Group Kruskal Report

1 Introduction

The purpose of this project is to design and implement a C++ program that applies a range of filters and orthographic projections to input 2D images or 3D data volumes. The program includes eight 2D image filters, two 3D data volume filters, and 3D image projections and slices.

2 The implementation of filters, projections and slices

2.1 2D Image Filters:

Grayscale Filter: This filter converts a color image into grayscale, resulting in a black and white image. This process is started by the user interacting with the console application to load in an image using STB_LOAD. The image properties, such as width, height, and channels are checked. A check is implemented that will handle images of RGBA, which is an important consideration when handling PNG images. We then iterate over every pixel and compute the mean for the channels within the pixel. This results in a final pixel with only 1 or 2 channels, therefore generating a grayscale image.

Median blur filter: The 2D median blur is performed by sliding a square window of a predetermined size over the image, and replacing each pixel's value in the window with the median value of the pixels inside the window. The median value is calculated by sorting the pixel values in the window from lowest to highest and selecting the middle value. This process is repeated for each pixel in the image.

Box blur filter: The 2D box blur is performed by sliding a square kernel over the image, and replacing each pixel's value in the window with the average value of the pixel values inside the kernel. The kernel used in the box blur has equal weights, which means that all pixels inside the kernel contribute equally to the new pixel value.

Gaussian blur filter: The function first calculates the kernel values based on the kernel size and sigma using the Gaussian function. The kernel is then normalized to ensure that its values sum up to 1.0. Next, the function applies the kernel to each pixel value within the kernel for each channel in the image. The resulting convolved value is then set as the pixel value at the center of the kernel.
Prewitt edge detection: The code is designed into three parts: gradient of X direction, gradient of Y direction, computing output pixels using the square root of the sum of X direction and Y direction. Each part has two loops to calculate each pixel in the picture. The values of output pixels are in the range 0-255. Before this can be computed, the image is converted to grayscale.

Sobel edge detection: It determines the gradient magnitude of each pixel by utilizing a pair of 3x3 convolution kernels - one for X direction and one for Y. The code executes the convolutions in X, followed by Y, and then computes the magnitude by taking the absolute value of X and Y convolutions. The final output pixel is capped at a value of 255 since that is the maximum value that can be represented by an 8-bit integer. Before this can be computed, the image is converted to grayscale.

2.2 3D Image Filters:

3D Gaussian blur: The function first calculates the values of the 3D Gaussian kernel using the specified kernel size within the kernel. It uses a nested loop to iterate through all pixels in the image and convolve them with the kernel. The resulting pixel value at the center of the kernel is then set as the convolved value for the current channel.
3D Median blur: The implementation of the function uses quick sort to iterate through each pixel of the image and a kernel centered at each pixel to compute the median value. The kernel is a 3D cubic volume centered on the current pixel. If a pixel value is not available in the image for a given kernel position, it is not considered in the median computation. Then the pixel value at the center of the kernel is set to the median value for the current channel.

2.3 3D Image Projections:

Our code implements several image processing operations on a volume of images using the Projection class. The code supports four comparative functions: max, min, mean, and median. The IP method takes a compare function, output name, min and max z values, and an optional test flag. The code reads a list of images from a provided volume and selects a subset of images within the specified z range. It then performs the selected operation on the subset of images and writes the result to an output file. The code also includes several helper functions, including my_max and my_min, used for computing the maximum and minimum of two input values, respectively.

2.4 3D Image Slices:
The class ThreeDee contains the methods for computing the XZ and YZ slices. These methods are designed to take a 3D volume of images, with a given X or Y dimension, that allows the user to slice at a chosen X or Y dimension. There are two loops to get the pixels in a specific row or column of each image in the volume. New memory is allocated to store the slice, and a pointer to this memory is the output of the functions.

2.5 Other Files:
The documentation is generated by Doxygen. The project is under MIT license.

3 The performance of algorithms
As indicated in the provided PDF, we observed that edge detection performed better on images that were box or Gaussian blurred, which can be found in the "Output/observation" folder. This is because edge detection operators are sensitive to image noise when calculating gradient magnitudes, which can result in false edges being detected. By applying blur filters before edge detection and replacing each pixel with either the average value of its neighbors (box) or the weighted average of its neighboring pixels determined by Gaussian (Gaussian), the operators were able to perform more accurately. However, it's important to note that a suitable kernel size should be chosen to avoid losing too much detail in the images.

For 3D blurs, both 3D blurs work well. When dealing with images in src/ fracture, gaussian blur finished in 1 minute but median blur took about 20 minutes. The median blur with kernel size 5 made images vaguer than gaussian blur with kernel size 3 but caused anamorphose. The time consuming of gaussian blur is significantly less than median blur. The time consuming of both gaussian and median blur increases with the increase of kernel size. The time consuming of gaussian blur also increases with the increase of sigma.

For 3D projections, everything works in 30s on a macbook-air under O3-mode.
For 3D slices, the time consuming is affected by the size of the image and the number of images in the volume. The time complexity is width*number of images or height*number of images because there are two loops in the code.

4 The design decision of classes, unit tests, and interface
The Image class has two constructors. The first initializes an Image object with provided data for testing, and the second loads an image from a file path using the stbi_load() function. We also added get_width(), get_height(), and get_channels() methods to access image properties and keep member variables private.

The Filter class contains 8 2D filters and 3D blurs, with the constructor taking an Image object as input and optional kernel size, sigma value (for blurs), and filter depth (for 3D operations). After initializing member variables, functions are accessed without passing arguments (except for brightness, used only once), with image data obtained via image.data from the image or trans_volume. Corrected_img gets initialized with a proper size, and the filtering gets performed according to algorithms for grayscale or RGB images. The save functions store the corrected_img output of each filtering function, either as one image (2d) or within a folder (3d) to a user-defined path, while get_corrected_img is only for testing. separation1 is a helper function for quickSort1 and performs a partition operation on the input vector. quickSort1 implements the QuickSort algorithm to sort the input vector value in ascending order.

The Volume class manages a collection of Image objects. It provides methods to add and retrieve individual images, as well as to load and add multiple images from a folder. The class also contains a destructor to free the memory allocated for the images. The sorting of file paths is performed using a custom comparison function. Overall, the Volume class facilitates the management and organization of images, making it useful for tasks such as volume rendering and medical imaging.

The Projection class is an implementation for generating 2D image projections from a 3D volume dataset. The class provides four image projection methods, including maximum intensity projection

(MaxIP), minimum intensity projection (MinIP), mean intensity projection (MeanIP), and median intensity projection (MedianIP). The class takes a pointer to a volume object as input, which contains the 3D volume data to be projected. The projection methods take additional parameters such as output filenames, z range, and comparison function. The class also includes helper functions for sorting and comparing values. Overall, the Projection class is a versatile and efficient tool for generating 2D image projections from 3D volumetric data.
<u>The 2D filter unit tests</u> are created uniformly for each function. We generate a range of sample input images, such as RGB images, grayscale images, alpha channel images, images of various sizes, and 3D images, based on the filter's requirements. Next, we instantiate a Filter object for each filter and apply the filter using the relevant methods to be tested. We use the get_corrected_img() method to retrieve the output image, which is a private member variable of the Filter object. Then, we compare the output image with the anticipated output, either computed manually or obtained from other libraries. If the difference between the output image and the expected output is less than 1, we consider the test to have passed.

<u>The User Interface:</u> When designing a console application's user interface in C++, it's important to keep several key considerations in mind:

1. Aesthetic Design: The user interface should be designed in a way that allows for clear communication between the user and the console. This can be achieved through ensuring that the layout of the application is clean and that text is formatted in a readable manner.
2. User Input: Careful consideration should be given to how the user interacts with the application. A number-based filter key system can be effective for allowing users to navigate through the system quickly and easily.
3. Error Handling: Effective error handling is crucial for preventing issues such as incorrect file paths and data types. For example, file paths can be managed through checks that ensure files exist, have the correct extensions, and are saved in the correct location. While loops with data type checks can be used to handle incorrect data types.
4. Performance: Console applications should be designed to be fast and efficient. To optimize performance, only the relevant filters necessary for the user's task should be implemented. This can be achieved through using multiple scopes that are triggered based on the user's input.
5. Longevity: It's important to design the user interface in a way that allows for future scalability. Using maps to store methods, for example, can make the addition of new methods straightforward and efficient.

By keeping these considerations in mind, a console application's user interface can be designed to be both effective and efficient.

5 advantages and disadvantages
<u>Class System</u>
Advantage:
Handling our code with in classes that link to source code files has enabled our program to have an extra level of security as without the source code file a user will have restricted access to our program. Additionally, as we utilized classes we were able to keep the majority of our user variables private, meaning that the user has more protection overall. Finally, the modular nature of classes helped keep code clean and organized.
Disadvantage:
Implementing many classes made the code more complex and made the code harder to develop. Additionally, having many classes could increase the amount of memory and processing power required to run the code, which can be a problem on resource-constrained systems. We have tried to circumvent this by paying attention to our scope within the main file.

<u>Volume Class</u>
Advantage:
Having a dedicated class for handling image folders as a volume of images was key to making our program successful. It enabled us to handle lots of images in the correct order and apply the appropriate filters.
Disadvantage:
The volume class meant that we were less flexible with file path names. As the way we read in the files, we would require the user to have a “template” filename, such as “confruc0001” where the final 4 digits signify how the files would be sorted. This means that the program is currently limited

at a max of 10,000 images. A solution to this would be to identify only numeric values that are contained within the path name. It also assumes that the image dimensions and channels of all input images are the same.

User Interface
Advantage:
The UI allows the user to communicate with the program and perform the required operations effectively. There are several fail safes that help the user to correctly specify file paths and folders, however, incorrect entries don't provide much information.
Disadvantage:
The UI lacked to provide the user clarification when needed. If the user enters a wrong path it will return 'Invalid' which may not be enough of an error message to efficiently guide the user. So we could have improved it by giving more info for the users -- for instance, we could have specified that the kernel size is supposed to be an odd number for the blurring filters to work properly.

Unit Test
Advantage:
Unit tests enabled us to continuously develop our software and ensure that the results were always consistent with our known base results. As a group we benefited largely from these tests and helped find errors that were not visible by the output image alone.
Disadvantage:
Our unit tests provided a lot of coverage over our functions, however they could still be more comprehensive.

Projection Class
Advantage:
The use of multi-threading in the median operation helps improve its performance.
Disadvantage:
The test flag implementation in Projection class is not ideal for the unit test mode.

Filter Class
Disadvantage:
For 3d median blur, the sort function we defined (quick sort) can be replaced by a faster sort function.

6 Contribution of Members

Hang Zhao (edsml-hz822) Image class, Filter class, filtering (2d, 3d) unit test, sobel detection
Luwen Liang (edsml-ll2822) Prewitt edge detection, 3D Slice class, documentation, ReadMe file
Elliott Mcquire (acse-ecm22) All 2D Projections, User Interface and 'main' script
Zhuoran Yang (acse-zy622) Volume class design, Projection class design, all 3D projections, projection unit test
Qingyang Lu (acse-ql1522) Median blur(2D,3D), Box blur(2D), Gaussian blur(2D,3D)