Group Warshall Report

Design:

We made many decisions of our own during the project development. The code is structured as follows. The key classes are the Image class and the volume class which stores the 2d data and 3d data. Data is stored as 1-d consecutive unsigned char array in both classes. Furthermore, we provide various methods to access the properties of the classes. The data storage structure is for good cache behaviours. The Filter and Projection class are defined as static functions since we want to apply it rather than create instances of the class. The Slice class will have an image stored in it. We also create some helper classes for extra functionalities. The Exception classes have different exceptions and error handling. The **MemManager** class is created for automatic memory management [1].

Padding: In the current version, all the filters will be pad with 0 if the user wants to pad the images which could be improved as padding with 0 doesn't always result in the best behaviours.

Algorithm: The median filter uses quickselect[2] which is an O*(n) algorithm to find the median. However, suppose we have a kernel size of 5*5*5, then the adjacent kernels will have 100 elements in common and we didn't exploit this information to accelerate the algorithm. Ideally, it would be faster to use a rolling window median or a histogram selection to find the median. Another improvement we would make is that the pixel operations are highly independent, which is perfect for multi-thread programming. The last improvement we want to make is that we're using a lot more memory than needed in the 3d gaussian filter. For example, suppose we're applying a 3*3 kernel to a 100*100 image, we only need 300*4 bytes extra memory, which means we only need kernel size/2+1 rows to keep the original data.

Description:

2D Filter: For all types of 2D filters, we only care about RGB channel (or grayscale channel if the image is already in grayscale). If there is a transparency/alpha channel, we leave it unchanged.

Color correction filters:

Grayscale: Grayscale filter converts RGB image to grayscale image. We use the luminance method in our project. The formula is 0.299R + 0.587G + 0.114B.

Brightness: Brightness filter is to change the brightness of an image. We implement two types of brightness filter. The first one is to add a user-defined value to each pixel. And the second one is to set the average of pixels in all channels to 128. We first calculate the average value of each channel and use that to obtain the overall average. We then calculate the rate of the overall average to 128 and multiply each pixel to that rate to set the overall average to 128. In this function, we use the max (min (pixel, 255), 0) to prevent the pixel from going out of bounds.

Automatic color balance: Automatic color balance is to adjust the color levels of an image, which can make the appearance more robust and natural-looking. In this function, we first calculate the average value of each channel and use the minimum one as standard. This will avoid imbalance caused by value overflow. For each channel, we calculate the relative rate of the average value to the minimum one and then multiply each pixel to the corresponding rate.

Contrast enhancement filters:

Histogram equalization**:** This technique is used to redistribute the pixel values in an image to achieve a more uniform histogram, which results in a better contrast image.

1. First create a vector with 256 integers. Calculate the histogram of the input image, which is a graph that shows the frequency of occurrence of each pixel value in the image. This is done by looping over the image pixels and incrementing the value in vector corresponding to the image pixel.
2. Calculate the cumulative distribution function (CDF) of the input image by summing up the histogram values in the vector.
3. Normalize the CDF values to obtain a probability distribution function (PDF) and multiply the PDF by 255 to get pixel ranges from 0 to 255.
4. Create an empty image pointer and map the values of the input image pixels to PDF function to get the output image pixels. Replace the pointer in Image class with this new pointer and destroy the previous pointer.

HSV filter: We implemented this algorithm in the code. We did not describe it because it is an optional filter.

RGB contrast enhancement: This filter is used to enhance the contrast of the image by increasing the difference between the light and dark regions of the image. Under the hood this filter applies histogram equalization. However, rather than equalizing all the colour channels that can lead to colour imbalance, this filter converts RGB image into HSV colour space and then histogram equalizes values channel only. Then converts the image back to RGB.

1. Convert each pixel values from RGB to HSV by looping over each pixel as described in above **HSV filter**.
2. Apply histogram equalization as described in the above **Histogram equalization filter** to **values** channel only. Equalisation is not done for **Hue** and S**aturation** channels, because they contain colour information.
3. Convert enhanced pixels back to RGB as described below.
    - The function takes H, S ,V, R, G and B values as reference.
    - The function first calculates the intermediate values p, q, and t based on the input values.

Then, based on the range where the Hue lies in the colour space, set the values of r, g, and b using a set of switch statements. This filter gives visually detailed images compared to simple brightness filters.

Image blur filters:

Median blur: Median blur filter replaces each pixel in an image with the median value of the pixels in its surrounding neighborhood. It can reduce noise and smooth out an image. We let the user select the padding size. According to the kernel size and the padding size, we can derive that the starting loop point is at (kernel/2 – pad). For each pixel, we use an array of size kernel*kernel to store the pixel values around the center point. Then, we use the merge sort to get a sorted array and use the median index to obtain the median value in that kernel. To prevent the original image from being changed during the iteration, we allocate a block of memory with the same size as the original image on the stack to represent the updated value, and then copy that value to the original image after all iterations.

Box blur: This is same as median blur. The only difference is we set the center point to average value inside that filter.

Gaussian blur: Gaussian blur replaces each pixel in an image with the weighted average value of the pixels in its surrounding pixels. Our implementation loops through every image pixel. In each iteration, we apply the Gaussian kernel to its surrounding pixels to get the weighted average value. The user can decide the standard deviation of the Gaussian function. We then create two 1-D vectors fitting the Gaussian distribution and do the vector multiplication to get a 2-D Gaussian kernel. Like Median blur and Box blur, we use an allocated buffer to prevent the original image from being changed during iterations. This algorithm can be improved as the gaussian kernel is separable, but we didn't implement it for 2d version as it is already fast.

Edge detection: For the four edge detection filters, they all detect edges by calculating the gradient of each pixel. They all have two filters, **one for detecting horizontal edges and one for detecting vertical edges**. For each pixel, the horizontal edge gives the result G_x, and the vertical edge gives the result G_y. The gradient is approximated by $\sqrt{G_x^2 + G_y^2}$. The only difference is that the filter coefficients are not the same, so we abstract the edge detection filter into two parts: One for setting coefficients for different filters, which is where filters differ from each other, and the other for applying the filter. Due to the symmetry of filter, we use a trick here: for each type of filter except for Roberts-cross, we use a 1-D array to store the coefficients of both horizontal and vertical filters. In this way, for horizontal filter, we can index the array in row-major way while for vertical filter, we can index in column-major way. After box blur, the edge detection algorithms can work better.

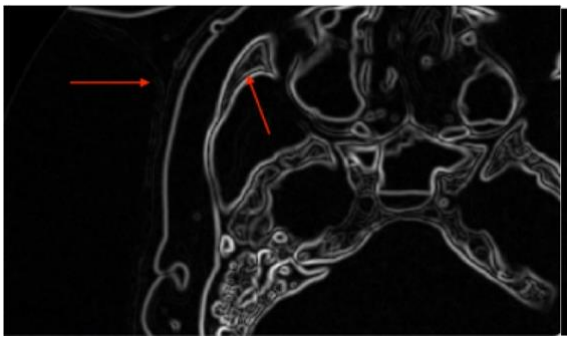

Figure: 1

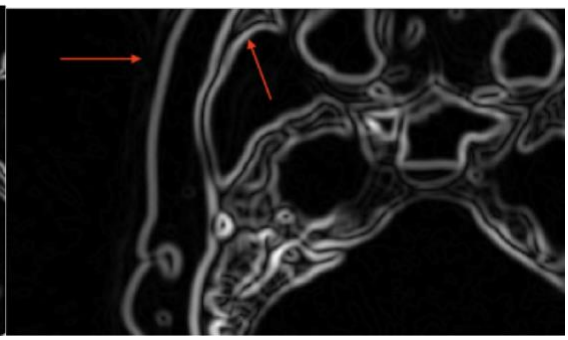

Figure: 2

The Figure: A1 is the edge detection after box blur, and Figure: A2 is without box blur. We can see that a lot of "false" edges disappear after box blur (As shown by the red arrays). That's because box blur removes noise and small details that are not edges, which causes the edge detection to be more robust and accurate.

3D Filter:

Gaussian blur: Gaussian blur replaces each pixel in an image with the weighted average value of the pixels in its surrounding pixels. The input is the same as the 2d version. However, because 3d version needs significantly more computation than the 2d version, we employ different algorithms for the 3d version. The key observation here is that the gaussian kernel is separable, this means that we can apply the 1d kernel individually to reduce the running time from O(Y*n^3) to O(Y*3n) with Y being the size of the volume [3]. However, this will require 8 times more memory than the naïve implementation as to achieve accuracy we need to keep the data in float array, which is 4 times larger than an unsigned char array, and we can't utilize the original memory, so we need two float arrays. The upside is this will offer roughly x10 acceleration. As a result, we implement both versions and let the user decide which version to use. This algorithm doesn't have good cache behavior compared to projection algorithms. It will have 1 cache miss per kernel size.
Median blur: This is same implementation as 2D, except that this has 1 more dimension. The key difference from the 2D version is that this function uses **quickselect** to find the median, which is faster than sorting the whole neighborhood. However, the running time of this function is still slow, by profiling the code, the selecting function makeup 75% of the instruction executed. Further improvement suggestions are discussed in the design choice part of the report. Same as gaussian blur, this function doesn't take advantage of the cache coherence. In general, it will have 1 cache miss per kernel size.

Projections:

The projection class offers a projection function which takes a volume as input and apply the input function on the input slab coordinates. Because we expect the user to perform multiple projections after loading the data and applying the blur filters, the data storage structure is mainly optimized for this function. The way we store the data will store the pixel for different images at same height, width and channel consecutively in the memory. Which provides a very good cache behavior. We also notice that different projections are using the same data, so we write the function in a way that the projection method is controlled by the passed-in function. Though a downside for this is it will use more memory as max, min and average functions can be applied rolling. This function will return a pointer pointed to an image on heap whose address is stored in the variable my_grabage_collector. To free the image, we need to delete the pointer as well as remove the address from my_grabage_collector.

Slice:

The slice function is performed by calling the constructor of the slice class. The class contains a pointer to the volume which generates the slice, a pointer to the resulting image, and the direction of the slice(xz or yz). The image pointer is managed by the slice class but the volume pointer is not. Upon calling, the constructor will call the slice_xz function or slice_yz function depending on the passed_in direction. It will also check if the position is valid. If the position is not valid it will throw an exception which is caught in the main control flow.

Performance:

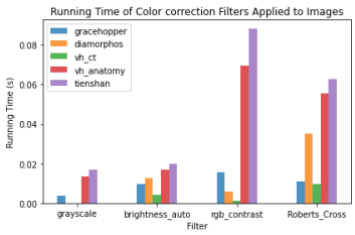

Figure:3

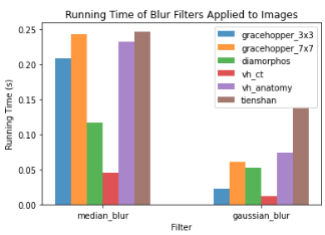

Figure: 4

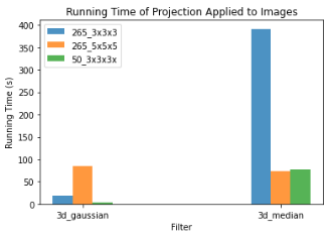

Figure: 5

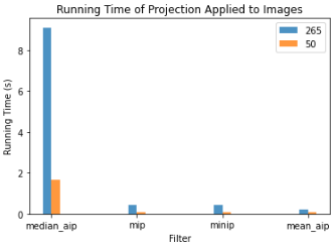

Figure: 6

Some images in Images folder are used for 2d filter analysis and confuciusornis is used for 3d operations analysis. For 2d filter, comparing images (512x600x4) with (512x512x1), with similar image size, the processes of applying the filters on images with single channel are faster than more channels (Fig 3). Comparing images (512x600x4) with (2160x1440x4), with the same number of channels, those on images with smaller size are faster than larger size (Fig 3). Each filter performs slightly differently according to the features. For roberts' cross, it is generally faster on images of all cases than other edge detection methods. For blur filter (Fig 4), bigger kernels always slow down the running speed. For both 3D filters, with the same number of images, larger kernel size consumes more running time (Fig 5). While with the same size of kernel, the more the images are the slower they are processed. And this is the same case for all methods of projection (Fig 6). It is worth mentioning that every operation relative to median requires sorting the data, which greatly slows down the running compared to others horizontally. See more accurate data from Appendix A.

Advantages/ Disadvantages:

Reflecting on the project, we can acknowledge that our code has its advantages and disadvantages based on decisions we had to make, and the time spent on each section. One of our main priorities was to maximize the speed which our program runs, in which we have implemented this priority in many ways seen below:
Advantages:

- Used '-O3' compiler which provides faster execution and better code optimization [4].
- Quickselect sorting algorithms used for finding median faster.
- A memory management class, being a failsafe program that is used when use opts to quit the program.
    - Will delete pointers and allocate/deallocate memory in a nested pointer situation to optimise memory.
- Store data as unsigned char array, which can be easily modified (act like bytes in which can access each bit individually)
    - High flexibility, high accessibility speed (compared to e.g. vectors)
    - Good cache behaviour: faster access in 3D for tasks such as projection
- Make file automates and speeds up the build process of the whole project
    - Will combine all compiling instructions, keep track of dependencies between files, and won't compile the same thing twice (recompiles only areas affected when changes are made)

Disadvantages

- Median finding filters still take longer to run compared to other filters. Could possibly test other methods such as 'median of medians' algorithm [5].
- Many nested for loops, could be avoided in using containers such as vectors which have iterators
    - Could explore using parallel processing (multi-threading) to increase speed.
- Repeated code in filter class which can be replaced by function templates within the filter class
    - Have already done so for convolution filters (e.g function 'apply_kernel_det', however due to time constraints has not yet been implemented in other methods (e.g blurring).

Team and Responsibility:

Username	Name	Github
gmn22	Gulam Nabi	edsml-gmn22
yz3522	Yihang Zhou	acse-yz3522
xq322	Xingjian Qi	acse-xq322
lbc19	Luisina Canto	edsml-lbc19
hl1222	Hongyuan Liu	edsml-hl1222
ky322	Kexin Yan	acse-ky322

Luisina Canto: Code and report of colour correction filters, commenting, advantages and disadvantages of report.
Hongyuan Liu: Code and report of image blur, user interface, makefile writing.
Gulam Nabi: Code and report of contrast filters, exceptions and memory management, license.
Xingjian Qi: Code and report of 3d filters, slicing, user interface, decision of report.
Kexin Yan: Code and report of edge detection, documentation, performance of report.
Yihang Zhou: Code and report of projections, testing, generating executable files for both systems.

Reference:

[1] Zehra, F., Javed, M., Khan, D., & Pasha, M. (2020). Comparative analysis of c++ and python in terms of memory and time.
[2] Wikipedia contributors. (2023, January 9). Quickselect. In *Wikipedia, The Free Encyclopedia*. Retrieved 15:00, March 24, 2023, from https://en.wikipedia.org/w/index.php?title=Quickselect&oldid=1132662679
[3] Zhang C. (2018). A Basic Introduction to Separable Convolutions. https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
[4] Caio Rordrigues. (2021). CPP/C++ Compiler Flags and Options. https://caiorss.github.io/C-Cpp-Notes/compiler-flags-options.html#org3aa59c3
[5] Wikipedia contributors. (2023, March 2). Median of medians. In *Wikipedia, The Free Encyclopedia*. Retrieved 15:21, March 24, 2023, from https://en.wikipedia.org/w/index.php?title=Median_of_medians&oldid=1142498530

Appendix A

Table A1: Performance documentation of all 2D filters.

Operation	Images	Kernel size	Seconds
Grayscal filter	Gracehopper(512x600, 3+1 channels)	/	0.003950
	Dimorphos(1446x1080, single channels)—-size+channel		0.000011
	Vh_ct(512x512, single channel)--channel		0.000004
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.013643
	Tienshan(2160x1440, 3+1 channels)--size		0.016915
Automatic colour balance filter	Gracehopper(512x600, 3+1 channels)		0.005714
	Dimorphos(1446x1080, single channels)—-size+channel		0.013171

	Vh_ct(512x512, single channel)--channel		0.002973
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.017636
	Tienshan(2160x1440, 3+1 channels)--size		0.019922
Brightness filter- Automatic	Gracehopper(512x600, 3+1 channels)		0.009542
	Dimorphos(1446x1080, single channels)—-size+channel		0.012857
	Vh_ct(512x512, single channel)--channel		0.004145
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.016745
	Tienshan(2160x1440, 3+1 channels)--size		0.019748
Brightness filter- shift	Gracehopper(512x600, 3+1 channels)		0.006231
	Dimorphos(1446x1080, single channels)—-size+channel		0.009139
	Vh_ct(512x512, single channel)--channel		0.001913
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.009786
	Tienshan(2160x1440, 3+1 channels)--size		0.013447
RGB contrast enhancement	Gracehopper(512x600, 3+1 channels)		0.015818
	Dimorphos(1446x1080, single channels)—-size+channel		0.005853
	Vh_ct(512x512, single channel)--channel		0.001153
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.069395
	Tienshan(2160x1440, 3+1 channels)--size		0.088235
Histogram equalisation for greyscale	~~Gracehopper(512x600, 3+1 channels)~~		
	Dimorphos(1446x1080, single channels)—-size+channel		0.008417
	Vh_ct(512x512, single channel)--channel		0.001821
	~~Vh_anatomy(2048x1216 3 channels)—size+channel~~		
	~~Tienshan(2160x1440, 3+1 channels)--size~~		
HSV colour space	Gracehopper(512x600, 3+1 channels)		0.008440
	~~Dimorphos(1446x1080, single channels)—-size+channel~~		
	~~Vh_ct(512x512, single channel)--channel~~		
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.045326
	Tienshan(2160x1440, 3+1 channels)--size		0.059395
Median blur filter	Gracehopper(512x600, 3+1 channels)	3x3	0.208397
		7x7	0.243183
	Dimorphos(1446x1080, single channels)—-size+channel	3x3	0.116842
	Vh_ct(512x512, single channel)--channel		0.044678
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.232784

	Tienshan(2160x1440, 3+1 channels)--size		0.247089
Box blur filter	Gracehopper(512x600, 3+1 channels)	3x3	0.038873
		7x7	0.099538
	Dimorphos(1446x1080, single channels)—-size+channel	3x3	0.049492
	Vh_ct(512x512, single channel)--channel		0.011687
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.073006
	Tienshan(2160x1440, 3+1 channels)--size		0.138729
Gaussian blur filter	Gracehopper(512x600, 3+1 channels)	3x3	0.022073
		7x7	0.060404
	Dimorphos(1446x1080, single channels)—-size+channel	3x3	0.052304
	Vh_ct(512x512, single channel)--channel		0.011935
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.074413
	Tienshan(2160x1440, 3+1 channels)--size		0.139040
Sobel edge detection filter	Gracehopper(512x600, 3+1 channels)	/	0.020360
	Dimorphos(1446x1080, single channels)—-size+channel		0.064848
	Vh_ct(512x512, single channel)--channel		0.013470
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.088998
	Tienshan(2160x1440, 3+1 channels)--size		0.104793
Prewitt edge detection filter	Gracehopper(512x600, 3+1 channels)		0.021299
	Dimorphos(1446x1080, single channels)—-size+channel		0.062431
	Vh_ct(512x512, single channel)--channel		0.014272
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.089047
	Tienshan(2160x1440, 3+1 channels)--size		0.102768
Scharr edge detection filter	Gracehopper(512x600, 3+1 channels)		0.021686
	Dimorphos(1446x1080, single channels)—-size+channel		0.062372
	Vh_ct(512x512, single channel)--channel		0.013851
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.090536
	Tienshan(2160x1440, 3+1 channels)--size		0.102926
Roberts' Cross edge detection filter	Gracehopper(512x600, 3+1 channels)		0.011122
	Dimorphos(1446x1080, single channels)—-size+channel		0.035229
	Vh_ct(512x512, single channel)--channel		0.009577
	Vh_anatomy(2048x1216 3 channels)—size+channel		0.055702
	Tienshan(2160x1440, 3+1 channels)--size		0.062890

Table A2: Performance documentation of all 3D filters and projections

Operation	#Images	Kernel size	Seconds
3d gaussian	265	3x3x3	19.481720
	265	5x5x5	85.410211
	50	3x3x3	3.583619
3d median	265	5x5x5	391.635575

	265	3x3x3	74.170238
	50	5x5x5	77.026588
Median_aip	265	/	9.102626
	50	/	1.655205
mip	265	/	0.428950
	50	/	0.097048
Minip	265	/	0.428013
	50	/	0.097227
Mean_aip	265	/	0.230383
	50	/	0.066726