CSE 232 Spring 2017

Programming Project #6

Assignment Overview

This project focuses on digital image hole counting. It is worth 40 points (4% of your overall grade). It is due Monday 3/13 before midnight (because of spring break).

The Problem

When working with digital images, there are many simple, but powerful algorithms we can use to extract properties of those images. One such operation is hole counting.

Some Background on Hole Counting

(from Shapiro and Stockman's 2001 *Computer Vision*)

> In the late 1970s an engineer in Milwaukee implemented a machine vision system that successfully counted the number of bolt holes in crossbars made for truck companies. The truck companies demanded that every crossbar be inspected before being shipped to them, because a missing bolt hole on a partly assembled truck was a very costly defect. Either the assembly line would have to be stopped while the needed hole was drilled, or worse, a worker might ignore placing a required bolt in order to keep the production line running. To create a digital image of the truck crossbar, lights were placed beneath the existing transfer line and a digital camera above it. When a crossbar came into the field of view, an image was taken. Dark pixels inside the shadow of the crossbar were represented as 1s indicating steel, and pixels in the bright holes were represented as 0s, indicating that the hole was drilled. The number of holes can be computed as the number of external corners minus the number of internal corners all divided by four.
>
> An *external corner* is just a 2 x 2 set of neighboring pixels containing exactly 3 ones, while an *internal corner* is a 2 x 2 set of neighboring pixels containing exactly 3 zeros.

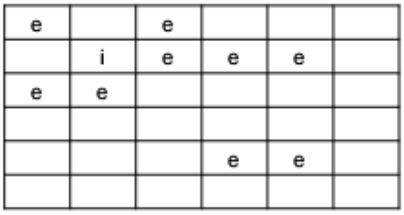
Hole Counting Practice

To visualize a binary (black and white) image, we will assume that 0s are black (the "holes") and 1s are white (the "steel"). Here is a simple 6 x 6 set of pixels forming an image.

1	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	1	1	0	1
1	1	1	1	0	1
1	1	1	1	1	1

Fill in the blank cells with "e" or "i" if they are external or internal corners, and calculate the number of holes using the formula. The solution is on the following page.

Here are the correctly-filled-in cells:



Calculate the number of holes using the formula described in the background:

```
#holes = (#e - #i) / 4
#holes = (9 - 1) / 4
#holes = 2
```

A Note

The hole counting algorithm will not work for all arrangements of binary pixels. Can you think of examples in which the hole counting algorithm would fail?

Your program will only be tested on cases where this simple algorithm produces the correct hole count.

Your Tasks

Complete the Project 6 by writing code for the following functions. Details of type for the functions can be found in `functions.h` (provided for you, see details below):

- `readImage` – Reads in lines of 0s and 1s and returns a binary multidimensional vector.
  - Inputs: Two integers – the number of columns and the number of rows, respectively.
  - Output: The image stored in a two-dimensional vector of ints.
- `printImage` – Outputs the binary image of the multidimensional vector returned by `readImage`. For formatting, see the output files.
  - Input: Two dimensional vector of ints.
  - Output: Nothing (void). Should print to terminal.
- `countHoles` – Returns the number of holes in the multidimensional vector returned by `readImage`.
  - Inputs: The two dimensional vector of ints (the image).
  - Outputs: The number of holes as an int.

Test Cases

- 6 test cases

Assignment Notes

1. You are given the following files:
   a. `main.cpp` – This file includes the main function where the test cases will be run. **Do not modify** this file.
   b. `functions.h` – This file is the header file for your functions.cpp file. **Do not modify** this file.
   c. input#.txt – These four text files will be used to run the test cases.

   d. correct_output_#.txt – These text files will be used to grade your output based on the corresponding input files. Be sure that your output matches these text files exactly to get credit for the test cases.
2. You will write only `functions.cpp` and compile that file using `functions.h` and `main.cpp` in the same directory (as you have done in the lab). You are only turning in `functions.cpp` for the project.
3. Comparing outputs: You can use redirected output to compare the output of your program to the correct output. Use redirected output and the "diff" command in the Unix terminal to accomplish this:
   a. Run your executable on the desired input file. Let's assume you're testing input1.txt, the first test case. Redirect the output using the following line when in your proj06 directory: `./a.out < input1.txt > output1.txt`
   b. Now your output is in the file output1.txt. Compare output1.txt to correct_output1.txt using the following line: `diff output1.txt correct_output1.txt`
   c. If the two files match, nothing will be output to the terminal. Otherwise, "diff "will print the two outputs.

Deliverables

`functions.cpp` -- your completion of the functions based on `main.cpp` and `functions.h` (both provided).

1. Remember to include your section, the date, project number and comments.
2. Please be sure to use the specified file name, i.e. "functions.cpp"
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient

Getting Started

1. Download proj06.cpp from the course website.
2. If you are in a CSE lab or on x2go, select the H: drive as the location to store your file.
3. Save the name of the project: proj06.cpp
4. Prompt for some of the values and print them back out, just to check yourself.
5. When you do a calculation, use the provided table as a way to check yourself.
6. Use the Handin web site to hand in the program (incomplete as this point, but you should continually hand things in).
7. Now you can calculate the individual elements of the project. After writing code for each of the below aspects, compile and print out values to make sure you are getting the correct values. Do it ***INCREMENTALLY***, that is write and check various calculations
8. Now you enter a cycle of edit-run to incrementally develop your program.
9. Use Handin to hand in your final version.
10. ***Remember***: the project must compile on the lab machines or the x2go environment for project credit. Check it before you hand it in!