IMAGE PROCESSING : DIGITAL IMAGE

> This course is proposed by all the sources at the end of this document. The data below are not own by us. It's a filtering of all the data that we could gather to enhance our notes.

1. DIGITAL IMAGE

In image processing softwares like ImageJ, what's a digital image? how the grays, colors are encoded? how these data are stored in a file? ImageJ is a software wich is :

- Free
- Open source
- Based on Java
- Composed by addon

There are different type of images :

- **2D** :
    - Photography
    - Optical microscopy
    - Electron microscopy
- **2,5D** : Stereoscopy, Scan electron microscopy (like an actual 3D movie)
- **3D** : Confocal microscopy, Tomography

> 2,5D = pair of 2D images 3D = several 2D images

> 3D = Serial sections ==> Several 2D images
> 3D = ??? ==> Several 2D images

Digital image : it's an array of pixels (= picture+element) composed by width and height.

> The first pixel (0,0) is always on the top left corner
> The last one is therefore (w-1,h-1)

An image is defined by :

- **Resolution** : pixels number per unit of length (or dpi : dots per inch (ppp in French))
- Range of gray levels or color, wich define the **dynamic range**. The dynamic range is a **value** (1 bit, 8 bit, RGB, etc ...)
- **Color Images** : One pixel per chanel (color space), either RGB, CMYK, etc.

A. Binary images

Binary images (black and white) are everywhere in ImageJ. In a binary image, the pixels are in two states: ON or OFF. Better than saying ON and OFF, the TRUE and FALSE keywords are used.

What are TRUE and FALSE in a computer ?

In programming languages, a condition returns TRUE or FALSE but behind these two keywords, there are numbers where **TRUE equal to 1** (one) and **FALSE to 0** (zero) as shown in this small IJ macro/script:

```
print( (1==1) ); // Display 1 (TRUE)
print( (1==2) ); // Display 0 (FALSE)
```

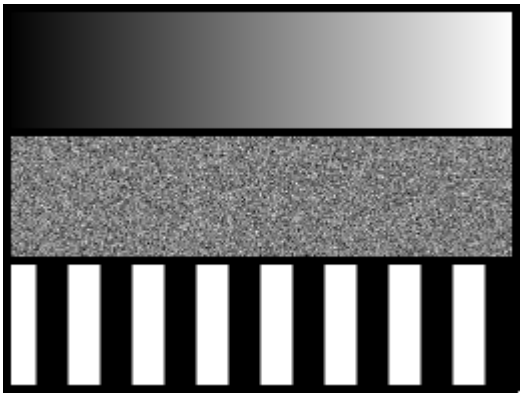
Now, what about the binary images?

In ImageJ, a binary image is an **8-bit image** where FALSE corresponds to a pixel value of 0 (in terms of "color", this is **black**) and TRUE to a value of 255 (equivalent to **white**). Thus, a binary image is black and white.

B. Gray-level images

Introduction

Gray-level images - unlike the binary contain **shades of gray** from black to white as shown :



Bits per pixel and Dynamic range

In photography, the **dynamic range** corresponds to the luminance of the scene and is captured/converted by the digital camera. The limits of luminance range - that is the number of shades available in the digital image - is directly related to the number of bits allocated per pixel: **8, 16, and 32 bits**.

Just take the example of the 8-bit image where each pixel is defined by **one byte (8 bits)**. Thus, the minimum value is ...

00000000 = 0

... and the maximum is ...

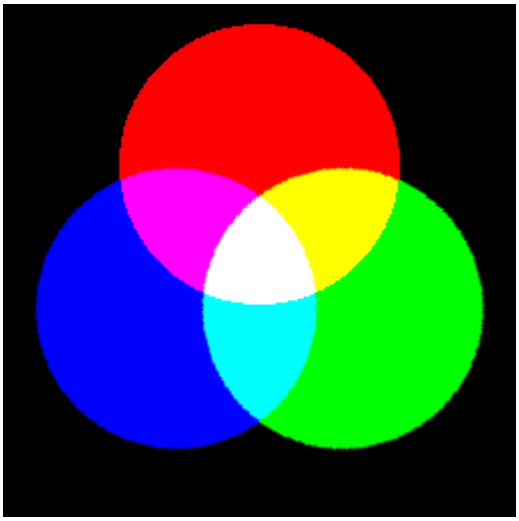
11111111 = $\mathbf{1x2^7+1x2^6+1x2^5+1x2^4+1x2^3+1x2^2+1x2^1+1x2^0}$ = 2^8-1 = 255

In this case, we have a **maximum of 256** (from 0 to 255) different shades in a 8-bit image . For a scientific image, this is rather **limited** because most of the CCD or CMOS camera work with 12- or 16-bits precision. Thus, in a scientific project, we are working with **16-bits images** with 65536 (216 from 0 to 65535) shades of gray.

C. Color images

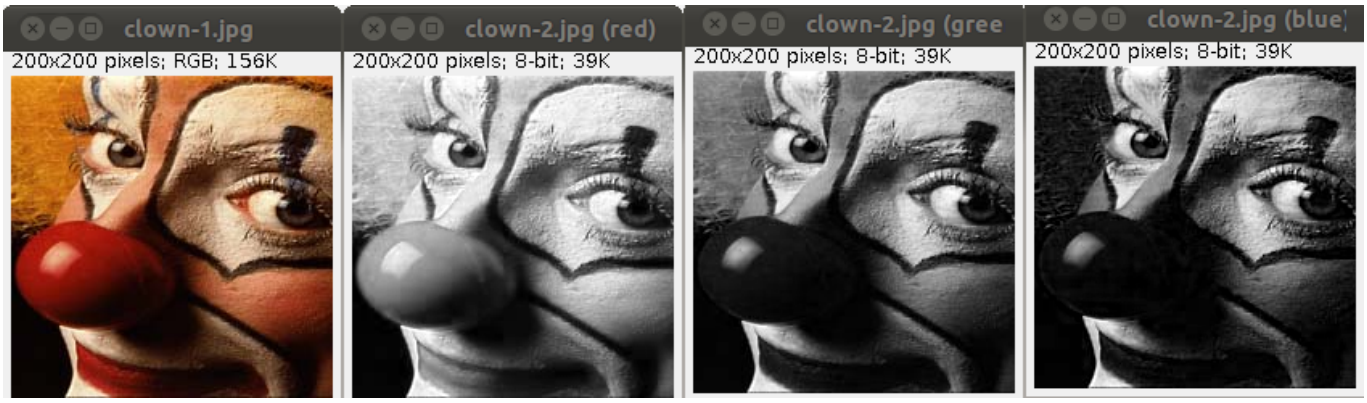
After the binary and gray-level images, the color images represent the last (and the most sophisticated) family of images used in image processing softwares. Colors are obtained by the **combination of three primary colors**. The latter may be different depending of the device displaying/printing the image. In ImageJ, the most usual color images are encoded with the **RGB color space**.

RGB (for **Red,Green, Blue**) is the most common color space used by computers. It belongs to the **additive color space family**. That means that you **start from the black color** and by **adding** various quantities of red, green, and/or blue, you obtain all the color shades as shown:



How can we observe the three color channels?

Three 8-bit images are created corresponding respectively to the red, green, and blue channels. This means that a color image is a **24-bit image** (equal to three channels of 8-bit). This type of RGB is also called **RGB-24 or RGB-888**. Now, what about the colors? For example, the clown's red nose appears as a combination of light gray in the red channel and dark gray in the two others (green and blue) leading to the color red (corresponding roughly to red > 200; green ≈ 0; blue ≈ 0).

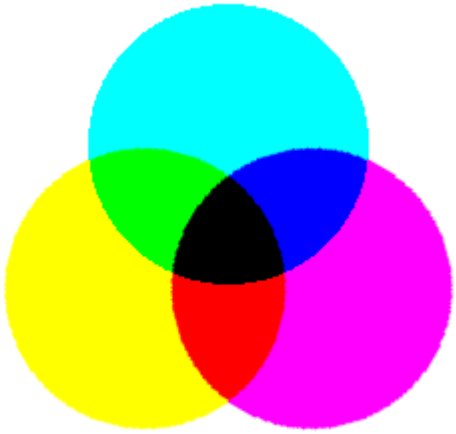


CMY(K) color image

CMY (or its variant CMYK) color space uses as primary colors: **cyan, magenta, and yellow**. This color space used by printers is interesting because, this is an example of a **subtractive color model**. When you **add color(s), you converge towards black color**.

CMY color space

In this color space, **(0,0,0)CMY is white and (255,255,255)CMY is black**. As shown, the primary colors cyan, magenta, and yellow have respectively the values (255,0,0)CMY, (0,255,0)CMY, and (0,0,255)CMY. For example, green is equal to (255,0,255)CMY corresponding to the complementary color in RGB (0,255,0)RGB.



2. IMAGE FILE

From 1D to 2D

The image is stocked as an 1D array in the memory, so to know the composition of the 2D image, we use a formula using the width (or the length)
$$2D:p(i,j)=p(i+w*j):1D$$*
$$3D:p(i,j,k)=p(i*+w*j+wh*k):1D$$

Hexadecimal

To take less time, informatician invented the **hexadecimal** which allows to communicate more information with less signs. To do so, they cut the 24 bits of RGB into pieces of 4, and those 4 bits are converted into an hexadecimal formt which goes from 0 to F, F worthing 15 and 0, 0.

> 1000.1001.1011.1000.1010.0110
> 1000 1001 1011 1000 1010 0110
> $\hspace{15px}8\hspace{30px}9\hspace{30px}B\hspace{30px}8\hspace{30px}A\hspace{30px}6$
> $\hspace{90px}$#89B8A6

Signed, Unsigned pixel values

Among the various image types proposed to import a **gray-level image**, even though you know the dynamic range of your image, you have to choose **'signed' or 'unsigned'**...

1- A 'bit' of theory

Java (the programming language of ImageJ and its virtual processor, the JVM) encodes signed numbers with the method called "two's complement". **The last bit** (the most significant) is used for the **sign** (a value of **0 for positive** and **1 for negative**). For positive 16-bit numbers, no extra process are done, the bits #0 to #15 are

used allowing the encoding of numbers from 0 (00000000 000000002) to +32767 (01111111 111111112). For negative numbers, a small calculation is done consisting of a **bits inversion and an addition of +1** as shown in the following example ...

- Example #1: -32767
    - +3276710 = 01111111 11111111~2~
    - inversion : 10000000 00000000~2~
    - addition + 1 : 10000000 00000001~2~ = -3276710
- Example #2: -1
    - 110 = 00000000 00000001~2~
    - inversion : 11111111 11111110~2~
    - addition + 1 : 11111111 11111111~2~= -1

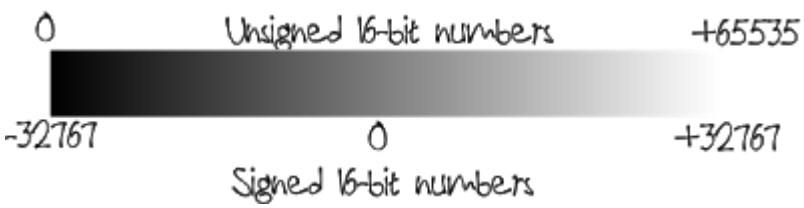
Thus, **signed 16-bit** numbers are comprised in the range of **[-32767;+32767]** whereas **unsigned 16-bit** numbers between **[0;65535]**.

2- Consequences for gray-level images

Now, we've seen the theory, what about gray-level images ?

If you import an image with the wrong 'sign', you get image with inverted areas as shown (right panel).

Let's go further. Create an 16-bit image 512x50 with a ramp background as shown. If you look at the pixel values by hovering the image, values are comprised between 0 and 65535 (ImageJ always creates unsigned 16-bit image).



If you save this ramp as a raw (16-bit unsigned, by default) image and import it as a '16-bit signed', you'll get the image following :

Now, if we compare the encoding of the signed and unsigned numbers, they are identical from 0 to +32767, but from +32768 to + 65535, depending of the encoding, they appear as negative (or positive) numbers (e.g. signed -110 has the same encoding as the unsigned 6553510). That explains the strange pattern of the ramp. The left part (medium gray to white) corresponds to the range **[0;+32767]** whereas the right part corresponds to negative numbers **[-32767;-1]**.

Just look at some key numbers in the ramp image...

- In unsigned 16-bit,
    - 00000000 00000000~2~ = 010
    - . . . . . .
    - 01111111 11111111~2 = 3276710
    - 10000000 000000002~ = 3276810
    - 10000000 00000001~2~ = 3276910
    - . . . . . .
    - 11111111 11111111~2~ = 6553510
- In signed 16-bit,
    - 00000000 00000000~2~ = 010
    - . . . . . .
    - 01111111 11111111~2~ = +3276710
    - 10000000 00000001~2~ = -3276710
    - . . . . . .
    - 11111111 11111111~2~ = -110

3- What about signed or unsigned 8-bit image?

In ImageJ, the only option called '8-bit' corresponds to **unsigned 8-bit numbers** (from 0 to 255) and there is no way to import a signed 8-bit image (-127 to +127).

Endianness

In this series of posts dedicated to image file, the endianness is one of the parameters available in the Import dialog box. What is this strange 'little-endian byte order'?

1- Little- and Big-endian? What does it mean?

In a file, the numbers are stored as **bytes**, thus, a 16-bit or 32-bit number must be splitted in **2 of 4 bytes** before being put in the file. There is two different ways to do this job. The first family of computers (e.g. ARM processors used by the smartphones) stores the bytes of a number from **left to right** as shown in the example of with the hexadecimal number 0807B7A016. This is called **Big Endian** because the first byte (byte 0 in Fig) is the most significant.

... whereas in the other family (x86 processor in every PC computers), the storage is done from **right to left** (Fig). This is called **Little Endian** and the first byte is now the least significant.

> **Note**: ImageJ runs on top of a virtual processor called 'Java Virtual Machine' and as indicated by its name acts as a virtual computer with its own virtual processor which belongs to the family of the 'Big Endian'.

3- When I import an unknown image file, how do I know if I add a problem of endianness?

First, keep in mind that **ImageJ is a big-endian program** and by default, import the raw file using a big endian byte order. Thus, if you know which computer (processor family) and which program was used to save your image, you can determine the byte order and decide to (un)check the 'litte-endian byte order' in the import dialog box.

Example #1: The computer belongs to the x86 family (e.g. an AMD or Intel processor) and the program was written in language C/C++, thus, your image is saved with a little-endian byte order.

Example #2: The computer belongs to the x86 family but the program is written in Java and consequently runs on top of a big endian Java Virtual Machine. Thus, the image will be saved as a big endian file...

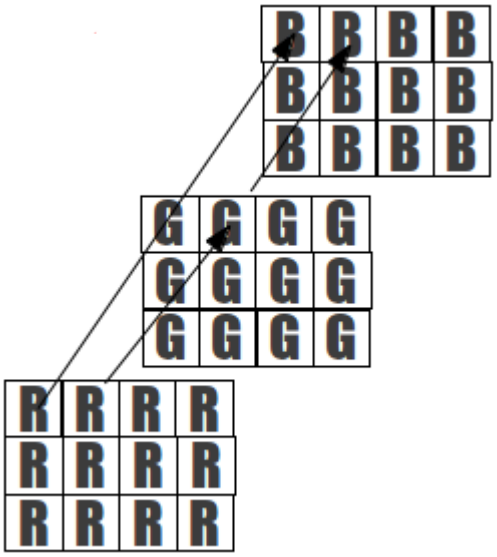
Second, you have no clue about the computer and/or the used program, you have to check both options. In most cases, when you use the wrong byte order, the image appears **noisy** and it is very difficult to recognize some features in the displayed image (Fig).

RGB file: Packed or Planar?

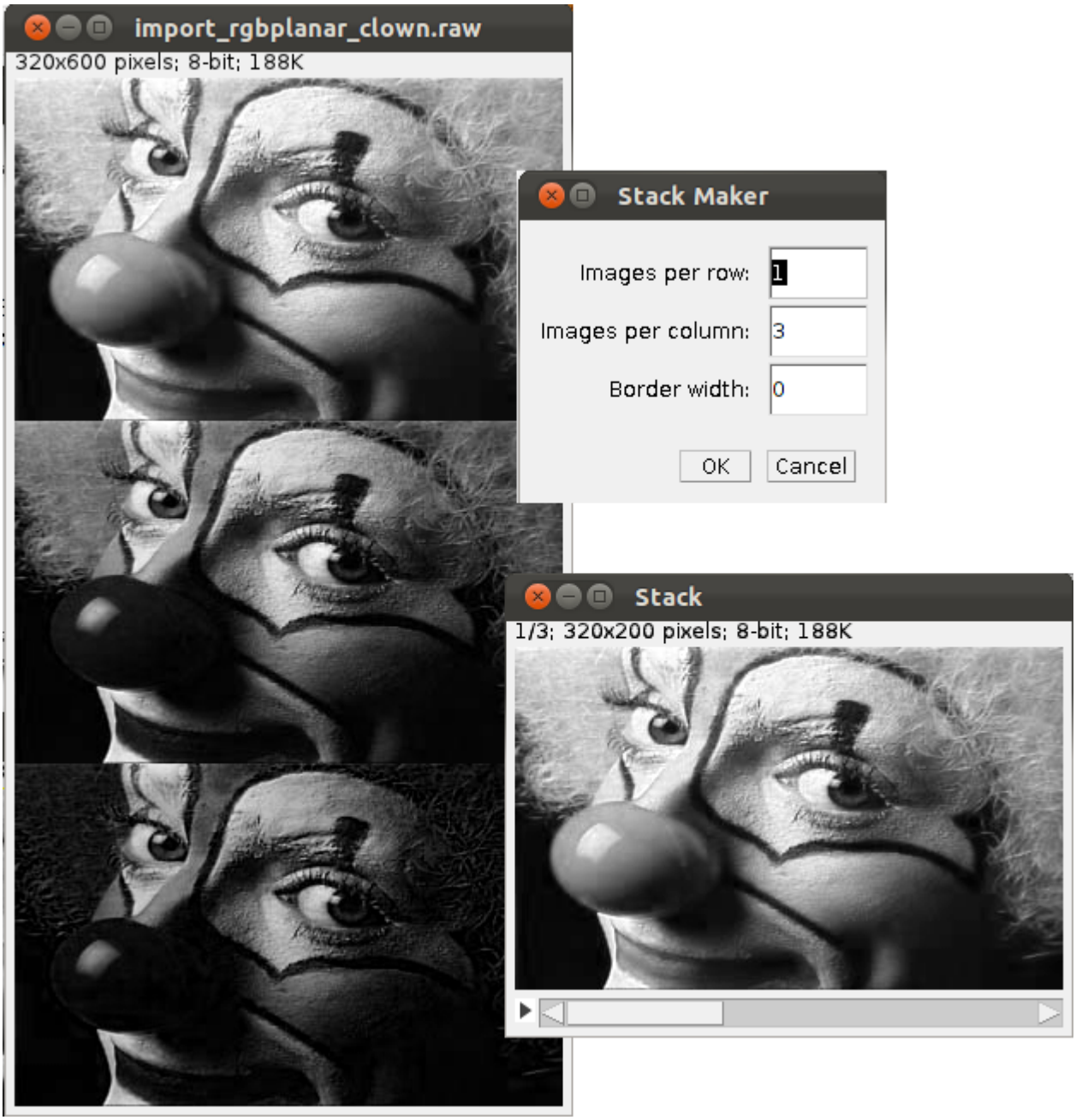
Save RGB color images requires specific strategies - compared to the gray-level images - to store the three red, green, and blue values defining a pixel. Two main strategies exist: **Packed and Planar**.

1- Planar format

If a color image is saved in the planar file format, the three red, green, and blue channels composing the pixels are **stored separately** in the file.



It is like a **stack of three slices** containing the red, green, and blue values for each pixel. Here is an example of planar file format (Fig).



2- Packed format

By default, ImageJ doesn't use the planar format to save color images, it uses the **packed format**. In Fig, the clown is visible, however he appears stretched in the horizontal direction.

This pattern is typical of a packed RGB format. Indeed, the red, green, and blue channels are **interleaved** in the file. Thus, the file contains a succession of red, green, and blue values corresponding to each pixel. Each value is stored as a **8-bit number**. Example of storage of two pixels in a packed format :

3- Other formats

ImageJ can directly import the packed 24-RGB and the planar 24-RGB formats. Other formats are available like the packed **24-BGR** (blue value, first and then, green, and finally red). Two other file formats can be read: **32-bit ARGB and 32-bit ABGR**, they are packed formats but the image contains a fourth 8-bit channel named **'alpha'** (A) dedicated to **opacity**. As transparency is not supported by ImageJ, this channel is skipped during the import.

IMAGE PROCESSING & PIPELINES

Image Enhancement

Because your scientific images are not perfect and the objects of interest are difficult to see, you have to **enhance** your images. Usually, three main defects can be fixed:

- Brightness and Contrast
- Non uniform illumination
- Noise

Enhancement: Brightness/Contrast

The brightness and contrast of an image can be enhanced by modifying the **transfer function** or playing with the pixel distribution of the **histogram**. Here is the first part with the transfer function...

1. Playing with the transfer function

The transfer function in ImageJ is defined by a straight line whose formula is:

> **y = Ax + B** with A is the slope and B is the y-intercept

Thus, two factors can be modified:

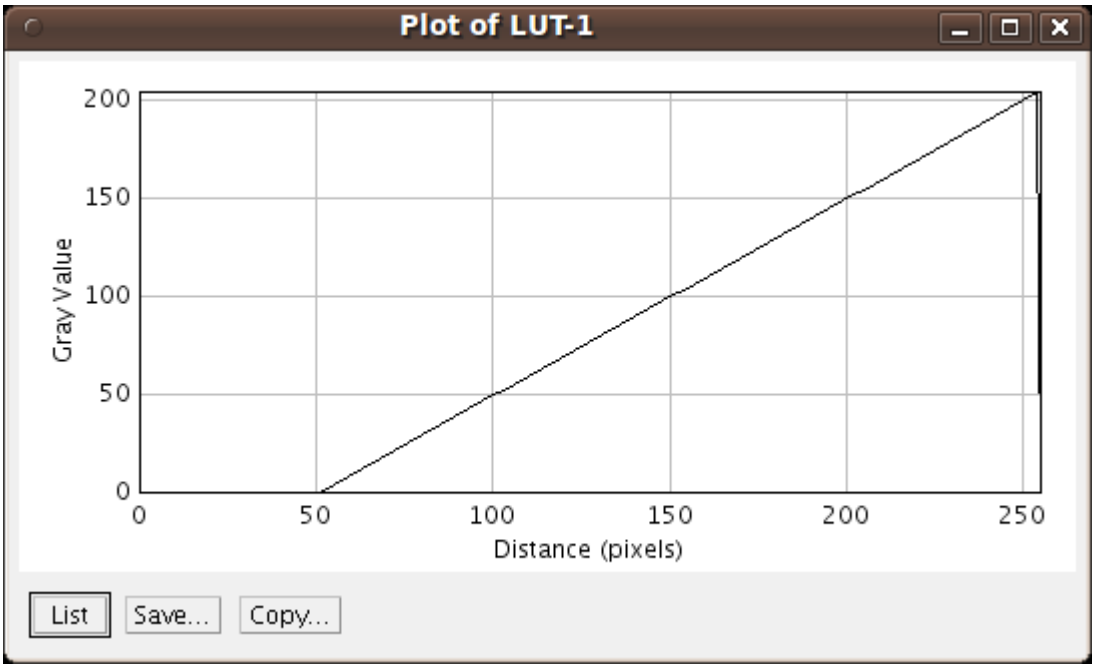
- The slope A
- The y-intercept B

Let's see the influence of these two parameters on the image :

1.1 The y-Intercept ~ The Brightness

If we modify the Y-intercept, the transfer function is now ...

y = x - 50

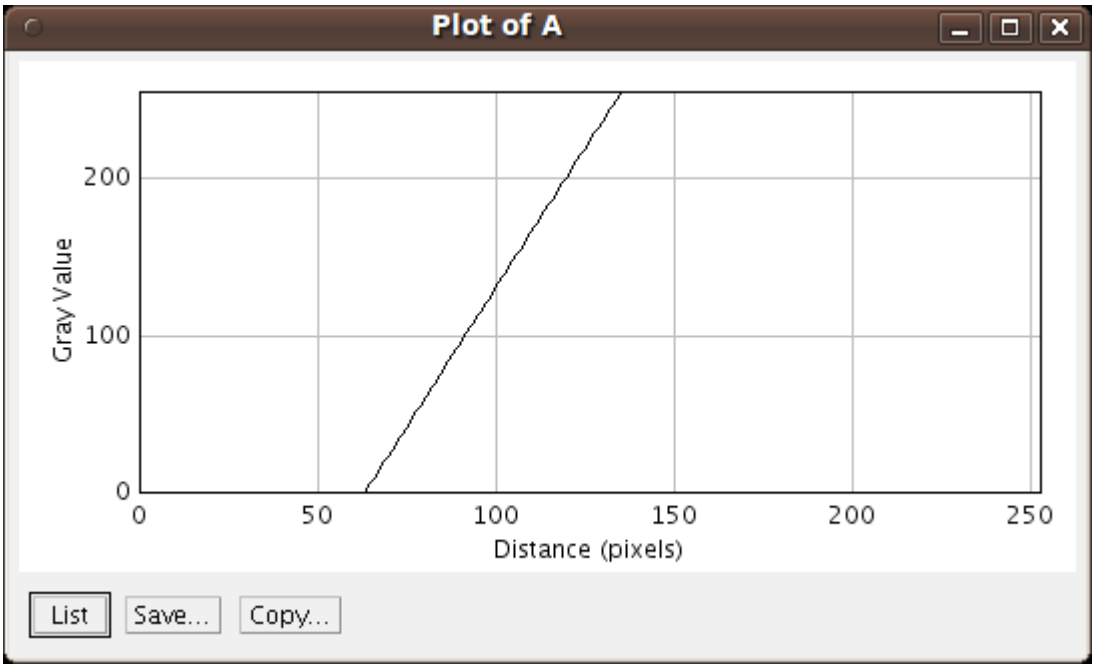


In this case, a pixel value of 50 in the input image is displayed as a **black (value = 0) pixel** in the output. Consequently, the LUT is now darker than the original as shown and the input pixel value 255 now is **equal to 200** (a light gray but not a white).

1.2. The Slope ~ The Contrast

In this case, the **slope is equal to 3.5** leading to a variation of grays (gradient) from black to white restrained in the range of [60;130].

y = 3.5x - 210



.. and the LUT appears as ...

In conclusion, the **brightness** corresponds to the y-intercept and the **contrast** to the slope of the transfer function. The brightness is the amount of light, and the contrast, the speed necessary to go from 0 to 255;

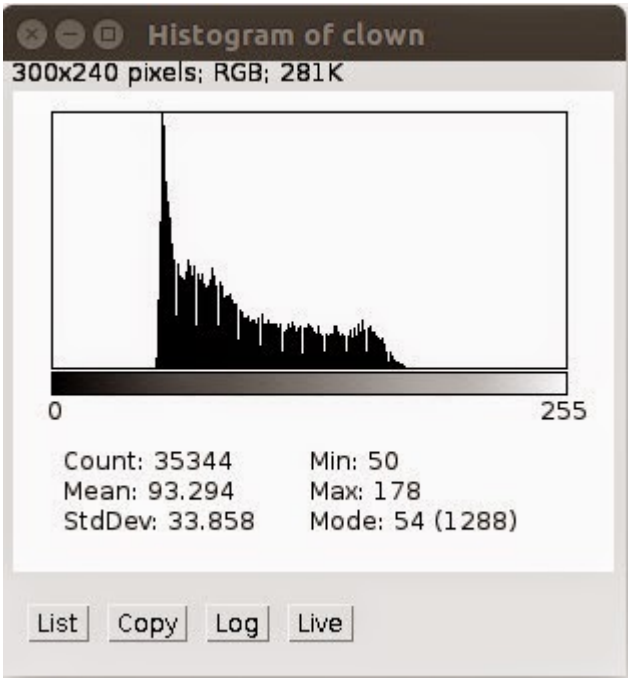
There are multiple others transfer functions possible:

$$Logarithme : Y = log(X)$$
$$Racine : Y = \sqrt{X}$$
$$Exponentielle : Y = e^{X}$$
$$Sigmoide : Y = K\frac{1}{1+ae^{-rX}}$$

2. Playing with the histogram

The previous method based on the transfer function is powerful but you have to manually define the slope and the y-intercept of the transfer function. It is faster to directly **modify the histogram**.

Here is our test image...



2.1. Normalization of the histogram

In the histogram, there is no pixel value in the range of [0-50] and of [178-255]. It can be interesting to **use all the pixel values** available in a 8-bit image and thus to stretch the histogram between its extreme values (0 and 255) by **interpolating** all the values between 0 and 255. This operation is called the **normalization** of the histogram.

2.2. Equalization of the histogram

An ideal transfer function based on the **cumulative histogram**. One disvantage is that it ehance the noise of the image, it'snot interesting for scientific images.

Enhancement: Non uniform illumination

In the Image Enhancement series, the **non uniform illumination** is the second main defect that you must correct in your images.

1. With a reference image

If you are using a scientific device like a microscope (optical or electron), all the environmental parameters are well known (no clouds, fog, sun,etc.). In this case, you can record a blank image at the beginning of your

session of images collection. A **blank image** or reference image is an image **without samples**. Thus, if there is a problem of illumination, this image **contains the defect**.

> **Note**: With this technique, we assume that the defect is constant during the session.

2. Without a reference image

Unfortunately, the reference image is outdated or you didn't backup this image or you didn't know where this image is located in the hard disk. So, the result is: **You have no reference image!**

All the techniques available tried to model the background image and then to subtract it to the images.

- Use filters like the 'Rolling Ball' or the 'Top Hat' filters.
- Model the background with a polynomial function.
- Use of Mathematical Morphology method.
- Removal of low frequencies in frequency - Fourier - space.

In ImageJ, only the 'Rolling Ball' filter is implemented to subtract the background.

Noise

Intrinsic Noise

Image noise is random (not present in the object imaged) **variation of brightness or color information** in images, and is usually an aspect of electronic noise. It can be produced by the sensor and circuitry of a scanner or digital camera. Image noise can also originate in film grain and in the unavoidable shot noise of an ideal photon detector. Image noise is an undesirable by-product of image capture that adds spurious and extraneous information. (random)

External Noise

- Dust
- "Moves" of sensor or of the object of interest
- Perturbation of atmosqpheric parameters (clouds, humidity, fog ...)
- Perturbations electromagnetic (can direcly act on sensor)

Noise due to digital processing

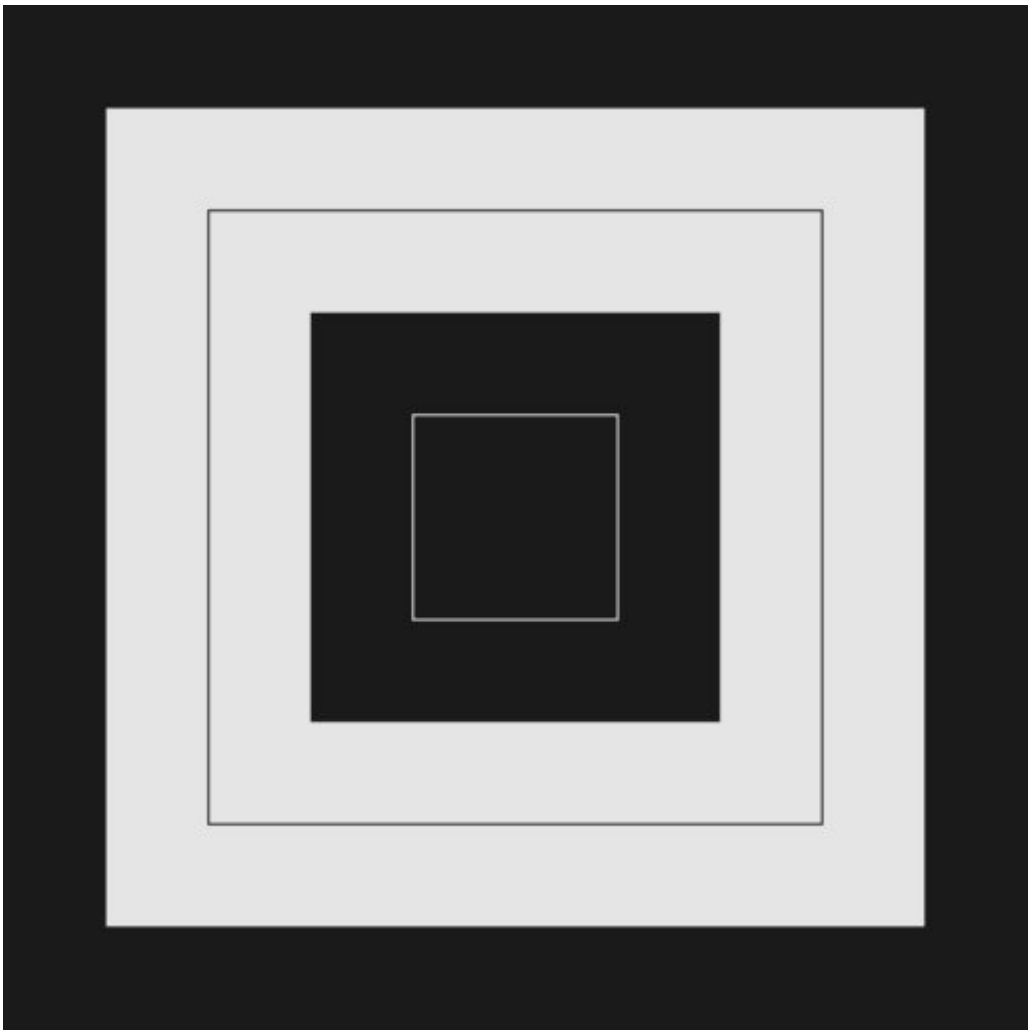
Different type of noise

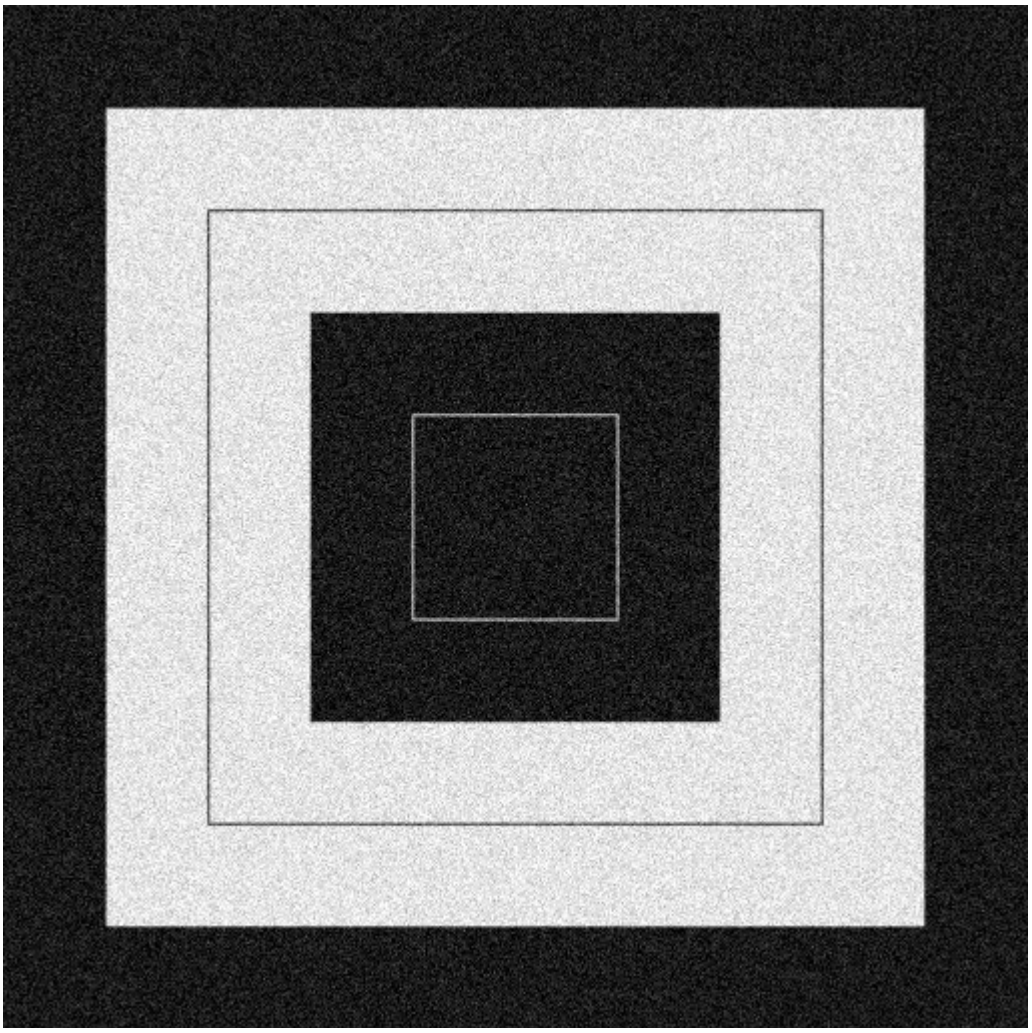
1 Gaussian noise

Gaussian noise is statistical noise having a **probability density function (PDF)** equal to that of the normal distribution, which is also known as the **Gaussian distribution**. In other words, the values that the noise can take on are Gaussian-distributed (in histogram).
Principal sources of Gaussian noise in digital images arise **during acquisition** e.g. sensor noise caused by poor illumination and/or high temperature, and/or transmission

In digital image processing Gaussian noise can be reduced using a **spatial filter**, though when smoothing an image, an undesirable outcome may result in the blurring of fine-scaled image edges and details because

they also correspond to blocked high frequencies. **Conventional spatial filtering techniques** for noise removal include: mean (convolution) filtering, median filtering and Gaussian smoothing.



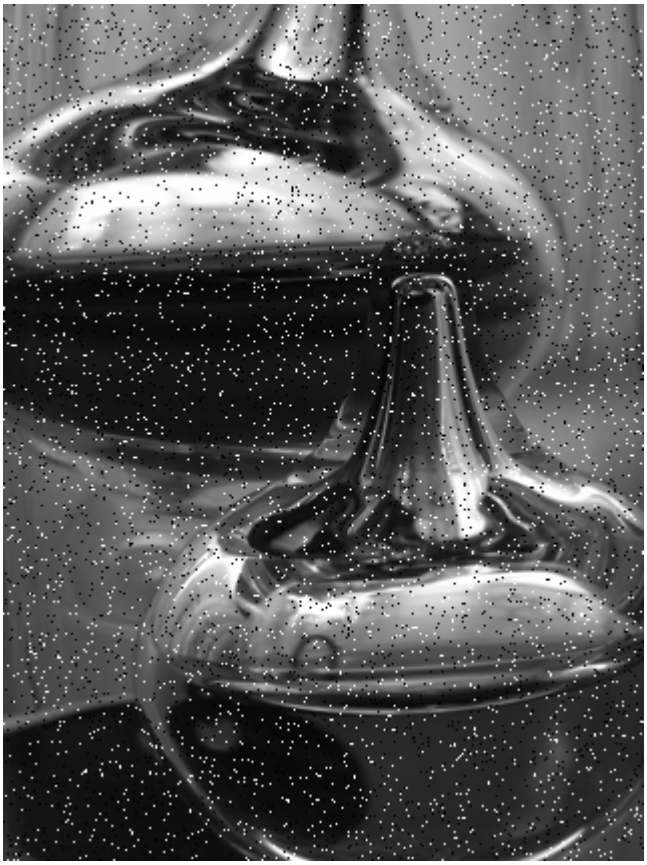


> By Me - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10114819

2 Salt and Pepper

Salt-and-pepper noise is a form of noise sometimes seen on images. It is also known as **impulse noise**. This noise can be caused by sharp and sudden disturbances in the image signal. An image containing salt-and-pepper noise will have **dark pixels** in bright regions and **bright pixels** in dark regions. This type of noise can be caused by analog-to-digital converter errors, bit errors in transmission, etc. It can be mostly eliminated by using dark frame subtraction, median filtering and interpolating around dark/bright pixels (extreme value of pixels in histogram).

Dead pixels in an LCD monitor produce a similar, but non-random, display.



> By User Markome on en.wikipedia, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1352731

3 Additive noise

Soit une image non bruitée R et I la même image avec un bruit **additif** A, chaque pixel j
$$I_j=A_j+R_j$$
Où $A_j$est une variable **aléatoire** de moyenne égale à 0

4 Multiplicative noise

Soit une image non bruitée R et I la même image avec un bruit **additif** B, chaque pixel j
$$I_j=B_j*R_j$$

Où $B_j$est une variable **aléatoire** de moyenne égale à 0
More noise in Wiki

> **Note for each type** : In either case, the noise at different pixels can be either correlated or uncorrelated; in many cases, noise values at different pixels are modeled as being independent and identically distributed, and hence uncorrelated.

Noise reduction

Noise reduction is the process of **removing noise** from a signal.
All recording devices, both analog and digital, have traits that make them susceptible to noise. Noise can be random or white noise with no coherence, or coherent noise introduced by the **device's mechanism or processing algorithms**.

Images taken with both digital cameras and conventional film cameras will pick up noise from a variety of sources. Further use of these images will often require that the noise be (partially) removed – for aesthetic purposes as in artistic work or marketing, or for practical purposes such as computer vision.

1 Removal

1.1 Mean of several images

A good way to reduce noise is taking several pictures and averaging pixels from all the pictures to reduce the random part of the image

1.2 Tradeoffs

In selecting a noise reduction algorithm, one must weigh several factors:

- the available **computer power and time available**: a digital camera must apply noise reduction in a fraction of a second using a tiny onboard CPU, while a desktop computer has much more power and time
- whether **sacrificing some real detail** is acceptable if it allows more noise to be removed (how aggressively to decide whether variations in the image are noise or not)
- the **characteristics** of the noise and the detail in the image, to better make those decisions

1.3 Chroma and luminance noise separation

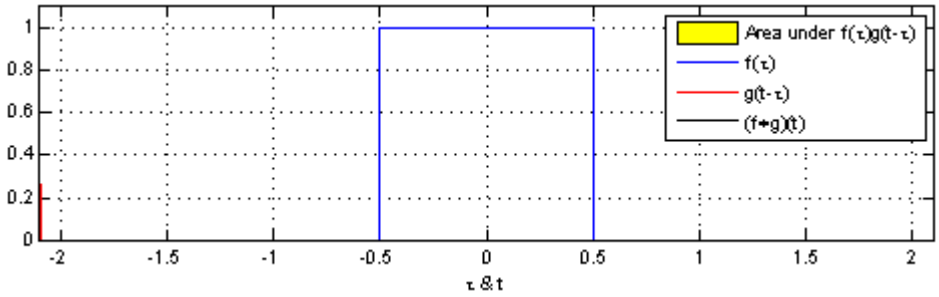
In real-world photographs, the highest spatial-frequency detail consists mostly of variations in **brightness** ("luminance detail") rather than variations in **hue** ("chroma detail"). Since any noise reduction algorithm should attempt to remove noise without sacrificing real detail from the scene photographed, one risks a **greater loss** of detail from luminance noise reduction than chroma noise reduction simply because most scenes have little high frequency chroma detail to begin with.

Most dedicated noise-reduction computer software allows the user to control chroma and luminance noise reduction separately.

1.4 Linear smoothing filters

One method to remove noise is by **convolving** the original image with a **mask** that represents a low-pass filter or smoothing operation.

> **Convolution**
> In mathematics (and, in particular, functional analysis) convolution is a mathematical operation on **two functions** (f and g); it **produces a third function**, that is typically viewed as a modified version of one of the original functions, giving the integral of the pointwise multiplication of the two functions as a **function of the amount** that one of the original functions is translated. Convolution is similar to cross-correlation. It has applications that include probability, statistics, computer vision, natural language processing, **image and signal processing**, engineering, and differential equations.



> By Convolution_of_box_signal_with_itself.gif: Brian Ambergderivative work: Tinos (talk) - Convolution_of_box_signal_with_itself.gif, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11003835

In image processing, the filter is therefore a **convolution mask** (*kernel*) which is basically a small matrix by which every pixel on the original image is multiplied

Kernel calculation

For example, the **Gaussian mask** comprises elements determined by a Gaussian function. This convolution brings the value of each pixel into closer harmony with the values of its neighbors. In general, a smoothing filter sets each pixel to the **average value**, or a weighted average, of itself and its nearby neighbors; the Gaussian filter is just one possible set of weights.

Smoothing filters tend to **blur** an image, because pixel intensity values that are significantly higher or lower than the surrounding neighborhood would **"smear"**(salir) across the area. Because of this blurring, linear filters are seldom (rarement) used in practice for noise reduction; they are, however, often used as the basis for **nonlinear noise reduction filters**.

Edge managment

- Smaller resulting images
- Edges of the image identical to those of the original images

1.5 Nonlinear filters

A **median filter** is an example of a non-linear filter and, if properly designed, is very good at preserving image detail. To run a median filter:

- consider each pixel in the image

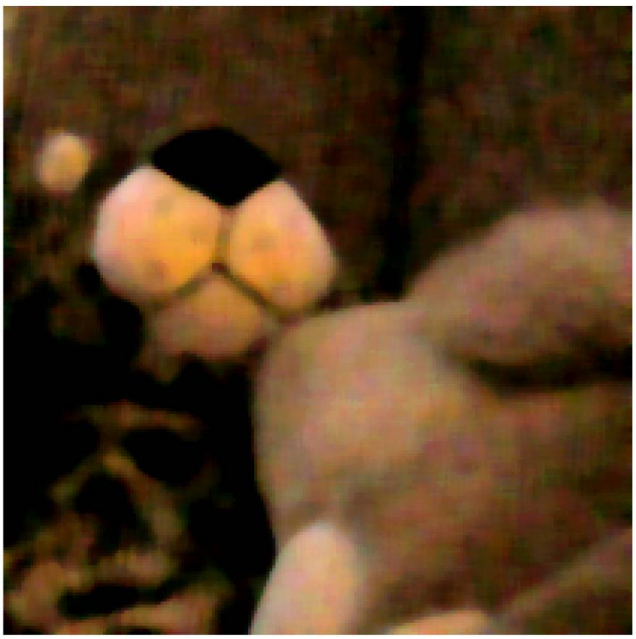
- sort the neighbouring pixels into order based upon their intensities
- replace the original value of the pixel with the median value from the list

A median filter is a **rank-selection (RS) filter**, a particularly harsh member of the family of rank-conditioned rank-selection (RCRS) filters; a much milder member of that family, for example one that selects the closest of the neighboring values when a pixel's value is external in its neighborhood, and leaves it unchanged otherwise, is sometimes preferred, especially in photographic applications.

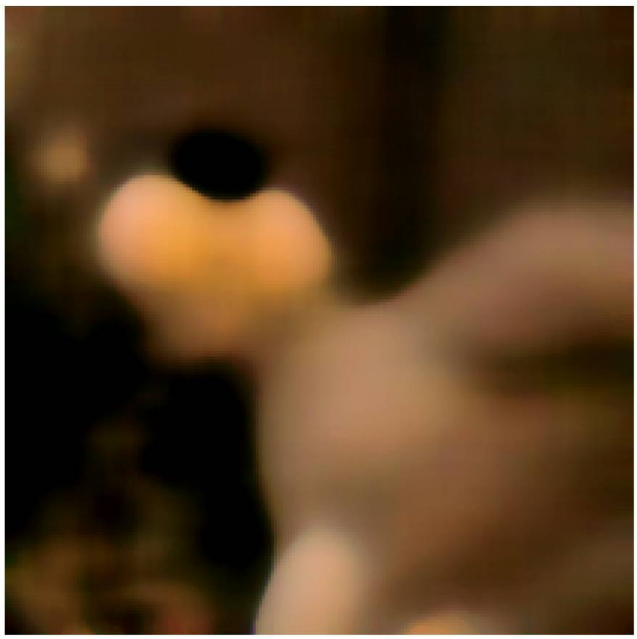
Median and other RCRS filters are **good at removing salt and pepper noise** from an image, and also cause relatively **little blurring of edges**, and hence are often used in computer vision applications.

original image

1px median filter



3px median filter



10px median filter

> By Debivort at en.wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17001283

1.6 Anisotropic diffusion

- Bilateral filter (extension of Gaussian filter)
- Filtre à diffusion anisotropique (based on partial differential equations *Too slow for us because too complicated*)

In image processing and computer vision, anisotropic diffusion, also called **Perona–Malik diffusion**, is a technique aiming at reducing image noise without removing significant parts of the image content, typically edges, lines or other details that are important for the interpretation of the image. Anisotropic diffusion resembles the process that creates a **scale space**, where an image generates a parameterized family of successively more and more blurred images based on a diffusion process. Each of the resulting images in this family are given as a **convolution** between the image and a 2D isotropic Gaussian filter, where the width of the filter increases with the parameter. This diffusion process is a **linear and space-invariant** transformation of the original image. Anisotropic diffusion is a generalization of this diffusion process: it produces a **family of parameterized images**, but each resulting image is a combination between the original image and a filter that depends on the local content of the original image. As a consequence, anisotropic diffusion is a **non-linear and space-variant** transformation of the original image

1.7 Non-local means

Non-local means is an algorithm in image processing for image denoising. Unlike **"local mean"** filters, which take the mean value of a group of pixels surrounding a target pixel to smooth the image, non-local means filtering takes a **mean of all pixels in the image**, weighted by how similar these pixels are to the target pixel. This results in much greater post-filtering clarity, and less loss of detail in the image compared with local mean algorithms.

Références

Fourier transform

Common Names: Fourier Transform, Spectral Analysis, Frequency Analysis

The Fourier Transform is an important image processing tool which is used to **decompose** an image into its **sine and cosine components**. In fact every complex signal can be decomposed into addition of sine and cosine adjusted with coefficiants, giving an linear application : $complex_{signal}=a_1\cdot \cos{x} + a_2\cdot \cos{2x} + a_3\cdot \cos{3x} + ...$The output of the transformation represents the image in the Fourier or **frequency domain**, while the input image is the **spatial domain** equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

When you are close to the center of the fourrier transformed image, you look at the low frequencies, and when you're near from the edges, you're looking at the high frequences, so the small details.

Therefore, to reduce noise, you can use Fourrier Transform (**FT**), then add a circular mask to hide high frequencies and then invert the **FT** to get the image without the noise.

FT are the best way to retrieve periodic noise.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression.

See more

Pass-filter

1 High-pass filter

A high-pass filter (HPF) is an **electronic filter** that passes signals with a frequency **higher** than a certain cutoff frequency and attenuates signals with frequencies **lower** than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design. A high-pass filter is usually modeled as a **linear time-invariant system**. It is sometimes called a low-cut filter or bass-cut filter. High-pass filters have many uses, such as blocking DC from circuitry sensitive to non-zero average voltages or radio frequency devices.

2 Low-pass filter

A low-pass filter (LPF) is a filter that passes signals with a frequency **lower** than a certain cutoff frequency and attenuates signals with frequencies **higher** than the cutoff frequency. The exact frequency response of the filter depends on the filter design. The filter is sometimes called a high-cut filter, or treble-cut filter in audio applications. A low-pass filter is the complement of a high-pass filter.

Filter designers will often use the low-pass form as a **prototype filter**. That is, a filter with unity **bandwidth and impedance**. The desired filter is obtained from the prototype by scaling for the desired bandwidth and impedance and transforming into the desired bandform (that is low-pass, high-pass, band-pass or band-stop).

3 Band-pass filter

A band-pass filter (also spelled bandpass) (BPF) is a device that passes frequencies within a certain range and rejects **(attenuates)** frequencies **outside that range**.

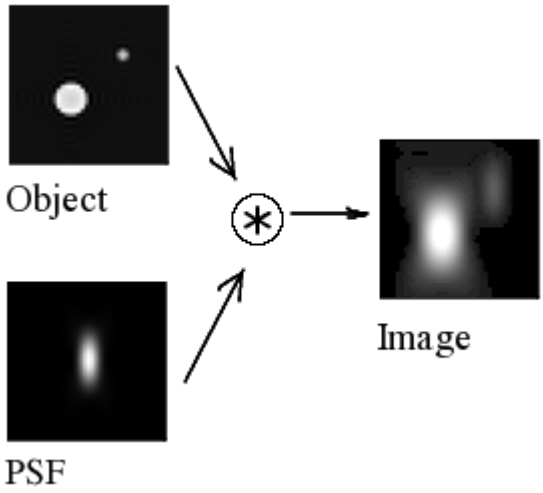
The Point Spread Function (PSF)

The point spread function (PSF) describes the **response of an imaging system** to a **point source** or point object. A more general term for the PSF is a **system's impulse response**, the PSF being the impulse response of a focused optical system. The PSF in many contexts can be thought of as the extended blob in an image that represents an **unresolved object**. In functional terms it is the spatial domain version of the optical transfer function of the imaging system.

It is a useful concept in **Fourier optics**, astronomical imaging, medical imaging, electron microscopy and other imaging techniques such as 3D microscopy (like in confocal laser scanning microscopy) and fluorescence microscopy. The **degree of spreading** (blurring) of the point object is a measure for the **quality** of an imaging system.

In non-coherent imaging systems such as fluorescent microscopes, telescopes or optical microscopes, the image formation process is linear in power and described by linear system theory. This means that when two objects A and B are imaged simultaneously, the result is equal to the **sum of the independently imaged objects**. In other words: the imaging of A is unaffected by the imaging of B and vice versa, owing to the non-interacting property of photons.

The image of a complex object can then be seen as a convolution of the true object and the PSF. However, when the detected light is coherent, image formation is linear in the complex field. Recording the intensity image then can lead to cancellations or other non-linear effects.

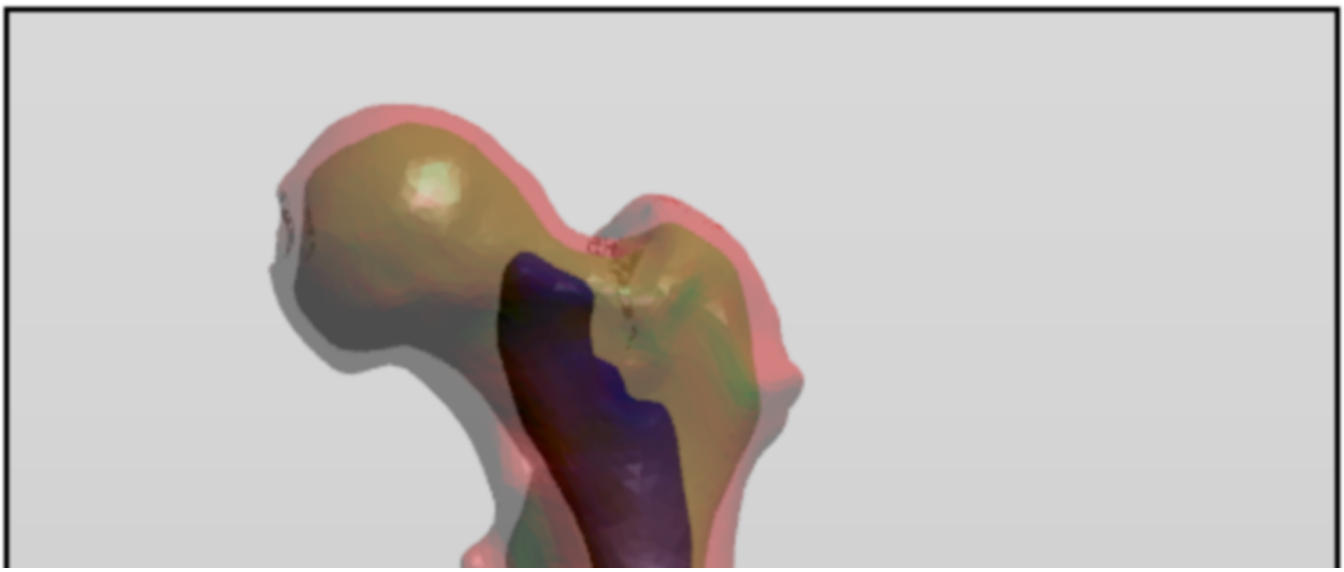


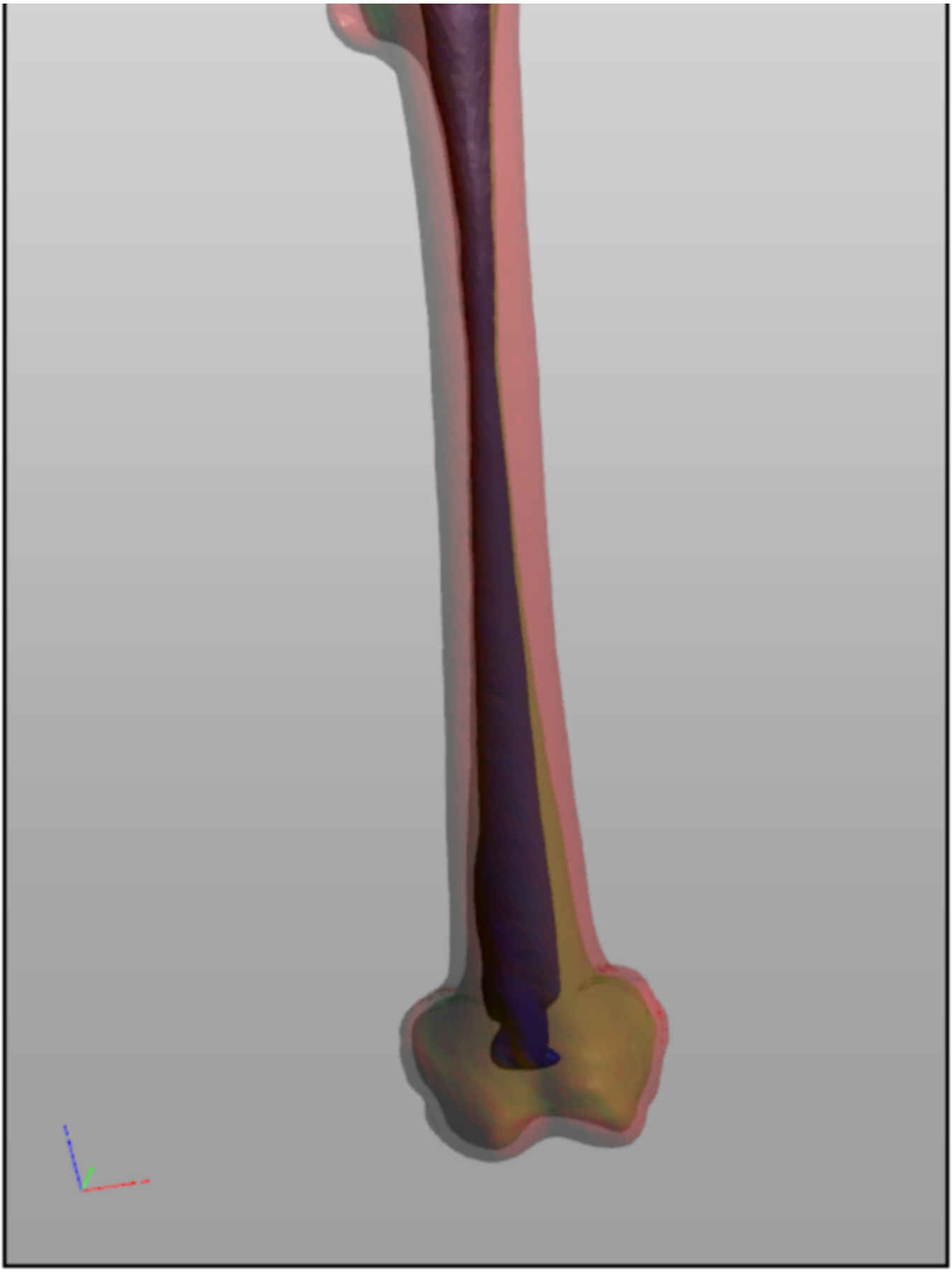
> By Default007 - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=877065

Image segmentation

In computer vision, image segmentation is the process of **partitioning** a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to **locate objects and boundaries** (lines, curves, etc.) in images. More precisely, image segmentation is the process of **assigning a label** to every pixel in an image such that pixels with the same label share certain characteristics.

The result of image segmentation is a **set of segments** that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s). When applied to a stack of images, typical in medical imaging, the resulting contours after image segmentation can be used to create 3D reconstructions with the help of interpolation algorithms like Marching cubes.





Threshold

The simplest method of image segmentation is called the **thresholding method**. This method is based on a clip-level (or a threshold value) to turn a **gray-scale image into a binary image**. There is also a balanced histogram thresholding.

The key of this method is to select the threshold value (or values when multiple-levels are selected). Several popular methods are used in industry including the maximum entropy method, Otsu's method (maximum variance), and **k-means clustering**.

Otsu's method (ImageJ)

Finir avec une image binaire : "background","Zone of interest".

Clustering methods

The K-means algorithm is an iterative technique that is used to partition an image into **K clusters**. The basic algorithm is :

- Pick K cluster centers, either randomly or based on some heuristic method, for example K-means++
- Assign each pixel in the image to the cluster that minimizes the distance between the pixel and the cluster center
- Re-compute the cluster centers by averaging all of the pixels in the cluster
- Repeat steps 2 and 3 until convergence is attained (i.e. no pixels change clusters)

Source

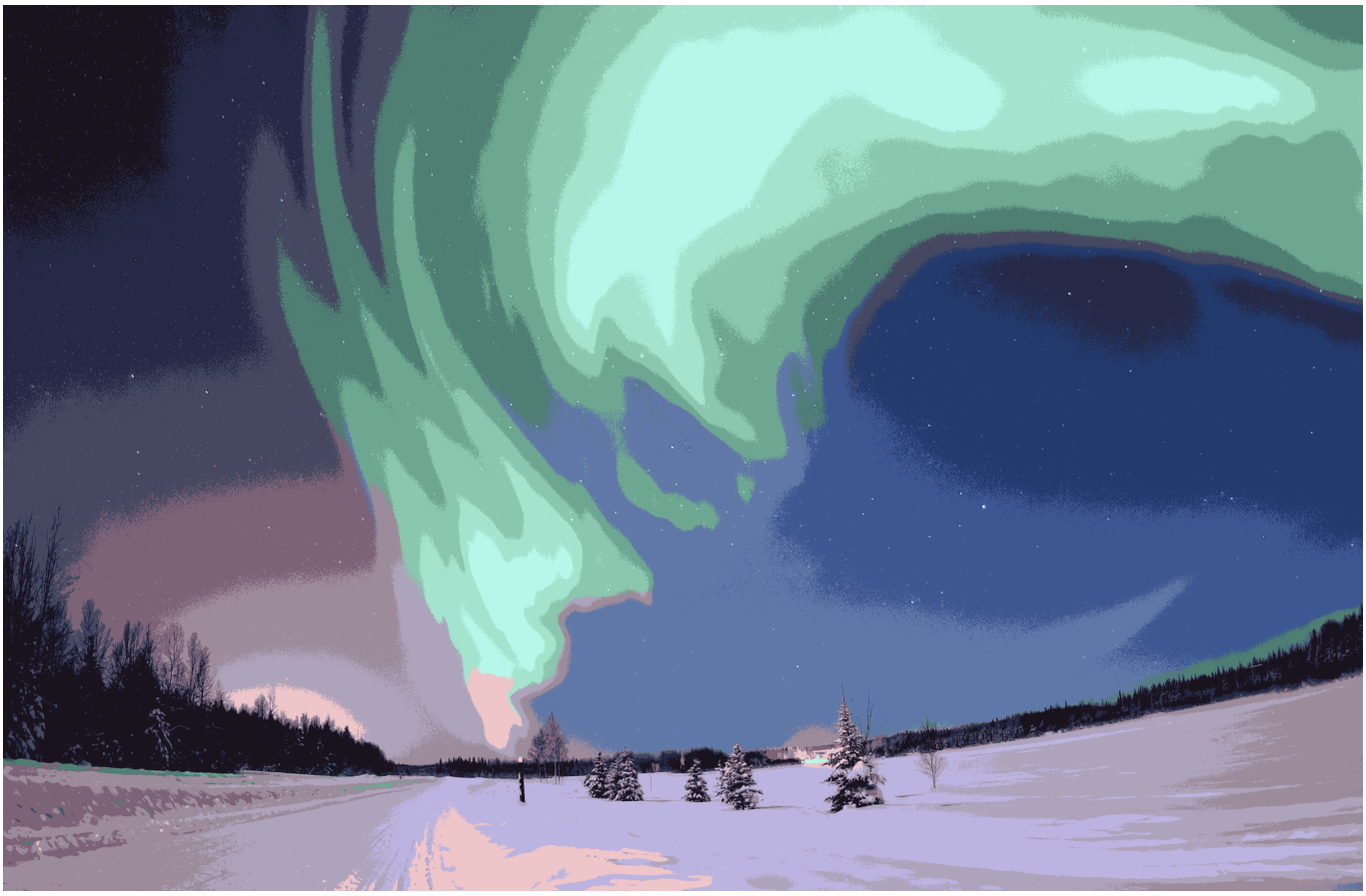


Image after running k-means with k = 16.

> By Senior Airman Joshua Strang, derivative work by King of Hearts - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=22790024

In this case, **distance** is the squared or absolute difference between a **pixel and a cluster center**. The difference is typically based on pixel color, intensity, texture, and location, or a weighted combination of these factors. K can be selected manually, randomly, or by a heuristic. This algorithm is guaranteed to **converge**, but it may not return the optimal solution. The quality of the solution depends on the initial set of clusters and the value of K.

Histogram-based methods

Histogram-based methods are **very efficient** compared to other image segmentation methods because they typically require only **one pass** through the pixels. In this technique, a histogram is computed from all of the pixels in the image, and the **peaks and valleys** in the histogram are used to **locate the clusters** in the image. Color or intensity can be used as the measure.

A refinement of this technique is to **recursively** apply the **histogram-seeking method** to clusters in the image in order to divide them into smaller clusters. This operation is repeated with smaller and smaller clusters until no more clusters are formed.

One disadvantage of the histogram-seeking method is that it may be difficult to identify significant peaks and valleys in the image.

Histogram-based approaches can also be quickly adapted to apply to **multiple frames**, while maintaining their single pass efficiency. The histogram can be done in multiple fashions when multiple frames are considered. The same approach that is taken with **one frame** can be **applied to multiple**, and after the

results are **merged**, peaks and valleys that were previously difficult to identify are more likely to be distinguishable. The histogram can also be applied on a **per-pixel basis** where the resulting information is used to determine the **most frequent color** for the pixel location. This approach segments based on active objects and a static environment, resulting in a different type of segmentation useful in video tracking.

Edge detection

Edge detection includes a variety of mathematical methods that aim at **identifying points** in a digital image at which the **image brightness** changes sharply or, more formally, has **discontinuities**. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed **edges**. The same problem of finding discontinuities in one-dimensional signals is known as **step detection** and the problem of finding signal discontinuities over time is known as **change detection**. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction.

The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. It can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to:

- discontinuities in depth,
- discontinuities in surface orientation,
- changes in material properties and
- variations in scene illumination.

Canny edge detection applied to a photograph

> By JonMcLoone at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=44894482

Mathematical morphology

Mathematical morphology (MM) is a great toolbox for image processing. In ImageJ, most of these operations are available for binary images and are often used to prepare an image before analysis. Moreover, the same operations can be used for gray-level images offering new functionalities in terms of filtering, segmentation,etc.

Playing with binary images

In a binary image, the pixels are in two states: ON or OFF. Better than saying ON and OFF, the **TRUE and FALSE** keywords are used.

A binary image

What are TRUE and FALSE in a computer ?
In programming languages, a condition returns TRUE or FALSE but behind these two keywords, there are numbers where TRUE equal to 1 (one) and FALSE to 0 (zero)

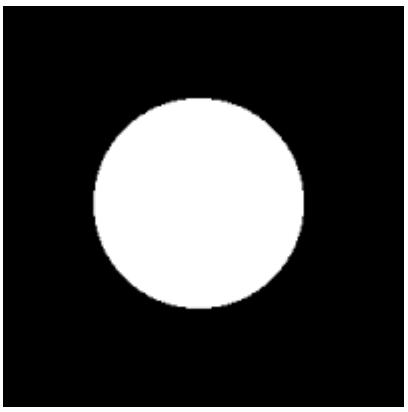
Boolean

One of the basic functions for binary images (containing TRUE and FALSE pixels) are the **boolean operators** useful to manipulate this kind of images.

Operators

AND, OR, XOR operators

Here 2 images :



Now if we apply the boolean operator on those images :

To understand how it works, the simplest way is to fill an **array** where all the combinations of TRUE (value = 255) and FALSE (value = 0) pixels are met :

	A	B	C	D	E	F	G	H	I
	XY-coords	image A	image B	A AND B					
	(131;150)	TRUE	TRUE	TRUE					
	(134;67)	TRUE	FALSE	FALSE		XY-coords	image A	image B	A OR B
	(140;113)	FALSE	TRUE	FALSE		(131;150)	TRUE	TRUE	TRUE
	(20;20)	FALSE	FALSE	FALSE		(134;67)	TRUE	FALSE	TRUE
						(140;113)	FALSE	TRUE	TRUE
						(20;20)	FALSE	FALSE	FALSE
			XY-coords	image A	image B	A XOR B			
			(131;150)	TRUE	TRUE	FALSE			
			(134;67)	TRUE	FALSE	TRUE			
			(140;113)	FALSE	TRUE	TRUE			
			(20;20)	FALSE	FALSE	FALSE			

For example, with the **AND operator**, the pixel of XY-coordinates (131;150) is TRUE in 'A', and TRUE in 'B', the **resulting image A AND B** has a TRUE pixel value at (131;150). Repeat the same approach with the three other cases (TRUE/FALSE, FALSE/TRUE, and FALSE/FALSE).

To summarize,

- AND: The resulting pixel is TRUE if and only if a pixel of 'A' is TRUE *and* the other of 'B' is TRUE.
- OR: The resulting pixel is TRUE if and only if a pixel is TRUE *or* the other is TRUE *or* both are TRUE.
- XOR (for exclusive OR); The resulting pixel is TRUE if and only if a pixel is TRUE *or* the other is TRUE but **not when the two pixels are TRUE**.

The boolean operators are useful when one of the **image acts as a mask** to remove or highlight objects of interest.

Black and White, TRUE or FALSE?

When playing with **binary images** in ImageJ, that's not really simple to remember if TRUE corresponds to white or black (specially, if you use a lot of different binary functions).

In ImageJ, there are **four** different families of operations working with binary images:

- Image Calculator with boolean operators (AND, OR, and XOR).
- Thresholding in Image > Adjust> Threshold
- Morphological operations in Process > Binary
- Image analysis in Analyze > Analyze Particles...

Unfortunately, **they don't use the same standards**.

In **boolean operators**, FALSE corresponds to black pixels (pixel value of 0) and TRUE to white (255). In contrast, the other families **(morphology and analysis)** assume that a pixel value of 0 is TRUE and 255 is FALSE.

Basic operators

In MM, the vocabulary used is a little bit different than those used for linear filters. Here, the image is seen as various pixels sets and the two basic operators (erosion and dilation) correspond to the combination of a structuring element with the image. Even though the mathematics underlying these processes are different, this is rather equivalent to a convolution with a mask (or kernel).

For sake of clarity, we consider that white pixels (value of 255) are TRUE.

Erosion

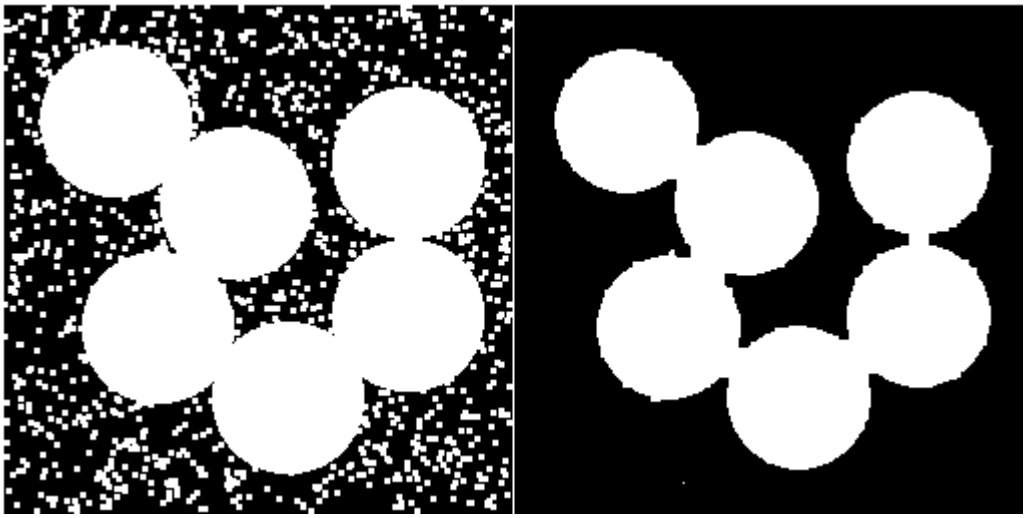
In MM, the **erosion** corresponds to the **minimum pixel value** found in the structuring element assuming that the FALSE (value of 0) corresponds to the **background** and TRUE to the object(s) of interest.

Erosion with ImageJ. A) Original image. B) Eroded image obtained after 3 cycles.

1 Use

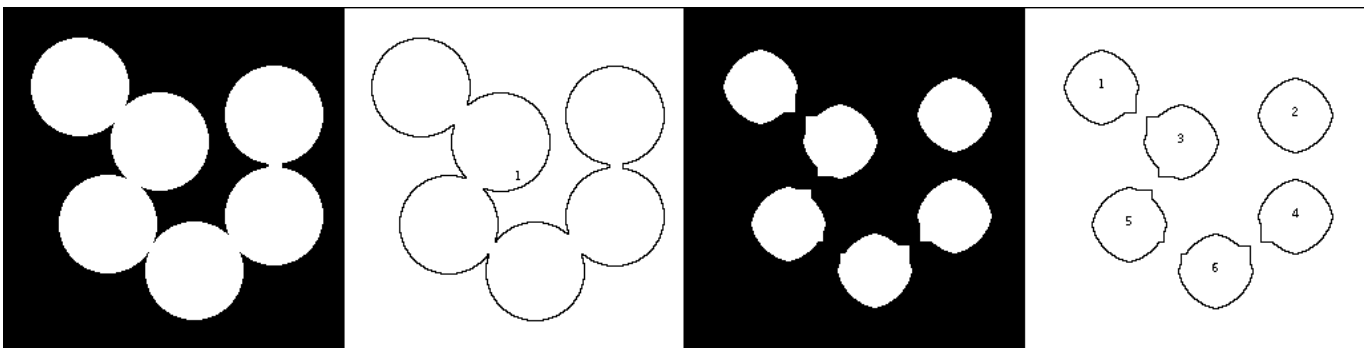
Erosion is often used before a** particle analysis** to clean up a segmented (thresholded) image or to separate touching particles.

1.1 Removing noise or unwanted particles



Removing noisy particles by successive cycles of erosion.

1.2 Separating touching objects of interest

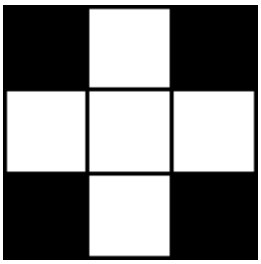


Use of erosion to separate objects of interest. A) Original image. B) Result of the Particles analysis. Only one

object (region) is counted. C) Eroded image (several cycles of erosion were required). D) Six objects are now counted by the Particles analysis

2 Erosion calculated with a minimum filter

The **binary ImageJ** erosion operator is rather limited because you can't modify the shape of the structuring element. This limitation can be bypassed by using the **minimum filter** and its associated masks.



The 'cross' structuring element is available with a filter radius of 0.5 and the 3x3 square with a radius of 1.0.

2.1 Erosion of binary images ...

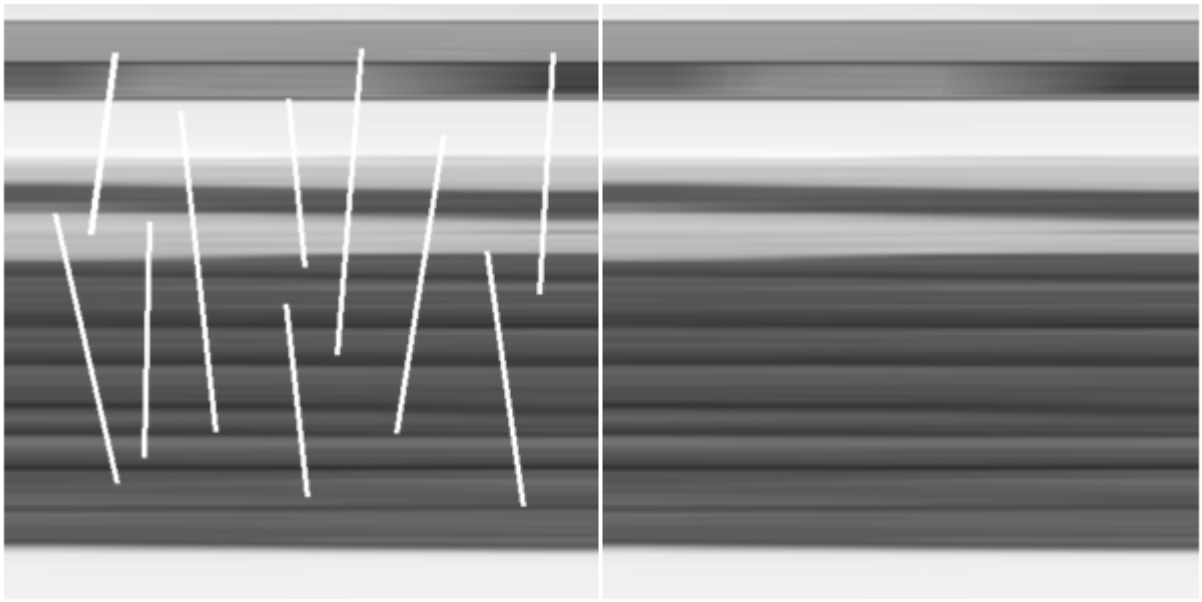
To remove unwanted particles, this approach is more convenient. Indeed, you don't need to repeat n times the erosion process, just estimate the radius of these particles and run a **minimum filter** whose **radius is a little bit larger** than the measured defects, these particles will disappear.

2.2 ... and of gray-level images

The other main advantage to replace the erosion by a minimum filter is the ability of **working with gray-level images**. In this case, the erosion allows to remove the **lightest image**.

Original image Lena sprinkled with white dots. B) Eroded image using a 'cross' structuring element (radius = 0.5 in minimum filter).

Another example with a gray-level image... composed of a horizontal blurred gray-level area. In this case, we plan to remove the vertical lines and design a horizontal structuring element. In this case, we'll use erosion with other shapes of structuring elements.



A) Original image. B) Eroded image calculated with a horizontal 1x20 structural element. By subtraction, you can get the vertical lines.

Erosion is one of the two basic operators of MM.

Dilation

Morphological dilation is the second basic operator in MM and the **counterpart** of the erosion.

In Wikipedia, this is defined by the union between the image and the structuring element. And this definition can be extended to the gray-level images as:

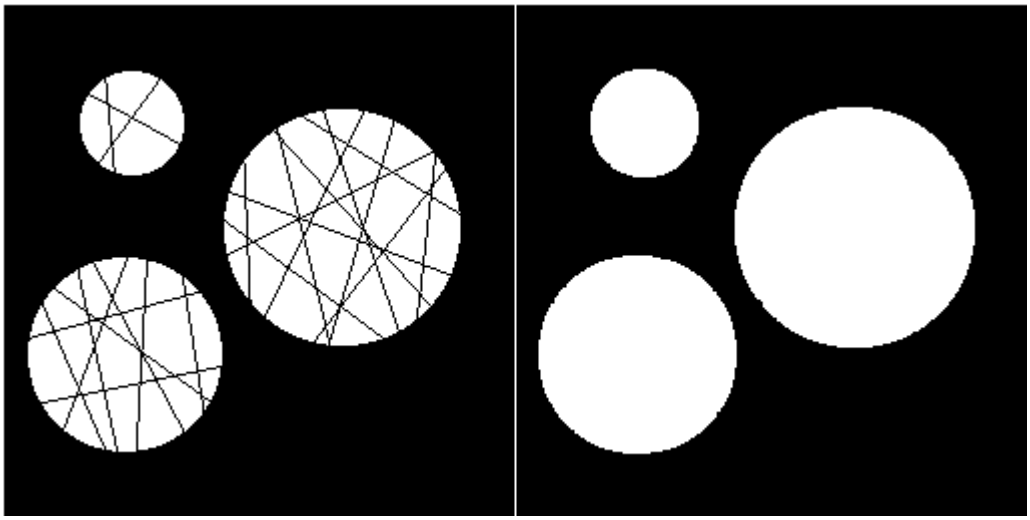
δ(I)(x) = max(I(x + s)) with s ∈ S
where I is the image and S is the structuring element.

Dilation **only works with binary images**. In this implementation, a 3x3 (square ?) structuring element is used expanding the TRUE (white) pixels.

A) Original image. B) Dilated image after 3 cycles of dilation.

1 Use

Dilation is used to **remove pepper noise** (black dots in your image) or **FALSE defects**.



A) Original image. B) Dilated image.

2 Dilation of gray-level images : Maximum filter

The equivalent for gray-level images is **the maximum filter**. Here, dilation is used to **remove darkest features** in your image.The Lena's eyes appear **brightest** than those of the original gray-level image.

A) Original image Lena converted to 8-bit. B) Dilation using a square 3x3 structuring element (radius = 1 in Maximum filter).

Influence of the structuring element

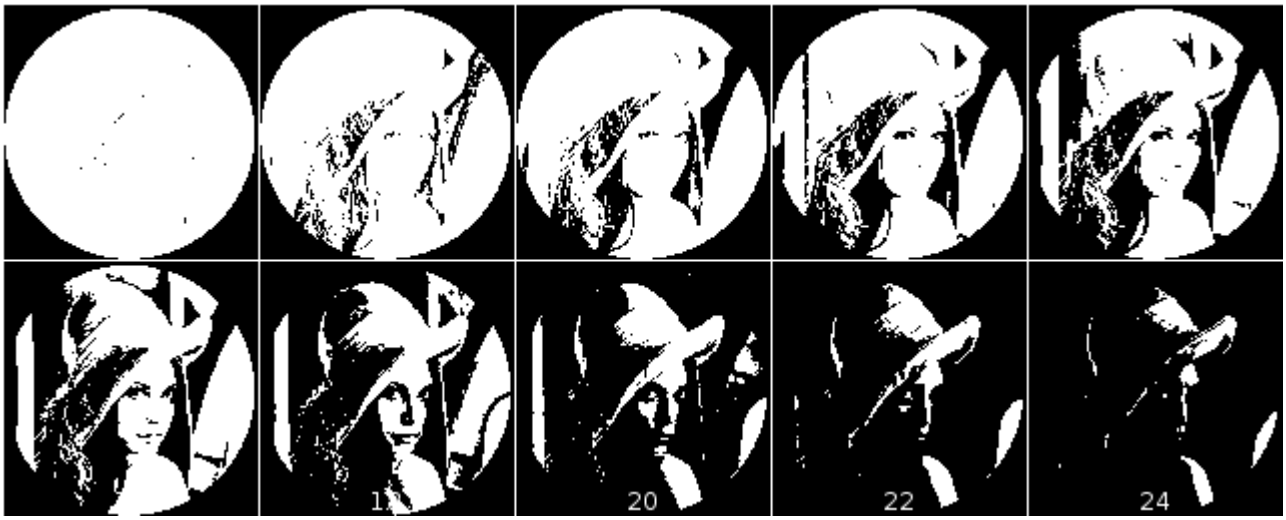
One of the main problem of erosion and dilation is the fact that the shape of structuring element **distorts the objects** of interest and can't be used for an analysis (specially if you are interested in areas).

Gray-level images and morphology

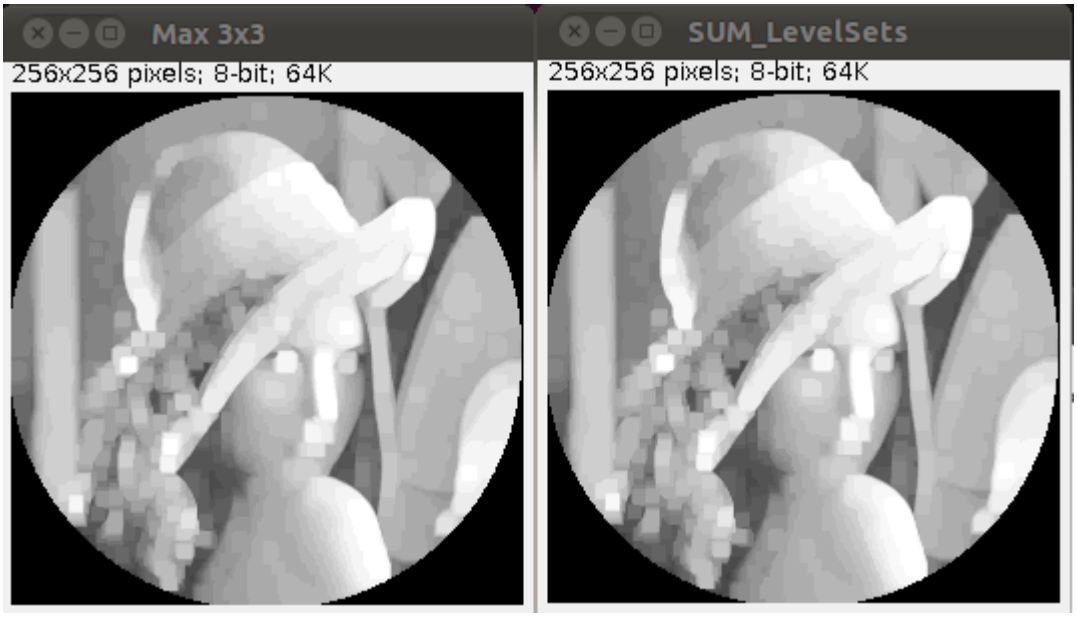
In MM, the gray-level images are considered as a **stack** of binary images, what they called **level sets**.

Mathematical Morphology (MM) only uses **binary images**. However, it is possible to use the same functions with gray-level images by decomposing them into a series of binary images (called level sets).

A 256x256 8-bit image of Lena with 32 gray levels



Decomposition of Lena into level sets.



Result of: A) three Maximum filters with a radius = 1 applied to the image of Fig. 1 and B) three dilations of the level sets.

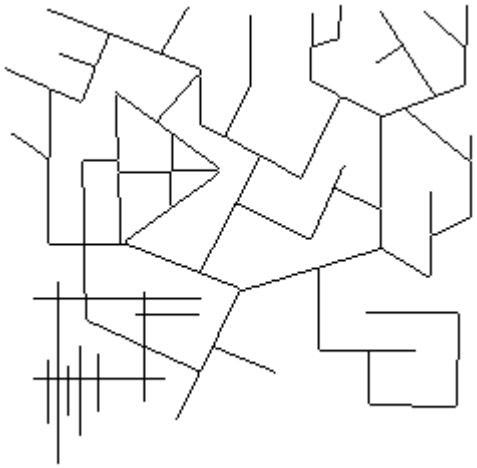
It is possible to work with gray-level images in mathematical morphology, you have to first transform these images into level sets.

Operators using Erosion/Dilation

Hit-or-miss

In Mathematical morphology, the **Hit-or-Miss operator** is useful to describe the **topology** of a graph-like object. Even though this is not implemented − by default − in ImageJ, we can obtain a similar result...

The picture below is a classical example of what can do a Hit-or-Miss operator for describing the topology of the graph (leaves, nodes, corners, etc.).



The **Hit-or-Miss** (also called Hit-and-Miss) operator is an **erosion-like function** that only works for a specific topology of TRUE and FALSE pixels in a 3x3 kernel (structuring element in the terminology of Mathematical Morphology).
If the **combination of TRUE and FALSE pixels** in the structuring element exactly match the pixels in the image, then the **pixel underneath the origin** of the structuring element is set to TRUE otherwise it is set to FALSE.

For example, if you want to detect the **lower left** corner of an object of interest, you can design the 3x3 kernel. The **central pixel** corresponds to the **lower left corner**. However, this first version is limited to filled rectangles and can be improved by removing the unnecessary pixels. The **'X' pixels are called 'Don't Care' pixels** because whatever their value (0 or 1), it does not change the topology of the feature detection.

```
0 1 1    0 1 X    X 1 X
0 1 1    0 1 1    0 1 0
0 0 0    0 0 0    0 0 X
```

Fig.2: Designing a kernel to detect a 90° corner. A) First attempt. Only works with filled rectangles. B) We don't care of the inner pixel. C) If we are working in a 4-pixels connectivity, the diagonal pixels are irrelevant. They are set to 'Don't Care' pixels ('X').

As there is four different patterns of corners, you need four kernels.

```
X 1 X    0 0 X    X 0 0    X 1 X
0 1 1    0 1 1    1 1 0    1 1 0
0 0 X    X 1 X    X 1 X    X 0 0
```

Fig.3: Four kernels required to detect corners. The '0' and '1' correspond to the FALSE and TRUE pixels, respectively. The 'X's are the 'Don't care' pixels (0 or 1).

Applying these four kernels to the image :

Detecting corners with a Hit-or-Miss operator. For sake of convenience, the corners were dilated.

Note: If you want to restrict corners detection to 90° corner in a 8-pixels connectivity, you have to use the following pattern:

```
0 1 X
0 1 1
0 0 0
```

> The next part is all due to the great informations of this site : https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm, ©2003 R. Fisher, S. Perkins, A. Walker and E. Wolfart.

Opening and Closing

Opening and closing are two important operators from mathematical morphology. They are both **derived** from the fundamental operations of **erosion and dilation**. Like those operators they are normally applied to binary images, although there are also graylevel versions.

Opening

The basic effect of an **opening** is somewhat like erosion in that it tends to **remove some of the foreground** (bright) pixels from the **edges of regions** of foreground pixels. However it is **less destructive than erosion** in general. As with other morphological operators, the exact operation is determined by a structuring element. The effect of the operator is to preserve foreground regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of foreground pixels.

An opening is defined as an **erosion followed by a dilation** using the **same structuring element** for both operations. The opening operator therefore requires **two inputs**: an image to be opened, and a structuring element.

Graylevel opening consists simply of a graylevel erosion followed by a graylevel dilation.

Closing

Closing is similar in some ways to **dilation** in that it tends to **enlarge the boundaries of foreground** (bright) regions in an image (and shrink background color holes in such regions), but it is less destructive of the original boundary shape. As with other morphological operators, the exact operation is determined by a **structuring element**. The effect of the operator is to **preserve background regions** that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of background pixels.

Closing is **opening performed in reverse**. It is defined simply as a dilation followed by an erosion using the same structuring element for both operations. The closing operator therefore requires **two inputs**: an image to be closed and a structuring element.

Graylevel closing consists straightforwardly of a graylevel dilation followed by a graylevel erosion.

Opening is the **dual** of closing, i.e. opening the foreground pixels with a particular structuring element is equivalent to closing the background pixels with the same element.

Skeletonization/Medial Axis Transform

Skeletonization is a process for reducing foreground regions in a binary image to a **skeletal remnant** that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. To see how this works, imagine that the foreground regions in the input binary image are made of some uniform slow-burning material. Light fires simultaneously at all points along the boundary of this region and watch the fire move into the interior. At points where the fire traveling from two different boundaries meets itself, the fire will extinguish itself and the points at which this happens form the so called **`quench line'**. This line is the **skeleton**. Under this definition it is clear that thinning produces a sort of skeleton.

The terms **medial axis transform (MAT)** and **skeletonization** are often used interchangeably but we will distinguish between them slightly. The skeleton is simply a **binary image** showing the simple skeleton. The MAT on the other hand is a **graylevel image** where each point on the skeleton has an intensity which represents its distance to a boundary in the original object.

The skeleton/MAT can be produced in **two main ways**. The first is to use some kind of morphological **thinning that successively erodes** away pixels from the boundary (while preserving the end points of line segments) until no more thinning is possible, at which point what is left approximates the skeleton.
The alternative method is to first **calculate the distance transform** of the image. The skeleton then lies along the **singularities** (i.e. creases or curvature discontinuities) in the distance transform. This latter approach is more suited to calculating the MAT since the MAT is the same as the distance transform but with all points off the skeleton suppressed to zero.

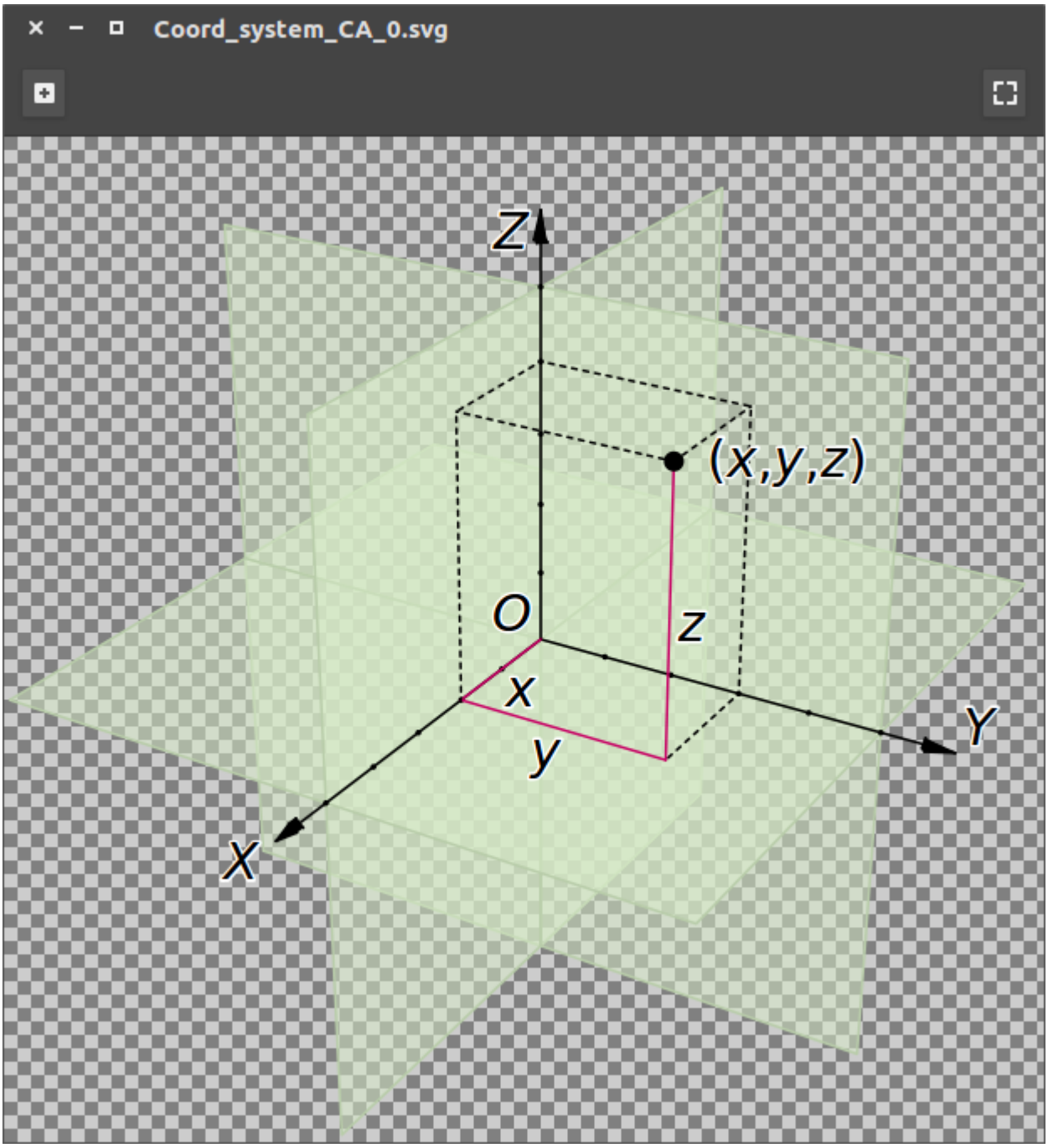
> End of the informations. Reminder : https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm, ©2003 R. Fisher, S. Perkins, A. Walker and E. Wolfart.

Distance transform

A **distance transform**, also known as distance map or distance field, is a **derived representation of a digital image**. The choice of the term depends on the point of view on the object in question: whether the initial image is transformed into another representation, or it is simply endowed with an additional map or field.

Euclidean distance

In mathematics, the **Euclidean distance** or Euclidean metric is the "ordinary" straight-line distance between two points in **Euclidean space**.



Euclidean space,

> By Jorge StolfiiThe source code of this SVG is invalid due to 11 errors, caused from superfluous hyphens. - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=6692547

With this distance, **Euclidean space becomes a metric space**.

Watershed

In the study of image processing, a **watershed** is a transformation defined on a **grayscale image**. The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a **topographic map**, with the **brightness** of each point **representing its height**, and finds the lines that run along the tops of ridges.

There are different technical definitions of a watershed. In graphs, watershed lines may be defined on the **nodes, on the edges, or hybrid lines** on both nodes and edges. Watersheds may also be defined in the

continuous domain. There are also many different algorithms to compute watersheds. Watershed algorithm is used in image processing primarily for segmentation purposes.

Measurment

Length

The length of a line in **ImageJ** can be quite hard to get, in fact, because o the numeristation of anaogical world, a diagonal or a curve are described discretly by two differents ways:

- **NSEW** : all connected so you have only four directions North South East West
- **8-dir** : not connected because there is 8 directions

$$
D _{Real} = D _{8-dir} / 0.9 = D _{NSEW} / 1.273 \
D _{8-dir} = \text{Nombres de pixels d'une ligne en 8-dir} \
D _{NSEW} = \text{Nombres de pixels d'une ligne en NSEW}
$$

 Exemple anal to num
Caracteristics of complex wormly object:

- Length
    1. Skeletonize
    2. Get number of pixel (Hist ou AREA)
- Width
    3. Euclidian Distance Map
    4. Width = $Hist_{max\ pixel}$ x 2

Number and diameter of many circles:

- Ultimate Points
- Histogram of result
    - Number of points = Number of Centers of circle
    - Diameter = Value of center pixels x 2

Sources

- Wikipedia contributors, "Digital image processing," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Digital_image_processing&oldid=803425387 (accessed October 5, 2017).
- Wikipedia contributors, "Noise reduction," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Noise_reduction&oldid=798926923 (accessed October 5, 2017).
- Wikipedia contributors, "Euclidean distance," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=803746524 (accessed October 5, 2017).

- https://crazybiocomputing.blogspot.fr/p/image.html , posted by Jean-Christophe T.
- https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm, ©2003 R. Fisher, S. Perkins, A. Walker and E. Wolfart.