EE231000 Introduction to Programming

Lab13. Image Processing

Due: May 31, 2014

Today's digital images are composed of individual picture elements, call pixels. With a large number of pixels, the image can be clear. But, with a large number of pixels the image file would need a larger disk space for storage. Thus, most of the popular image file standards involve some compression scheme to reduce the storage overhead. In this lab, we'll concentrate on one of the simplest color image format, portable pixmap format (PPM).

An image can be represented by a two-dimension array of pixels, the x-direction is the width and y-direction is the height of the image. In PPM, each pixel is represented by 3 bytes for red, green and blue colors. Thus, the total number of color levels is 2^{24}, ∼16M. The format of a PPM file is

```
P6
W H
255
```
$\texttt{R}_0\texttt{G}_0\texttt{B}_0\texttt{R}_1\texttt{G}_1\texttt{B}_1 \cdots \texttt{R}_N\texttt{G}_N\texttt{B}_N$

Where `W` and `H` are two integers and 255 is the intensity level for each color components (R, G, and B). Since each color component R, G, B takes one byte, the intensity level is 255. After the header, the remaining file consists of `3N`, `N=W×H`, bytes, each byte representing the intensity of one of the color components. This file can be read as following: the first line read using a string of two characters, the second line read using `"%d %d"` for two integers, the third line read using a string of 3 characters, then the remaining file read using `3N` characters. Note that the new-line character after the 3rd line should be handled carefully to ensure the image are read in correctly. Also, the pixels are arranged in column-major fashion, instead of row-major matrix storage use by `C` compiler.

In this lab, you will need to read in an image file, an EE Department logo and the NTHU logo. You will need to covert the original color image to a **blue-and-white** image, add the EE Department logo to the lower right corner of the image and the NTHU logo to the center of the image. The EE department logo retains its colors, but the NTHU logo modifies the image to a purple tone.

To convert a color image to a black-and-white image, the following method has been popular. Let R_c, G_c and B_c be the intensities of a color pixel and R_g, G_g, B_g be the intensities of the gray-scale image, then

$$R_g = G_g = B_g = R_c \times 0.2126 + G_c \times 0.7152 + B_c \times 0.0722.$$

Note that when $R = G = B$ at a pixel, this pixel has a gray color. And the levels of gray scale is 256. When $R = G = B$ =255, the color is white and when $R = G = B = 0$, the pixel has the color black.

However, in this lab we will convert the image to a **blue-and-while** image, which can be done simply making the blue component equals to the maximum intensity at all time. Thus, when the gray level of the pixel is 255, we see a white pixel as it should be. And when the gray level is 0, we will see a blue pixel instead. In this way, a blue-and-white image is obtained.

When placing EE Department logo, the pixels of the original image are simple replaced by the logo pixels *when the logo pixel is not white*. But in placing the NTHU logo, we simply set the red component of the pixel to be 255 *when the NTHU logo pixel is not white*. This will make the logo purple since both blue and red color components are always on.

Five image files are available for you to test your program: `pic1.ppm`, `pic3.ppm`, `pic7.ppm`, `pic8.ppm`, and `pic9.ppm`. The EE Department and the NTHU logos are also given: `EElogo.ppm` and `NTHU.ppm`.

Your program needs to read an image file, EE Department log and NTHU logo files and produce an output file. Thus, the execution of your program should be invoked by the following command line

```
$ ./a.out ~ee231002/lab13/pic1.ppm ~ee231002/lab13/EElogo.ppm \
~ee231002/lab13/NTHU.ppm pic1_out1.ppm
```

Where `pic1.ppm` and `pic1_out1.ppm` can be replaced by other image input and output files.

In order to read from a file, a `FILE` variable needs to be declared, the file opened and assigned to the file variable, then `fscanf` can be used to read and `fprintf` for write. After all data have been read and written, those file variables should be `close`d. The following example shows reading a integer from a file `data.in` and write to `data.out`.

```
FILE *fin,*fout;
int k;

fin=fopen("data.in","r");
fout=fopen("data.out","w");
fscanf(fin,"%d",&k);
fprintf(fout,"%d\n",k);
fclose(fout);
fclose(fin);
```

Of course, your can use `fopen(argv[1],"r");` to open a file specified by the command line argument.

Notes.

1. Create a directory **`lab13`** and use it as the working directory.
2. Name your program source file as **`lab13.c`**.
3. The first few lines of your program should be comments as the following.

   ```
   /* EE231002 Lab13. Image Processing
   ```

```
   ID, Name
   Date:
*/
```

4. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee231002/bin/submit lab13 lab13.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee231002/bin/subrec
   ```

   It will show the last few submission records.

5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.

6. An example of data structure to store the PPM image is shown below. You can but not required to use this data structure.

```
typedef struct sPIXEL {     // a single pixel
    unsigned char r,g,b;    // three color components
} PIXEL;

typedef struct sIMG {       // an image of PPM style
    char header[3];         // header, either P3 or P6
    int W,H;                // width and height of the image
    int level;              // intensity level of each color component
    PIXEL **PX;             // two-dimensional array for all the pixels
} IMG;
```