EECS 10: Assignment 7

November 11, 2006

Due on Monday 11/27/2006 12:00 noon. Note: this is a two-week assignment.

1 Digital Image Processing [80 points]

In this assignment you will learn some basic digital image processing (DIP) techniques by developing an image manipulation program called *PhotoLab*. Using the *PhotoLab*, the user can load an image from a file, apply a set of DIP operations to the image, and save the processed image in a file.

1.1 Introduction

A digital image is essentially a two-dimensional matrix, which can be represented in C by a two-dimensional array, of pixels. A pixel is the smallest unit of an image. The color of each pixel is composed of three primary colors, red, green, and blue; each color is represented by an intensity value between 0 and 255. In this assignment, you will work on images with a fixed size, 640×480, and type, Portable Pixel Map (PPM).

The structure of a PPM file consists of two parts, a header and image data. In the header, the first line gives the type of the image, P6; the next line shows the width and height of the image; the last line is the maximum intensity value. After the header is the image data, arranged as RGBRGBRGB..., pixel by pixel.

Here is an example of a PPM image file:

```
P6
640 480
255
RGBRGBRGB...
```

1.2 Program Specification

In this assignment, your program should be able to read and save image files. To let you concentrate on DIP operations, the functions for file reading and saving are provided. These functions are able to catch many file reading and saving errors, and show corresponding error messages.

Your program is a menu driven program (like the previous assignment). The user should be able to select DIP operations from a menu similar to the one shown below:

```
--------------------------------
 1:  Load a PPM image
 2:  Negative image
 3:  Change color image to grey
 4:  Flip image horizontally
 5:  Flip image vertically
 6:  Threshold image
 7:  Add noise to image
```

```
 8:  Blur image
 9:  Enhance image contrast
10:  Create histogram image (extra credit)
11:  Exit
please make your choice:
```

1.2.1 Load a PPM Image

This option prompts the user for the name of an image file. You don't have to implement a file reading function; just use the provided one, *ReadPPM*. Once option 1 is selected, the following is shown:

```
Please input the file name to load:
```

After a name, for example *myimage.ppm*, is entered, the *PhotoLab* will load the file. If it is read correctly, the following is shown:

```
myimage.ppm was read successfully!
--------------------------------
 1:  Load a PPM image
 2:  Negative image
 3:  Change color image to grey
 4:  Flip image horizontally
 5:  Flip image vertically
 6:  Threshold image
 7:  Add noise to image
 8:  Blur image
 9:  Enhance image contrast
10:  Create histogram image (extra credit)
11:  Exit
please make your choice:
```

In this case, you can select other options. If there is a reading error, for example the file name is entered incorrectly or the file doesn't exist, the following message is shown:

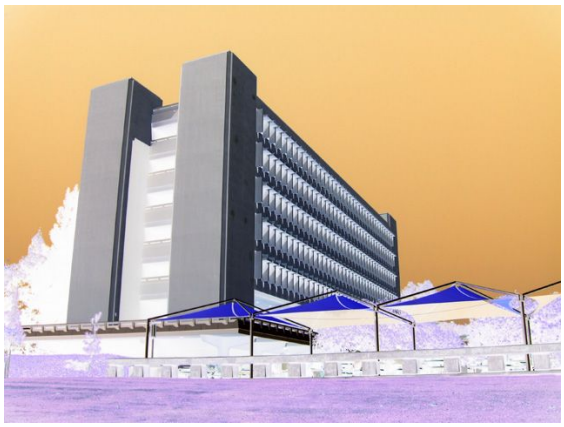
```
Cannot open file "myimage.ppm" for reading!
-------------------------------
 1:  Load a PPM image
 2:  Negative image
 3:  Change color image to grey
 4:  Flip image horizontally
 5:  Flip image vertically
 6:  Threshold image
 7:  Add noise to image
 8:  Blur image
 9:  Enhance image contrast
10:  Create histogram image (extra credit)
11:  Exit
please make your choice:
```

In this case, try option 1 again.

1.2.2 Negative Image

A negative image is an image in which all the intensity values have been inverted. To achieve this, each intensity value at a pixel is subtracted from the maximum value, 255, and the result is assigned to the pixel as a new intensity. You need to define and implement a function to do the job. Figure 1 shows an example of this operation.

(a) Original image



(b) Negative image

Figure 1: An image and its negative counterpart.

(a) Color image

(b) Grey image

Figure 2: A color image and its grey counterpart.

1.2.3 Change Color Image to Grey

A grey image is the one that the intensity values are the same for all color channels, red, green, and blue, at each pixel. To change a color image to grey, assign a new intensity, which is given by $(R + G + B)/3$, to all the color channels at a pixel. The R, G, B are the old intensity values for the red, the green, and the blue channels at the pixel. You need to define and implement a function to do the job. Figure 2 shows an example of this operation.

1.2.4 Flip Image Horizontally

To flip an image horizontally, the intensity values in horizontal direction should be reversed. The following shows an example.

```
                          1 2 3 4 5                                  5 4 3 2 1
before horizontal flip:   0 1 2 3 4     after horizontal flip:     4 3 2 1 0
                          3 4 5 6 7                                  7 6 5 4 3
```

You need to define and implement a function to do the job. Figure 3 shows an example of this operation.

(a) Original image

(b) Horizontally flipped image

Figure 3: An image and its horizontally flipped counterpart.

(a) Original image

(b) Vertically flipped image

Figure 4: An image and its vertically flipped counterpart.

1.2.5 Flip Image Vertically

To flip an image vertically, the intensity values in vertical direction should be reversed. The following shows an example:

```
                        1 0 5                          5 4 9
                        2 1 6                          4 3 8
before vertical flip:   3 2 7    after vertical flip:  3 2 7
                        4 3 8                          2 1 6
                        5 4 9                          1 0 5
```

You need to define and implement a function to do the job. Figure 4 shows an example of this operation.

1.2.6 Threshold Image

Image thresholding set intensity values less than the threshold to zero, i.e., if the intensity value is below the threshold, it needs to be replaced with 0, while the intensity value remains unchanged if it is above the threshold. For example, if the green color channel has an threshold of 128

(a) Original image

(b) Threshold settings: r=50, g=110, b=250

Figure 5: An image and its thresholded counterpart.

(a) Original image

(b) Noise setting: n=10

Figure 6: An image and its noise (salt-and-pepper) corrupted counterpart.

```
before thresholding:  100 112 150 170      after thresholding: 0   0   150 170
                      165 110 115 130                          165 0   0   130
```

You need to define and implement a function to do the job. The function needs three input parameters, which are the thresholds for the red, the green, and the blue color channels. Figure 5 shows an example of this operation.

1.2.7 Add Noise to Image

In this operation, you add noise to an image. The noise added is a special kind, called salt-and-pepper noise, which means the noise is either black or white. You need to define and implement a function to do the job. The function needs a parameter, which is the percentage of noise added to the image. If the percentage of noise is n, then the number of noise added to the image is given by n*$WIDTH$*$HEIGHT/100$, where $WIDTH$ and $HEIGHT$ are the image size. You need the knowledge of random number generator from the previous assignments. Figure 6 shows an example of this operation.

(a) Original image

(b) Blurred image

Figure 7: An image and its blurred counterpart.

Implementation hint: calculate first the number of noise pixels depending on the noise percentage; then, use the random number generator to determine the position (x and y coordinate) of each noise pixel. Create a white or a black pixel by setting the intensity value at each color channel to its maximum (255) or minimum (0) respectively. Note: the number of white pixels should be approximately equal to the number of black pixels.

1.2.8 Blur Image

Image blurring works this way: the intensity value at each pixel is mapped to a new value, which is the average of itself and its 8 neighbours. The following shows an example:

```
X X X X X
X 4 1 2 X
X 2 0 1 X
X 4 1 3 X
X X X X X
```

To blur the image, the intensity at the pixel with a value of 0 is changed to, (4+1+2+2+0+1+4+1+3)/9 = 2. Repeat this for every pixel, and for every color channel (red, green, and blue) of the image. You need to define and implement a function to do the job. Special care has to be taken for pixels located on image boundary (hint: for ease of implementation, you may not modify the pixels at the border of the image).

1.2.9 Enhance Image Contrast

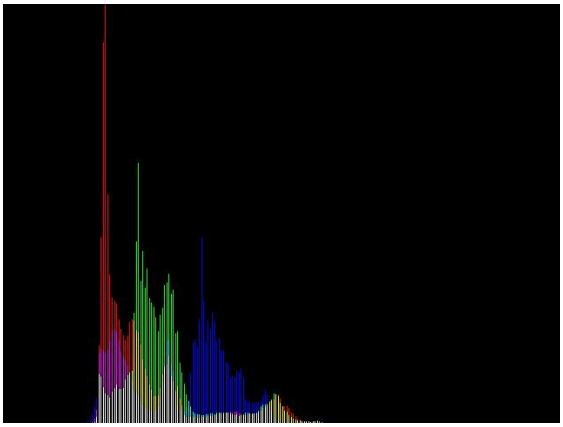
The contrast of an image can be improved by stretching its histogram to its full range. To do this the maximum and the minimum intensity values of the image are found first, then the intensity value for each pixel is mapped to a new value by the following transfer function:

$$I_{new} = 255 * (I_{old} - I_{min})/(I_{max} - I_{min})$$

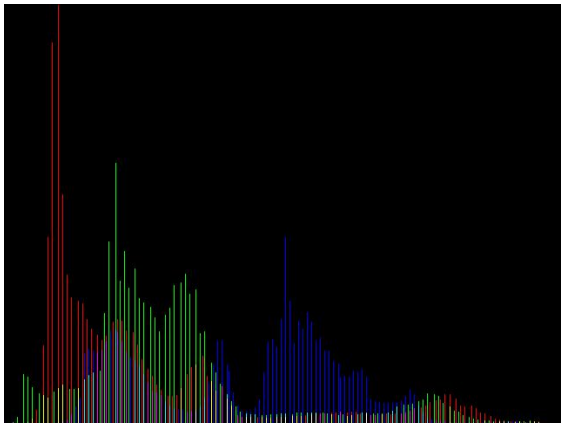
where I_{new} and I_{old} are the new and the old intensity values at a pixel; I_{max} and I_{min} are the maximum and the minimum intensity values of a color channel of the image. Do the same mapping for all the color channels (red, green, and blue). You need to define and implement a function to do the contrast enhancement. Figure 8 shows an example of this operation. Take a look at the histograms, you can see that the low contrast image has a narrow histogram, and the enhanced image has a much wider one.

(a) Low-contrast image

(b) Contrast-enhanced image

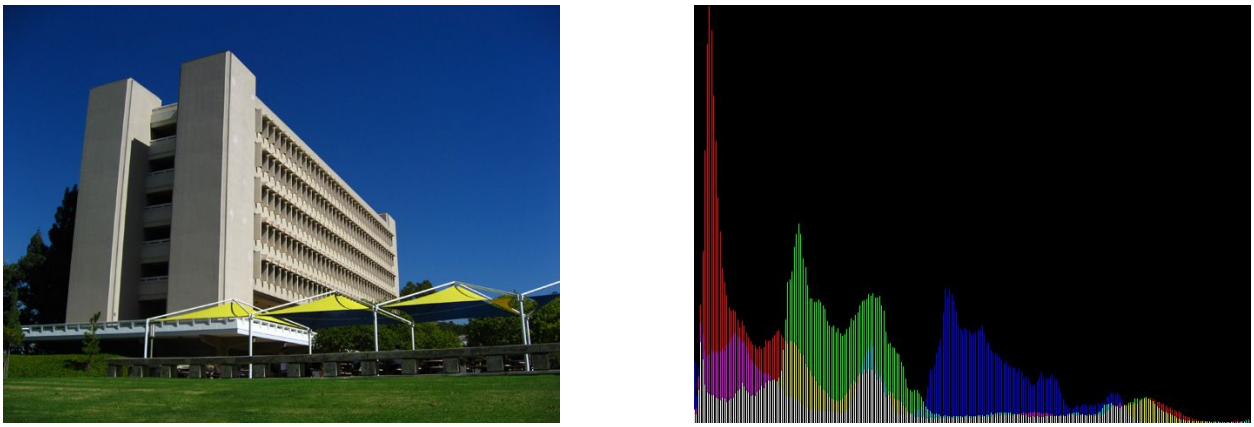


(c) Histogram of the low-contrast image



(d) Histogram of the contrast-enhanced image

Figure 8: A low-contrast image and its enhanced counterpart.



(a) An image

(b) Histogram of the image

Figure 9: An image and its histogram.

1.2.10 Create Histogram Image (extra credit: 20pt)

Image histogram shows the intensity distribution of an image. To be more specific, the histogram of an image shows how often each color intensity value occurred in the image. Since intensity values are in the range of 0 to 255, we need an array with 256 elements to store the histogram. For example, we will use the array *int HG[256]* for storing the histogram of green color channel. In the array, *HG[0]* captures how often the image had pixels with the intensity value of 0 in the green color channel. Similarly, a value of 736 for *HG[138]* indicates that *736* pixels of the image have the intensity value of 138 for the green color channel.

If you want the extra credit, you need to define and implement a function to get the histogram of an image, and produce an image from it (like the one shown in Figure 9). Here are some hints:

```
1: define arrays to hold the histogrms of color channels, red, green, and blue:
   int HR[256]; int HG[256]; int HB[256];
2: find the histogram of the image, and save it in HR, HG, and HB
3: draw vertical lines from i=0 to i=255; the height of
   line is determined by
       (HR[i]/mx)*(HEIGHT-1) for red channel
       (HG[i]/mx)*(HEIGHT-1) for green channel
       (HB[i]/mx)*(HEIGHT-1) for blue channel
   where mx is the maximum value in the integer arrays; HEIGHT is image height.
4: map the i (histogram index) to x (pixel position) by: x = i*WIDTH/255
```

Figure 9 shows an image and its histogram. You should get similar results in this operation.

1.2.11 Summary

For this assignment, you need to define the following functions:

```
int  ReadPPM()                  /* provided */
int  WritePPM()                 /* provided */
void Negative()                 /* reverse image color */
void Grey()                     /* change color image to grey */
void HFlip()                    /* flip image horizontally */
void VFlip()                    /* flip image vertically */
void Threshold( int, int, int ) /* threshold image */
```

```
void AddNoise( int )            /* add salt-and-pepper noise to image */
void Average()                  /* blur image */
void Enhance()                  /* increase image contrast */
void Histogram()                /* extra credit (20pt) */
```

You may want to define other functions as needed.

You also need the following global constants and data structures:

```
#define WIDTH 640   /* image width */
#define HEIGHT 480  /* image height */
#define SLEN 80     /* maximum file name length */
unsigned char R0[WIDTH][HEIGHT]; /* for original image red channel data */
unsigned char G0[WIDTH][HEIGHT]; /* for original image green channel data */
unsigned char B0[WIDTH][HEIGHT]; /* for original image blue channel data */
unsigned char R[WIDTH][HEIGHT];  /* for modified image red channel data */
unsigned char G[WIDTH][HEIGHT];  /* for modified image green channel data */
unsigned char B[WIDTH][HEIGHT];  /* for modified image blue channel data */
int HR[256];   /* for red channel histogram */
int HG[256];   /* for green channel histogram */
int HB[256];   /* for blue channel histogram */
```

2 Setup and Implementation

Before start working on the assignment, do the following:

```
setenv LD_LIBRARY_PATH /dcs/packages/Xlocal/lib
cd
mkdir hw7
mkdir public_html (if it doesn't exist)
chmod 755 public_html
cd hw7
cp ~eecs10/hw7/PhotoLab.c .
cp ~eecs10/hw7/imgColor.ppm .
cp ~eecs10/hw7/imgEnhance.ppm .
cp ~eecs10/hw7/index.htm ../public_html
```

The file, *PhotoLab.c*, is where you get started; *imgColor.ppm* and *imgEhance.ppm* are the PPM images used to test the DIP operations. Once a DIP operation is done, you can save the modified image. You will be prompt for a name of the image. The saved image will be automatically converted to a JPEG image and sent to the folder, *public_html*. You are able to see the image at: *http://east.eecs.uci.edu/~userid/*

Note that whatever you put in the *public_html* directory will be publicly accessable; make sure don't put files that you don't want to share, for example your source code for the assignment, in the directory.

2.1 Script File

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

1. Start the script by typing the command: *script*
2. Compile and run your program
3. Load *imgColor.ppm*
    - generate a negative image and save it as *negative*
    - generate a grey image and save it as *grey*
    - generate a horizontally flipped image and save it as *hflip*
    - generate a vertically flipped image and save it as *vflip*
    - generate a thresholded image with settings, r=50, g=110, b=250, and save it as *threshold*
    - generate a noisy image with setting, n=10, and save it as *noise*
    - generate a blurred image image and save it as *blur*
    - generate a histogram image and save it as *histogram* (only if you do the extra credit)
4. Load *imgEnhance.ppm*
    - generate a contrast enhanced image and save it as *enhance*
5. Exit the program
6. Stop the script by typing the command: *exit*
7. Change the script file to *PhotoLab.script*

NOTE: make sure use exactly the same names as shown in the above steps when saving modified images! The script file is important, and will be checked in grading; you must follow the above steps to create the script file.

3 Submission

Use the standard submission procedure to submit the following files:

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

Please leave the images generated by your program in your *public_html* directory. Don't delete them as they will be checked when grading! You don't have to submit any images.