PA3

PROGRAMMING ASSIGNMENT 3
BME 121, FALL 2016

Due Fri Nov 11 at 4:30 pm

ASSIGNMENT

In this assignment, you need to convert an input image to an output image where the output image is formed by applying "Kirsch edge detection" to the input image. Edge detection is often a part of medical image processing.

The files "21_training.tif" and "21_training.csv" contain the input image.

The files "21_training_edges.tif" and "21_training_edges.csv" contain my output (for comparison).

The input image "21_training.tif" is a color image of a retina taken from the DRIVE (digital retinal images for vessel extraction) database at the url http://www.isi.uu.nl/Research/Databases/DRIVE/. Details of the database can be found in the following paper.

- J. J. Staal, M. D. Abramoff, M. Niemeijer, M. A. Viergever, B. van Ginneken, "Ridge based vessel segmentation in color images of the retina", *IEEE Transactions on Medical Imaging*, 2004, vol. 23, pp. 501-509.

The original image is in a format called TIFF (tagged image file format) which I hope is viewable on your computer.

To process an image in a C# program, we usually convert it to a two-dimensional array of pixels. The class "System.Drawing.Bitmap", which one would normaly use to do this in C#, has not yet been ported to the .Net Core framework in which we are coding. To get around this, I have written two C# programs to convert TIFF images back and forth from a simple CSV (comma-separated values) text format which we can easily read or write. Executable versions of these programs are provided in the files "TiffToCsv.exe" and "CsvToTiff.exe" which will probably only run on Windows computers. I'll see if we can post executable versions for the Mac as well. Your program will take the input image in CSV format and produce the output image in CSV format. The TIFF converter is only useful if you want to view the resulting output image.

A starting version of the assignment is given in the file "pa3-start.cs". You will need to complete it by completing the part which reads the input image from its CSV file, the parts which do the Kirsch edge detection, and the part which writes the output image to its CSV file. Details on the pixel format, the CSV file format, and Kirsch edge detection follow.

PIXEL FORMAT

Each pixel in a color image can be represented by an object of type System.Drawing.Color. Struct System.Drawing.Color is supposed to be part of the .Net Core API (according to docs.microsoft.com) but I can't find it. Thus, the starting solution contains my implementation of the needed parts of the System.Drawing.Color struct. In particular, you can create a Color object using the static method Color.FromArgb( ). You can extract parts of the Color object using the get-only properties A, R, G, and B.

One pixel (one Color object) uses a format called ARGB. This has four integer-valued components called alpha, red, green, and blue. The transparency of a pixel is represented by alpha (from 0 being fully transparent to 255 being fully opaque). The intensities of each of the additive primary colors red, green, and blue are represented by the other three parts (from the lowest intensity being 0 to the highest intensity being 255). Each of these can be fit into one byte so the four pieces can be put together in one 32-bit integer when efficient storage is necessary. Here, we will just use "int" for everything. In this assignment, every pixel in both images will have alpha=255 (i.e., every pixel is fully opaque).

CSV FILE FORMAT

The images are stored in a very simple CSV format. Line 1 contains only one integer giving the height (number of rows) in the image. Image rows are numbered from top to bottom. Line 2 contains only one integer giving the width (number of columns) in the images. Columns are numbered from left to right. Following these are multiple longer lines, one for each row in the image, with each line containing integers separated by commas. The first four integers are the alpha, red, green, and blue values for the first pixel in that row. The next four integers are the alpha, red, green, and blue values for the next pixel, and so on to the end of the row.

KIRSCH EDGE DETECTION

In edge detection, you essentially replace each pixel by an estimate of the first derivative of image intensity in the neighbourhood of that pixel. Here, we use a small neighbourhood comprising the eight pixels surrounding a given pixel. The Kirsch edge detector forms derivative estimates in eight different directions and keeps the maximum.

The following are called the Kirsch edge operators (or convolution kernels). If you think of the shaded square from one of them lined up with one image pixel in the input image, the eight surrounding squares line up with eight surrounding image pixels. You

form a sum of each number in the operator multiplied by its associated pixel (an operation called convolution). After doing this separately for the eight operators, you use the maximum of the eight as the value of the output image (in the pixel position where you aligned the shaded square). The input image should not be altered.

5	**5**	**5**
-3		-3
-3	-3	-3

-3	**5**	**5**
-3		**5**
-3	-3	-3

-3	-3	**5**
-3		**5**
-3	-3	**5**

-3	-3	-3
-3		**5**
-3	**5**	**5**

-3	-3	-3
-3		-3
5	**5**	**5**

-3	-3	-3
5		-3
5	**5**	-3

5	-3	-3
5		-3
5	-3	-3

5	**5**	-3
5		-3
-3	-3	-3

To handle the color image, we apply Kirsch edge detection separately to each of the red, green, and blue components. The alpha component is not altered.

To avoid overflowing the representation, each of the red, green, and blue values in the output image must lie between 0 and 255. If the Kirsch edge detection gives a value less than zero, use zero. If it gives a value greater than 255 use 255.

Rather than use two-dimensional arrays, I used separate variables for the eight neighbours, numbered as shown below. This makes the code easier to write. Then, the only arrays needed are for the input image and the output image.

1	2	3
4		5
6	7	8

The starting point in "pa3-start.cs" uses one method to return a Kirsch edge-detection intensity given eight image pixel intensities for the eight numbered neighbouring pixels shown above. Another method is given nine Color values for pixels at the eight numbered positions above plus the centre position. It uses the first method to find the Kirsch edge-detection value for each of the color components (i.e, calling the first method with eight red values, then with eight green values, then with eight blue values) and forms its resulting Color object from these plus the alpha value copied from the centre pixel.

Note that pixels along any edge of the input image don't have eight neighbours. Here, we just skip those in processing, that is, along all image edges, the output image will have the same pixel values as the input image.

SUBMISSION

Name your C# program file as pa3.cs. Use Bme121.Pa3 as your namespace identifier. Include the standard doc-comment block. Submit pa3.cs at the following url.

https://georgefreeman.ca/fileuploader