Imperial College
London

Advanced Programming

Assessment 2: Group Project

Image Filters, Projections and Slices

Tom Davison
thomas.davison@imperial.ac.uk
RSM 4.85

Core task: Image Processing with C++

- Build a C++ program using the programming techniques you have learned during the Advanced Programming course.
- Take inputs of 2D images or 3D data volumes (e.g. CT scans)
- Apply image filters, orthographic projections and slices
- Output result as an image

Groups

- Groups have been automatically generated
- Aim to balance the number of ACSE/EDSML students in each group
- Also balance the average grade, to make it as fair as possible
- More about working in your teams later in the briefing
- Repositories will be released later today

Group names: Algorithms

Apriori	Monte-Carlo
Bellman-Ford	Naive-Bayes
Binary-Search	Otsu
Canny-Edge-Detection	Prim
Depth-First-Search	QuickSort
Dijkstra	Radix-Sort
Euclidean	Selection-Sort
Fibonacci	Tarjan
Gradient-Descent	Ukkonen
Huffman	Viterbi
Insertion-Sort	Warshall
Johnson	Xavier-Initialization
Kruskal	Yen
Linear-Regression	Ziggurat
MergeSort	

2D Image filters

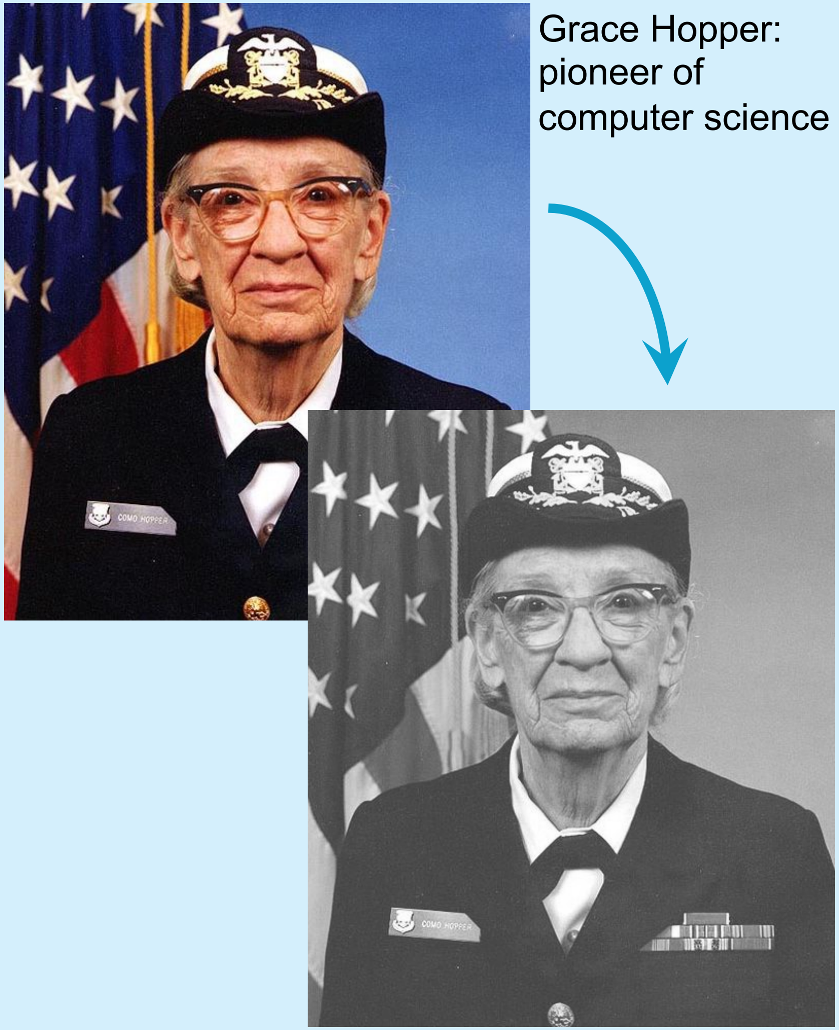
Colour correction filters

a) Grayscale

b) Automatic colour balance

c) Brightness

d) Histogram equalization



Grace Hopper: pioneer of computer science

2D Image filters

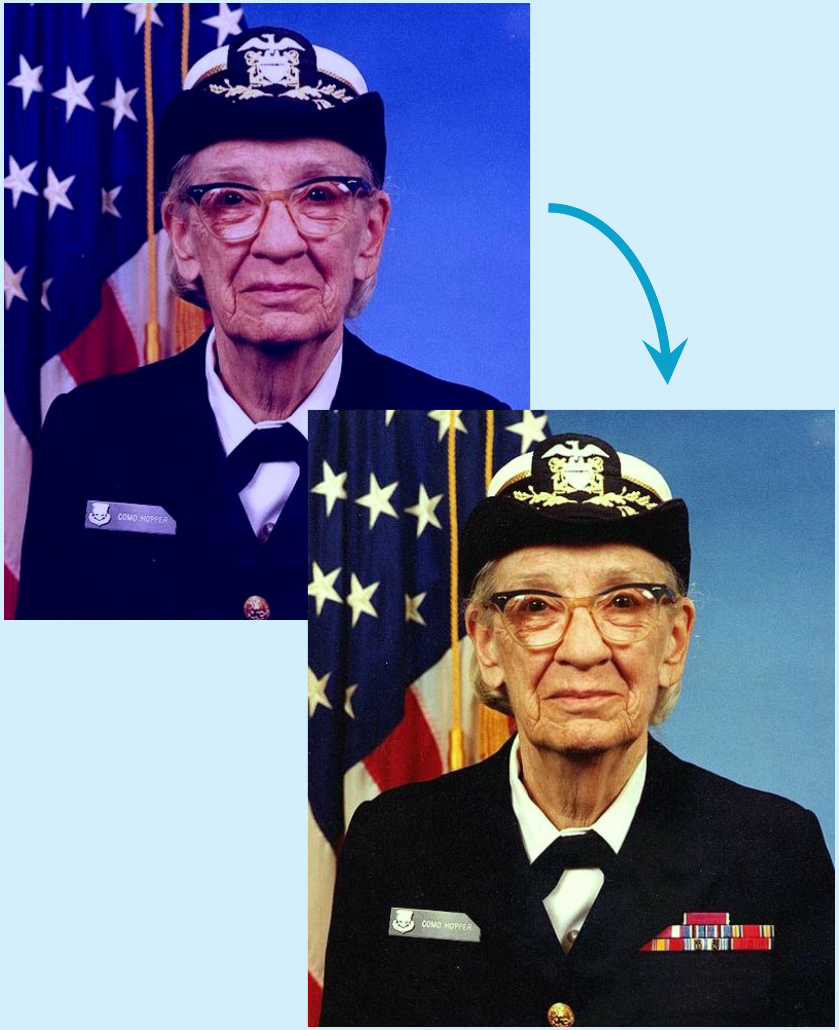
Colour correction filters

a) Grayscale

b) Automatic colour balance

c) Brightness

d) Histogram equalization



2D Image filters

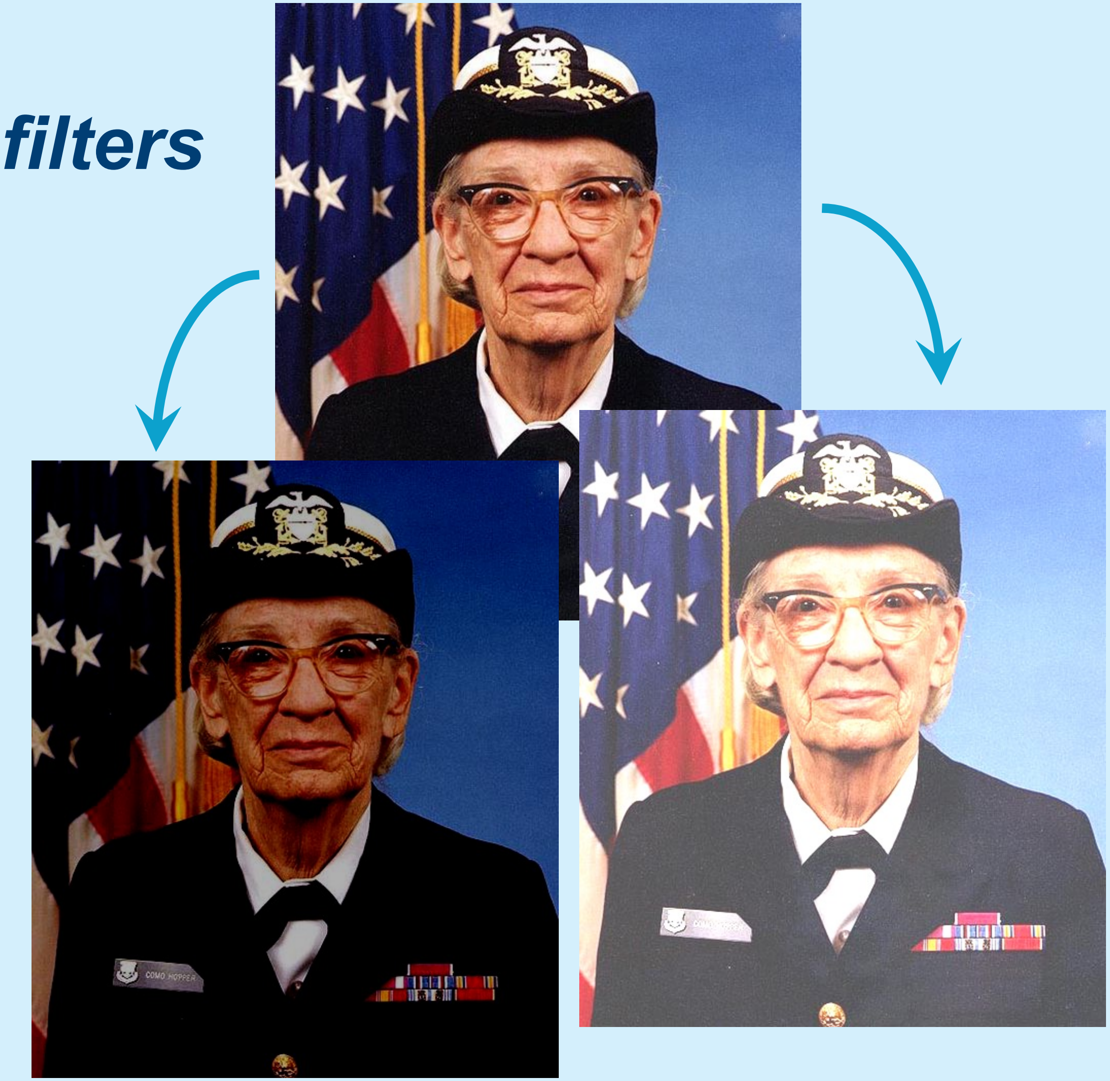
Colour correction filters

a) Grayscale

b) Automatic colour balance

c) Brightness

d) Histogram equalization



2D Image filters

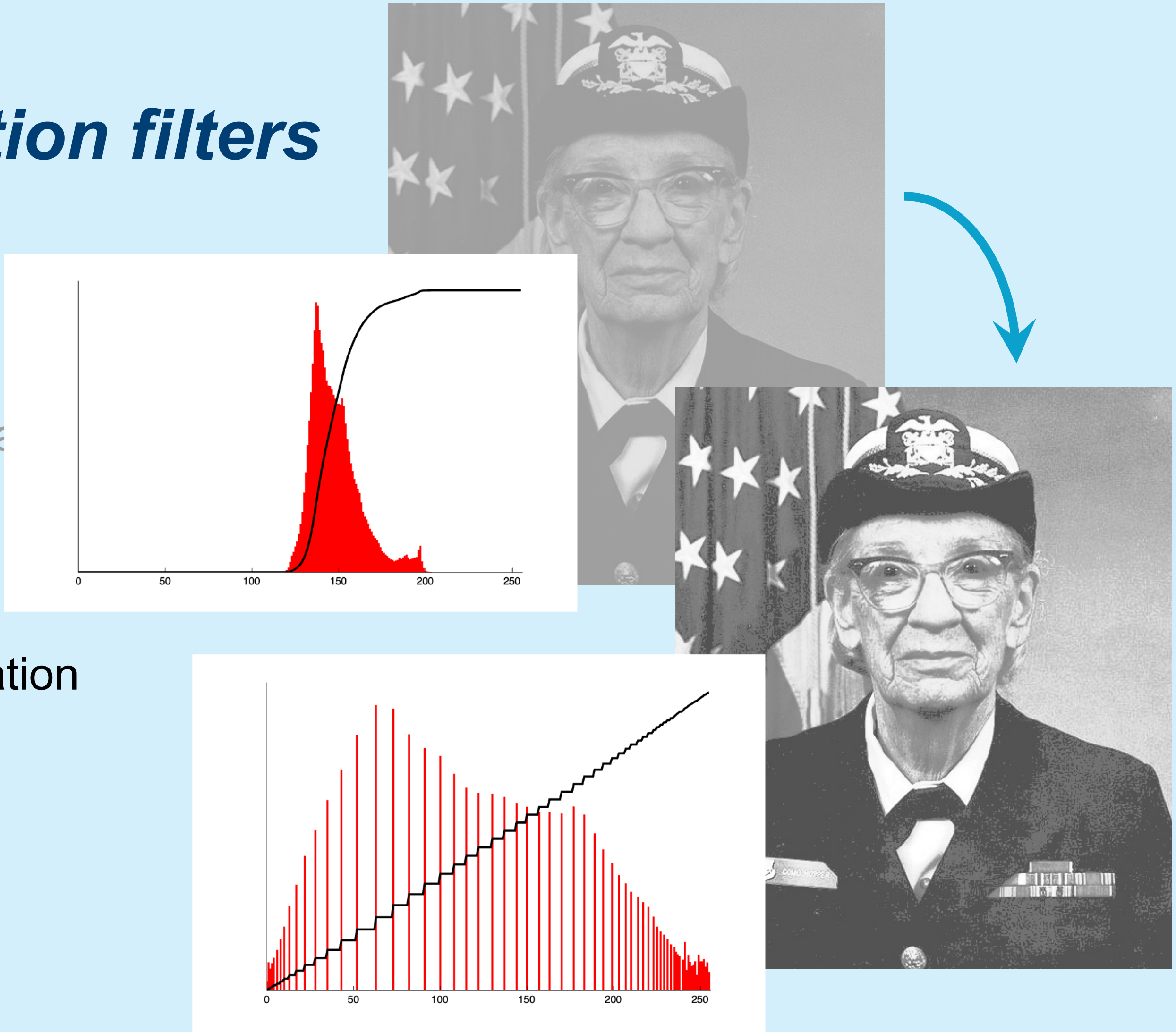
Colour correction filters

a) Grayscale

b) Automatic colour ba

c) Brightness

d) Histogram equalization



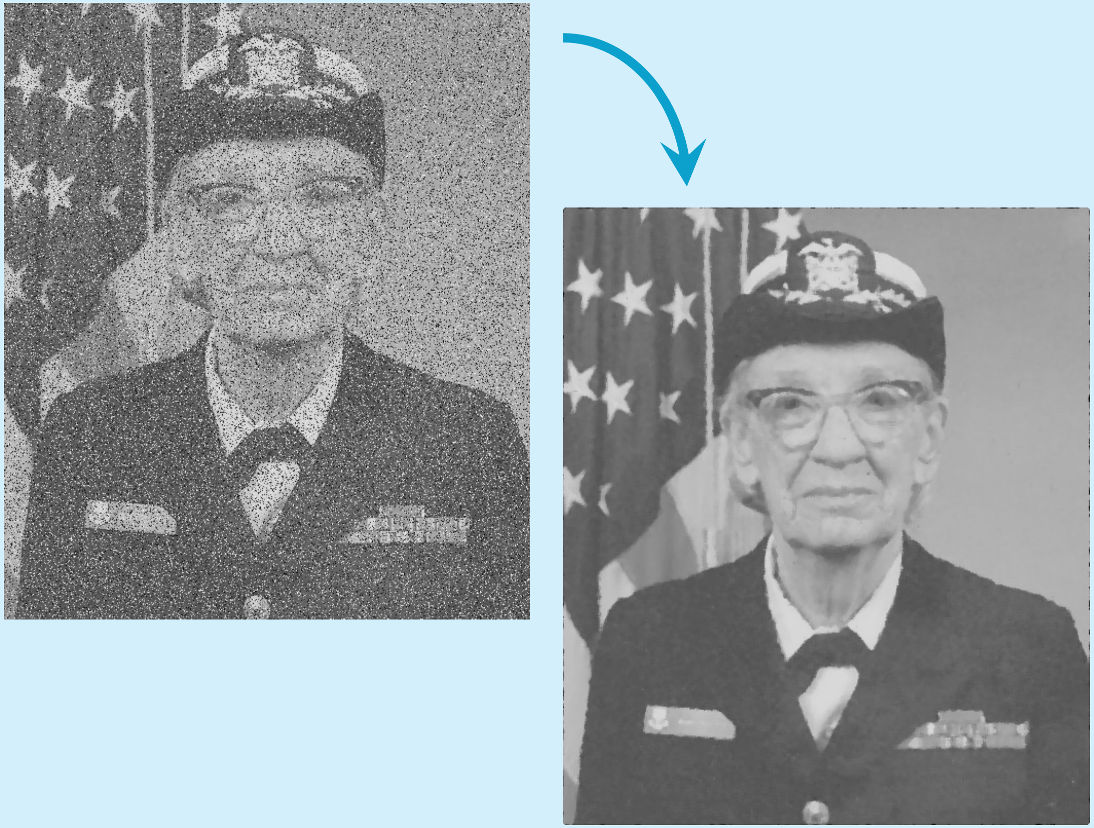
2D Image filters

Convolution filters for image blur – arbitrary kernel size

a) Median blur

b) Box blur

c) Gaussian blur



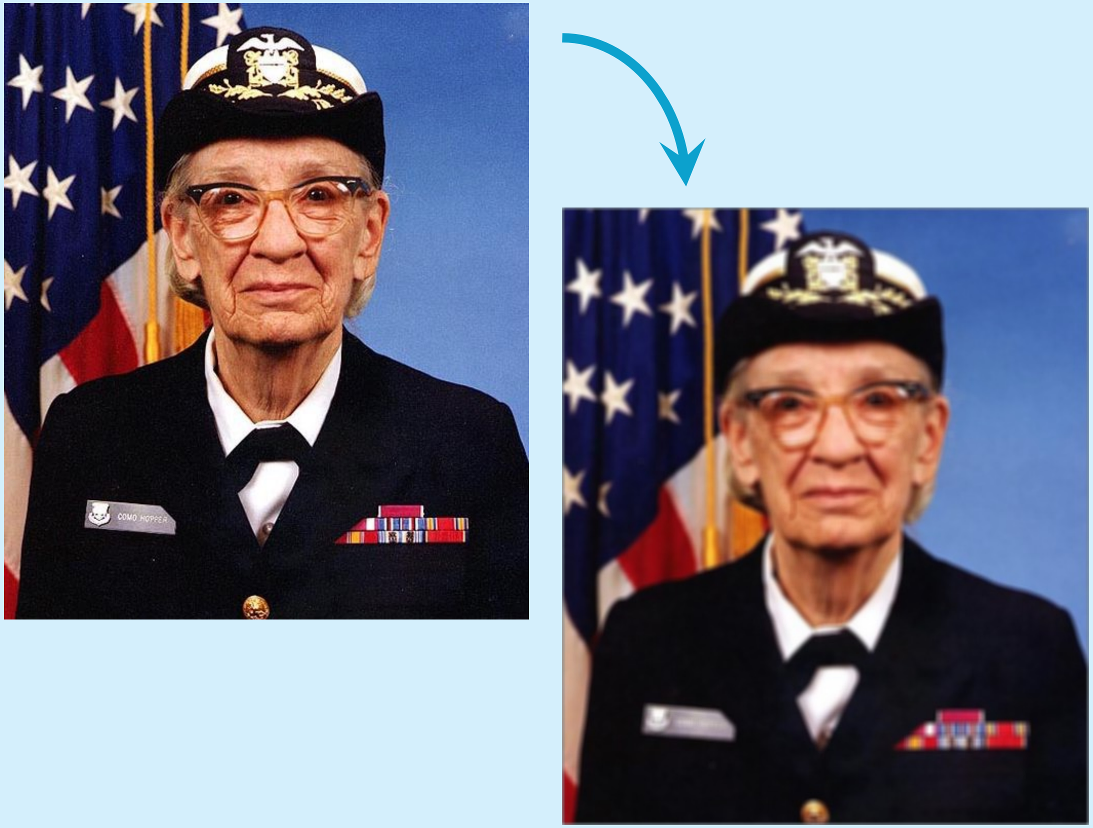
2D Image filters

Convolution filters for image blur – arbitrary kernel size

a) Median blur

b) Box blur

c) Gaussian blur



2D Image filters

Convolution filters for image blur – arbitrary kernel size

5x5 Box blur kernel

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04

Example 5x5 Gaussian blur kernel

0.003	0.013	0.022	0.013	0.003
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.003	0.013	0.022	0.013	0.003

Imperial College London

2D Image filters

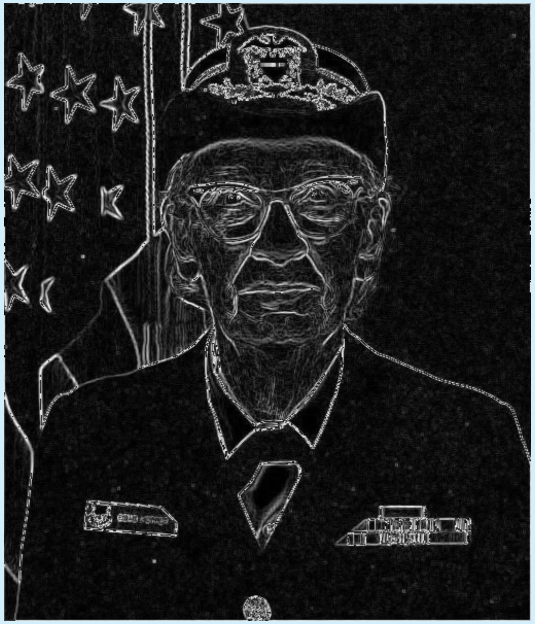
Convolution filters for edge detection – fixed kernel size

a) Sobel

b) Prewitt

c) Scharr

d) Robert’s Cross



(convert to grayscale first – use your filter!)

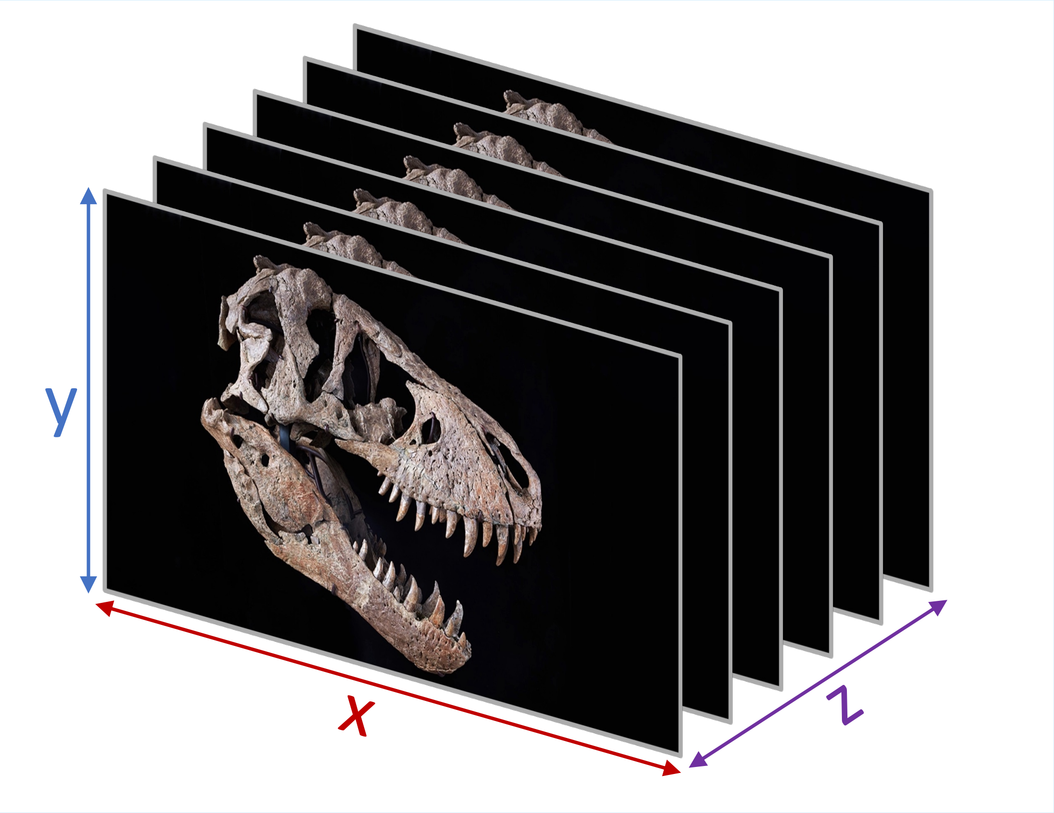
2D image filters

- Each member of the group should write at least one 2D image filter
  - Everyone gets a chance to write some code
- Write at least 6 filters as a group, more if you have time

3D Volumes

CT scans

- Stack of images
- CT scans are good examples of such data, e.g.
  - Medical
  - Palaeontology
  - Porous media



Imperial College London

3D Volumes

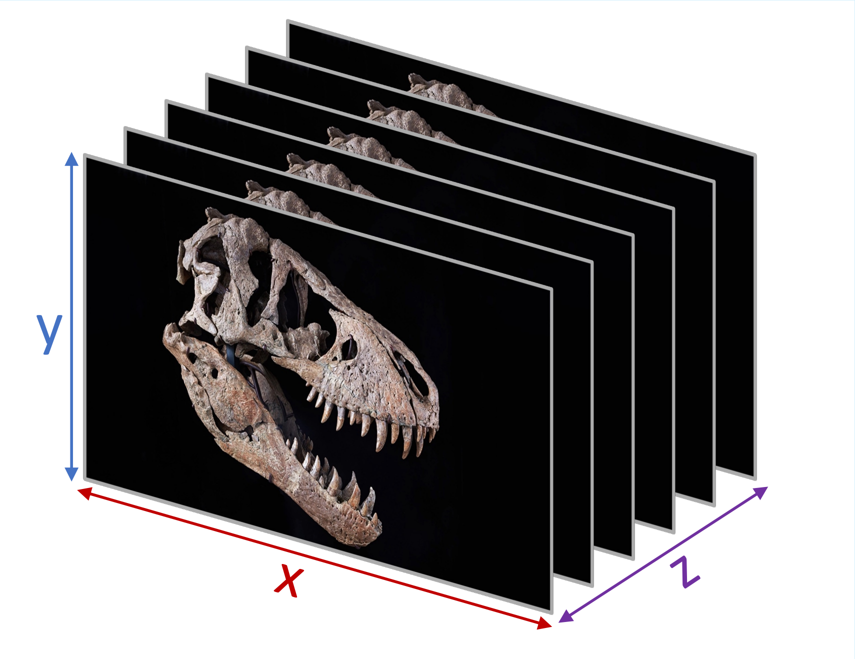
Filters

3D filters work in much the same way as 2D filters, but over the 3D volume

- i.e. need to consider neighbouring images in the stack

a) 3D Gaussian blur

b) 3D Median blur



3D Volumes

Orthographic projection

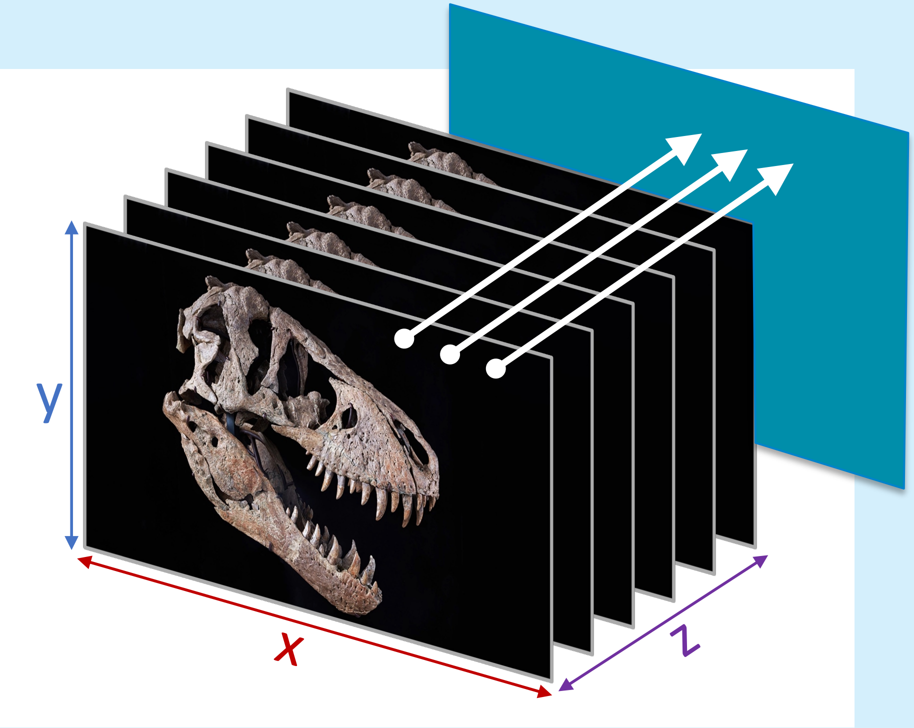
Projections look through each image in the stack, and project a feature onto a single image of the same dimensions

a) Maximum intensity projection (MIP)

b) Minimum intensity projection (MinIP)

c) Average intensity projection (AIP)

   – either using the mean or the median

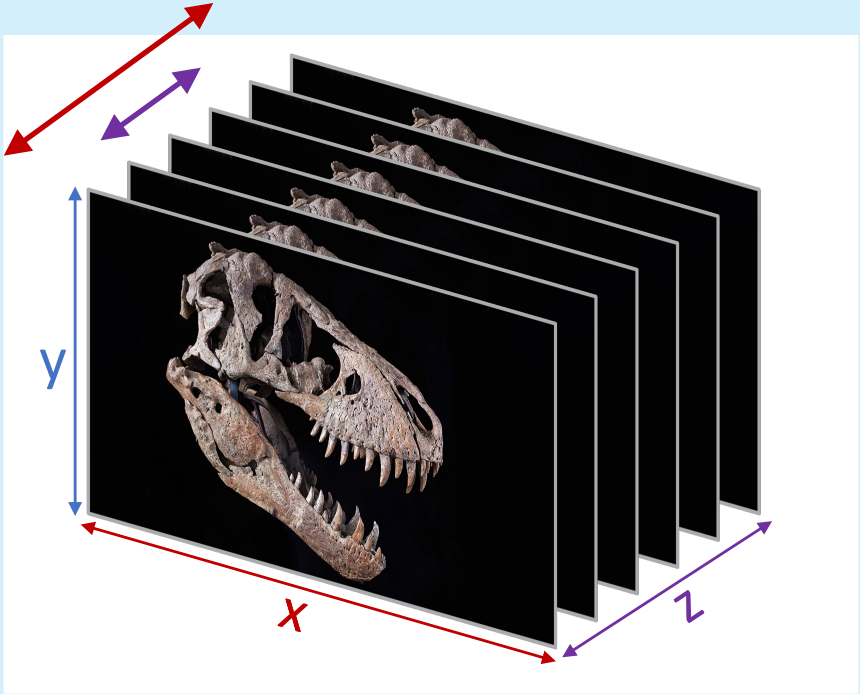


3D Volumes

Orthographic projection

Projections can operate on either:

- The whole volume available
- A thin slab (user defined region) in the z-direction

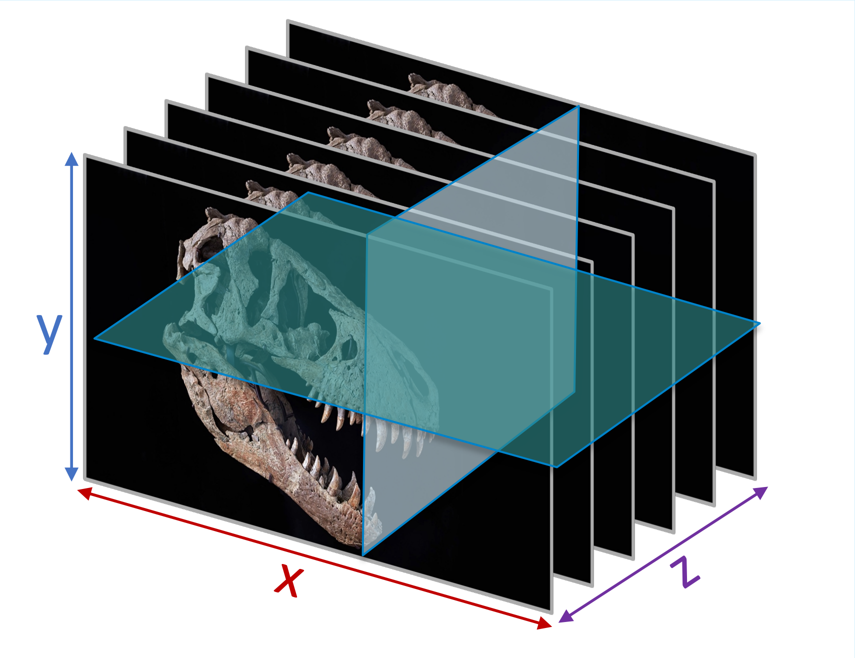


Imperial College London

3D Volumes

Slicing

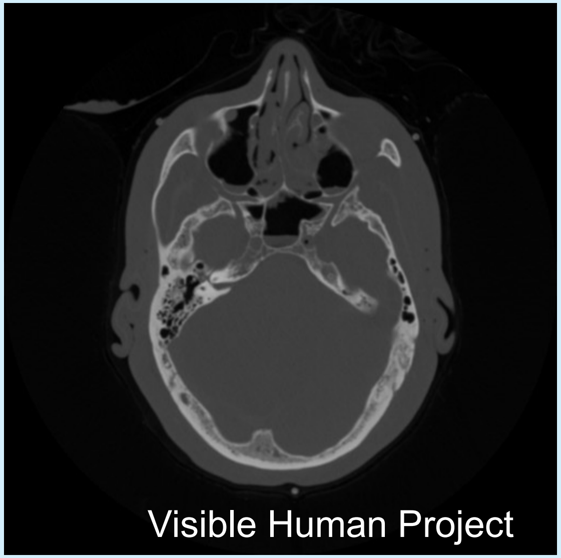
- Images provided in the *x-y* orientation
- Your code should be able to output an image in a different plane:
  - *y-z* (user defined *x*)
  - *x-z* (user defined *y*)



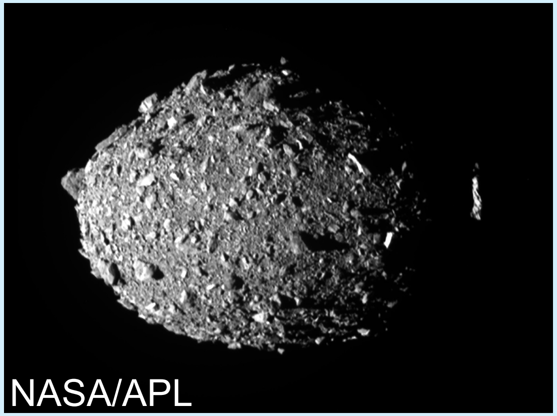
Example images

- We have provided you with a set of example images

- Please feel free to find other images which can demonstrate the capabilities of your code

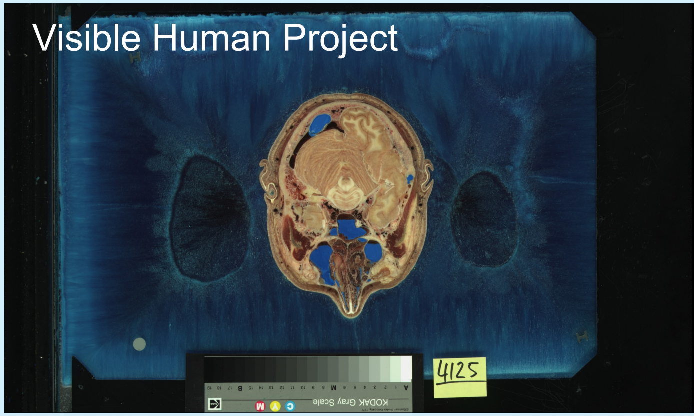


Visible Human Project



NASA/APL

NASA



Visible Human Project

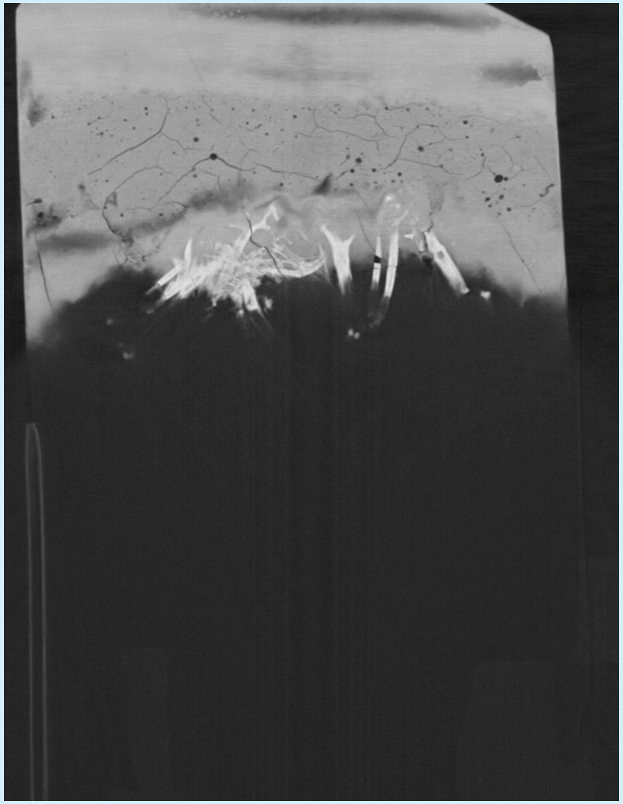
Imperial College London

Provided 3D Volumes (CT Scans)

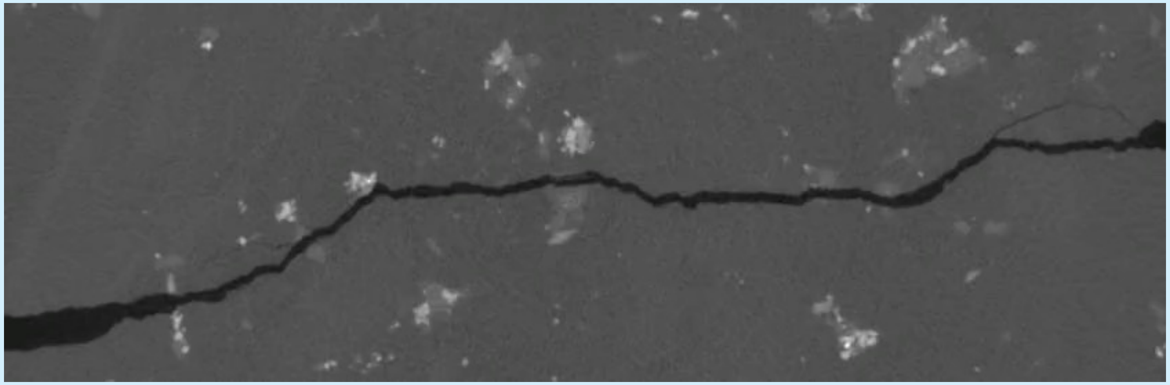
- Fossilized Confuciusornis (prehistoric bird) CT scan

- Fractured granite CT scan

See project description for a list of output images we would like you to provide from these scans



https://doi.org/10.6084/m9.figshare.c.1612235_D59.v1



https://doi.org/10.17612/P7QX1X

Imperial College London

Code structure

- A main program file is required with a `main()` function
- You should have a class for each of the following:
  - `Image`
  - `Volume`
  - `Filter`
  - `Projection`
  - `Slice`
- They should all have a `.cpp` and `.h` file
- Add more classes if you think they are appropriate
- Add a header to each source code file with the names and github usernames of each group member

Image reading and writing

- `stb_image.h` and `stb_image_write.h`
- Provided in your group repositories
- Open-source image I/O library
- Works with a range of image formats
  - e.g., png, jpeg, gif

- Minimal example in repository showing how to import the headers and how to read and write an image

Image reading and writing

```
#include <iostream>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

int main() {

    int w, h, c;
    unsigned char* data;

    // Read in image file
    data = stbi_load("example.png", &w, &h, &c, 0);

    // Print image size to screen
    std::cout << "Image loaded with size " << w << " x " << h << " with ";
    std::cout << c << " channel(s)." << std::endl;

    // Save image to new filename
    int success = stbi_write_png("output.png", w, h, c, data, 0);

    // Deallocate memory
    stbi_image_free(data);

    return 0;
}
```

Sustainability

- Follow all best-practices you have learned in other courses up to now
- Make sure all code is commented well
- Provide documentation for uses to know how to compile and run your code
- Include a readme and license file in your repository
- Use pull requests, code review, issues, etc. on GitHub
- Add a testing framework (unit tests, etc)

- Add a compiled executable to the repository
  - Windows and/or MacOS (preferably both!)

Teamwork

- Remember – the main aim of this module and project are for you to learn how to write C++ code

- Some of you have more experience than others
  - Doesn’t mean we want one or two people writing all the code!
  - Everyone should write at least one image filter, but we would prefer you all to write more than that!
  - Team effort, work together and support each other

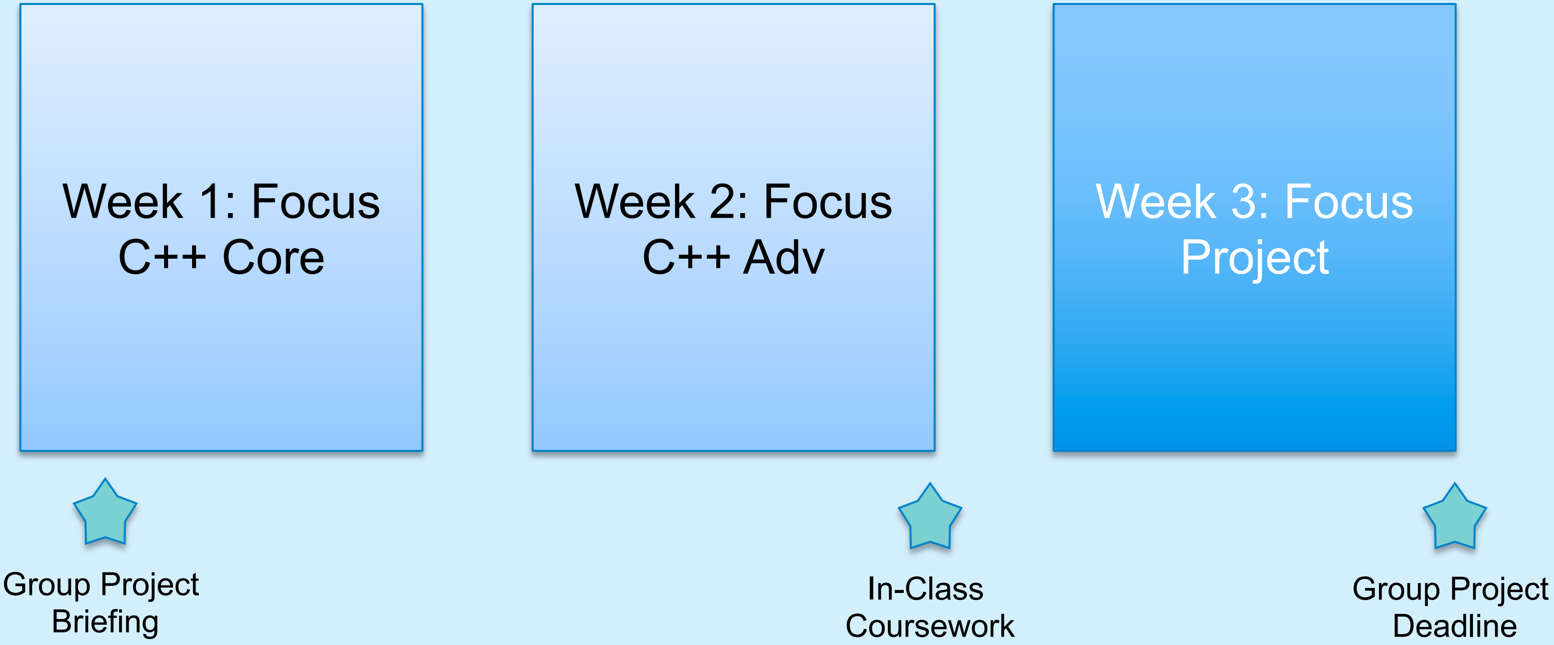
- Paired programming can help. Ask each other questions if you are unsure about anything in the code

Advanced Programming

Imperial College London

Schedule

- Possible to complete project in one week.
- Focus on the lectures an individual assessment for the first two weeks
- Apply that knowledge during week 3 for this project.



Week 1: Focus C++ Core

Week 2: Focus C++ Adv

Week 3: Focus Project

Group Project Briefing

In-Class Coursework

Group Project Deadline

Recommendations

- Start simple
- Build some of the easy filters and parts of the interface first
- Add comments and unit tests as you go
- Build complexity later (e.g. convolution filters, 3D projections, etc.)

External libraries

- Other than the `stb_image.h` and `stb_image_write.h` headers, **no other external libraries are allowed**

- Standard libraries are fine
  - e.g., iostream, string, vector, cmath

- Best way to learn programming is to build something from scratch!

- **Do not copy code from the internet or other sources**
  - **(that includes sharing code between groups)**

Imperial College London

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:

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

Imperial College
London

Report

- Maximum 4 pages (single space, 11pt, pdf)
- Demonstrate you understand the algorithms employed for your filters and projections
- Each group member write about their own filter
- Performance – how well does your library scale with image/volume size and kernel size
- What would you change if you did the project again?
- Breakdown of who worked on which aspects of the project

Submission

Upload your code by committing to the main branch on GitHub by:

4pm on Friday 24th March 2023

Anything included in commits after this time/date will not be assessed.

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.
7) Windows and/or MacOS executables.