**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 |  |  |
| Leo Mok |  |  |
| 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 |  |  |

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

# Algorithms Explanation

## 2D Image Filters

### Grayscale:

The grayscale filter algorithm converts a colour image into a grayscale (black and white) image.

1. Check Channels: Ensures the image has at least three channels (RGB) necessary for colour.
2. Loop Over Pixels: Iterates through each pixel in the image.
3. Calculate Grayscale: For each pixel, combines its red, green, and blue values using the formula to calculate the grayscale equivalent.
4. Apply Grayscale: Sets the red, green, and blue channels of each pixel to the calculated grayscale value, turning the image into grayscale.

### Brightness:

The brightness filter algorithm modifies the brightness of an image.

1. Loop Through Pixels: The function iterates through all the pixels in the image using a row (`height`) and column (`width`) structure. For each pixel, it calculates the pixel's address in the data array using its position and the number of colour channels (`channels`).
2. Adjust Brightness: It loops through each colour channel of the pixel (like Red, Green, and Blue) and increases or decreases the channel's value by a specified amount (`delta`) to adjust brightness.
3. Clamping Values: The new channel values are clamped to the range 0 to 255 to avoid underflow or overflow, ensuring valid colour values.

### Histogram equalisation:

The histogram equalisation method enhances image contrast using histogram equalization.

(1) For grayscale image equalization

- Histogram Calculation: First, it computes the histogram of the grayscale image, which represents the frequency of each intensity level.

- Cumulative Distribution Function (CDF): Next, it calculates the CDF from the histogram to determine the mapping from old intensity values to new ones.

- Equalization: Each pixel's intensity is then updated based on the CDF, effectively spreading the most frequent intensities across the whole range (0-255), which enhances image contrast.

(2) RGB Image Equalization in HSL/HSV Space

- Colour Space Conversion: For colour images, it converts each pixel from RGB to HSL (Hue, Saturation, Lightness) or HSV (Hue, Saturation, Value), depending on the method specified.

- Histogram and CDF for L/V Channel: It calculates the histogram and CDF, but only for the Lightness (L) channel in HSL or the Value (V) channel in HSV, as these channels represent the brightness of the image.

- Equalization: The L or V channel of each pixel is equalized based on the CDF. This step adjusts the brightness while keeping the hue and saturation (colour information) unchanged.

- Conversion Back to RGB: The modified HSL/HSV is then converted back to RGB for each pixel.

### Thresholding:

Thresholding method implements image thresholding, a technique to convert an image into a binary (black and white) image based on a threshold value.

1. For Grayscale Thresholding:

* Processes each pixel individually.
* Pixels with intensity above the threshold become white (255), others become black (0).

(2) Colour Thresholding in HSV/HSL

- Converts RGB pixels to HSV (Hue, Saturation, Value) or HSL (Hue, Saturation, Lightness).

- Applies thresholding to the V (Value) or L (Lightness) channel. If V or L is above the threshold, the pixel is set to white (V or L = 1.0, saturation set to 0 for pure colour). If V or L is below the threshold, the pixel is set to black (V or L = 0, saturation set to 0).

- Converts the modified HSV/HSL back to RGB.

### 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.

### 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 neighborhood.

### 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.

### 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.

### Edge detection filters:

Edge detection filters are used to identify the boundaries within images by detecting discontinuities in brightness. These filters highlight the edges in images by emphasizing areas with high intensity contrasts.

For Sobel Filter, two 3x3 matrices (kernels) for horizontal (`gx`) and vertical (`gy`) gradient calculations were used. The gradient magnitude at each pixel, combining horizontal and vertical derivatives to highlight edges was calculated. It is good at detecting edges and their orientations in images, often used for edge detection in image processing tasks.

For Prewitt Filter, similar to Sobel, 3x3 matrices was used but with different coefficients, emphasizing changes in intensity. The Prewitt operator was applied to estimate the magnitude of the spatial gradient, identifying areas where the image brightness changes sharply. It provides a simpler approximation of the gradient, often used as a standard edge detection method.

Scharr Filter, 3x3 matrices were used with coefficients that offer a better approximation of the gradient. The edge strength using the Scharr operator was computed, known for better handling of rotation in images. It is Known for its accuracy in edge orientation, providing improved edge detection performance, especially in terms of rotation invariance.

For Roberts Cross Filter, it employs 2x2 matrices, focusing on diagonal differences in pixel values. It detects edges by responding maximally to edges running at 45-degree to the pixel grid, highlighting diagonal features. It is used in applications requiring fine detection of high-frequency components, typically in detailed or noisy images.

1. Grayscale Conversion: Initially, the image is converted to grayscale to simplify the analysis of intensity changes.
2. Gradient Calculation: The filter kernels (`gx`, `gy`) are applied to compute the gradients, which represent the change in intensity at each pixel.
3. Edge Magnitude: The magnitude of the gradient is calculated, often using the Euclidean norm, to determine the strength of the edge at each pixel.
4. Normalization: The resulting edge magnitude image is normalized to fit within the display range (0-255), highlighting the edges effectively.

## 3D Data Volume

### 3D Gaussian Blur:

The 3D Gaussian Blur applies a smoothing filter to 3D volumetric data (like medical images or 3D textures), using a Gaussian kernel.

(1) Kernel Creation: A 3D Gaussian kernel is generated, which is a cube of numbers representing the blur effect. The kernel size and standard deviation (sigma) determine the blur amount.

(2) Applying the Kernel: The algorithm iterates over every voxel (3D pixel) in the volume. For each voxel, it calculates a weighted average of itself and its neighbours, using the values from the Gaussian kernel.

(3) Normalization: The Gaussian values are normalized so their sum equals one, ensuring the total brightness of the image remains constant.

(4) Edge Handling: The algorithm uses boundary checks to handle edges correctly, typically clamping positions outside the volume to the nearest valid voxel.

(5) Result Storage: The computed value from the Gaussian kernel application replaces the original voxel value, smoothing the data.

### 3D Median Blur:

The 3D Median Blur filter removes noise from 3D volumetric data while preserving edges:

(1) Neighbourhood Analysis: For each voxel, the algorithm considers a block of neighbouring voxels defined by the kernel size, creating a neighbourhood.

(2) Median Calculation: It sorts the intensity values of these neighbouring voxels and finds the median value.

(3) Assignment: The median value is then set as the new value of the central voxel.

(4) Iteration: This process is repeated for each voxel in the volume.

(5) Edge Handling: For voxels near the edges, the algorithm clamps the neighbourhood to the available voxels, ensuring the kernel remains within the volume boundaries.

### Maximum intensity projection:

- Purpose: 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.

- Process: 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.

- Result: 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.

### Minimum intensity projection:

- Purpose: 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.

- Process: Similar to MIP but instead tracks the lowest intensity value for each pixel across the selected slices of the volume.

- Result: Each pixel in the resulting 2D image represents the minimum intensity value found along the viewing path, emphasizing the darker regions.

### Average intensity projection:

- Purpose: 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.

- Process: It sums the intensity values for each pixel across the selected slices and divides by the total number of slices to calculate the average.

- Result: The resulting 2D image represents the average intensity encountered for each pixel position, giving a more balanced view of the internal structures.

### Slicing:

Slicing in the volumetric data (like a 3D medical scan) refers to extracting a 2D image from the 3D volume.

Slicing in the context of volumetric data (like a 3D medical scan) refers to extracting a 2D image from the 3D volume. Here’s how the slicing algorithm works in the provided code:

(1) Determine Plane:

- The slice can be from different planes: `xz`, `yz`, etc., depending on the desired cross-section.

- The `plane` parameter specifies the slicing direction.

(2) Extract Slice:

- For `xz` plane: Iterates through the `depth` and `width` of the volume, keeping the `y` coordinate constant (as specified by `index`), to extract a slice.

- For `yz` plane: Iterates through the `depth` and `height`, fixing the `x` coordinate, to obtain the slice.

(3) Build Slice Data:

- Constructs a 2D array (`sliceData`) representing the extracted slice by selecting the appropriate voxels from the 3D volume.

(4) Specialized Slicing Methods

- For Thin Slab AIP (Average Intensity Projection), it averages the voxels between `startSlice` and `endSlice` along the z-axis to create a slice with averaged intensity values. It sums the intensity values of the voxels in the specified range for each `x` and `y` position, then divides by the number of slices to compute the average.

- For Thin Slab MIP (Maximum Intensity Projection), it Finds the maximum intensity voxel between `startSlice` and `endSlice` for each pixel location in the slice. It iterates through each `x` and `y` position, checking all `z` values in the range to find the highest intensity value, which is then used in the final slice.

# Performance Evaluation

Tienshan and stinkbug images are used to compare the performance evaluation for 2D filters.

## Image 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.

## Volume size

## Kernel size

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.

# Potential Improvements/Changes

**Code Profiling and Refactoring:** Profiling the code to identify bottlenecks and optimize them could lead to improvements. For instance, loops over pixels and colour channels might be refactored for better cache coherence or to leverage SIMD (Single Instruction, Multiple Data) instructions.

**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.