

COMP 206 Fall 2018 – Assignment 5

November 27th, 2018

Note that this is only make-up work:

This assignment is not one of the “core” 4 assignments and should not be done by most people. This is considered a make-up assignment only. Note that the difference between $40/3=13.3$ and $40/4=10$ is 3.3%, and we would in general expect this assignment to take more effort than this would be worth. Therefore, this assignment is only a final back-up for a few special cases. Anyone can of course complete it for unmarked extra practice.

Objectives

Re-visit several elements of Assignment 3 (processing binary image data) and Assignment 4 (web/CGI). We will provide you code that is similar to the solution of Assignment 3 and implements the basics of BMP file input and output. You must understand these and use them to implement a new function “filtering”, both on the terminal and via CGI. We will also practice the common system feature of command history.

Background on Image Filtering:

A common operation in graphics and video production is to apply a so-called filter to an image through the process of **convolution**. By filtering, we can extract the edges within an image, blur it, and accomplish several visually interesting and useful effects. For example, this is a common edge filter:

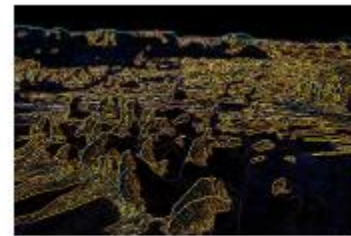


utah.bmp

The left image filtered with...

1	1	1
1	-8	1
1	1	1

Becomes the right image...



utah_edges.bmp

Applying a filter with convolution means “placing” the filter centered at every pixel in the image, multiplying each pixel in the region with its corresponding filter element, summing values and placing them into the output. More background and examples are at:

[http://en.wikipedia.org/wiki/Kernel_\(image_processing\)](http://en.wikipedia.org/wiki/Kernel_(image_processing)).

Getting Started:

Download the Assignment5.zip file from GitHub, unzip it and read all the provided code. This will help you get started

Submitting:

As this Assignment is “make-up” in nature, when we confirm that your case requires a make-up, then we’ll find a submission plan that suits the situation.

When finished, create a zip file with all code and submit to the Assignment 5 folder on My Courses.

Question #1 – Simple filtering (30 marks)

Create a C program, that will filter the BMP images in the Assignment 5 directory based on a filter specified on the command line. The following arguments are necessary:

- The input BMP image path
- The output BMP image path
- The width of the square image filter, W , as one integer number
- The filter weights, as $W \times W$ floating point numbers

Note that applying a filter near the image boundary is undefined. You should set all the boundary pixels that cannot “fit” the filter to black. We recommend that you use the code in `A5_provided_functions (.c and .h)` to interact with the BMP image. In particular, the 3D array format should make it very easy to loop over the pixels and apply the filter.

Place all of your code in the file `q1_image_filter.c` so that the command “`$ make q1_test`” succeeds and produces the desired outputs shown on the last page of this document.

Question #2 – Command History (50 marks)

Extend the functionality of Q1 by adding a session history that allows undo and redo, like a text editor. Each call to your program can now take any of 4 commands, each with its own sub-arguments, as follows:

- **LOAD** `input_image_path`: loads the specified file, discards any existing history and makes the new image the active output.
- **FILTER** `filter_width filter_weights` (same argument format as Q1): applies the specified filter to the previously active image. The filtered result becomes the new active image and is placed at the next step in the history. Any previous history from this point forwards must be truncated (e.g., if we had done undo-undo-undo, then filter, it is no longer possible to redo the 3 “undone” operations, but we can go back past the filter towards the load command).
- **UNDO**: most the active image backwards by one step in history, if this is possible (otherwise write a short error to stdout and leave the active image and history as is).
- **REDO**: moves the active image forwards by one step in history, if this is possible (likewise, if not possible, write error and leave everything alone).

Again, using the code in `A5_provided_functions` will help with this. Also, this time we have given you a starter template for `q2_filter_with_history.c` with a suggested data structure for the history. You can choose to use your own approach, but completing our code is likely to be the quickest method.

The active image must always be written in the present working directory as “`result.bmp`”. In order to implement the history, you are only allowed one additional file called `history.dat`. The main trick to this question is coming up with a good scheme for how to store the history and implementing it with each command. Test with “`$ make q2_test`”.

Question 3: Web Filtering

Adapt the functionality from Q2 to form a C CGI that will call the load, filter, undo, redo, as commanded by a user via a web browser. Model your interaction after the example here:

https://www.cs.mcgill.ca/~dmeger/A5Q3_template.html

Note that example was from a previous year and handles the image upload, while you should instead scp the whole A5 folder within your public_html on mimi, including the provided BMP files. Then, select which file to load from those on the server by typing the file name (the reason for this: it's more difficult to access the entire image data uploaded using C).

Your CGI interface must only allow 3x3 filters to be entered. Each time the user requests a filter operation, the next page shown must have all the same filter elements and the active image must show on the page (output.bmp) instead of the utah.bmp, the default file.

Your CGI code must be completed within `q3_filter.cgi.c`. In order to have it connect with the provided HTML, you must compile your executable as:

```
$ gcc q3_filter_cgi.c -o ~/public_html/cgi-bin/q3_filter.cgi
```





You just write the initial HTML file to point to this code and start the filtering interaction, as: `q3_filter_entry_point.html`.

Filters to Try

Here are some good test cases that give interesting output images (each case shows the width and filter coefficients just as you must type them to Q1 or Q2, for example):

- Identity Filter: 1 1
- Emboss Filter: 3 -2 -1 0 -1 1 1 0 1 2
- Mean Filter: 5 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04
0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04 0.04
- LoG Filter : 9 0 1 1 2 2 2 1 1 0 1 2 4 5 5 5 4 2 1 1 4 5 3 0 3 5 4 1 2 5 3 -12 -24 -12 3 5 2 2 5 0 -24 -40
-24 0 5 2 2 5 3 -12 -24 -12 3 5 2 1 4 5 3 0 3 5 4 1 1 2 4 5 5 5 4 2 1 0 1 1 2 2 2 1 1 0

Visual Examples

 Identity	1
 Emboss	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$
 Mean filter	$\begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \end{bmatrix}$
 Laplacian of Gaussian (LoG)	$\begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -24 & -40 & -24 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$