 and 1.0
- The intermediate values are interpolated (appropriate values in between are computed)
The agate texture with the colour maps shown on the previous slide.

```
color_map
{
  [0.00 color rgb <1, 1, 1>]
  [1.00 color rgb <0, 0, 0>]
}
```

```
color_map
{
  [0.0 color rgb <.8, .7, .5>]
  [0.3 color rgb <.21, .21, .21>]
  [0.5 color rgb <.5, .51, .51>]
  [0.6 color rgb <.41, .2, .0>]
  [0.8 color rgb <1, 1, 1>]
  [1.0 color rgb <.8, .7, .5>]
}
```
BUILT IN PATTERNS

agate
bozo
wood
marble
brick
checker
hexagon
radial
mandel
AGATE

pigment

{
  agate
color_map

  {
    [0.0 rgb <1, 0, 0>]
    [0.5 rgb <0, 1, 0>]
    [1.0 rgb <0, 0, 1>]
  }
}

Random marble like patterns

Default colour_map (if you don’t specify a colour map, it will use these colours):
pigment
{
  //rgbf <1, 1, 1, 0>
  bozo
  color_map
  {
    [0.0 rgb <1, 0, 0>]
    [0.5 rgb <0, 1, 0>]
    [1.0 rgb <0, 0, 1>]
  }
}

A smooth, random noise function, traditionally used with turbulence to create clouds.
Concentric circles modeling growth rings in wood
pigment
{
  wood

  color_map
  {
    [0.0 rgb <1, .81, .5>]
    [1.0 rgb <.7, .51, .30>]
  }
}

pigment
{
  wood
turbulence 1
color_map
  {
    [0.0 rgb <1, .81, .5>]
    [1.0 rgb <.7, .51, .30>]
  }
}
pigment
{
  wood
turbulence 1
color_map
{
  [0.0 rgb <1, .81, .5>]
  [1.0 rgb <.7, .51, .30>]
}
scale <2, .2, .2>
}
pigment
{
  marble
color_map
{
  [0.0 rgb <1, 0, 0>]
  [0.5 rgb <0, 1, 0>]
  [1.0 rgb <0, 0, 1>]
}
}

A very simple repeating gradient pattern
pigment
{
  marble
turbulence 1
color_map
{
  [0.0  rgb <.5, .4, .2>]
  [0.5  rgb <.7, .6, .4>]
  [1.0  rgb <.8, .8, .6>]
}
scale <2, 2, 2>
}
pigment
{
    checker
    Blue
    White
}

pigment
{
    checker
    pigment { agate }
    pigment { Black }
}
#include "colors.inc"

pigment
{
    brick
    pigment { White }
    pigment { Red }
}

pigment
{
    brick
    pigment { White }
    pigment
    {
        marble
        turbulence 1
        scale <2, .2, .2>
    }
}