OBJECT-ORIENTED PROGRAMMING

Introduction

For this assignment, I was tasked with searching a large image for the closest match to a given sub image. This problem simulates the traditional game *'Where's Wally?'*.

I was presented with two txt files, and tasked with outputting one. A description of these can be found below:

Filename	Description	Type
Cluttered_scene.txt	A text file that contains a two-dimensional array of doubles that represents the full image that is to be searched. ***Width:*** 1024 ***Height:*** 768	Input File
Wally_grey.txt	A text file that contains a two-dimensional array of doubles, like ***Cluttered_scene.txt***, that represents the image to find a match for. ***Width:*** 36 ***Height:*** 49	Input File
output.pgm	A pgm file that is generated from a subset of the ***Cluttered_scene.txt*** that represents the closest match to ***Wally_grey.txt*** ***Width:*** 36 ***Height:*** 49	Output File

The two text files must both be read in, and then interpreted into class data. Once in this format, the full image must be searched to find a close match to the search image. Once the image has been fully searched, the best match should then be outputted into a pgm format.

I plan to complete the search using a Linear Nearest-Neighbour Search Algorithm. This algorithm computes the similarity between two data sets and therefore allows me to rank each match relative to each other.

Program Structure

Filename	Description
main.cpp	This file contains functions for: • The main program logic. • Reading image data from txt files. • Outputting image data to pgm files. • Trimming file names from full file paths. • Removing file paths from full paths. • Bubble sorting the ***MatchData*** values. • Searching a ***LargeImage*** for the closest match to an Image. • Calculating Sum of Squared Difference.
Image.h	This class contains: • ***structure*** – an array of doubles that contains the pixel values. • ***width*** – int value for the pixel width. • ***height*** – int value for the pixel height. • ***count*** – int value for the total number of pixels. • ***length*** – int value for the last index of the structure.
Image.cpp	• ***getMatrix()*** – returns the entire matrix. • ***setMatrix()*** – sets the entire matrix. • ***getValue()*** – returns the value of the requested index. • ***setValue()*** – sets the value at the requested index. This is a virtual function that can be redefined within the derived classes. • ***fill()*** – fills the entire matrix with a value. • ***print()*** – prints out the matrix to the console. • Operator overloads for addition, subtraction and multiplication.
LargeImage.h	This class inherits from ***Image*** but contains no additional variables.
LargeImage.cpp	• ***CreatePartial()*** – returns a ***MatchImage*** that represents one possible match to the search image.
MatchImage.h	This class inherits from ***Image*** and contains two additional variables: • ***offsetX*** – int value of the x coordinate relative to the full image. • ***offsetY*** – int value of the y coordinate relative to the full image.
MatchImage.cpp	• ***setValue()*** – sets the value at the requested index, relative to a given offset. This allows the values to be set using the index from the ***LargeImage.***
MatchData.h	This class contains: • ***x*** – int value of the starting x coordinate. • ***y*** – int value of the starting y coordinate. • ***comparison*** – double value of the similarity score.
MatchData.cpp	This class contains no methods.
Globals.h	Contains the definition of program wide global variables: • ***LARGE_WIDTH*** – Width of ***Cluttered_scene.txt*** • ***LARGE_HEIGHT*** – Height of ***Cluttered_scene.txt*** • ***LARGE LENGTH*** – Product of above Width and Height. • ***SEARCH_WIDTH*** – Width of ***Wally_grey.txt*** • ***SEARCH_HEIGHT*** – Height of ***Wally_grey.txt*** • ***SEARCH_LENGTH*** – Product of above Width and Height.

Searching Algorithm

The pseudo code below illustrates the intended method for finding a match. ***Program()*** first loads in all the required image data. After this, it loops through all the pixels in the full image and creates a match image for the current pixel set. This match is passed over to the ***CompareImages()*** function.

The ***CompareImages()*** takes this match as well as the search image and completes a Sum of Difference Squared calculation on each pixel within the two images to determine the similarity between the two. The similarity score for that match is then returned and if the current match is better than the previous best, it is overwritten.

```
FUNCTION Program()
    // Load our images
    DEFINE full = LoadImage("Cluttered_scene.txt") // full image
    DEFINE search = LoadImage("Wally_grey.txt") // image to compare
    DEFINE bestMatch // the current best matching set

    // Loop through every coordinate set within the image
    FOR x is (full.width - search.width)
        FOR y is (full.height - search.height)
            DEFINE currentMatch(x, y) // get portion of full image for comparison
            currentMatch.comparision = CompareImages(current, search) // Do Comparison

            IF currentMatch.comparison < bestMatch.comparison
                bestMatch = currentMatch
            ENDIF
        ENDFOR
    ENDFOR
ENDFUNCTION

FUNCTION CompareImages(current, search)
    DEFINE sum = 0

    FOR x is (search.width)
        FOR y is (search.height)
            DEFINE currentValue = current[x, y] // Get values
            DEFINE searchValue = search[x, y] // Get values
            DEFINE difference = (currentValue - searchValue) // Calculate Difference
            DEFINE similarity = difference * difference  // Square Difference

            sum += similarity // Add to current similarity score
        ENDFOR
    ENDFOR

    RETURN sum
ENDFUNCTION
```

This algorithm was later adapted to consider the N best matches, where N is a non-negative integer value supplied by the user. This is done by storing a vector of best matches. The current match is added to the vector, the vector is then sorted, and the worst match in the vector is removed.

Results

The program will output a user specified number of matches, in decreasing similarity from the target image. The program finds an almost exact match to the image. Due to the level of accuracy that the program finds matches, it also finds lots of similar matches, that are offset in each direction by a few pixels. As you can see by the image shown, when outputting 10 matches, 9 of them are almost identical to the target image.

Discussion and Conclusion

In summary, I had to find the closest match to a given search image within a full scene. This was achieved by performing a Nearest-Neighbour Search on the image and comparing the Sum of Squared Differences to assess the similarity.

My solution finds a very accurate match to the image. It does however take many cycles to find this match, 786432 to be exact, but does result in a very thorough search. One possible optimisation would be to consider every other pixel set, instead of all of them. This would halve the time taken but should still find a match very close.

I also ran into an issue when attempting to read in the files. The problem was that depending how the program is ran, varies what path the application it believes it is in. this was solved by accessing the direct path to the executable, and then trimming the program name off the end.