ENGI E1006

PPM Image Format *(Thanks to Joshua Guerin, Debby Keen, and SIGCSE's Nifty Assignment session)*

The PPM (or Portable Pix Map) image format is encoded in human-readable ASCII text. For those of you who wish to have the experience of reading real documentation, the formal image specification can be found here.

Sample ppm file:

```
P3
4 4
255
0  0  0    100 0  0        0  0  0     255    0
255
0  0  0     0 255 175      0  0  0      0     0
 0
0  0  0     0  0  0        0 15 175     0     0
 0
255 0 255  0  0  0        0  0  0     255
 255 255
```

Image Header

You can think of the image as having two parts,

a **header** and a **body**. The **header** consists of four entries:

```
P3
4 4
255
```

P3 is a "magic number". It indicates what type of PPM (full color, ASCII encoding) image this is. For this assignment it will always be P3.

Next comes the number of columns and the number of rows in the image (**4** x **4**).

Finally, we have the maximum color value **255**. This can be any value, but a common value is 255.

The way you see the header presented is how it should be spaced out.

Image Body

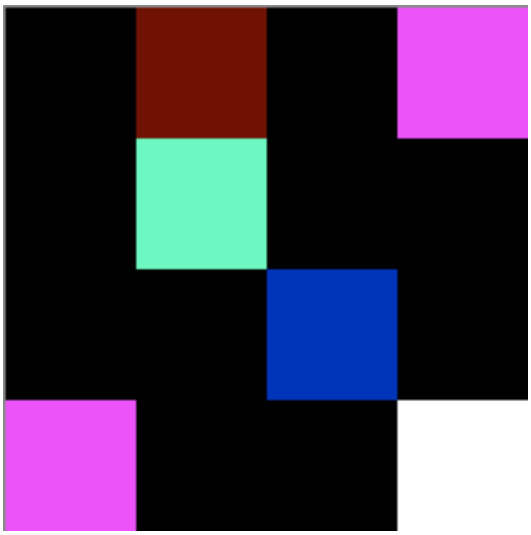
The **image body** contains the actual picture information. Each pixel of the image is a tiny, colored square. The color is determined by how much red, green, and blue are present. So, **0 0 0** is the first color of the image, which is black, and the last pixel in the image is **255 255 255**, which is white. By varying the levels of the RGB values you can come up with any color in between.

Note that color values must be separated by a space, but after than additional whitespace is ignored by the image viewer. In the sample ppm above we used additional whitespace to format the image so that it is easy for a human to understand, but the computer doesn't care if everything is on one line, if there is one line per line of the

image, or some mix.

Putting it all together

The example image above would look something like this:



Keep in mind, each square is one pixel, so the real thing is much smaller (the rendered image was blown up by 5000%).

How to view PPM files

While PPM files are easy to view as text (you can use Notepad, for instance), and easy to work with in code, they are highly inefficient. Most modern image formats use some kind of compression to make their size reasonable while preserving the image appearance. This is not to say that PPMs don't still have some life in them--one modern use for PPM is an intermediate format when converting

images from one type to another.

You may need to install a program to view these images on a your machine. Irfanview or Gimp for Windows are both suitable for Windows machines. If you have a Mac you probably already have Gimp installed or you can download it from here. These will also allow you to convert your own images to PPM so you can practice with pictures you took in the past (keep in mind that you may need to make them very small or the resulting PPM will be **quite large**!).

Your Assignment

In this assignment you will write an application in Python that takes as input a series of three similar images and outputs a new image with unwanted objects from the original series removed. The images will be provided as ppm files so all your program needs to do is edit them as text files to accomplish your goal. I have provided three images attached to this assignment:

1. tetons1.ppm

2. tetons2.ppm

3. tetons3.ppm

See if you can guess which objects should be removed from these images.

How do you remove the unwanted objects?

Notice that the unwanted objects appear in different parts of each image. This allows you to simply write a new ppm

file where each RGB value will be whatever the majority of the three images above suggests. So in this case at least 2 of the files will always have the right pixel values.

More details please....

Write a module called `effects`. In it go ahead and write a function called `object_filter` that takes as input (parameters) the file objects to filter and the name of the new file to create. The function should then create a new ppm file using the majority rules approach described above for each pixel's RGB value.

but wait, there's more....

In addition to the `object_filter` function above write a function called `shades_of_gray` that converts a color image to a black and white image. One way to convert a color image to black and white is to replace each pixel's individual RGB values with the average of the three values. So for example if a particular pixel had RGB values 100 200 300 you could change them to 200 200 200 for the black and white version.

did I mention....

Add another function called `negate_red` that will change just the RED color numbers into their "negative". That is, if the red number is low, it should become high and vice versa. The maximum color depth number is useful here. For files with maximum color depth of 255 (all of the files I have provided), if the red were 0, it would become 255; if it were 255 it would become 0. If the red were 100, it would become 155. Write `negate_green` and `negate_blue`

functions also. Your functions should work with any maximum color depth number provided it's a positive integer (so not just 255!).

Are you kidding me?

Write another function called `mirror` which will flip the picture horizontally. That is, the pixel that is on the far right end of the row ends up on the far left of the row and vice versa (remember to preserve RGB order!).

So what do you turn in?

Along with your `read_me` turn in two files. The first must be called `effects_tester.py`. In it write a main function to test your effects module. The other file is called `effects.py` which is the module containing the effects functions described above. Use the attached template for the `effects.py` module. It's important that you use this exact format as this is how we will test your code. Have the `main` function ask the user which effects they wish to try, then prompt the user for the correct number of input file names and then for the output file name. The appropriate output file should then be created.

Example session with a user:

```
Portable Pixmap (PPM) Image Editor!

Choose the effect you would like to try:

1) object_filter

2) shades_of_gray
```

```
3) negate_red

4) negate_green

5) negate_blue

6) mirror

Enter a number: 2

Enter an input file name: test.ppm

Enter name of output file: out.ppm

out.ppm created.
```

And the output file created would be called out.ppm and would have the desired effect. Your program is NOT responsible for displaying the image in the file, just for manipulating the pixels and creating an output file in the proper ppm format.

Test this with small files and large files. You can check to see if works by viewing the images and by viewing the text files with a text editor.

The following example ppm files are attached to this assignment on coursework's. You can also make your own using Gimp. Remember your code should work on any ppm file as long as it's not too big (like really big).

- pic1.ppm - A picture of a slice of pie
- tetons1.ppm - An image to use for testing object_filter
- tetons2.ppm - An image to use for testing object_filter
- tetons3.ppm - An image to use for testing object_filter
- tinypix.ppm - The example image from above

NOTE: None of your manipulations may cause a color number to be less than 0 nor larger than the maximum color depth specified in the file.

Submission:

- Put all of your files into a single folder called **hw4.**
- Compress your hw4 folder into a zip file named **UNI_hw4.zip** (eg. ac1076_hw4.zip).
- Submit that zip file via courseworks.
- Remember to include your read_me file.
- ***Remember that your homework is not submitted unless you receive a confirmation email!***

Grading

Your grade will be based on three components using the following wights:

Runs properly: 60% (Does your program run and do what it's supposed to?)

Style: 20% (Are you following PEP-8 guidelines?)

Design: 20% (Are you making use of functions? Is your code efficient? Easy to modify? Easy to debug?)