# EECS 10: Assignment 7

Novermber 14, 2008

Due on Monday 12/01/2008 12:00pm. Note: this is a two-week assignment.

## 1 Digital Image Processing [80 points + 20 bonus 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, $640 \times 480$, 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 specifies 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 follows the image data, arranged as RGBRGBRGB..., pixel by pixel in binary representation.

Here is an example of a PPM image file:

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

### 1.2 Initial Setup

Before you start working on the assignment, do the following:

```
cd ~
mkdir -p hw7
cd hw7
cp ~eecs10/hw7/PhotoLab.c .
cp ~eecs10/hw7/balloons.ppm .
cp ~eecs10/hw7/balloon.ppm .
```

**NOTE**: Please execute the above setup commands only **ONCE** before you start working on the assignment! Do not execute them after you start the implementation, otherwise your code will be overwritten!

The file *PhotoLab.c* is the template file where you get started. It provides the functions for image file reading and saving, as well as the DIP function prototypes and some variables (do not change those function prototypes or variable definitions). You are free to add more variables and functions to the program.

The files *balloons.ppm* and *balloon.ppm* are the PPM images that we will use to test the DIP operations. Once a DIP operation is done, you can save the modified image. You will be prompted for a name of the image. The saved image *name.ppm* will be automatically converted to a JPEG image and sent to the folder *public_html* in your home directory. You are then able to see the image in a web browser at: *http://newport.eecs.uci.edu/∼userid* (If you access the server out of campus, use the link *http://newport.eecs.uci.edu/∼userid/imagename.jpg* to access the photo)

Note that whatever you put in the *public_html* directory will be publicly accessable; make sure you don't put files there that you don't want to share, i.e. do not put your source code into that directory.

## 1.3 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 as the one shown below:

```
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Flip the image horizontally
 5:  Flip the image vertically
 6:  Rotate the image 180 degrees
 7:  Zoom in
 8:  Image Overlay
 9:  Age the image
10:  Blur the image (10 extra points)
11:  Detect image edges (10 extra points)
12:  Exit
please enter your choice:
```

### 1.3.1 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, *ReadImage*. Once option 1 is selected, the following should be shown:

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

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

```
balloons.ppm was read successfully!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Flip the image horizontally
 5:  Flip the image vertically
 6:  Rotate the image 180 degrees
 7:  Zoom in
 8:  Image Overlay
 9:  Age the image
10:  Blur the image (10 extra points)
```

```
11:  Detect image edges (10 extra points)
12:  Exit
please enter your choice:
```

Then, you can select other options. If there is a reading error, for example the file name is entered incorrectly or the file does not exist, the following message is shown:

```
Cannot open file "balloons.ppm.ppm" for reading!
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Flip the image horizontally
 5:  Flip the image vertically
 6:  Rotate the image 180 degrees
 7:  Zoom in
 8:  Image Overlay
 9:  Age the image
10:  Blur the image (10 extra points)
11:  Detect image edges (10 extra points)
12:  Exit
please enter your choice:
```

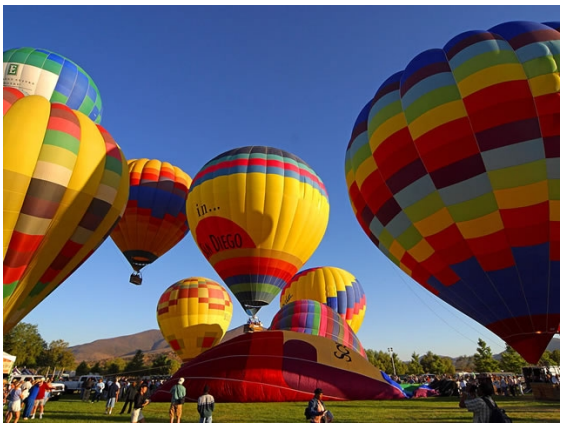
In this case, try option 1 again with the correct filename.

### 1.3.2 Save a PPM Image

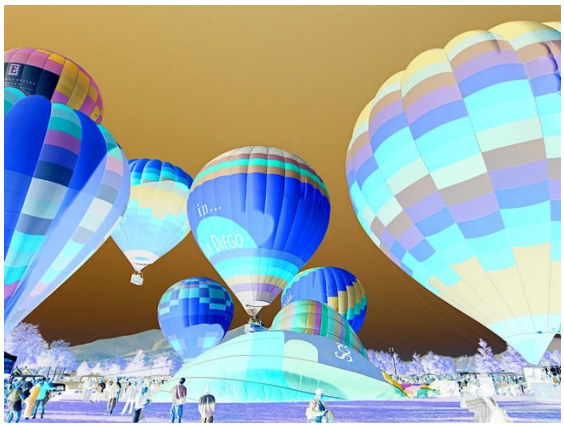
This option prompts the user for the name of the target image file. You don't have to implement a file saving function; just use the provided one, *SaveImage*. Once option 2 is selected, the following is shown:

```
Please enter your choice: 2
Please input the file name to save: negative
negative.ppm was saved successfully.
negative.jpg was stored for viewing.
--------------------------------
 1:  Load a PPM image
 2:  Save the image in PPM and JPEG format
 3:  Make a negative of an image
 4:  Flip the image horizontally
 5:  Flip the image vertically
 6:  Rotate the image 180 degrees
 7:  Zoom in
 8:  Image Overlay
 9:  Age the image
10:  Blur the image (10 extra points)
11:  Detect image edges (10 extra points)
12:  Exit
please enter your choice:
```

The saved image will be automatically converted to a JPEG image and sent to the folder *public_html*. You then are able to see the image at: *http://newport.eecs.uci.edu/∼userid* (For off campus, the link is: *http://newport.eecs.uci.edu/∼userid/imagename.jpg*)



(a) Original image (source:Freedigitalphotos.net)



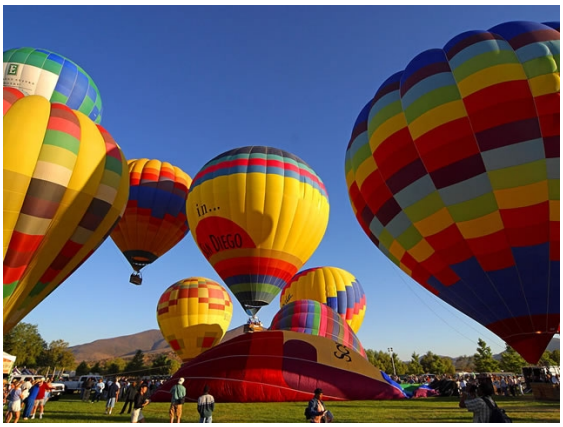
(b) Negative image

Figure 1: An image and its negative counterpart.

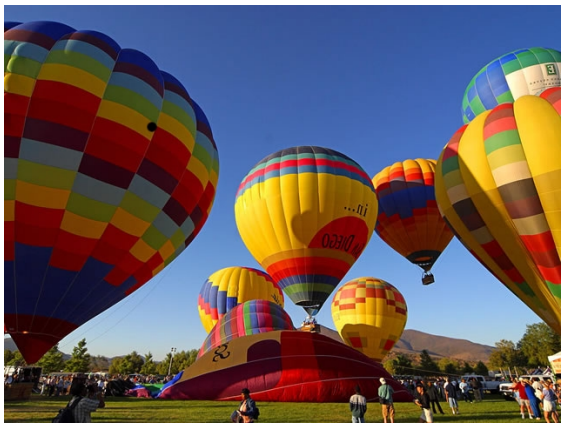
### 1.3.3 Make a Negative of the Image

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 this DIP.
Figure 1 shows an example of this operation. Your program's output for this option should be like:

```
Please enter your choice: 3
"Negative" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```



(a) Original image (source:Freedigitalphotos.net)



(b) Horizontally flipped image

Figure 2: An image and its horizontally flipped counterpart.

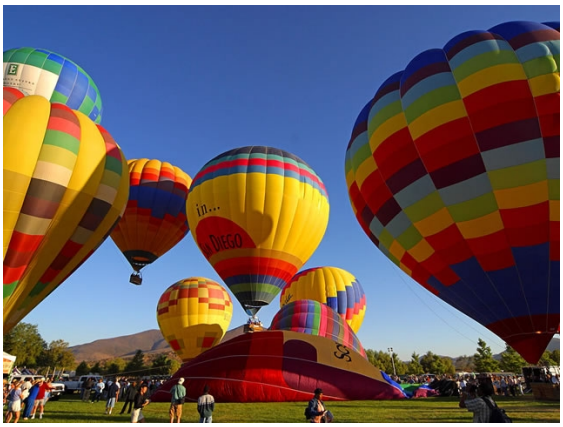
#### 1.3.4 Flip Image Horizontally

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

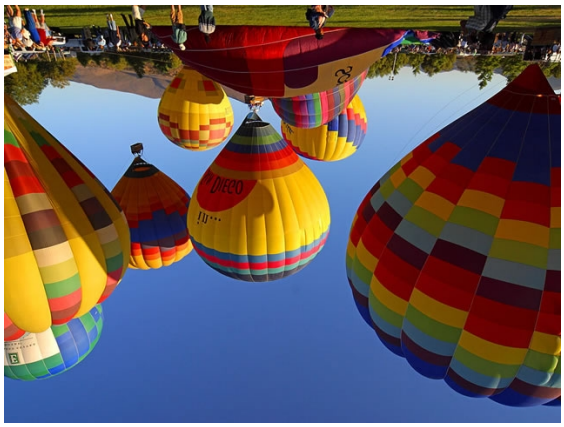
```
                          1 2 3 4 5                                5 4 3 2 1
  before horizontal flip: 0 1 2 3 4     after horizontal flip:     4 3 2 1 0
                          3 4 5 6 7                                7 6 5 4 3
```

You need to define and implement a function to do this DIP. Figure 2 shows an example of this operation. Your program's output for this option should be like:

```
Please enter your choice: 4
"HFlip" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```



(a) Original image (source:Freedigitalphotos.net)



(b) Vertically flipped image

Figure 3: An image and its vertically flipped counterpart.

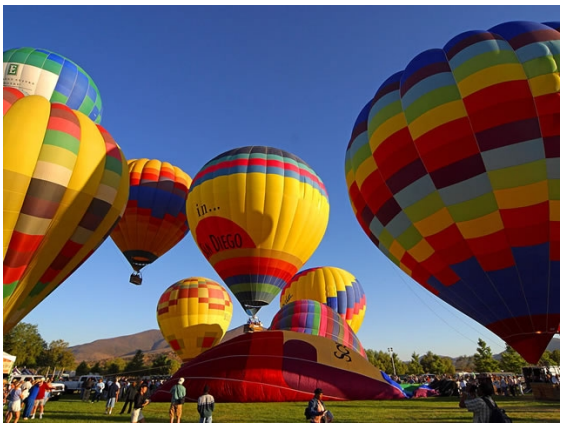
### 1.3.5 Flip Image Vertically

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

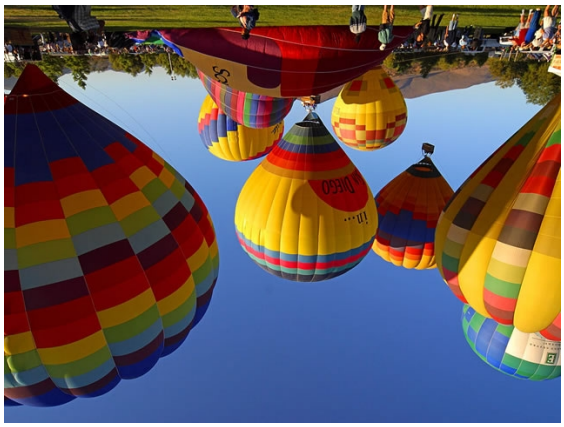
```
                          1 0 5                           5 4 9
                          2 1 6                           4 3 8
  before vertical flip:   3 2 7    after vertical flip:   3 2 7
                          4 3 8                           2 1 6
                          5 4 9         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

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

```
please enter your choice: 5
"VFlip" operation is done!
---------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```



(a) Original image (source:Freedigitalphotos.net)



(b) 180 degree rotated image

Figure 4: An image and its 180 degree rotation counterpart.

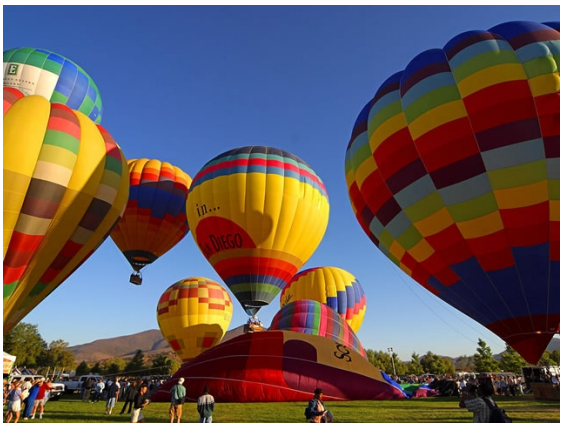
### 1.3.6 Rotate the Image 180 degree

To rotate an image 180 degrees, the intensity values in both horizontal and vertical direction should be reversed. The following shows an example:

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

Figure 4 shows an example of this operation.

```
please enter your choice: 6
"180 degree rotation" operation is done!" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```



(a) Original image (source:Freedigitalphotos.net)

(b) 2x Zoom-In image

Figure 5: An image and its 2x Zoom-In counterpart

### 1.3.7 Zoom-In

In this section, you need to implement a simple zoom-in 2x function, that enlarge the image center by a factor of two.
To zoom in, the intensity values in the center of the picture are distributed to the whole picture. The following shows an example:

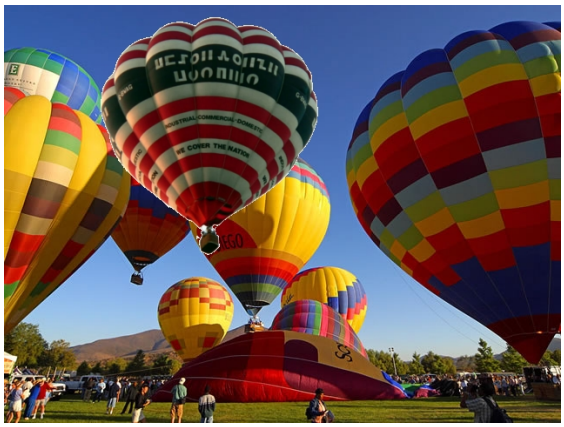
```
                        1  0 5  10                          1 1 6 6
                        2 |1 6| 11                          1 1 6 6
  before zoom-in:       3 |2 7| 12      after zoom-in:      2 2 7 7
                        4  3 8  13                          2 2 7 7
```

You need to define and implement a function to do this DIP. Figure 5 shows an example of this operation.

```
Please enter your choice: 7
"Zoom in 2x" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```

-

(a) Overlay image (source:Freedigitalphotos.net)



(b) Overlay image over original image

Figure 6: Overlay image and result of overlay operation

### 1.3.8 Image Overlay

This function overlies the current image with a second image. In our program, we will use an image with another balloon at the top left on a white background as the second image(see Figure6(a)).

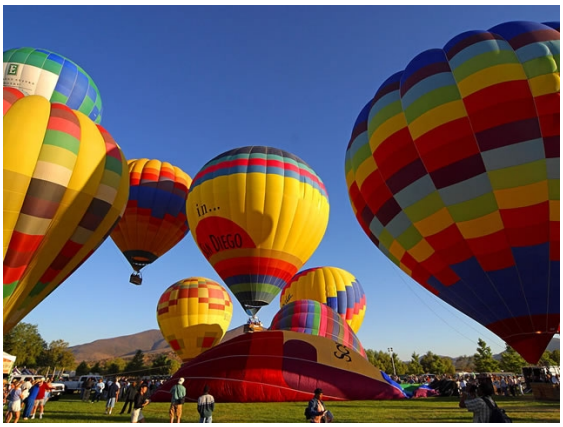
For your convenience, we have already prepared the overlay image, *balloon.ppm*, which is of the same size as the original photo, 680 by 480 pixels. In other words, both of these images have the same number of pixels, and each pixel in the overlay picture has its corresponding pixel in the original image (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 ballon.ppm will be overlayed to its corresponding pixel in the original image, whereas background pixels will stay as in the original image. Whether or not a pixel in *balloon.ppm* 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 image file (*balloon.ppm*) and then overlay the pixels in the original image with their corresponding non-background pixels in the second image.

Once the user chooses this option, your program's output should like this:

```
Please enter your choice: 8
Please input the file name for the second image: balloon
balloon.ppm was read successfully!
"Overlay" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
```



(a) Original image (source:Freedigitalphotos.net)

(b) Aged image without blur

Figure 7: An image and its aged counterpart.

```
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```

After the successful overlay operation, the overlayed image should like the figure shown in Figure 6(b):

#### 1.3.9 Aging

This function ages the original photo to make it look like an old photo. For each pixel, the colored pixels are converted into gray first, then yellow color is added to each of the pixels proportionally. The formula to get the aged pixel is given below:

$p[B] = (p[R] + p[G] + p[B])/5$
$p[R] = p[R] * 1.6$
$p[G] = p[G] * 1.6$

Figure 7 shows an example of this operation. Once the user chooses this option, your program's output should like this:

```
Please enter your choice: 9
"Aging" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```

(a) Aged image without blur

(b) Aged image with blur

Figure 8: An aged image and its blur counterpart.

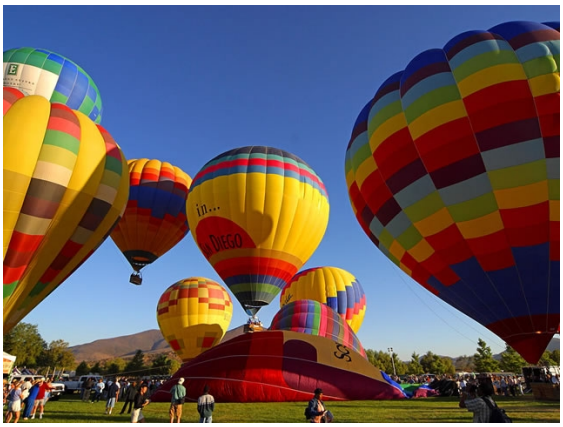
#### 1.3.10 Blur (bonus points: 10pt)

After color aging, the image should to be blurred to make it look more like an old photo. The 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 of the center pixel with the 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 this DIP.
Note that special care has to be taken for pixels located at the image boundaries. For ease of implementation, you may choose to ignore the pixels at the border of the image where no neighbour pixels exist.
After the two process, the aged image should look like the figure shown in Figure 8(b):

```
Ple1ase enter your choice: 10
"Blur" operation is done!
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```



(a) Original image

(b) Edge detection with k = 50

Figure 9: An image and its edge detection counterpart.

#### 1.3.11 Edge Detection (bonus points: 10pt)

The aim of edge detection is to determine the edge of shapes in a picture and to be able to draw a result image where edges are in white on black background. The idea is very simple; we go through the image pixel by pixel and compare the color of each pixel to its right neighbor, and to its bottom neighbor. If one of these comparisons is greater than a threshold K, the pixel studied is part of an edge and should be turned to white (RGB code = 255, 255, 255), otherwise it is turned to black (RGB code = 0, 0, 0). Figure 9 shows an example of this operation when k is set to 50.

To compare two colors, we can quantify the "difference" between two colors by computing the geometric distance between the vectors representing those two colors. Let's consider two color pixels C1 = (R1,G1,B1) and C2 = (R2,B2,G2). The difference between the two color pixels is given by the formula:

$$D(C1, C2) = \sqrt{(R1 - R2)^2 + (B1 - B2)^2 + (G1 - G2)^2}$$

So, here is the algorithm for edge detection in pseudo code.

*For every pixel ( i , j ) on the source image*

- *Extract the (R,G,B) components of this pixel C, its right neighbor C1(R1,G1,B1), and its bottom neighbor C2(R2,G2,B2)*
- *Compute D(C,C1) and D(C,C2)*
- *If D(C,C1) or D(C,C2) are greater than a given threshold K, then we have an edge pixel and turn this pixel to white, otherwise we turn this pixel to black.*

Note that you can avoid the need for the square-root function in your program (making it much faster!) by comparing the squares of the distances with $K^2$.
Note also that you should skip the pixels at the very bottom and the very right side since there are no neighbour pixels to compute.

Once user chooses this option, your program's output should be like:

```
Please make your choice: 11
Enter the difference threshold: 50
"Edge Detection" operation is done!
```

```
--------------------------------
     1:  Load a PPM image
     2:  Save the image in PPM and JPEG format
     3:  Make a negative of an image
     4:  Flip the image horizontally
     5:  Flip the image vertically
     6:  Rotate the image 180 degrees
     7:  Zoom in
     8:  Image Overlay
     9:  Age the image
    10:  Blur the image (10 extra points)
    11:  Detect image edges (10 extra points)
    12:  Exit
please enter your choice:
```

## 1.4 Implementation

### 1.4.1 Function Prototypes

For this assignment, you need to define the following functions (those function prototypes are already provided in *PhotoLab.c*. Please do not change them):

```
/* print a menu */
void PrintMenu();

/* read image from a file */
int  ReadImage(char fname[SLEN], unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);


/* save a processed image */
int SaveImage(char fname[SLEN], unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* reverse image color */
void Negative(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);


/* flip image horizontally */
void HFlip(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* flip image vertically */
void VFlip(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* aging the photo */
void Aging(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* blur the photo */
void Blur(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* Zoom in the photo */
void Zoomin(unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);

/* detect the edges of the objects in the image */
void EdgeDetect(double threshold, unsigned char R[WIDTH][HEIGHT], unsigned char G[WIDTH][HEIGHT],
unsigned char B[WIDTH][HEIGHT]);


/* Load an image and overlay it on the current image */
void Overlay(char fname[SLEN], unsigned char R[WIDTH][HEIGHT],
unsigned char G[WIDTH][HEIGHT], unsigned char B[WIDTH][HEIGHT]);
```

You may want to define other functions as needed.

### 1.4.2 Global constants

You also need the following global constants (they are also declared in *PhotoLab.c*, please don't change their names):

```
#define WIDTH 640   /* image width */
#define HEIGHT 480  /* image height */
#define SLEN 80     /* maximum file name length */
```

### 1.4.3 Pass in arrays by reference

In the main function, three two-dimensional arrays are defined. They are used to save the RGB information for the current image:

```
int main()
{
unsigned char R[WIDTH][HEIGHT];  /* for image data */
unsigned char G[WIDTH][HEIGHT];
unsigned char B[WIDTH][HEIGHT];
}
```

When any of the DIP operations is called in the main function, those three arrays: R[WIDTH][HEIGHT], G[WIDTH][HEIGHT], B[WIDTH][HEIGHT] are the parameters passed into the DIP functions. Since arrays are passed by reference, any changes to R[ ][ ], G[ ][ ], B[ ][ ] in the DIP functions will be applied to those variables in the main function. In this way, the current image can be updated by DIP functions without defining global variables.

In your DIP function implementation, there are two ways to save the target image information in R[ ][ ], G[ ][ ], B[ ][ ]. Both options work and you should decide which option is better based on the specific DIP manipulation function at hand.

**Option 1: using local variables** You can define local variables to save the target image information. For example:

```
void DIP_function_name()
{
unsigned char RT[WIDTH][HEIGHT];  /* for target image data */
unsigned char GT[WIDTH][HEIGHT];
unsigned char BT[WIDTH][HEIGHT];
}
```

Then, at the end of each DIP function implementation, you should copy the data in RT[ ][ ], GT[ ][ ], BT[ ][ ] over to R[ ][ ], G[ ][ ], B[ ][ ].

**Option 2: in place manipulation** Sometimes you do not have to create new local array variables to save the target image information. Instead, you can just manipulate on R[ ][ ], G[ ][ ], B[ ][ ] directly. For example, in the implementation of Negative() function, you can assign the result of 255 minus each pixel value directly back to this pixel entry.

# 2 Script File

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

1. Start the script by typing the command: *script*
2. Compile and run your program
3. Perform the following operations
   - Load *balloons.ppm*
   - Make a negative of the current image and save it as *negative*
   - Reload the original picture *balloons.ppm*
   - Flip the current image horizontally and save it as *hflip*
   - Reload the original picture *balloons.ppm*
   - Flip the current image vertically and save it as *vflip*
   - Reload the original picture *balloons.ppm*
   - Rotate the current image 180 degree and save it as *rotate*
   - Reload the original picture *balloons.ppm*
   - Zoom in the current image 2X and save it as *zoomin*
   - Reload the original picture *balloons.ppm*
   - Overlay the current image with image balloon and save it as *overlay*
   - Reload the original picture *balloons.ppm*
   - Age the current image and save it as *age*

   For the bonus parts only:
   - Blur the aged image and save it as *age*
   - Reload the original picture *balloons.ppm*
   - Detect the edges of the current image using $difference\_threshold = 60$ and save it as *edges*(only if you do the bonus points)
4. Exit the program
5. Stop the script by typing the command: *exit*
6. 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 we may consider them when grading! You don't have to submit any images.