Advanced Programming Group Assignment

Image Filters, Projections and Slices

March 2025

1 Core task

Build a `C++` program to apply a range of **image filters**, **slices** and **orthographic projections** to an input 2D image or 3D data volume, using the programming techniques you have learned during the Advanced Programming course.

Your program should contain a `main` function which reads an image or data volume, and based on user input command line flags, determines which filter or projection to apply, calls separate functions for the appropriate filter and/or projections, and saves the new image with the filter/projection applied. Your program should work with any arbitrary image/data volume size.

You can use the existing libraries we have included in the src directory in your GitHub repository to read and write the image (`stb_image.h` and `stb_image_write.h`). A minimal example showing how to include these header files in your code and how to read and write image files is included in the `src/main.cpp` file; you should modify this main function to include the functionality described above.

You may also use C++ standard libraries and headers (e.g., `string`, `iostream`, `cmath`, `vector`). All other classes and functions in your code should be written by your group. No other external libraries may be used. You may use mathematical functions from the `cmath` header, but for any computational algorithms required (e.g. searching, sorting, etc.), you must write your own functions to do this.

There is no need to use `openMP` / `threading`—I will be testing the code performance on a set of standard inputs with `OMP_NUM_THREADS=1`. You are welcome to implement this if it helps your own testing and development, but it will not be considered in the evaluation of your code's performance. Instead, I would recommomend that you focus on writing good code first, and then think about optimisation later.

Code structure: Your code should include a `main.cpp` file containing the main method for your program. You should create classes called `Image`, `Volume`, `Filter`, `Projection`, and `Slice`. Each class should have a `.cpp` and `.h` file associated with it. You may create additional source code and header files for any other classes or helper functionality that you consider appropriate.

Do not copy code from the internet or other sources. You may look up the underlying algorithms of different filters and projections, but you should write your own implementation of each filter and projection in your program.

1.1 2D Image Filters

Your program must include the following **2D filter** types. Each member of the group should try to write at least one image filter to practice writing code.

1. **Colour correction and simple per-pixel modifiers:**
   (a) Greyscale: reduce image from 3 (or 4) RGB(A) channels to 1 grayscale channel, using the definition luminance, Y = 0.2126R + 0.7152G + 0.0722B. If there is an alpha channel in the input image, it should be removed.
   (b) Brightness: may take an optional brightness value in the range -255 – 255 to add to all pixels, or perform an automatic brightness filter by setting the average value of all pixels in all channels to 128.
   (c) Histogram Equalisation: for grayscale, stretch the intensity histogram to fill the range 0-255. For RGB images, equalise the intensity values, and apply this new intensity to all RGB channels. You might find converting to HSV or HSL colour space useful to achieve this.
   (d) Thresholding: all pixels with intensity below a given value are set to black, and above or equal to the threshold are set to white. Can operate either on a grayscale image or on an RGB image by first converting to HSV or HSL space and thresholding the intensity (V) channel.
   (e) Salt and Pepper Noise: add noise to an existing image. User defines the percentage of pixels which should have noise. Random selection of pixels are randomly converted to either 0 (black) or 1 (white).
2. **Image blur** using convolution filters of different kernel sizes
   (a) Box blur: a simple blur filter where each pixel is replaced by the average of its neighbouring pixels. The user should be able to specify the size of the kernel (e.g. 3x3, 5x5, etc.).
   (b) Gaussian blur: a more sophisticated blur filter where each pixel is replaced by a weighted average of its neighbouring pixels. The user should be able to specify the size of the kernel (e.g. 3x3, 5x5, etc.) and the standard deviation of the Gaussian distribution. A default value of 2.0 should be used if the user does not specify a standard deviation.
   (c) Median blur: a blur filter where each pixel is replaced by the median value of its neighbouring pixels. The user should be able to specify the size of the kernel (e.g. 3x3, 5x5, etc.).
3. **Sharpening filter**: uses a convolution with a single 3x3 Laplacian kernel to find edges

$$G = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$I_{\text{sharp}} = I_{\text{original}} + G$$

   By summing the result G with the original image, the result is a "sharpened" image.

4. **Edge detection** using convolution filters. For these filters, your code should first apply the grayscale filter from category 1 first if the image is RGB(A), before applying the edge detection filter. Remember to normalise the output from the convolution.

   (a) Sobel filter: uses 3x3 kernels to detect edges in the horizontal and vertical directions.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

   (b) Prewitt filter: uses 3x3 kernels

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

   (c) Scharr filter: uses 3x3 kernels

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

   (d) Roberts' Cross filter: uses 2x2 kernels

$$G_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$G_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

1.2 3D Data Volume Filters, Projections and Slices

Your program should also be able to read in a 3D data volume (e.g., a medical CT scan or an X-ray CT scan of porous media), optionally apply a **3D filter**, and then perform either an **orthographic projection** or a **slice** of the 3D data volume.

1.2.1 3D Filters

You should implement both of the following **3D filters**:

1. **3D Gaussian blur**: a 3D extension of the 2D Gaussian blur filter. Each voxel in the 3D volume should be replaced by a weighted average of its neighbouring voxels. The user should be able to specify the size of the kernel (e.g. 3x3x3, 5x5x5, etc.) and the standard deviation of the Gaussian distribution. A default value of 2.0 should be used if the user does not specify a standard deviation.
2. **Median blur**: a 3D extension of the 2D median blur filter. Each voxel in the 3D volume should be replaced by the median value of its neighbouring voxels. The user should be able to specify the size of the kernel (e.g. 3x3x3, 5x5x5, etc.).

1.2.2 Orthographic Projections

Your code whould be able to perform the following **orthographic projections**:

1. **Maximum intensity projection (MIP)**: find the maximum intensity of all pixels with a given *(x, y)* coordinate in all images in the z-direction, and output that maximum value on the final image.
2. **Minimum intensity projection (MinIP)**: find the minimum intensity of all pixels with a given *(x, y)* coordinate in all images in the z-direction, and output that minimum value on the final image.
3. **Average intensity projection (AIP)**: find the average (mean) intensity of all pixels with a given *(x, y)* coordinate in all images in the z-direction, and output that average value on the final image.
    - Additional challenge: Also implement the AIP for the *median* intensity of all pixels; note this can be computationally expensive, so you might need to get creative!

The orthographic projections should additionally have the following functionality:

1. To operate over the **entire data volume** (i.e., all images in the z-direction).
2. To operate over a **thin slab** (where the user can specify the minimum and maximum z coordinates—i.e., the first and last images in the scan to use—note that user inputs should use indices starting from 1, not the filename numbering).

1.2.3 Slices

Finally, your code should include the functionality to **slice** the 3D volume in a different plane. For example, if all images are given in the x-y plane, the user should be able to output an image in the x-z or y-z plane (for a given y or x coordinate, respectively, where coordinate indices start from 1 for each dimension in the volume).

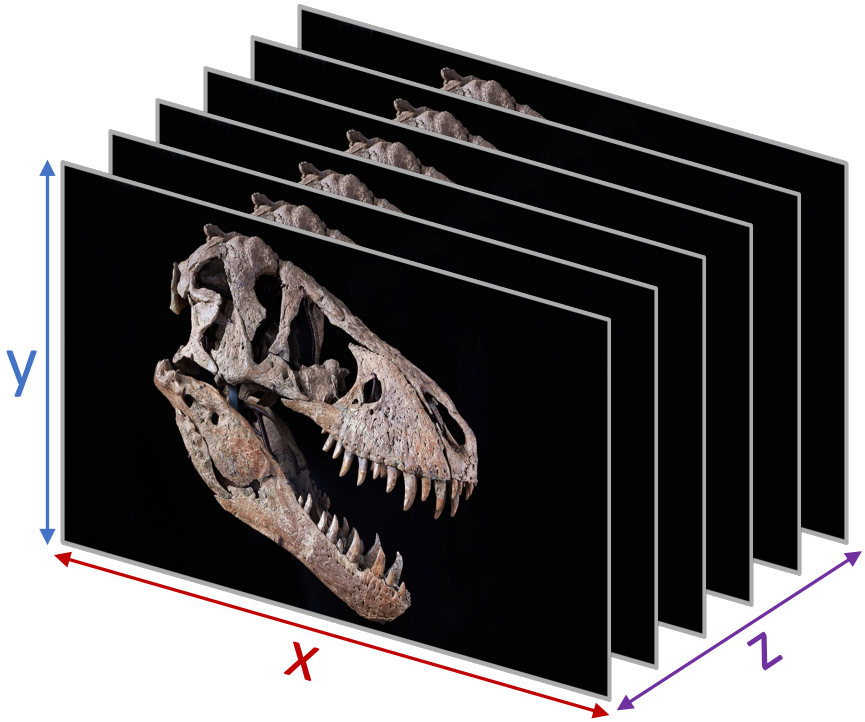

Figure 1: Schematic showing the definition of the x, y, and z coordinates in a 3D data volume.

2 Images and datasets

Example images are available in the Images directory of your repository. CT scan datasets can be downloaded from the link below.

2.1 2D Images:

1. Example images are included in the Images directory to get you started.
2. You are welcome to find and use any other images that you think will demonstrate the capabilities of your program.

2.2 3D Volumes:

Download the example scans here: Scans.zip. This will give you the following datasets to work with:

1. Fossilized Confuciusornis (prehistoric bird) CT scan[1]
   (Unzips into `Scans/confuciusornis`)

[1] `https://doi.org/10.6084/m9.figshare.c.1612235_D59.v1`

2. Fractured granite CT scan[2] (Unzips into `Scans/fracture`)

Note: Do not include the original or any filtered CT scan images in your repository, as they are >100MB each; this will negatively affect your sustainability grade! You should only upload the outputs of the projections and slices.

Note that the `stb_image` headers work with several different image formats, including `.png` and `.jpg`, but not `.tif` or `.dicom`. Therefore, any extra images or CT scans you choose to test your code with in your project may need to be converted to an appropriate format before using them (e.g., `png`).

3 Additional Tasks

As well as the core functionality listed above, your project should follow all sustainability best-practices you have learned in previous modules. For example:

1. Add **comments** to your code (explain in the code what the different blocks of code are supposed to do; remember that good commenting explains why choices were made).
2. Include a **license**, **readme file**, and **documentation** (e.g., using Doxygen, or any other documentation package you are familiar with); make sure to also remove any unneeded files or old code before submission.
3. Include your **group name**, and the **names** and **GitHub usernames** of all members of your group in a header to all submitted source code (`.cpp`) and header (`.h`) files.
4. Remember to **acknowledge and reference** ChatGPT or other generative AI tools where you have used them and include links to any relevant conversations.
5. Add **unit tests** and any other **appropriate testing** (e.g., each function in your code should include an associated unit test). This testing can take any form you consider appropriate.
6. Include **compiled Windows and/or MacOS executables** of your program. If your group only has access to one of those operating systems, you can just upload one executable.

4 Teamwork and collaborative software development

Like many pieces of software, we would recommend you build up the complexity of your methods as you go; start by building some of the simpler colour-correction filters first, the interface/API to your code, adding comments, etc. Remember to add unit tests as you write each function—this is easier than trying to add all the tests at the end! Once you have finished this, you should then attempt to build the more difficult filters (e.g., the convolution filters) and the 3D volume projections/slices. We would also recommend discussing your choices as you go with the teaching team; building an

[2] `https://doi.org/10.17612/P7QX1X`

easy-to-use library is a difficult process, complete with many difficult design decisions to be made; the teaching team are available to answer questions throughout this assignment.

A final comment regarding teamwork: remember that the main aim of this project is learning to write `C++` code collaboratively. Some of you may have experience before the project, while for others this may be completely new. So that everyone has a chance to develop their `C++` skills, we would encourage you all to spend some time writing code (remember everyone should write at least one image filter), rather than leaving the code to just one or two members who have prior experience. You may choose to use paired programming here to make sure everyone is involved in developing your program. If you do use paired programming, make sure each pair writes at least two filters, and remember to add both your names to those filters in the docstrings/report!

4.1 Peer evaluation

After the project, we will conduct a peer evaluation exercise, in the same format as the ACDS module. This will allow you to give feedback on your group members' contributions to the project, and protext against the case where one or two members do not contribute sufficiently to the project. Please let the teaching team know ASAP if a group member is not in contact with the group or not working at the expected level so we can attempt to rectify the situation early.

4.2 Use of GitHub

You should be using GitHub for version control and for working collaboratively. Make sure you use pull requests to merge changes into the main branch, to help you avoid any conflicts. Also try to make use of issues, the project tools and GitHub Actions to help build the best tool you can.

4.3 Use of generative AI tools

As in the individual coursework, we encourage the use of generative AI tools, as long as they are used responsibly. Recall that this means you should be testing/verifying all code produced by an AI tool to make sure it is doing what you think it is doing. You should include a reference or acknowledgement for any code or text produced by an AI tool.

5 Evaluation

Your code will be compiled and executed as part of your evaluation. You should ensure that at a minimum, it can compile using the `gcc` compilers—there is a GitHub actions workflow which compilers the code, so make sure this Action is passing in your final submission.

Code that does not compile, or that does not output an image with the appropriate filters/projection/slice applied will not score highly.

We will make your projects based on:

1. **Functionality and performance** (40%): Does your code work? Does it apply the filters and projections correctly? Does it produce the expected output?
2. **Implementation** (20%): Is your code well-structured? Have you included good object-oriented practices (e.g. classes, inheritance, polymorphism)?
3. **Sustainability** (15%): Have you followed best practices for sustainability? Have you included tests, comments, a license, a readme file, and documentation?
4. **Report** (15%): Include a 4-page report to discuss your implementation (see below)
5. **Peer evaluation** (10%): Based on the peer evaluation exercise

5.1 Report

Your report should be a maximum of 4 pages of A4 (single spaced, 11pt font, 2.5cm margins, PDF format). It should include:

- A brief summary of who worked on which tasks
- A discussion of the design of your code, including the classes you have used and how they interact; you should also discuss the design decisions you made and why you made them, and mention how you have implemented inheritance and polymorphism in your code.
- A discussion of the performance of your library—in particular, you might carry out a performance analysis of some of the different operations your code performs, and discuss how you have optimised your code.
- Any other interesting features of your code, or novel ways you have implemented a given filter or projection algorithm.

6 Submission

Commit and push your code to the main branch of your GitHub repository by **16:00 on Thursday, 20th March 2025**.

Your repository should contain:

1. Your `C++` source code and header files.
2. Documentation and readme, showing how to install and use your program.
3. Your code testing framework.
4. An appropriate license file.
5. Your 4-page PDF report.

If you have any questions, please email thomas.davison@imperial.ac.uk, or ask me in person or on Teams.