Advanced Image Processing Application with C++ Advanced Programming

Chenyu Ren, Gustavo Machado, Vidushee Geetam, Yifan Wang, Yulin Zhuo

24th March 2023

Introduction

Image processing is crucial in various fields, including computer vision, graphics, and medical imaging. 2D images and 3D data volumes can be considerably improved in visual quality and better for analysis by using filters and orthographic projections. In this project, our group used several filters, including grayscale, automatic colour balance, brightness adjustment and convolutional filters. For reading and writing image files, we used the libraries like 'stb_image' and 'stb_image_write'. For other tasks, we used the standard libraries. In addition, we developed our program using C++17 or later, ensuring compatibility with modern programming. The report will cover the details of algorithms and implementation specifics for each filter and projection, as well as a performance study of the programme and a discussion of the advantages and disadvantages of our strategy.

Work Breakdown

1. **2D Image Filters Development:** Vidushee Geetam, Yifan Wang, Yulin Zhuo, Chenyu Ren, Gustavo Machado
2. **3D Volume Projection & Slicing Development**: Vidushee Geetam, Yifan Wang, Gustavo Machado
3. **Testing Unit Development and Integration**: Chenyu Ren, Yulin Zhuo, Vidushee Geetam, Gustavo Machado
4. **User Interface Development**: Vidushee Geetam, Yulin Zhuo
5. **Project Closure (Integration and optimisation)**: Vidushee Geetam, Yifan Wang, Yulin Zhuo

During the project implementation, team members actively and positively collaborate to ensure that the project proceeds smoothly and effectively.

Methods

1. **Grayscale Filters** (Gustavo Machado):

The grayscale filter compresses the colour channels in RGB images to a single channel so that the pixels of each channel are reduced to a single value. To prevent the image from assuming a black hue, an effect that is observed using the average, a different weighting per channel was adopted to obtain a more reliable greyish colouration. For this, Red contributes 30%, Green contributes 59%, which is more significant in all three colours, and Blue contributes 11%.

2. Automatic colour balance Filter (Vidushee Geetam):

This filter is meant to balance all the channels (RGB) of an image such that in the resulting image the average intensity of each channel is equal. The function first calculates the average intensities for each channel then calculates the overall average intensity by taking the mean of individual averages. It then calculates the scaling factors for each colour channel by dividing the overall average intensity by individual channels' average intensities. Then, it iterates over all pixels again and multiplies them with the respective scaling factors and saves the altered pixel data in the new unsigned char pointer allocated earlier in the code. Finally, the code use input image's dimensions and the altered pixel data to return a new image.

3. Brightness Adjustment Filter (Yulin Zhuo):

The brightness filter aimed at adjusting the overall brightness of image to enhance visual performance. The filter is based on linear scaling method by multiplying each pixel's RGB value by a fixed coefficient (e.g. adding the pixel by 1 means to increase brightness). If the user wants to achieve automatic brightness adjustment, the filter can perform it by setting the average value of all pixels in all channels to 128. The filter can be used to enhance image visibility, particularly in low-light situations.

4. Histogram Equalisation Filter (Vidushee Geetam)**:**

This function first checks for the image's number of channels, and then either a greyscale or colour histogram equalization is performed accordingly. In the case of greyscale images, it first calculates a histogram of the image's intensity values then calculates the image's cumulative distribution function (CDF) of the histogram, and uses it to modify the intensity values of each pixel in the image to enhance contrast, and returns the modified image. For colour images, the function first converts the image to YCbCr colour space, as it separates the luminance component from the color components. It calculates a histogram and a CDF of the Y channel(luminance values), then uses these two to alter Y channel data. The modified YCbCr values are then converted back to RGB values, and the resulting image is returned by the function.

5. **Convolution Filter** (Yifan Wang):

The Convolution Filter is an effective method for processing images that can be used for edge detection, and also blurring. It computes the output image (or volume) by applying a kernel to the input image or volume. Our implementation supports doing both 2D and 3D convolutions by a single '**conv_3d**' function (where the '**conv_2d**' function converts the 2D kernel into a 3D kernel, then call the '**conv_3d**' function to do the convolution.) Iterating through the input volume and applying the kernel are the main steps in the convolution process. Instead of padding, our approach performs a boundary value detection to handle boundary conditions without producing an after-padding array, which is a faster and more memory-saved method. If the input has an alpha channel, it will be copied to the output directly. This implementation allows us to perform various image processing using a single convolution function, simplifying the process, and ensuring consistency in applying different filters.

6. Image blur Filter

There are three different image blur filters available in our programme, which are Median blur, Box blur and Gaussian blur. They are all implemented using our flexible convolution functions. These methods apply different blur filters to the image by utilising the separate kernels. The details are shown below:

a) **Median Blur** (Chenyu Ren):

The median blur filter is a technique that replaces each pixel value with the median value of its neighbouring pixels within a specified kernel. The process is to extract the values of neighbouring pixels within the kernel region for each pixel in the input image. Sort the extracted pixel values in ascending order. Finally return the median pixel value — in the case of an even-number sorting, the output is the average of the central two samples after sorting.

b) Box blur filter (Yifan Wang & Yulin Zhuo):

The box blur filter smooths an image by replacing each pixel value with the average of neighbouring pixel values within a specified kernel size (kernel size can be changed by users). The kernel is calculated by using the formula $\frac{1}{size\ of\ kernel\ (e.g\ 5*5)}$. We are then applying the kernel to the image by performing a convolution operation. Finally, the new pixel value is calculated by summing the product of the kernel values and their corresponding pixel values within the kernel region.

c) Gaussian blur filter (Yifan Wang & Yulin Zhuo):

Gaussian blur filter uses a Gaussian kernel to smooth an image. This makes the image less noisy and less detailed. 2D and 3D Gaussian blur filters allow users to give a standard deviation and a kernel size to produce the image after blurring. The Gaussian blur kernel is calculated by using the formula, $\frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{x^2}{2\sigma^2}\right)$, where x is the distance from the current pixel to the central pixel, and then normalises it. Finally, apply the gaussian kernel with the convolution filter to blur the image. The new pixel value is calculated by summing the product of the kernel values and their corresponding pixel values within the kernel region.

7. Edge Detection (Yifan Wang & Yulin Zhuo):

In image processing and computer vision, edge detection is used to locate the boundaries of objects in images. The edge detection first converts the images to grayscale images using the 'grayScale' function, simplifying the images, and reducing computational complexity. Next, the grayscale image is blurred using a Gaussian blur, which helps to reduce noise and smooth the image. Then we apply different kernels such as Sobel[1] , Prewitt[1], Scharr[1], and Roberts' Cross[2] to the image by utilising convolution operation. All the operators use two different kernels, which are convolved with the original image to calculate approximations of the gradients – one for horizontal changes and the other for vertical. Then we use both kernels to calculate the gradient horizontally and vertically and get the final gradient magnitude using the formula, $G = \sqrt{(G_x^2 + G_y^2)}$, where G_x and G_y are two images that contain the horizontal and vertical derivative approximations respectively.

8. Orthographic Projection:

The source code contains a class that performs 2D projections of 3D image volumes. It takes projection type, start image and end image as arguments. Start image and end image refer to the slab of 3D volume that is being projected. The function traverses over each pixel in that slab of volume and depending on the projection type either takes the minimum, maximum or average intensity of the pixel values across the images in the volume slice. The projected data is then returned as a new Image object. Overall, this method enables the generation of projections from a 3D volume, which can be useful for medical imaging, scientific visualization, and other applications where it is necessary to visualize a 3D volume as a 2D image.

9. Slice:

The Slice class in the source code allows for the creation of 2D slices from a 3D image volume. This method takes a 'Volume' object, the desired plane (either XZ or YZ), and the position of the slice along the chosen plane as arguments. The function first checks if the user input is valid and then retrieves the necessary information about the volume such as its dimensions and total number of images. It then performs the slicing operation by iterating over each pixel in the volume at the slice position (position is the method argument) and copying the corresponding pixel values from the 3D volume. The resulting data is stored in a new unsigned char array and returned as an Image object. The slice method can be useful in a variety of applications such as medical imaging, scientific visualisation, and computer graphics, where it is necessary to extract and analyse a 2D section of a 3D volume.

Implementation

The implementation of our image filtering and projection program is designed to be modular, efficient, and memory-saved, employing well-structured classes and functions. The following are the key components of the implementation included:

1) **Main Function**: User interface.
2) **Classes:** A few classes are created to support different operations, such as:
   a) **Image:** This class is designed to operate 2D image data. It provides a convenient interface for various image operations, such as loading and saving. This class acts as a foundation for other classes like **Filter** and **Projection** for image processing tasks.
   b) **Volume:** Similar to 'Image' class, which is designed to manipulate 3D data volume.
   c) **Filter**: It provides numerous filtering methods for both 2D and 3D data, such as grayscale, brightness adjustment, box blur, Gaussian blur, and edge detection.
   d) **Projection**: It is for applying orthographic projections in 3D data volumes, which can operate over the entire data volume or over a thin slab.
   e) **Slice:** It can provide a slice of 3D data volume in different planes.
3) **Convolution Tool:** This tool class provides functions for performing 2D and 3D convolutions, which can handle the boundary value detection, and casting data between double and unsigned char formats as needed. This tool is used by the Filter class methods to apply convolution-based filters such as box blur, Gaussian blur, and edge detection.
4) **Kernels Tool:** This tool class generates different types of kernels, such as box and Gaussian kernels. These kernels are used by the Filter class methods to apply the corresponding filters.
5) **Test:** The class is designed to test the functionality for every function involved.

Results

This project successfully implemented all 2D image filters, with 3D volume projection and slicing. There were no error conditions during the test. The 2D image filter test is performed by passing in an array of pixel values of an image, processing the image by transforming it and calling a filter function, and then obtaining the image data (the array of pixel values) by a specific function and comparing it with an accurately calculated array of pixel values. The example shown below (Figure 1) is the output image after applying Roberts' Cross operator.

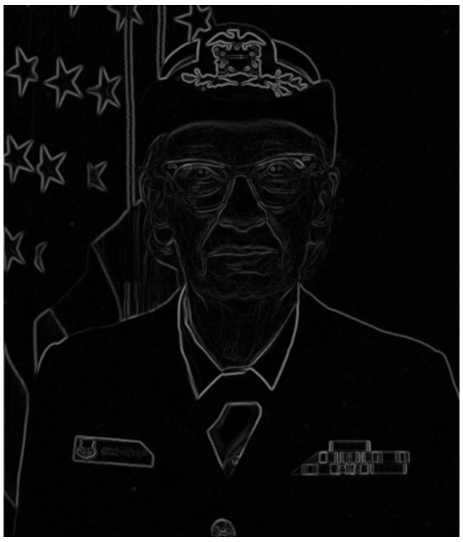
Figure 1*: Gracehopper after Robert's Cross*

The 3D volumetric projection test compares the length, width, and number of channels of the resulting image by comparing the different projections of the scanned image; as the projection contains three types and affects the image pixel values, the pixels of the resulting image are also judged. For the slicing test, the slicing function is tested in two directions; after the slicing operation, determining whether the sliced length, width and the number of channels are equal to specific values, which are obtained by calculation.

However, our program still has some problems with applying projections and sliced images on blurred volume objects. We would generate some images with corrupted dataset. We checked the functionality of 3D algorithms, and they can work individually and generate outputs. We checked the saving image functionality, and it can convert the original volume to a set of image files. Hence, we narrow down our problem to memory leaks. Even if we write destructors and delete every object created by "new", the memory is still not fully freed. Then we tried to reduce the memory used in our program. In other words, we keep as few copies as possible. Originally, to ensure the result accuracy, we cast all unsigned char arrays to double arrays before calculation and then cast them back when we generate the result. It increases the memory used during run time. We tried to cast an array to float and remove all the casting (do all calculations based on unsigned char), even though they would boost the speed of our program, but they are still not able to generate result images by applying projections and sliced images on blurred volume objects.

Among all the 2D blurring filters, the median blur gives the most stable result among all 3-blurring metrics. The box blur does not have a strong effect especially when we are using small kernels. The gaussian blur needs a sigma parameter, every image should have their own sigma value based on how complex a picture is.

The 3D gaussian kernel (3*3*3) runs relatively fast which is around 128s on a Mac. The 3D median blur (3*3*3) is slow. The reason is that we use a bubble sort inside the algorithm which slows our program down. In order to improve this in the future, we can implement a binary tree data structure for storing the data unsorted and then use binary search to reduce the time complexity from O(n^2) to O(log(n)).

The result of 3d image after blurring is provided below:
https://drive.google.com/file/d/19VB5SM6FYKk5vbDYsoN7GThcU8t3UN-w/view?usp=sharing

Conclusion

This report presents our C++ program designed to apply different image filters and orthographic projections to 2D images or 3D data volumes. During the project, we considered the trade-offs between clarity and computational complexity. Our programme is designed to be well-structured, efficient, and user-friendly. We made an effort to simplify our program as much as possible without sacrificing performance, as seen in the implementation of the convolution tool. However, there are still some drawbacks to our programme, such as code duplication. The 'for loops' used for median blur and other blur filters are similar and could potentially be combined into a single function, with additional conditions in the 'else' part to differentiate between them. To further improve programme performance, some parallel processing techniques could be applied, such as GPU hardware acceleration. Furthermore, we consider adding more advanced filters and image processing methods to cover a wider range of user requirements.

References

[1] **pintusaini** *Edge detection using Prewitt, Scharr and Sobel Operator,* 2022. Available here: https://www.geeksforgeeks.org/edge-detection-using-prewitt-scharr-and-sobel-operator.

[2] **Roberts** *Machine Perception of 3-D Solids, Optical and Electro-optical Information Processing*, 1965