Computer Structure 89-230
2022-2023 תשפ"ג

Exercise 4

Submission Guidelines

1. Work individually
2. Submission date is Sunday, 8.1.2023, at 23:30
3. Submission is through the "submit" system
4. Make sure that your solution compiles and runs with no errors on the server "planet"
5. On the first row of every file you submit, write in a comment your id number and full name. For example:
   # 123456789 Israela Israeli

General Background

In this exercise you are given a valid code and your goal is to make it run as fast as possible, using any tools, techniques and principals that you've learned during the course, without changing the original output. The grade will be given based on how much you've been able to shorten the run-time of the program and also with respect to your fellow students, where a shorter run-time will gain a higher grade.

Note: The only files that you are allowed to change are the following:

- myfunction.c, myfunction.h
- myfunction1.c, myfunction1.h

You may conduct any change you see fit, including deletion and insertion of methods, algorithmic changes, usage of "magic numbers" and so forth, as long as the original output stays identical.

Installation

In your Terminal execute the following command:

```
sudo apt-get install freeglut3-dev libxmu-dev libxi-dev
```

The Code

All code files are included in ex4_files.zip. In the file myfunction.c, the method myfunction receives an image and another value.

1. If the value is 1, the method performs smoothing of the image and then sharpening of the smoothed image. Both the smoothed and sharpened images are saved as bmp files. The writing process is conducted by a call the the method writeBMP, that is located in file writeBMP.c, which you are not allowed to change.

2. If the value is 2, the method performs a different smoothing and sharpening processes, and then writes the new images, as described above.

3. If the value is different than 1 or 2, the method performs smoothing that uses an additional filter, and then again performs sharpening. The new images are saved as described above.

4. In case no other value is input after the image, an error message will be shown and the program will be terminated.

Representing Images Digitally

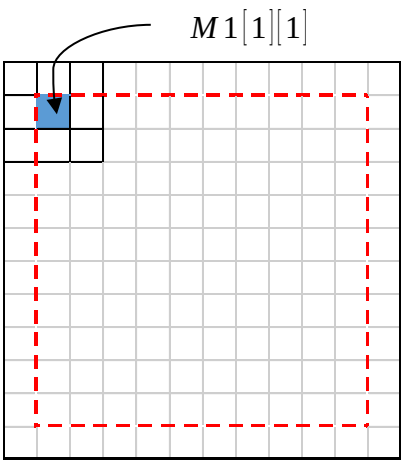
An image is saved as a matrix, where every cell holds the value of a pixel. A common representation for colored images is RGB: every cell then holds 3 values – red, green and blue. Each color is represented by a mixture of these three colors (also called channels).

Smoothing and Sharpening of an Image (Convolution)

Smoothing of an image is obtained by replacing every pixel by an average of the pixels around him using window of some size, for instance a 3X3 window, where its center is the original pixel location. Smoothing in RGB is done separately for each channel.

For example, given an image M1 of size NXN and some channel (R, G or B), the pixel in the [1,1] location will receive the following value:

1 $$M2[1][1]=\frac{\sum_{i=0}^{2}\sum_{j=0}^{2} M1[i][j]}{9}$$



Where M2 is the image after smoothing. Notice that the values of the neighboring pixels are taken from the original image.

Further note that only pixels that can be “placed” in the center of a window are affected by the smoothing process. In the illustration above these are the pixels that lay inside the area marked by a red dashed line. Formally, these are pixels with row index i that satisfies $\frac{kSize}{2} \le i < N - \frac{kSize}{2}$ and column index j that satisfies $\frac{kSize}{2} \le j < N - \frac{kSize}{2}$, where $kSize$ is the window size (in the example above $kSize = 3$).

After applying the window over all relevant pixels in the original image, the obtained image seems more smooth (also blurred), hence the operation is called smoothing.

Let us take a matrix K of size 3X3 (which is called a mask, or kernel) and use for smoothing. If K is of the form:

$$K=\frac{1}{9}\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}=\begin{bmatrix}\frac{1}{9} & \frac{1}{9} & \frac{1}{9}\\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9}\\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9}\end{bmatrix}$$

Then, the above formula for M2[1][1] is equivalent to the following formula:

$$M2[1][1]=\sum_{i=0}^{2}\sum_{j=0}^{2}M1[i][j]\cdot K[i][j]$$

In fact, this operation can be generalized to use any set of weights for the value of K, to obtain different weighted averages for the pixels in the center of the mask, which correspond to different meanings in the resulting image.

For example, for K of the form

$$K=\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix}$$

a sharper image is obtained.

Generally, applying a mask over an image and calculating the resulting weighted sum (corresponding to the mask) for each pixel is called convolution.

Compilation

A Makefile is attached along with all other code files in ex4_files.zip. The Makefile should not be changed, as it is the same one that will be used for compilation when testing the solutions. As usual, in order to compile simply execute the command make.

Running the Program

After successful compilation the executable file showBMP in created. Running the program is done as follows:

```
./showBMP <imageName> <kernelOption>
```

For instance:

```
./showBMP gibson_500.bmp 1
```

or

```
./showBMP gibson_500.bmp 2
```

or

```
./showBMP gibson_500.bmp 3
```

These commands will run the program for the image file gibson_500.bmp received as the first input and apply specific masks and filters when smoothing and sharpening according to the value of the second input.

Note: you may assume that each image file received as input will correspond to a square image of size $n \times n$, with $100 \leq n \leq 1000$.

Measuring performance

The changes you apply in the code should affect how the code is run, but not the final result. Verify that your code and the original code create the same output images.

To facilitate in testing your code, ex4_files.zip includes three directories with images in sizes 200X200, 500X500 and 700X700. Each of these directories already includes the ground truth, namely the images created by applying the original code, with the following names:

- For input '1': Blur_correct.bmp for the smoothed image, and Sharpen_correct.bmp for the sharpened image.
- For input '2': Row_Blur_correct.bmp for the smoothed image, and Row_Sharpen_correct.bmp for the sharpened image.
- For input different than '1' or '2': Filtered_Blur_correct.bmp for the smoothed image, and Filtered_Sharpen_correct.bmp for the sharpened image.

You may use the command cmp to compare between these images and your newly generated images to make sure they are the same.

In order to test performance on more images that meet the criteria, you can utilize image search online, for example by using https://images.google.com/ and searching by imagesize:WIDTHxHEIGHT (for instance imagesize:200x200). After downloading the image to your computer you may use any tool for conversion to bmp format, for example by uploading to https://online-converting.com/image/convert2bmp/ and re-saving the resulting image (after conversion). Notice specifically that on this website the default choice is to save the new image with the definition image depth = 24 bit which is what we require (as we have 3 colors, each described using 8 bits). This can be verified as follows:

Lastly, it is important to note that the run-time of the same program can change between two subsequent runs on the same input, and this occurs mainly due to changing write times to the file (as every run writes and saves two new images). Thus, in order to analyze the running time you obtain, it is advised to average over at least 10 identical runs.

Submission

You have to submit only the following files: myfunction.c, myfunction.h, myfunction1.c, myfunction1. The rest of the code files will be identical to those in ex4_files.zip.

Grading

A significant amount of points will be reduced for an output that is not identical to that of the original code. Given a valid output, the primary parameter for grading will be the average run-time obtained over multiple tests, where shorter is better. In addition, creativity and extraordinary effort can also gain extra points.

Remarks

1. We will test images of different sizes, all abiding $100 \leq n \leq 1000$.
2. Use comments to explain your code. This can be important for appeals.
3. The solutions will be tested on the server planet. Remember that the final grade for the task is comparative to other students, hence you can ignore the grade that may appear in the automatic email that is received upon submission to submit.

Good luck!