Sunday, October 3, 2021

HW2 – Connected Components Labeling

This assignment was used to teach connected components labeling using a two-pass, eight-connected, union-find algorithm in C++. The program is able to label pixels belonging to the same component, calculate statistics on the individual components, and randomly colorize each component.

Ryan Orf

CS4650: DIGITAL IMAGE PROCESSING

For this assignment, students were challenged to label pixels of a binarized image based off of their connected components. My program accepts an optional invert flag if the user would like to invert the binary image, and the program will binarize an image if the provided image is not already binarized. To accomplish the goal of connecting components, a two-pass union-find algorithm was performed on an image to obtain its eight-connected components. The first pass will call a function to assign intermediate labels to all foreground, or white, pixels, and the function will assign new intermediate labels to a parent array or call union and find functions to conceptually join two components that should be connected, based off of specific cases. The second pass will find the true number of connected components by calling a function that reassigns each foreground pixel a label set equal to the determined root label in the parent array. After finding the connected components, the program calculates each component's area, centroid, and covariance matrix by calling appropriate functions, another function is called which colorizes each component with a random color, and the results are displayed for the user.

Although this experiment did not produce any physical, statistical result, such as a graph or table, the process behind the inner workings of the programs and its functions can be discussed in detail. The first and second pass of the connected components algorithm has essentially been described already. For the first pass, the pixels of a binary image are looped through, and if they are a foreground/white image, their labels are set accordingly in a matrix based off of sixteen cases discussed in class, as long with a few other modified cases to ensure the user does not attempt to access pixel row and column values outside the bounds of the image. If a foreground pixel is connected to two foreground pixels with different intermediate labels, a union function is called which utilizes a find function, exactly as demonstrated in class. For the second pass, the parent's array is looped through and labels belonging to the same connected

component are all set equal to the label value of the root. This will make computation of components easier. Then, the pixels are looped through and the labels of the foreground pixels in the label's matrix are set using the find function. Finally, the parent array is looped through, and the number of components is calculated based on how many root labels, or labels with a value of zero, are found in the parent array.

The program proceeds by declaring an array and calling a function to store the label corresponding to each component. Next, arrays to hold the values of each component's area, centroid, and covariance matrix are declared and initialized. Although statistics for only the largest ten of these components will be displayed to the user, I decided I should calculate the statistics for all of the components regardless. Functions are then called to fill the area, centroid, and covariance arrays with values corresponding to each component. These functions use the equations provided in class, implemented with loops and variables. The main program then determines whether there are more than ten components, but regardless loops through the arrays using a bubble sort algorithm to sort the components in decreasing area size.

Before outputting the image, the statistics for the number of components, number of intermediate labels, area of each object, centroid of each object, and covariance matrix of each object are printed to the user. One final helper function is called to colorize the image. This function randomly sets values of arrays representing blue, green, and red pixel values using built-in rand functions. The image is looped through one last time, and if the pixel is a foreground/white pixel, the pixel is assigned a random color value by indexing the previously discussed blue, green, and red arrays on the remainder of the label value at that pixel when divided by twenty. In order to work with a binary image in color, a second instance of the input image was actually initialized to a colored image in the main program, which would eventually

become our output image. The output image can now be written to disk and displayed, showing the connected components in color.

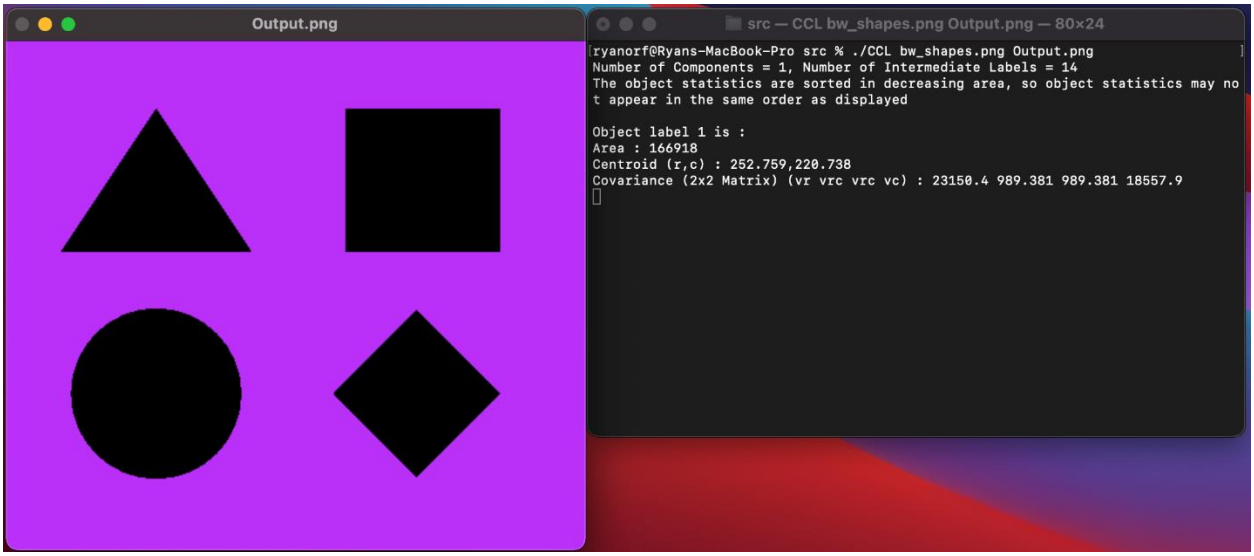
This assignment proved to be a trying task. The implementation alone took nearly 440 lines of code. More difficult was the conversion from logic to code implementation. In order to complete the program in the way I envisioned it, I had to expand my knowledge of C++. Headers for iostream and vector, along with a type definition I implemented so I could use the word 'Matrix' in referencing, were added. This is the only use of these headers, and they do not assist in labeling components or calculating statistics for this algorithm, therefore there should be no issue. C++ does not allow for passing matrices into functions in the same way as C, so this is the only way I could figure out how. Additionally, the size of the parent array was set using a global constant, in the theoretical case of guidelines changing for this assignment. Other than these special remarks, I believe my program is straightforward. I attempted to avoid reusing code as much as I could.

Another issue I could not resolve involves differences in some of the provided examples' output from my program's own. The major area of concern in this sense involves the count of the intermediate labels used. I spent several hours attempting to fix this, and I simply could not find the flaws in my logic. Oddly enough, the correct number of components were still obtained in every image, so I figured this could not be that much of a setback. The statistics are also slightly different, although the differences in area, centroid, and covariance matrix values are slight, especially in comparison to the count of intermediate labels. After examining my work again, I figured the differences may be caused by differences in the size of the image on different machines. I tested this by running the area function, singling out a single connected component while looping, and recoloring every pixel found in the area function to ensure that the function

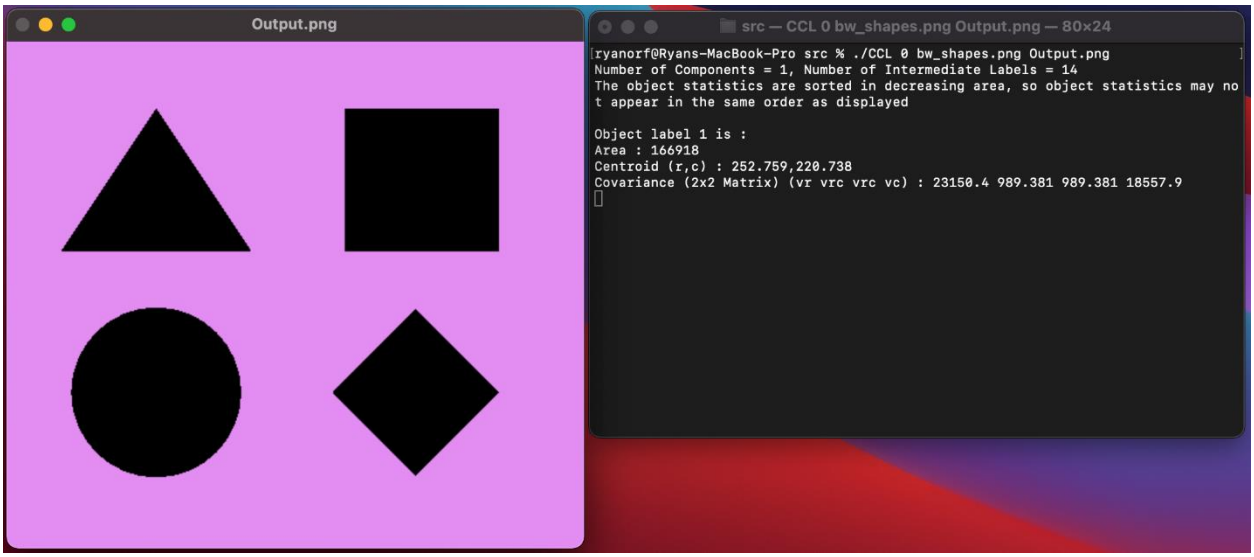
was only calculating the summation of the desired foreground pixels belonging to that connected component. Sure enough, the area/number of pixels varied slightly from the example area, therefore I concluded that there is nothing wrong with my program in this regard.

After spending more time than I would have liked to debugging and analyzing the implementation of the requirements for this program, I believe I entirely understand how the two-pass union-find algorithm for eight-connected components labeling works, as well as how to calculate the area, centroid, and covariance matrix for a connected component. As a final display of my work, please find below sample images of the program running.

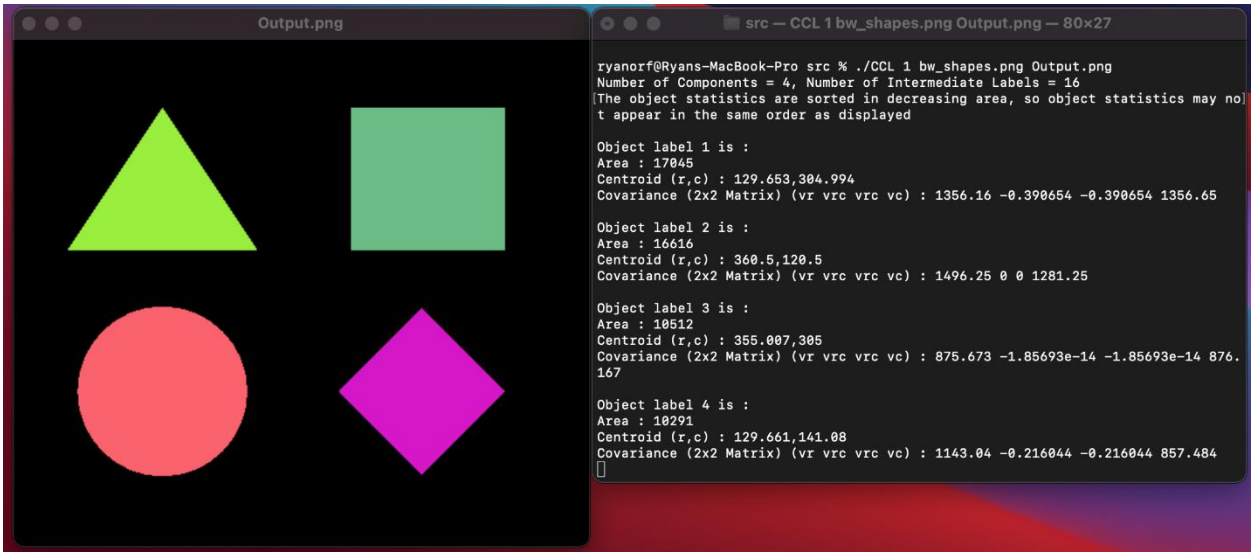
Proof that the program runs, and functions, as intended (without the optional invert flag):



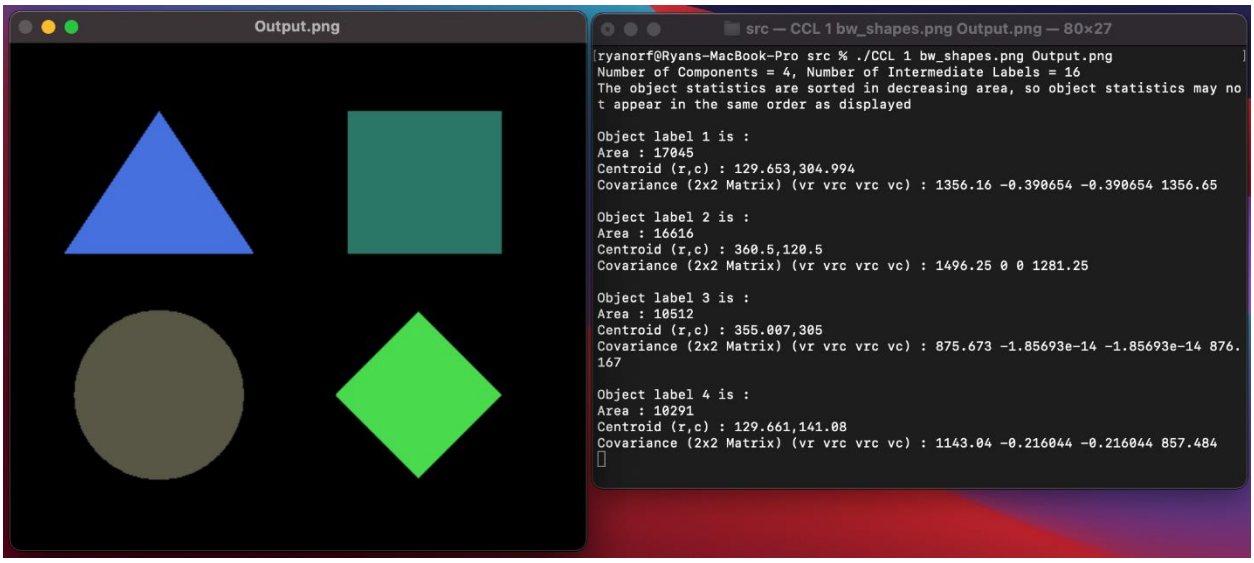
Program output when 0 is chosen as the optional invert flag:



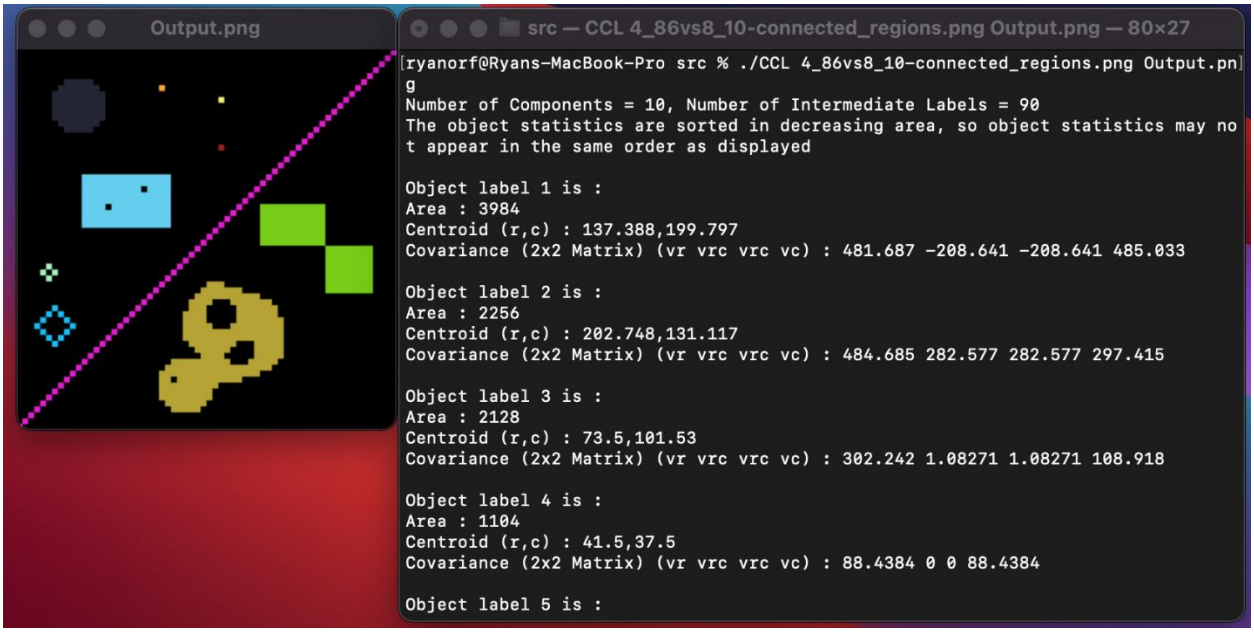
Program output when 1 is chosen as the optional invert flag:



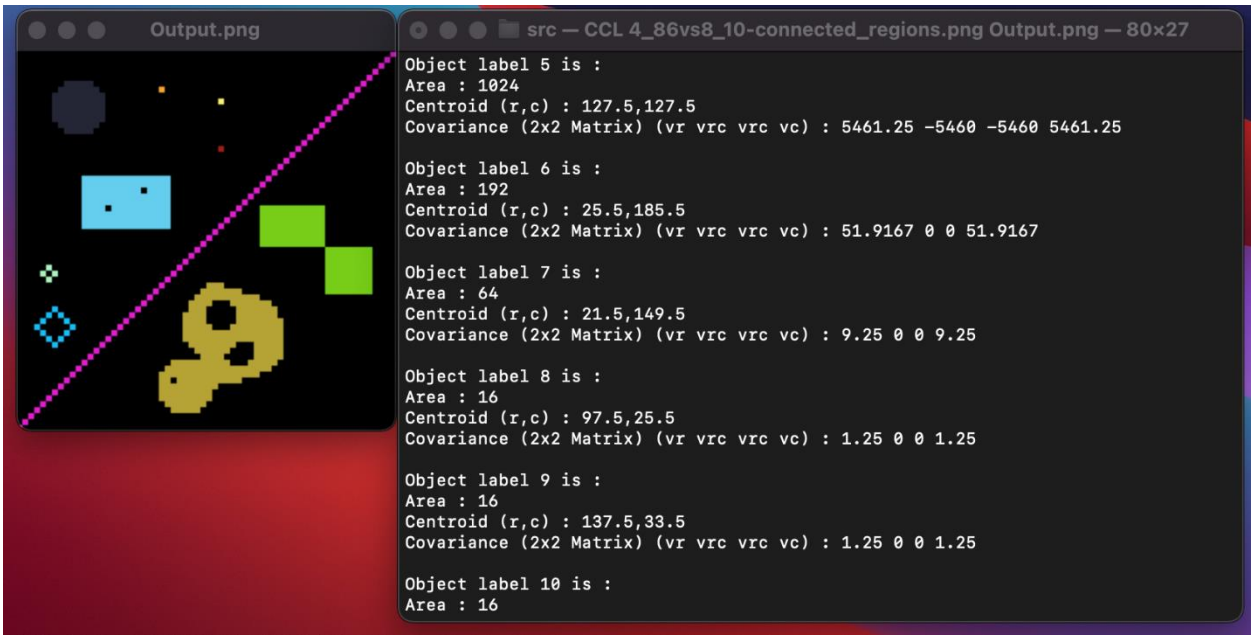
Program running again exactly as before to demonstrate the random color assignment:



Programming running with a new image:



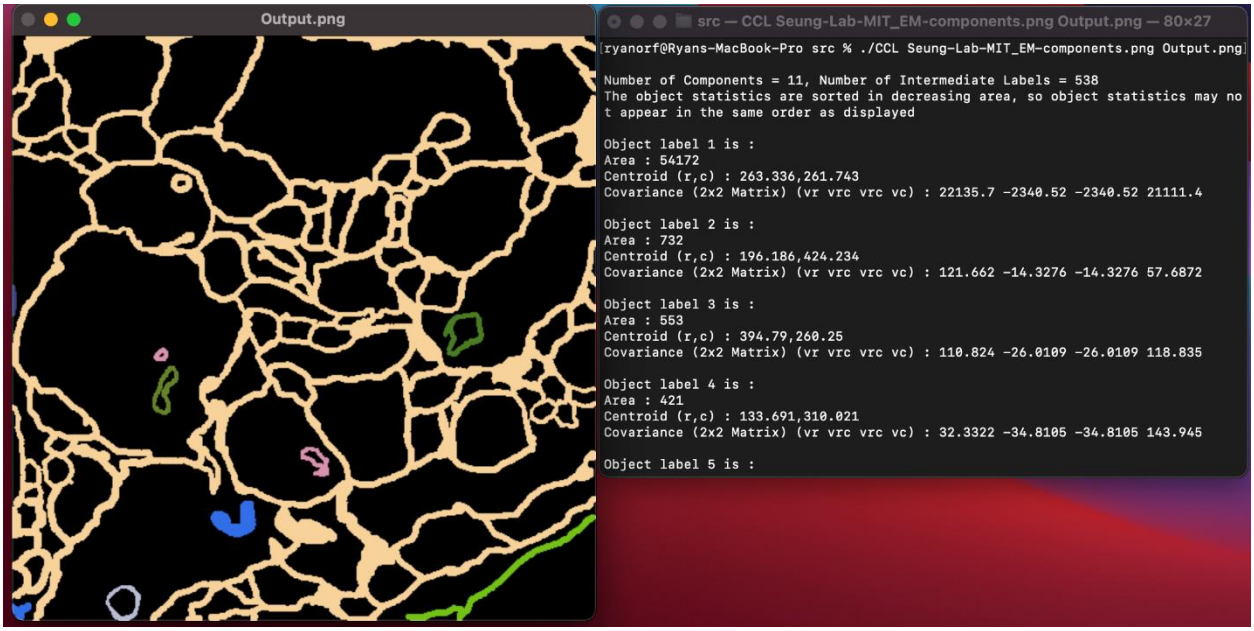
Statistics for objects 5-9 of the same image:



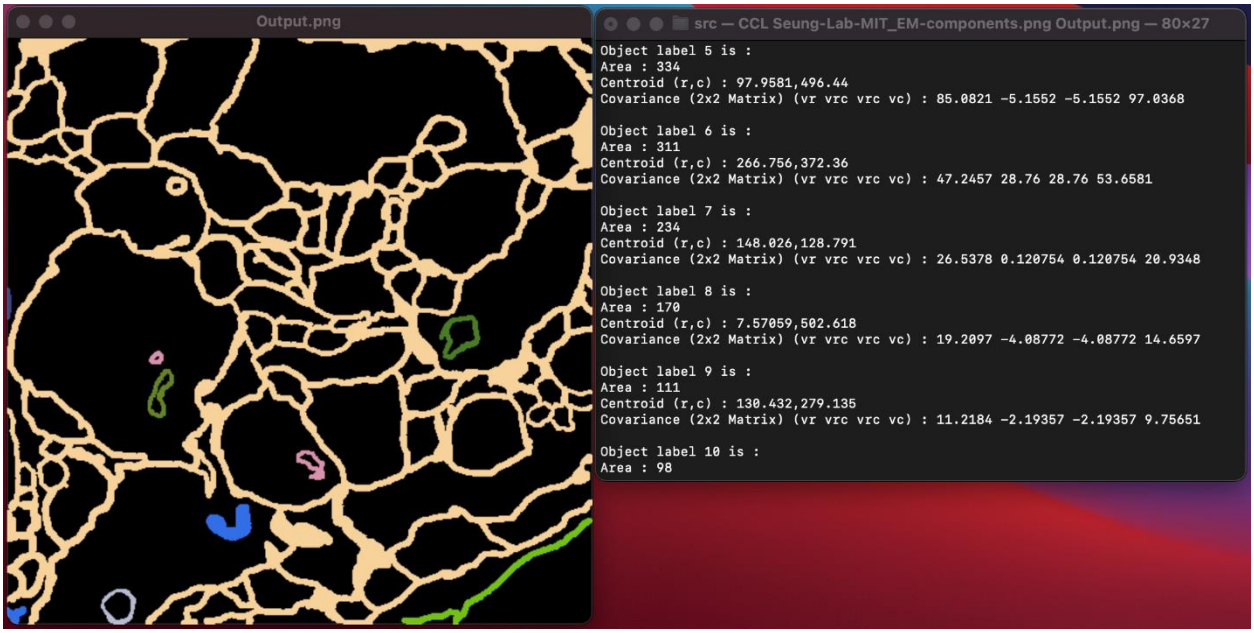
Statistics for object 10 of the same image:

```
Object label 10 is :
Area : 16
Centroid (r,c) : 137.5,65.5
Covariance (2x2 Matrix) (vr vrc vrc vc) : 1.25 0 0 1.25
```

Program running with another new image:



Statistics for objects 5-9 of this image:



Statistics for object 10 of this image (to show that only statistics for the 10 largest components are displayed since this image has more than 10):

```
Object label 10 is :
Area : 98
Centroid (r,c) : 1.30612,231.5
Covariance (2x2 Matrix) (vr vrc vrc vc) : 1.11037 2.17513e-16 2.17513e-16 55.678
6
```