Project 4: Connectivity Analysis

Introduction:

The connectivity analysis program makes use of a Disjoint set class to keep track of connected components or 'blobs' in an image. Each of the connected components in an image is colored differently by random palette generation to produce images that have a wide variety of applications. Since images often contain tiny blobs, these just serve as noise and its useful to clean them out through size filtering.

Program structure:

The program is written in an object-oriented fashion. The class Canvas has the original image, gray scale image, binary image, labelled matrix, pseudo-colored image and the filtered image. The member functions of this class serve to create these different images by using variables from the canvas object.

The program starts out with reading in an image using readPNG. this image (imageRGB) is then stored into Canvas. The gray scale image is also calculated right away since this would not be modified by anything else in the program. InitEvent then calls display which calls canvas.display(). The images are now displayed onto the screen after calling the appropriate functions to create them. (Note that grayscale() is not being called again, just displayed since the grayscale image does not change. All other images are subject to change due to the change in the threshold value which starts out as 128. To increment the threshold value press '>' and to decrement it press '<'. Every time these keys are pressed, canvas.display() is called and the 3 images (binary, pseudo, filtered) must be recalculated and the threshold value in the window is updated. The threshold value and filter size are displayed at the top. The window title contains the file name. The ESC key closes the window.

Description of Canvas member functions:

void grayscale() : Calculates the gray scale image from the original image by using the simple formula : 0.3 * r + 0.6 * g + 0.1 * b + 0.5 . (thus from three pixels (rgb) you get one pixel in grayscale)

void binaryThreshold() : A pass is made through the grayscale image and all the pixel values are compared to a threshold . If the pixel value is below the threshold, its value in the binary image is assigned to 0. If the pixel value is above or equal to the threshold, its value in the binary image is assigned to 255.

void assignLabels() :

uses the L shaped model to put members in the same connected component into the same set using the binary image. A disjoint set is allocated on the heap with a size of height*width. Two passes are made through the image. The first pass looks at two adjacent cells in the same row (horizontally adjacent in this case the left cell). If they are both non-zero, i.e significant in the binary image, a union is performed on them since they belong to the same connected component. The second pass looks at two adjacent cells in the same column (vertically adjacent in this case the top cell). If they are both non-zero, i.e

significant in the binary image, a union is performed on them since they belong to the same connected component

void makePseudoColorImage() :

Creates a label matrix based on the connected components. An array is allocated on the heap. A pass is made through the label matrix. If the cell does not hold 0, (significant in the binary image), a find is performed on that node and the value obtained is stored in root. If array[root]==0 , it means it's the first time this connected component is being encountered and it is assigned a new label in the array. If it is not zero, because part of the component has already been encountered, assign it to array[root] which contains the label for this component. Thus all connected components get assigned a unique integer value.

An image histogram (an unordered map is used as the underlying data structure) is also built in this function to avoid another pass through the image. A find is performed on every cell of the label array (every cell contains the unique integer for that connected component). If it isn't already present in the histogram (new component encounter), its value in the histogram is assigned to 1. Subsequent encounters are just added on to the existing value and this gives us the size of each connected component.

void pseudocolor() :

This function uses the random palette generated in the constructor of canvas. (This was done so that the size filtered and connected components images both had the same colors for corresponding components). A bool variable of canvas, filter is set to false if we want the unfiltered connected components image and set to true if we want the filtered image. If its set to true, the histogram is queried and all values below the filter size value are set to 0 or cleared.

Conclusion:

It was a fun project to work on. A possible addition would be to modify the size filter value as with a key press. (not done here because it wasn't a requirement).