University of Nevada, Reno

CS 302 — Data Structures

Assignment #4

Students:
Joshua Gleason
Josiah Humphrey

Instructor:
Dr. George Bebis

April 18, 2010

Contents

List of Figures

1 Introduction

In this programming assignment, we continued and built upon image processing. We took out previous programming assignment and added more functionality. Our goals were to represent the regions found by connected components in a list and to compute a number of useful properties to characterize their position, orientation, shape, and intensity. This was a challenge that tested our ability to understand the data structures and image processing. We were also asked to use our skills to manipulate items in lists, use templates, and learn about feature extraction and classification.

In this assignment, we learned about geometric properties of objects and how to objectively classify them using mathematical formulas. We were able to implement the equations that found various geometric properties of objects found in images. Since this is a introduction to image processing, we were only asked to implement geometric and intensity properties only. Using more advanced techniques we could have done more complicated things.

To calculate these properties, we used a technique from probability theory called moments. A moment is defined as

$$M_{p,q} = \sum_{i,j \in R} i^p j^q$$

This was the basis for many of our calculations and was implemented as a generic function that would take any numbers for p and q.

2 Use of Code

3 Functions

3.1 Image.h

DILATE

```
void ImageType<pType>::dilate()
```

Purpose

This function changes any pixel to black that is touching a black pixel on any of its 8 sides.

Input

None

Output

None

Assumptions

Nothing is assumed but it makes sense to actually have a defined picture.

4 Bugs and Errors

During the creating of this program, there was one single bug that took a very, very long time to track down, following is a detailed explanation of the bug and the methods used to track down and repair it.

The problem originally manifested itself as a segmentation fault when the choice to 'Classify Regions' was selected in the main menu. At first I looked through the classifyRegions function for any obvious problems, after that search came up empty I began using the GDB debugger to track down the fatal error.

The first thing I needed to know was where the actual error was occurring, so I executed the program in GDB. After the re-creating the segmentation fault I found that the crash was occurring a conditional statement inside of the == operator overload function inside of the Region Type class. By examining parameters passed to the function I discovered that the right hand side was actually an invalid value, printing the address of the parameter I found the value was actually NULL. This seemed very strange, so I used GDB's backtrace command, which indicated that the comparison was taking place in the deleteItem function of the sortedList class or more specific the list of regions for the image.

Before debugging further, I pondered the recently acquired information and came to a hypothesis. I believed that the deleteItem function was not finding the value that it was passed even thought the RegionType values were being directly taken from the list of regions. This was the only way I could conceive the == operator being passed NULL from deleteItem. Some more debugging was definitely needed to verify this claim and also answer some other questions if this was the case.

After setting a breakpoint in the deleteItem function I ran the program and selected the Classify Regions option. The program paused at the first breakpoint where I obtained some very interesting information about the RegionType in question. I ran the command `print *this` in GDB so that I could quickly see all of the private members of the current object. To my surprise a few of the values were definitely invalid values, which may explain why the == operator never returned true, even if the values had the same data members. What would happen if you tried to compare two invalid double values, even if they were copies of one another? I had to determine the answer to this question. By continuing execution I found where the two had all the same valid data members and when finished the == function I discovered that the returned value was false, which would explain why deleteItem never found the right value.

At this point I was feeling pretty good about having narrowed down the problem to a calculation issue, but why was I getting invalid values for eccentricity and theta for some of the regions? To determine this I set a breakpoint in the setData function of RegionType and recreated the error yet again. To my surprise the first region had some invalid values, but I also noticed that the value for lambdaMin was zero; I thought I recalled the eccentricity requiring dividing by lambdaMin, so I checked it out. I verified that this was the indeed true, so I decided to find why lambdaMin was zero. After using similar techniques I discovered that lambdaMin was zero because the central moment was returning zero with non-zero parameters because the centroids were equal to the location of the pixel. This calculation was correct and I received the same value when I did the calculations by hand, so the problem wasn't actually a problem with the code, it was a problem with the function (I found that this is true for any shape that is 1 pixel wide or long). As for the value of theta, I discovered that if the shape is a single pixel or has an orientation of exactly parallel to the x-axis then the central moment with parameters 1, 1 was returning zero. I added exceptions for both of these (making the orientation one-hundred eighty degrees if in this case, although zero would have been equally as correct). This single bug took nearly two hours to track down, but after discovering the cause I feel much more experienced with GDB.

5 What was Learned

6 Division of Labor

7 Extra Credit