MSc in Robotics

Programming Methods for Robotics Assignment

Irene Moulitsas & Peter Sherar
Cranfield University
Hand in date: 11/01/19 (FT), 25/01/19 (PT), 2:00pm

1. Introduction

In this assignment you are asked to write and test some C++ code for performing filtering operations that can be applied to digital images. Some existing image handling code is made available to you through Blackboard.

Background

Suppose we are given an image, and we can access each pixel in the natural way, that is, using two coordinates [x, y] (row x, column y). Each pixel is a combination of three colours: Red, Green, and Blue, each colour represented by an integer value from 0 to 255. (0 means complete absence of the colour, 255 means the colour participates with full intensity.)

For this assignment you are going to use a particular image format called ppm.

The ppm image format consists of a header followed by the image pixel data. The header contains the following information:

1. A number which indicates the type of storage used for the pixel values of the image. If the number is P3 it means that the pixel data is stored in ascii text format which is a seven bit character code. Each byte of an ascii text file is interpreted as one ascii character.

   If the number is P6 it means that the pixel data is stored in compressed binary format following the header. Here all eight bits of each byte are used.
2. Optional comment which begins with the # tag.
3. Width, height and maximum colour value of the image (usually 255)

The pixel data that follows consists of RGB values in the range 0-maximum colour value

Example ppm file

The following data represents an ASCII ppm image for a 200 by 200 pixel red box:

```
P3
#RedBox.ppm
200 200
255
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
```

255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
....
and so on for a total of 4,000 rows.

Applying a filter to an image

Suppose we want to "sharpen" an image. What we need to do is "pass a filter" over every pixel in the image. For example, a sharpening filter could be the following:

-1	-1	-1
-1	9	-1
-1	-1	-1

To *pass this filter over the image* means the following: For every pixel [x, y] of the image consider first its red-value component. This will be an integer between 0 and 255, as explained above -- call it R for ease of reference. Now imagine that R is placed at the *centre* of the 3x3 filter shown above, i.e., at the location of number 9. Multiply R by 9. Now do the same for each horizontal, vertical, and diagonal *neighbour* of pixel [x, y], that is, take the pixel [x-1, y-1] and multiply its red-value component by -1 (because that is the value in the corresponding location in the filter); same for pixels [x, y-1], [x+1, y-1], [x-1, y], and so on. Finally, *sum up* these nine products. The result is the red-value component of the new ("filtered") pixel. Do the same for the green and blue components, and you have the whole pixel of the new image. If the values are less than zero, you make them zero; and if they are over 255, you make them 255. This procedure can be repeated for every pixel of the original image *except* the ones at the very edge (first & last row, leftmost & rightmost column); you ignore those pixels and copy them directly from the original image.

The three filtering operations relevant for the assignment are the following:

Smooth

Smoothing is an operation used to reduce noise within an image or to produce a less pixelated image. A filter for smoothing an image is as follows:

0	1	0
1	0	1
0	1	0

Sharpen

Sharpening an image Increases contrast and accentuates detail in the image or selection, but may also accentuate noise. The filter below uses appropriate weighting factors to replace each pixel with a weighted average of the 3x3 neighbourhood:

-1	-1	-1
-1	12	-1
-1	-1	-1

Edge Detection

An edge detector highlights sharp changes in intensity in the active image or selection. Two 3x3 convolution kernels (shown below) are used to generate vertical and horizontal derivatives. The final image is produced by combining the two derivatives using the square root of the sum of the squares.

The final integer RGB values are computed by rounding the square root value obtained to the nearest integer.

vertical filter

1	2	1
0	0	0
-1	2	-1

horizontal filter

1	0	-1
2	0	-2
1	0	-1

2. Tasks

You are provided with the following code:

1. A Pixel class with implementation for reading, writing, setting and getting, and a few other operations on RGB values.

2. The following 3 image functions for reading, writing and converting ppm files in binary to ascii format:

```
/* opens a binary ppm file for reading and opens an asci ppm file
for writing */
void openIOFiles(ifstream& fin, ofstream& fout,
                                                char inputFilename[]);

/* converts a binary image data file to P3 ascii format */
void convertP6ToP3(ifstream& bin, ofstream& out,
                   vector<vector<Pixel> >& image, int info[1]);

/* write P3 header and image data to a file */
void writeP3Image(ofstream& out, vector<vector<Pixel> >& image,
                                  char comment[], int maxColor);
```

There are also three helper functions used by the above image functions:

```
/* reads binary image data and writes asci image data (called from
convertP6ToP3) */
void readAndWriteImageData(ifstream& fin, ofstream& fout,
                   vector<vector<Pixel> >& image, int w, int h);

/* reads the header from a ppm file (called from convertP6ToP3) */
void readHeader(ifstream& fin, ofstream& fout, int imageInfo[]);

/* writes the header for a ppm file (called from readHeader and
writeP3Image) */
```

```
void writeHeader(ofstream& fout, char magicNumber[], char comment[],
                                    int w, int h, int maxPixelVal);
```

The filtering functions to write are:

```
void smooth(vector<vector<Pixel> >& image);
void sharpen(vector<vector<Pixel> >& image);
void edgeDetection(vector<vector<Pixel> >& image);
```

Using the functions provided and the above filtering functions, you also need to write a main function which should perform the following sequence of operations:

- Open the binary ppm image file
- Convert the binary file to P3 format
- Perform the filtering operation on the pixel data
- Write the P3 image file containing the filtered pixel values

Test your code on the images provided in the zip file.

3. Source Code and Report Requirements

The source program will need to compile on the IT lab machines using Visual Studio 2015 or 2017 without any other external dependencies/libraries/source codes of third parties.

Write a report to present and discuss your findings. The report should be no less than 2,000 words and must not exceed 3,000 words. The report can contain any number of figures/tables, however all figures/tables should be numbered and discussed. The report should include a description of your implementation explaining the method used. The source code should be included as an Appendix to the report.

4. Assignment Submission

The source code files should be submitted electronically via the **Blackboard submission point** by 2:00 pm on 11th January (full-time students) or the 25th January (part-time students).

The report should be submitted electronically via the **TurnItInUK submission point** by the prescribed deadline, for the assignment submission to be considered complete.

5. Marking

The assignment will be assessed based on the following marking scheme:

- 20% Introduction, methodology, conclusions
- 40% Source code, commenting
- 30% Analysis of the results
- 10% Report structure, presentation, references