EECS 10: Assignment 7

November 12, 2004

Due Monday 11/29/04 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 "SimonsToys.ppm"

Introduction

An image, such as the photo above, is just like a two-dimensional matrix and most image transformations are simple matrix operations like the ones performed for the previous assignment.

For this assignment, we will use photo images of the size 640 by 480 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, green and blue, each of which is represented by an intensity value between 0 and 255.

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

```
#define WIDTH  640
#define HEIGHT 480
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 newline 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 RGB values.

Example: A sample PPM file

```
P6
640 480
255
RGBRGBRGB…
```

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

The 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. The generated image file can then be viewed with other programs (i.e. a web browser).

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)  Image quantization
(8)  Image binarization
(9)  Blurred image
(10) Color manipulation (opt.)
(11) Noise addition (opt.)
(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 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.

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

   Same as option 1, but use `WritePhotoPPM` to store the image data into a file specified by the user.

3. **Generate the negative image**

   Generate the negative or the inverse image of the provided image. A negative image is basically an image in which all the image values have been subtracted from the maximum (255 in this case). So, in a black and white image, black areas would become white and vice versa.

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

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

   GreyValue = (R + G + B) / 3

[Hint: A fact to consider here is that the Gray Scale image developed would be $1/3^{rd}$ the size of the original image. So the gray values need to be copied to all the color channels.]

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 need to store the transient 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. Quantization

Quantization is a way of reducing the dynamic range of an image. It means that the image information is slotted into specified quanta(s) or categories. Quantize it by checking the following:

If the pixel value is between 0 and 63, make the value to 0, if the value is between 64 and 127 then make it to 64, if between 128 and 191, make it to 128, if between 192 and 255, make it 192. With this, the image is effectively quantized to four packets.

When you see the effect of this transformation in the image, it should appear blocky since now the number of colors values have been reduced to 4.

Perform this transformation for all the three color channels together.

8. Binarization

A binary image is one where the image information is only 0 or 1. To convert an image to binary, a test needs to be done with the image information against a threshold value. If the image pixel value is above the threshold value, then the image pixel value should be made 255. Otherwise, it should be set to 0.

Here, the threshold value should be asked from the user and it should be used for comparison for forming the binary image.

9. **Blurring**

   Blurring an image is often done for removing unwanted noise from an image. It also removes sharp features and gives smooth edges. Blurring can be done by averaging the value of the color of the pixel with its neighborhood pixels.

   In this function, consider a pixel and average its value with its neighbors. Use the pixels to the left, right, top, and bottom of the current pixel as its neighbors and consider the average of the five for the pixel value of the present location.

10. **Color manipulation (Extra credit: 10 Points)**

    Varying the different color channels can have funny effects on the image. Here, we allow the user to manipulate each color channel separately by adding or subtracting a fixed amount.

    Prompt the user for three values (r, g, b) in the range of -255 through 255 to be added to each color channel. Add the values to the RGB values of each pixel, but make sure the final value is still in the valid range of 0-255. Thus, any value below 0 should be set to 0, and any value above 255 should be set to 255.

11. **Noise addition (Extra credit: 10 Points)**

    Noise usually means unwanted information in an image. However, different types of noise can be added to an image. Here, we will work similar to option 10.

    Prompt the user for *one* value in the range of -255 through 255. Add this value to each color channel of each pixel in the same fashion as described above, but scale the value each time by a random factor between 0 and 1. The random factor should be created by use of the random generator in the standard C library.

12. **Exit**

    Exit the program.

Setup and Implementation

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

```
% cd
% mkdir hw7
% chmod 755 hw7
% ln –s hw7 public_html
% cd hw7
% cp ~eecs10/hw7/PhotoLab.c .
% cp ~eecs10/hw7/SimonsToys.ppm .
% pnmtojpeg SimonsToys.ppm >SimonsToys.jpg
```

(b) 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 hw7 directory! If not, please go back to step (a).

(c) 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`*`/SimonsToys.jpg`

(d) To convert a PPM image into a JPG image that your browser can display, use the `pnmtojpeg` command in the following fashion:
```
% pnmtojpeg filename.ppm >filename.jpg
```

(e) Vice versa, to convert a JPG image into a PPM image that your program can read, use the `jpegtopnm` command in the following fashion:
```
% jpegtopnm filename.jpg >filename.ppm
```

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

Script file

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

1. Load the image SimonsToys.ppm, generate a negative image and save it as Negative.ppm
2. Load the image SimonsToys.ppm, generate a greyscale image and save it as Grey.ppm
3. Load the image SimonsToys.ppm, flip the image horizontally *and* vertically, and save it as Flip.ppm
4. Load the image SimonsToys.ppm, quantize the image and save it as Quant.ppm
5. Load the image SimonsToys.ppm, convert it to a binary image using the threshold value 128 and save it as Bin.ppm
6. Load the image SimonsToys.ppm, blurr the image and save it as Blurr.ppm
7. For 10 points extra credit: Load the image SimonsToys.ppm, manipulate the colors by adding 0/-255/-128 (r/g/b offsets) to the image colors and save it as Color.ppm
8. For another 10 points extra credit: Load the image SimonsToys.ppm, add noise of level 50 to the image and save it as Noise.ppm
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 the images generated by the script in your local `hw7` directory! Do not submit any images.