Digital Image Processing Laboratory: Image Filtering

July 10, 2014

0. Introduction

In this laboratory, you will filter full color images using both FIR and IIR filters, and you also will learn to use MATLAB® to read, display, and export your images. In this laboratory assignment, the filter algorithms should be implemented in the C programming language, unless otherwise stated. Low level programming languages are necessary for implementing many complex image processing operations, so one objective of this laboratory is to give you some experience in using C to solve image processing problems. Simple example programs are provided, so you should be able to write the required programs even if you have very little previous experience with C. When processing images on a computer, there are usually special cases that must be properly handled. In particular, all filters will be implemented using **free** boundary conditions along edges, and pixel values will be clipped to a range of [0, 255].

- **Free** boundary conditions - In order to understand a **free** boundary condition, consider an M×N image denoted by img(i, j) where $0 \le i < M$ and $0 \le j < N$. Some filtering operations require that you index outside of this valid range. Pixels indexed outside this range should be considered equal to zero.
- Clipping - Image processing operations will sometimes cause a pixel to exceed the value 255 or go below the value 0. In these cases, you should clip the pixel's value before displaying or exporting the image.

$$y(m,n) = \begin{cases} \mathrm{y(m,n)} = 0 & \text{if } \mathrm{x(m,n)} < 0 \\ \mathrm{y(m,n)} = 255 & \text{if } \mathrm{x(m,n)} > 255 \\ \mathrm{y(m,n)} = \mathrm{x(m,n)} & \text{if } 0 \le \mathrm{x(m,n)} \le 255 \end{cases}$$

1. C Programing

An important goal of these laboratories is for you to learn good C programming skills. You do not need much prior experience in C programming, because the laboratories will lead you through the use of the language with examples and program templates. However, to learn good C programming skills, you should strictly follow the following set of rules at all times.

- Use strict ANSI C constructs - All C compilers allow for enforcement of strict ANSI C. You should use these options at all times. With the gcc compiler under the Linux operating system, the compiler flags are gcc -ansi -Wall –pedantic. These flag settings will insure that you receive warnings for any poor programming constructs in your code.

- Never use global variables - Global variables should never be used. There are exceptions to this rule, but you will never face a situation where they are truly needed. Global variable will lead to “spaghetti code,” that cannot be understood or debugged.

- Fix all compiler warnings - Any warning from the compiler means that you have an error in you program or are using poor programming style. Either problem is serious and should be corrected. Typically, warnings are generated when variables are not properly retyped. The only exception to this rule is when you dynamically allocate memory. In this case, you may receive a warning about the byte location.

- Use prototypes for all subroutines - Every subroutine you write should have a prototype that defines its argument types. This prototype should generally be contained in a single header file with a name such as **defs.h**.

For this exercise, we will be using the image that you created in class. Please convert it to a Portable Bitmap format (ppm). I used Photoshop to do the conversion. This image will be used in this project.

2. Displaying and Exporting Images in Matlab

Most of the lab exercises in this class will require you to produce one or more output images. These will need to be incorporated into your report either in the form of a hard copy, or imported into an electronic document. Since many exercises will use Matlab, we will describe here some basic I/O and display commands in the Matlab environment.

- Reading Images - You can read an image file, img.ppm, into the Matlab workspace using the command

  ```
  x=imread('img.ppm');
  ```

  This will produce an image matrix x of data type uint8.

- Displaying Images - You can display the image array x with the following commands,

  ```
  image(x);
  colormap(gray(256)); % only needed for gray scale image
  truesize;
  ```

  If x is a gray scale image, the colormap function is needed to tell Matlab which color to display for each possible pixel value. The truesize command, which is included in the image processing toolbox, maps each image pixel to a single display pixel to avoid any interpolation on the display. This will be important in future labs.

- Writing Images - When producing a lab report document, you should strive to present the best representation of your output images. Therefore it is best to export your images to a lossless file format, such as PPM, TIFF or BMP, which can then be imported into your lab report document. You can write the image array x to a file using the imwrite function:

  ```
  imwrite(x,'img_out.ppm')
  ```

  Note that if x is of type uint8, the imwrite function assumes a dynamic range of [0,255], and will clip any values outside that range. If x is of type double, then imwrite assumes a dynamic range of [0,1], and will linearly scale to the range [0,255], clipping values outside that range, before writing out the image to a file. To convert the image to type uint8 before writing, you can use

  ```
  imwrite(uint8(x),'img_out.ppm')
  ```

  ```
  imwrite(uint8(x),'img_out.tif')
  ```

In most of the following exercises, your C routines will output PPM files so these can be imported into your lab document directly. For your own viewing you can display the output images using Matlab as described above, or using any other standard image viewing software. Regardless, you are encouraged to experiment with the effect of each of the above commands using one of the images provided on the lab web page.

3. Converting to Grayscale using MATLAB equation

In this problem, you will convert from an RGB image to a Grayscale image. MATLAB uses a simple formula

$y = 0.299\ R + 0.59\ G + 0.11\ B$ from page 29 in textbook

1. Download the bundle PPM.zip containing the C source code for reading and writing Portable Bitmap (PPM) images. Compile the application using Visual Studio C++.
2. Convert your picture to a grayscale image using MATLAB to see what it should look like. I used `rgb2gray()` to convert my image.
3. Now use the above formula to convert your image to grayscale in the C++ code. You may use RGB all the same values to simulate gray scale.
4. Compare the two images. Are they the same? Include the filtered images in your report.

Section 3 Report:

Hand in:

1. The original color image.
2. The original MATLAB grayscale image and your program generated grayscale image.
    a. Consider using subplot for these images
3. Compare the two images. Are they the same?
4. A listing of your C code.

4. Laplacian Contrast Enhancement

In this problem, you will analyze and implement a simple low pass filter given by the 3×3 mask

$$\lambda_1 = \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} \qquad \lambda_2 = \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

This low pass filter application is known as convolution.

1. Download the bundle PPM.zip containing the C source code for reading and writing Portable Bitmap (PPM) images. Compile the application `Driver` using the Visual Studio compiler.
2. Modify the program `Driver` so that it filters the red, green and blue components of your input image with each of the λ filters and generate an output image.
    a. Use your original color image. (two output images, i.e., λ_1 and λ_2)
    b. Use the grayscale image you created from section 3. (two output images)

    Warning: Make sure to rename the directory containing your modified program so that it cannot be overwritten with the original PPM program directory.
    Include the filtered images in your report.

Section 4 Report:
Hand in:
1. The original color image and the two filtered color images.
2. The original grayscale image and the two filtered grayscale images.
    a. Consider using subplot for these images
3. Which worked better, the color image or the grayscale?
4. A listing of your C code.