Lab 7 - Images

Due: end of class

OVERVIEW

For this lab, you will do some basic manipulations of gray-scale images. The obvious data structure for an image is a 2-D array, so you will be writing functions that manipulate 2-D arrays.

TASKS

For all tasks:

There are many image formats (jpg, png, etc.), and we will use PGM (Portable GrayMap) since it is a simple grayscale format.

You may need to use the following linux programs while developing your programs.

- Viewing a PGM image: eog and gimp are standard Linux program for viewing images in various formats.
- Converting between image formats: The Linux utility convert will convert between all standard formats, where its 2 arguments are the input and output files, with format determined by filename suffix. You won't need this unless you want to play with other image formats.

We need a way to convert gray-scale images to arrays, and output arrays into image files. Your instructor will supply you two functions as follows:

- writeImage: writes the argument image (a 2-D array) to outImage.pgm in PGM grayscale format.
- readImage: reads a PGM grayscale format from inImage.pgm into an array, and updates the height/width arguments appropriately (since they are passed by reference).

You may use these functions as is.

Task 0:

First, we need a gray-scale image to work on. Save the one supplied by your instructor to $PWD. This file format has 8-bit images - i.e., each pixel is a gray scale value between 0 and 255 inclusive.

Since all our tasks will involve reading an image file, processing the image, and writing the processed image back to a different file, we will first get the basics setup. Write a small program that uses the given readimage/writeimage functions to input the PPM file into an array, copies the image to a 2nd array, and writes the 2nd array back. View the resulting image to make sure its the same as the original image.

Task 1:

One way to highlight objects in an image is to make all pixels below a threshold (t1) 0, and all pixels above a threshold (t2) 255. Write a function to highlight the image, using the following prototype:

```
    void highlight(int image[][MAXHEIGHT],int width, int height, int t1, int t2)
```

Write a main program that inputs t1 and t2 from the user, highlights the image, and then writes the image.

This highlighting is called segmentation, though most segmentation algorithms are much more complicated.

Task 2:

One way to scale an image is to replace every nXn *non-overlapping* window with one pixel whose value is the average of the pixels in the window. For example, if n=2, the following image:

```
10 20
11 21
```

may be transformed to a single pixel with value 16, since 16 is the average of 10, 11, 20, and 21 (after rounding). If this is done on every window in the image, you now have an image that is 1/4 as large and reasonably approximates the original image.

Repeat the earlier tasks, except with a function named scale. Your function should work for arbitrary values of n, though you are free to simplify your work by defining arbitrary behaviors for the problematic values of n (I suggest you define such behaviors to be whatever falls out of your code). But make sure to state in comments how you handle any problematic values for n.

Task 3:

A *sliding* window operator replaces each pixel with some function of its 8 neighbors (and itself). Consider the following 3X3 window:

```
a b c
d e f
g h i
```

Then, pixel e would be replaced by some function f(a,b,c,d,e,f,g,h,i). One way to detect horizontal edges is to use the function (g+2h+i)-(a+2b+c). Note that this is a *sliding window* operator unlike the non-overlapping windows in the previous task - that is, the window is always a window around the pixel whose value is being computed.

Write a horizontal edge detection function, named hedgeDet. For simplicity, you can ignore the left/right columns and top/bottom rows of the image.

HAND IN

Your 136 instructor will tell you what to hand in and how.

GENERAL COMMENTS FOR ALL PROGRAMS THIS SEMESTER

You should have the following header on all programs:

```
/*
  Author: <name>
  Course: {135,136}
  Instructor: <name>
  Assignment: <title, e.g., "Lab 1">

  This program does ...
*/
```

GRADING

All 135 and 136 programs this semester will be graded on:

- Correctness: Does your program work?
- Testing: Have you generated sufficient and good test data to give reasonable confidence that your program works?
- Structure: Have you structured your code to follow proper software engineering guidelines? This includes readability and maintainability.
- Documentation: How well documented is your code? Good documentation does not repeat the code in English, but explains the point of each code block, highlighting any design decisions and/or tricky implementation details.