ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Lab 7 **Tuesday 3 November**

Edge Detection Using Arrays

In today's lab, your group must use arrays to implement Sobel edge detection. Beginning with a photo (a .PNG file, as shown to the right—the topmost image), your code will first calculate a monochromatic version of the photo (the middle image), then perform edge detection to create the bottom image. The goal of this exercise is to help you understand how arrays are laid out in memory and how to make use of arrays to solve problems. Use of multi-dimensional arrays is an important skill for MP6.

Begin by checking out the **`lab7`** subdirectory in your Subversion repository. The directory contains a total of eleven files: a copy of this document (**`lab7.pdf`**), two C source files for your code (**`lab7a.c`** and **`lab7b.c`**), a copy of the campus photo (**`campus.png`**), three other C source files and three C header files, and a Makefile to build both executables for you.

The Task, Part I

This time, start by compiling the two programs. Simply type "**`make`**" (without quotes). Then execute the first program by typing "**`./lab7a`**"—notice that the two pointers are the same. Now edit the **`lab7a.c`** file and look at the types of the **`A`** and **`B`** parameters to the function **`show_mapping`**. The **`B`** parameter points to a contiguous block of 105 (3×5×7) 32-bit 2's complement values. Treating **`B`** as an array, write a **`for`** loop to fill the array with the numbers 0 to 104.

Next, write a triply-nested loop to print the values of **`A`**. The outer loop should walk over the values of the first index (0 to 2), the middle loop should cover the second index (0 to 4), and the inner loop should walk over the third index (0 to 6). Print the contents of **`A`** as a set of three matrices, with all values of the inner loop on one line of output, and blank lines between the matrices to make them easy to separate visually. Use **`%3d`** in your **`printf`** for each element of A to make the values line up nicely in the output.

Make and execute **`lab7a`** again. Notice how the first loop wrote into the various elements of **`A`** to understand how multidimensional arrays are mapped into sequential memory addresses.

The Task, Part II

Now, you should be ready to deal with image data. The function that you must write in **`lab7b.c`** has many parameters:

```
void edgeDetect (int32_t width, int32_t height,
                 const int32_t* sobelX, const int32_t* sobelY,
                 const uint8_t* inRed, const uint8_t* inGreen,
                 const uint8_t* inBlue, uint8_t* outMono,
                 uint8_t* temp);
```

The first two parameters give the width and height of the photo in pixels. Ignore the **`sobelX`**, **`sobelY`**, and **`temp`** parameters for this task. The **`inRed`**, **`inGreen`**, and **`inBlue`** parameters provide arrays of image data: each pixel is represented by three bytes of data, a value of 0 to 255 for each of red, blue, and green. Each of the three color parameters points to an array of **`height × width`** pixels (the same as **`uint8_t inRed[height][width]`**). Use what you learned in the first task to access these data. Finally, the **`outMono`** parameter is to be used for your results. It also points to an array of **`height × width`** pixels.

Loop over all pixels in the photo and use the following equation to calculate an output value, which should be stored in the corresponding pixel of **`outMono`**:

$$monochrome = 0.2125 \times red + 0.7154 \times green + 0.0721 \times blue$$

Now make the executables and run **`lab7b`** by typing:

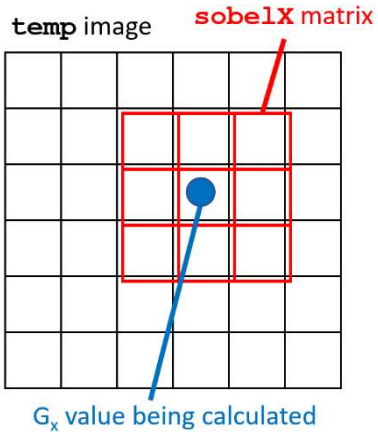
```
./lab7b campus.png mono.png
```

Look at the **`mono.png`** image—it should match the monochrome version of the campus photo on the first page of this lab.

The Task, Part III

Change the assignment of the last task to write into the **`temp`** parameter (another array of **`height × width`** pixels) rather than the **`outMono`** parameter. In this third part, you must implement Sobel edge detection using two 3×3 matrices (parameters **`sobelX`** and **`sobelY`**) and the monochrome image that you calculated already (now stored in **`temp`**).

Look at the illustration to the right, which shows a copy of the 3×3 matrix **`sobelX`** lined up over a pixel in the temp image. For each element in the matrix, you must multiply the matrix element by the corresponding pixel value, then sum up all nine products to calculate G_x for that pixel. Do the same with **`sobelY`** to find G_y. Then calculate the square root of $G_x^2 + G_y^2$ (using the math function **`sqrt`**). Write the result into **`outMono`** for each pixel, but be careful to limit the values to 255 (if a result is larger, use 255 instead).



Compile and execute **`lab7b`** again to produce the final image.

Feel free to try other photos for fun!