EECS 10: Assignment 7

November 11, 2005

Due Monday 11/28/05 at 12:00pm

1 Image Manipulation [80 points + 20 bonus points]

The goal of this final assignment for our programming class is to develop an image manipulation program called *"PhotoLab"*. Using *PhotoLab,* the user can load an image (such as a digital photo) from a file, apply a set of transformations to it, and then save the manipulated image again in a file.

Original Image "Disney.ppm"

Introduction

A digital image, such as the photo shown above, is essentially a two-dimensional matrix of pixels and most image transformations are simple matrix operations. As you know, a matrix can be represented in C best by a two-dimensional array. Thus image transformations can be implemented in C by operations on two-dimensional arrays.

For this assignment, we will use photo images of the size 480 by 640 pixels. A pixel is the smallest unit of an image. The color of each pixel is composed of the three primary colors, i.e. red (R), green (G) and blue (B), each of which is represented by an intensity value between 0 (black) and 255 (most extreme red/green/blue).

More specifically, we will use three two-dimensional arrays to represent an image, as follows:

```
#define WIDTH  480
#define HEIGHT 640
unsigned char R[WIDTH][HEIGHT];
unsigned char G[WIDTH][HEIGHT];
unsigned char B[WIDHT][HEIGHT];
```

We will use PPM (Portable Pixel Map) format as the file format of the input and output images of our program. This format is a convenient (simple) method of storing image data and is easy to use in applications like ours (unlike JPG images, PPM files are not compressed).

A PPM file consists of two parts, a header and the image data. The header consists of at least three parts separated by new line characters or white space. The first line is a magic PPM identifier that can be "P3" or "P6" (not including the double quotes) for 24-bit color images. The next line consists of the width and height of the image as ASCII numbers. The last part of the header gives the maximum value of the color components for the pixels (which allows the format to describe more than single byte (0...255) color values). After the header, the image data is stored in textual (P3) or binary (P6) format, pixel by pixel as RGB values.

Example: A sample PPM file

```
P6
480 640
255
RGBRGBRGB…
```

Since we have not covered file input and output in our class yet, C code for reading and writing an image in PPM format is provided with a template program file (see below).

The provided image read function `ReadPhotoPPM` reads an image from a specified file. The filename and three arrays R, G and B need to be supplied as arguments. After a successful call to `ReadPhotoPPM`, the specified arrays hold the image information.

Similar, the image write function `WritePhotoPPM` writes image data provided by three arrays R, G and B into a new image file specified by the given filename. Note that, for your convenience, the `WritePhotoPPM` function not only writes the image to a PPM file, it automatically converts the written image file also to JPG format and puts that file into a directory for easy viewing with your web browser. Thus, after saving an image, it can immediately be viewed with any standard web browser (more on this below).

Both functions, `ReadPhotoPPM` and `WritePhotoPPM`, return the value 0 to indicate success. Any value greater than zero indicates that some problem occurred, for example, the file specified was not found.

Program Specification for *PhotoLab*

Your program should be a menu driven program (just like the previous assignment). The user should be able to select image operations from the following menu:

```
(1)  Load image file
(2)  Save image file
(3)  Negative image
(4)  Grey-scale image
(5)  Mirror image (horizontal)
(6)  Flip image (vertical)
(7)  Color filter
(8)  Paint area
(9)  Overlay picture
(10) Shift and overlay picture (optional)
(11) Your Special Function (optional)
(12) Exit

Please enter your choice:
```

Your program should perform the following operations for the options:

1. **Load an image from a file**

   Prompt the user for an image file name (without any suffix) and load the specified file into your image arrays. Use the prepared `ReadPhotoPPM` function for loading the image file. Make sure to inform the user if the return value indicates any errors.

   ```
   Please input the file name to load: Disney.ppm
   Image file Disney.ppm loaded successfully.
   ```

2. **Save the image into a file**

   Same as option 1, but use `WritePhotoPPM` to store the image data into a file under a name specified by the user (again, the user should specify the name without any suffix). The `WritePhotoPPM` function will create the PPM file and also a corresponding JPG file in your public_html directory so that you can view the saved image by opening it in your web browser.

   ```
   Please input the file name to save: Disney2
   Image file Disney2.ppm saved successfully.
   Image file Disney2.jpg stored for viewing.
   ```

   Note that you probably will have to click “Refresh” in your browser before you can click on the image to view it.

3. **Generate the negative image**

   Generate the negative (inverse) image of the current photo. A negative image is basically an image in which all the intensity values have been inverted. That is, each value is subtracted from the maximum value, 255 in our case. So, in a black and white image, black areas would become white, and vice versa.

   For your reference, the original image shown at the beginning will look as follows:

4. **Convert to a grey-scale image**

   Convert the color image into a grey-scale image. A grey scale image is one without colors. It can be calculated by averaging all the colors at a pixel location:

   GreyValue = (R + G + B) / 3

   The calculated grey value is then used for each color channel, i.e. R, G, and B.

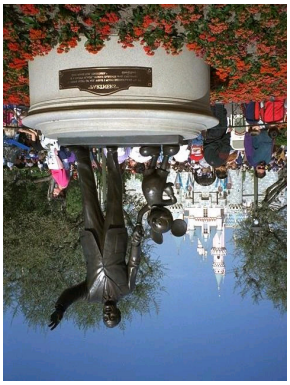
5. **Mirror the image horizontally (along the x axis)**

   Flip the image along the x-axis so as to see the mirror image of the given image with respect to the x-axis (i.e. right becomes left, and vice versa).

[Hint: Copy the first pixel value to the last and so on. A temporary data structure might be needed to store the intermediate values]

6. Flip the image vertically (along the y axis)

Flip the image along the y-axis so that it appears upside down. Compute this in a way similar to the previous case only that it's over the y-axis this time.



7. Color Filtering

A selective color filter is a powerful tool that allows you to specify changes applied to each primary color (R, G, and B). For example, selective color filtering can be used to increase or decrease the intensity of only the red colors in the image, increase saturation of greens, etc.

This function should allow the user to enter intensity factors for *each* primary color separately. The input values should obviously be larger than 0. For example, a scale factor of 1.0 for a certain primary color means "no change", so the filter is a "do nothing" filter (the primary color values of all pixels multiplied by 1.0 will not change). In another example, if the user inputs 3.0 as a scale factor for color red (and 1.0 for green and blue), all red pixel values will be multiplied by 3.0 and thus all red objects will "lighten up" in the photo.

Note that all computed pixel values after this transformation must fit again into the range 0 – 255 (the maximum range available for type **`unsigned char`**). Thus, you will need to round each computed value to an integer and check if it exceeds 255. If so, set it to the maximum value 255.

```
Please input the scale factor for Red:   3.0
Please input the scale factor for Green: 2.0
Please input the scale factor for Blue:  1.0
```

Given the above scale factors, your photo should look as follows:

8. Painting area

This function allows the user to set an area and paint it solid with a certain color. In particular, the user is asked first to enter the position of a rectangular area by specifying the left, right, upper and lower bounds. We will assume that the pixel in the upper left corner is at position (0, 0), and the pixel in the lower right corner has the coordinates (479, 639). Second, the user specifies the painting color as R, G, B color values. Your program should then paint the specified area with the specified color by setting all the pixel values inside the area to the entered color values.

```
Please input the left bound:  50
Please input the right bound: 200
Please input the upper bound: 50
Please input the lower bound: 400
Please specify the R value: 100
Please specify the G value: 100
Please specify the B value: 100
```

Given the above choices, your photo should look as follows:

9. Image overlay

This function overlies the current image with a second image similar to two transparencies on an overhead projector. For our program, we will use the UCI mascot, Peter the Anteater, and place him next to the statue of Walt Disney.

For your convenience, we have already prepared the overlay image, ***Anteater.ppm***, which is of the same size as the original photo, 480 by 640 pixels. In other words, both of these images have the same number of pixels, and each pixel in the overlay picture has its corresponding pixel in the original image (the one with the same "width" and "height" values).

To achieve the overlay effect, we will treat the background in the second image as transparent color. That is, each of the **non-background pixels** in Anteater.jpg will replace its corresponding pixel in the original image, whereas **background pixels** will stay as in the original image. Whether or not a pixel in Anteater.jpg is a background pixel can be decided by the RGB values of this pixel. More specifically, if the RGB values of a pixel are 255/255/255, which represents white color, then this pixel is a background pixel.

The function first needs to load the second file that will be used to do the overlay operation, and then replace the pixels in the original image with their corresponding non-background pixels in the second image.

```
Please input the name for the overlay: Anteater
Image file Anteater.ppm loaded successfully.
```

After the successful overlay operation, your image should look as follows:

10. Image shift and overlay (Extra Credit, 10 points)

In addition to the "Image overlay" function, this function not only overlies the original image with a second image, but also shifts the second image to a user-specified position. In our case, for example, we could place the anteater down on the floor simply by shifting its image down by 200 pixels.

To make this function general, an offset value between -480 and 480 for the x-axis, and an offset between -640 and 640 for the y-axis should be entered by the user. Then, each pixel in the second image positioned at (width, height) will be used to replace the pixel in the original image at the location positioned at (width+Xoffset, height + Yoffset). Any pixels outside the valid ranges should be treated as transparent background.

```
Please input the name for the overlay: Anteater
Image file Anteater.ppm loaded successfully.
Please input the X offset: 0
Please input the Y offset: 200
```

Given the offset values shown above, your resulting image should look as follows:

11. Your Special Function (Extra Credit, up to 10 points)

Here you have a chance to use your imagination. Provide an additional function that manipulates the image in some interesting fashion. For a function in similar complexity to the ones listed above, you will earn 10 extra points.

12. Exit

Exit the program.

Setup and Implementation

In order to be able to view the images generated by your program by use of a standard web browser, follow the following steps to setup your `hw7` directory:

```
% cd
% mkdir hw7
% mkdir public_html
% chmod 755 public_html
% cd hw7
% cp ~eecs10/hw7/PhotoLab.c .
% cp ~eecs10/hw7/Disney.ppm .
% cp ~eecs10/hw7/Disney.jpg ../public_html
% cp ~eecs10/hw7/Anteater.ppm .
% cp ~eecs10/hw7/Anteater.jpg ../public_html
```

Note that your `public_html` directory is posted on the web by the EECS web server. Whatever files you place into this directory will be publicly accessable by anyone on the Internet. So, please make sure that your `public_html` directory only contains files (e.g. the pictures used in this lab) that are OK for public posting. In particular, do not put your source code into that directory.

(a) Double-check your setup by opening an internet browser (i.e. Netscape or Internet Explorer) and type in the following URL. Make sure to replace **`userid`** with *your own* userid!
`http://east.eecs.uci.edu/~userid/`
This should list the files you have in your `public_html` directory! If not, please go back to the setup instructions above.

(b) To view the original photo that we will use, click on the JPG file or type the following URL into your browser:
`http://east.eecs.uci.edu/~userid/Disney.jpg`

(c) Now you are all set. Use the file `PhotoLab.c` as a starting point for your program and fill in the spaces marked for your extension. Have fun!

If your setup is correct, once you choose option 2 "Save image file" and input the filename, the following is the output:

```
Please input the file name to save: Negative

Image file Negative.ppm saved successfully.
Image file Negative.jpg stored for viewing.
```

Script file

To demonstrate that your program works correctly, perform the following steps and submit them as your script file:

1. Load the image Disney.ppm, generate a negative image and save it as “Negative”.
2. Load the image Disney.ppm, generate a grey scale image and save it as “Grey”.
3. Load the image Disney.ppm, flip the image horizontally *and* vertically, and save it as “Flip”.
4. Load the image Disney.ppm, choose filter color and input 3.0 for color red, 2.0 for color green and 1.0 for color blue. Save it as “Filter”.
5. Load the image Disney.ppm, set the bound value as following: left bound 50, Right bound 200, upper bound 50, lower bound 400, R, G, B values are all 100. Save it as “Paint”.
6. Load the image Disney.ppm, and load the second image Anteater.ppm, and then overlie the Disney.ppm with Anteater.ppm.
   Save the result image as “Overlay”.
7. Load the image Disney.ppm, and load the second image Anteater.ppm, and then shift and overlie the Disney.ppm with Anteater.ppm.
   Save the result image as “Shift”.
8. Load the image Disney.ppm and call your special function. Save the result image as “Special”.
9. Exit the program

What to submit

Use the standard submission procedure to submit the following files:

- PhotoLab.c (version with your code filled in!)
- PhotoLab.script

Please leave all the images generated by the script in your public_html directory! Do not submit any images.