

The preferred programs for examining, manipulating, and simple image analysis are ImageJ and Fiji

ImageJ and FIJI

- Developed at the National Institute of Health in the USA
- The ImageJ application is very popular and has a large support community
- Written in Java, it will run on almost anything ☺
- It supports many file formats
 - Some inbuilt, some via 'plugins'
- It handles stacks of 2-D images better than many other applications
- It's free! (<https://imagej.nih.gov/ij/>)

(Schneider, C. A.; Rasband, W. S. & Eliceiri, K. W. (2012), "NIH Image to ImageJ: 25 years of image analysis", Nature methods 9(7): 671-675)



ImageJ is an open source Java application.

It runs on any computer with a Java 1.8 or later virtual machine.

Fiji is just ImageJ

- Over the years (since 1997) many plugins have been written for ImageJ by many groups and individuals
- Fiji has many of these built in. (Originally aimed at neuroscience)
- Fiji has a more sophisticated updating system than ImageJ (<https://fiji.sc/>)

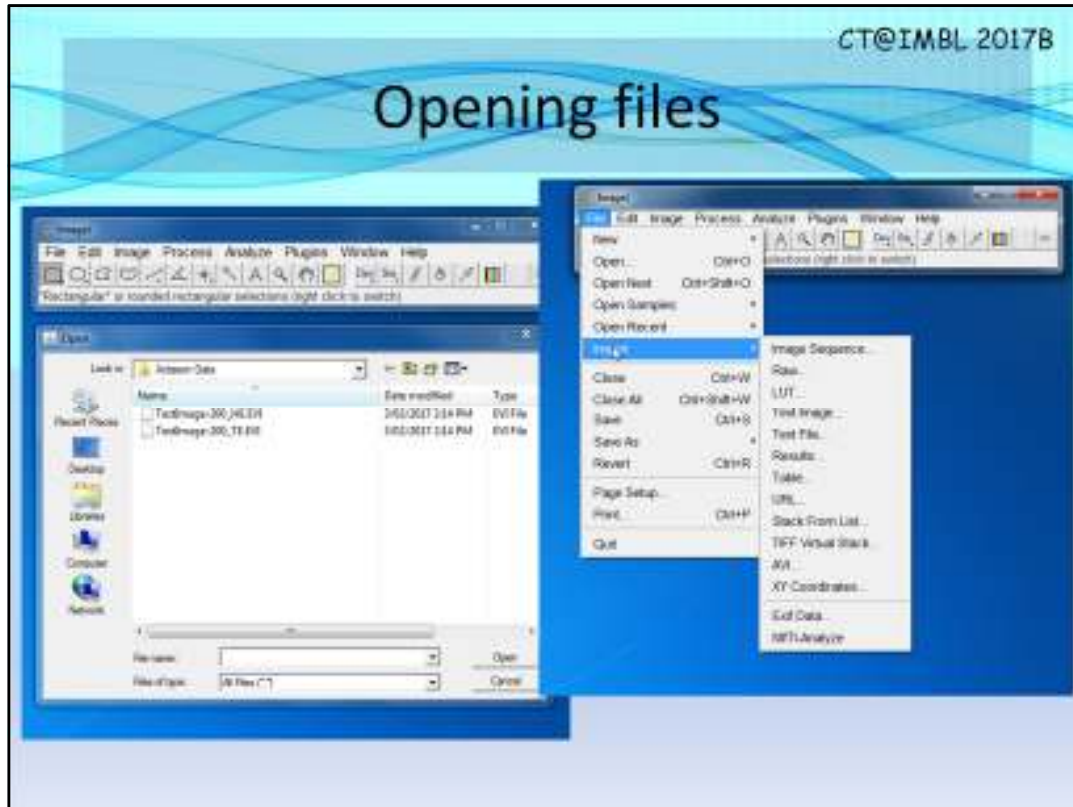


(Schindelin, J.; Arganda-Carreras, I. & Frise, E. et al. (2012), Schneider, C. A.; Rasband, W. S. & Eliceiri, K. W. (2012), "NIH Image to ImageJ: 25 years of image analysis", *Nature methods* 9(7): 671-675 *Nature methods* 9(7): 676-682)



Fiji is built from ImageJ

It is a "batteries-included" distribution of [ImageJ](https://imagej.nih.gov/ij/), bundling a lot of plugins which facilitate scientific image analysis



First step: opening up image files.

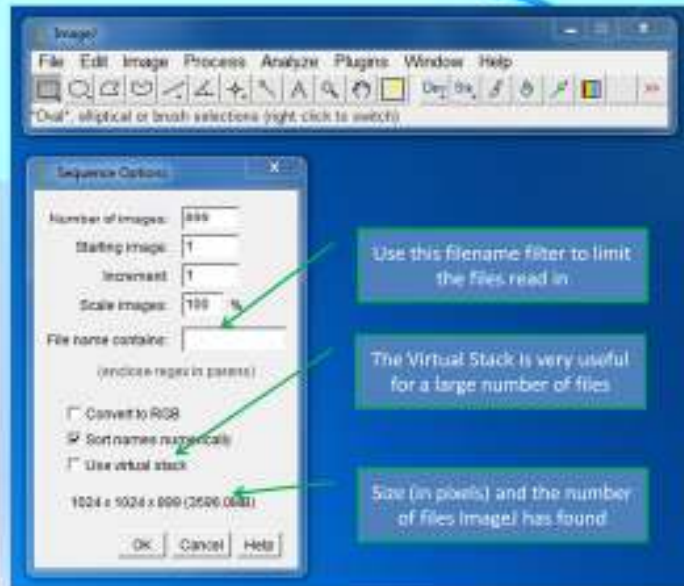
If the image file format is recognised by ImageJ, either by the extension or by recognition of the first part of the file it will just open it up.

Otherwise you 'Import' the data.

There are several importable file formats, including raw binary data, and text (ascii) files. ImageJ will even read in movies in .avi format. In doing this it will create an image stack.

Opening several files

- If all the images are the same size. The result is an image **stack** within ImageJ (more on this later)



The image Sequence import is a very useful way to read in a lot of images.

If the files format is standard and the images are the same size. ImageJ will read them all in to a 'Stack'.

The image stack is a very useful construct. It is a 3-D cube created by layering each 2-D image data.

The natural X and Y axes are those of individual images. The Z axis runs down through the stack.

The raw CT data can be read in as a stack and examined quickly and easily this way.

If the stack of images is very large, ImageJ can leave them on the storage, rather than reading them all into memory.

ImageJ Image types

- There are several ways that a 2-D array of numbers (pixels) can be shown on the screen as an image.
- A pixel value is always *mapped* to screen greyscale intensity or colour (value can be integer or real)
- Pixel values can index into a colour chart (only integers)
- RGB: Three numbers per pixel = **red**, **green** and **blue** intensities (only integers)

1.42	0.08	1.58	1.22
1.95	1.90	1.69	0.16
0.33	0.47	1.74	1.47
1.30	1.50	0.78	1.03

240	242	123	214
132	24	232	69
57	58	201	32
51	133	155	31

128,53,190	189,185,240	135,238,13	90,36,104
29,96,5	14,92,176	239,71,222	157,174,219
203,75,36	163,88,20	12,174,219	214,21,194
186,115,121	9,236,7	31,182,243	129,174,130

A brief introduction to image types. An image is really a 2-D array of numbers. Where the number usually represent some spatially resolved physical quantity.

To display them on a screen the numbers can be mapped to optical light colours, using the red/green/blue (RGB) system. Three integer numbers.

Colours can be confusing so often the image pixel intensity is mapped to just a grey scale luminosity. One integer number.

The numbers in the image array can be real, or integer values. They are generally called 'pixel values'.

Raw data from the detectors is always quantised as integers.

Viewing images

- Our screens allow '32 bit' colours = three 8 bit numbers, representing the **red**, **green** and **blue** intensity, for each pixel.
- The mapping of the pixel values will always end up as an 'RGB' value for the display.
- Using a real value, then mapping it to one 8 bit number (0 to 255) determines the intensity (black to white, $R=G=B$)
- Real numbers (or integers) in pixels are mapped to 8 bit RGB integer(s) first through a Look Up Table (LUT)

To display the image on the screen the pixel values are mapped to either a single integer for grayscale, or three integers for full colour.

The mapping is made through a look-up table or LUT.

ImageJ has a variety of in-built LUTs. It also has the ability to invert the LUT. So for instance the greyscale can range from white to black instead of black to white.

Full editing of the LUT is also supported.

Viewing images

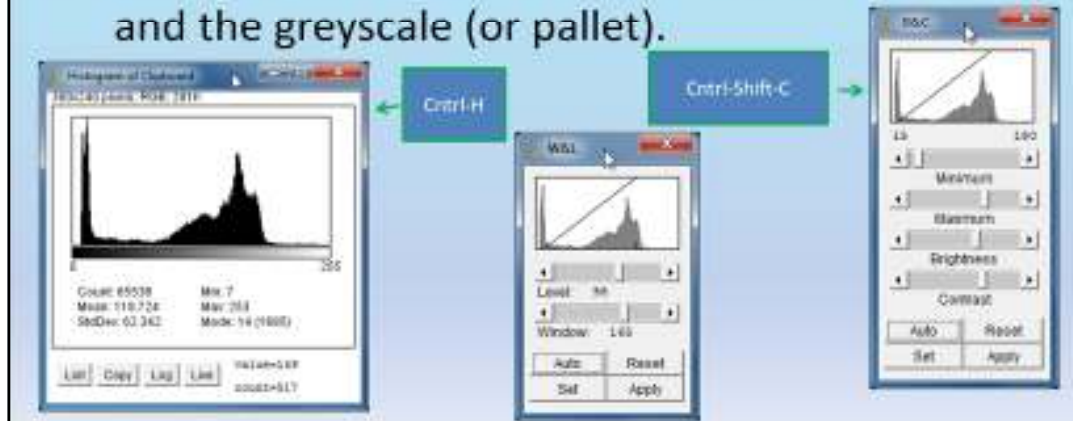
- There are several standard LUTs in ImageJ. Some are more confusing than useful!



Choosing the correct LUT for your image can sometimes just be a matter of preference. The extra information gained from a colour LUT can be confusing in some cases.

Brightness and contrast

- Two best ways to adjust: *brightness and contrast*, or *window and level*.
- Changes the mapping between the pixel value and the greyscale (or pallet).



Changing the range of pixel values covered by the LUT can be done in several ways. ImageJ provides a histogram of the pixel values. Then allows the maximum and minimum values used by the LUT to be set.

When using a greyscale the slope, and offset of the intensity mapping is overlaid on the histogram.

This can be manipulated using the end-points, the brightness and contrast, or the window and levelling controls.

Making calculations on images

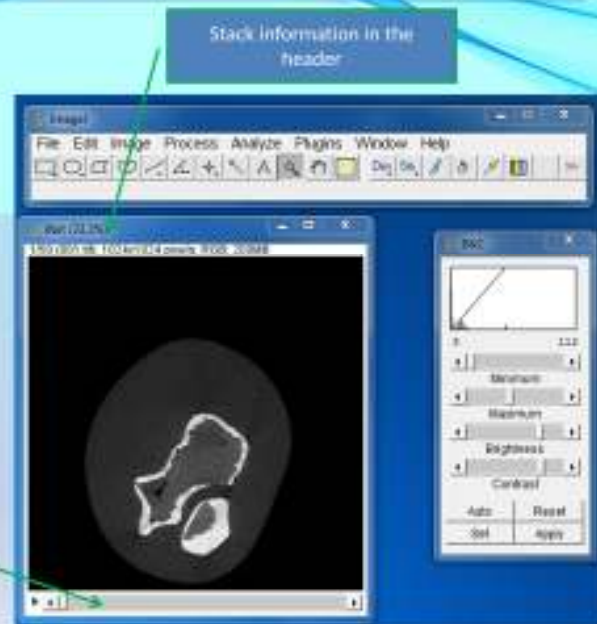
- 'Process' operations perform calculations on a single image (e.g. add a number to all pixel values)
- 'Image Calculator' operates on *two images* (e.g. add together corresponding pixels values in each image)
- E.g. A rough flat field correction can be made using 'divide' (Rough because of dark pedestal
 $\frac{I_a+P}{I_b+P} \neq \frac{I_a}{I_b}$)

Calculations on pixel values are most often made from the Process dropdown. An Image Calculator for allows the operands for the calculation to be pairs of images. You can use this to make a rough flat field corrections to images with corresponding white fields by making a simple division between the two images.

Working with stacks

- Input as an 'Image sequence'
- Or a single file containing multiple images

The slide bar will show the image sequence in the stack



The 'stack' is a very useful data structure in ImageJ and Fiji.

It is a series of images which can be viewed and manipulated as a 3-D array (volume) of data.

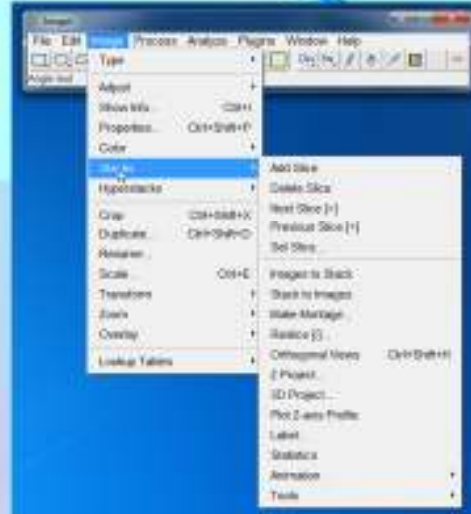
Sequences of images (including movieis0 can be read into an image stack.

Individual 2D images or planes through the array, can be viewed in the image area.

As the slide bar on the bottom is moved, images or planes are displayed at a 'depth' corresponding to the slider position.

Stack operations

- Many ways to manipulate and view the stack
- Note: Some plugins for 3-D viewing are rather slow
- Use:
Image>Stacks>Plot Z-Axis profile
- Use:
Plugins>Stacks>Measure Stack



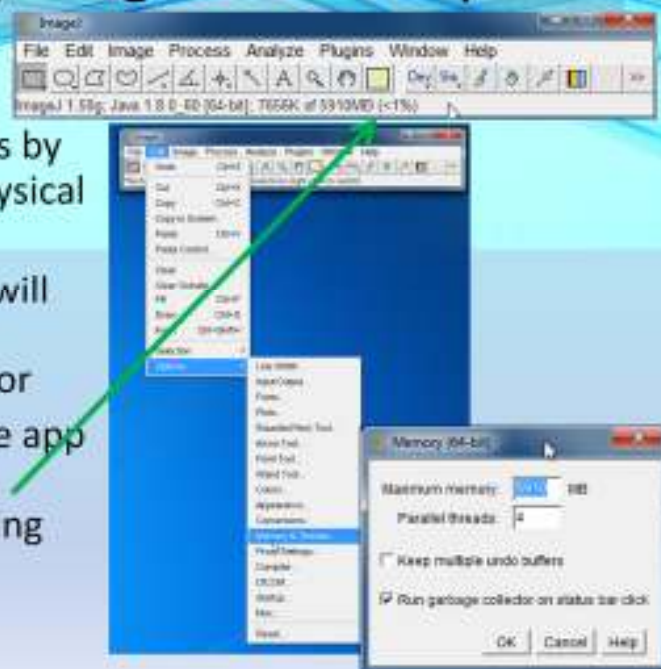
Measurements, line-outs (profiles) and many other parameters can be extracted from the image stack.

Most (but not all) of these abilities are in the Image>Stacks drop down menu.

When using the image calculators tool, ImageJ will ask if you want to make the calculation on the whole stack, or just the visible image.

Altering ImageJ's memory

- Allocated memory is by default ~ 75% of physical memory
- Trying to use more will result in an 'OutOfMemory' error
- The status bar in the app tells you how much memory you are using (and tidies it up if clicked)



If the stack is deep and the images are large, the memory requirements for the stack can be high.

One way around this is to use the 'Virtual stack' option when reading in the image sequence.

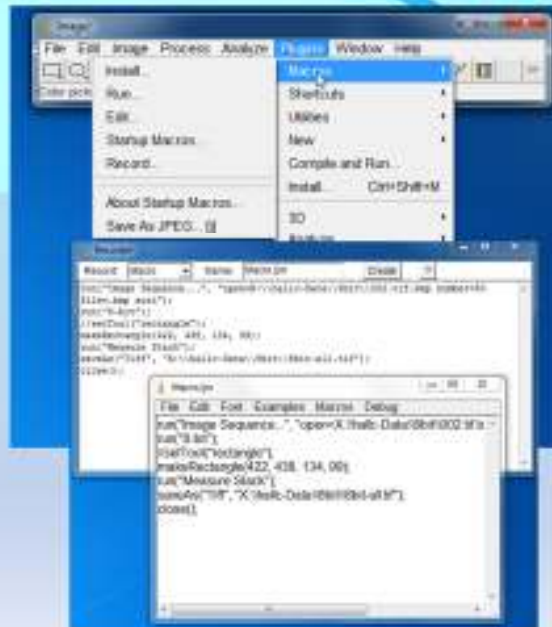
Some operations cannot be run on virtual stacks and do require the whole set of images to be stored in RAM.

The maximum memory ImageJ/Fiji can allocate is determined by the parameter in the Edit>Options>Memory and Threads preferences.

It is recommended that no more than 75% of the available memory is allocable to the program.

Scripting (macros)

- A sequence of operations can be captured and saved in an edit box
- Then saved as a text macro file (.ijm)



Any ImageJ operation can be captured and replayed through the macro capability. Macros can be recorded whilst performing the operations. Then saved as java-script like text in a file. Fiji has even more extensive scripting options. Allowing Python scripts to be interpreted.

Rich macro language

The diagram illustrates the features of a rich macro language by pointing to specific lines of code in a script editor window titled 'DisplayWindowTitles.py'. The features and their corresponding code examples are:

- Many built-in functions:** Points to the `getList("window.titles")` function call.
- Conditional flow control (if-then-else):** Points to the `if (list.length==0)` and `else {` block.
- Simple print to log or console:** Points to the `print("No non-image windows are open");` statement.
- Conditional flow control (for loop):** Points to the `for (i=0; i<list.length; i++)` loop.
- 'Background' mode for increased speed:** Points to the `setBatchMode(true);` statement.

```
File Edit Font Examples Macros Debug
// Display Window Titles
// Displays the titles of non-image and image windows.
list = getList("window.titles");
if (list.length==0)
    print("No non-image windows are open");
else {
    print("Non-image windows:");
    for (i=0; i<list.length; i++)
        print(" " + list[i]);
}

print("");
if (nImages==0)
    print("No images are open");
else {
    print("Image windows:");
    setBatchMode(true);
    for (i=1; i<=nImages; i++) {
        selectImage(i);
        print(" " + getTitle());
    }
}
print("");
```

The macro (script) language is full of features. Including program flow control, and many built-in functions.

Many other features to explore

- Filtration
- Segmentation
- Maths macro
- Calibration and measurements
- Batch macro
- Stitching (Fiji)
- Etc.



There are lots of other features to explore. Too many to cover in this short talk.

A few of the more prominent ones are listed here.

In particular the ability to register, and stitch two adjacent images (with some overlap) is important.

That facility will be used in the presentation on image stitching later in the workshop.

The recommendation from me is to try these out yourself with your own images or image stacks.