Traitement de sons et d'images

Introduction et Cas d'utilisation

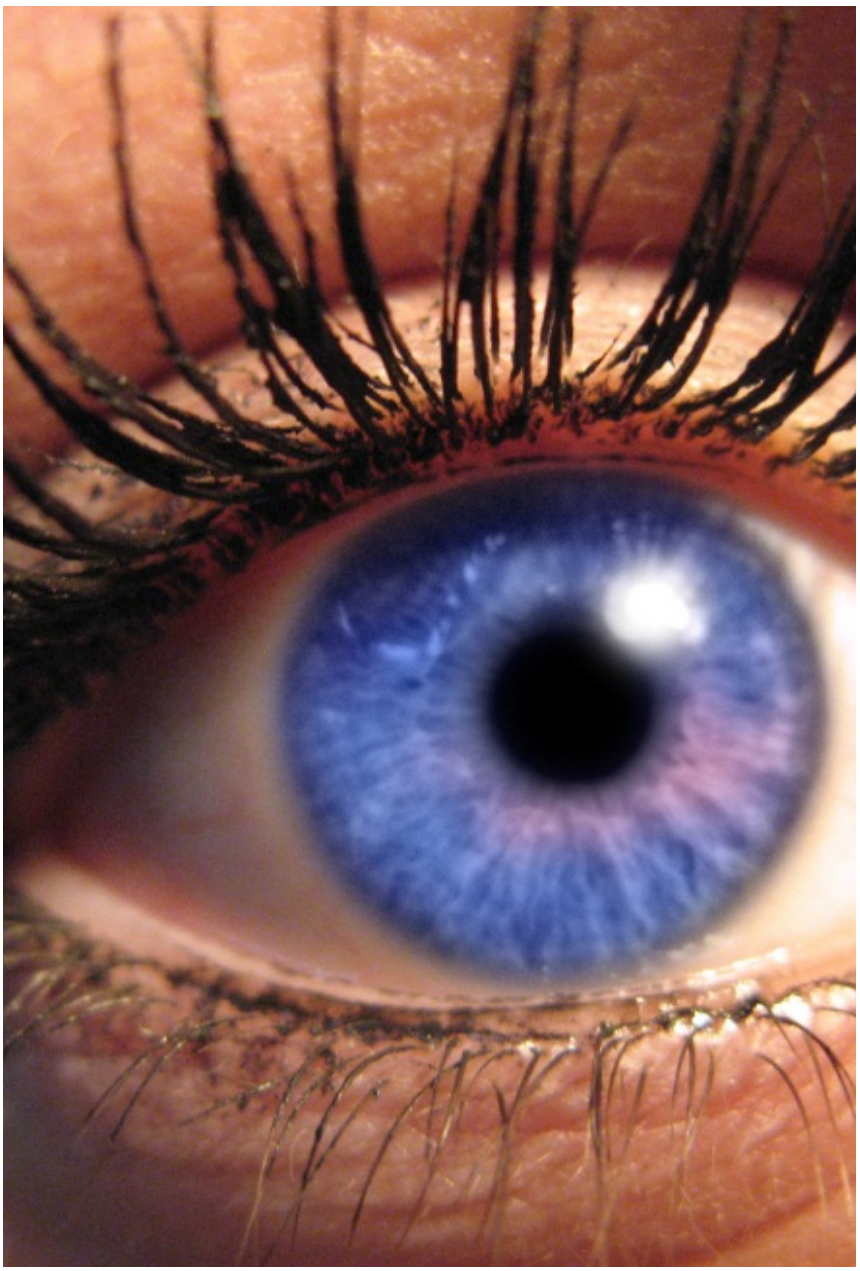
Rudi Giot

1re édition 2016 - Dernière révision le lundi 30 août 2021

Chapitre 1

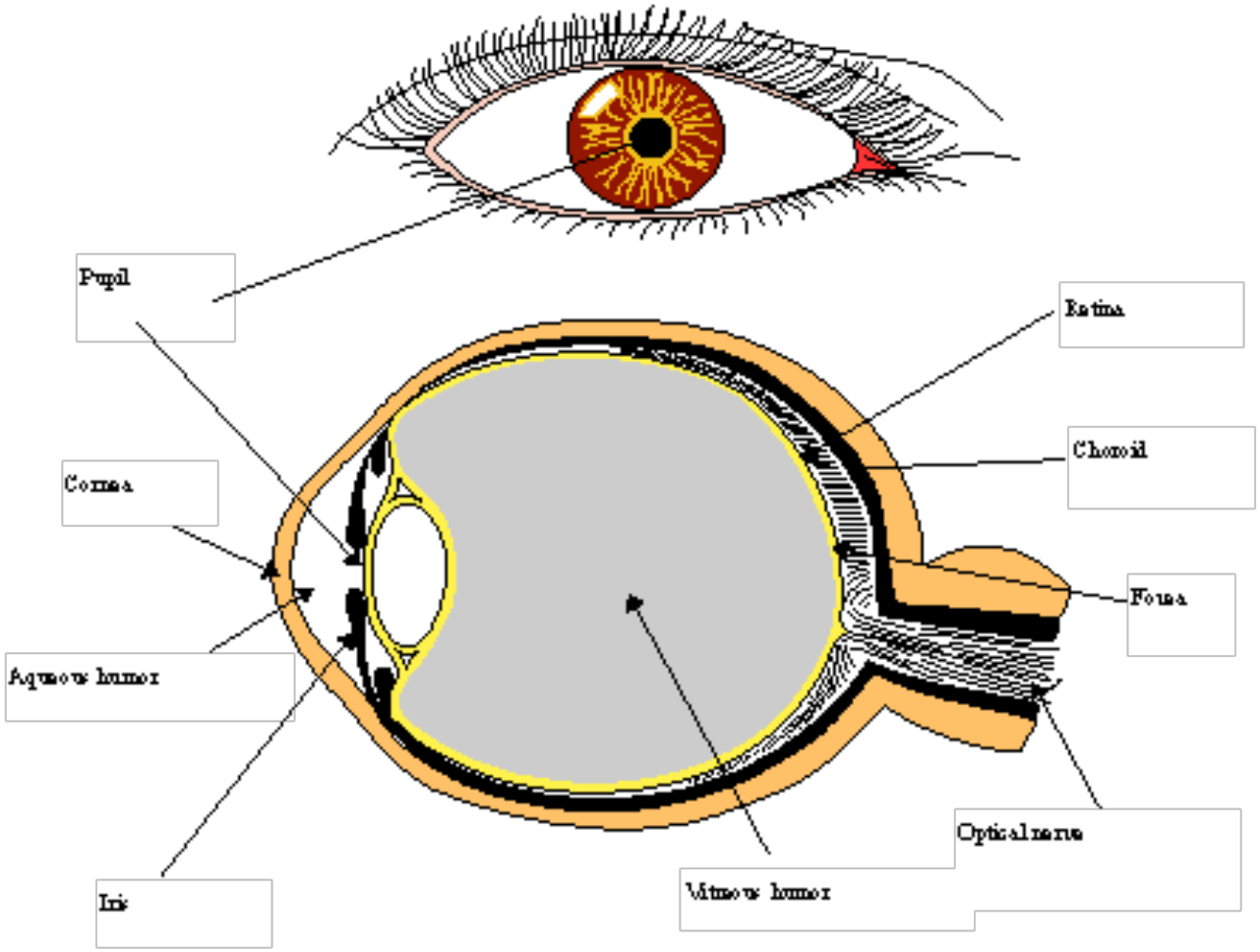
Introduction

The difference between the user's vision a of the color and the techniques allowing its visualisation on the computer screen, is explained by the fact that each element of a computer chain is characterised by its own sensibility to that color. Therefore it is required to begin studying the characteristics of the light and those of the interaction between the eye and the brain.



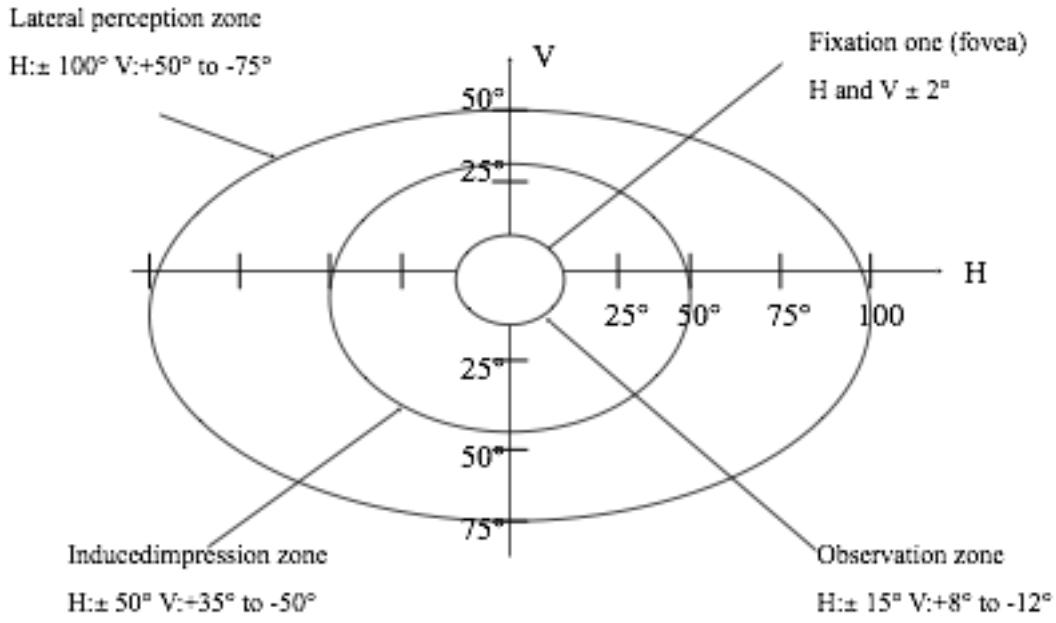
The Eye

The study of the colours requires an approach of the eye functioning. We will distinguish two parts in this section. The first deal with the vision field which is useful to determinate image resolution, the second explain how colour is perceived and interpreted by the eye.



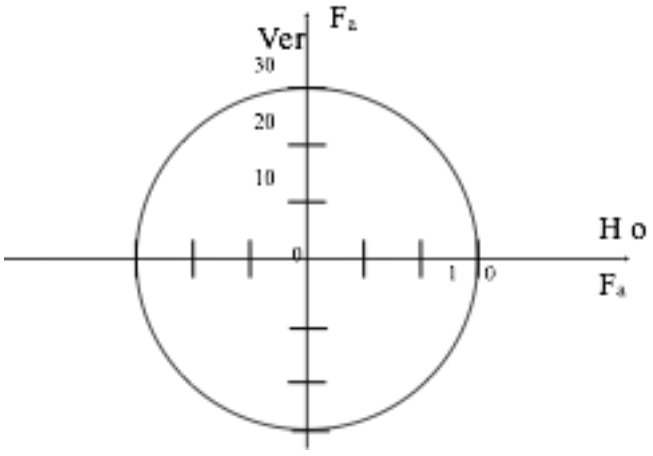
Vision field

The sensorial structure of the retina is modifying when moving away from the fovea zone. The density of the sensory nerves decreases and the keenness as well. There are four different concentric zones in the field of vision: the fixation zone, the observation zone, the induced impression zone and the lateral perception zone.



Sensible zone of the eye

If we define the angular frequency Fa as the number of lines per degree (unit: lines/ degree) that the eye can distinguish, we can draw the figure of the visual keenness. In reality this diagram is not a circle. The vision is not isotropic.



Limit of the visual keenness

A human eye can distinguish moreover an angle of 1' (60 lines per degree).

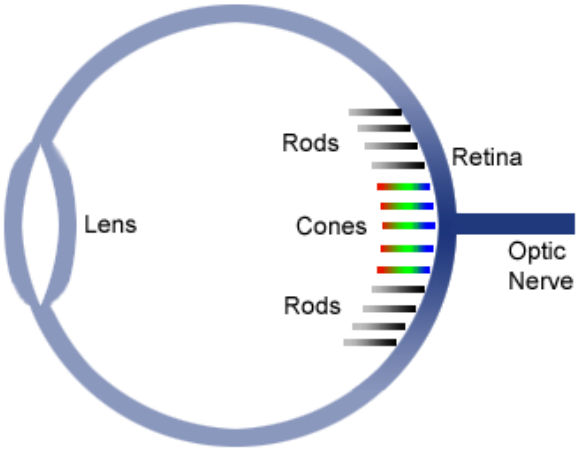
Colour detection

The functioning of the eye is similar to the dark room of a camera. The sensitive area of the eye (the retina) is made up of sensors which are the sensitive endings of a large amount of sensory nerves collected on a single nerve (the optical nerve).

The terminal links (of the order of one micrometer) are called cones (if they have an individual nervous link) and rods (if they are brought together in grapes on a nervous link). The cones give an accurate sensory information (the colour).

The rods give information on the very weak levels of luminous energy (luminosity). If the luminous intensity decreases, the cones will not work as well as and the perception of the colours will not be that clear.

The biologists have displayed three types of cones, each of them being sensitive to only one luminous field : blue, green, red. In other words, each type of cone is only sensitive to one interval of the visible spectrum.



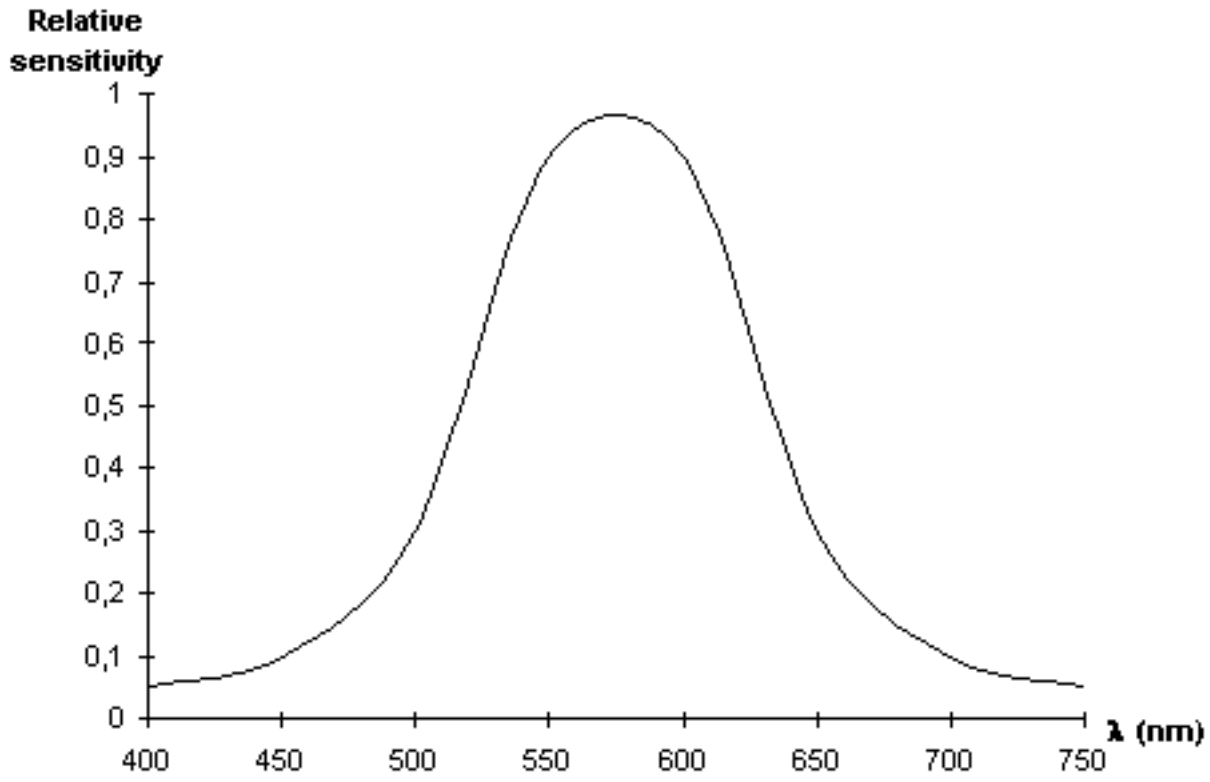
Cones and Rods in the eye

Like all detectors, the cone sensitivity is not constant on its interval and thus even with an equal radiance intensity, some wave lengths will appear to be more or less luminous.

This phenomenon, valid for each type of cone is cumulative. Therefore the eye sensivity varies greatly according to wave length of the light.

If we take for instance a monochromatic light, which only radiates in a single wave length, we see that the eye sensitivity reaches its maximum around 5.500 angstroms, which approximately corresponds to yellow. The eye sensitivity decreases quickly around the end of the visible spectrum (blue and red). On the other side, the shortest wave lengths (around the blue) are those which seem to be the less luminous.

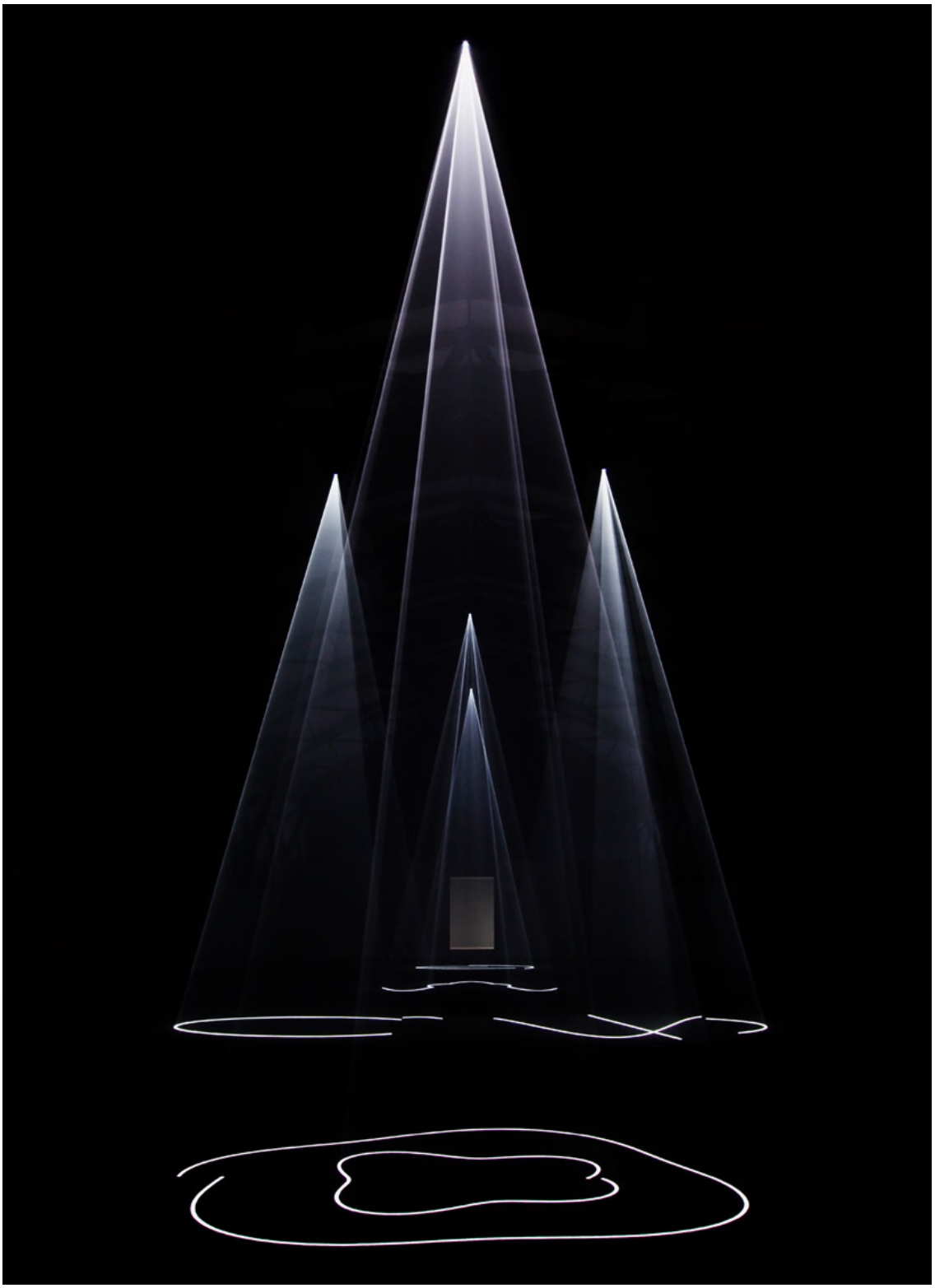
However the eye can not distinguish between a monochromatic radiation and a radiation with a spectral discrete distribution. The brain assumes an integration of the whole of the radiations to give the impression of a determinated colour (or of the white).



Curve of the relative sensitivity of the eye

The light

Light is an electromagnetic radiation made of photons, i.e. elementary particles with no mass. This energy is subjected to all the corresponding physical rules. Light speed in a vacuum is an universal constant, referred to by C and approximately equal to 300.000 km/s.



The propagation speed varies according to the space that is covered :

$$\lambda = \frac{C}{n.f} \qquad (1.1)$$

« n » depends on the spaces, it is equal to « 1 » in the vacuum.

Monochromatic light has a frequency f independent of the space through which it propagates, but its wave length λ depends on the covered space.

$$V = \frac{C}{n} \qquad (1.2)$$

Therefore it seems obvious to characterise monochromatic light by its frequency since it is a constant. Nevertheless usually, wave length is chosen to define it.

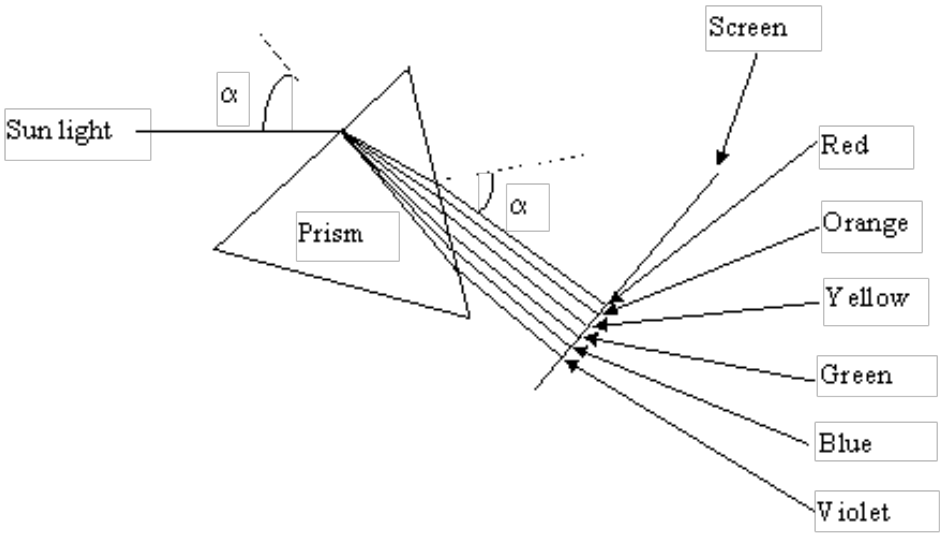


Figure 1 Light dispersion in the Newton's prism

Physically white light does not exist. It is the result of a combination of a collection of radiations. Newton's prism (Figure 1) facilitates this observation of luminous dispersion, supplied by the sun or by a incandescent lamp, passed threw the prism with an incident angle α in relation with the perpendicular. In the glass, the beam changes direction and moves on towards the perpendicular at the separation area on the incident point. It is the refraction phenomenon. The same phenomenon is observed at the prism exit, at the time of the glass-air transition.

The total angle of refraction varies with the wave lengths of the incident radiations (the highest λ corresponds to the weakest deflections). On a white screen located at the exit of the prism, you can see a luminous stripe the colour of which gradually varies from red to violet. To each of these radiations (called « monochromatic ») a wave length is associated (400ηM for violet until 780ηM for red). Therefore the eye sees thus mixtures of monochromatic lights in different proportions. This is what gives a sensation of colour. The electromagnetic spectrum for wavelengths ranges from 10^{-10} µm to 10^{5} km. The spectrum is given in the following table:

EM portion	Wavelength [µm]
long electrical osc.	10^{14} to 10^{11}
radio waves	10^{11} to 10^{6}
microwaves	10^{6} to 10^{3}
infrared	10^{3} to 1
visible	1 to 10^{-1}
ultraviolet	10^{-1} to 10^{-2}
X rays	10^{-2} to 10-6
gamma rays	10^{-4} to 10^{-8}
cosmic rays	10^{-8} to 10^{-10}

EM portion (visible part)	Wavelength [ηm]
middle infrared	6000 to 1500
near infrared	1500 to 770
red	770 to 622
orange	622 to 597
yellow	597 to 577
green	577 to 492
blue	492 to 455
violet	455 to 390
near ultraviolet	390 to 300

Table 1 The electromagnetic spectrum

The light can be represented by the curve of the spectral distribution of its luminous energy (distribution of the energetic flux according to the wave length).

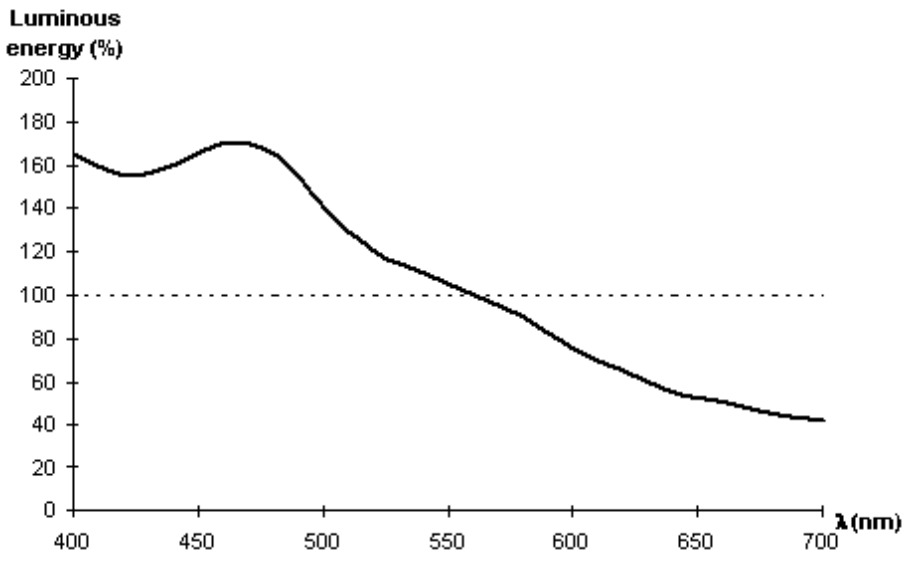


Figure 2 Curve of the spectral distribution of the luminous energy in a blue sky

The colour

Colour sensation of an object comes from the fact that part of the incident light is reflected. So it is possible to say that an object does not have an own colour.

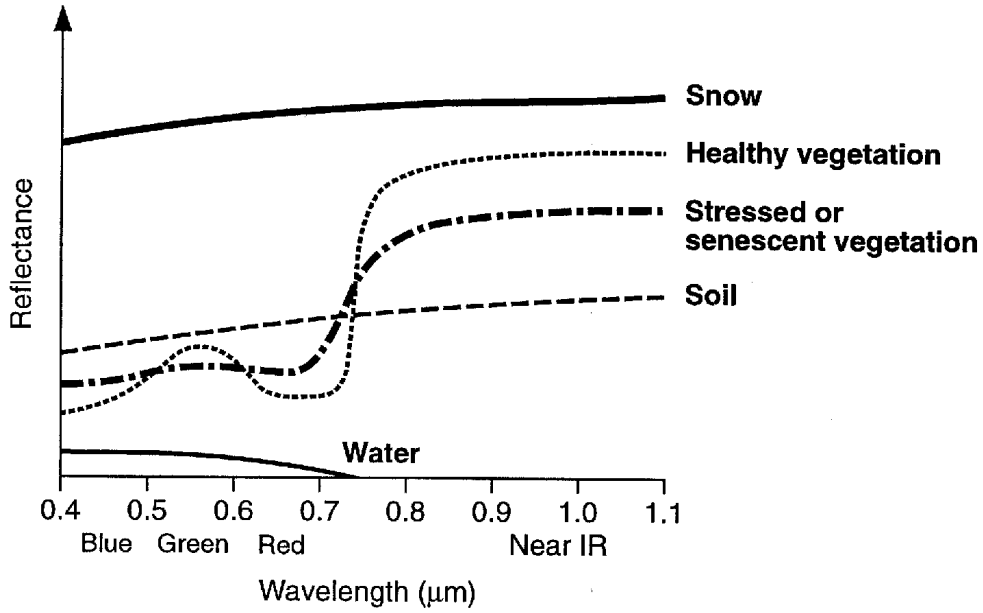
The detected colour effectively depends on:

- the light source
- the way in which light travels
- the sensitivity differential of the eye

Nevertheless, the colour of an object could be completely defined by its reflecting curve. This curve represents in % the reflected luminous energy in function of the length of the incident wave.

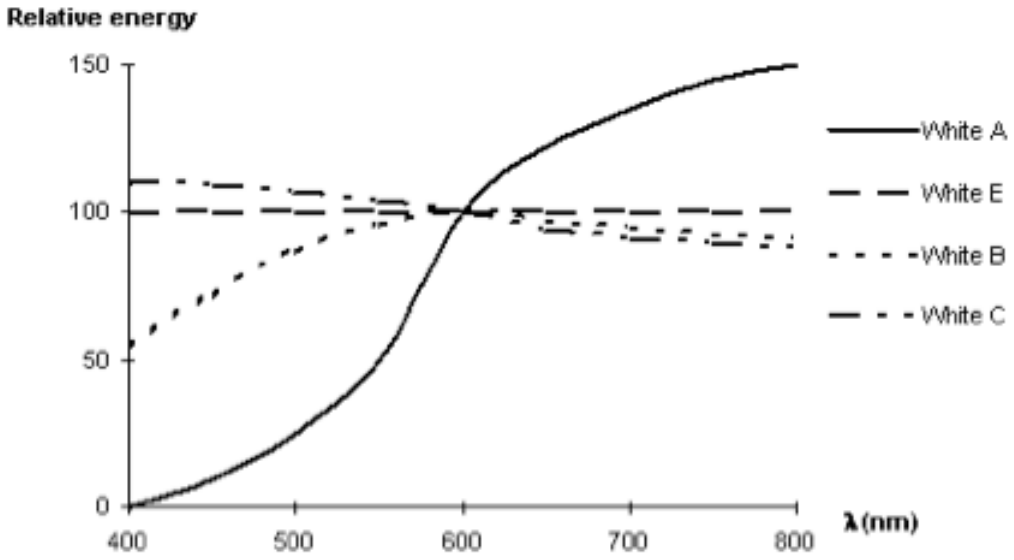
The different whites

It is necessary in any attempt of modelisation to define the references.



Legende

The white given by a constant power of each of the radiations of the visible spectrum (WHITE E) is no doubt the one which seemed to be the more logical to define this reference point. Yet it is not considered as such because it is difficult to produce.



Legende

Three standard sources have been defined by the International Commission of Lighting (ICL). These may be obtained by a black body heated until glowing at a certain temperature.

- the white A (lighting of lamp with a tungsten filament) corresponds to a temperature of the black body : 2848 °K

- the white B (daylight at midday in summer) corresponds to the temperature of the black body : 4800 °K

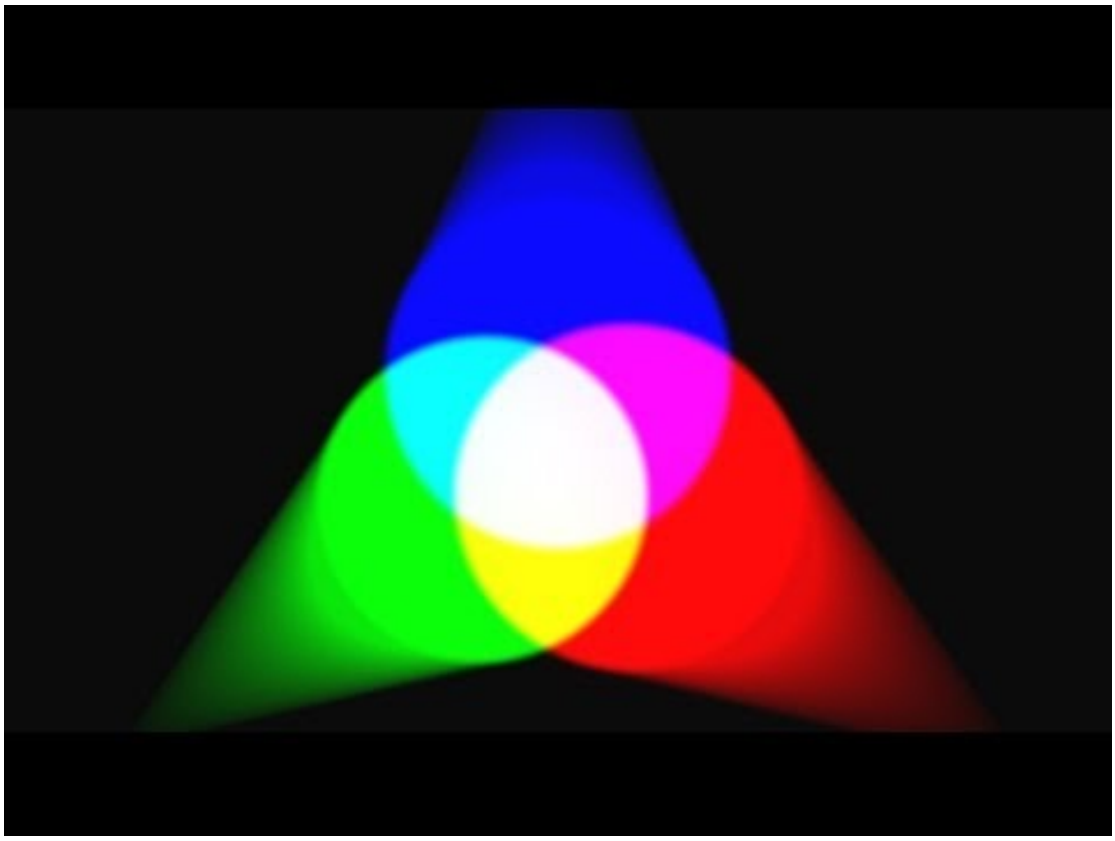
- the white C (sun light filtered by clouds) corresponds to the temperature of the black body : 6500°K. This one has been chosen as reference.

Additive synthesis

A simple and practical experience consists in the lighting of any support with a given light. Then an observer has to reproduce the same value, under the form of coloured adjacent stain, with the help of one or more monochromatic sources (only one wave length) the intensity of which he may vary. Many experiences of this type have shown several interesting properties of the human visual system.

If the observer has only one monochromatic source, he will only be able to pair his light test with the reference if it has the same wave length. However, he will always be able to obtain an equivalent luminous intensity.

On the other hand, the sole use of three different sources will allow him to pair practically all of the tested lights, on the condition that the reference lights are sufficiently far away from each other in the visible spectrum. So this corresponds respectively to former sources of weak (blue), medium (green) and high (red) wave length.



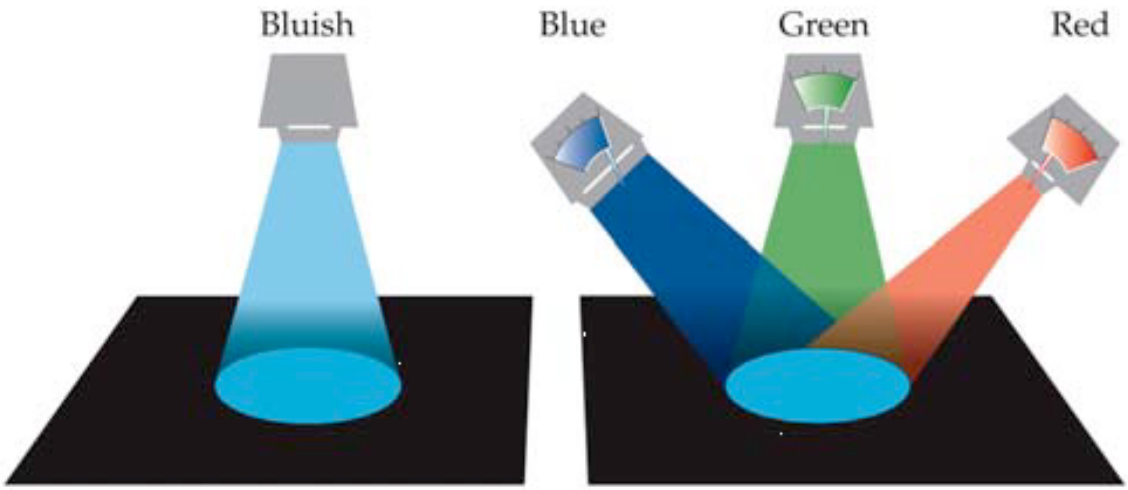
Additive synthesis of colours

If the mixture is made between the extreme radiations of the spectrum, we obtain a subjective range (which does not exist in the visible spectrum) which is called « the range of the purples ».

The combination of these parameters gives more than one million possibilities. This differentiation is less visible in certain conditions (weak light). We will generally admit that the eye detects about 400.000 different colours.

Basic principles and rules

The principle of the three colours process have been formulated by YOUNG, MAXWELL and HELMHOLTZ: « Any coloured sensation may be obtained from a mixture of three colours judiciously selected. This mixture may be either additive (like in the experience) either subtractive (use of filters).



The rules of GRASSMAN have the same origin. If S1, S2, S3 are the three standards sources and « m » the colour to reproduce then:

- any colour may result from the sum of positive or negative quantities of the three primary colours. Those quantities are unique.

$$m = a \cdot S_1 + b \cdot S_2 + c \cdot S_3$$

- if a, b and c vary (the energy of each source) in the same proportions, the equation of the first rule remains valid.

$$k \cdot m = k \cdot a \cdot S_1 + k \cdot b \cdot S_2 + k \cdot c \cdot S_3$$

For the additive mixtures

if $$n = a_1 \cdot S_1 + b_1 \cdot S_2 + c_1 \cdot S_3$$

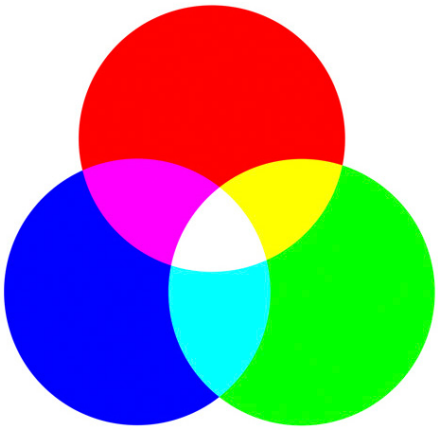
and $$m = a_2 \cdot S_1 + b_2 \cdot S_2 + c_2 \cdot S_3$$

then $$m + n = (a_1 + a_2) \cdot S_1 + (b_1 + b_2) \cdot S_2 + (c_1 + c_2) \cdot S_3$$

The different models

1. RGB

At this stage we could think that a given colour results from the superposition of their components green, red, and blue.



This conclusion, a little hasty, would not be fully complete, since on the one hand, the different cones are not sensitive to a sole wave length (but to an interval) and on the other hand the eye only constitutes a physical receptor. It is, indeed, the brain which produces the colour from the measures which it receives from its « sensors ». So the interpretation method that it develops contributes considerably to the characteristics of the human vision.

The RGB model is probably the most used but not the only one.

2. HSL

If you listen to a person describing a colour, it would give something like: « I am looking for a very clear pastel blue ». We find in this sentence the basic notions of hue (blue), of saturation (pastel) and of luminosity (very clear). The HSL model have been classified by MUNSELL in 1905.

A pure colour will be saturated with 100%. If the saturation reaches 100%, a colour dominant no longer exists and the colour is called achromatic (lack of colour). That reveals a shade of grey, between white and black, according to the luminosity. So the brain acts like an integrator which detects three physical quantities which are:

- The **hue** : characterised by the dominant wave length of the radiation, ± 250 monochromatic hues detectable by the eye + the purples.
- The **luminosity** or **lightness** : luminous energy received, ± 1000 levels of different luminosity's (according to the hue).
- The **saturation** (or colour purity) : contributes to the luminosity of the whole and allows us to differentiate a bright colour and a pastel colour (toned down), ± 20.000 differentiations.

3. ICL

In 1931, the International Commission of Lighting fixed a universal model for the colour representation in order to allow the comparison between the indications provided by several colorimeters.

The chosen system as reference (RGB) is given by the three primaries :

- RED : 700 hm
- GREEN : 546,1 hm
- BLUE : 435,8 hm

If we compute the primary quantities red, green and blue that we have to mix in order to obtain the equalising of a white light E, we obtain these trichromatic coefficients (R, V, B).

$$r = \frac{R}{R + V + B}$$

$$v = \frac{V}{R + V + B}$$

$$b = \frac{B}{R + V + B}$$

By definition: $r + v + b = 1$

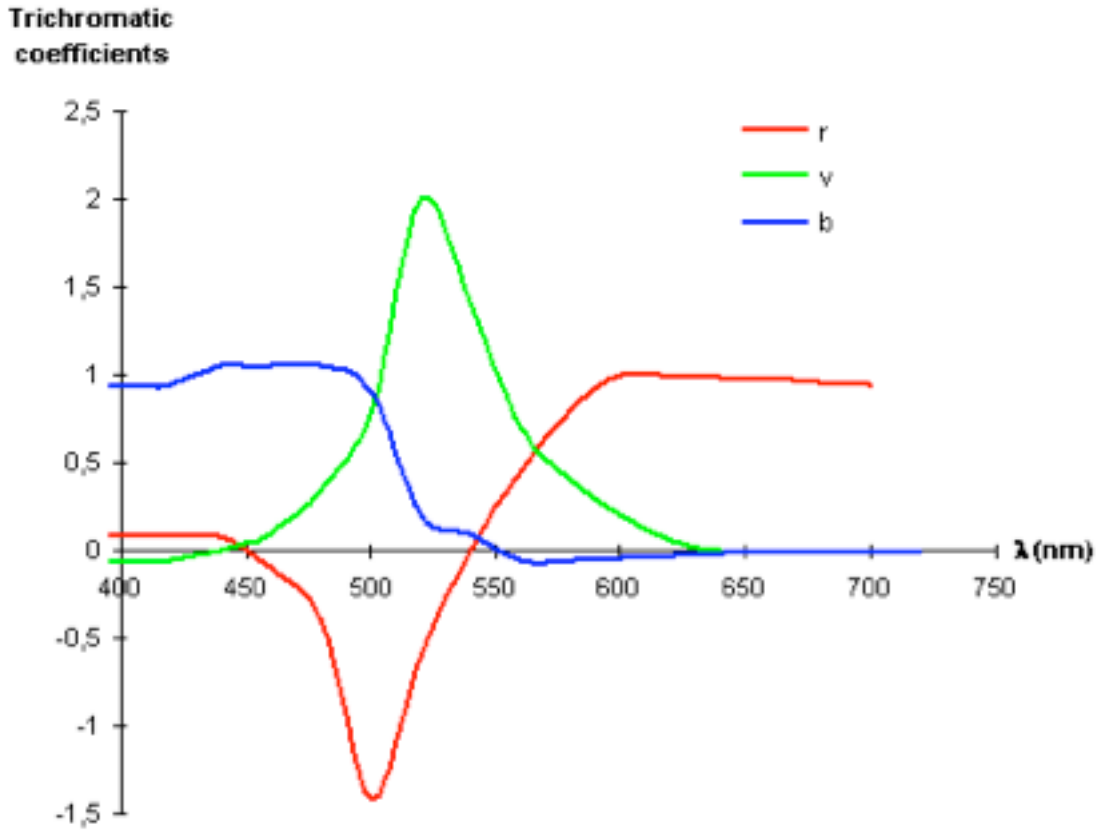
Afterwards we may calculate these coefficients for each of the monochromatic radiations of the spectrum (with the same primaries and the same lighting) and trace the equalising curves of the equi-energy spectrum.

However we prefer a representation by projection on a given plane of a three axis system with the co-ordinates R, G and B. If we associate at each one of the wave lengths (from 400 ηm to 700 ηm) a vector OC having the trichromatic coefficients which make up the component, we visualise then a left curve in a standard three dimensional reference. This curve is the locus of the extremities of these vectors. From this curve we will delimit part of the colourimetric space.

The limits of this sub-space are :

- The conic area formed by the ray OC (C belongs to the left curve).
- The surface plane Od (d straight line of the purples).

All the points of this space represent the whole of the real colours. The external points will be associated with unreal colours (not physical).



Curve of the equi-energy spectrum with the primary colours

By cutting this space with the plane r + v + b = 1 projected on rg we obtain the diagram of chromaticity. The trace of the cone delimiting the real colours is a curve which is concave, closed, on which the real colours are located. The curvilinear part of this curve represents the monochromatic colours and is graduated in wave length. It is called « spectrum locus ».

If we put this curve in the standard defined by the ICL

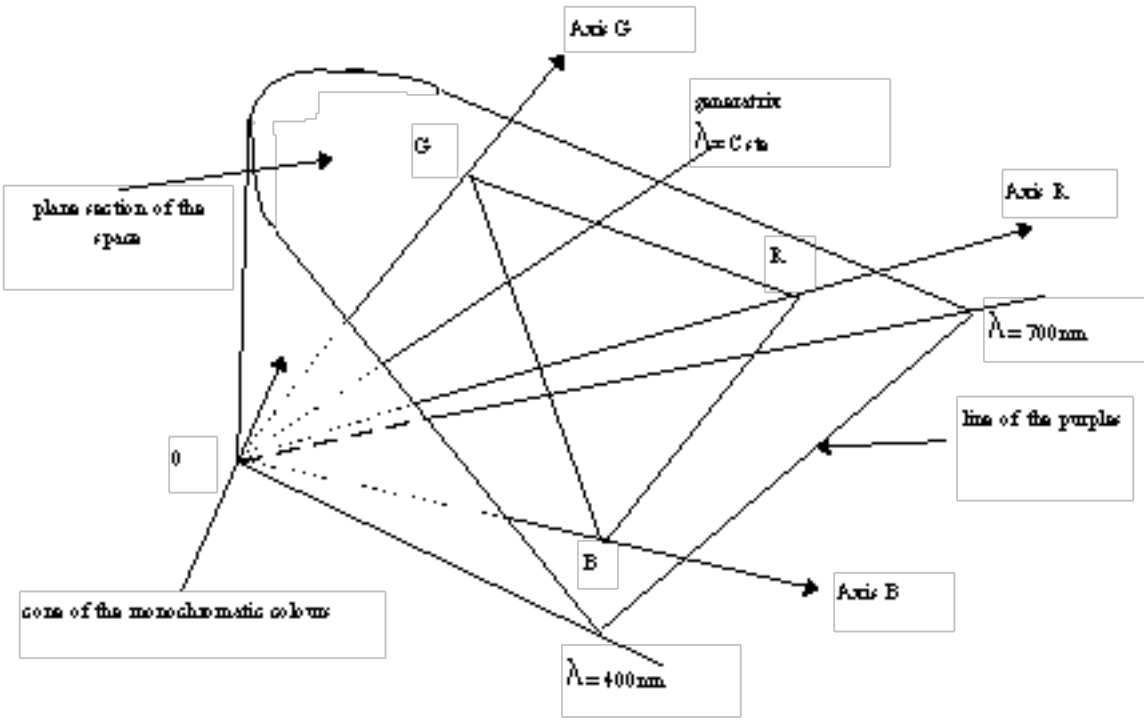
- the three primary axis Ox, Oy, Oz are trirectangles,
- the planes Ox and Oz are in the plane of no lighting

We change the co-ordinates:

$$x = 0.490 \cdot r + 0.310 \cdot g + 0.200 \cdot b$$
$$y = 0.177 \cdot r + 0.812 \cdot g + 0.011 \cdot b$$
$$z = 0.000 \cdot r + 0.010 \cdot g + 0.990 \cdot b$$

Then we proceed as in the preceding case. We cut the space by a plane (x + y + z) = 1 and we make a projection on xOy.

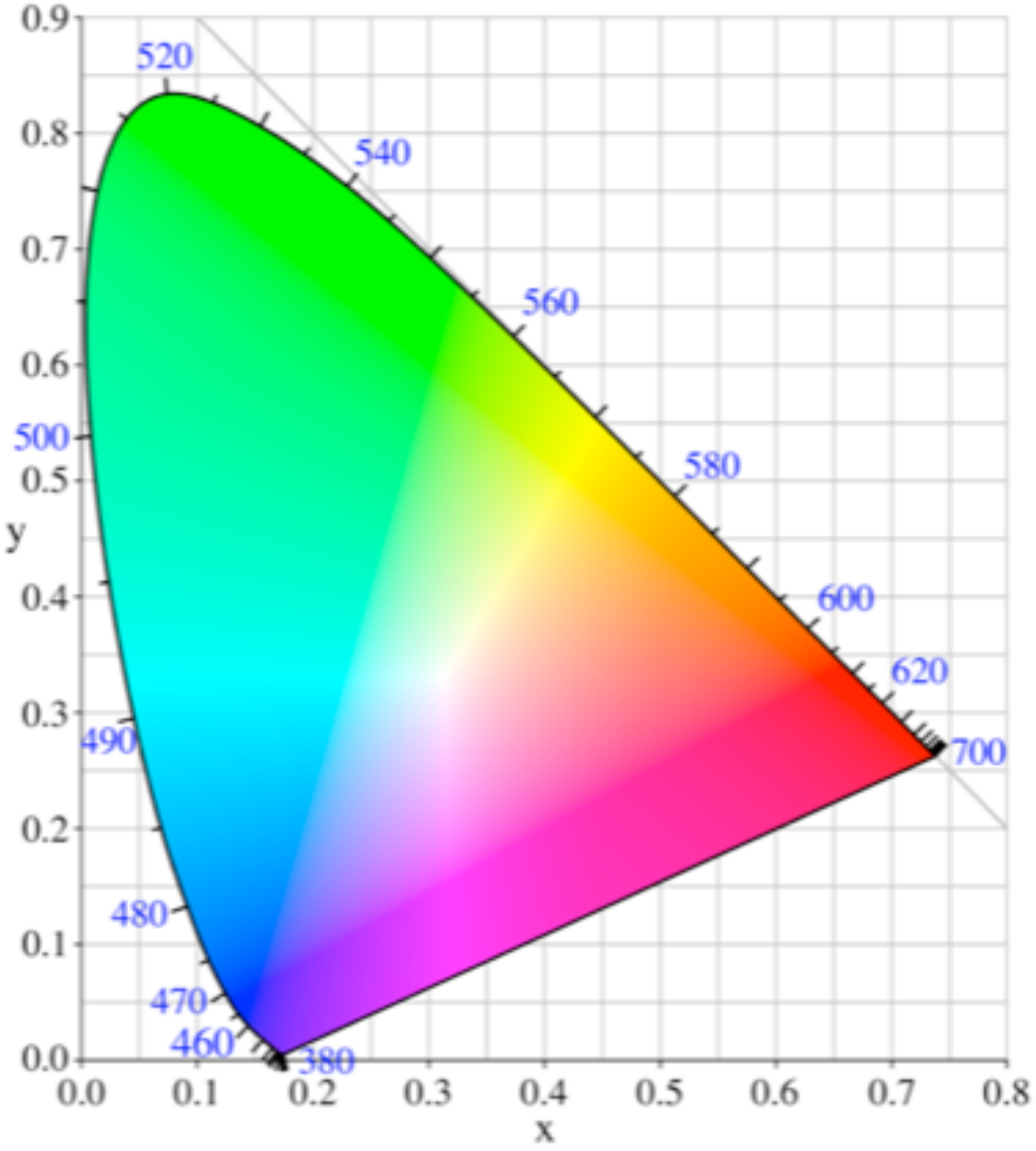
This diagram is also called the triangle of colours.



Sub-space of the real colours

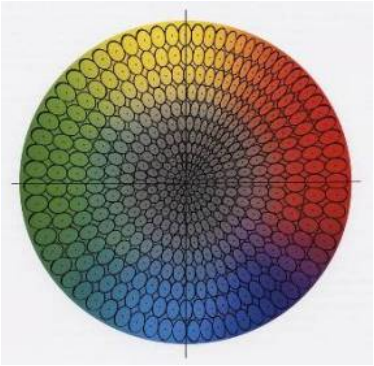
Chromacity diagram x,y of the ICL

In 1960 the ICL held a uniform chromacity diagram (USC - 1960) based on a transformation of reference points (co-ordinates u, v). In 1976 the same diagram was again modified in order to improve the computing of the chromatic deviations (co-ordinates u',v').



ICL colour diagram of chromacity

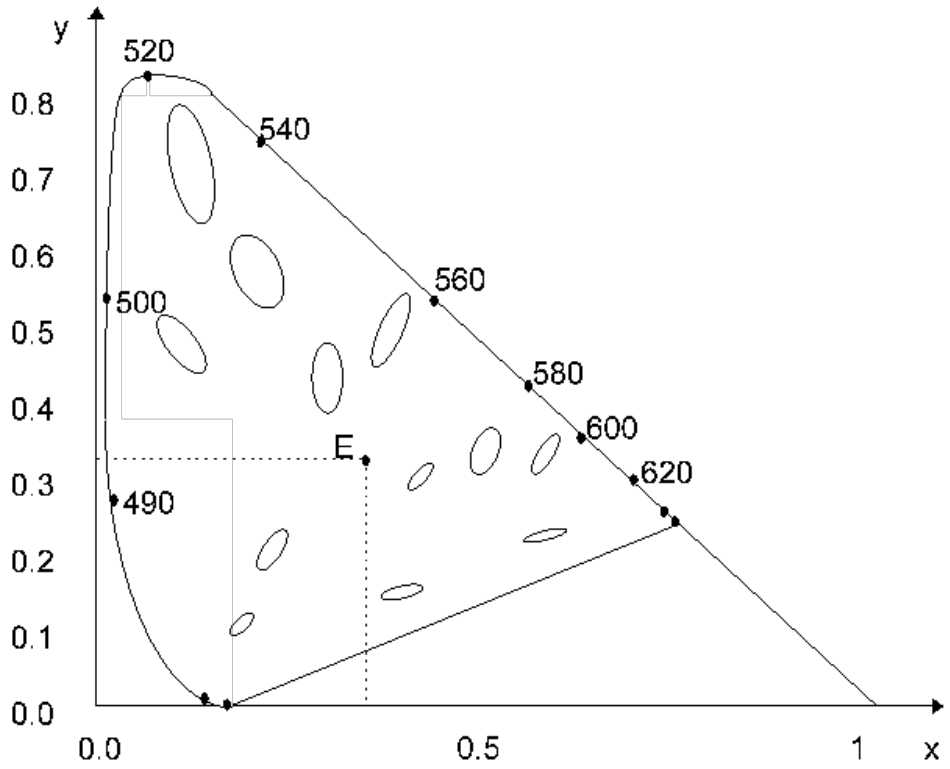
Mac Adam's ellipses



If in the diagram, around a multitude of points C, representing different colours, each one of the closed curves corresponds to the discernible colours, we obtain a distribution of ellipses such as shown in the following figure.

These Mac Adam studies also give the parameters of these confusion cells (dimension of the two axis and orientation of the long axis) for a very important number of points, covering a large part of the chromacity diagram.

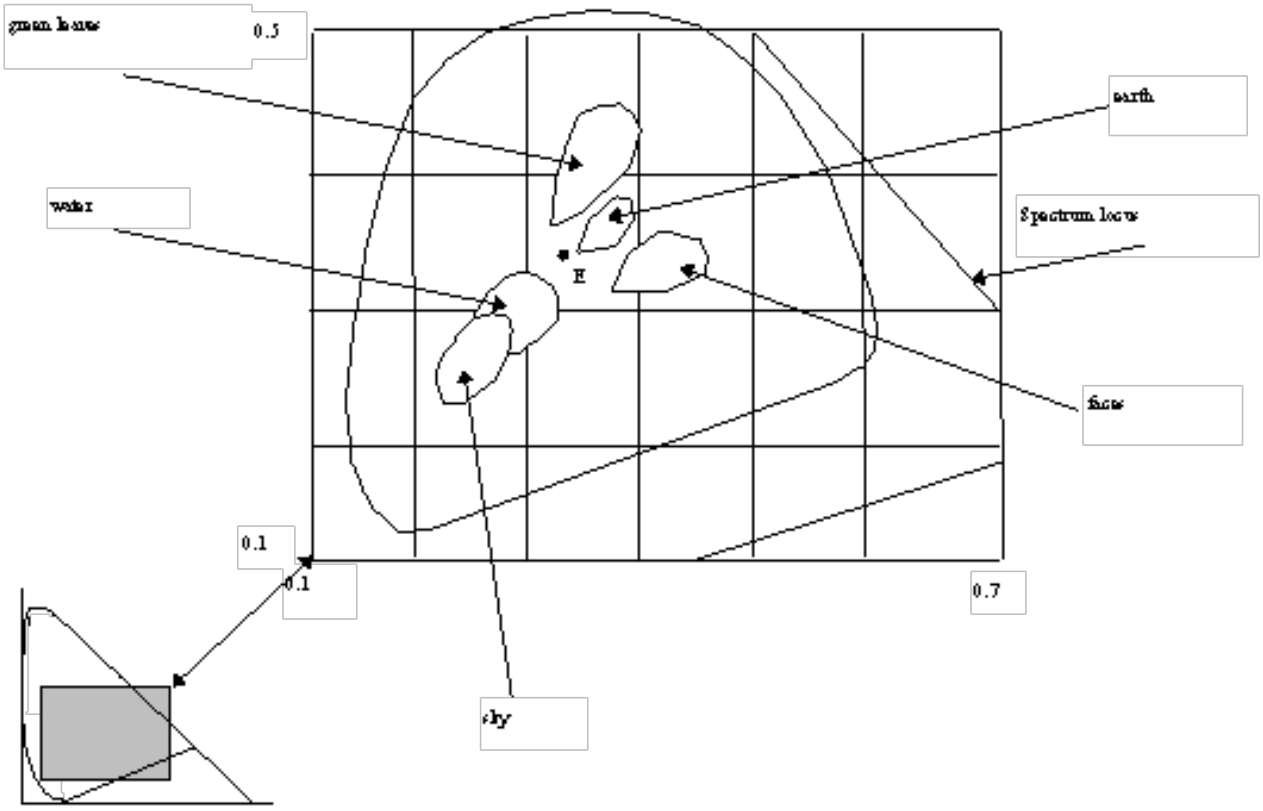
We observe from the diagram that the dimensions of the ellipses are variable (big in the green zones, reduced in the blue zones), that their eccentricity is big and that the orientation of the long axis evolves towards E whites, one direction following the axis yellow-blue.



Mac Adam's ellipses

Natural colours

In practice very few natural colours, illuminated by the sun have a strong saturation. The purity of excitation is generally beyond to 0,5. On the other hand the artificial colours may be more saturated (particularly the orange, yellow and red hues).

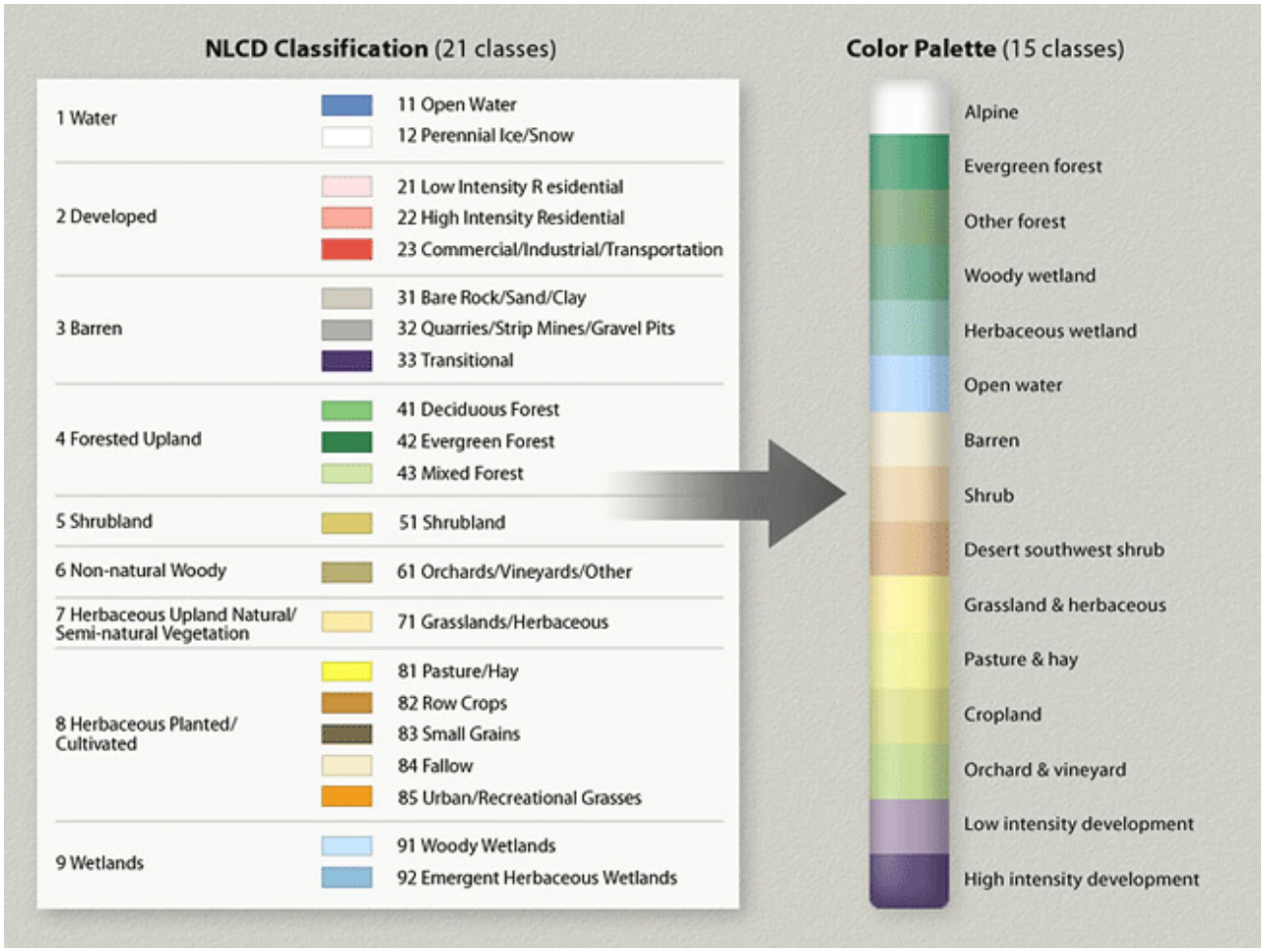


Natural colours in the x,y diagram

The majority of the usual hues stands, in the chromatic diagram x,y, in a zone around the whites. This area represents more or less half of the total diagram.

We will also notice that a range of a flash-coloured hues is defined in order to obtain a better rendering of the skin colour. The European Union of Radio Broadcasting centres this zone on x=0,38 ; y=0,36.

These notions are very interesting for the choice of palettes.



Chapitre 2

Color Coding

That chapter will present the different way to code the colors. The different models RGB, HSL, CMY, ICL will be illustrated and some equations will help you to swap from one color space to another.

RGB model

This coding system is the most universal known in the computer world. It is just like the working of a colour monitor or a graphic card (additive synthesis).

Clown RGB

A colour is defined with three values: one for the **R**ed, one for the **G**reen and one for the **B**lue. They are called the three components. Usually those component values are between 0 and 255 (one byte). So, it is possible to represent 16.581.375 different colours. The RGB system consists in an additive synthesis of the colours. If each one of the components is at its maximum the result is white, if each is at its minimum the result is black (no light). If we have the same proportion of red, blue and green we have a grayscale. The increasing of one component increases the brightness of the result.

Example: If the previous image called « clown » is splited in its three components R, G and B we will get that result :

Unfortunately this coding system is not very satisfactory. When used in the computer world RGB coding is peripheral dependant. That leads to a colour drift.

ICL Model

It would be interesting to transform the RGB to the ICL co-ordinates but it is rarely done in the common graphic softwares.

However this processing is not very difficult:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 2,7689 & 1,7519 & 1,1302 \\ 1 & 4,5909 & 0,0602 \\ 0 & 0,0565 & 5,9541 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$x = \frac{X}{X+Y+Z}$$

$$y = \frac{Y}{X+Y+Z}$$

And the inverted equation are:

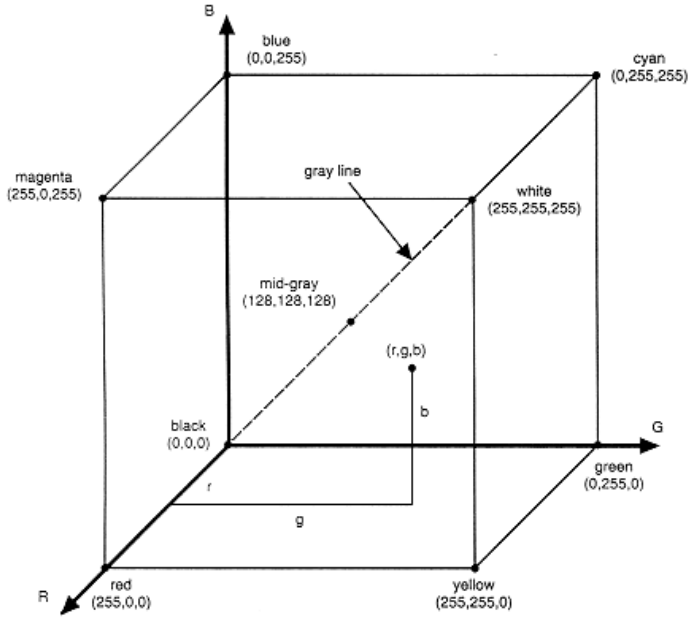
$$X = \frac{x*Y}{y}$$

$$Z = \frac{1-x-y}{y} * Y$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0,4185 & -0,1587 & -0,0778 \\ -0,0912 & 0,2524 & 0,0148 \\ 0,0009 & -0,0024 & 0,1678 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

CMY and CMYK model

These models represent the colour space build by subtractive addition of the three primary components: **C**yan (pale blue), **M**agenta (close to red) and **Y**ellow. The colour is build by the subtraction of the primary component from a white light. The colour printing comes from the reflection of an incident light on a coloured ink. In theory, the superposition of the three maximum components will produce the suppression of the light and give black colour. The printing process on paper naturally use the subtraction synthesis and is used in the printing house (to mix the inks) but rarely in the computer world in spite of its interest for the understanding of the printing.



Cube of the RGB and CMY primary colours

Each component C, M and Y is complementary of the R, G and B components. So, we can write (R is complementary to C, G to M and B to Y) the following equations:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1-C \\ 1-M \\ 1-Y \end{pmatrix}$$

or in the other direction

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1-R \\ 1-G \\ 1-B \end{pmatrix}$$

We represent the CMY model (like the RGB) in the colour space with a cube. The primary components (CMYRGB) are at the cube vertices. The white and the black stand at the two last vertices. Note that the diagonal white-black represents the grayscale. The classical extension of this system is the CMYK (K for black). This « four-colour » synthesis becomes necessary due to technical constraints (poor quality of the inks). Since the black or the grayscale are obtained by the superposition of an equal quantity of each component, we can write :

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} C-K \\ M-K \\ Y-K \end{pmatrix} + \begin{pmatrix} K \\ K \\ K \end{pmatrix}$$

with

$$C \geq K,\ M \geq K,\ Y \geq K$$

and then:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} \longrightarrow \begin{pmatrix} C' \\ M' \\ Y' \\ K \end{pmatrix} = \begin{pmatrix} C-\min(C,M,Y) \\ M-\min(C,M,Y) \\ Y-\min(C,M,Y) \\ \min(C,M,Y) \end{pmatrix}$$

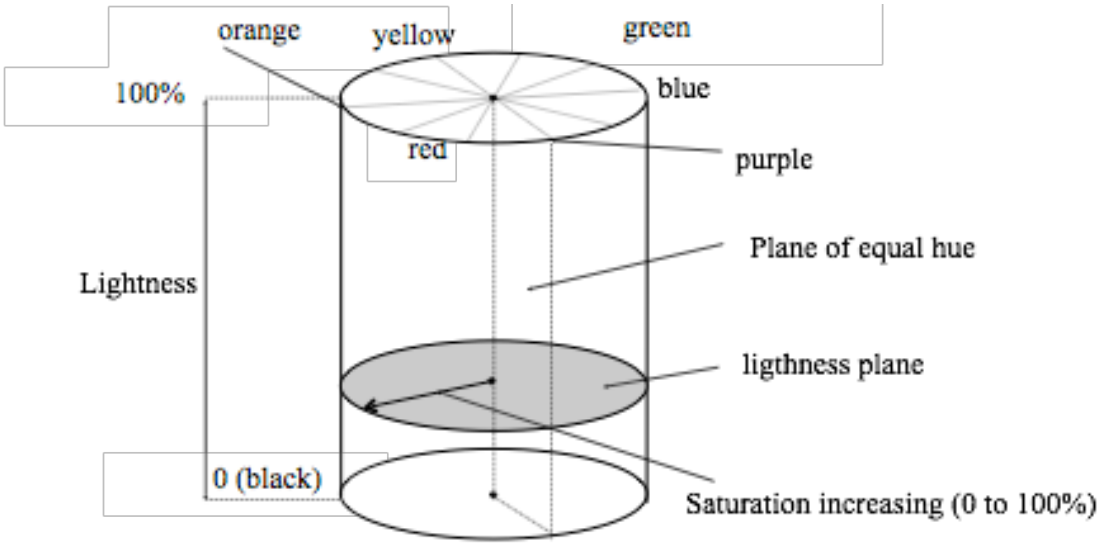
For example, the colour C (0.6, 0.2, 0.9) in the CMY model becomes C (0.4, 0.0, 0.7, 0.2) in the CMYK model because the minimum of C is min(0.6, **0.2**, 0.9) = 0.2.

Example : Let's split once again the famous clown with its four components C, M, Y et K

Clown in CMYK

HSL model

The HSL model is closer than RGB and CMY to the usual way of defining colours, that is with hue, lightness and saturation. In space we can represent the HSL with a cylinder containing all colour points (Munsell representation). The angular position measured in degrees on the circular base of the colour corresponds to hue. We can find on the circle the reference marks: red, yellow, green, cyan and magenta. The distance between the vertical axis of the cylinder and the colour point indicates saturation (maximum on the external ring and zero on the centre). The vertical co-ordinate corresponds to lightness (top down decreasing).



HSL Munsell representation

HSL model is interesting for the user as it allows him to modify the colour naturally. So it is interesting to have the cross-equations RGB to HSL. If R, G, B, S and L are defined in the interval [0,1] and H in the interval [0,360] :

$$L = \frac{Max(R,G,B) + Min(R,G,B)}{2}$$

if L<0.5:

$$S = \frac{Max(R,G,B) - Min(R,G,B)}{Max(R,G,B) + Min(R,G,B)}$$

else:

$$S = \frac{Max(R,G,B) - Min(R,G,B)}{2 - Max(R,G,B) + Min(R,G,B)}$$

if R=Max(R,G,B):

$$H = 60.\frac{G - B}{R - Min(R,G,B)}$$

if G=Max(R,G,B) :

$$H = 60.\left(2 + \frac{B - R}{G - Min(R,G,B)}\right)$$

if B=Max(R,G,B):

$$H = 60.\left(4 + \frac{R - G}{B - Min(R,G,B)}\right)$$

Let's split the Clown in its three components H, S and L

Clown in HSL

Remark: We can sometimes find the HSV or HSB model where V (value) or B (Brightness) refers to brightness. It is expressed as:

$$V = Max(R,G,B)$$

YIQ model

The National Television Standard Committee (NTSC) has developed a technique to broadcast colour images within a bandwidth similar to the black and white images bandwidth. The result of this research is the creation of a new colour space: the YIQ.

The lightness is still represented as Y whereas the two chromatic components are I and Q.

If we choose an orthogonal reference and draw the vector V with its co-ordinates I and Q, the vector extremity gives the colour point. The amplitude of the vector corresponds to the colour saturation and the angle between the vector and the Q axis corresponds to the hue. We can easily transform the YIQ space in the RGB space.

The transformation equations YIQ -> RGB and RGB -> YIQ are:

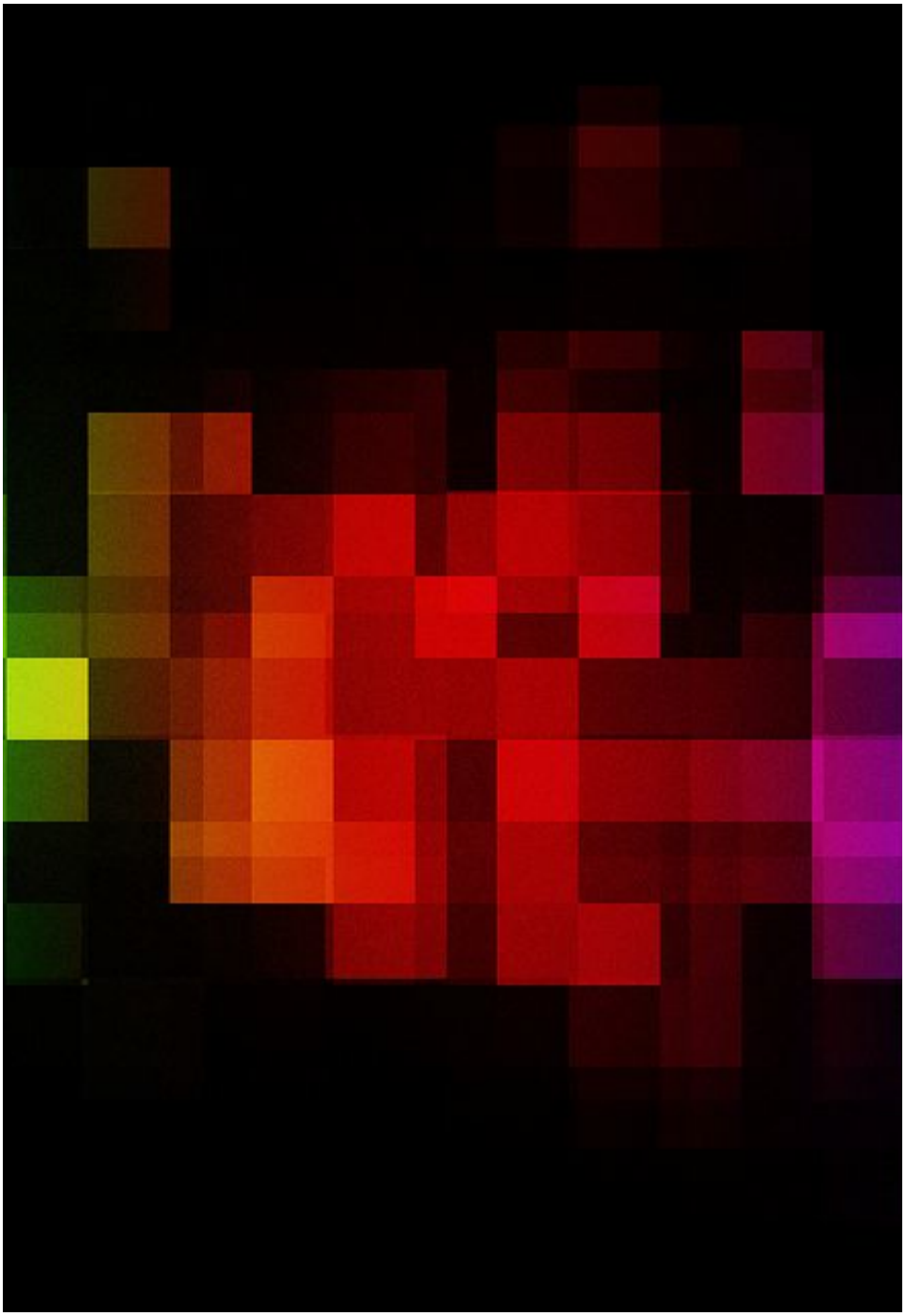
$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.522 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

And the inverted transformation equations are:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0.956 & 0.623 \\ 1 & -0.272 & -0.648 \\ 1 & -1.105 & 1.705 \end{pmatrix} \begin{pmatrix} Y \\ I \\ Q \end{pmatrix}$$

Chapitre 3

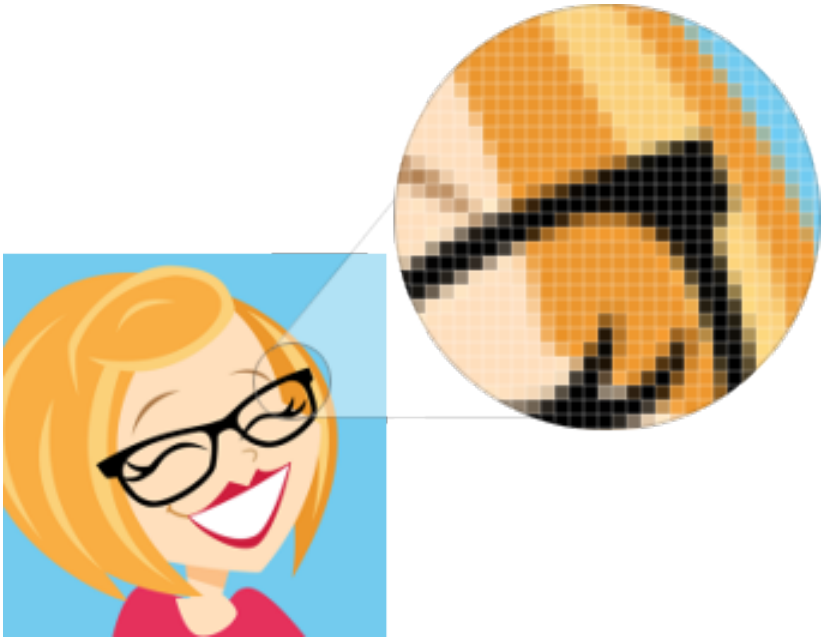
Image Digitisation



The image of the object we see is the impression of a light beam on the retina. The beam reflected by the object is made of a set of beams coming from each point of the object. The smaller the size of the points are, the greater the number of beams there will be. That gives the observer the sensation of a very high image resolution.

Photography consists of replacing the retina with a photographic plate in a darkroom. The resolution is then limited by the grain dimension.

Moreover the image is limited by the plane portion boundary of the photographic plate. Usually this surface has a rectangular shape and the « grain » can be at the outset considered as a square. The squaring will define the sharpness of the image details. Each square of the image is named a « pixel ».



What is a pixel

The image digitisation is done in a similar way. The « grain » is replaced by a transducer which measures the colour and the brightness of each pixel. Those values are stored in binary code.

The important thing is to define the squaring dimension in relationship with the desired target. First of all we have to find the optimum distance to watch the image.

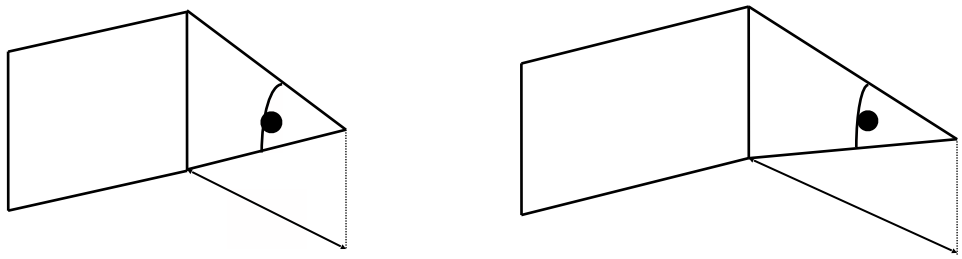
Optimum distance versus number of pixels

At the cinema the best seats are not those closest to the screen. There are two reasons:

1) The sharpness of the image is limited by the image grain and if you are close to the screen you see this grain, which is unpleasant.

2) The most sensible part of the retina (yellow spot) is a small surface, so the field of vision angle is limited. If you are too close, one part of the screen is out of the yellow spot and then the eye or the head has to move. This gives a feeling of fatigue.

So people are always looking for the good seats to watch the movie within their field of vision. To stay in the observation zone, the right angle of vision is an average of 12° according to 5 times the screen height.



4/3 and 16/9 images resolution

When looking at a 4/3 image, the observer will see the image height with an angle of

$$2.arctg\left(\frac{1.5}{15}\right) = 11.4^{\circ} = 685'.$$

If it is a 16/9 image, it will be the same angle

$$2.arctg\left(\frac{4.5}{45}\right) = 11.4^{\circ}.$$

The field of vision allows a precision of 1', so it means 685 lines. So for cinema and television 600 lines of vertical definition is probably not enough. That is why we have now high and UH definition television.

In the computer world, the user is closer to the screen and then a higher definition is needed.

Memory needed to store an image

We have seen in the previous pages that we can rebuild any colour with three fundamental components. The eye is able to perceive a maximum of 400.000 different colours. So, a colour pixel will be represented with three codes. The total number of bits have to allow a minimum of 400.000 combinations and 19 bits per pixel would allow 524.288 combinations. But usually the digital circuit deals with bytes (8 bits). So, we shall use 8, 16 or 24 bits (1, 2 or 3 bytes).

Today almost all of the images are defined with 8 bits or 24 bits. The three bytes representation is called « true colour » because the number of combinations 16.777.216 is much larger than the 400.000 perceived by the eye.

Each colour will need Nbc bits to be coded. With Npc the number of pixels in one column and Npl the number of pixels in one line, the total memory Q, in bytes, needed to store the image will be:

$$Q = Npl * Npc * \frac{Nbc}{8}$$

For a single colour image of quality of 800x600 pixels with 24 bits for each colour, we need 1.440.000 Bytes (800 x 600 x 3). With 25 images per second (SD television mode) we need almost 36 Megabytes.

That is why we shall be developing compression methods to reduce these important memory volumes.

Exemple for UHD television :

Chapitre 3

Data Compression

We have seen that the volume of memory needed to save an image may be very important. So we shall resort to adapted compression methods. That is what we are tackling. Compression consists in transforming an initial volume of data in a data set representing the same information within a smaller volume. We distinguish between two classes of compression techniques. The first does not allow any loss of data but gives relatively small compression rate and is often inadequate in image compression. The second allows high compression rate but with data loss. So the image is damaged. For a low compression rate, the damage will be imperceptible in most cases and thus acceptable in many applications.

We have seen before that an image of 800 by 600 pixels coded with 24 bits needs 1,4 Mbytes for storage. Would it be possible to code the same information with less bits? For example if some colours appear with a higher frequency than others, there will be a compression if the number of bits to code these colours is smaller.

Most plain images have correlated zones. So it is possible to have a compression by decreasing this correlation. For example: if we have an image representing a sine curve and we sample this image we obtain 256 Kbytes (considering that the image is monochrome and luminance is coded in 8 bytes). The only necessary information to be coded (for a reproduction without any loss) are the frequency, the amplitude, the phase and the orientation in the space. The total number of bytes needed to code this information is significantly reduced. It is a compression of data. Note that this information is not correlated (it is impossible to get the frequency from the amplitude).

It is in this case a mere example but the NxN pixels can be obtained by the digitalisation of an analog signal coming from the analysis line by line of the image. It is always possible to approach a signal with its Fourier series (a sum of sinusoids). The sine curves can be coded separately and replace the instant value of the signal. In practice successive portions (line or image portions MxM) of the image will be processed to compute the Fourier series factors. For each portion, we will limit the series which results in a suppression of the highest harmonics. But, those highest frequencies represent a high colour variation between two pixels and are imperceptible to the eye. Moreover those high frequencies may already have been eliminated due to the bandwidth of the sensor or of the transmitting circuit.

In fact the Fourier transform does not give the best results in the image case. The best transform is the Karhunen-Loève, but there is no fast algorithm for it. In most cases it is the cosine transformation which gives the best results which is very close to the Karhunen-Loève transform. We will see later how it can compresses images.

In conclusion the main objectives in data compression are:
- improving hard disk storage by the virtual increasing of its space
- improving transmission capacity of a data link by increasing virtually its data flow

The price to pay for this gain is the time needed to execute the compression and decompression process.

Lossless compression techniques

These techniques are based on redundancy. One character or one string which frequently appears in a file, if coded on a small number of bits, will involve a compression.

1. Coding by counting

This compression mode is derived from a simple idea. Its principle relies on counting the number of identical symbols following each other. All we need to do is to code the number of times a symbol appears followed by that same symbol. This compression mode is called RLC (Run Length Coding) or RLE (Run Length Encoding). However, these compression modes show some differences in their applications. So RLE-RLC are not standards.

Two different versions of this coding will be detailed further: Packbits (TIFF) and RLC-RLE (BMP).

2. Statistical model

Most theories dealing with compression are based on the information theory. The first works go back to 1940 with Claude Shannon. He defines the entropy (term coming from thermodynamics) of a symbol as the opposite of the logarithm of its occurrence probability. The higher the entropy the more information it contains. In order to determine the amount of information of a message by number of bits, we express entropy using logarithm in base 2:

$$NbB = -Log_2(P)$$

where NbB is the number of bits needed to code one symbol and P the probability to find the symbol in the file that is being coded.

The next table is computed with the zigzag file. Because we have this table, we can deduce the total number of bits needed to code the entire file: 1786,302. The original file contains 1024 hexa symbols to be coded on 4 bits (1024 . 4 = 4096 bits). The difference between the original and the coded file is: 4096 - 1786 = 2310 bits. This gain is theoretical because it is only possible to code a symbol on an integer number of bits.

Hexa code	Frequency	- Log_2(Probability)	NbB*Frequency
0	64	4	256
1	0	0	0
2	0	0	0
3	0	0	0
4	160	2,678	428,491
5	192	2,415	463,687
6	576	0,830	478,123
7	0	0	0
8	0	0	0
9	0	0	0
a	0	0	0
b	0	0	0
c	0	0	0
d	32	5	160
e	0	0	0
f	0	0	0
			1786,302

Total

Table 9 Entropy and number of bits

2.1. Shannon-Fano's algorithm

This algorithm created by Shannon and Fano is based on the knowledge of the occurrence probability of each symbol. From the preceding table (Page 49) we must build the code table but keep in mind that:

- different codes have different number of bits
- symbols with a weak probability have more bits to be coded than the symbols with a high probability
- in spite of the fact that codes have different lengths they must be coded in the same way

The table is created with the Shannon-Fano's tree. This tree is build with a very simple method:

1) fill in a table associating each symbol with its occurrence frequency
2) sort the symbol list by function of the frequency (from the most frequent to the less frequent)
3) divide the list in two parts so that each part of the sum of the frequencies are as close as possible
4) assign the binary number 0 to the highest part and 1 to the lowest
5) start again the 3 and 4 step for each part until you obtain unitary parts

Let 's apply this method to our previous example:

symbol	frequency
0	64
4	160
5	192
6	576
d	32

Example for the Shannon-Fano's method

First the table is sorted;

symbol	frequency
6	576
5	192
4	160
0	64
d	32

The example sorted

Then divided in two parts and marked in binary form:

symbol	frequency	
6	576	0
5	192	1
4	160	1
0	64	1
d	32	1

this continues for three more times:

symbol	frequency		
6	576	0	
5	192	1	0
4	160	1	1
0	64	1	1
d	32	1	1

symbol	frequency			
6	576	0		
5	192	1	0	
4	160	1	1	0
0	64	1	1	1
d	32	1	1	1

symbol	frequency	code			
6	576	0			
5	192	1	0		
4	160	1	1	0	
0	64	1	1	1	0
d	32	1	1	1	1

From the code column, we see that symbol 6 will be coded with one bit, 5 with 2 bits, 4 with 3 bits and finally 0 and d with 4 bits. We can now compute the necessary volume to store the file: 1*576 + 2*192 + 3*160 + 4*64 + 4*32 = 1824 (where theory gave 1786,30). We have to add the symbol/code table to this file (its volume is equal to 9 + 10 + 11 + 12 + 12 = 54). Finally after coding 1878 bits are needed for the compressed file. Whereas 4*1024 = 4096 bits were needed for the original one.

2.2. Huffman algorithm

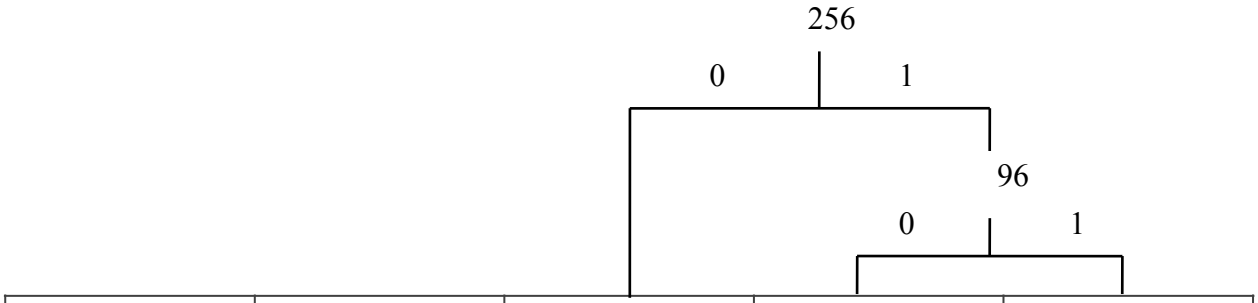
This algorithm is very similar in coding to the Shannon-Fanno algorithm (see page 50). But the building of Huffman's tree is quite different (ascending method).

The process is :

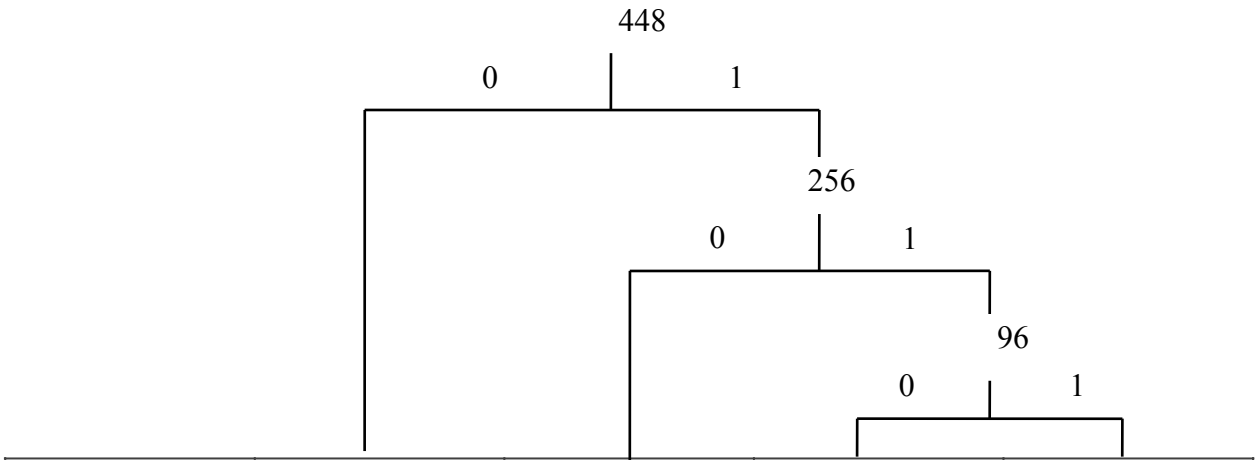
1) each symbol is considered as a leaf from which we connect branches with nodes. Each one of the nodes has its own weight (the cumulative frequency)
2) the two free leaves with the smaller weight are joined by a node (the weight of this node is the sum of the two weights), this node become a node which can be connected to another node or leaf
3) one of the two primary nodes or leaves take the value 1 and the other the value 0
4) we start again the process from the point 2) until there is only one node left

Let’s take a look at the example (sorting is not mandatory):

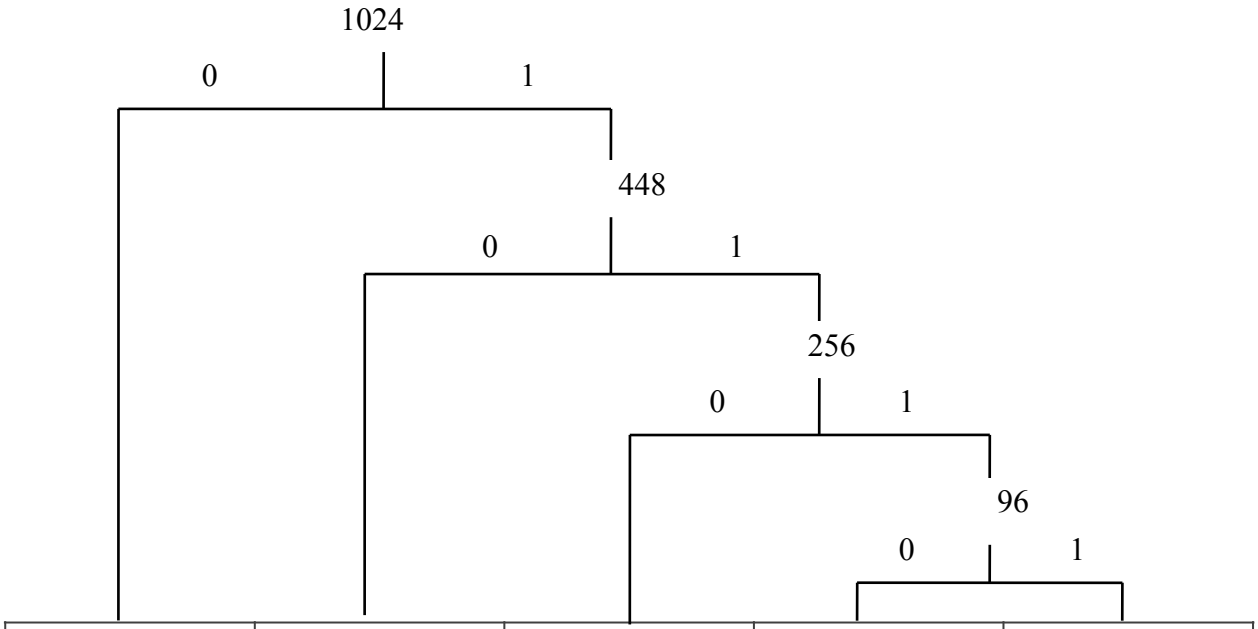
576	192	160	64	32
6	5	4	0	d



576	192	160	64	32
6	5	4	0	d



576	192	160	64	32
6	5	4	0	d



Looking at the tree from the root to the leaves, we obtain the following codes:

Symbol	Code
6	0
5	10
4	110
0	1110
d	1111

Huffman's coding result

The total number of bits of the coded file is 1*576 + 2*192 + 3*160 + 4*64 + 4*32 = 1854 + 54 (code table to be add). This method gives the same result (coding and gain) as Shannon-Fano's method. However it is important to keep in mind that Huffman's algorithm is able to compress the file in a smaller volume. That is a reason why, it is the most frequently used.

2.3. Arithmetic coding

In theory the minimum number of bits to code a symbol is a real number. The problem with the coding of Huffman and Shannon-Fano is they need an integer number of bits. An alternative method appeared fifteen years ago with the arithmetic coding.

We replace a set of symbols (strings) by a real floating number and not, like in the other methods, a single symbol by a single code. The number will be comprised between 0 and 1 and will have to be coded in binary (for a computer application).

We illustrate the arithmetic coding with the example: 6 d 5 4 6 4 5 0 6 6

As the occurrence frequency for a symbol is known, we can build this table:

symbol	probability
0	0.1
4	0.2
5	0.2
6	0.4
d	0.1

Arithmetic coding example

and then assign to each symbol an interval based on the cumulative probabilities. The extremes are 0 and 1:

Symbol	Probability	Interval [rl(S)≤ r < rh(S)]
0	0.1	0.0≤ r <0.1
4	0.2	0.1≤ r <0.3
5	0.2	0.3≤ r <0.5
6	0.4	0.5≤ r <0.9
d	0.1	0.9≤ r <1.0

Cumulative probabilities

This table helps us in coding. We will take the first symbol to be coded in the example (6) which give us the first interval of ($0.5 \leq r < 0.9$). Therefore this interval will be reduced when coding the next symbol and so on. The extremes of this interval are computed with the following equations until we reach the last symbol:

$$rh_{i+1} = rl_i + [rh_i - rl_i].rh(S_i) \quad (5.1)$$

and $$rl_{i+1} = rl_i + [rh_i - rl_i].rl(S_i) \quad (5.2)$$

So, the last minimum of the interval will code the string uniquely.

Applied to the example:

String	rl_s	rh_s	rl_i	rh_i
6	0,5	0,9	0,500000000	0,9000000000
d	0,9	1,0	0,860000000	0,9000000000
5	0,3	0,5	0,872000000	0,8800000000
4	0,1	0,3	0,872800000	0,8744000000
6	0,5	0,9	0,873600000	0,8742400000
4	0,1	0,3	0,873664000	0,8737920000
5	0,3	0,5	0,873702400	0,8737280000
0	0,0	0,1	0,873702400	0,8737049600
6	0,5	0,9	0,873703680	0,8737047040
6	0,5	0,9	0,873704192	0,8737046016

Arithmetic coding example

The unique code for this string is 0,873704192. The decoding process is relatively easy because we know from the beginning in which interval $rl(S_i) \leq r < rh(S_i)$ is the solution N_i can be found and we can then infer the first symbol S_i of the string. Next we subtract from this number N_i the lower value of the interval $rl(S_i)$ corresponding to the decoded symbol. The result is divided by the length of the interval of the same symbol. And so on, until we obtain 0. Then the decoding is over. The decoding process can be described with the following equation:

$$N_{i+1} = \frac{N_i - rl(S_i)}{rh(S_i) - rl(S_i)} \quad (5.3)$$

Let’s apply it to the resulting code from the example: 0,873704192

N_i	S_i
0,873704192	6
0,934260480	d
0,342604800	5
0,213024000	4
0,565120000	6
0,162800000	4
0,314000000	5
0,070000000	0
0,700000000	6
0,500000000	6
0,000000000	

Arithmetic decoding example

This type of coding requires the storing of the symbol/interval table. It diminishes the compression rate, specially for small files. It is also important to say that coding large real floating numbers in binary is not simple for a programmer. Moreover this algorithm is protected by copyright, so be careful when using it.

3. The models with dictionary

The compression algorithm based on the statistical model implies beginning with a statistical analysis of the file. The method with the dictionary model does not require this first step. These methods do not code a symbol on a reduced bit number but a string on the same number of bits. A relatively obvious example of compression with dictionary would be to use the Collins as the reference if the file contains a simple English text which is made up of common words of English language. Each one of the words would simply be coded by giving an index on the page and the position of the word in the dictionary.

The advantage of this method is that it does not require the storage of any corresponding table (like the statistical coding). The dictionary is build during the compression process. The compression becomes effective from the moment the dictionary reaches a reasonable size. So, this compression is not interesting for a small file.

It is in the 70th that those methods have been developed by Jacob Ziv and Abraham Lempel. Those two israelan searchers are the conceptors of the LZ77 and LZ78. An improvement has been developed by Terry Welch in 1984 (« A technique for high-performance data compression » in IEEE Computer) which gave birth to the LZW.

3.1. LZ77 compression

This compression uses the text preceding the string to be coded like a dictionary. In fact we move a window along the file to be compressed. This window is divided into two parts: the first part (text window) contains the text already compressed, the second one contains the symbols to be coded (pre-reading window). The whole window moves on the entire file.

Each string will be coded in three parts:

1) the shifting in the text window to find the string
2) the number of symbols in the string
3) the first symbol of the pre-reading buffer following the string

Example: let's compress the following text. For example the text window is 64 characters long and the pre-reading buffer is 16 characters long:

..	66666666666666666666545454066660000000000066666666666d5454506666	45454545450666666	44503 ...

Example string before coding ...

so the coding will be 23,5,'5' and the window is moves 5 symbols further and we start again:

...66666	666666666666666654545406666000000000066666666666d545450666645454	5454506666644503	..

... and after coding

3.2. LZ78 compression

If we look at the principle of coding with dictionary and its LZ77 application we immediately notice its main defects. First the text window is too small and we do not take advantage of the distant past. The second problem comes from the pre-reading buffer which allows a maximum compression equal to the length of this buffer. That is the reason why the two searchers tried to improve the LZ77 with the LZ78 algorithm.

The LZ78 totally leaves out the idea of using a moving window. The dictionary in the LZ78 contains a list of strings (illimited length - theoretically-) encountered in the file which is already compressed. The LZ78 generates (like the LZ77) a single code for a string.

This set is made of two parts:

1) The position (index) of the string in the dictionary
2) the character following the string

The big difference is that a new string is created each time a new set of symbol (minimum one) is coded. The dictionary is empty at the beginning (except for the empty string - index 0 -) and is filled in as the compression goes on.

Look back at the example: 6d5454506666666666

Coding		
String	Index	Character
6d5454506666666666		
6d5454506666666666	0	6
6d5454506666666666	0	d
6d5454506666666666	0	5
6d5454506666666666	0	4
6d5454506666666666	3	4
6d5454506666666666	3	0
6d5454506666666666	1	6
6d5454506666666666	7	6
6d5454506666666666	8	6

Dictionary	
Index	String
0	Nil
1	6
2	d
3	5
4	4
5	54
6	50
7	66
8	666
9	6666

LZ78 coding example

The result of the coding is: 060d05043430167686

3.3. LZW compression

It is in 1984 that Terry Welch edited in the IEE Computer « A Technique for high performance data compression ». His work on the Compress software for Unix gave birth to the LZW standard. The principal modification compared with its predecessors is that a dictionary exists at the beginning of the compression. It contains all the strings with only one symbol (ASCII codes). Each time a new code is emitted a new string is added to the dictionary. Two codes are particular: '256' signals the beginning of the compression and '257' signals at the end of the compression. So, the first free index in the table is 258.

Coding is made at the beginning on only 9 bits until 2^9=511 strings fill the table. At that moment the coding is at 10 bits, and so on until the coding is on 12 bits. Beyond 12 bits, the dictionary is reinitialised in order to not overload the table and to not overflow the memory.

Example: let's code the following string with the standard LZW:

11 28 05 05 05 32 58 58 11 28 05 05 32 58 58

At the beginning the dictionary contains the following values, whatever the string to be coded is:

Dictionary (table)	
index	value
0	00
1	01
2	02
3	03
4	04
5	05
...	...
255	255

Initial dictionary

The coding is made step by step in the following table:

String		Dictionary	
to be compressed	coded	index	value
11	256	255	255
28	11	258	11 28
05	28	259	28 05
05	05	260	05 05
05			
32	260	261	05 05 32
58	32	262	32 58
58	58	263	58 58
11	58	264	58 11
28	11	265	11 28
05			
05	259	266	28 05 05
32	05	267	05 32
58			
58	262	268	32 58 58
	58		
	257		

LZW coding example

And the coded string is then: 256 11 28 05 260 32 58 58 11 259 05 262 58 257, each one of decimal number is coded in binary (with 9 bits at the beginning) and paste one after the other to form a binary string that we can cut into hexa code: 80 02 C3 80 58 20 80 74 3A ...

This is of course a very simple example of the LZW compression. This algorithm becomes powerful when the dictionary is filled with a lot of long strings. It is efficient when applied to large files.

3. Lossy compression techniques

As opposed to text files or executable files, image files can be compressed with data loss. Imagine the course I am writing was so compressed. Most probably you would not be able to read these lines. In fact, as far as quality is concerned, you would not be able to distinguish an image compressed with data loss from the original one. Image and sound are the rare areas where lossy compression can be applied.

3.1. Differential modulation

This method is essentially applied to sound. Furthermore it is easy to compress in real-time. It is based on the fact that an analogue signal does not produce an abrupt variation. So, we have to code the variation and not the value. The advantage is obvious: a small variation will be coded with fewer bits than the value. If the variation is too large for the number of bits, there will be a loss of information. Those losses are often insignificant (the image is blurred). Unfortunately, the compression rate is not higher than the classical lossless compression and so it is seldom used.

3.2. Adaptive coding

In these methods we « forget » to code some pixels several times in the file. Various calculating methods can be used in order to restore the « forgotten » pixels, each of them leading to different results. Most adaptativ algorithms are based on the use of some neighbour pixels of the lost pixel to rebuild it. This saves coding. But these methods do not result in high compression rate in spite of the data loss.

3.3. DCT

The « Discrete Cosine Transform » is part of a mathematics class operations including FFT (Fast Fourier Transform). This operation converts the signal from a representation mode to another. In our particular case it transforms a spatial (X-Y plane) information into a spectral (frequency) information. The inverted transformation is called IDCT (Inverse DCT).

The two dimensional DCT equations are:

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y).Cos\left[\frac{(2x+1)i\pi}{2N}\right].Cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

where $C(x) = \frac{1}{\sqrt{2}}$ if x is equal to 0 and C(x)=1 if x>0

The inverted equation is:

$$pixel(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i).C(j).DCT(i,j).Cos\left[\frac{(2x+1)i\pi}{2N}\right].Cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

where $C(x) = \frac{1}{\sqrt{2}}$ if x is equal to 0 and C(x)=1 if x>0

The DCT applied to a MxN array will give another MxN array.

So far the method does not seem efficient because the information volume remains the same. To compress we must eliminate some data. Image contain mostly « low frequency » information, so we can eliminate the « high frequency » information. Those « high frequencies » are related to high values of the indexes i,j of the DCT. Moreover we may reduce the file with a Huffman or dictionary compression. The whole proves to be very powerful and is used in the JPG format (see page 78).

4. Compression format

4.1. BMP

This kind of format uses the counting coding (see page 48) to compress the data. The RLC or RLE exist in different forms. Two of them are described: the RLE 8 and the RLE 4.

4.1.1. RLE 8 compression:

This type of compression may be used when each pixel is coded with 8 bits (this corresponds to 256 colours in the colour map). There are two modes : encoded and absolute and each of them may appear anywhere in a bitmap.

- The encoded mode is characterised by two bytes: the first one specifies the number of consecutive pixels which have the colour given by the second byte (index on the colour map).

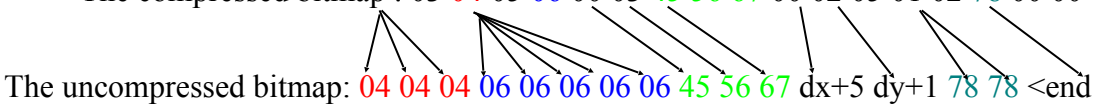
Moreover the first byte could be a 00 to signal an escape mode. This escape code indicates the end of the line, the end of the file or a delta (= moving of the cursor). In those cases the second byte takes one of the following values:

- 00:end of the line
- 01:end of the bitmap
- 02:delta; the two following bytes contain the value indicating the horizontal and vertical offset of the pixel starting from the actual position of the cursor.

- The absolute mode is generally characterised by two bytes, the first one being 00 and the second one (3 to 255) representing the number of following bytes which must be recopied.

Example:

The compressed bitmap : 03 04 05 06 00 03 45 56 67 00 02 05 01 02 78 00 00 09 1E 00 01



The uncompressed bitmap: 04 04 04 06 06 06 06 06 45 56 67 dx+5 dy+1 78 78 <end of line>
1E 1E 1E 1E 1E 1E 1E 1E 1E <end of bitmap>

4.1.2. RLE 4 compression:

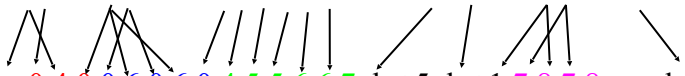
This type of compression is used when each pixel is coded on 4 bits (16 colours in the colour map). As in the RLE 8, we have two modes with similar characteristics:

As far as the encoded mode is concerned the first byte contains the number of pixels to decode. The second byte contains two colour indexes (in the first and in the second quartet). We recopy the quartets until the recopied number of quartets is equal to the value of the first byte.

In the absolute mode, the first byte is zero and the second contains the number of pixels which must be recopied. Each one is characterised by a quartet representing the colour index. The RLE 4 escape code of is the same as the RLE 8.

Example: (in hexa)

The compressed bitmap : 03 04 05 06 00 06 45 56 67 00 02 05 01 04 78 00 00 09 1E 00 01



The decompressed bitmap: 0 4 0 0 6 0 6 0 4 5 5 6 6 7 dx+5 dy+1 7 8 7 8 <end of line>
1 E 1 E 1 E 1 E 1 E 1 <end of bitmap>

Note:

1E represents 1 byte of 8 bits
1 E represents 2 quartets of 4 bits

Example: (the bitmap file zigzag.bmp):

Header:

42 4D 2E020000 00000000 76000000 28000000 20000000 20000000 0100 0400 02000000 B8010000 74120000 74120000 00000000 00000000

- (42)h = 66 is the ASCII code of the « B »
- (4D)h = 77 is the ASCII code of the « M »
- (0000022E)h = 558 is the file size in bytes
- (00000000)h
- size of the colour map + size of the heading (54 = (3C)h) in our case the size of the palette is (76)h - (36)h so (30)h = 64 bytes

- 00000028
- (20)h = 32 is the width in pixels
- (20)h = 32 is the height in pixels
- 01
- 04 bits to code one pixel
- 02 signal the compression RLE 4
- (01B8)h = 440 bytes is the size of the compressed file
- (1274)h = 4724 pixels per meter horizontally
- (1274)h = 4724 pixels per meter vertically
- (00000000)h
- 00 important colour

Colour map:

The colour map following the header is the same as the file not compressed.

00000000 00008000 00800000 00808000 80000000 80008000 80800000 80808000 C0C0C000 0000FF00 00FF0000 00FFFF00 FF000000 FF00FF00 FFFF0000 FFFFFF00

The data following the colour map are:

15 66 00 07 D5 45 45 00 04 66 00 00 15 66 00 07 54 54 54 00 04 66 00 00 15 66 00 07 D5 45 45 00 04 66 00 00 15 66 00 07 54 54 54 00 04 66 00 00 0B 00-0A 66 00 06 D5 45 45 00 05 00 00 00 00 03 45 40 07 54 02 06 09 66 00 03 54 50 08 45 00 00 .....

and are easily decompressed:

6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 <end of line>
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 <end of line>
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 <end of line>
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 <end of line>
0 0 0 0 0 0 0 0 0 0 0 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 0 0 0 0 <end of line>
4 5 4 5 4 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 5 4 5 4 5 <end of line>
.....

4.2. GIF

The Graphics Interchange Format was developed by CompuServe in 1987 to fill a need for a colour-image transfer protocol. GIF was designed to support image dimensions of up to 64000 pixels, 256 colours out of a 16-million-colour palette, multiple images in a single file, rapid decoding for on-line viewing, efficient compression, and hardware independence.

Description:

	Order
GIF Signature	1
Screen Descriptor	2 to 6
Global Colour Map	
Image Descriptor	7 to 12
Local Colour map	
Raster Data	
GIF Terminator	

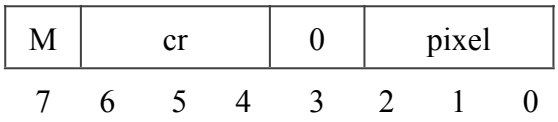
Summary of the GIF file description

Let's begin with the GIF signature and the screen descriptor block.

Order	**Number of bytes**	**Description**
1	6	GIF Signature: « GIF87a » or « GIF89a »
2	2	screen width of the image in pixels
3	2	screen height of the image in pixels
4	1	colour map description (see further)
5	1	provides the background colour index of the screen
6	1	pixel aspect ratio: bit7: classifying order of the global colour map bit 6-0: pixel aspect ratio

GIF header description

The colour map description is as follows :



Byte n°4 description

- if M=1 a global colour map follows the screen descriptor
- cr+1= number of bits of the colour resolution
- pixel+1= number of bits per pixel in the image

Following the Screen Descriptor block may be a Global Colour Map (this is optional). If the GIF file contains more than one image, then there may be one colour map for each individual image. Each entry consists of three bytes values representing the relative intensities of red, green and blue respectively.

The next portion of the GIF file is the Image Descriptor.

Order	**Number of bytes**	**Description**
7	1	comma: « , » ; Commas are used as image separators. If a Gif file has more than one image in it, each image will be separated by a comma
8	2	origin of the image top
9	2	origin of the image left
10	2	image width in pixels
11	2	image height in pixels
12	1	indicates if a local colour map exists for the image (bit 7), if the image is formatted in sequential order or in interlaced order (bit 6), the position order in the colour map (bit 5) and the number of bits per pixel-1 (bit 0-2), bits 3-4 are reserved
13	1	number of bits to code a colour
14	1	number of code to be decoded in the first part

Description of the GIF image descriptor

Local colour maps in GIF files are optional. If one is present, it will have the same format as the Global Colour Map. If a GIF file contains multiple images, it would be possible for each image to have its own local colour map.

The next part of a GIF file is the raster data, that is, the data which actually defines the image. The data consists of a series of colour index values into the global colour map or local colour map (if one exists). Each colour index value represents one pixel of the image. The pixel colour index values are stored sequentially left to right for an image row. Finally, the raster data is stored in a compressed format (LZW) to reduce the size of the GIF file.

The LZW algorithm used in GIF matches algorithmically with the standard LZW. Kep in mind that:

1. A special Clear code is defined which resets all compression /decompression parameters and tables to a start-up state. The value of this code is $2^{\text{code size}}$. For example if the indicated code size was 4 (image was 4 bits/pixel) the Clear code value would be 16 (10000 in binary). The Clear code can appear at any point in the image data stream and therefore requires the LZW algorithm to process succeeding codes as if a new data stream was starting. Encoders should output a Clear code as the first code of each image data stream.

2. An End of Information code is defined that explicitly indicates the end of the image data stream. LZW processing terminates when this code is encountered. It must be the last code output by the encoder for an image. The value of this code is Clear code + 1.

3. The first available compression code value is Clear code + 2.

4. The output codes are of variable length, starting at code size + 1 bits per code, up to 12 bits per code. This defines a maximum code value of 4095. Whenever the LZW code value would exceed the current code length, the code length is increased by one. The packing/ unpacking of these codes must then be altered to reflect the new code length.

Because the LZW compression used for GIF creates a series of variable length codes, of between 3 and 12 bits each, these codes must be reformed into a series of bytes that will be the characters actually stored or transmitted.

This provides additional compression of the image. The codes are formed into a stream of bits as if they were packed right to left and then picked off 8 bits at a time to be output. Assuming a character array of 8 bits per character and using 5-bit codes to be packed, an example layout would be similar to:

byte 1	byte 2	byte 3	byte 4	...	byte n
aaaaabbb	bbcccccd	ddddeeee	effffgg	ggghhhhh	...

Data organisation

The final part in a GIF image file will be the GIF Terminator. The Terminator is represented by a semicolon. This simply informs the program reading the GIF file that the end of the file has been reached and no images remain to be processed.

Example:

The file cindy.gif. is analysed:

47 49 46 38 37 61 : GIF87a version

04 01 : screen width (260 pixels)

B0 00 : screen height (176 pixels)

F7 : = 11110111 so M=1, cr=111=7, bit reserved=0, pixel=111=7
- M=1 indicates that a global colour map follows the descriptor
- cr+1=8 corresponds to $2^8 = 256$ is the number of colours in the map
- pixel+1=8 which is the number of bits used for one pixel in the image

00 : background colour (black)

000000 ... FFFFFF : colour map (256 colours coded on 3 bytes = 768 bytes = (300)h bytes)

2C : « ; » image separator

00 00 00 00 : image position on the screen

04 01 : image width (260 pixels)

B0 00 : image height (176 pixels)

00 : no local palette

08 : 8 bit to code a colour

FE : 254 codes to be decoded in the first part

00 99 C0 58 92 62 48 0A 19 29 12 76 48 E1 62 82 C2 00 ... 00 00 3B : the compressed data which can be decoded as following:

19	0A	48	62	92	58	C0	99	00
00011001	00001010	01001000	01100010	10010010	01011000	11000000	10011001	00000000

GIF decoding example

00	C2	82	62	E1	48	76	12	29
00000000	11000010	10000010	01100010	11100001	01001000	01110110	00010010	00101001

GIF decoding example

String			Dictionary	
binary	coded	decoded	index	value
			255	255
100000000	256		256	reserved
001001100	76	76	257	reserved
000110000	48	48	258	76 48
001001011	75	75	259	48 75
000101001	41	41	260	75 41
001000011	67	67	261	41 67
000101001	41	41	262	67 41
000110010	50	50	263	41 50
000101001	41	41	264	50 41

100001001	265	41 41	265	41 41
000011101	29	29	266	41 41 29
000101001	41	41	267	29 41
000101110	46	46	268	41 46
000010011	19	19	269	46 19
100001010	266	41 41 29	270	19 41 41 29
000000001	01	01	271	41 41 29 01
...	...	...	...	
			...	...

GIF decoding example

4.3. TIFF

In this chapter we shall here essentially study the compression of the raster data because the header has already been described in the preceding chapter (see page 41).

Three compression formats are used: the Packbits, the CCITT, the LZW. We will describe briefly the first one because it is close to the RLC-RLE compression (see page 48). The second one will be described in more details (group 3 and 4) but the third one will be scarcely tackled because it is completely described in the preceding chapter.

4.3.1. Packbits

This type of format is a variation of the RLC which has been defined by APPLE (for Mac). It allows to code solely monochrome images. This coding uses a structure of two bytes: the first indicates the number of symbols and the second indicates the symbol itself.

The first byte either positive or negative is coded into sign magnitude (the msb indicates the sign):

if the value n is positive and then comprised between 0 and 127 then the n+1 following bytes are simply recopied,

if n is negative and the following symbol is recopied -n+1 times.

The biggest number of identical bytes codable in two bytes is then 128 but the biggest string without any repetition will be coded into 129 bytes.

Observation: The zero could be coded in two different ways: 10000000 or 00000000

Example: if we code the next string: 74 38 38 72 56 56 56 38 95 18 C9 C9...

we will get the following code: 00 74 81 38 00 72 82 56 02 38 95 18 81 C9 ...

where 81 = 1000 0001 = -1 and then -(-1)+1 = 2 times 38,
where 82 = 1000 0010 = -2 and then -(-2)+1 = 3 times 56 ...

This mode appears rarely in the usual software because the compression rate is not very good and it is only applicable to the monochrome images.

4.3.2. CCITT/3

This format has been defined by the Consultative Comity International for the Telephone and the Telegraph (CCITT) for the data transfer by fax. We use in the TIFF format an adaptation of CCITT/3 for the raster data coming from images. Just like the preceding case it is only available for the monochrome image.

This format is based on the Huffman coding applied line by line from top to down and left to right. Each line will then be treated independently and called « exploration line ». Each one of those lines will be compressed in a « coded line ». It is composed of data bits, justification bits and end of line bits.

We consider the data as to be a set of white section followed by black section. If we decide that the 0 represents a zero and that the 1 represents a black we have a file described as follows (for an image with only two lines): first line, X1 white, X2 black, X3 white, X4 black, ... , second line, Y1 white, Y2 black, Y3 white, ... end of file.

We first code the number of white and then the number of black and so on. We always begin with the coding of the whites (to make no mistake in the decoding).

This coding is done in 64 base which means that a number of white (or black) between 0 and 63 will be coded with a single word. But a number between 64 and 1278 will be coded into two complementary words. The words are found in a corresponding table depending on the number and the colour (black or white). We observe that the number of white is coded with fewer bits then the number of black. This can be explained by the fact that this type of coding is used for the fax (black characters on a white background).

The justification bits are only used for the fax. They prolong artificially the transmission so as to make them compatible with the minimal duration of the tolerated transmission.

The end of the line is indicated with a 000000000001 and the end of the file (RTC) is indicated with six end of line. Those end of line and end of file are not used for the data compression in the images.

Each line begins with a new byte, so we will insert zero to fill the end of a line.

This is the code table (PL is the pattern length)

PL	White code	Black code	PL	White code	Black code
0	00110101	0000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111
16	101010	0000010111	48	00001011	000001100100
17	101011	0000011000	49	01010010	000001100101
18	0100111	0000001000	50	01010011	000001010010
19	0001100	00001100111	51	01010100	000001010011
20	0001000	00001101000	52	01010101	000000100100
21	0010111	00001101100	53	00100100	000000110111
22	0000011	00000110111	54	00100101	000000111000
23	0000100	00000101000	55	01011000	000000100111
24	0101000	00000010111	56	01011001	000000101000
25	0101011	00000011000	57	01011010	000001011000
26	0010011	000011001010	58	01011011	000001011001
27	0100100	000011001011	59	01001010	000000101011
28	0011000	000011001100	60	01001011	000000101100
29	00000010	000011001101	61	00110010	000001011010
30	00000011	000001101000	62	0010011	000001100110
31	00011010	000001101001	63	00110100	000001100111

Table 30 Code table for CCITT/3 (first part)

This is the next part of the corresponding table (from 63 to 2560):

PL	White code	Black code	PL	White code	Black code
64	11011	0000001111	1344	011011010	0000001010011
128	10010	000011001000	1408	011011011	0000001010100
192	010111	000011001001	1472	010011000	0000001010101
256	0110111	000001011011	1536	010011001	0000001011010
320	00110110	000000110011	1600	010011010	0000001011011
384	00110111	000000110100	1664	011000	0000001100100
448	01100100	000000110101	1728	010011011	0000001100101
512	01100101	0000001101100	1792	00000001000	00000001000
576	01101000	0000001101101	1856	00000001100	00000001100
640	01100111	0000001001010	1920	00000001101	00000001101
704	011001100	0000001001011	1984	000000010010	000000010010
768	011001101	0000001001100	2048	000000010011	000000010011
832	011010010	0000001001101	2112	000000010100	000000010100
896	011010011	0000001110010	2176	000000010101	000000010101
960	011010100	0000001110011	2240	000000010110	000000010110
1024	011010101	0000001110100	2304	000000010111	000000010111
1088	011010110	0000001110101	2368	000000011100	000000011100
1152	011010111	0000001110110	2432	000000011101	000000011101
1216	011011000	0000001110111	2496	000000011110	000000011110
1280	011011001	0000001010010	2560	000000011111	000000011111

Code table for CCITT/3 (second part)

Let's analyse the following example and let's apply the coding method: (the image is made of 1500 x 2 pixels and is described as studied previously) 63 whites, 1045 blacks, 380 whites, 12 blacks, end of line, 702 whites, 62 blacks, 604 whites, 132 blacks, end of file.

With the help of the table we can write the coding:

00110100 0000001110100 00001101100 00110110 01001011 0000111 00000
01100111 00110011 000001100110 01101000 0011000 000011001000 011 000000

The end of lines and of file are determinated in the header (the number of the row and columns are defined). So, there will be a checking during the decompression. In fact the end of file and of line are not indicated with the filling bits.

4.3.3. CCITT/4

This type of CCITT group 4 coding has the same objective then the CCITT/3: the fax transmission. However it is much more outstanding. In that case the coding is bidimentionnal. We don’t stop the coding at the line but we take care of the preceding line to code. The coding is of course only used with monochrome images. This format is rarely used in the TIFF format. This is why we will not spend more time on it.

4.4. JPEG

JPEG is by far the most sophisticated pixel format we discuss in this course. JPEG stands for the Joint Photographic Experts Group, a group working under the auspices of the ISO. JPEG is intended to be a standard encoding technique for digitised photographs, a function it serves well.

Unlike all of the other file formats we discuss, JPEG is lossy (see lossy compression page 63) but reconstruct an image full-colour (contrary to GIF which reconstruct an image with a palette).

JPEG is actually designed as a data-stream encoding standard, not a file standard. Hardware JPEG implementations should be fast enough to decode images as they arrive over a communication line, and JPEG has some optional sub-formats that permit initial display of low-quality image, with the quality improving as more the image is received. In this course we will not handle the progressive format.

Two JPEG file format have involved. The simpler is called JFIF, which consist of a JPEG data stream with a few mandatory element to make sure it can be decoded without having to refer to external data. Most JPEG files (.jpg) found are actually JFIF files. The other is a new TIFF sub-format introduced in the TIFF 6.0 which embed a JPEG image, possibly divided up into strips or tiles, into a TIFF file. JPEG-in-TIFF has not become popular, probably because it is considerably more complex, but not much more useful, than JFIF.

Turning a 24-bits image into a JPEG data streams is a four-step process: the colour coding, the DCT transformation, the quantization and the Huffman or arithmetic coding.

4.4.1. Colour coding

The useful first step in compressing a full-colour image is colour coding. This step is optional but obtains an immediate 50 percent saving of space, so it is almost invariably used. We have seen before that the human eye is far more sensitive to changes in brightness than it is to changes colour. Let's make the transformation RGB to YC_bC_r. This is of course a lossless change (we can reconstruct the entire RGB information with the YC_bC_r). But if we subsample the chrominance (for each 2x2 block of pixels we compute the one average C_b and C_r to store them) we have then a compression (each 2x2 block is coded with 4 Y, 1 C_b and 1 C_r). Each of the three components is now treated as a separate image plane and is transformed and compressed separately.

Colour coding does not apply the grayscale images, so their pixels value are fed directly into the second step: the DCT stage.

4.4.2. DCT

The next stage is the DCT (see page 63 for more information). In practice we have to round the values of the DCT but this rounding error is insignificant compared to quantization effects.

4.4.3. Quantization

Quantization is the major lossy step in JPEG encoding. Each of the resulting values from the DCT is a 12-bit integer. For any particular JPEG image, there is an 8x8 quantization table that is simply a table of values by which each of the DCT output value is divided. The JPEG decoder has the same table (it is stored with the image) and multiplies the values in the file by the table entry to reconstruct approximately the original DCT data. The larger the divisor, the less exactly the DCT values are reconstructed; but the smaller the quantized value is, the less space it takes in the JPEG file. The (0,0) component has a small divisor around 15, while the highest frequency components have divisor of 100 or more. By choosing an appropriate set of divisors, an application can control the trade of between the image quality and file size.

4.4.4. Huffman or arithmetic coding

The final step is to take the result of quantization and encode them compactly in a file (see Huffman algorithm page 53 and Arithmetic coding page 55).

Compression rate

Definition

The compression rate characterises the ratio between the initial volume of the file and the compressed volume. It represents the gain of the compression.

We can find two definitions of this gain: the first represents the proportion between the initial volume and the compressed volume whereas the second is simply the expression of the first in percent.

$$\tau = \frac{V_i}{V_c} \qquad \textbf{(5.6)}$$

$$\tau_{\%} = \frac{V_i - V_c}{V_i} \qquad \textbf{(5.7)}$$

The formula to go from one to the other:

$$\tau = \frac{1}{1 - \tau_{\%}} \qquad \textbf{(5.8)}$$

$$\tau_{\%} = \frac{\tau - 1}{\tau} \qquad \textbf{(5.9)}$$

So, with this rate, we can compare the efficiency of the different compression modes. It is important to say (like we are showing next) that the gain depends on the image:

- density of the information
- nature of the image (B/W, grayscale, colour)
- repetition of the different pattern

It is also important to notice that it is rarely useful to compress twice the same image which is already compressed. In general this system gives a little gain compared with the time needed for the compression. It can happen that the rate becomes negative with this method. So be careful with the double compression.

Comparative analysis

In this paragraph, we are trying to show the performance as far as the compression concerned of each one of the saving format. These compressions are realised with a software called Photostyler V2.0.

The lossy compression (jpeg) has the same result as the original one.

Beware: all the images have been compressed with their « true colour 24 bits » representation and with their colour map representation. This last one is already a kind of compression with data lost.

The examples come from different source and are very different:

« Cat » is a Kodak picture « true colour »
« Girl » is a picture with a colour map (256 colours)
« Modern » is a drawing with 256 colours
« Magritte » which is a painting scanned in 24 bits per colour

« Modern » sample :

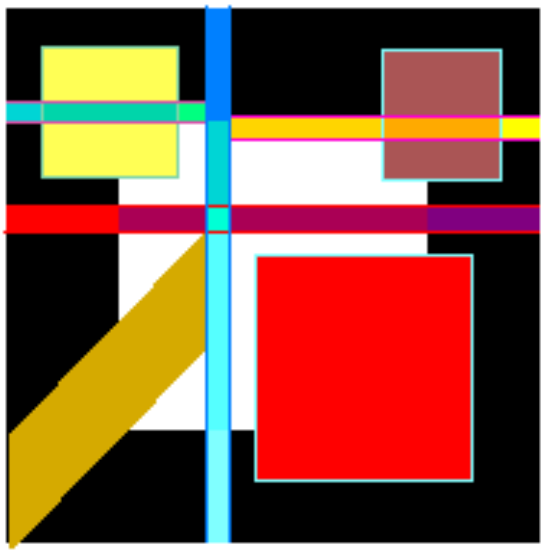


Image "Modern"

If the image is « true colour » (24 bits per colour):

File	Size (in bytes)	Compression rate
MODERNE.BMP	120.054	
MODERNE.TGA	8.628	14 or 93%
MODERNE.TIF	59.130	2 or 51%
MODERNE.JPG	10.130	12 or 92%
MODERNE.ARJ	2.094	57 or 98%
MODERNE.ZIP	1.962	61 or 98%

True colour Moderne.bmp file compressed

If the image is saved with a colour map (256 colours):

File	Size (in bytes)	Compression rate
MODERNE.BMP	41.078	
MODERNE.RLE	5.760	7 or 86%
MODERNE.TGA	5.090	8 or 88%
MODERNE.TIF	58.622	0.7 or -42%
MODERNE.GIF	4.000	10 or 90%
MODERNE.ARJ	1.525	27 or 96%
MODERNE.ZIP	1.480	28 or 96%

Colour map Moderne.bmp compressed

4.4.2. « Cat »

Image of the "cat"

If the image is « true colour » (24 bits per colour):

File	Size (in bytes)	Compression rate
CAT.BMP	1.179.702	
CAT.TGA	1.117.192	1.05 or 5%
CAT.TIF	680.500	1.7 or 42%
CAT.JPG	39.925	30 or 97%
CAT.ARJ	707.518	1.7 or 40 %
CAT.ZIP	704.818	1.7 or 40%

True colour Cat.bmp compressed

If the image is saved with a colour map (256 colours):

File	Size (in bytes)	Compression rate
CAT.BMP	394.294	
CAT.RLE	390.192	1.01 or 1%
CAT.TGA	381.252	1.03 or 3%
CAT.TIF	279.730	1.4 or 29%
CAT.GIF	225.510	1.7 or 43%
CAT.ARJ	199.236	2 or 50%
CAT.ZIP	199.551	2 or 50%

Colour map Cat.bmp compressed

4.4.3. « Girl » sample

"girl" image

If the image is « true colour » (24 bits per colour):

File	Size (in bytes)	Compression rate
GIRL.BMP	862.734	
GIRL.TGA	666.099	1.3 or 23%
GIRL.TIF	430.338	2 or 50%
GIRL.JPG	46.013	19 or 95%
GIRL.ARJ	229.703	4 or 73%
GIRL.ZIP	228.929	4 or 73%

True colour Girl.bmp compressed

If the image is saved with a colour map (256 colours):

File	Size (in bytes)	Compression rate
GIRL.BMP	288.638	
GIRL.RLE	279.002	1.03 or 3%
GIRL.TGA	275.177	1.05 or 5%
GIRL.TIF	243.200	1.2 or 16%
GIRL.GIF	188.971	1.5 or 35%
GIRL.ARJ	170.077	1.7 or 41%
GIRL.ZIP	170.896	1.7 or 41%

Colour map Girl.bmp compressed

4.4.4. « Magritte »

If the image is « true colour » (24 bits per colour):

File	Size (in bytes)	Compression size
MAGRITTE.BMP	1.271.342	
MAGRITTE.TGA	1.079.098	1.2 or 15%
MAGRITTE.TIF	352.158	3.6 or 72%
MAGRITTE.JPG	128.293	10 or 90%
MAGRITTE.ARJ	216.714	6 or 83%
MAGRITTE.ZIP	213.076	6 or 83%

True colour Magritte.bmp compressed

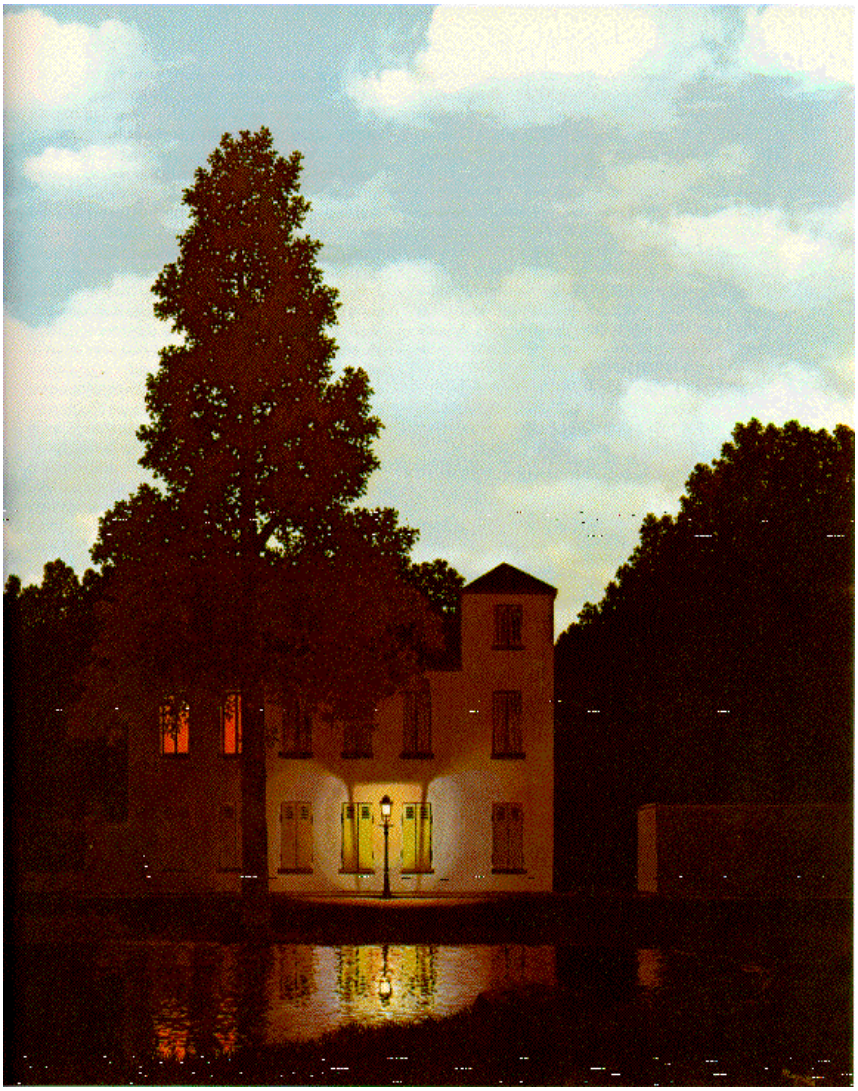


Image of the painting « L'empire des lumières » of Magritte

If the image is saved with a colour map (256 colours):

File	Size (in bytes)	Compression rate
MAGRITTE BMP	426.798	
MAGRITTE RLE	427.352	1 or 0%
MAGRITTE TGA	432.118	0.99 or -1%
MAGRITTE TIF	215.620	2 or 49%
MAGRITTE GIF	161.141	2.7 or 62%
MAGRITTE.ARJ	159.160	2.7 or 63%
MAGRITTE.ZIP	156.038	2.7 or 63%

Colour map Magritte.bmp compressed

4.4.5. Comparison

We can observe on the graphics the different compression rates with the file which is associated:

Compression rate comparison for true colour images

Compression rate comparison for palette images

If we study those results we can say as a general rule (based on the two graphics):

1) The **RLE** compression works very well with only one file (we have a high repetition of the pattern in the « modern »). In all the other cases the results are bad.

2) The **LZW** (GIF and TIFF) method gives very good results (between 1.5 and 10) but not for the TIFF. This bad score is probably due to the very big header of this kind of file. In one case we even have a compressed file bigger then the original.

3) The **JPG** reaches the best score (10 to 30). The lost information are only visible with the zoom in the « modern » case at the colour transitions.

4) The **ZIP** and **ARJ** methods (which are not image format) compress the totality of the file (including the header and the colour map) without any data lost. That explains that the LZW algorithm gives better results than TIFF and GIF.

Conclusion

In short we can say:

- if we accept the loss of data, we shall always prefer the JPG
- if we want to keep the image safe with a colour map we shall use the GIF
- if we want to keep the image safe in true colour we shall choose a TIFF file
- if we accept to use another software to compress (ZIP, ARJ or another) we shall use one of them on a BMP file

Remarks

Some « wavelet » or « fractal » methods are being worked on. Those lossy compression methods have algorithm that are very hard to use (very slow to compress). The compression rate reached are interesting but are not higher then the JPG. Moreover no standard exists. So, we will wait before opening a chapter dealing with such methods.

Chapitre 4

Image File Format

Image file formats are standardized means of organizing and storing digital images. Image files are composed of digital data in one of these formats that can be rasterized for use on a computer display or printer. An image file format may store data in uncompressed, compressed, or vector formats.

What to save ?

It is often necessary to save an image file on a hard-drive. There are three things to record:

1) information about the file
2) information about the image
3) the data, also called «raster data»

The information about the file allows us to know the saving context (coding type, compression type, document name, date, number of bytes, ...) and to recognise the file identifier.

The information about the image describes the image according to the strict sense of the term (number of pixels, width, height, colour number, …).

The information about the file together with the information about the image form the file header. The last section contains the image data differently organised according to the configuration described in the file header.

Format example : the BMP file

The BMP files (.bmp extension) are the easiest to process. They contain images defined in pixel mode, just like they are created. It is merely a bit table in which several bits represent each image pixel. This bit table is called a bitmap.

In a monochrome bitmap, each bit represents a single pixel (black or white). But in a colour bitmap, 8, 16 or 24 bits represent one pixel with a certain colour. The more bits there are per pixel, the bigger the colour number is: 8 bits -> 256 colours (28), 16 bits -> 65.536 colours (216) and 24 bits -> 16.777.216 colours (224).

Colours are coded in RGB but if a colour map (table containing colour values) exists the bits give the colour index of each pixel.

The bitmap file is made of a header and a raster data. The following table gives the header structure:

Order	Byte number	Description
1	2	Characters 'B'(66) and 'M'(77)
2	4	Total file size
3	4	0
4	4	54 + size of colour map
5	4	40
6	4	Width in pixels
7	4	Height in pixels
8	2	1
9	2	Number of bits required to code a pixel: 1,4,8 or 24 , depending of the number of colours
10	4	Compression type : 0: no compression 1: RLE 8 compression 2: RLE 4 compression
11	4	Image size (in bytes)

BMP Header

Next we find the optional colour map (see byte n°4). If present, it is a table with the following structure:

- blue byte;
- green byte;
- red byte;
- reserved byte = 0;

where blue, green and red respectively represent the intensity of each blue, green and red component on a 0 to 255 scale. The combination (0 0 0) gives a pure black, and the combination (255 255 255) gives the pure white.

Next we can read the image pixel list from top to bottom and from left to right.

Example : Let's retrieve the zigzag.bmp file (630 Kb) in the basic windows directory to illustrate the bmp file description.

ZIGZAG.BMP

This is the byte listing extracted from the file:

Header :

42 4D 76020000 00000000 76000000 28000000 20000000 20000000 0100 0400 00000000 00020000 00000000 00000000 00000000 10000000

- (42)h = 66 is the ASCII code of the « B »
- (4D)h = 77 is the ASCII code of the « M »
- (00000276)h = 630 is the file size in bytes
- (00000000)h
- (00000076)h = [header size (54 = (36)h) + colour map size], so in the example the colour map size is equal to (76)h - (36)h = (30)h = 64 bytes
- (28)h = 40
- (20)h = 32: width in pixels
- (20)h = 32: height in pixels
- 01
- 04 bits (an half byte) is used to code a pixel
- 00 indicates that the file is not compressed
- (0200)h = 512 bytes is the image size 32 * 32 * 4 = 4096 bits
  = 512 bytes

Colour map :

The first and the third following lines are found in the file, the second and the fourth (not in the file) give the colour index on which each pixel will point on.

00000000 00008000 00800000 00808000 80000000 80008000 80800000 80808000
00000000 11111111 22222222 33333333 44444444 55555555 66666666 77777777
C0C0C000 0000FF00 00FF0000 00FFFF00 FF000000 FF00FF00 FFFF0000 FFFFFF00
888888888 99999999 aaaaaaaaa bbbbbbbbb cccccccc dddddddd eeeeeeee ffffffffff

Raster data:

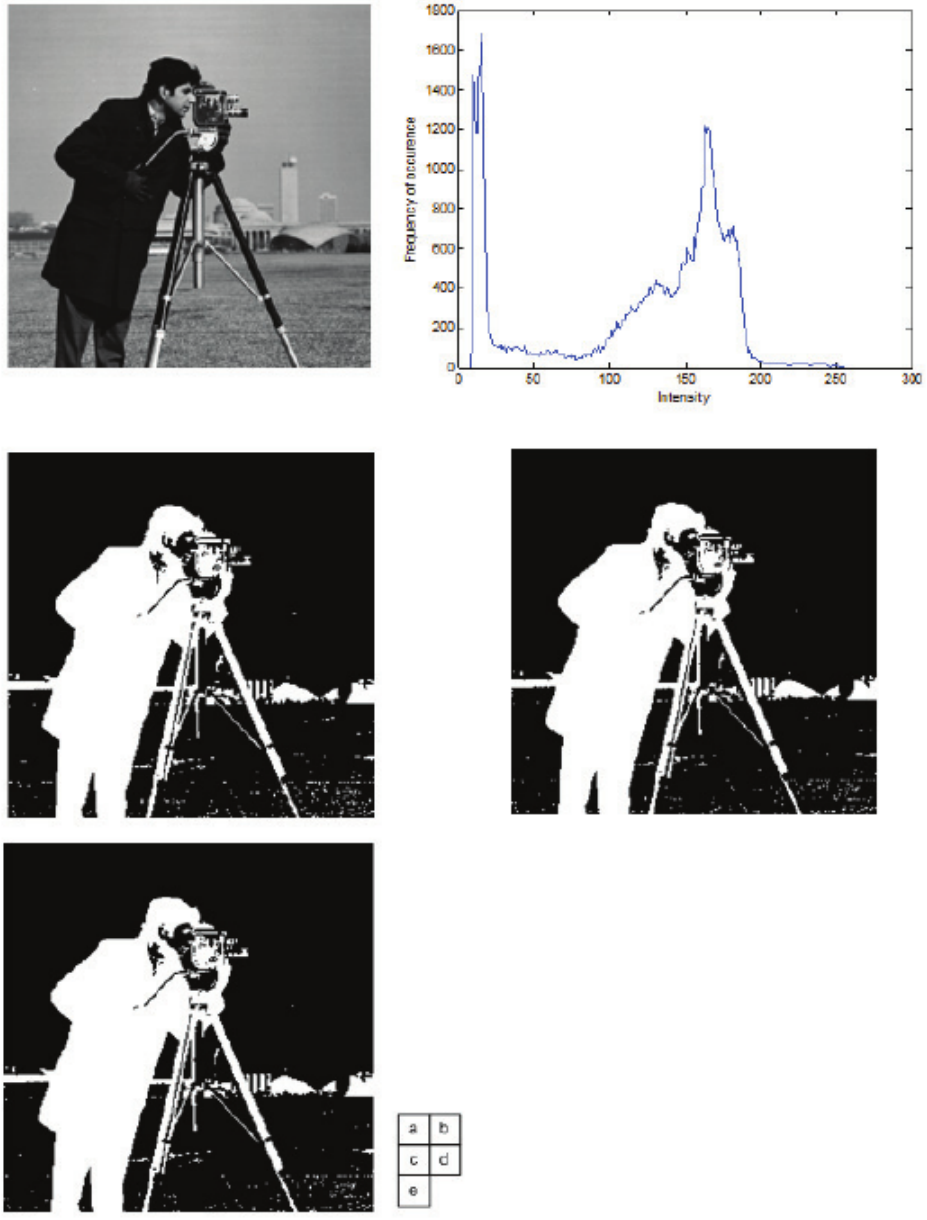
```
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6
0 0 0 0 0 0 0 0 0 0 0 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 0 0 0 0
4 5 4 5 4 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 5 4 5 4 5
5 4 5 4 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 4 5 4 5 4
4 5 4 5 4 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 5 4 5 4 5
5 4 5 4 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 4 5 4 5 4
4 5 4 5 4 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 5 4 5 4 5
5 d 5 d 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 d 5 d 5 d 5 d 5 d 5 d
6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 5 4 5 4 5 4 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 d 5 4 5 4 5 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 5 4 5 4 5 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 6 6 6
6 6 6 6 d 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 0 6 6 6 6
6 6 6 6 d 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 0 6 6 6 6
6 6 6 6 d 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 5 d 5 d 5 d 5 d 5 d 5 d 5 d 5 d 5 d 5 4 5 4 5 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 d 5 4 5 4 5 0 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 4 5 4 5 4 0 6 6 6 6
```

Raster data of the zigzag.bmp file

Chapitre 5

Image Processing Tools

The acquisition of an image is not the end of a process. Generally we will try to modify, to analyse the image to obtain better results or to get some information about the picture. Analysis can also lead to decide an automatic application for example. So it is interesting to make some process on the image. The two principal are: the restoration and the enhancement. Both use the same tools we are studying. Those tools are in fact operations defined on digital images. Here, the word *operation* refers to a rule which uniquely transforms one image into an other.



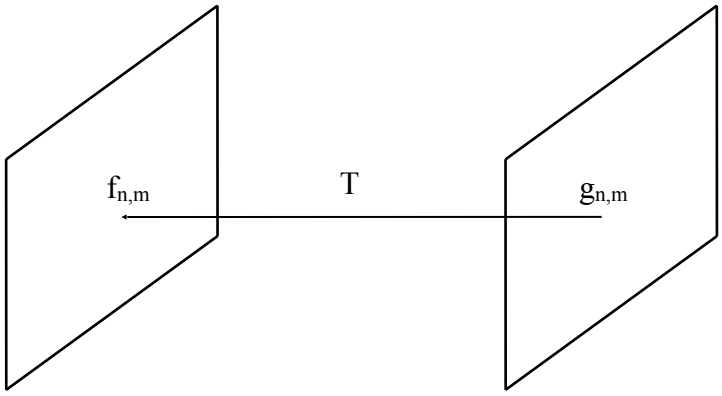
Point Operation

The images considered are defined on an NxM lattice (n,m) with n=0,1,...N-1 and m=0,1,...M-1. The domain of the amplitudes of the pixels will be referred to as the grey scale G.

The « points operations », also called « pixel-to-pixel » operations, are image operations in which the amplitudes of individual pixels are modified according to some rules. the operation is denoted mathematically by:

$g_{n,m} = T_{n,m}(f_{n,m})$ if the operator Tn,m depends on the position of the pixel

or by $g_{n,m} = T(f_{n,m})$ if the rule does not depend on the position



Point operation

4.5. Monadic operations

Monadic or unary operations are pixel-to-pixel operations with a single operand image.

4.5.1. Histogram

A graph showing the probability of each particular pixel value in an image is known as a *histogram*. To accumulate a histogram we need first to count the total number of pixels with each possible value which occurs in the image (denoted n_f). Then we divide each of these counts by the total number of pixels in the image (NxM) to obtain the probability value. All these values are represented in a graph probability = h_f (f), where f represent the grey level.

$$h_f(f) = \frac{n_f}{N.M} \qquad \textbf{(6.1)}$$

It can also be used for only one component of a colour. The histogram allows sometimes to choose a LUT or to choose a colour map. That will be studied later in this chapter.

We can also define the cumulative histogram $H_f(f)$ which is the percentage of pixels having grey levels less than or equal to f.

$$H_f(f) = \sum_{i=0}^{f} h_f(i) \qquad \textbf{(6.2)}$$

Example: From now on the examples we will use this area of number (it can represent the image greyscales):

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	0	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	0	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Numerical example

The histogram example can be represented by the following graphic:

Histogram example

4.5.2. Projection

A … https://www.keyence.com/ss/products/vision/visionbasics/basic/size/

4.5.3. Thresholding

One of the commonest requirements, in pixel transformation, is to generate a binary representation of an image. The obvious application is image printing and image analysis. It only consists in giving each pixel a zero bit or a 1 bit (black or white). Many different techniques exist (essentially for printing) but the most easy consist in deciding of a threshold. If the value of the pixel is higher than the threshold the bit becomes 1 else it becomes 0. The histogram can help us to find the best threshold. Let's see in our previous example which point will probably give the best result. It seems that a point between 120 and 240 will probably be the right threshold to binary this image.

Example: let's choose a threshold of 180 on this example:

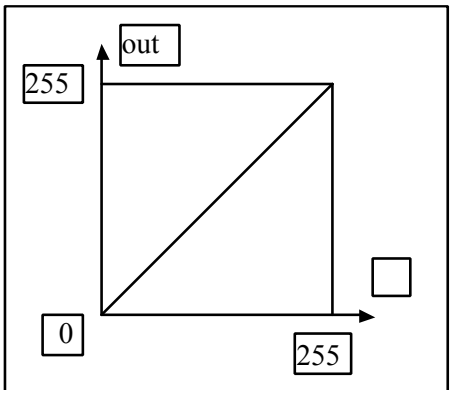
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	0	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	0	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	255	255

Threshold example

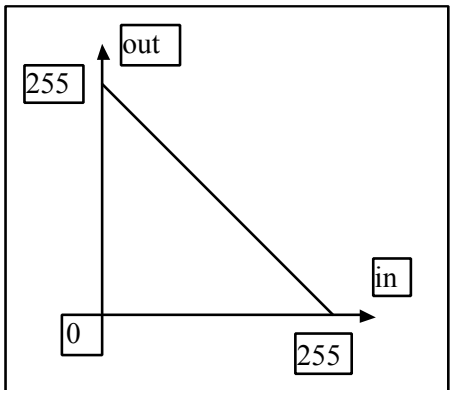
4.5.3. LUT

Those **L**ook **U**p **T**ables allow to associate one incoming value with an outcoming value. The function T is used to make the connection. We can modify each pixel value just as wanted. If we have a colour image we will use three tables (depending on the representation mode HSL, RGB, ...).

If we use the grayscale image and if we do not want to modify the image with a LUT, it would be in fact a mere line inclined with 45° (Figure 40). The LUT which inverts (negative) the image will be represented with a line inclined with -45° (Figure 41).



LUT example



LUT example

The advantages of using a LUT operation above other implementations are the flexibility and its speed. It only needs a re-fill of the table to change the operation. Moreover LUT is a fast implementation since it only need one indirect addressing per pixel.

The implementation can also be realised in hardware. Often, the video output circuitry of a computer vision system provided with (hardware) LUT. The contrasts and colours in those images can be manipulated easily.

The usual following examples are useful:

Name	Mathematical description	LUT	Parameters
Thresholding	$g_{n,m} = \begin{cases} 1 \ if \quad f_{n,m} > t \\ 0 \ elsewhere \end{cases}$		t:threshold
Multi-thresholding	$g_{n,m} = l_i$ if $t_{i-1} < f_{n,m} \le t_i$		t_i: i-th threshold l_i: i-th level ($t_i < t_{i+1}$)
Floor operation	$g_{n,m} = \begin{cases} f_{n,m} \text{ if } f_{n,m} > c_f \\ c_f \text{ elsewhere} \end{cases}$		c_f: floor
Ceiling operation	$g_{n,m} = \begin{cases} c_c \text{ if } f_{n,m} > c_c \\ f_{n,m} \text{ elsewhere} \end{cases}$		c_c: ceiling
Clip operation	$g_{n,m} = \begin{cases} c_c \text{ if } f_{n,m} > c_f \\ c_f \text{ if } f_{n,m} < c_f \\ f_{n,m} \text{ elsewhere} \end{cases}$		c_c: ceiling c_f: floor
Offset operation	$g_{n,m} = f_{n,m} + offset$		offset
scale operation	$g_{n,m} = scale \cdot f_{n,m}$		scale
square	$g_{n,m} = f_{n,m}^2$		
square root	$g_{n,m} = \sqrt{f_{n,m}}$		

Some monadic operation examples

4.2. Dyadic operations

Dyadic operations are pixel-to-pixel operation with two operand images:

$$h_{n,m} = T(f_{n,m}, g_{n,m}) \qquad \textbf{(6.3)}$$

4.2.1. Sum

The sum of two images is computed by addition of the pixel values of each image. We can also give a scale for each image (s_1 and s_2):

$$h_{n,m} = s_1 . f_{n,m} + s_2 . g_{n,m} \qquad \textbf{(6.4)}$$

4.2.2. Minimum and maximum

These operations arise from a comparison of two or more images. This could be used, for example, to discover the maximum and minimum values which uccur through a sequence of images taken of the same object over a period of time.

$$h_{m,n} = \min(f_{m,n}, g_{m,n}) \qquad \textbf{(6.5)}$$

$$h_{m,n} = \max(f_{m,n}, g_{m,n}) \qquad \textbf{(6.6)}$$

4.2.3. Logical operations

Just as standard arithmetic operations can be defined on a pixel by pixel basis for images, so can the usual bitwise operations. These include **and**, **or**, **exclusive or**, and **not**. In each case, the operation is applied to each pixel in turn and the result is placed in the corresponding position in the result image.

$$h_{m,n} = f_{m,n} \text{ or } g_{m,n} \qquad \textbf{(6.7)}$$

$$h_{m,n} = f_{m,n} \text{ and } g_{m,n} \qquad \textbf{(6.8)}$$

3. Neighbourhood operations

In contrast with pixel-to-pixel operation, an output pixel from a neighbourhood operation depends on a subset of the input pixels.

A local neighbourhood of a pixel (n,m) id the collection of pixels in a subarea of the image enclosing the pixel (n,m). Usually, the subarea is chosen rectangular with (n,m) in the centre of the area. Suppose that the size of the neighbourhood is (2K+1)x(2L+1), then the local neighbourhood $\varphi_{n,m}$ of the pixel (n,m) in the image f is:

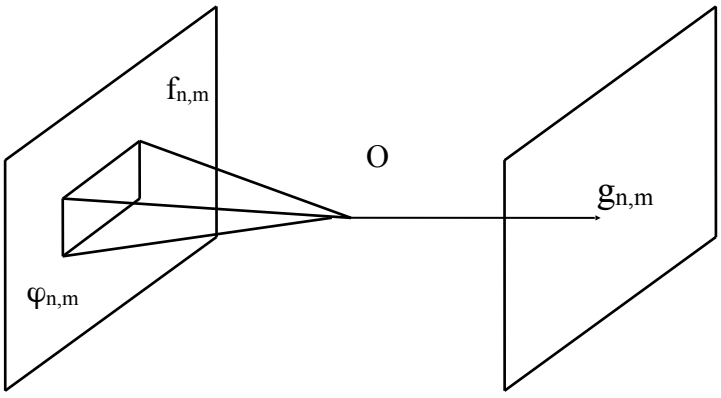
$$\varphi_{n,m} = \qquad \textbf{(6.9)}$$

We can write for those operations:

$$g_{n,m} = O(\varphi_{n,m}) \qquad \textbf{(6.10)}$$

This operation will be called *linear* when:

$$\textbf{(6.11)}$$



Neighbourhood operation

3.1. Masks

A mask is define with an array $h_{kL,}$ a norm N and eventually a threshold (σ).

$$h = \begin{bmatrix} h_{-K,-L} & h_{-K,-L+1} & \cdots & h_{-K,L} \\ h_{-K+1,-L} & & & \cdots \\ \cdots & & & \\ h_{K,-L} & \cdots & \cdots & h_{K,L} \end{bmatrix} \qquad \textbf{(6.12)}$$

h is the mask used to compute the pixels $g_{n,m}$ with the following equations:

$$g_{n,m} = \frac{1}{N} \sum_{i=-K}^{K} \sum_{j=-L}^{L} h_{i,j} \cdot f_{n+i,m+j} \qquad \textbf{(6.13)}$$

In a general way N is equal to the sum of the values in the mask and then:.

$$N = \sum_{i=-K}^{K} \sum_{j=-L}^{L} h_{i,j} \qquad \textbf{(6.14)}$$

If a threshold σ is define the new value is:

$$g_{n,m} \text{ if } |g_{n,m} - f_{n,m}| > \sigma \qquad \textbf{(6.15)}$$

$$\text{or } f_{n,m} \text{ if } |g_{n,m} - f_{n,m}| \leq \sigma \qquad \textbf{(6.16)}$$

Each pixel in the image can be computed with this method, all you have to do is to begin with the first pixel and move on the mask to the left and so on with the following pixels until the end of line. Then we do it again with the next line and so on until the end of the image. Those equations are valid when $n \geq K$, $n < N-K$, $m \geq L$ and $m < M-L$. An ambiguity occurs near the edges of the image since here the definition of the local neighbourhood is not valid.

Three methods to solve this ambiguity are known:

- discarding the pixels near the edge of the output image. Here, the image is regarded as not being defined for n<0, n≥N, m<0 and m≥M and then the mask will only be performed for pixels $g_{n,m}$ for which n ≥ K, n < N-K, m ≥ L and m < M-L.
- Padding the input image with zeroes. Outside the definition area (n<0, n≥N, m<0 and m≥M) the image is padded with zeroes.
- The image can be periodically repeated.

3.2. Convolution

In the equation (6.10), the operation is called *space invariant* if O(.) does not depend on the pixel position (n,m). If, in addition the operation is also linear, then it is a ***discrete convolution***. Masks are used to compute the convolution.

3.2.1. Low-pass filtering

The low pass filters allow low frequencies to pass unchanged but attenuate high frequencies. They are also called *smoothing filters* because they reduce the noise in an image.

3.2.1.1.Local mean

A local mean (also called averaging operator) can be calculated for any collection of value. The mean value is computed by summing all the pixels of the image into a single value and then dividing it by the number of pixels in the image. It can be applied for a local portion of the image:

$$g_{n,m} = \frac{1}{(2K+1)(2L+1)} \sum_{i=-K}^{+K} \sum_{j=-L}^{+L} f_{n+i,m+j} \quad \textbf{(6.17)}$$

Example: let's take the following values for a pixel and its neighbours: the old value is 12 and the height neighbours are equal to 59, 72, 67, 76, 54, 73, 69 and 71, the mean value is then:

$$g_{2,2} = \frac{59+72+67+76+12+54+73+69+71}{9} \approx 61 \quad \textbf{(6.18)}$$

We can replace the old value with the new one or we can also change it only if the difference between the old and the new is higher than a threshold. In that case, if the deviation $\Delta = |f_{n,m}-g_{n,m}|$ is higher than σ then $f_{n,m}$ is replaced with $g_{n,m}$. In the following example if the threshold is 40 and $\Delta = 61 - 12 = 49$ is higher than σ, then the value 12 is replaced with 61. In that case the function is no longer linear, then it is not a convolution if a threshold is used.

The local mean operation can be computed by using a mask where the values $h_{i,j}=1$ (i=1,2,...,K and j=1,2,...L) and N=9.

59	72	67
76	12	54
73	69	71

Table 43 Example before local mean operation ...

59	72	67
76	61	54
73	69	71

Table 44 ... and after the local mean transformation

If we apply the mask for local mean to the larger example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	22	33	42	55	63	70	77	87	97	100	100	100	100	100	100	100	100
10	22	33	42	55	63	70	77	87	97	100	100	100	100	100	100	100	100
10	22	33	42	55	63	70	77	104	130	150	150	150	150	150	150	150	100
10	22	33	42	55	63	70	77	120	163	200	200	200	200	200	200	200	250
10	22	33	42	55	63	70	77	137	197	250	250	250	250	250	250	250	250
10	22	33	42	55	63	70	77	137	197	250	250	250	250	250	250	250	250
10	22	33	42	55	63	70	77	137	197	250	250	250	222	222	222	250	250
10	22	33	42	55	63	70	77	137	197	250	250	250	222	222	222	250	250
10	22	33	42	55	63	70	77	137	197	222	222	222	222	222	222	250	250
10	22	33	42	55	63	70	77	137	197	222	222	222	250	250	250	250	250
10	22	33	42	55	63	70	77	137	197	222	222	222	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Local mean example

3.2.1.2. Gaussian filtering

Gaussian filtering can be implemented in various ways. The 2-dimensional Gaussian function is separable. This implies that the Gaussian filter can be implemented as a cascade of row and column operation. For instance, a Gaussian filter is approximated by the mask:

N=16

1	2	1
2	4	2
1	2	1

Gaussian filter mask

Let's apply this filter to the example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	23	33	43	54	65	70	76	88	98	100	100	100	100	100	100	100	100
10	23	33	43	54	65	70	76	88	98	100	100	100	100	100	100	100	100
10	23	33	43	54	65	70	76	97	126	138	138	138	138	138	138	138	100
10	23	33	43	54	65	70	76	116	182	213	213	213	213	213	213	213	250
10	23	33	43	54	65	70	76	125	210	250	250	250	250	250	250	250	250
10	23	33	43	54	65	70	76	125	210	250	250	250	250	250	250	250	250
10	23	33	43	54	65	70	76	125	210	250	250	250	234	219	234	250	250
10	23	33	43	54	65	70	76	125	210	250	250	250	219	188	219	250	250
10	23	33	43	54	65	70	76	125	210	234	219	234	234	219	234	250	250
10	23	33	43	54	65	70	76	125	210	219	188	219	250	250	250	250	250
10	23	33	43	54	65	70	76	125	210	234	219	234	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Gaussian filter result

3.2.2. High-pass filtering

The high pass filters allow high frequencies to pass unchanged but attenuate low frequencies. Actually, the term is still applied even when the filter actively boosts the high frequency components of the image. Since low-pass filtering is complementary to high-pass filtering the latter can be achieved by subtracting of a low-pass filtered image from its original:

$$H_{high-pass}(u,v) = 1 - H_{low-pass}(u,v) \qquad \textbf{(6.19)}$$

For instance, the counterpart of the Gaussian low-pass filter (Table 46) is:

N=16

-1	-2	-1
-2	12	-2
-1	-2	-1

High-pass filter example

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100
10	3	0	3	0	5	0	0	2	3	0	0	0	0	0	0	100
10	3	0	3	0	5	0	0	2	3	0	0	0	0	0	0	100
10	3	0	3	0	5	0	0	0	0	0	0	0	0	0	0	100
10	3	0	3	0	5	0	0	0	68	38	38	38	38	38	38	250
10	3	0	3	0	5	0	0	0	40	0	0	0	0	0	0	250
10	3	0	3	0	5	0	0	0	40	0	0	0	0	0	0	250
10	3	0	3	0	5	0	0	0	40	0	0	0	16	31	16	0	250
10	3	0	3	0	5	0	0	0	40	0	0	0	31	0	31	0	250
10	3	0	3	0	5	0	0	0	40	16	31	16	16	31	16	0	250
10	3	0	3	0	5	0	0	0	40	31	0	31	0	0	0	0	250
10	3	0	3	0	5	0	0	0	40	16	31	16	0	0	0	0	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

High-pass filter result

These two high pass filter examples will be used further:

N=1

-1	-1	-1
-1	9	-1
-1	-1	-1

High pass filter example 1

N=1

-1	-2	-1
-2	13	-2
-1	-2	-1

High pass filter example 2

and the effect of the high pass filter 1 on our example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	55	0	75	8	123	70	17	117	130	100	100	100	100	100	100	100	100
10	55	0	75	8	123	70	17	117	130	100	100	100	100	100	100	100	100
10	55	0	75	8	123	70	17	0	0	0	0	0	0	0	0	0	100
10	55	0	75	8	123	70	17	0	255	255	255	255	255	255	255	255	250
10	55	0	75	8	123	70	17	0	255	250	250	250	250	250	250	250	250
10	55	0	75	8	123	70	17	0	255	250	250	250	250	250	250	250	250
10	55	0	75	8	123	70	17	0	255	250	250	250	255	255	255	250	250
10	55	0	75	8	123	70	17	0	255	250	250	250	255	0	255	250	250
10	55	0	75	8	123	70	17	0	255	255	255	255	255	255	255	250	250
10	55	0	75	8	123	70	17	0	255	255	0	255	250	250	250	250	250
10	55	0	75	8	123	70	17	0	255	255	255	255	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

High pass filter 1 effect example

3.2.3. Other linear filters

Different types of mask define different other types of filtering. Particularly the following masks are known for the pattern detection. They are called « Sobel filters », « Prewitt filters » and « isotropic filters ».

-1	0	1
-2	0	2
-1	0	1

Component of a Sobel filter

-1	-2	-1
0	0	0
1	2	1

Component of a Sobel filter

-1	0	1
-1	0	1
-1	0	1

Component of a Prewitt filter

0

-1	-1	-1
0	0	0
1	1	1

Component of a Prewitt filter

1	0	1
-√2	0	√2
-1	0	1

Component of an Isotropic filter

-1	-√2	-1
0	0	0
1	√2	1

Component of an Isotropic filter

-1	2	-1
-1	2	-1
-1	2	-1

Component of a vertical filter

-1	-1	-1
2	2	2
-1	-1	-1

Component of an horizontal filter

In the first column the filters allow to detect the vertical shape and in the second column they allow to detect horizontal shape. We can also combined the use of two or more masks.

Remarks:

- the vertical filters give similar result and then only one example will be illustrated.
- we find in some books the name « Sobel filters » instead « Prewitt filters ». So be careful when using these names.

Example: let's apply the Sobel filters to observe its effect on the example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100
10	80	80	80	96	80	8	80	116	40	0	0	0	0	0	0	100
10	80	80	80	96	80	8	80	116	40	0	0	0	0	0	0	100
10	80	80	80	96	80	8	80	255	190	0	0	0	0	0	0	100
10	80	80	80	96	80	8	80	255	255	0	0	0	0	0	0	250
10	80	80	80	96	80	8	80	255	255	0	0	0	0	0	0	250
10	80	80	80	96	80	8	80	255	255	0	0	0	0	0	0	250
10	80	80	80	96	80	8	80	255	255	0	0	0	0	0	250	0	250
10	80	80	80	96	80	8	80	255	255	0	0	0	0	0	255	0	250
10	80	80	80	96	80	8	80	255	255	0	0	250	0	0	250	0	250
10	80	80	80	96	80	8	80	255	255	0	0	255	0	0	0	0	250
10	80	80	80	96	80	8	80	255	255	0	0	250	0	0	0	0	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Vertical Sobel filter example

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
10	0	0	0	0	0	0	0	150	255	255	255	255	255	255	255	100
10	0	0	0	0	0	0	0	150	255	255	255	255	255	255	255	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	250	255	250	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	250	255	250	0	0	0	0	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Horizontal Sobel filter

Some other filters are also known to detect oblique shape. For example the following filter detect shape inclined with 45°:

N=1

2	1	0
1	0	-1
0	-1	-2

Sobel 45° filter

With the following result on our example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	255	250	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	250	0	0	0	250
10	0	0	0	0	0	0	0	0	0	255	250	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	250	0	0	0	0	0	0	250
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Sobel 45° filter result

The omnidirectionnal detection masks are the Laplacian's with the following values:

1	-2	1
-2	4	-2
1	-2	1

Laplacian mask 1

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacian mask 2

0	-1	0
-1	4	-1
0	-1	0

Laplacian mask 3

We can see the effect of the second one on the example:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	30	0	30	0	54	0	0	27	30	0	0	0	0	0	0	0	100
10	30	0	30	0	54	0	0	27	30	0	0	0	0	0	0	0	100
10	30	0	30	0	54	0	0	0	0	0	0	0	0	0	0	0	100
10	30	0	30	0	54	0	0	0	255	255	255	255	255	255	255	255	250
10	30	0	30	0	54	0	0	0	255	0	0	0	0	0	0	0	250
10	30	0	30	0	54	0	0	0	255	0	0	0	0	0	0	0	250
10	30	0	30	0	54	0	0	0	255	0	0	0	250	250	250	0	250
10	30	0	30	0	54	0	0	0	255	0	0	0	250	0	250	0	250
10	30	0	30	0	54	0	0	0	255	250	250	250	250	250	250	0	250
10	30	0	30	0	54	0	0	0	255	250	0	250	0	0	0	0	250
10	30	0	30	0	54	0	0	0	255	250	250	250	0	0	0	0	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Laplacian filter example

4. Filtre de Canny

https://en.wikipedia.org/wiki/Canny_edge_detector

3.3. Non linear filters

4.1.Median filter

The most common of the non linear filters is the median filter. This filter is useful to reduce the random noise in the images. We use a window on the image to compute the result of the median filtering. Each position of the mask will compute the median value of the pixel neighbouring. The old value will always be replaced with the new one (no deviation).

The median is the value which occurs in the middle of the sorted list of the window.

Example: If the values in the window are: 59, 72, 67, 76, 76, 12, 54, 73, 69, 71.
If the list is sorted it gives: 12, 54, 59, 67, **69**, 71, 72, 73, 76.
The median value is 69.

59	72	67
76	12	54
73	69	71

Window before filtering

59	72	67
76	**69**	54
73	69	71

and after the median filtering

If we apply this method (it is not a mask) to the example, it results:

10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	100	100	100	100	100	100	100	100
10	25	30	45	50	69	70	71	90	100	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250
10	25	30	45	50	69	70	71	90	250	250	250	250	250	250	250	250	250

Median filter example

For the unvarying area the median is close to the average and the filter reduces the noise. The advantage of this system is to keep the sharp of the shape. But the disadvantage of this filter type appears when the ratio signal-noise is small because it produces breaks in the image and then false outlines.

Améliorer en ne remplaçant la valeur du pixel que si elle est vraiment différente (fixer un seuil) du pixel actuel.

To improve this filter we can use the MTM filter (Modified Trimmed Mean filter) where all the values in the window change. The pixel on the centre take the median value and all the

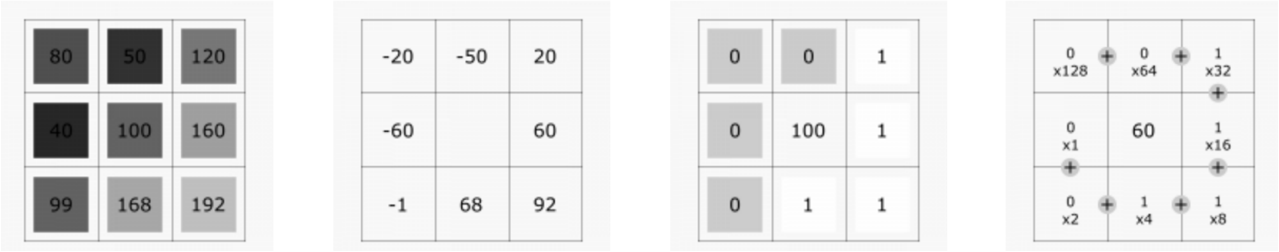
other pixels take as new value the average of the values and the median. The window has to move on all the pixels but not two times on the same.

Local Binary Pattern

Les Local Binary Patterns (motifs binaires locaux) ont été définis avec comme objectif de pouvoir repérer des motifs répétitifs dans une image. Principalement utilisés pour détecter et reconnaître des textures, ils peuvent être exploités localement pour reconnaître des contours, des coins, ou des formes.

Pour calculer la “dérivée LBP” d’une image, on procède de la manière suivante :

Pour chaque pixel p de l’image disposant de 8 voisins
Soit une valeur LBP sur 8 bits numérotés b0 à b7
Pour chacun des 8 voisins v (0 à 7) du pixel considéré, dans le sens horaire
Calculer la différence d’intensité entre ce voisin v et le pixel p
Si la différence est positive, mettre le bit correspondant de LBP à 1
Sinon mettre le bit correspondant de LBP à 0



Exemple de LBP[1]

[1] https://fr.eggs-iting.com/blog/prototypage/detection-des-oeufs-dans-un-nid-de-poule.html

3.4. Zoom

3.4.1. Enlarging (zoom in)

To enlarge by n an image we reproduce n times each pixel in the direction to enlarge. But we can also interpolate the pixel to add. If we use a linear interpolation, we have for a zoom in by 2:

a	b
c	d

a1	a2	b2	b1
a3	a4	b4	b3
c3	c4	d4	d3
c1	c2	d2	d1

with : $a_1 = a,\ b_1 = b,\ c_1 = c,\ d_1 = d,$

$$a_2 = \frac{a+b}{2},\ b_2 = \frac{a+b}{2},\ c_2 = \frac{c+d}{2},\ d_2 = \frac{c+d}{2}$$

$$a_3 = \frac{a+c}{2},\ b_3 = \frac{b+d}{2},\ c_3 = \frac{a+c}{2},\ d_3 = \frac{b+d}{2}$$

$$a_4 = \frac{a+b+c+d}{4},\ b_4 = \frac{a+b+c+d}{4},\ c_4 = \frac{a+b+c+d}{4}$$

and $d_4 = \frac{a+b+c+d}{4}$

The operation can be demonstrated on a part of the previous example:

69	70	71	90	100	100	100	100
69	70	71	90	100	100	100	100
69	70	71	90	100	100	100	100
69	70	71	90	250	250	250	250
69	70	71	90	250	250	250	250
69	70	71	90	250	250	250	250

Example to be enlarged

and the result of the zoom in is:

69	70	70	70	71	81	81	90	100	100	100	100	100	100	100	100
69	70	70	70	71	81	81	90	100	100	100	100	100	100	100	100
69	70	70	70	71	81	81	90	100	100	100	100	100	100	100	100
69	70	70	70	71	81	81	90	100	100	100	100	100	100	100	100
69	70	70	70	71	81	81	90	100	100	100	100	100	100	100	100
69	70	70	70	71	81	81	90	175	175	175	175	175	175	175	175
69	70	70	70	71	81	81	90	175	175	175	175	175	175	175	175
69	70	70	70	71	81	81	90	250	250	250	250	250	250	250	250
69	70	70	70	71	81	81	90	250	250	250	250	250	250	250	250
69	70	70	70	71	81	81	90	250	250	250	250	250	250	250	250
69	70	70	70	71	81	81	90	250	250	250	250	250	250	250	250
69	70	70	70	71	81	81	90	250	250	250	250	250	250	250	250

Zoom in example

3.4.2. Reduction (zoom out)

Reduce an image can be done in two different ways:

3.4.2.1.Reduction by decimation

Decimation means that we take a pixel in a set and we loose the other. If we have a NxM image and the reducing factor is n so we just keep N/n pixels horizontally and M/n pixels vertically, all the other pixels are lost. <u>Example:</u> if we reduce the example with a factor n=2 we have:

10	30	50	70	90	100	100	100	100
10	30	50	70	90	100	100	100	100
10	30	50	70	90	250	250	250	250
10	30	50	70	90	250	250	250	250
10	30	50	70	90	250	250	0	250
10	30	50	70	90	250	250	250	250
10	30	50	70	90	250	250	250	250

Reduction by decimation example

3.4.2.2.Mean reduction

We divide the image in $\frac{N}{n} x \frac{M}{n}$ blocks. Each one of the blocks is replaced by a single pixel which have a value equal to the mean value of the pixel set in the block.

Example: if we reduce the same example with the same factor:

18	38	60	71	95	100	100	100	100
18	38	60	71	95	100	100	100	100
18	38	60	71	170	250	250	250	250
18	38	60	71	170	250	250	250	250
18	38	60	71	170	250	250	188	250
18	38	60	71	170	188	250	250	250
18	38	60	71	170	250	250	250	250

Mean reduction example

4. MORPHOLOGICAL OPERATIONS

Morphological operation are neighboorhood operation and spatial invariant.

4.1. Set theory

An object A can be defined in Euclidean spaces such as a set of pixels « a » $a \in \mathbb{Z}^2$. In a binary image we represent the pixels of A as :

$$a_{n,m} \begin{cases} 1 & \text{if } (n,m) \in A \\ 0 & \text{elsewhere} \end{cases}$$

So in digital morphology we define the set A as a ***région*** and its ***complement*** A^C as the set of all pixels not contained in A.

The usual operations from set theory can be applied :

Union : $A \cup B$ all the pixels belong to A or B

Intersection : $A \cap B$ all the pixels belong to A and B

Difference : $A - B$ all the pixels belong to A but not to B ($= A \cap B^C$)

All th eproperties from set theory also apply to our sets :

Commutative :

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative :

$$A \cap (B \cap C) = (A \cap B) \cap C$$

$$A \cup (B \cup C) = (A \cup B) \cup C$$

Distribution :

$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$$

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$$

De Morgan's law :

$$(A \cup B)^C = A^C \cap B^C$$

4.2. Distance measure definition

The Euclidean distance between (a_1,a_2) (b_1,b_2) :

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

In morphological mathematics we very often use the City-Block distance :

$$d_4 = |a_1 - b_1| + |a_2 - b_2|$$

where the pixels with an equal distance from the centre represent a diamond

5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	0	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5

City block d(4)

and the chessboard distance

$$d_8 = \max\left(|a_1 - b_1| : |a_2 - b_2|\right)$$

where the pixels with an equal distance from the centre represent a square.

4	3	2	2	2	2	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	1	0	1	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	2	2	2	2	3	4

Chessboard d(8)

Notation d_4 and d_8 are use to show that the number of pixels equidistant from a pixel is equal to 4 and 8.

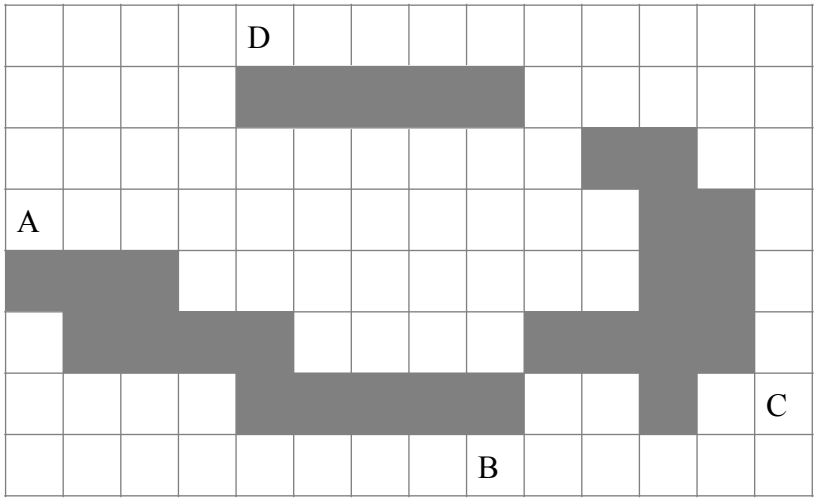
4.3. Other définitions

Two pixels are ***adjacents*** if d_4 or d_8 distance between them is 1.
Adjacent is ***symmetric*** a adj b <=> b adj a
but ***not transitive*** a adj b et b adj c ≠> a adj c

Two pixels adjacents are ***connected.***

A ***figure*** A is ***adjacent*** to B if un ou plusieurs pixels des 2 figures sont adjacents ou connectés. On parlera donc de connecté 4 (ou adjacent 4) et de connecté 8 (ou adjacent 8) suivant que l'on utilise la distance citiblock ou la chessboard.

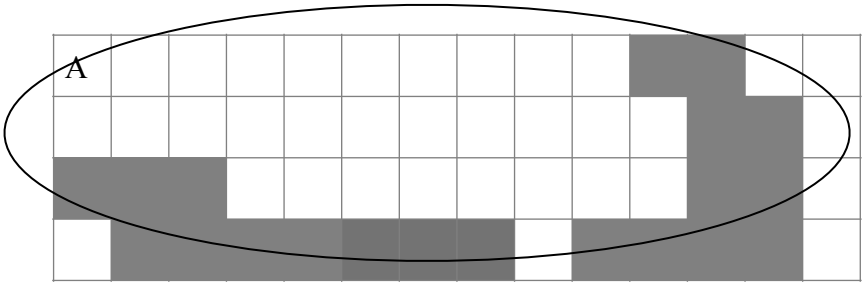
Un ***chemin*** est une série de M pixels tel que chacun est connecté à un autre. Plusieurs chemins peuvent relier deux pixels différents.



Adjacents figures example

A et B sont adjacents-4 et -8 (ou 4 et 8 connected).
B et C sont adjacents-8 (ou 8 connected).
Aucun de A, B et C n'est adjacent avec D.

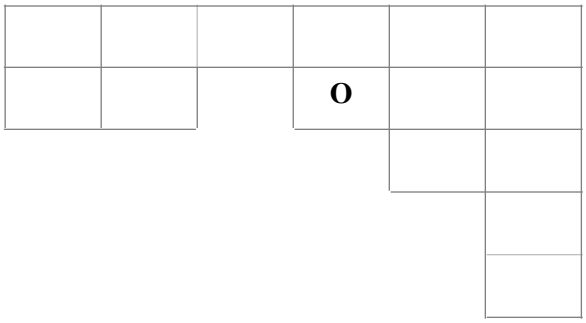
Un « ***composant-connecté*** » d'une région A est un sous-ensemble de A où tous les pixels sont connectés entre eux mais pas connectés à l'autre partie de A.



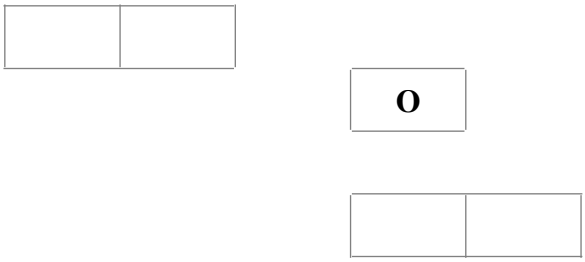
Example de composants-connectés

4.4. Opérations de base

Un élément structurant est un sous-élément de $\mathbb{Z}^2$ noté B qui a a une origine et comporte des pixels qui ne sont pas forcément connectés.



Exemple d'élément structurant

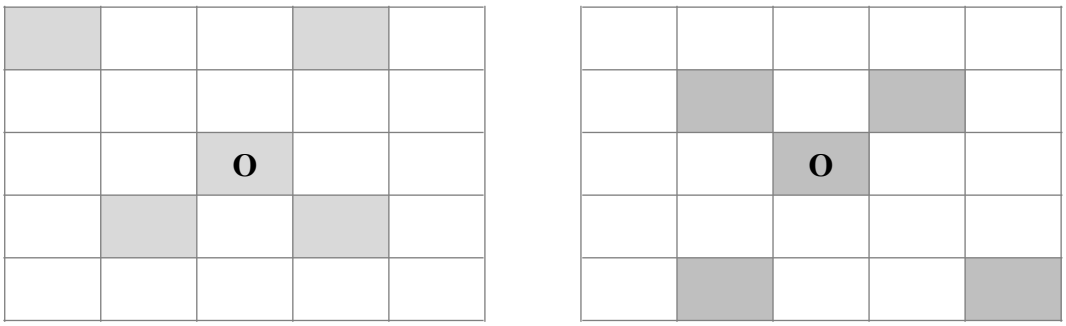


Autre exemple d'élément structurant

Cet élément structurant est « promené » sur toute l'image avec son origine sur le point à traiter.

4.4.1. Translation : $B_t = \{b + t \quad \forall b \in B\}$ où t est un vecteur $\in \mathbb{Z}^2$

4.4.2. Réflection : $\overset{\cup}{B} = \{-B \quad \forall \ b \in B\}$



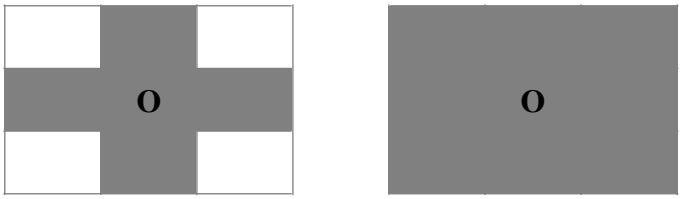
Exemple de réflection

4.4.3. Dilatation

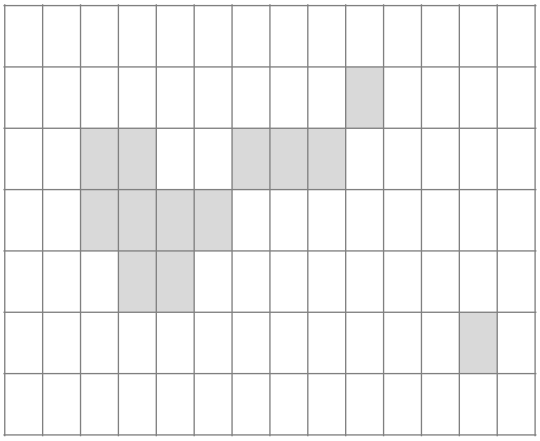
La dilatation d'un ensemble A par l'élément structurant B noté A ⊕ B. C'est en fait la figure correspondant la réponse à la question : Pour quelles positions de B, B touche A ? Mathématiquement la dilatation s'écrit :

$$A \oplus B = \{ t \mid B_t \cap A \neq \varnothing \} \text{ où B est l'élément structurant et } t \in \mathbb{Z}^2$$

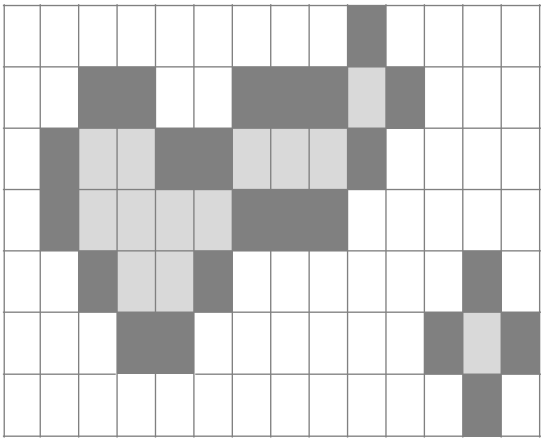
Exemple d'une dilatation avec les éléments structurants B4 et B8



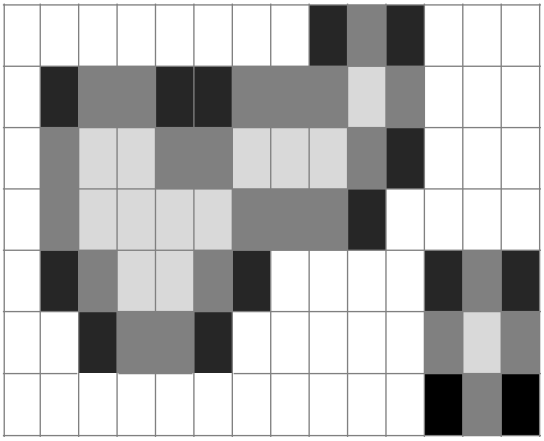
Eléments structurants B4 et B8



Avant Dilatation



Après dilatation B_4



Après dilatation B_8

La dilatation rempli les petits trous, adoucit les contours, connecte parfois plus évidemment des éléments isolés. Dilate d'une façon générale le motif idem pour la 8 N mais encore plus accentué.

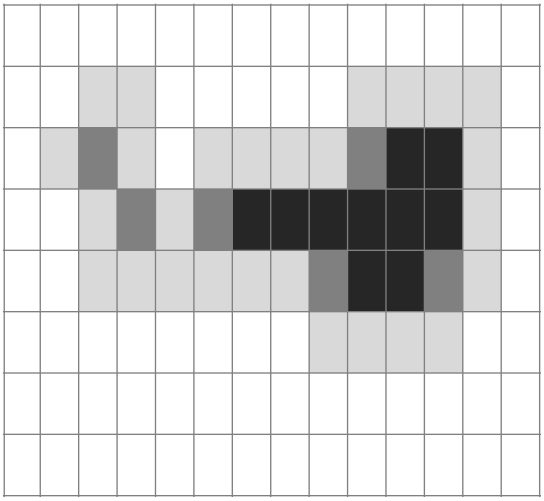
4.4.4. Erosion :

L'érosion d'un ensemble A par l'élément structurant B noté $A \ominus B$.

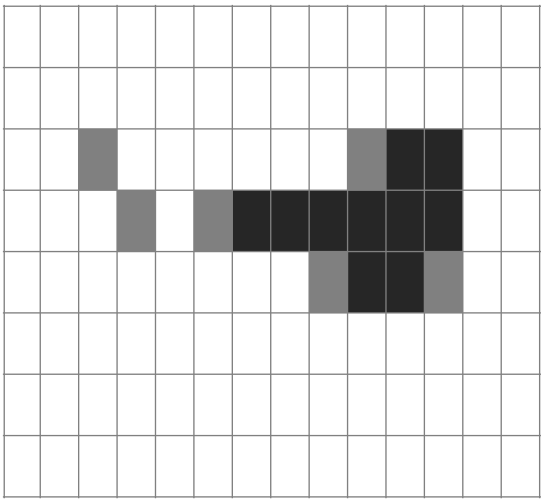
Dans ce cas c'est la figure répondant à la question : Pour quelles positions de B, A contient B ? L'expression mathématique est la suivante :

$$A \ominus B = \{ t \mid B_t \subset A \}$$ où B est l'élément structurant et $t \in \mathbb{Z}^2$

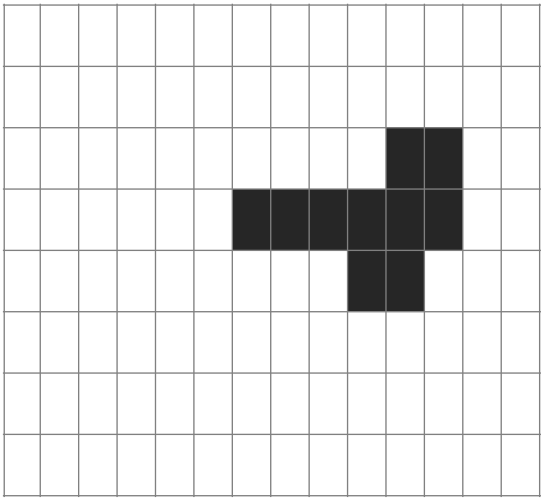
Exemple d'une érosion avec les mêmes éléments structurants B4 et B8



Avant Erosion



Érodé B_4



Érodé B_8

L'érosion produit l'effet inverse de la dilatation : les trous s'élargissent, les éléments adjacents se déconnectent, les motifs sont érodés.

4.4.5. Ouverture

L'érosion est une opération irréversible, en effet deux figures différentes peuvent avoir le même érodé l'opération n'est donc pas biunivoque et est donc irréversible. On peut quand même imaginer faire une opération inverse en faisant un certain choix, par exemple sélectionner le plus petit de ces ensembles. Cette opération est l'***ouverture***, et consiste donc à faire une érosion suivie d'une dilatation. On note donc cette opération :

$$A \circ B = (A \ominus B) \oplus B$$

Exemple :

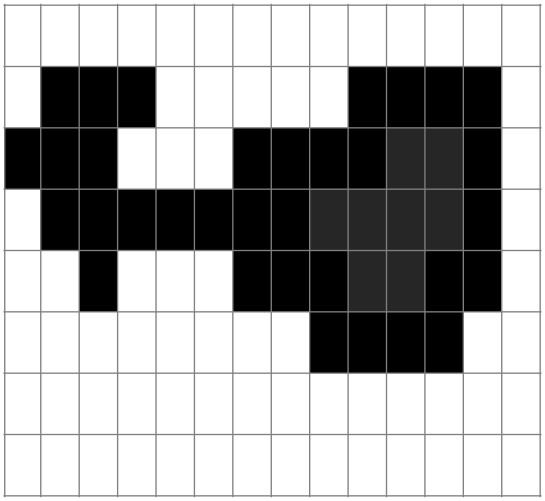
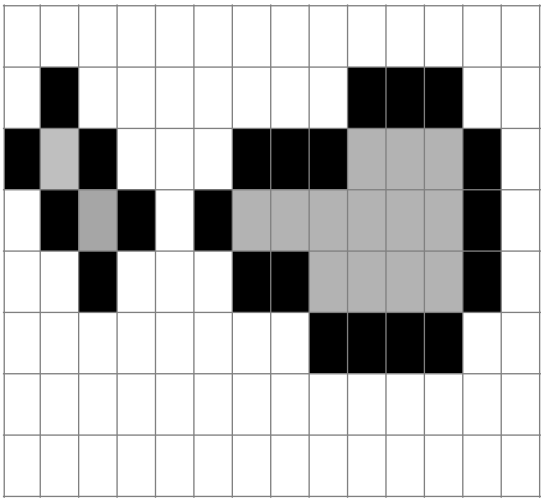


Figure 54 Avant Ouverture



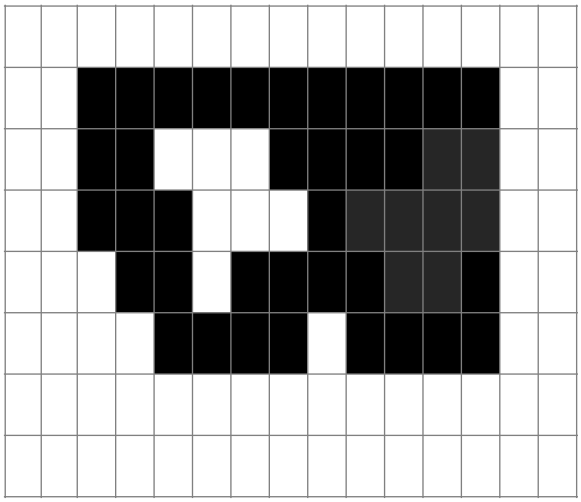
Après Ouverture B4

4.4.6. Fermeture

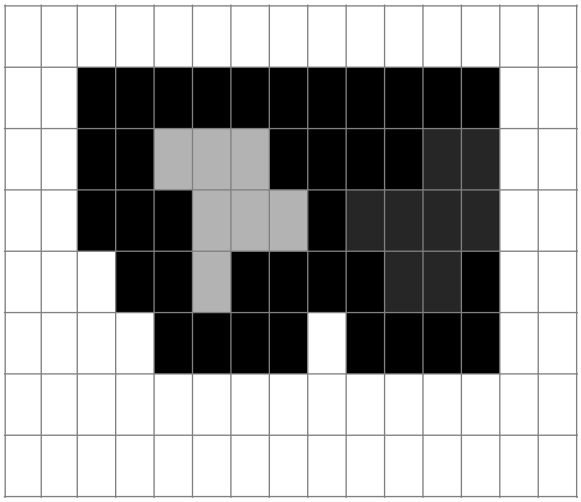
L'opération inverse de l'ouverture consistant à d'abord dilater puis ensuite éroder s'appelle la ***fermeture*** et se note :

$$A \bullet B = (A \oplus B) \ominus B$$

Exemple :



Avant Fermeture



Après Fermeture B4

Propriété : Si on applique deux ouvertures ou deux fermetures l'une à la suite de l'autre en utilisant le même élément structurant, on remarque (et on peut démontrer) que la deuxième opération ne change rien au premier traitement. On dit que l'opération est ***idempotente***.

$$(A \bullet B) \bullet B = A \bullet B \quad \text{et} \quad (A \circ B) \circ B = A \circ B$$

4.4.7. Hit & Miss

Cette opération nécessite 2 éléments structurant :

- un pour toucher « hit » l'objet intéressant (souvent à l'avant-plan) B_{HIT}
- un pour toucher le fond (background) B_{MISS}.

$$A \circledast B = (A \ominus B_{HIT}) \cap (A^C \ominus B_{MISS})$$
$$\text{ou } A \circledast B = (A \ominus B_{HIT}) - (A \oplus B_{MISS})$$

avec $B_{HIT} \cap B_{MISS} = \varnothing$ sinon la résultante serait vide .

L'élément structurant peut donc être représenté par un seul tableau dans lequel B_{HIT} est représenté par des cases blanches, B_{MISS} par des cases noires alors que les cases grises n'appartiennent à aucun des deux.

Remarque : Dans une certaine littérature cette opération est notée HMS(A,B).

4.5. Utilisation des différentes opérations

4.5.1. Suppression du bruit par HMT

Les éléments structurants sont par exemple :

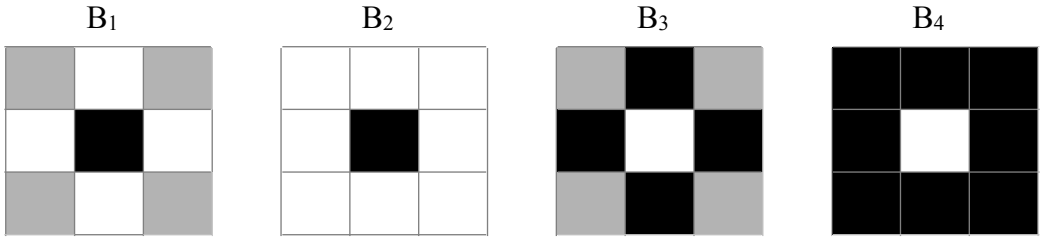


Figure 58 Exemple d'éléments structurants pour élimination de bruits

B_1 et B_2 extraient les pixels isolés de l'arrière plan tandis que B_3 et B_4 extraient les pixels isolés de l'avant plan si les objets intéressants à l'avant plan sont blancs.

Dans ce cas, l'opération de réduction de bruit :

$A \cup$ HMT $(A,B_{1/2})$ arrière plan

$A -$ HMT $(A,B_{3/4})$ avant plan

Si on veut appliquer cette réduction de bruit dans un cas général (avant-plan + arrière-plan) on utilisera alors la formule :

$$C \cup \ HMT(C_1B_{½})$$
$$C = A - HMT\ (A\ ,\ B_{¾})$$

4.5.2. Extraction de contours.

Un point intérieur à 1 ensemble est tel que son voisinage ∈ à l'ensemble. Or par exemple pour la connection 4 cet ensemble est son érosion. Il découle donc que tous les points de A qui ne sont pas intérieurs à A sont en fait à la frontière de A.

$$\Rightarrow \qquad IB(A) = A - (A \ominus B)$$

avec IB pour « Intern Boundary »

On peut définir de la même façon la frontière externe par l'intersection de l'ensemble dilaté avec le complément de l'ensemble.

$$OB\ (A) = A^C \cap (A \oplus B)$$

avec OB pour « Outer Boundary »

4.5.3. Amincissement

Amincir un ensemble par un élément structurant B est défini par :

$$A \otimes B = A - (A \circledast B)$$

Une application principale de l'amincissement est l'obtention du squelette : un squelette de A est le sous-ensemble de A tel que : un pixel du squelette est le centre d'un cercle de rayon maximum qui est contenu dans l'ensemble original ; le squelette préserve l'homotopie de l'ensemble.

Pour garder ces propriétés d'homotopie on utilise un ensemble d'éléments structurants :

$$\{ B \} : B_1, B_2, \ldots \ B_n$$
$$\text{et} \qquad A \otimes \{B\} = ((\ldots(A \otimes B_1) \otimes B_2) \ldots B_n$$

4.5.4. Epaississement

$$A \odot B = A \cup (A \circledast B)$$
$$\Rightarrow \quad A \odot B = ((\ldots(A \odot B_1) \odot B_2) \ldots B_n)$$

Eventuellement, on aurait pu amincir A^C et faire l'inversion du résultat (mais des points isolés apparaîssent parfois).

4.5.5. Squelette

$$S(A) = \bigcup_{K=0}^{K} S_K(A)$$

Où

$$S_K(A) = (A \Theta K_B) - [(A \Theta K_B) \circ B]$$
$$K = \max\{K / (A \Theta KB) \neq \Phi\}$$

On répète l'amincissement autant de fois que nécessaire c'est-à-dire que l'opération devient « indempotence ».

$$A^{i+1} = A^i \otimes \{B\}$$

4.5.6. Ebarbulage (purning)

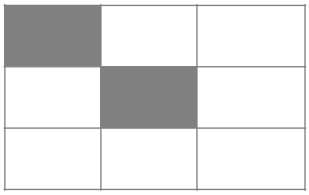
Cette transformation est destinée à éliminer les composantes parasites d'un squelette

1ère étape :

$X_1 = A \otimes \{B\}$

à faire i fois

où B :



+ rot. 90°

X		
X		

+ rot. 90°

si on veut enlever les branches d'au moins i pixels

2ème étape :

restauration conditionnelle (facultative) à faire i fois

$$X_2 = \bigcup_{K=1}^{8} (X_1 \otimes B^K)$$

$$X_3 = (X_2 \oplus H) \cap A$$

où H :

1	1	1
1	1	1
1	1	1

3ème étape : le résultat $X_4 = X_1 \cup X_3$

5. Correlation

Correlation is a technique that can be used to detect and/or localise a known shape in the image. Suppose that the appearance of an object (or part of an object) within the image is exactly known. Furthermore, suppose that the image of this « object of interest » is given by $h(x-\xi,y-\eta)$. The parameters (ξ,η) define the position of the object in the image plane. Then the observed image $f(x,y)$ can be thought of as a composition of $h(x,y)$ and the background $n(x,y)$ of the scene:

$$f(x,y) = h(x-\xi, y-\eta) + n(x,y) \qquad \textbf{(6.20)}$$

It is understood that the background contains a deterministic part and a random part. Of course, this model holds true only if the scene really does contain the object of interest. If it is not the case, then the observed image is written as:

$$f(x,y) = n(x,y) \qquad \textbf{(6.21)}$$

The vision task is to check if the scene contains the object, and if so, to estimate its position within the image plane. One way to accomplish this task is to shift a template of the object across the image plane, to match this template locally with the image data, and to select the position (ξ_1,η_1) of the template for which the match is best. A particular choice of the measure of match is the Euclidean distance between the template and the observed image.

$$d^2(\xi,\eta) = \int_x \int_y (f(x,y) - h(x-\xi, y-\eta))^2 \qquad \textbf{(6.22)}$$

$$\text{with } d(\xi_1,\eta_1) = \min\{ d(\xi,\eta) \} \qquad \textbf{(6.23)}$$

Note that, upon substitution of integration variables, the equation can be written as:

$$d^2(\xi,\eta) = \int_x \int_y (f(x+\xi, y+\eta) - h(x,y))^2 \qquad \textbf{(6.24)}$$

In the discrete case, finite summations replace the integrals. Hence:

$$d^2_{n,m} = \sum_{i=-K}^{K} \sum_{j=-L}^{L} (f_{n+i,m+j} - h_{i,j})^2 \qquad \textbf{(6.25)}$$

The ranges of summations are determined by the constants K and L. These constants must be chosen in such a way that the template $h_{K,L}$ fully embeds the image of the object of interest.

250	250	250
250	5	250
250	250	250

Template example

If the Table 79 represents the template, the correlation between this template and our example can be computed in the Table 80.

ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns
ns	648	613	592	554	536	513	492	471	445	435	435	435	435	435	435	435	ns
ns	648	613	592	554	536	513	492	471	445	435	435	435	435	435	435	435	ns
ns	648	613	592	554	536	513	492	447	392	349	349	349	349	349	349	349	ns
ns	648	613	592	554	536	513	492	421	426	357	357	357	357	357	357	357	ns
ns	648	613	592	554	536	513	492	393	370	245	245	245	245	245	245	245	ns
ns	648	613	592	554	536	513	492	393	370	245	245	245	245	245	245	245	ns
ns	648	613	592	554	536	513	492	393	370	245	245	245	350	350	350	245	ns
ns	648	613	592	554	536	513	492	393	370	245	245	245	350	5	350	245	ns
ns	648	613	592	554	536	513	492	393	370	350	350	350	350	350	350	245	ns
ns	648	613	592	554	536	513	492	393	370	350	5	350	245	245	245	245	ns
ns	648	613	592	554	536	513	492	393	370	350	350	350	245	245	245	245	ns
ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns

Correlation example

It can be interesting to expand the expression

(6.26)

in the following equation :

$$d^2_{n,m} = \sum_{i=-K}^{K} \sum_{j=-L}^{L} h^2_{i,j} - 2 \sum_{i=-K}^{K} \sum_{j=-L}^{L} f_{n+i,m+i} h_{i,j} + \sum_{i=-K}^{K} \sum_{j=-L}^{L} f^2_{n+i,m+j} \qquad \textbf{(6.27)}$$

The first term on the right hand side does not depend on the observed image. It is the *norm* (sometimes called *energy*) of the template. Since the term does not depend on (n,m) it can be omitted in further calculations without affecting the quality of the detection and the localisation.

The last term solely depends on the observed image. This term represents the local energy of the image calculated in a (2K+1)x(2L+1) neighbourhood surrounding each pixel (n,m). This term depends of course on (n,m), but in some applications it can also be omitted. If so, the second term, called *cross* correlation remains. In fact, correlation is then equivalent to convolution.

ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns
ns	4	7	8	11	12	14	16	17	19	20	20	20	20	20	20	20	ns
ns	4	7	8	11	12	14	16	17	19	20	20	20	20	20	20	20	ns
ns	4	7	8	11	12	14	16	21	27	31	31	31	31	31	31	31	ns
ns	4	7	8	11	12	14	16	25	31	39	39	39	39	39	39	39	ns
ns	4	7	8	11	12	14	16	29	38	50	50	50	50	50	50	50	ns
ns	4	7	8	11	12	14	16	29	38	50	50	50	50	50	50	50	ns
ns	4	7	8	11	12	14	16	29	38	50	50	50	44	44	44	50	ns
ns	4	7	8	11	12	14	16	29	38	50	50	50	44	50	44	50	ns
ns	4	7	8	11	12	14	16	29	38	44	44	44	44	44	44	50	ns
ns	4	7	8	11	12	14	16	29	38	44	50	44	50	50	50	50	ns
ns	4	7	8	11	12	14	16	29	38	44	44	44	50	50	50	50	ns
ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns	ns

Correlation computed with convolution

In this example the performance of the detection/localisation is bad. It is because the local energy is not constant everywhere on the image: the image is not stationary and then the third term on the equation can not be omitted in that case.

Another situation in which template matching often fails is when the object of interest is not perfectly known. Many factors may contribute to this lack of knowledge: unknown factors in the illumination, the imaging geometry, the shape, the size and orientation of the object, etc. In many occasions this hinders the utilisation of correlation techniques. However, the influence of some unknown factors can be neutralised quite easily. For instance, if a constant background level « b » of the image is unknown:

$$f(x,y) = h(x-\xi, y-\eta) + n(x,y) + b \qquad \textbf{(6.28)}$$

The influence of b can be neutralised by normalisation of the local average grey level within the template and the image. For this purpose define:

$$\bar{h} = \frac{1}{(2K+1)(2L+1)} \sum_{i=-K}^{K} \sum_{j=-L}^{L} h_{i,j} \qquad \textbf{(6.29)}$$

$$\overline{f_{n,m}} = \frac{1}{(2K+1)(2L+1)} \sum_{i=-K}^{K} \sum_{j=-L}^{L} f_{n+i,m+j} \qquad \textbf{(6.30)}$$

A measure of match can be formulated between the fluctuating part in $h_{n,m}$ and $f_{n,m}$. That is, between $h_{n,m}$-~~h~~ and $f_{n,m}$-~~$f_{n,m}$~~. For instance, if stationarity is assumed it suffices to calculate the cross correlation normalised with respect to background level:

$$\sum_{i=-K}^{K} \sum_{j=-L}^{L} (f_{n+I,m+j} - \overline{f_{n,m}})(h_{i,j} - \bar{h}) \qquad \textbf{(6.31)}$$

Another nuisance parameter that sometimes hampers the detection and/or localisation is the gain *a* :

$$f(x,y) = a\,h(x-\xi, y-\eta) + n(x,y) \qquad \textbf{(6.32)}$$

The parameter *a* can be eliminated by normalisation of the local energy. Define:

$$\overline{h^2} = \frac{1}{(2K+1)(2L+1)} \sum_{i=-K}^{K} \sum_{j=-L}^{L} h_{i,j}^2 \qquad \textbf{(6.33)}$$

$$\overline{f_{n,m}^2} = \frac{1}{(2K+1)(2L+1)} \sum_{i=-K}^{K} \sum_{j=-L}^{L} f_{n+i,m+j}^2 \qquad \textbf{(6.34)}$$

Invariance with respect to the parameter *a* is obtained by division of $f_{n,m}$ by $(f^2_{n,m})1/2$. Note that due to this normalisation the first and third term in equation essantially become constants, so that only the second term is important. The result is the following measure of match:

$$r_{n,m} = \frac{\sum_{i=-K}^{K} \sum_{j=-L}^{L} f_{n+i,m+j} h_{i,j}}{\sqrt{\overline{f_{n,m}^2}} . \sqrt{\overline{h^2}}} \qquad \textbf{(6.35)}$$

If gain and background are unknown, then normalisation with respect to both parameters must be achieved:

$$r_{n,m} = \frac{\sum_{i=-K}^{K} \sum_{j=-L}^{L} (f_{n+i,m+j} - \overline{f_{n,m}})(h_{i,j} - \overline{h})}{\sqrt{\overline{(f_{n,m} - \overline{f_{n,m}})^2}} \sqrt{\overline{(h - \overline{h})^2}}} \qquad \textbf{(6.36)}$$

This last measure is called *cross correlation coefficient* because of its analogy with the normalised correlation coefficient defined between two random variables.

Chapitre X

Cas d’utilisation

Détection d’intrusion

Enoncé

Ce cas d’utilisation propose de détecter une intrusion dans le champs d’une caméra. On peut imaginer une caméra de surveillance qui « surveille » une scène et qui détecte automatiquement un changement brusque dans le flux vidéo. L’alarme doit donc se déclencher si une personne arrive dans la scène mais ne doit pas provoquer d’alerte si la luminosité globale de l’image change (par exemple, à cause d’un nuage qui passe devant le soleil).

Solution

Comptage des oeufs dans un nid de poule

Enoncé

Solution

https://fr.eggs-iting.com/blog/prototypage/detection-des-oeufs-dans-un-nid-de-poule.html