TEAM HUFFMAN: 2D & 3D IMAGE PROCESSING

Members:

Yuheng Chen	acse-cc223	2D filters (Gaussian, median, box blurring), 3D filters (Gaussian, median blurring, testing framework, 3D unit tests, 3D performance evaluations
Francois Crespin	acse-fc1223	2D filters (grayscale, histogram, salt and pepper, threshold), 3D filters (slicing), 3D units tests, 2D user interface, doxygen documentation
Yixuan Jiang	acse-yj3023	2D filters (Prewitt, Sobel, Scharr, Robert's cross), 3D projections (MIP, MinIP, AIP), 3D unit tests, 2D unit tests, 3D user interface
Atys Panier	edsml-azp123	2D filters(grayscale, brightness, salt and pepper), 2D user interface, Classes (image, 2D filters class)
Walid Sheikh	edsml-was23	2D filters (grayscale, histogram, salt and pepper), testing framework, unit tests (2D filters – colour correction), report writing, report compilation, readme, license
Xuan Zhan	edsml-xz3323	2D filters (Prewitt, Sobel, Scharr, Robert's cross), 2D unit tests, 2D performance evaluations, readme file for main and performance evaluations

---

Summary:

This project involved creating a C++ program focused on sophisticated image and 3D data processing techniques. It required the implementation of 2D image filters, including colour correction, brightness adjustment, histogram equalisation, thresholding, noise addition, and convolution-based blurring and edge detection. For 3D data volumes, we developed Gaussian and median filters, orthographic projections like Maximum Intensity Projection (MIP), and slicing through volumes to reveal internal structures. In addition, recommended guidelines were followed on code and library structure, a custom testing framework was included, and emphasis was placed on code sustainability and scalability, as well as a user interface to enhance accessibility. This report delves into these listed aspects of the project, and caps it off with optimisation and performance reporting, and future modifications for wider deployment and use.

2D IMAGES

- <u>**Colour Correction & Per-Pixel Modifiers**</u>
    - **Grayscale Filter:**

The grayscale filter works using the luminance method where it applies specific weights to each of the three image channels (0.2126 for red, 0.7152 for green, and 0.0722 for blue), reflecting their luminance contributions. It also checks if the input image is already grayscale by accounting for the number of channels at the onset. Additionally, this function like others after it also highlights if there are errors in loading or saving the image as required by the user.

- - **Brightness Filter:**

This filter operates in two modes, automatic and manual, where automatic sets the brightness to be 128 by default across all channels. This is ensured through checking the average brightness value continuously. For manual mode, the user can specify a brightness adjustment value between -255 and 255 to be incremented on all channels, enhancing or reducing overall brightness. This adjustment also considers pixel values have to be maintained between 0 to 255, and clamps values that fall outside this range.

- - **Histogram Equalisation Filter:**

The histogram equalisation filter fundamentally enhances image contrast by redistributing the most common intensity levels across the full available range, effectively "stretching" the histogram. For grayscale images, this process involves adjusting the intensity levels to cover the entire spectrum from black to white. In RGB images, the algorithm first converts each pixel's colour from the RGB space to HSL (Hue, Saturation, Lightness) or HSV (Hue, Saturation, Value), focusing the equalisation process on the Lightness or Value component, which represents the intensity of the colour. This component's histogram is equalised, meaning the cumulative count of each intensity level is recalculated to ensure a uniform distribution across the range, then applied back to the image, either directly for grayscale or by converting the HSL/HSV values back to RGB for colour images.

  - **Thresholding Filter:**

The thresholding filter applies a binary thresholding technique to an input image, setting pixels to black or white based on a specified intensity threshold. For RGB or RGBA images, it first converts each pixel to the HSL colour space, applying the threshold to the lightness component to determine the output intensity: pixels with lightness above the threshold are set to white, while those below are set to black. In the case of grayscale images, the process is more straightforward—the pixel intensity itself is compared directly to the threshold. This method effectively segments the image into two distinct regions, enhancing contrast and isolating features based on brightness.

  - **Salt & Pepper Noise Filter:**

This filter introduces salt and pepper noise to an image, a process where a defined percentage of pixels are randomly selected and set to either black (0) or white (255), simulating a visual noise effect reminiscent of salt and pepper grains. This is achieved by first determining the total number of pixels to be affected based on the specified noise percentage by the user. Utilising the shuffle function from the algorithm library, the function selects these pixels randomly throughout the image. For each selected pixel, it assigns the colour value to either black or white, affecting all colour channels (RGB) while optionally preserving the alpha channel if present.

- **<u>Image Blurring</u>**
  - **Median Blur Filter (3x3, 5x5, 7x7 kernels etc.):**

The median blur filter collects the values from neighbouring pixels within a specified kernel size for each pixel, calculating the median by first gathering these values into a vector and then sorting them. This median value is then assigned to the current pixel, effectively blurring the image. The process is repeated for each channel separately, ensuring that colour images are processed correctly without mixing channel information. For images with an alpha channel (RGBA), the alpha channel is copied without modification, preserving transparency.

  - **Box Blur Filter (3x3, 5x5, 7x7 kernels etc.):**

The box blur filter works by averaging the pixel values within a square kernel of specified size for each pixel in the image. The single channel box blur function calculates this average for each pixel by summing the values of neighbouring pixels within the kernel's reach, adjusting for edge cases, and then applying an inverse kernel area factor to determine the new pixel value. This process is applied separately to each colour channel to prevent colour mixing, ensuring that the box blur effect is uniformly applied across the image. For RGBA images, the alpha channel is copied as-is, maintaining the original transparency.

  - **Gaussian Blur Filter (5x5, 7x7 kernels):**

The Gaussian blur involves applying a precomputed Gaussian kernel to each pixel in the image. This kernel, based on the specified kernel size and `sigma`, weights the contribution of each neighbouring pixel to the target pixel's new value, simulating the effect of light diffusing through a lens. The single channel gaussian blur function performs this operation for each pixel, adjusting the contribution of neighbours based on their distance from the centre pixel, and normalising the result by the sum of the kernel weights to maintain the original image brightness. This blurring technique is separately applied to each channel of the image to preserve the integrity of the colours, and for images with an alpha channel, transparency is retained by copying the alpha channel unchanged.

- **<u>Edge Detection Filters</u>**

Prior to the application of the following edge detection filters, the grayscale filter was applied to all images. In some cases, the edge detection filters performed better with certain image blurring filters applied beforehand. In our findings, this was the case for Tienshan.png (Gaussian, 5x5 kernel), Gracehopper.png (Gaussian, 5x5 kernel), Dimorphos.png (box, 3x3 kernel), Vh_anatoy.png (box, 5x5 kernel). This is illustrated in Figure 1 for Gracehopper.png.

- **Sobel (3x3 operators):**

Utilises two 3x3 kernels to detect horizontal and vertical edges by calculating the gradient of the image intensity at each pixel. This filter emphasises edges by computing the magnitude of the gradient, balancing edge strength and noise resistance.

- **Prewitt (3x3 operators):**

Similar to the Sobel filter, the Prewitt filter also employs two 3x3 kernels but with equal weight across the x and y directions. This results in a less computationally intensive process, making it faster but slightly less accurate at detecting subtle edges compared to the Sobel filter.

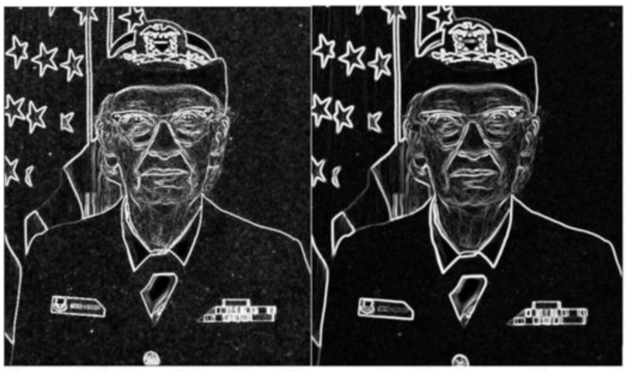
Figure 1. Edge detection filter applied to Grace Hopper's portrait. Gaussian blur with a 5x5 kernel applied beforehand (right), and no blur filter applied (left).

- **Scharr (3x3 operators):**

Offers a refinement over the Sobel filter by using kernels that give more weight to the pixels directly adjacent to the centre pixel. This results in a better approximation of the image gradient and more precise edge detection.

- **Robert's Cross (2x2 operators):**

Applies a simpler approach with two 2x2 kernels, focusing on diagonal differences in pixel intensities. This method is computationally efficient and particularly suited for detecting high-contrast edges, though it may be less effective at identifying subtle variations in intensity.

3D DATA VOLUMES

- <u>**Filters**</u>
  - **Gaussian Blur Filter:**

The 3D Gaussian blur filter starts by generating a Gaussian kernel based on the provided kernel size and `sigma`, ensuring the kernel is normalised so that its values sum to one. This kernel is used to apply a weighted average across each voxel (3D pixel) and its neighbours, determined by the kernel size, to blur the image. The function accommodates multi-channel data, applying the blur to each channel independently. The blurred data is then written back to the original volume. This method effectively smooths the volume data, reducing noise and detail.

- **Median Blur Filter:**

For the 3D median blur filter, instead of sorting all values within the kernel window to find the median, it approximates the median by calculating the average of the minimum and maximum values found within the kernel's reach. This method operates under the assumption that the data, such as CT scans, is uniformly distributed, making the approximation reasonable. The kernel iterates through each voxel in the volume and its neighbours defined by the kernel size, adjusting the values based on this median approximation. This approach significantly reduces computational complexity compared to a true median filter, especially beneficial for large datasets or real-time processing where performance is critical. Finally, the processed data is copied back into the original volume, completing the median blurring effect.

- <u>**Orthographic Projections**</u>

The orthographic projection functions read the 3D volume data and perform projection operations on the x-y plane, offering three different projection methods. Users can select different projection modes and specify the projection slab by input. In summary, for each of the following functions, they can:

  - **Maximum Intensity Projection (MIP):** Select the maximum pixel value along the z-axis corresponding to the current x-y point for projection.
  - **Minimum Intensity Projection (MinIP):** Select the minimum pixel value along the z-axis corresponding to the current x-y point for projection.
  - **Average Intensity Projection (AIP):** Select the average pixel value along the Z-axis corresponding to the current x-y point for projection.

- **Slicing**

  Given that the dataset provides images in the x-y plane, the program after processing into a 3D volume can output resulting images in the y-z and x-z planes.

PERFORMANCE EVALUATION

Since optimisation and scalability was a significant requirement, it was necessary to test and quantify the performance of the code in various deployment scenarios. These were varying image, volume, and kernel sizes, which would have some of the most notable effects on running times.

- **Image Size**

The data demonstrates graphed in Figure 2 showcases a clear correlation between image complexity and processing time. Median and Gaussian blurs increase time significantly, with the median blur on 4-channel images being the most time-consuming. Box blurs are consistently faster, suggesting a simpler calculation. Additionally, single channel images also processed the fastest.

- **Volume Size**

Table 1 shows the running time of 3D filters (Median and Gaussian) based on two given datasets based on two kernels.

In order to further analyse the performance based on volume sizes, we perform experiments based on different slice numbers with a fixed kernel size of 3, and the results are shown in Figure 3.

Figure 3 shows that as the slice numbers increase, the running time increases with a basic linear relation. This observation illustrates that each 2D image represents a cross-section or perspective of a 3D body, thus, the total processing time increases with the number of images that need to be processed. We can also analyse the volume size based on different image sizes from this plot. The running time of confuciusornis dataset with dimensions of 996x1295 for each image is much longer than that of fracture dataset (843x275), showing that the scale of images could also significantly influence the model efficiency.

- **Kernel Size**

We also performed experiments based on different kernel sizes (3, 5, 7) on the two datasets with a fixed slice number (30), and the results are shown in Figure 4.

The complexity of the median filter and Gaussian filter is directly related to the size of the kernel. For the median filter, as the kernel size increases, the number of neighbouring pixels considered for each pixel increases, as does the amount of data that needs to be sorted, thereby increasing the computation cost of determining

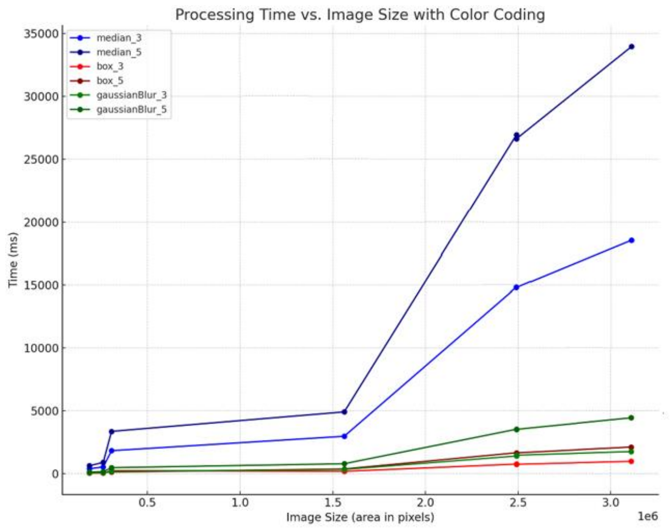

Figure 2. Running time for different image sizes

Table 1. Running time of the two datasets

Dataset	Scale	Filter	Kernel Size	Time (s)
confuciusornis	996x1295x265	Median	3	123
			5	520
		Gaussian	3	82
			5	320
fracture	843x275x1300	Median	3	112
			5	461
		Gaussian	3	72
			5	281

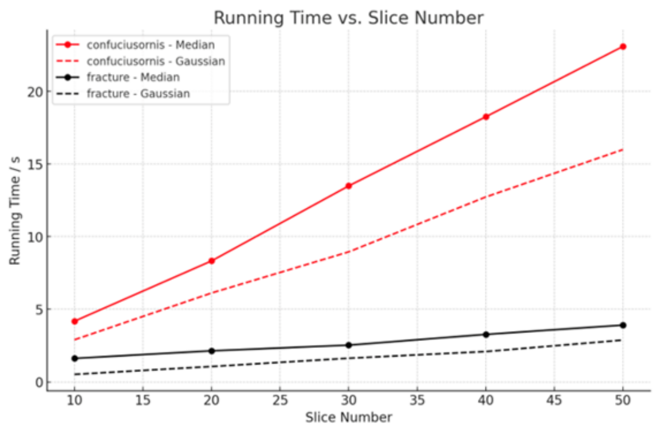

Figure 3. Running time with different slicing numbers

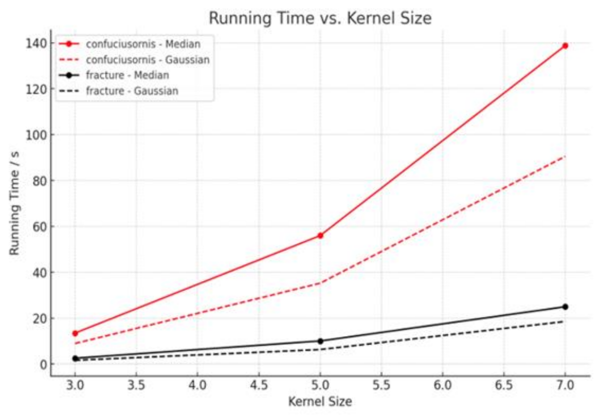

Figure 4. Running time with different kernel sizes

the median. Since the sorting operation is central to the median filtering process, its computational cost is significantly higher for larger kernel sizes compared to smaller ones. For the Gaussian filter, because it involves the multiplication of weights and pixel values and their accumulation, its sensitivity to increased kernel sizes may be slightly less than that of median filtering. Even so, as the kernel size increases, the number of multiplications and additions required for each pixel significantly increases, leading to an overall increase in runtime.

FUTURE IMPROVEMENTS & MODIFICATIONS

Despite the restriction against using OMP/threading for this project, we would suggest exploring parallel processing techniques to enhance the performance of our filtering and projection operations, making the application suitable for real-time processing and handling large images and data volumes. We also intend to integrate advanced machine learning-based filters, such as deep learning for denoising and super-resolution, to significantly improve accuracy and visual quality, particularly for complex datasets such as CT scans. Another great addition would be to develop a more intuitive graphical user interface (GUI) to improve user experience, featuring real-time previews and customisable workflows. Expanding our testing framework to include comprehensive automated tests will help us maintain code quality and performance. These modifications aim to extend our application's capabilities, improve its performance, and guarantee its adaptability to future technological advancements and user needs.