ADVANCED PROGRAMMING GROUP ASSIGNMENT

IMAGE FILTERS, PROJECTIONS AND SLICES

Team Members:

Zepeng Chen	acse-zc522
Ruichen Ding	acse-rd422
Georgie Mercer	edsml-gm1722
Chunyang Wang	acse-cw17222
Lizzie Withers	edsml-lcw22

FILTERS

2D

COLOUR CORRECTION

The **grayscale** filter takes a three channel RGB image and returns a single channel image of the same dimensions. The output value of each pixel is equal to the average intensity of that pixel across its three input channels. This is calculated by looping through the range of the width and height of the image, summing the pixel values across the three channels at each pixel, dividing this by 3 to find the average and rewriting this to the first channel of the image which is what the function returns.

The **colour balance** filter can be used to enhance RGB images which are dominated by a colour. Good example of this are underwater photos, which before processing are various shades of blue. Colour balancing ensure the average of all three channels is equal, resulting in a less tinted output. The difference between each channel's average and the average of all three channels is found, and this difference is then subtracted from all pixels in each channel. The result effectively increases/decreases the brightness of each channel, to ensure no one channel dominates. It is worth noting, if the shift results in a pixel intensity below 0 or above 255, the value is restricted to these.

The **brightness** filter can be used to improve the appearance of very dark, or very bright images. It works for both grayscale and RGB images and has two options:

- Manual brightness adjustment – the user passes a value that specifies the increase or decrease in pixel intensity desired (ranging from -255, which would make the whole image black, to 255, which would make the image white). Every pixel in every channel is adjusted by the amount specified, and as with the colour balance, the results are restricted between 0-255.
- Automatic brightness adjustment – if the user passes 0 as the adjustment factor, the program will automatically find the difference between the average intensity of the image (across all pixels and channels) and the default (128). Every pixel in every channel is then adjusted by this difference, resulting in an image with an average intensity of 128.

The implemented **histogram equalisation** algorithm aims to improve the contrast of an input image by more uniformly distributing pixel intensities across each colour channel. After calculating the histogram of each channel, the function computes the cumulative distribution function (CDF), normalised CDF, and a lookup table for mapping pixel values (Lu et al., 2010). To generate the equalised image, the lookup table is applied to each pixel in each channel. This method is useful for improving the quality of low-contrast images, making them more visually appealing and suitable for further processing.

IMAGE BLUR

The **image blur** implementation is a versatile function that allows for image blurring using three different methods: median, box (mean), and Gaussian filters (Chandel & Gupta, 2013). It takes an input image, the desired filter method, and the kernel size as parameters. For the Gaussian filter, it convolves the Gaussian operator (σ=1) with the image. For median and box filters, the function pads the image, iterates through each pixel and channel, and collects the values within the specified kernel. Depending on the method, the median or mean of these values is assigned to the output image. This implementation offers flexibility in blurring techniques to cater to various image processing requirements.

EDGE DETECTION

This function performs **edge detection**, which is a method to outline distinct objects within an image. The filter class function `Object()` sets the kernel matrix and padding size based on the edge detection operator selected, which can be based on the established implementations '*Sobel*', '*Prewitt*', '*Scharr*', or '*Roberts Cross*' (Khrayshchev, 2010). The edge detection function converts the input image to grayscale before selecting the appropriate operator pair based on the method specified. The image is convolved separately with the x and y operators, and the results are combined to produce the final edge-detected image (Peli & Malah, 1982). It is worth noting that for the '*Scharr*' and '*Sobel*' operators in particular, better results are often achieved if a Gaussian or box blur is applied first, as these will smooth out any edges resulting from image noise.

3D

IMAGE BLUR

The implemented **3D blurs** follow a Gaussian or Median method. The Gaussian filter first calculates a kernel based on the input kernel size and applies it to each image in the 3D dataset. The kernel is generated using the Gaussian function below (with σ=1), and the filtering is done by iterating over each pixel in the image and multiplying it by the corresponding kernel value (Bomans et al., 1990).

$$k_\sigma(x,y,z) = \frac{1}{\sqrt{(2\pi)^3\sigma^3}} exp\left(-\frac{x^2+y^2+z^2}{2\,\sigma}\right)$$

The filtered image is then written to the output folder. The median filter works by sliding a window over the image, computing the median pixel value within the window for each channel, and writing the result to the output folder. The window size is based on the input kernel size, and the median value is calculated using a quicksort algorithm. Both algorithms are designed to handle 3D image datasets and can be adjusted according to the input kernel size and output folder.

PROJECTION

The **projection** feature defines a function that performs projection on a 3D volume to produce a 2D image. The function takes a Volume object as input, which contains the paths to the 3D image files. The user is prompted to select a projection method (max, min, or average) and a projection plane (XY, XZ, or YZ). The dimensions of the output image are determined based on the selected plane. The function iterates through each 3D image in the volume, reads it in, and calculates the projection of the image onto the selected plane using the selected method. The final projected image is returned as an FImage object. This technique has important applications when volumetric measurements are analysed in 2D, e.g. in the fields of medical imaging or material science (Li et al., 2020) .

SLICE

The **slice** feature provides a function for slicing a 3D image dataset into 2D images. It accepts the start and end indices of the images to be sliced, the axis along which to slice, and the path to write the sliced images to. It uses different approaches for each axis to slice the 3D image data. If the axis is 0, it slices the voxel along the z-axis, and if the axis is 1 or 2, it slices the voxel along the y-axis or x-axis, respectively. The function also creates a new folder if the specified path does not exist and writes the sliced images to that folder.

PERFORMANCE

CODE PROFILING

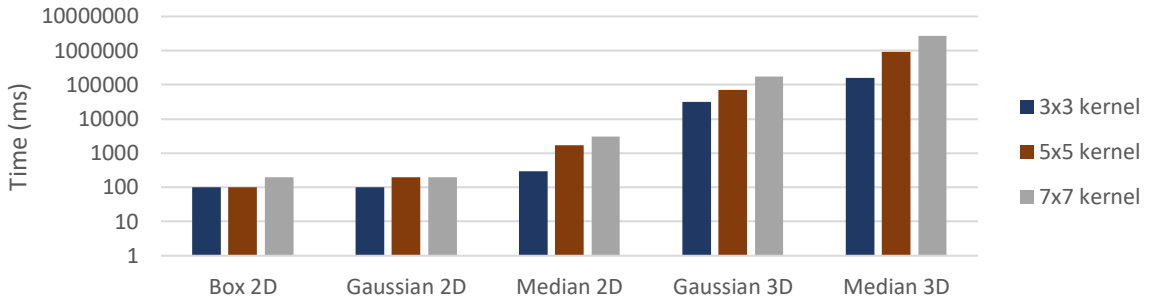

Figure 1: Performance data of implemented convolution filters ***(all 2-D filters tested on gracehopper.png and all 3D on the confuciusornis dataset).***

IMAGE READ/WRITE

Two classes have been constructed to facilitate image writing and reading. The first one is Image class (defined in `image.h`), which use a 3D vector to store the image data. This implementation is more intuitive and easy to use, because programmer can use indexing to intuitively access elements in the voxel. However, it is time consuming to iterate over all images when the dataset is large, so a different class, FImage has been created (defined in `fastImage.h`) to address this problem (especially when applied to 3D datasets). In this version, the data is stored in a 1D array, and data can be accessed using getter and setter. This method is significantly faster, but less intuitive to programmers, as 1D array a pixel at (i, j, k) cannot be directly accessed using a multiple `[]` operators.

PROJECTION & FILTERS

The program is designed to be able to handle large datasets. So, when handling 3D datasets, instead of loading the entire dataset into the memory, a more memory-efficient method was applied: A buffer is constructed using double queue, (which ensure constant access time and insertion/deletion at end/front), which only stores most relevant data. That is, for projection, at each time step, only the next image to be processed is stored. After comparing maximum/minimum or summing the image pixel value, this pixel is removed from memory. For 3-D filters, only number of the kernels size of images is loaded into the buffer at one time, and in the next processing step, instead of constructing a new buffer, the old buffer is utilized, only the image at the front of the buffer is removed, and after that, a new image is inserted to the buffer. This implementation ensures we can handle arbitrary sized datasets (if the image in the buffer is not larger than the available memory) without running into memory issue. However, this method is slower compared to the memory intensive method because it read files form disk frequently.

VOLUME READING

Loading the volume into the memory in one go does not work when dealing with large datasets. Therefore, in the volume implementation, only the file paths of the images are stored. When dealing with small dataset and doing projection, it is sometimes useful to load the dataset into the memory, so a `preload` method is implemented, which allows the programmer to load the images using with loaded path. An `unload` method is also created to prevent memory leakage.

FURTHER IMPROVEMENTS

There are several areas of further development we would have liked to work on if time permitted:

- Building out a more intuitive graphical user interface, that allowed the batch processing of multiple images
- The ability to process additional types of input image (.tif for example)
- Investigating further methods to speed up the processing of larger images/3D datasets

APPENDIX – TASK BREAKDOWN

File structure & Git management	Chunyang Wang
Image read & write	Chunyang Wang
2D colour correction filters	Georgie Mercer & Lizzie Withers
2D blur filters	Lizzie Withers
2D edge detection filters	Lizzie Withers
3D blur filters	Ruichen Ding & Zepeng Chen
3D projections	Ruichen Ding
3D slices	Chunyang Wang
User interface (main.cpp)	Lizzie Withers & Ruichen Ding
Testing	Georgie Mercer, Zepeng Chen & Ruichen Ding
Documentation	Lizzie Withers

REFERENCES

Lu, L., Zhou, Y., Panetta, K., & Agaian, S. (2010). Comparative study of histogram equalization algorithms for image enhancement. *Mobile Multimedia/Image Processing, Security, and Applications 2010*, *7708*, 337-347.

Khrayshchev, D. A. (2010). On a method of edge detection in digital images. *Vestnik of Astrakhan State Technical University. Series: Management, Computer Sciences and Informatics*, (2), 181-187.

Peli, T., & Malah, D. (1982). A study of edge detection algorithms. *Computer graphics and image processing*, *20*(1), 1-21.

Chandel, R., & Gupta, G. (2013). Image filtering algorithms and techniques: A review. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(10).

Bomans, M., Hohne, K. H., Tiede, U., & Riemer, M. (1990). 3-D segmentation of MR images of the head for 3-D display. *IEEE transactions on medical imaging*, *9*(2), 177-183.

Li, M., Chen, Y., Ji, Z., Xie, K., Yuan, S., Chen, Q., & Li, S. (2020). Image projection network: 3D to 2D image segmentation in OCTA images. *IEEE Transactions on Medical Imaging*, *39*(11), 3343-3354.