Elec4622 Laboratory Project 2, T2 2019

David Taubman

July 10, 2019

1 Introduction

This is the second of three mini-projects to be demonstrated and assessed within the regular scheduled laboratory sessions. This project is due in Week 8. The project is worth a nominal 10 marks, out of the 30 marks available for the laboratory component of the course. However, there are optional tasks which could attract bonus marks.

In this project, you will first write a program which can be used to convert grey-scale images into bi-level images, based on a user-defined threshold. This generally produces an image which contains undesirable noise artefacts, in the form of holes and unwanted speckle. You will then implement a morphological opening filter and (for bonus marks) a morphological closing filter, in order to experiment with their impact on the bi-level thresholded image.

2 Implementation Issues

2.1 Erosion

As discussed in Chapter 6 of your lecture notes for this course, we write $X \ominus A$ for the erosion of an image $X \subseteq \mathbb{Z}^2$ by the structuring set $A \subseteq \mathbb{Z}^2$ and define the operator according to

$$X \ominus A = \left\{ \mathbf{h} \in \mathbb{Z}^2 | A_{\mathbf{h}} \subseteq X \right\}$$

That is, $X \ominus A$ contains all pixels such that the structuring set A, translated to that pixel, is entirely contained in the original image X.

In implementing the erosion operator for this project, you will need to decide how to represent the image X and structuring set A in the memory of the computer. We should remember that X is just the indicator set of the image intensity function $f[\mathbf{n}] \in \{0, 1\}$, where $\mathbf{n}$ ranges over the region of support of the image. In practice, you might find it slightly more convenient to use $f[\mathbf{n}] \in \{0, 255\}$, where $f[\mathbf{n}] = 0$ means that $\mathbf{n}$ is a background pixel location and $f[\mathbf{n}] = 255$ means that $\mathbf{n}$ is a foreground pixel location. If you store all images directly as intensity values (rather than attempting to store them as sets), you will find it easier to read and write the image files.

As for the structuring set A, a strongly recommended approach is to store A as an array of displacements. Suppose, for example, that A has N_A elements $\mathbf{a}[0], \mathbf{a}[1], \ldots, \mathbf{a}[N_A - 1]$ where each element $\mathbf{a}[i] = (a_1[i], a_2[i])$ consists of a vertical (downward) displacement $a_1[i]$ and a horizontal (rightward) displacement $a_2[i]$. In this case, the morphological erosion operation $Y = X \ominus A$ can be expressed as

$$\begin{aligned} y[\mathbf{n}] &= \begin{cases} 255, & f[\mathbf{n}+\mathbf{a}[i]] = 255 \text{ for all } i = 0,1,\ldots,N_A-1 \\ 0, & \text{otherwise} \end{cases} \\ &= \bigwedge_{i=0}^{N_A-1} f[\mathbf{n}+\mathbf{a}[i]] \end{aligned}$$

where $\bigwedge$ denotes the bit-wise logical AND operation. This is an operation that you can quite readily implement efficiently in a program. If you store the image $f[\mathbf{n}]$ in a raster-scan block with width W, height H and stride $S \geq W$, you can reduce the displacements $\mathbf{a}[i]$ to memory offsets

$$a_{\text{off}}[i] = a_1[i] \times S + a_2[i]$$

Then, if $p_{\mathbf{n}}$ is a pointer to input sample $f[\mathbf{n}]$, $(p_{\mathbf{n}} + a_{\text{off}}[i])$ is a pointer to input image sample $f[\mathbf{n}+\mathbf{a}[i]]$. This means that your code for erosion could look something like this:

```
for ($n_1 = 0$; $n_1 < H$; $n_1$++)
    for ($n_2 = 0$; $n_2 < W$; $n_2$++)
      {
        $p_{\mathbf{n}}$ =image_in +$n_1 * S + n_2$;
        val = 255;
        for ($i = 0$; $i < N_A$; $i$++)
            val &= $p_{\mathbf{n}}[a_{\text{off}}[i]]$;
        image_out[$n_1 * S + n_2$] = val;
      }
```

As with LSI filtering, you will need to think about what happens at boundaries. In particular, you must be sure that the addition of $a_{\text{off}}[i]$ to the address of any given sample in the input image will leave a pointer to a valid, initialized location in memory, for each i. One way to do this is to allocate the image memory block large enough to hold a border around all boundaries (exactly as we did for LSI filtering), where the upper border is at least as large as $\max_i(-a_1[i])$, the lower border is at least as large as $\max_i(a_1[i])$, the left hand border is at least as large as $\max_i(-a_2[i])$ and the right hand border is at least as large as $\max_i(a_2[i])$.

For the purpose of consistency in realization of this project, you are required to extend boundaries using the **"symmetric extension"** method, but see the detailed discussion of boundary extension in connection with the individual tasks below.

2.2 Opening

Recall that opening involves an erosion operation, followed by dilation. In Chapter 6 of your lecture notes, we also found that opening can be represented in a more compact (and intuitively meaningful) form as

$$X \circledcirc A = \bigcup_{A_{\mathbf{h}} \subseteq X} A_{\mathbf{h}}$$

In your practical implementation it also makes very little sense to perform erosion and dilation operations one after the other, in strict sequence. Rather, it is not hard to see that they can be implemented in a combined fashion, as follows.

In order to take a union of translated structuring sets, as suggested by the above equation, you should start by initializing your output image to $g[\mathbf{n}] = 0$ for all $\mathbf{n}$ (i.e., everything starts in the background). You can then scan through the input image, looking for those locations $\mathbf{n}$ such that $f[\mathbf{n} + \mathbf{a}[i]] \neq 0$ for all i. Whenever you find such a location, you can simply set $g[\mathbf{n} + \mathbf{a}[i]]$ to 255 for each $i = 0, 1, \ldots, N_A - 1$. You should be able to readily adapt the efficient pointer manipulation techniques described above to the case of opening.

3 Tasks

For demonstration purposes, you may like to create a separate project for each task, within a single workspace. You can do this by using the "File → New → Project" option in Visual C++, replacing the "Create new solution" option with "Add to solution".

Be sure to arrange for all your executable programs to be placed in a single location (e.g., `c:\elec4642\slot1\bin` or `c:\elec4642\slot2\bin`), which is referenced from the `path` environment variable. Also, please place all the images you are working with into a single directory (e.g., `c:\elec4642\slot1\data` or `c:\elec4642\slot2\data`). That way, it will be much easier to demonstrate your work in a time effective manner – it will also save you personally a lot of time.

Task 1: (2 marks) For this task, you are to write a program which accepts three command-line arguments:

1. the name of an input image file (monochrome is fine);
2. the name of an output image file; and
3. a threshold value T in the range 0 to 255.

Your program should read the input image and write an output image whose sample values are set to 0 if the corresponding sample in the input image is less than the threshold T, and equal to 255 otherwise. Play around with your program to see what thresholded versions of the Lenna, Bike and Pens images look like.

Task 2: (4 marks) For this task, you are to write a program which accepts the following command-line arguments:

1. the name of a bi-level input image (you can assume that it has one image component with values of 0 or 255);
2. the name of an output image, whose values will also be written as 0 or 255; and
3. one or more vectors $\mathbf{a}[i]$, corresponding to the elements of a structuring set A (your program should read pairs of integers until there are no more command line arguments, so that the structuring set can be arbitrarily defined via the command line).

Your program should perform morphological erosion, following the suggestions in Section 2.1, using the structuring set A defined by the vectors $\mathbf{a}[i]$ which are supplied on the command-line. To run your program with complex structuring sets, you might need a lot of command line arguments, so you will probably find it convenient to use batch files. A batch file is any text file whose name ends in ".bat". When you type the name of such a file, its contents are executed as if you were typing them on the command line. If you like, you can put batch files anywhere in your executable path (e.g., `c:\elec4642\slot1\bin` or `c:\elec4642\slot2\bin`) and then invoke them from another directory (typically `c:\elec4642\slot1\data` or `c:\elec4642\slot2\data`).

Be sure to extend boundaries using the **"symmetric extension"** method, with odd symmetry – i.e., **do not replicate** the samples that actually lie on the input image borders.

Task 3: (1 mark) Modify the program of Task 2 (or just extend its command-line syntax) so that it can accept a radius parameter r, in place of the list of vectors $\mathbf{a}[i]$. Internally, the radius r should be converted into the list of all structuring set locations $\mathbf{a}[i]$ such that

$$a_1^2[i] + a_2^2[i] < \left(r + \frac{1}{2}\right)^2$$

Try running your program on thresholded images, with these roughly circular structuring sets of various radius values r.

Task 4: (3 marks) In this task, you should build upon what you have implemented for Tasks 2 and 3, creating a program which implements morphological opening, following the suggestions in Section 2.2 above. Your program should allow the structuring set to be specified on the command line, as a set of vectors $\mathbf{a}[i]$, or as a circle with radius r.

Try running your program on thresholded images, with structuring sets of various sizes which roughly approximate circles. Be prepared to explain the results you observe.

Task 5: (up to 3 bonus marks) In this optional task, you extend your implementation in Task 5 to implement morphological closing. One very simple way to do this is to take the complement of the input image, apply the opening operation, with a mirror image version of the structuring set, and then take the complement of the result. Complementation, of course, consists simply in exchanging 0 with 255. For the full bonus marks, however, your program should not involve the explicit step of replacing an image in memory with its complement; instead you should collapse these steps into equivalent logic operations – it should prove illuminating.

As in Task 3, try running your program on thresholded images, with structuring sets of various sizes which roughly approximate circles. Be prepared to explain the results you observe.

4 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. There will be some opportunity to do this within your Week 7 laboratory and at the start of your Week 8 laboratory, but you must be prepared to demonstrate your work and be assessed individually starting from the second hour of your Week 8 lab session.

In order to obtain the full marks for any given task, you must have a working program to demonstrate and you must be able to explain how it works and **answer questions very quickly**!! We will not be able to wait around while you fiddle with running your program from the debugger or viewing images with the Windows previewer, so have things set up for easy validation by the course staff and **use "mi_viewer"** to show your images directly from the command-line, allowing multiple images to be viewed simultaneously and individual pixel values to be read-off by the marker with simple mouse clicks.

You may feel free to re-use code from previous laboratory sessions, so long as you understand it. You may also discuss the project with other students in the class, but **your programs should be your own original work**. As with assessment tasks in other subjects, UNSW regards **plagiarism as a serious offense**; for a first offense, the penalty would be loss of all marks for the project.

You will also be asked to submit your code via an Assignment item on the course's Moodle page, **within 2 hours of the end of your Week 8 lab session**.