CMP2090M

Object-Oriented Programming Assignment Report

Neil Miran Ranasingha

Student ID: 15589538

1. Introduction

The main objective of this assignment is designing and implementing a Nearest-Neighbour Search algorithm using good object oriented principles and practice. The problem detailed in the assignment brief, tells us that we must find the ‘Wally’ character from a cluttered scene image, using the provided template image of Wally. Both images of the cluttered scene and Wally are given to us in text format, where each pixel of the images are represented by a value between 0 – 255, each of which represents a greyscale colour. Since the images are both 2D matrices, the data for each image will be stored in a pointer array, which means the data will be stored in heap, therefore becoming a 1D array with 2D properties. Using pointer arrays is a much faster and a more efficient method of storing such large quantity of data, as it simply has a pointer to the data rather than all of the in-built functions that come along with standard arrays or vectors.

The program searches through the cluttered scene image to find Wally. To do this, it extracts blocks from the cluttered scene image of the same size as the Wally image, to compare them together. A linear search is then used to extract the blocks of data, which sifts through the whole image from the top left to the bottom right. The user can then select how many blocks of data the linear search should look through by selecting how many pixels to shift each time a block is compared. For example, the user can choose to extract the blocks of data every 6 pixels in a row and shift the row by 12 pixels down when it reaches the end, this will decrease accuracy and in turn increase efficiency and speed. Whereas, extracting the blocks of data every 1 pixel in a row and shifting the row by 1 pixel down will increase the accuracy but decrease efficiency and speed.

As the program sifts through the data, extracting blocks, it compares the extracted data with the one from the Wally image using the normalised correlation (NC) algorithm. This outputs a similarity score ranging between -1 and 1, where 1 represents the highest similarity. Each score is compared with the previous score and the top one will be the most similar to the Wally image. The user can also select how many best results they would like to be displayed, the program will output the same amount of NC scores as the number chosen by the user.

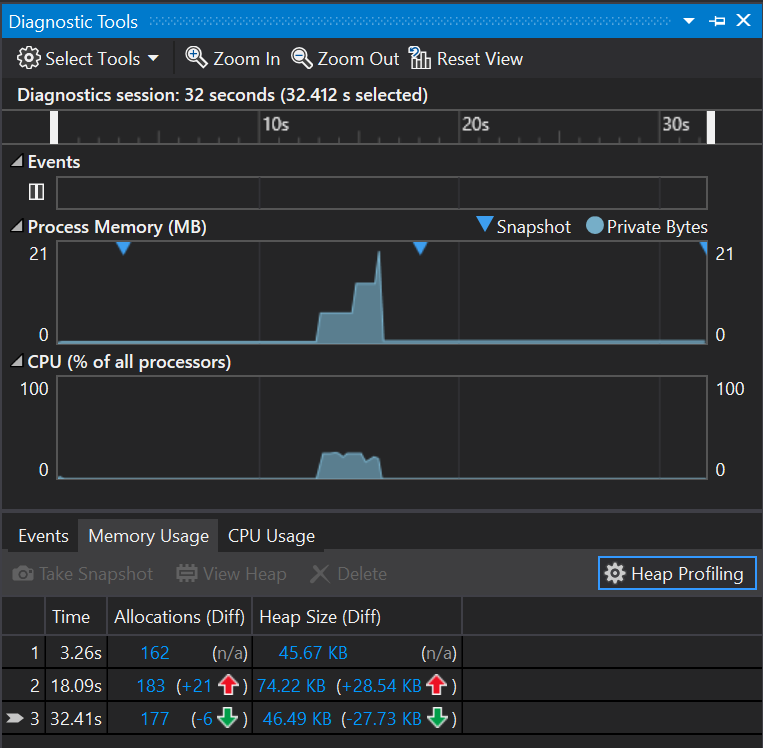
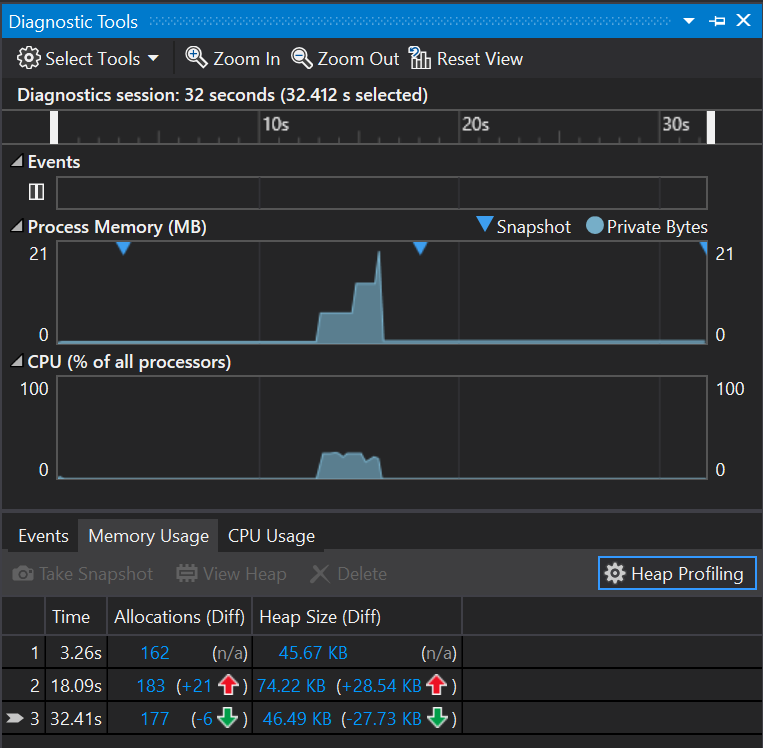
Once the program has outputted the results, it also outputs the top results by drawing a box around the block which corresponds to that specific result, for example Wally will have a box drawn around him in the cluttered scene image output. Each of the results will be outputted in the cluttered scene image with a box drawn around the block which has a score of similarity. Other information will be displayed in the console along with the NC scores and their coordinates, such as the number of blocks that have been compared to Wally and the shifts made by the linear search, which depends on the user’s choice at the start.

At the end, a timer is shown detailing how long the program has taken to import, calculate and export the data. The program also asks the user if they would like to run the program again. If they select yes, the program will loop back to the start. If not, the program will terminate.

2. Programme Structure

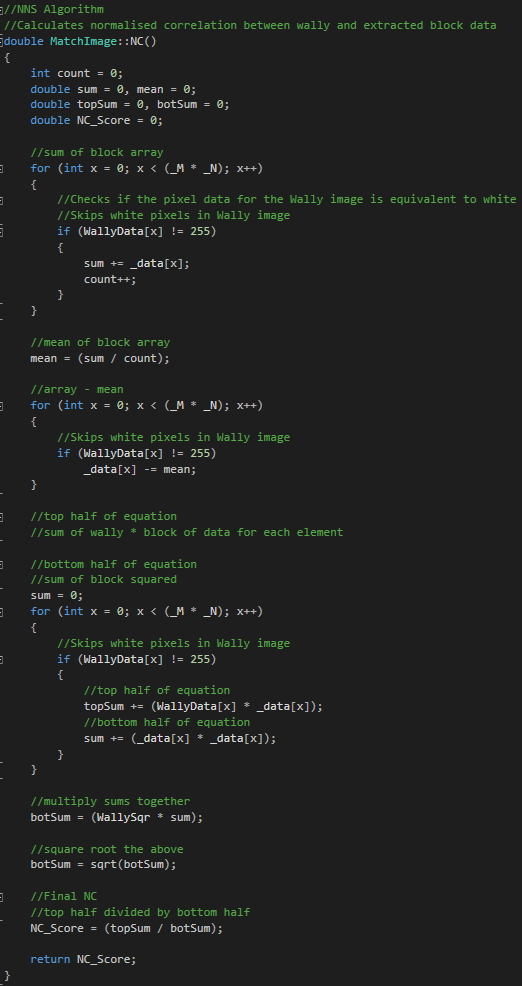
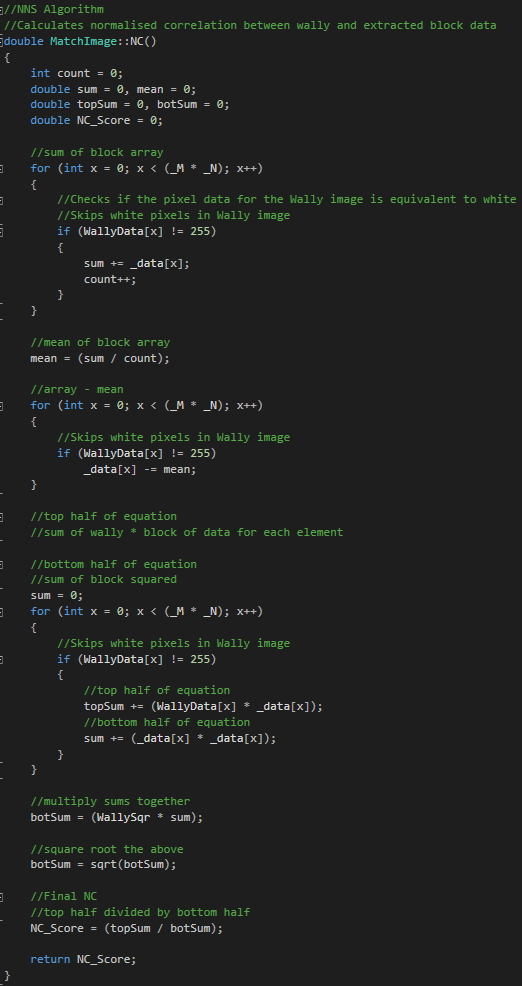
The program contains three classes, the BaseImage class, LargeImage class and MatchImage class each of these are in their own header files with corresponding source files. The BaseImage class is the parent of the other two, where the child classes share three protected variables and one virtual function from the parent class. The three protected variables, \_M, \_N and \_data allow each class to have its individual data store without having to reinitialise them in each class. Since they are protected, they can be used by the child classes whereas if they were private, the child classes would not be able to access them. The virtual class is a simple void function which deletes the \_data pointer array, clearing all its data. The virtual function means that each of the child classes are able to call this function and delete the data used by their individual class.

The operators in each class populate their own data into the \_data pointer array which is then manipulated throughout the functions in their classes. Once the program has finished using the data, the program calls the virtual void function to clear the memory of the \_data variable and then once the main() has reached return 0; the program will call the destructors of each class which also clear memory for other variables used by the class. Using the in-built diagnostic tools in Visual Studio, I ensured that all the memory being used by the program was cleared.



3. NNS Algorithm

The NNS algorithm that I have chosen is the normalised correlation (NC) algorithm. I quickly found that this method is much more accurate than the sum of squared differences algorithm, as it compares the data more thoroughly by comparing the average sums from each array. Since the NC algorithm outputs a value between -1 and 1, finding Wally is simple, as he would be in the block of data that has an NC score closest to 1. As the linear search sifts through the cluttered scene image comparing new blocks of data to the Wally data every time, the calculated NC score will then be compared with the previous score to check if it is higher or not. If the result is higher, the variable that stores the highest score will be replaced, in the end the highest score will be shown.

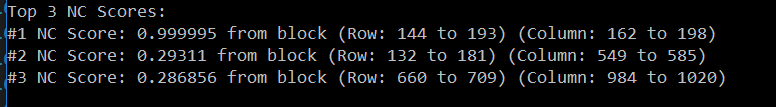


As you can see, the algorithm only calculates the data for the extracted block of data, which is stored in the MatchImage class, this is because there is a function within the class which is called at the constructor to work out the calculations required for the Wally data. Doing this, means that calculations for the Wally data do not need to be repeated within the NC algorithm itself and in turn this increases efficiency. The calculations also ignore where there are white pixels in the Wally template image provided, this is done for both calculations of the Wally and extracted block of data. Therefore, only the pixels where Wally has a grey scale colour other than white are being compared, ignoring all of the other things in the rest of the block.

The overall speed of the whole calculation depends on how many blocks are being compared to Wally, which depends on what the user chooses at the start of the program. If they chose to shift the blocks 12 by 6 pixels, there will be around 20,000 blocks to compare with. Whereas is they chose to shift the blocks 1 by 1 pixels, there will be a much larger total of around 710,000 blocks to compare with. However, as I mentioned earlier, this would affect the accuracy of the search too since there are more different combinations of pixels in a block, if there are more blocks.

4. Results

4.1 Best matching

The best matching score outputs Wally, the console displays the top N-best values where the top value should correspond to Wally. The NC score and the coordinates for where Wally is, are displayed as shown below. As you can see the best NC score is “0.999995”, this means that this block of data is the most similar to the template image provided. The best NC score is never 1 in this case, this is because the template image is not identical as the image that is being searched.

Once the best match is found, the program uses the coordinates of Wally and draws a small 3-pixel thick box around the block that contains Wally as shown on the right.

Once Wally was found, I decided to experiment with the program, testing its limits. Therefore, I created a function in the LargeImage class which changed the location of the block that contained the actual Wally character and placed it randomly around the cluttered scene, leaving a black box where I moved Wally from. The purpose of this was to see if my program would be able to find the displaced Wally. The results showed that using the less accurate linear search of shifting the blocks 12 by 6 pixels could only find Wally if the block containing Wally is within the shifted blocks. However, if the linear search shifted the blocks 1 by 1 pixels, Wally could still be found as none of the block combinations were being missed. The drawback of this was a much slower search speed.

4.2 N-best list

The number selected by the user at the start of the program is used as N to determine how many top results to store and display. However, since best NC scores must be accompanied by the coordinates of the block that they are contained in, I decided to use a 2D vector to store both the NC score and the coordinates of the block that the score belongs to. Each row within the 2D vector represents a block, the columns contain the NC score and coordinates of that block.

Since I am using a vector I am able to use the sort(); function which allows me sort the NC scores in the vector by descending order, therefore the bottom row will contain the highest NC score and its coordinates.

Just like the best score, the N-best scores are displayed the same as shown in the image above. The program also outputs the pgm files for each of the N-best blocks just as it does with the best score, drawing a box around the result block.



5. Discussion & Conclusion

Overall the program functions as intended. The normalised correlation algorithm works very well at comparing blocks of data to the Wally data. The ability for the user to change the amount the blocks should shift by when running the linear search, is a great way of showing how the program could run at different speeds, demonstrating different efficiencies and accuracy. The program is able to find the N-best scores very well too, displaying their NC scores and coordinates along with illustrating where each result is in relation to the cluttered scene image using a box around the block.

Something else that functioned well was the experiment function I created to displace the block containing Wally, randomly placing him around the cluttered scene image. This really helped me understand how the algorithm functions. The more the blocks to compare with, the more accurate the outcome.

One drawback of the program is that to find a very randomly displaced Wally, the linear search has to shift the block 1 by 1 pixels at a time, which is not time efficient in comparison to the other shifts. If I were to improve the program, I would want to find a faster way to find a randomly displaced Wally faster.

References

Contributor and Ringnes, E. (2005) *What is heap?* - definition from WhatIs.Com. Available at: http://whatis.techtarget.com/definition/heap (Accessed: 02 December 2016).

cplusplus (no date) *Pointers - C++ tutorials*. Available at: http://www.cplusplus.com/doc/tutorial/pointers/ (Accessed: 04 December 2016).

*C++ multi-dimensional arrays* (2016) Available at: https://www.tutorialspoint.com/cplusplus/cpp\_multi\_dimensional\_arrays.htm (Accessed: 05 December 2016).