Image processing project

The aim of this project is to implement various image processing techniques, including 2D and 3D filters, projections, and slicing. In this report, we will introduce the implementation of these image processing algorithms, as well as their optimisation and performance evaluation. Additionally, we will discuss the advantages and disadvantages of our approach, along with future directions and potential improvements. Through this report, we aim to showcase the capabilities of our C++ project and demonstrate its potential for image processing applications.

Overview

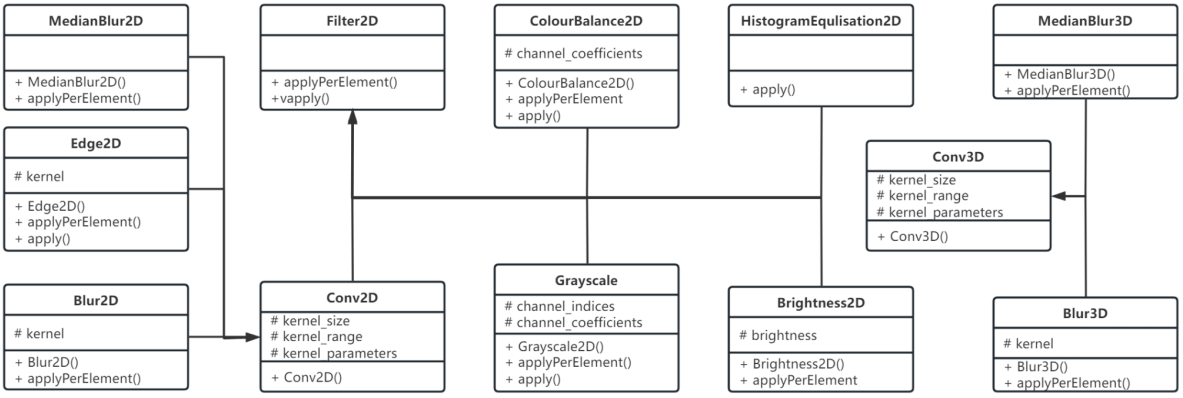

Figure 1: The inheritance of our implemented algorithm

We implemented Filter 2D and Conv3D classes for filters. As for Filter2D, we have two virtual functions: apply every pixel and apply the filter on the image. These two virtual functions were overridden depending on different algorithms introduced below. Some filters are convolutional, which need additional parameters and methods. Therefore, Conv2D was implemented to handle blur and edge detection convolutional filters, as was Conv3D. Conv3D works in conjunction with the Volume, Image, Projection, and Slice classes to produce the desired output. To avoid duplicates of our code, common parts of our code were not reused over and over again.

Colour correction filters

The grayscale algorithm converts a colour image into grayscale by calculating the weighted average of the red, green, and blue components of each pixel. In our implementation, we use the formula: $Y = 0.2126R + 0.7152G + 0.0722B$. The calculated luminance value replaces the original RGB values of each pixel, resulting in a grayscale image. This approach is computationally efficient and can be used in various applications such as image analysis and edge detection.

Our colour balance algorithm adjusts the intensity of colours in an image by equalising the average values of the red, green, and blue channels. The algorithm calculates the average value for each channel and applies a scaling factor to balance the colour components. This method corrects colour casts and imbalances in images, resulting in a more natural and aesthetically pleasing appearance.

Brightness refers to the overall lightness or darkness of an image and is a key aspect of human visual perception. In image processing, the Brightness function is implemented by adjusting the intensity of each pixel in the image. Our function achieves this by adding a specified value to all pixels, either by adding a desired value of brightness to each pixel in all channels or adding a value to all pixels to make the average of values equal to 128.

Histogram equalisation is a technique used in image processing to enhance the contrast of an image by redistributing the intensity levels of the pixels. The goal is to create a uniform distribution of pixel

intensities, making the image's overall appearance more balanced in terms of brightness and contrast. Our function accomplishes this by calculating and storing the cumulative distribution function of all pixel values in each channel and then assigning each pixel value by multiplying the corresponding probability in CDF. This results in an image with a uniform distribution of pixel values.

Define y as the new pixel value, x as the original pixel value, L as the total number of pixel value, $p_X(x)$ as the PDF of x, and $p_Y(y)$ as the PDF of y. Convert the pixel value by

$y = f(x) = (L-1)\int_0^x p_X(t)dt$. Then the PDF of y,

$p_Y(y) = p_X(x)|f^{-1}(y)| = p_X(x)\frac{1}{(L-1)p_X(x)} = \frac{1}{(L-1)}$ is uniform.

Convolution - Image Blur and Edge detection

Blur is a fundamental image processing technique that serves to reduce noise, smooth out textures, and create a more visually appealing result. Blurring an image involves averaging the pixel values in the neighbourhood of each pixel to produce a new, smoother image. This is accomplished by convolving the image with a kernel or matrix, which is typically a small grid of numerical values. In our design, users can choose to use box, gaussian, and median blur, each with their own specific characteristics and applications.

Box blur uses a uniform kernel with equal weights for all pixels within the neighbourhood. This filter is computationally inexpensive and easy to implement, making it suitable for real-time applications. However, the box blur tends to produce less smooth transitions between regions and can leave visible artefacts, such as "boxy" shapes, in the output image.

The median blur and Gaussian blur algorithms are widely used image processing techniques that can be applied to both 2D and 3D data. The 2D median blur algorithm replaces each pixel's value with the median value of the surrounding pixels within a specified kernel, making it effective at removing impulsive noise while preserving edges. The 3D median blur extends this approach to volumetric data by considering the spatial neighbourhood in the x and y dimensions as well as the temporal or depth (z) dimension. In our algorithm, 3D blur had one more for loop compared to 2D.

2D Gaussian blur is a widely used blurring technique that utilises a Gaussian kernel, a bell-shaped curve that distributes weights according to the distance from the centre pixel. This results in a smoother and more natural-looking blur compared to box blur. The amount of blur can be controlled by adjusting the kernel size and the standard deviation (sigma) of the Gaussian function.

In 3D, Gaussian blur can be applied to volumetric data (e.g., medical imaging or 3D models) to generate the kernel: $G(x, y, z) = (1 / (2\pi\sigma^2))^{1.5} * exp(-(x^2 + y^2 + z^2) / (2\sigma^2))$

where (x, y, z) are the spatial coordinates, and σ is the standard deviation controlling the extent of the blur. Then normalise the value so that they sum to 1. Finally, the kernel is convolved with the 3D data to get new pixel values.

The **edge detection** algorithm is an image processing technique that identifies and highlights the boundaries between different regions within an image. In our implementation, we use four methods: Sobel, Prewitt, Scharr, and Robert Cross. The Sobel, Prewitt, and Scharr operators approximate the image's gradient by convolving the image with specific kernels, which measure the change in pixel intensity values. Robert Cross operator, on the other hand, uses a simple 2x2 kernel to calculate the gradient along the diagonals of the image. By calculating the gradient magnitude and direction for each pixel, these operators detect edges and outline the image's structure. Edge detection is widely used in applications such as object recognition, image segmentation, and feature extraction.

When the Sobel kernel is selected, a 3x3 kernel with the parameters: {1, 0, -1, 2, 0, -2, 1, 0, -1}. This kernel is used to detect horizontal edges in the image. The corresponding vertical kernel would be the transpose of this matrix. As for the Prewitt kernel, a slightly different 3x3 kernel is set: {1, 0, -1, 1, 0, -1, 1, 0, -1}. The Scharr kernel uses an optimised 3x3 kernel for more accurate gradient estimation: {3, 0, -3, 10, 0, -1, 0, 3, 0, -3}. In contrast, Roberts' Cross uses a pair of 2x2 kernels. The given kernel parameters are {1, 0, 0, -1} and {0, 1, -1, 0}, which compute diagonal gradient magnitudes.

Projection- Slicing

The Projection class plays a crucial role in generating 2D projections from 3D data stored in a Volume. The primary method in the Projection class is take, which accepts a Volume, an Intensity enumeration value, and an optional Axis. The Intensity enumeration value determines the type of projection to be performed (min, max, mean, or median). The Axis parameter defines the direction of the projection (elevation, azimuth, or range). The take method iterates through the 3D data and computes the projection for each 2D plane along the specified axis, resulting in a new Image.

The Slice class is designed to extract 2D slices from 3D data stored in a Volume. The primary method in this class is take, which accepts a Volume, an Axis, and an index representing the position of the slice along the specified axis. The take method extracts the 2D slice from the 3D data and returns it as an Image. This functionality is useful for visualising specific planes within a Volume or applying 2D processing techniques to a particular cross-section of the data.

Performance of library

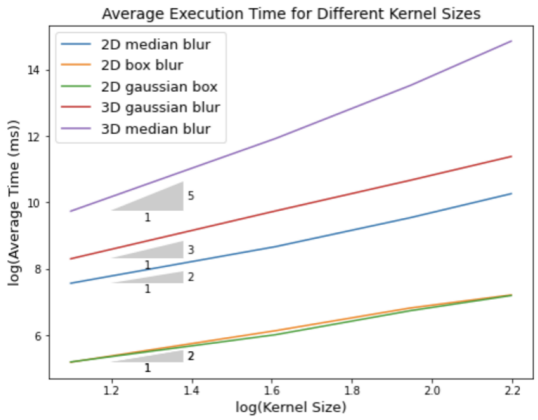

Image 2: Log-log plot lot of average time taken against kernel size for blur filters.

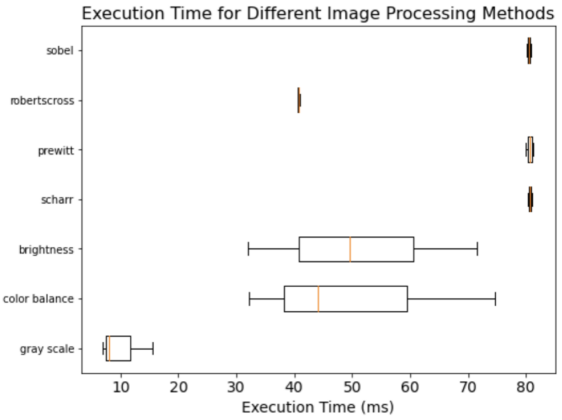

Image 3: Box plots of min, max, mean with other methods

To improve efficiency and reusability, we wrote a function to test our performance. Each test with different 2D filter sizes was run for 10 times, and the average time taken was recorded.

According to the plot above, it is clear that as the kernel size increases, the time taken increases with $O(n^2)$ for 2D filters and $O(n^3)$ for 3D filters, mainly due to larger memory footprint and increased loop iterations. (Capobianco et al., 2021) Kernel sizes of 3x3 and 5x5 are commonly used, because they offer a good balance between quality of output and computational/memory efficiency, making them suitable for a wide range of image processing tasks.

In addition, median blur takes more time than other filters because it involves sorting the pixel values. In our algorithm, quick select was implemented to find the median as quickly as possible, but it may be slower compared to the built-in nth-element algorithm in C++.

Also, Median blur is a non-linear filtering technique, whereas box blur and Gaussian blur are linear filters. (Ohki et al., 1995).

From image 2, it is clear that edge detection takes generally longer time than colour correction but with less variation in time.

The reason is that colour correction filters are pointwise operations hence less computationally intensive than the convolution and gradient calculations required for edge detection. The smaller time range for edge detection filters compared to colour correction filters might be due to the consistency in the computational complexity of edge detection algorithms.

Discussion

Advantages of the current approach:

1. Modularity: The class hierarchy allows for a clear separation of functionality, making it easy to extend the library with new filters or operations.
2. Reusability: Our code is organised into classes with well-defined responsibilities, making it easier to reuse components for various image processing tasks. A basic pipeline is built.
3. Flexibility: The use of templates enables the library to work with different data types, allowing for more precise or efficient storage and processing.
4. Consistency: The Image and Volume classes provide a consistent interface via virtual functions for working with 2D and 3D data, simplifying the interaction between different components.

Disadvantages of the current approach:

1. Complexity: The inheritance hierarchy will become complicated if whole image filters (non-pointwise) are implemented.
2. Performance: If the call stack gets too deep because of nested functions, execution time may increase. This may be alleviated if the compiler inlines some of the functions during optimization.

Future development:

1. Inheritance hierarchy: The current inheritance hierarchy is designed to provide modularity and reusability, but it may become complex as more filters and operations are added. A more flexible composition-based design, such as using an advanced pipeline (API) or filter graph approach, might be considered for future iterations.
2. The current implementation focuses primarily on 2D image processing, with the Projection and Slice classes providing basic 3D functionality. More advanced 3D processing operations can be included in the future design.
3. User-friendly interfaces: Developing user-friendly interfaces, such as graphical user interfaces (GUIs) or command-line tools, will make it easier for users to interact with the library and apply filters without deep programming knowledge.
4. Support for additional file formats: Expanding the library's support for various images.
5. Enhancing compatibility with popular image processing and machine learning libraries to help users to leverage existing tools and knowledge while using the new library.

Conclusion

In conclusion, our image processing project has effectively implemented a variety of techniques for 2D and 3D images. Although the current approach has its advantages, some limitations, such as complexity in the inheritance hierarchy and potential performance issues, need to be addressed. Future iterations could investigate more flexible composition-based designs or pipeline approaches to better manage the library's complexity. Additionally, the inclusion of more advanced 3D processing operations would expand the library's capabilities and increase its applicability across a broader range of applications. Overall, this project demonstrates a robust foundation for image processing, emphasising the significance of designing and implementing versatile and efficient algorithms to cater to the diverse requirements of the image processing field.

References

1. Capobianco, G., Cerrone, C., Di Placido, A., Durand, D., Pavone, L. M., Russo, D., & Sebastiano, F. (2021). Image convolution: a linear programming approach for filters design. *Soft Computing*, *25*(14), 8941–8956. https://doi.org/10.1007/s00500-021-05783-5

2. Ohki, M., Zervakis, M., & Venetsanopoulos, A. N. (1995). 3-D Digital Filters. *Control and Dynamic Systems*, 49–88. https://doi.org/10.1016/s0090-5267(05)80038-6