Advanced Programming 2023/24 – Assessment 3 – Group Project

Image Filters, Projections and Slices

Group Name:	Yen	
Student Name	**GitHub username**	**Tasks worked on**
Antony Krymski	acse-agk123	Colour correction filters, 3D image filters, user interface
Leo Mok	edsml-lm1823	Edge detection filters 3D projections
Bofan Liu	edsml-bl1023	Colour correction filters, Report,
Zeyi Ke	edsml-zk23	Image Blur, Report
Tianzi Zhang	acse-tz2523	Image Blur, 3D image test
Yifan Wu	acse-yw11823	Edge detection filters

[Note, this title page does not count towards the 4 page limit]

1 Algorithms Explanation

1.1 2D Image Filters

1.1.1 Grayscale:

The grayscale filter algorithm converts a colour image to grayscale by ensuring it has at least three channels (RGB), iterating through each pixel, and calculating the grayscale value for each pixel using a weighted formula. This formula combines the red, green, and blue values of each pixel to produce the grayscale equivalent. Finally, the algorithm applies the calculated grayscale value to the red, green, and blue channels of each pixel, resulting in a grayscale image.

1.1.2 Brightness:

The brightness filter algorithm operates by iterating through each pixel of an image, adjusting the intensity of its colour channels (such as Red, Green, and Blue) by a specified amount to modify its brightness. This process ensures that the pixel values remain within the acceptable range of 0 to 255, preventing any potential distortion or loss of colour fidelity.

1.1.3 Histogram equalisation:

The histogram equalization method is employed to enhance image contrast through two approaches:
Firstly, for grayscale images, it calculates the histogram to analyse intensity frequency, computes the Cumulative Distribution Function (CDF) for intensity mapping, and subsequently equalizes pixel intensities based on this CDF, effectively spreading intensities across the entire range.
Secondly, for RGB images, it converts pixels to HSL or HSV space, focuses on the Lightness (L) or Value (V) channel for brightness analysis, performs histogram equalization, maintaining hue and saturation, and finally converts the modified space back to RGB, thus enhancing image contrast while preserving colour integrity.

1.1.4 Thresholding:

The thresholding method is utilized to convert images into binary representations, distinguishing between foreground and background based on a predetermined threshold value. For grayscale images, it individually processes each pixel, assigning white (255) to those above the threshold and black (0) to others. Meanwhile, for colour images in HSV/HSL space, RGB pixels are converted accordingly. By thresholding the Value (V) or Lightness (L) channel, pixels with values above the threshold are set to white, with saturation adjusted for pure colour, while those below are set to black. Finally, the modified HSV/HSL is converted back to RGB, facilitating effective binary image creation while maintaining colour information.

1.1.5 Salt and pepper noise:

Salt and Pepper Noise method introduces salt-and-pepper noise by randomly turning a percentage of an image's pixels either black or white, affecting all colour channels except alpha.

1.1.6 Median blur:

The median blur filter is used to reduce noise and smooth images in image processing. It replaces the value of each pixel with the median value of the pixel values within a window, thereby achieving the blurring effect.
We use the QuickSelect method to find the median value of the pixels in the window. QuickSelect operates by selecting a pivot element and dividing the array into two subarrays based on the value of the pivot element. The kth smallest element in one of the subarrays is then recursively searched for based on the relationship between index k and the size of the subarray relative to the pivot. To handle pixels outside the boundaries, a boundary reflection method is used to ensure the integrity of the neighbourhood.

1.1.7 Box blur:
The box blur filter achieves its effect by applying a small window called a box kernel around each pixel in the image. This window calculates the average value of the surrounding pixels and assigns this average value to the current pixel, thereby achieving the blur effect.

We use the sliding window method to apply the blur filter. For each pixel and each colour channel in the image, a window of size kernelSize is created by sliding over the surrounding pixels with the current pixel as the centre. Only the actual number of pixels within the window is considered when calculating the average of the pixels within the window. If there are no pixels within the window (e.g., the current pixel is on an image boundary), the value of the current pixel is set to 0, thus avoiding out-of-bounds access.

1.1.8 Gaussian blur:
Gaussian blurring is achieved by applying a Gaussian kernel around each pixel of the image. First, we generate a Gaussian weight vector using the given standard deviation and kernel size to compute the Gaussian weights. Subsequently, Gaussian blur is applied to each row and column of the image separately. At each pixel position, the blurred pixel values are obtained by summing the weighted neighbouring pixel values and the results are stored in the output array. To deal with the out-of-boundary cases, the boundary reflection method is used. By applying Gaussian blur along rows and columns separately, the entire image can be blurred effectively.

1.1.9 Edge detection filters:
Edge detection filters such as Sobel, Prewitt, Scharr, and Roberts Cross are crucial tools for identifying image boundaries by highlighting areas of significant intensity changes. Initially, grayscale conversion simplifies intensity analysis, followed by gradient calculation using specific filter kernels ('gx' and 'gy'). These kernels compute horizontal and vertical derivatives, emphasizing intensity changes. The edge magnitude, determined by the Euclidean norm, represents the strength of the edges. Normalization within the display range (0-255) enhances edge visibility. Sobel, known for its directional detection, utilizes a 3x3 matrix for accurate edge detection. Prewitt, similar to Sobel but with different coefficients, provides a simpler gradient approximation. Scharr's improved gradient approximation enhances rotation invariance and edge direction accuracy. Roberts Cross uses a 2x2 matrix, specializing in detecting diagonal edges, suitable for high-frequency components in detailed or noisy images. These methods collectively offer robust edge detection capabilities essential for various image processing tasks.

1.2 3D Data Volume
1.2.1 3D Gaussian Blur:
The 3D Gaussian Blur technique applies a smoothing filter to volumetric data, such as medical images or 3D textures, using a Gaussian kernel. It begins by generating a 3D Gaussian kernel cube based on specified size and standard deviation parameters to determine the blur effect. Then, the algorithm iterates over each voxel in the volume, calculating a weighted average of the voxel and its neighbors using values from the Gaussian kernel. Normalization ensures the Gaussian values sum up to one, maintaining consistent image brightness. Edge handling is implemented through boundary checks, typically clamping positions outside the volume to the nearest valid voxel. The resulting smoothed value from the Gaussian kernel replaces the original voxel value, effectively smoothing the data.

1.2.2 3D Median Blur:
The 3D Median Blur filter effectively removes noise from volumetric data while preserving edges by analyzing each voxel's neighbourhood defined by the kernel size, calculating the median intensity

value within this neighbourhood, and assigning it as the new value for the central voxel. This process is iterated for every voxel in the volume, with edge handling ensuring the kernel remains within volume boundaries by clamping neighbourhoods near edges to available voxels.

1.2.3 Maximum intensity projection:

MIP visualizes the highest intensity values in a 3D volume along a particular viewing direction, often used in medical imaging to highlight structures like blood vessels. It iterates through a specified range of slices (or the entire volume), tracking the highest intensity value encountered for each pixel position across the slices. The output is a 2D image where each pixel represents the maximum value found along the line of sight through the volume, highlighting the brightest structures.

1.2.4 Minimum intensity projection:

MinIP is used to display the lowest intensity values, which helps in viewing structures with lower densities, such as airways in lungs or bones in CT scans. MinIP tracks the lowest intensity value across the selected slices of the volume along the z axis.The resulting 2D image represents the minimum intensity value found along the viewing path, emphasizing the darker regions.

1.2.5 Average intensity projection:

AIP calculates the average intensity of the structures along a viewing direction, useful for providing a general overview of the density or intensity distribution within the volume. It sums the intensity values for each pixel across the selected slices and divides by the total number of slices to calculate the average. The resulting 2D image represents the average intensity encountered for each pixel position, giving a more balanced view of the internal structures.

1.2.6 Slicing:

Slicing in volumetric data, such as 3D medical scans, involves extracting a 2D image from the 3D volume. The algorithm determines the slicing plane, either 'xz' or 'yz', specified by the 'plane' parameter. It then iterates through the volume to extract the slice, maintaining either the 'y' or 'x' coordinate constant. The extracted slice data is represented in a 2D array ('sliceData'). Specialized methods like Thin Slab AIP and MIP are available for specific slicing requirements, such as averaging voxel intensities or finding maximum intensity, respectively, along the z-axis within a specified range.

2 Performance Evaluation

Tienshan and stinkbug images are used to compare the performance evaluation for 2D filters. Volume 1 containing 5 slices and volume 2 containing 10 slices to evaluate performance.

2.1 Image size and Volume size

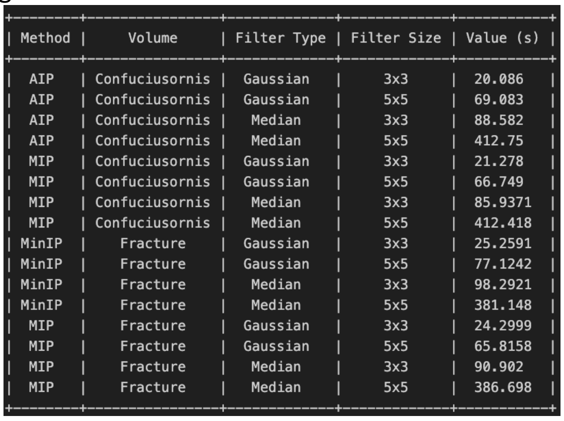
Filters applied to the Tienshan image, which has a higher resolution, consistently took longer to run. When applied to the stinkbug image, which has a lower resolution, all filters required less time. This pattern holds across all filters, indicating that the increase in the number of pixels leads to a proportional increase in computation time. Doubling the volume size roughly doubled the runtimes for 3D filters, showing a direct correlation between volume size and computational load. This suggests that the processing time for 3D filters scales linearly with volume size.

2.2 Kernel size and Complexity

The effect of kernel size on performance is evident for image blur filters. A larger kernel size results in a longer runtime because more pixels are involved in the calculations for each output pixel for all Median Blur, Box Blur and Gaussian Blur. Evidently below, median blur took significantly longer to run than any other algorithm, due to it being first sorted using quicksort which is O(nlog(n)). Every other algorithm only iterates over the image once and applies multiplications (with a kernel or a value) and is in constant time.

2.3 3D

Similarly, the 3D functions are mainly affected by kernel size and median blur, where larger kernel size further slows down the runtime. The projections are also significantly faster than the blur, taking around 10 seconds to compile for either sets of images; this is due to it having significantly less multiplications. All the 3D runtimes are displayed in Figure 2.



```
+--------+----------------+-------------+-------------+-----------+
| Method |     Volume     | Filter Type | Filter Size | Value (s) |
+--------+----------------+-------------+-------------+-----------+
|  AIP   | Confuciusornis |  Gaussian   |     3x3     |  20.086   |
|  AIP   | Confuciusornis |  Gaussian   |     5x5     |  69.083   |
|  AIP   | Confuciusornis |   Median    |     3x3     |  88.582   |
|  AIP   | Confuciusornis |   Median    |     5x5     |  412.75   |
|  MIP   | Confuciusornis |  Gaussian   |     3x3     |  21.278   |
|  MIP   | Confuciusornis |  Gaussian   |     5x5     |  66.749   |
|  MIP   | Confuciusornis |   Median    |     3x3     |  85.9371  |
|  MIP   | Confuciusornis |   Median    |     5x5     |  412.418  |
| MinIP  |    Fracture    |  Gaussian   |     3x3     |  25.2591  |
| MinIP  |    Fracture    |  Gaussian   |     5x5     |  77.1242  |
| MinIP  |    Fracture    |   Median    |     3x3     |  98.2921  |
| MinIP  |    Fracture    |   Median    |     5x5     |  381.148  |
|  MIP   |    Fracture    |  Gaussian   |     3x3     |  24.2999  |
|  MIP   |    Fracture    |  Gaussian   |     5x5     |  65.8158  |
|  MIP   |    Fracture    |   Median    |     3x3     |  90.902   |
|  MIP   |    Fracture    |   Median    |     5x5     |  386.698  |
+--------+----------------+-------------+-------------+-----------+
```

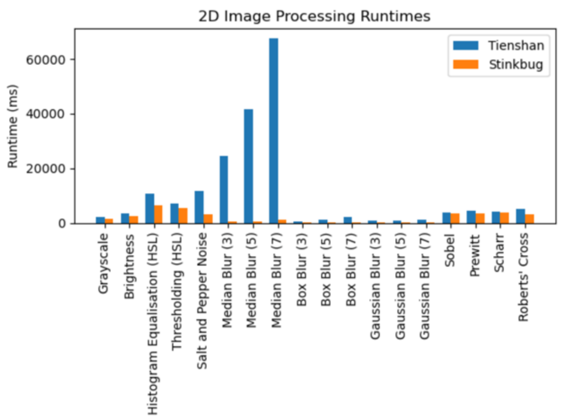


Figure 1. 2D processing Runtimes

Figure 2. 3D processing Runtimes

3 Potential Improvements/Changes

Implementing a GUI: After implementing most of the methods, the UI became quite convoluted with options branching 4 to 5 times and there are some edge cases of input that terminates the program instead of asking for a proper input. With a GUI and functionality such as a back button and always checking for correct input, our program would be much easier to use.

Default filters: We have tested various combinations of filters, such as first blurring to improve the performance of edge detection. Although most applications are different, we would benefit from having a default (set of) filters and parameters that are ready to use instead of having to search for the optimal order of filters and parameters.

Early Termination in Edge Detection: Introducing early termination logic in edge detection filters for areas with uniform intensity could save computation time by avoiding unnecessary calculations.

Use more complex algorithms: It can improve the performance of functions such as median blur. For example, consider using max-min heap sort, which has a more stable time complexity than fast selection and is suitable for large-scale data. In addition, it requires no additional space. Using more of the STL such as std::sort would also be efficient and good practice in C++.