Advanced Programming Group Assignment

Image Filters, Projections and Slices

March 2023

1. Core task:

Build a `C++` program to apply a range of image filters 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, asks the user 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/minimal.cpp` file. 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.

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 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 **at least six** of the following **2D filter** types, and **at least two** from each category. Each member of the group should write at least one image filter.

1) **Colour correction** (simple per-pixel modifiers):
    a. Grayscale (reduce image from 3 RGB channels to 1 grayscale channel)
    b. Automatic colour balance (adjust each RGB channel so the average intensity of each channel is equal)
    c. 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)
    d. Histogram equalization

2) **Image blur** using convolution filters (of different kernel sizes):
    a. Median blur (replace each pixel value with the median of those around it), arbitrary kernel size (3x3, 5x5, 7x7, etc)
    b. Box blur, arbitrary kernel size (3x3, 5x5, 7x7, etc)
    c. Gaussian blur (5x5, 7x7 kernels)

3) **Edge detection** convolution filters. For these filters, you should first apply the grayscale filter from category 1 first, before applying the edge detection filter. You may find that edge detection filters work better on images which have been blurred with a box or gaussian filter first; if so, document that in your report. You program could either output an image after each filter is applied or allow the user to specify multiple filters to use sequentially.
    a. Sobel (3x3 operators)
    b. Prewitt (3x3 operators)
    c. Scharr (3x3 operators)
    d. Roberts' Cross (2x2 operators)

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 an **orthographic projection**.

You should implement both of the following 3D filters:

1) 3D Gaussian (to replace the centre voxel with the weighted average of all the voxel intensities in the neighbourhood), for an arbitrary filter size (e.g., 3x3x3, 5x5x5).
2) 3D Median (similar to the 2D median filter above, but for an arbitrary 3D kernel size, e.g., 3x3x3, 5x5x5).

The orthographic projections to implement are:

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) — like the MIP, but for the minimum intensity at each (*x, y*) coordinate.
3) Average intensity projection (AIP) — like the MIP, but for the average (mean or median) intensity of all pixels with a given (*x, y*) location.

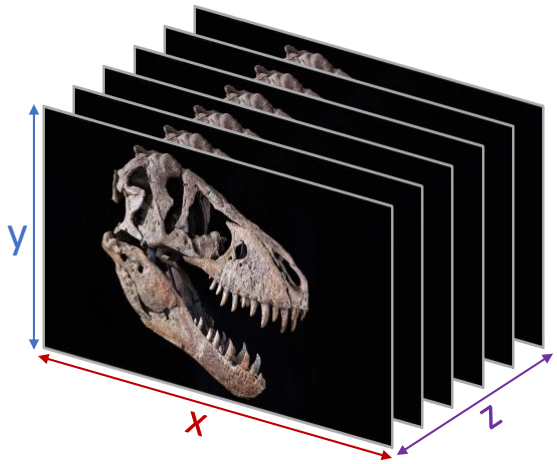

Figure 1: schematic showing the definition of the *x*, *y*, and *z* coordinates in a 3D data volume.

The orthographic projections should additionally have the following option:

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).

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).

2. Images and datasets:

Example images and datasets for your program to use are available on these links:

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.

3D Volumes (Download here: *Scans.zip*):

1) Fossilized *Confuciusornis* (prehistoric bird) CT scan[1] (Unzips into `Scans/confuciusornis`)
2) Fractured granite CT scan[2] (Unzips into `Scans/fracture`)

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 use in your project may need to be converted to an appropriate format before using them (e.g., png).

For each image you use, you should add a subdirectory in the `Output` directory of your repository with the same name as the original image (without the .png extension). In each output directory, you should include one image for each of the image filters you have implemented. Note that we will run your code to confirm the output matches these output images. You may also want to use other images for testing purposes or for scaling analysis. For example, for the image called `stinkbug.png`, you should add a directory `Output/stinkbug`, inside of which you can add filtered images for each filter, such as `Output/stinkbug/median.png`, `Output/stinkbug/grayscale.png`, etc.

For both CT scan datasets, you should include an output directory in which you upload each of the orthographic projections (MIP, MinIP, mean-AIP, median-AIP). For each projection, upload 4 images:

1) Without any filters
2) With a 3D Gaussian filter (3x3x3)
3) With a 3D median filter (5x5x5)
   (Note: output images 1-3 should operate on the whole data volume)
4) Without any filters, but operating on a thin slab:
   a. *Confuciusornis*: images `confuYZ0010.png` – `confuYZ0070.png`
   b. Fractured granite: images `granite1_0600.png` – `granite1_0800.png`

Also include slices in the *x-z* and *y-z* planes.

1) *Confuciusornis*: Slice in the x-z plane with a constant y value of **420**.
2) *Confuciusornis*: Slice in the y-z plane with a constant x value of **400**.
3) Fractured granite: Slice in the x-z plane with a constant y value of **138**.
4) Fractured granite: Slice in the y-z plane with a constant x value of **275**.

Note: Do not include the original CT scan images in your repository, as they are >100MB each.

3. Additional tasks:

As well as the core functionality listed above, you program 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**; 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) 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.
5) Include **compiled Windows and MacOS executables** of your program. If your group only has access to one of those operating systems, you can just upload one executable.

[1] https://doi.org/10.6084/m9.figshare.c.1612235_D59.v1
[2] https://doi.org/10.17612/P7QX1X

4. Comments regarding teamwork and 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 easy-to-use library is a difficult process, complete with many difficult design decisions to be made; the teaching team are always 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. Some of you may have some 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.

5. Evaluation:

Your code will be compiled and executed as part of your evaluation. Code that does not compile or does not output an image with the appropriate filter/projection applied correctly will not be able to score highly.

We will mark the projects based on:

1) Implementation (**40%**)
2) Execution and output images (**20%**)
3) Sustainability (code documentation, commenting, and testing) (**15%**)
4) Short 4-page report (**25%**)

Report

Your report should be a maximum of 4 pages of A4 (single spaced, 11pt, PDF). It should show that you understand the algorithms you have used to apply the filters and projections. Each group member should write a short paragraph (~100 words) documenting the filter they wrote. You should document the performance of your library, and how well it scales as image/data volume/kernel size changes. Also include some discussion on the advantages/disadvantages of your approach, and any design decisions that you might change if you had to carry out this assignment again. Make sure to include the name and GitHub usernames of all members of your group in your report. You should add a breakdown of who worked on which tasks in your group.

6. Submission:

Upload your code by committing to the main branch on GitHub by **4pm on Friday 24th March 2023**.

Your repository should include:

1) Your C++ source code.
2) Documentation (and readme) of how to install and use your program.
3) Your code testing framework.
4) An appropriate licence.
5) Your 4-page PDF report.
6) The required output images listed above (see Section 2).
7) Windows and/or MacOS executables.

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