

# Lab: Color My World

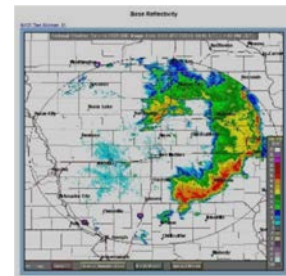
This lab is due in just under one week – by 11:59 pm on the day before your next lab day. All the files should be submitted as previously instructed. One of the files should be an executable called “colorIt.py”. There should also be two colormap files, as described below, and possibly an image file or two showing your best results. A README file is always a welcome addition, especially to explain the contents of the collection of files and to explain any procedures for their use.

## Introduction

An important part of Scientific computing, and particularly in HPC (or any field working with "Big Data") is the visualization of results. With HPC there is often a huge amount of data produced. One of the easiest ways to comprehend large quantities of data is with pictures. (“One picture is worth a thousand words” seems to be really rather an understatement.) A large collection of numbers may be the most accurate result from some HPC calculations, but it can be hard to understand what those numbers are telling you without seeing a picture. Often those pictures don't have an actual physical visibility (what "color" are microwaves?); rather we choose a color for the display based on the numeric values; this is called a **pseudo-coloring**. You likely have seen such pseudo-colorings in radar and satellite weather data. The NOAA website (<http://www.goes.noaa.gov/ECIR4.html>) tells us:

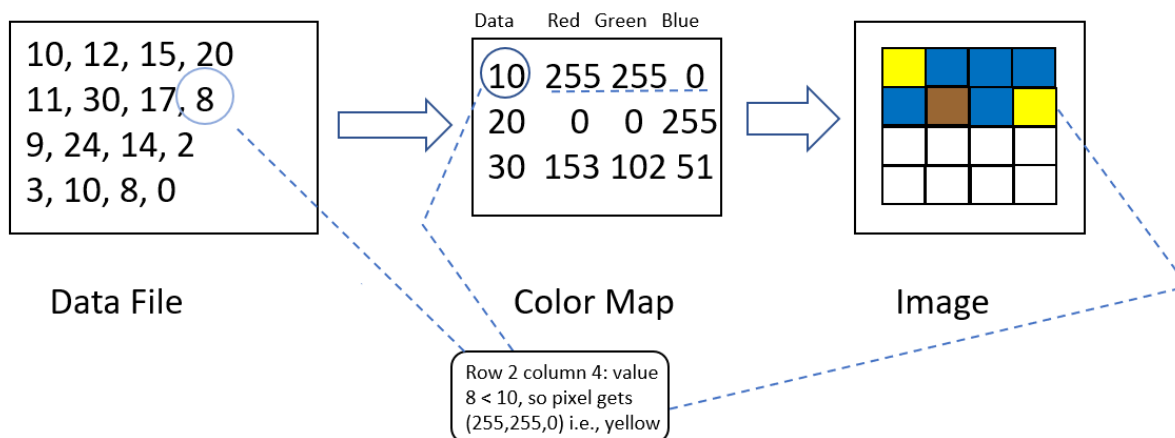
Meteorologists use color enhanced imagery as an aid in satellite interpretation. The colors enable them to easily and quickly see features which are of special interest. Usually they look for high clouds or areas with a large amount of water vapor.

In an infrared (IR) image cold clouds are high clouds, so the colors typically highlight the colder regions. The bar on the right side of the image indicates the pixel brightness values for the corresponding color. The intensity value represents emitted infrared radiation. The intensity of a pixel is recorded as a digital number (for example, in these images the numbers range from 0 to 255.)



## Color Maps

Pseudo-coloring is implemented using a **color map**. The color map is used to translate a data value into a pixel value (i.e., a color). It defines the range of data values that get mapped into which colors. Here's an example of how a color mapping might be implemented: Given a data file of numbers, and a color map to define which values get mapped to which colors, then we can create an image of those values. Each value becomes a pixel in the image; the color value of that pixel (defined by Red/Green/Blue triples) is defined in our colormap:



## Why

Why are we doing this? First, as an exercise in visualization, to demonstrate the concept of pseudo-coloring, and to understand how artificial, though important, the coloring is. This is also a support tool that will be useful for future labs when we generate data that we will want to visualize.

## Tasks

Your objective is to write a pseudo-coloring program. The input to your program, specified on the command line, should be two size parameters and the name of two .csv files (csv = Comma Separated Values). Your output should be an image displayed to the screen (and saved to a file). The two size parameters are the number of rows and the number of columns in the data file (and thus in the image). The first file (the 3<sup>rd</sup> parameter) will be the color map. It specifies the values ranges to use to map a data point value to an RGB triplet. The second file will be the datafile to be colored and converted to an image. An example invocation might look like this:

```
./colorIt.py 500 500 redMap.csv HotWater.csv
```

## Design

Here's a simple design (in pseudo code) for the program:

```
load the colormap
open the input file
open/create an image (based on the first two parameters)
for each line of the input file
    for each value, convert it to an RGB triplet (may involve scaling the
    value)
    assign the corresponding pixel that RGB value
display the image
```

One way to implement this is with the python “Image” package. Here's an example of a simplistic pixel assignment to create an image:

```
#!/usr/bin/python
# This example is based on:
# http://en.wikibooks.org/wiki/Python_Imaging_Library/Editing_Pixels
from PIL import Image

img = Image.new( 'RGB', (300,300), "black") # create a 300x300 image

pixels = img.load()           # create the pixel map
for i in range(img.size[0]):  # for every pixel:
    for j in range(img.size[1]):
        pixels[i,j] = (i, j, 100) # set a made-up color (R, G, B)

img.show()                   # to display the image
img.save('myimg.jpg') # to save the image in a file
```

You should try this out to see that it works.

Now you should write your own version that will color any given input data. You might use a similar approach, but the color value won't be based, as it is in this sample code, on the row and column indices of the image. Instead it will be based on data from a data file "mapped" (i.e., whose colors are chosen) via a colormap file.

Our colormaps will consist of an arbitrary number of lines of data, four pieces of data per line. Each line will be arranged like this: *datavalue* R-value G-value B-value

For example: 10 0 0 255  
20 0 22 225  
30 0 48 225  
...

Everything greater than the previous *datavalue* up to and including this *datavalue* maps to that (RGB) triplet. For example, if a data point in your input data was a 15, then using this particular color mapping it would be colored (0, 22, 225).

You will have to invent or hunt down a good colormap or two. One map should be a linear gradation for the values in your data file. You'll find it helpful to do a little bit of data analysis on your data file. See what range of values it has, what the min and max values are, and then construct your colormap accordingly.

Your other map should be a non-linear gradation, from 1 to 1000, one that is more weighted or detailed at the "cool" end, with distinct colors for values between 1 and 10, then less detail (fewer distinct colors) between 10 and 100, and even less between 100 and 1,000. (Hint: does this sound logarithmic?)

Once you have your colorizer(s) working, try it out on this dataset:

`http://courses.cs.xxx.edu/~mysite/mystery1.data`

which you can download with `wget` or `curl`. For example:

```
$ wget -O mystery1.data 'http://courses.cs.xxx.edu/~mysite/mystery1.data'
```

*Adjust the values in your colormap to get the most detail out of the image that you can get.*

Save the best result and submit that image along with your colormaps and your colorizer program.

## Clean Code Counts

This project will be graded on three criteria: 1) does the colorizer program work? (task completes without errors or aborts, produces a recognizable image of the correct size and shape), 2) is the image rendered well? (good level of detail), and 3) is the program well written? "Well-written" means clean, readable code, well-structured and well commented and documented.

## Extra Credit

Create another color map for black and white images and use it to color the data in `mystery3.data` (Hint: it's a medical x-ray image.)