

IFOSUP

Institut de Formation Supérieure
Ville de Wavre

Mathématiques

appliquées à l'informatique

G. Barmarin

2023-2024

$$Y = mX + C$$



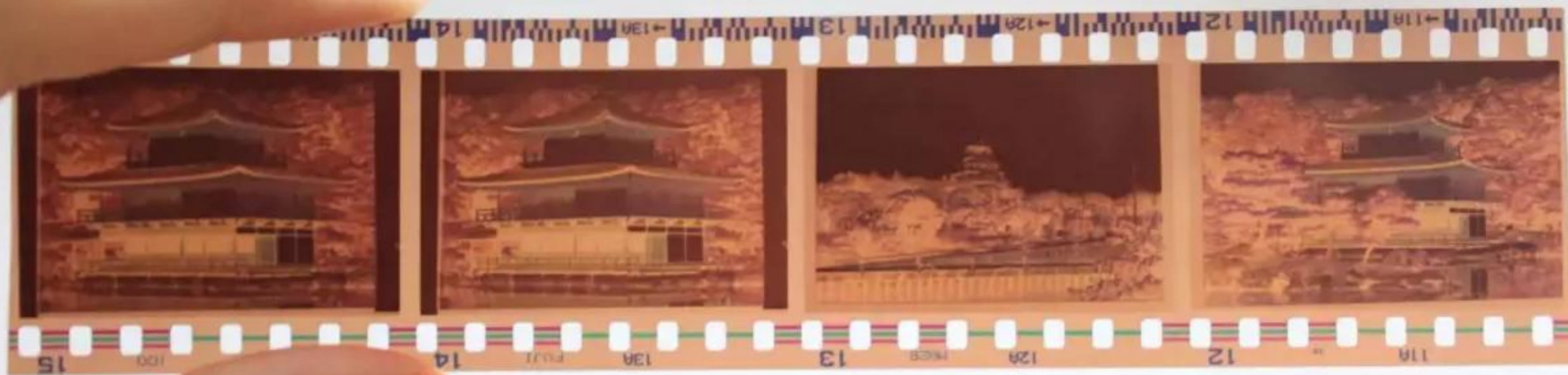
phi

$$\frac{X = X_1}{X_2 = X_1}$$



$$V = 543.25$$





Traitement d'images par calcul matriciel

G. Barmarin

2024-2025

2 types d'images

On rencontre couramment deux types d'images :

- les images matricielles (ou bitmap, ou raster),
- et les images vectorielles.

Les images matricielles consistent en un tableau à deux (ou plus) dimensions, dont chaque case contient la valeur d'un « pixel ». Le terme pixel est la contraction de picture element.

Les images fournies par les systèmes d'acquisition (microscope, appareil photo, scanner à plat...) sont de ce type. Elles sont stockées typiquement dans des fichiers aux formats .tif, .bmp, .png, .jpg, .gif...

Les images vectorielles représentent une image sous la forme d'une série de primitives géométriques : segment, point, cercle, polygone... Leur gros avantage est que contrairement aux images matricielles, on peut agrandir une image vectorielle autant que l'on veut sans perte de qualité.

Elles sont souvent utilisées pour sauvegarder le résultat de graphiques ou de dessins techniques. Elles sont sauvées dans des formats postscript (ps, eps), .fig, .svg...

Dans le cadre du traitement d'images, on travaille exclusivement sur des images matricielles.

De manière un peu plus formelle, on peut voir une image comme une application (au sens mathématique du terme) qui, à une position donnée p , fait correspondre une valeur v . Pour des images planes, la position est un point du plan, repéré par deux coordonnées x et y . La valeur fournie est en général une intensité (lumineuse pour une caméra, de rayonnement pour un tomographe, de température pour une caméra thermique...), ou une information plus complexe (couleur, spectre...). La valeur d'un pixel peut être représentée par un vecteur de q valeurs, avec $q = 1$ pour une image d'intensité ou $q = 3$ pour une image couleur basée sur le système additif des couleurs (RGB).

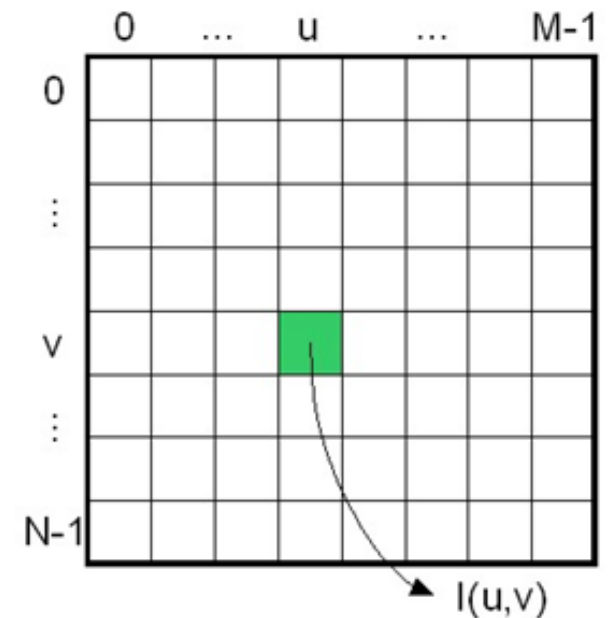
Une image idéalisée peut donc être vue comme une fonction de la forme :

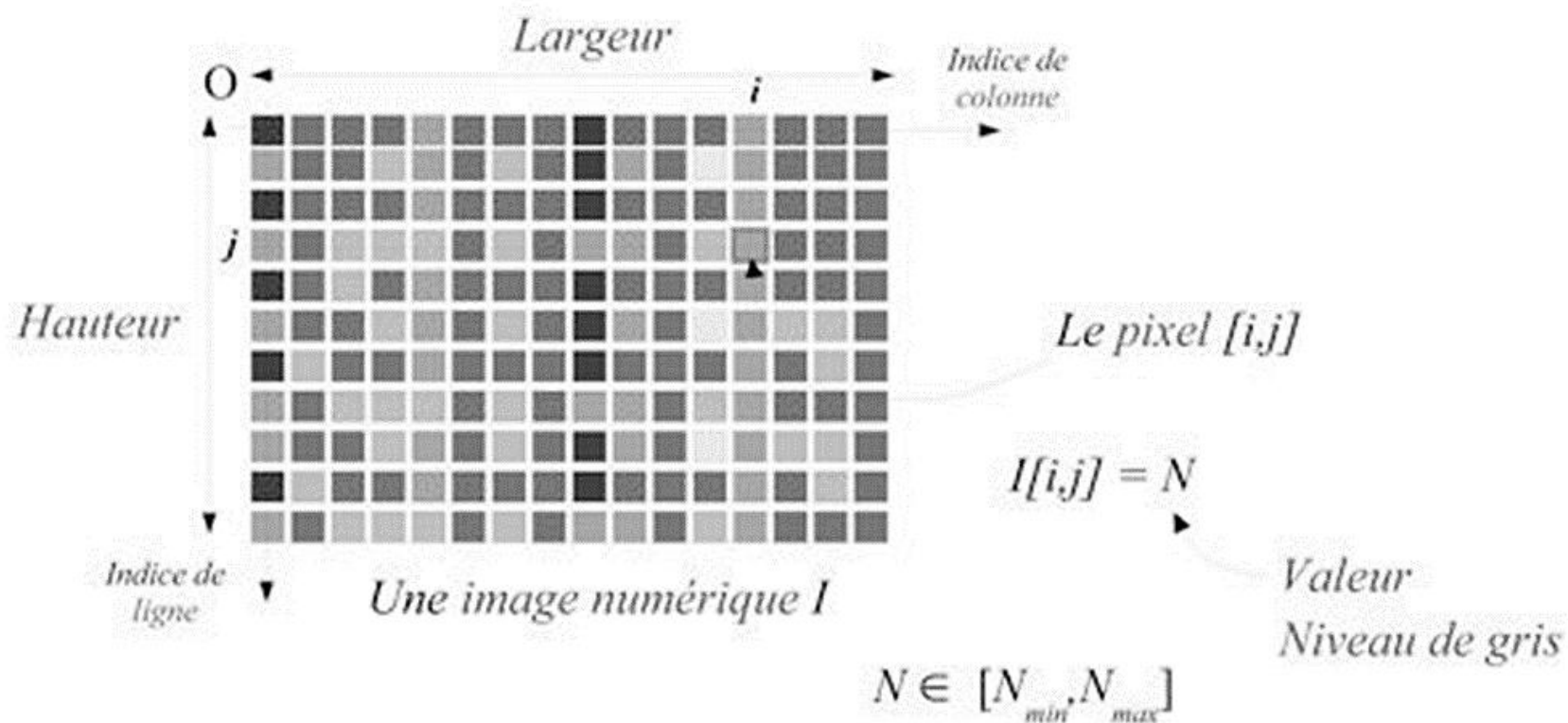
$$F : \begin{cases} \mathbb{R}^n & \rightarrow \mathbb{R}^q \\ p & \rightarrow v = F(p) \end{cases}$$

Dans les images matricielles 2D, les valeurs sont définies uniquement pour chaque élément de la matrice image. De manière traditionnelle, l'origine de l'image se situe en haut à gauche, en comptant les colonnes vers la droite et les lignes vers le bas, tous les deux en commençant à 0. Les valeurs disponibles sont donc de la forme :

$$F(x, y) = I(u, v), u \in [0; M - 1], v \in [0; N - 1]$$

où u et v sont des valeurs entières représentant les indices des colonnes et des lignes, et M et N représentent la largeur et la hauteur de l'image





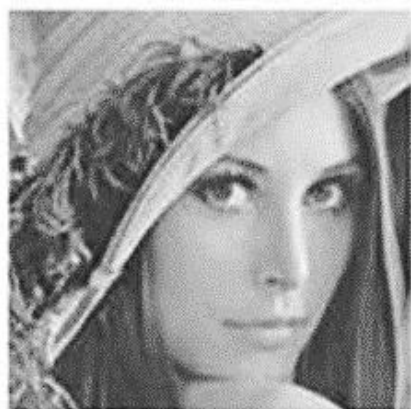
$$(N_{max} - N_{min}) = \text{nombre de niveaux de gris}$$

$$\text{Log}_2(N_{max} - N_{min}) = \text{dynamique}$$

Résolution...

...spatiale :

Échantillonnage



256x256



128x128



64x64



32x32

...tonale :

Quantification



6 bits



4 bits



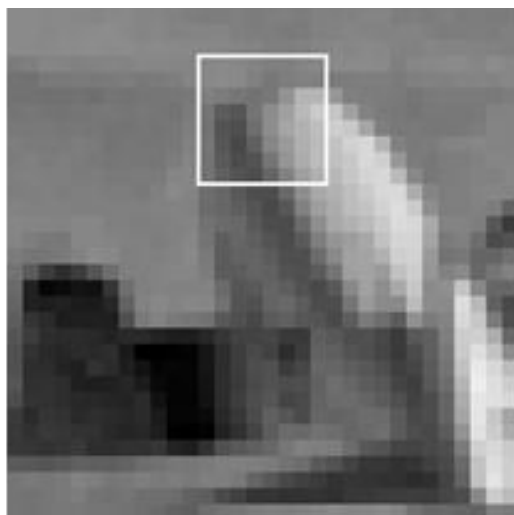
3 bits



2 bits

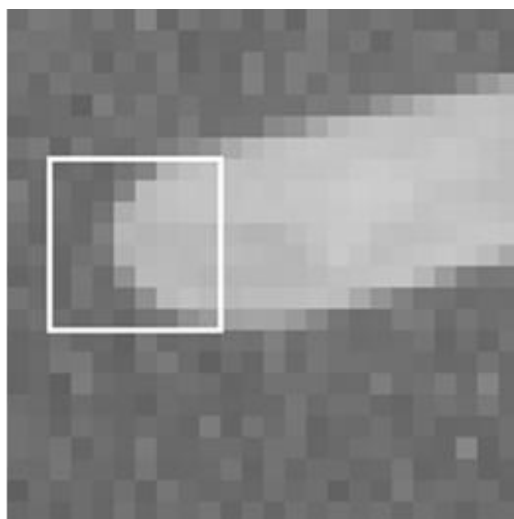
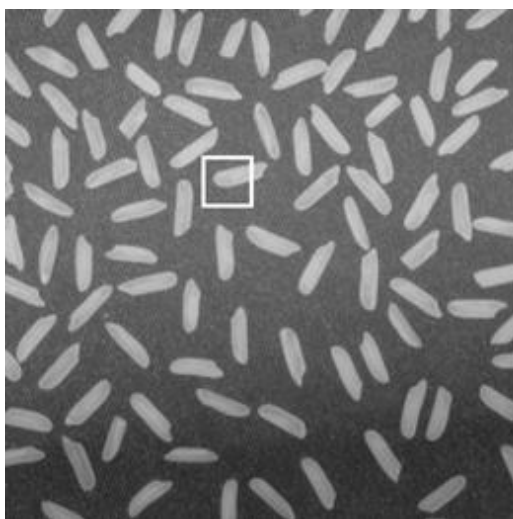


1 bit



| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 137 | 139 | 137 | 128 | 119 | 126 | 130 | 129 |
| 129 | 124 | 118 | 115 | 120 | 147 | 181 | 182 |
| 117 | 93 | 87 | 117 | 141 | 160 | 185 | 203 |
| 115 | 84 | 79 | 111 | 142 | 166 | 178 | 191 |
| 119 | 87 | 73 | 97 | 135 | 155 | 176 | 187 |
| 116 | 91 | 77 | 84 | 118 | 150 | 173 | 188 |
| 117 | 97 | 84 | 78 | 101 | 131 | 160 | 177 |
| 111 | 104 | 90 | 78 | 78 | 105 | 142 | 170 |

2 exemples d'images en niveau de gris avec agrandissement pour voir les pixels



| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 107 | 112 | 104 | 118 | 132 | 179 | 190 | 195 |
| 114 | 105 | 107 | 132 | 186 | 193 | 196 | 195 |
| 112 | 106 | 117 | 174 | 186 | 192 | 194 | 195 |
| 108 | 107 | 128 | 183 | 180 | 183 | 185 | 184 |
| 104 | 114 | 120 | 176 | 183 | 185 | 184 | 186 |
| 103 | 120 | 108 | 146 | 186 | 186 | 184 | 181 |
| 103 | 119 | 109 | 113 | 144 | 185 | 192 | 186 |
| 112 | 106 | 111 | 104 | 116 | 121 | 145 | 156 |

La résolution d'une image spécifie la distance en unités métriques entre deux pixels.

Pour les images destinées à l'impression, elle est souvent exprimée en points par pouce (dots per inch, ou DPI).

En général, les pixels sont carrés et la résolution est la même pour les directions horizontales et verticales.

Les informations que l'on cherche à extraire des images étant souvent des paramètres de taille ou de distance entre objets, il est donc important de disposer de la résolution pour l'analyse de l'image.

Certains appareils d'acquisition enregistrent la résolution dans les métadonnées du fichier image.

Si aucune information de résolution n'est disponible (par exemple pour les caméras), il est nécessaire d'utiliser des mires ou des points de repère de dimension connue.

Images en niveau de gris

Les images les plus simples contiennent une valeur d'intensité, codée sur un nombre fini de niveaux de gris. Le plus souvent, le nombre de niveaux de gris est codé sur 256 valeurs correspondant à un nombre binaire de 8 bits ($2^8=256$), le noir correspondant à la valeur 0 et le blanc à la valeur 255. Cette étendue de 8 bits est désignée sous le terme de dynamique.

Images binaires: noir ou blanc

Les images binaires sont des images en niveau de gris particulières, dont les valeurs valent soit 0 (noir), soit 1 (blanc). On cherche souvent à ramener les images de départ à des images binaires, car il est très facile ensuite d'identifier les objets présents dans la scène

Images en couleur

Les images en vraies couleurs stockent pour chaque pixel 3 composantes : rouge, verte et bleue (ou RGB, pour Red-Green-Blue). Ces trois composantes permettent de représenter toutes les autres couleurs, y compris les nuances de gris. Si chaque composante est codée sur 8 bits, on peut représenter jusqu'à $256 \times 256 \times 256 = 16$ millions de couleurs différentes. On parle aussi d'images couleurs 24 bits. Cet espace couleur est basé sur la synthèse additive des couleurs, c'est à dire que c'est le mélange des trois composantes (R, V, B) qui donne une couleur particulière donnée.

Pour le stockage, on rajoute parfois 8 bits supplémentaires. Cet ajout permet de coder chaque pixel sur un entier de 32 bits, qui est l'unité de calcul de base de nombreux ordinateurs, et d'extraire les composantes par des opérations opérant au niveau du bit. On utilise parfois des espaces de couleurs différents, qui sont plus adaptés que les composantes RVB. L'espace HSV décompose chaque pixel en trois composantes de teinte (hue, qui représente la couleur principale), de saturation (dosage entre «coloré» et «gris»), et de valeur (qui correspond à l'intensité globale du pixel).

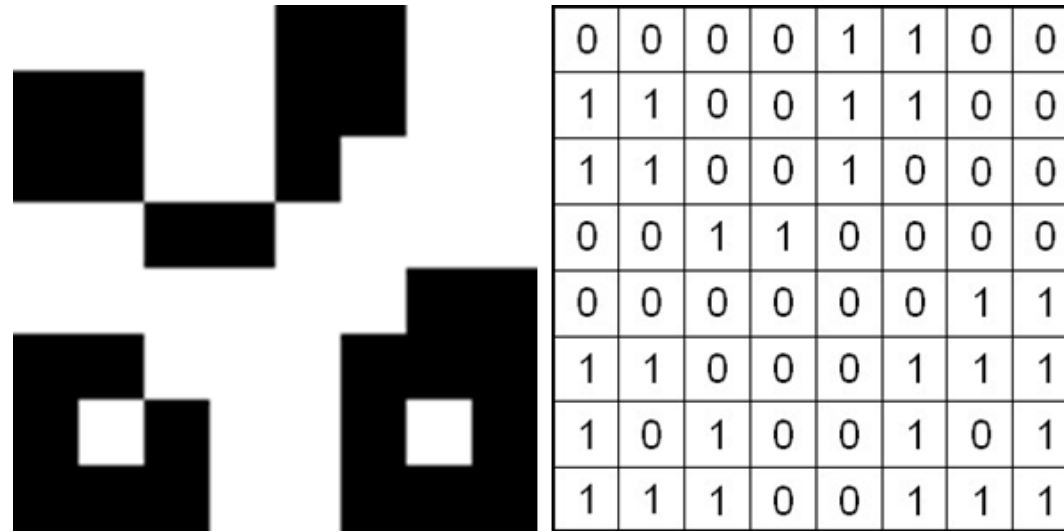


Image binaire 8x8 et la matrice correspondante

Les images binaires ou en niveaux de gris sont généralement stockées avec la convention que 0 (la plus petite valeur, intensité lumineuse nulle) est représentée par le noir et que la plus grande valeur possible (1 ou 255, intensité lumineuse maximale) est représentée par le blanc.

Pour les documents imprimés, ou pour la présentation de certains résultats, on préfère souvent inverser la convention 1 ou 255 = noir et 0 = blanc (voir schéma ci-dessus). Cela a plusieurs avantages : les structures fines sont plus visibles quand elles sont affichées en noir sur fond blanc, le résultat est (souvent) plus « esthétique », et enfin cela évite d'utiliser beaucoup d'encre...

Il convient donc d'être particulièrement attentif à la convention utilisée lorsque l'on travaille sur des images binaires ou d'intensité de gris.

Images couleurs indexées

Les images couleurs indexées stockent pour chaque pixel un numéro de couleur (son index), qui fait référence à une couleur stockée séparément dans une palette.

L'intérêt de ces images est de réduire l'espace de stockage nécessaire, au prix d'une perte de qualité par rapport aux images en vraies couleurs.

Par exemple, une image en 256 couleurs indexées occupera sensiblement la même place qu'une image en 256 niveaux de gris (la place occupée par la palette est généralement négligeable par rapport à la taille de l'image).

Formats d'image

La sauvegarde des images sous forme de fichier nécessite de stocker non seulement la liste des valeurs des pixels, mais aussi la taille de l'image et la manière dont sont codés les pixels.

Les dimensions des images étant souvent grandes, on peut être tenté de les compresser. Cependant, les algorithmes de compression très efficaces induisent souvent une perte de données parfois incompatible avec leur utilisation scientifique.

Il vaut donc mieux stocker les images à travailler sous une forme non compressée, comme le format TIFF (ou RAW).

Structure des fichiers

Un fichier image est en général composé de deux parties : un en-tête, et des données.

L'exception concerne les fichiers RAW, qui ne contiennent que les données brutes (avec les EXIFs), et qui ne peuvent pas être lus sans connaître les conditions dans lesquelles a été acquise l'image.

L'en-tête contient la taille de l'image, le type de donnée (binaire, couleur, niveau de gris...), le nombre de bits utilisés pour le codage, l'ordre dans lequel sont stockés les valeurs, le type de compression utilisé...

On peut parfois y trouver des informations utilisateurs (meta-données), telles que les données EXIF (EXchangeable Image file Format).

Suivent ensuite les données, éventuellement compressées, qui ne peuvent être lues qu'une fois l'en-tête décodé.

Compression des images

La compression permet de transformer un ensemble de données de départ, correspondant aux valeurs des pixels, en un autre ensemble de données, **de taille plus réduite**, mais qui contient au maximum l'information de l'ensemble de départ. La décompression est l'opération de reconstruction des valeurs de départ à partir des données compressées.

On distingue deux modes de compression : avec perte (ou destructif) et sans perte (ou non destructif).

Dans un mode de compression sans perte, l'image obtenue après décompression correspond exactement à l'image initiale. Plusieurs algorithmes existent, tels que le RLE (Run-Length-Encoding), qui détecte les plages de pixels de même valeur, ou le LZW (Lempel-Ziv-Welch), utilisé par les images GIF. En pratique, ces types de compression sont plutôt efficaces pour des images avec beaucoup de redondances (beaucoup de zones avec des couleurs uniforme). Les images scientifiques, au même titre que les photographies de famille, contiennent très peu de redondances, et les algorithmes de compression sans perte sont donc assez peu efficaces.

RLE: voir https://fr.wikipedia.org/wiki/Codage_par_plages

LZW: voir <https://fr.wikipedia.org/wiki/Lempel-Ziv-Welch> ou <https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch>

Dans un mode de compression avec perte, on accepte que l'image obtenue après décompression présente quelques différences avec l'image initiale. Les algorithmes utilisent pour cela une décomposition de l'image selon les composantes fréquentielles (compression JPEG) ou basée sur des ondelettes (JPEG2000), et ne conservent que les composantes principales de la décomposition. Il en résulte une détérioration de l'image reconstruite.

Quelques formats de fichiers d'image

Il existe une grande quantité de formats de fichier d'image, mais seuls quelques-uns sont intéressants en pratique.

Tagged Image File Format (TIFF)

C'est le format de prédilection pour sauvegarder des images scientifiques. Il permet de stocker des images en niveaux de gris, en couleur, binaire, compressée (sans perte) ou non, ainsi que des données utilisateurs (les tags). En général, les appareils scientifiques sauvent également les conditions d'acquisition.

Bitmap (BMP) ce format est très répandu sous Windows, moins sous Linux. Il permet de compresser sans pertes des images en couleurs ou en niveaux de gris.

Portable Network Graphics (PNG) il s'agit d'un format plus standard que le bitmap, qui autorise plus de possibilité de compression, mais qui reste plus adapté pour les images de type dessin.

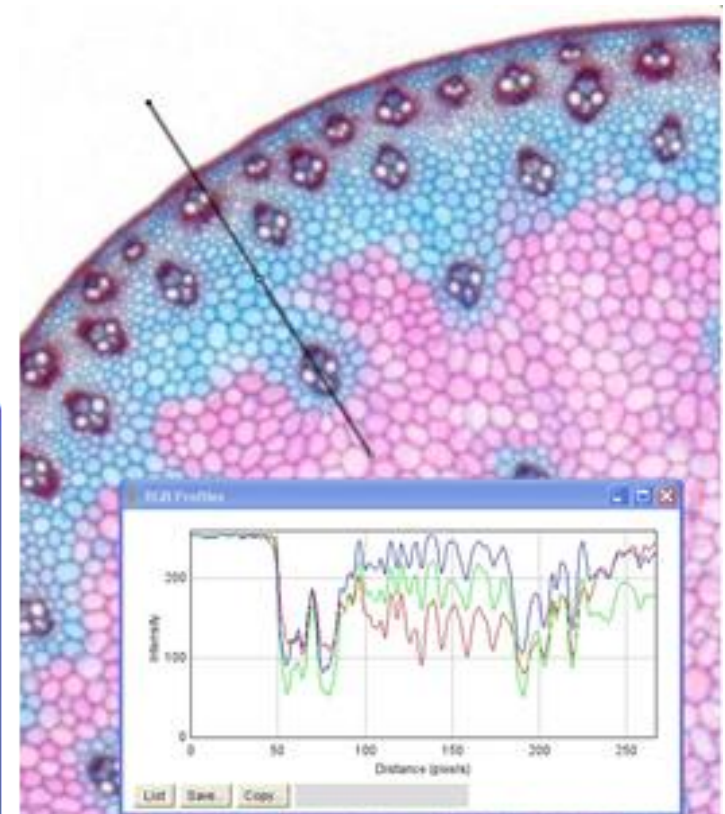
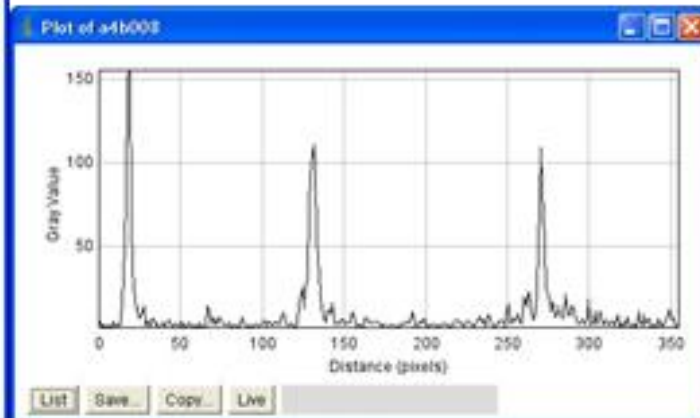
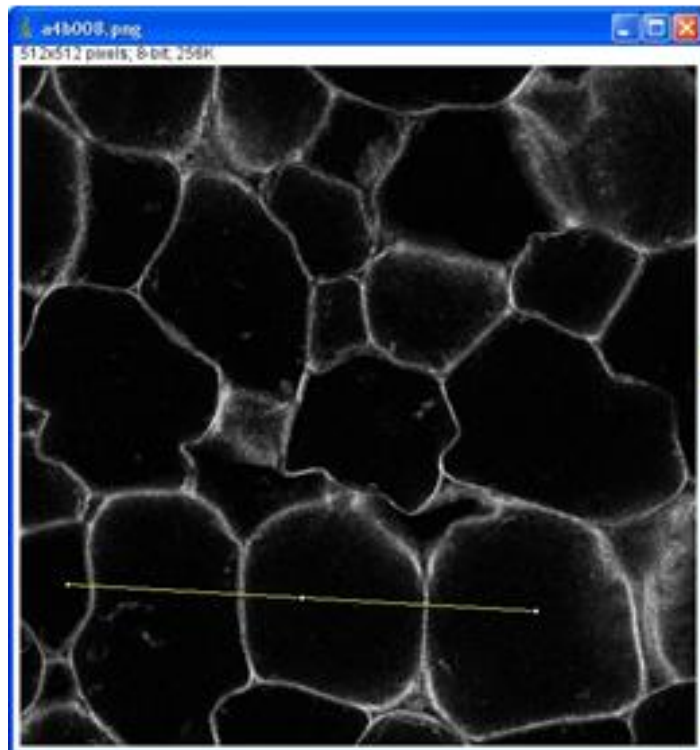
Joint Photographic Experts Group (JPG ou JPEG) ce format est beaucoup utilisé pour stocker les photos, mais il compresse les images avec pertes. Il est à proscrire pour l'analyse d'images scientifiques.

Portable bit map (PBM), portable gray map (PGM) et portable pixel map (PPM) : ces 3 trois formats stockent les valeurs des pixels dans un fichier texte. L'avantage est qu'un être humain peut lire directement le fichier, mais la taille de ces fichiers est beaucoup plus importante que pour les autres formats.

Profil d'intensité

Un outil interactif très intuitif pour visualiser les variations d'intensité dans une image est le profil des intensités.

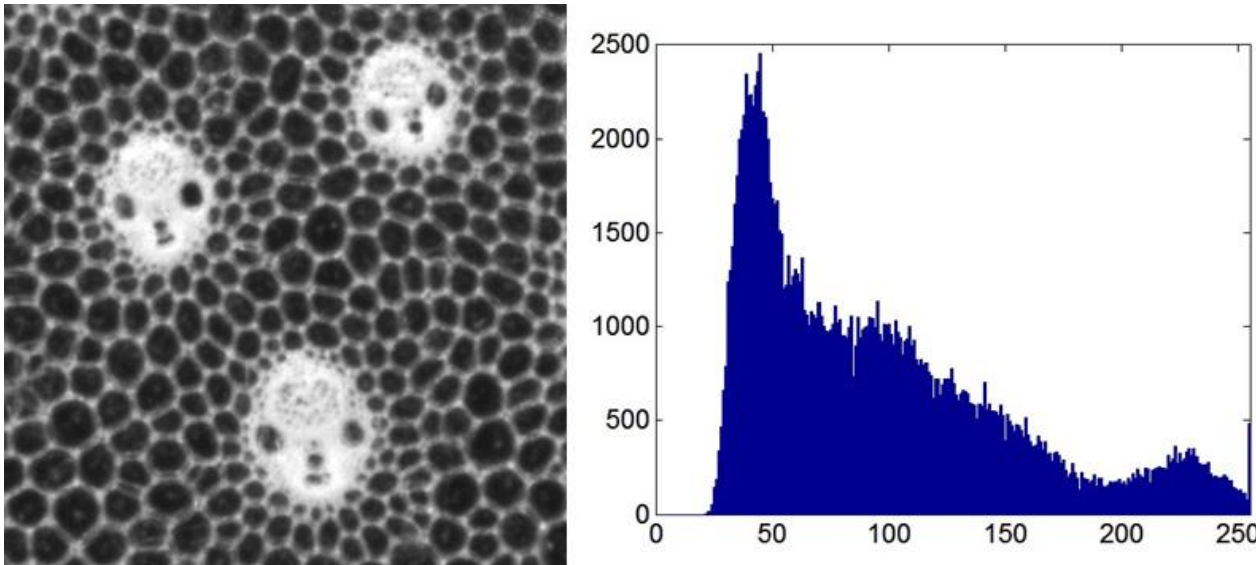
Il faut tout d'abord choisir un outil de sélection linéaire (segment, droite polygonale...), et tracer la région d'intérêt désirée. Le profil d'intensité permet de visualiser les variations d'intensité le long de la sélection.



Histogramme

L'histogramme d'une image est une application qui à chaque valeur de niveau de gris dans l'image associe le nombre de pixels ayant cette valeur. Pour une image en 256 niveaux de gris, le résultat de l'histogramme sera une liste de 256 valeurs. La figure ci-dessous représente une image en niveaux de gris de coupe de tige de maïs, et l'histogramme qui lui correspond.

Le pic le plus à gauche de l'histogramme correspond aux pixels du fond (de couleur noire). Les valeurs de pixels 200 à 250 correspondent aux structures claires, tandis que les valeurs centrales (autour de 50 à 100) représentent les parois des cellules.



L'histogramme est un outil rapide permettant de vérifier la qualité de l'image :

- image globalement surexposée (trop claire) ou sous-exposée (trop sombre)
- histogramme idéalement centré, et le plus étalé possible
- présence ou absence de zones ayant un niveau de gris homogène

Quelques statistiques de base permettent de synthétiser les informations sur l'image : valeur moyenne de l'histogramme, position du mode (la valeur maximale), écart-type...

Images 3D, 4D, 5D...

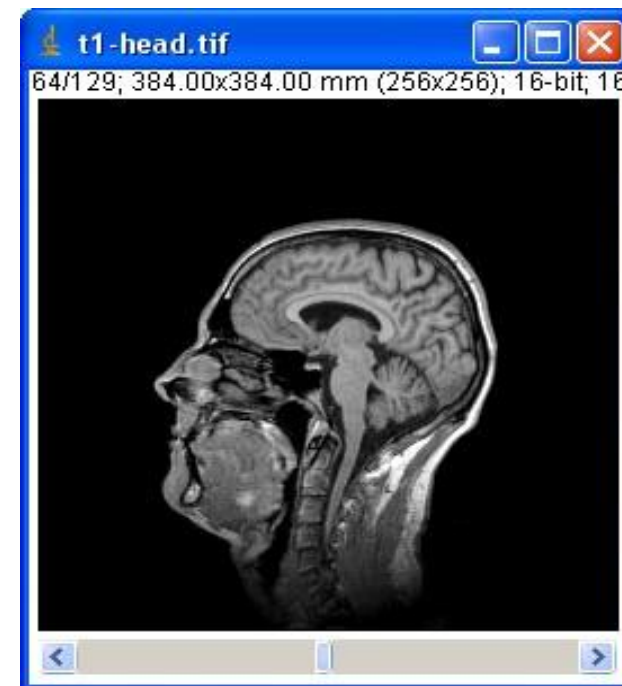
De plus en plus d'appareils d'acquisition produisent des images en 3 dimensions : IRM, tomographie, microscopie confocale... Les valeurs de l'image sont stockées dans des tableaux à 3 dimensions, et les éléments sont appelés des **voxels** (pour volume element) par analogie avec les pixels.

Si les images sont acquises à des temps différents, on obtient des films, où le temps joue le rôle d'une troisième ou quatrième dimension selon que l'image acquise est en 2D ou 3D. Pour les images hyperspectrales (acquisition sur plusieurs canaux ou filtres différents), on peut encore rajouter des dimensions supplémentaires. On arrive donc à des images allant jusqu'à 5 dimensions, ce qui rend parfois leur exploration problématique...

Visualisation plan par plan

Le mode de visualisation le plus simple dans ce cas est de fixer les dimensions supérieures à la deuxième, et de représenter une « coupe 2D » de l'image. Le logiciel **ImageJ** permet ainsi de représenter une image 2D choisie dans une pile en faisant varier le canal ou la dimension en z.

Ci-contre un exemple: le curseur sous l'image permet de choisir la « tranche » qui est affichée.



Les filtres

Le but du filtrage d'image est d'améliorer leur aspect, en rendant plus visibles les structures d'intérêt, tout en réduisant l'influence des informations parasites (bruit de l'image). On utilise pour cela différents filtres, qui vont transformer l'image de départ en une image filtrée, plus facile à analyser.

On peut classer les filtres en fonction de la quantité d'information nécessaire pour calculer la valeur d'un pixel de l'image résultat.

Certains filtres appliquent simplement une fonction de conversion à la valeur de chaque pixel (traitement ponctuel, ne dépendant que de la valeur du pixel). C'est le cas des filtres qui modifient l'histogramme, ou des filtres de binarisation.

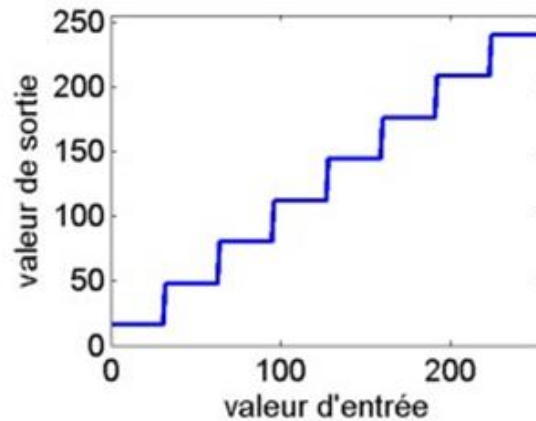
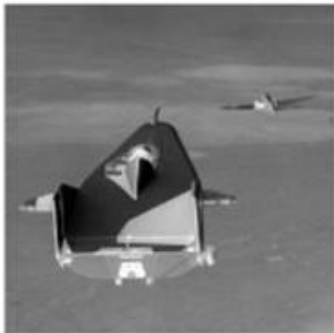
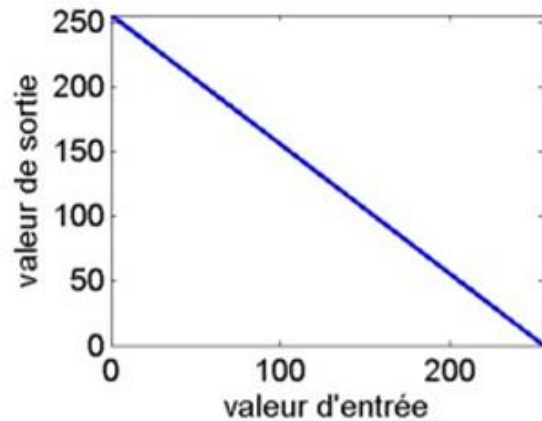
D'autres filtres opèrent sur un voisinage local de chaque pixel. Pour chaque pixel, le résultat du filtre ne dépendra que des valeurs des pixels voisins. Par exemple, une opération courante consiste à remplacer la valeur de chaque pixel par la valeur moyenne de ses voisins, ce qui a pour effet de réduire le bruit dans l'image. Une autre possibilité consiste à calculer la différence entre les valeurs des pixels du centre et de la périphérie du voisinage. L'effet sera alors de mettre en évidence les contours dans l'image. Le résultat du filtrage dépendra en grande partie de la taille du voisinage : plus le voisinage est grand, et plus le résultat sera différent de l'image de départ. Il est aussi possible d'utiliser des formes de voisinage différentes. Les plus classiques sont les voisinages carrés de taille 3×3 , 5×5 ... Mais on peut préférer des voisinages en forme de disque, qui traitent toutes les directions de manière identique, au contraire des formes carrées. On parle de convolution ou de filtre convolutif.

Enfin, d'autres filtres ont besoin de la totalité de l'image pour calculer le résultat en un point. C'est le cas des filtres de reconstruction morphologique, de seuillage par hystérésis, ou de la ligne de partage des eaux.

Tables de correspondances (Look-Up Table, LUT)

On peut modifier une image en calculant pour chaque pixel une nouvelle valeur $I'(u, v)$ qui ne dépend que de la valeur de départ $I(u, v)$ du pixel.

Le nombre de valeurs de départ étant fixe (256 pour les images courantes), on peut pré-calculer les valeurs finales et stocker le résultat dans un tableau, appelé table de correspondance, ou (Look-up table UT).



Application de deux tables de correspondances (LUT) sur l'image de départ :

une inversion (à gauche: nouvelle valeur = 255 - ancienne valeur), on obtient une image en négatif

et une re-quantification sur 8 niveaux de gris (à droite: 8 conditions du type si $x_1 < \text{valeur} < x_2$, alors valeur = x_3).

Principe des filtres linéaires

Les filtres linéaires sont une classe particulière de filtres, dont les propriétés mathématiques les rendent intéressants à utiliser. Dans le cas d'un filtrage linéaire de l'image, les opérations impliquées ne sont que des additions (ou soustractions) et multiplications (ou divisions) par des valeurs constantes, c'est-à-dire des combinaisons linéaires d'où leur nom.

En pratique, cela consiste à réaliser une moyenne pondérée de la valeur des pixels du voisinage. Par exemple, une opération courante consiste à remplacer la valeur de chaque pixel par la valeur moyenne de ses voisins. La nouvelle valeur du pixel $I(u, v)$ est déterminée par l'équation :

$$\begin{aligned} I^0(u, v) \leftarrow 1/9 [& I(u - 1, v - 1) + I(u, v - 1) + I(u + 1, v - 1) \\ & + I(u - 1, v) + I(u, v) + I(u + 1, v) \\ & + I(u - 1, v + 1) + I(u, v + 1) + I(u + 1, v + 1)] \end{aligned}$$

On peut classer les filtres linéaires en fonction de leur effet sur l'image de départ. Les **filtres moyennneurs** réduisent le bruit tout en lissant les structures. Les **filtres dérivatifs** permettent de mettre en évidence les bords des structures, et sont très utilisés pour détecter les contours. Les filtres dérivatifs du second ordre sont une alternative aux filtres dérivatifs simples, et permettent aussi de mettre en évidence les contours. On passe d'un type de filtre à l'autre simplement en changeant la valeur des coefficients des termes.

Filtres moyenneurs

Les filtres moyenneurs, comme leur nom l'indique, calculent la moyenne, éventuellement pondérée, des pixels situés dans le voisinage de chaque pixel.

Cette famille de filtres permet de **réduire le bruit dans l'image**, ce qui rend les zones homogènes plus lisses.

Par contre, les contours sont fortement dégradés, et **les structures trop fines peuvent devenir moins visibles**.

Filtres plats

Les filtres qui calculent une moyenne directe des pixels sont appelés filtres plats (**flat smoothing**).

Ils utilisent en général des voisinages carrés (3×3 , 5×5 ...).

Pour un filtre plat de taille 3×3 , la matrice de voisinage correspondante est la suivante :

$$H = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Pour des filtres plats de taille $n \times n$, la matrice est de la forme :

$$\frac{1}{n^2} \times \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}$$

3x3



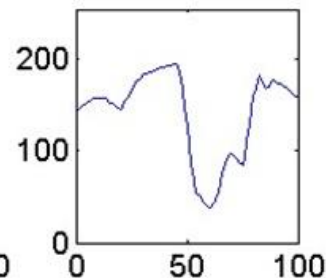
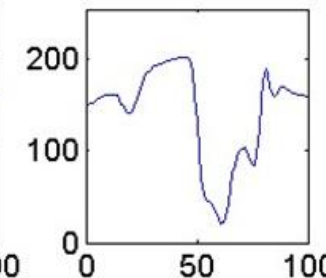
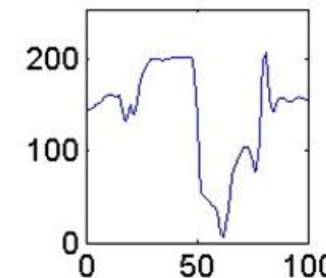
5x5



7x7



Ci-contre, le résultat d'un filtre plat avec trois tailles de filtre différentes montre l'influence de la taille du voisinage sur le résultat. En dessous, on montre les profils d'intensité des niveaux de gris le long d'une ligne horizontale passant au niveau de l'aile de l'avion. On constate la disparition des variations brusques (pics) dans le profil avec l'augmentation de la taille



Moyennes pondérées

On peut vouloir pondérer le poids associé à la valeur de chaque voisin. En particulier, on peut supposer que l'influence d'un voisin proche est plus importante que celle d'un voisin éloigné. On utilise donc aussi des filtres pondérés selon la distance au pixel central :

$$H = \begin{bmatrix} 0.05 & 0.15 & 0.05 \\ 0.15 & 0.20 & 0.15 \\ 0.05 & 0.15 & 0.05 \end{bmatrix} = \frac{1}{20} \begin{bmatrix} 1 & 3 & 1 \\ 3 & 4 & 3 \\ 1 & 3 & 1 \end{bmatrix}$$

Les pondérations les plus classiques sont d'utiliser un noyau gaussien ou parabolique.

Pour un noyau gaussien, les coefficients sont calculés à partir de la distance au pixel central du noyau du filtre :

$$G(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

Un noyau discret 5×5 approchant un filtrage par un noyau gaussien avec $\sigma = 2$ est le suivant :

$$H = \frac{1}{101} \begin{bmatrix} 1 & 3 & 4 & 3 & 1 \\ 3 & 5 & 7 & 5 & 3 \\ 4 & 7 & 9 & 7 & 4 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 4 & 3 & 1 \end{bmatrix} \quad \text{Ou encore (Canny)} \quad h = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Filtres dérivatifs

Dans une image en niveaux de gris, les contours des structures d'intérêt peuvent être mis en évidence en utilisant des filtres linéaires qui calculent les variations de niveaux de gris. Pour cela on utilise des noyaux qui calculent la différence entre les niveaux de gris dans une direction, et les niveaux de gris dans la direction opposée.

Les **filtres de Prewitt**, par exemple, sont de la forme :

$$P_{0^\circ} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, P_{90^\circ} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Un filtre similaire est le **filtre de Sobel**, qui donne un poids plus important aux pixels plus proches du pixel central :

$$S_{0^\circ} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_{90^\circ} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Il est aussi possible de considérer les directions diagonales, en orientant différemment les coefficients :

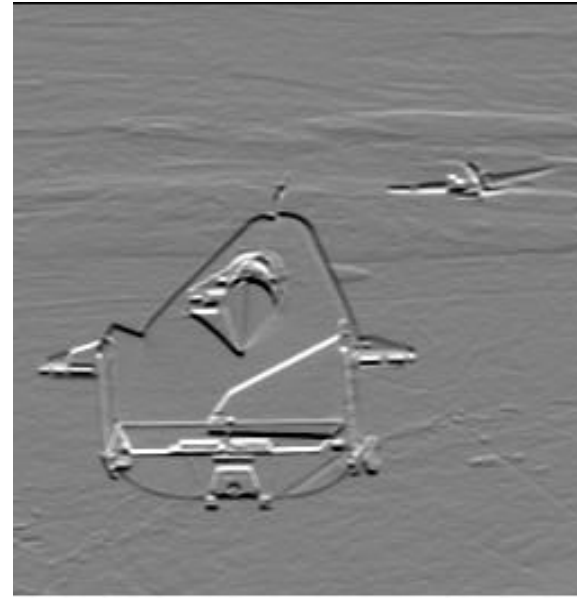
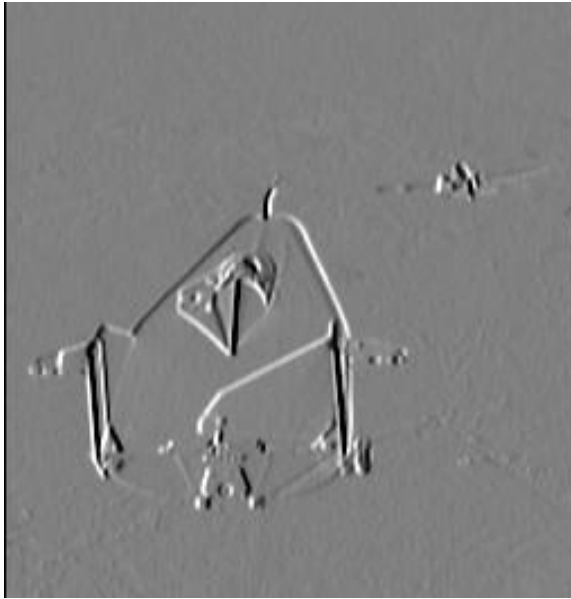
$$P_{45^\circ} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, P_{135^\circ} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

Filtre de contour

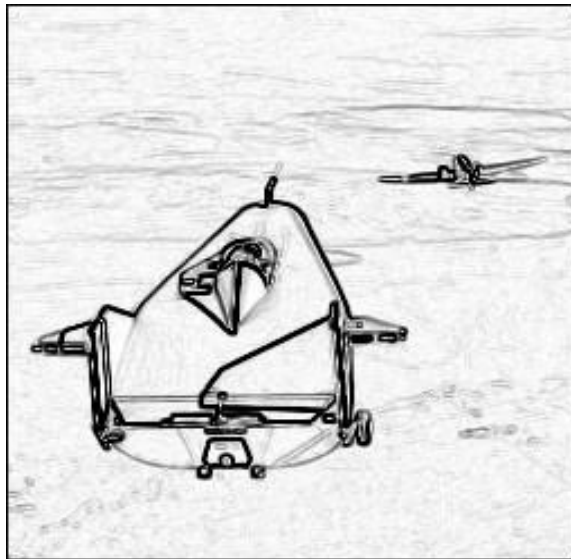
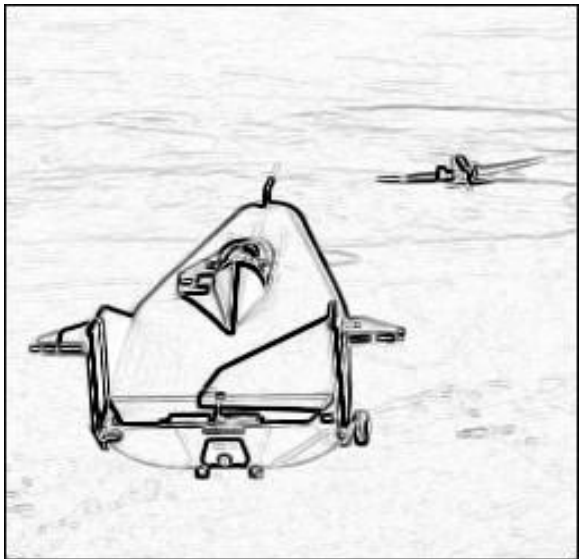
L'image finale des contours est obtenue en combinant les résultats des filtres Prewitt ou Sobel dans plusieurs directions.

Une combinaison pratique est de calculer le module des directions orthogonales, pour obtenir une image gradient :

$$I' = \sqrt{P_{0^\circ}(I)^2 + P_{90^\circ}(I)^2}$$



Une image en niveaux de gris, et le résultat d'un filtre de Prewitt dans les directions horizontale et verticale. Les images filtrées ont été renormalisées entre 0 et 255



Résultat de la détection des contours en utilisant les fitres de Prewitt (à gauche) ou de Sobel (à droite), et en calculant le module sur les directions horizontales et verticales.

Filtres d'ordre

Les filtres d'ordre considèrent l'ensemble des pixels du voisinage, classés par ordre croissant d'intensité, et sélectionnent le i -ème pixel de la liste pour l'affecter au résultat.

Les filtres les plus utilisés de cette famille sont les **filtres min, max, et médian**.

Le filtre médian, comme son nom l'indique, sélectionne la valeur médiane des pixels du voisinage. Pour un voisinage carré 5×5 , par exemple, les valeurs des 25 voisins sont ordonnées par ordre croissant, et la 13-ième valeur est gardée comme résultat. **L'avantage du filtre médian est qu'il permet de réduire fortement le bruit dans l'image, et qu'il préserve beaucoup mieux les bords que les filtres moyennés.**

Les **filtres minimum et maximum** ordonnent l'ensemble des pixels du voisinage, et sélectionnent soit la plus petite ou la plus élevée. Cette famille de filtre **permet de supprimer des petits détails très lumineux ou très sombres**, mais affecte fortement la taille des objets.

Le filtre de **dynamique locale («range filter»)** permet de quantifier si un pixel est dans une zone homogène ou non. Il se calcule à partir de la différence entre les filtres minimum et maximum. Il donne des résultats souvent assez similaires à ceux du gradient. Un avantage est que les résultats peuvent être codés avec le même type de données que l'image d'origine, alors que le gradient nécessite en général de passer par des nombres à virgule flottante. **Il existe de nombreux autres filtres de réduction du bruit dans les images numériques. Parmi les plus connus, on peut citer les filtres de Nagao, de Kuwahara, ou encore le filtre bilatéral.**

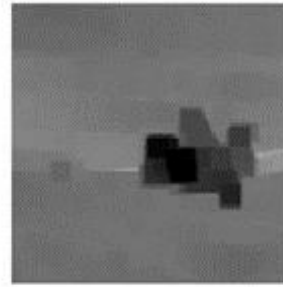
Min 3x3



Min 5x5

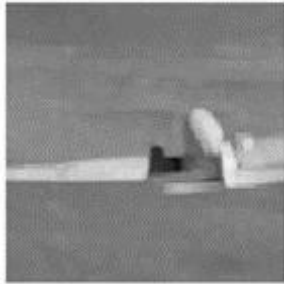


Min 7x7

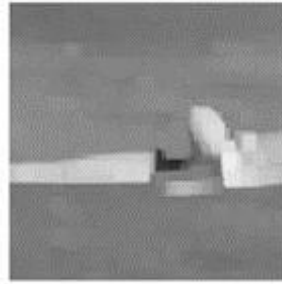


Influence de la taille du voisinage sur le résultat des filtres min et max.

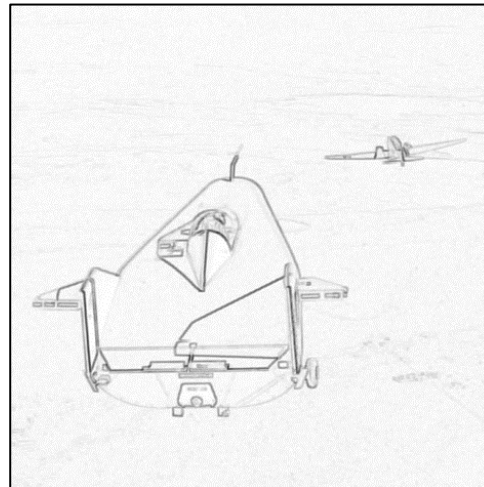
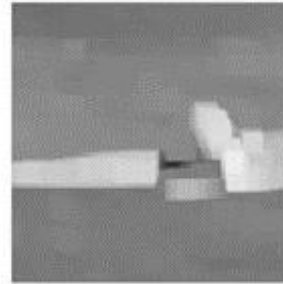
Max 3x3



Max 5x5



Max 7x7



Calcul du filtre de dynamique locale sur une image d'avion

3x3



5x5



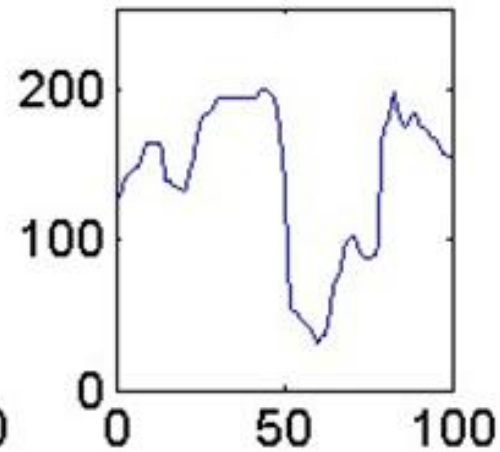
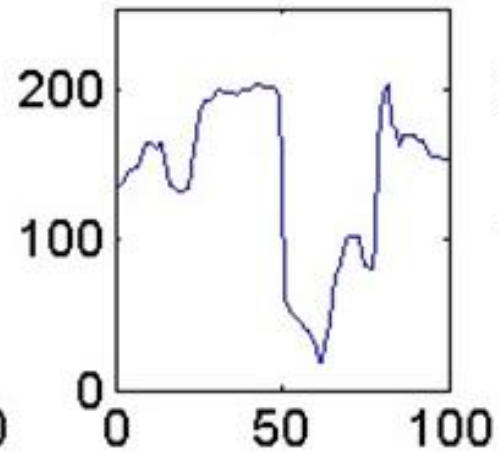
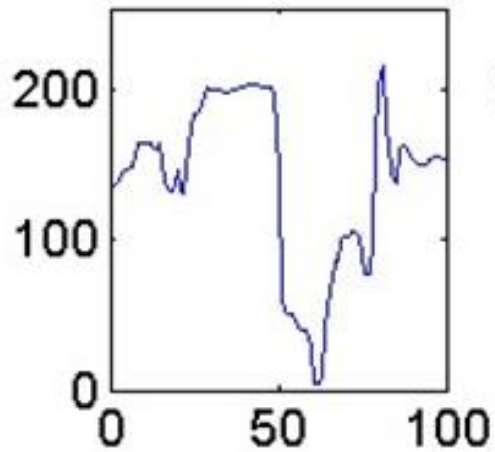
7x7



Influence de la taille du voisinage sur le résultat d'un filtre médian.

En dessous, on montre les profils d'intensité des niveaux de gris le long d'une ligne horizontale passant au niveau de l'aile de l'avion.

À comparer avec la figure équivalente pour les filtres moyennneurs



Traitement des images couleur

Le traitement des images couleurs pose parfois quelques difficultés, du fait que chaque pixel est représenté non pas par une valeur, mais par un ensemble de valeurs. Le même problème se rencontre pour le traitement des images multicanaux ou multi spectrales.

Une solution radicale consiste à convertir l'image en niveaux de gris, et à ne travailler que sur l'image d'intensité. Cette approche présente l'inconvénient de perdre la richesse de l'information couleur, mais est parfois la plus rapide à mettre en œuvre.

Une approche plus satisfaisante est l'approche marginale, qui consiste à séparer les différents canaux, et à travailler séparément sur chacun des canaux qui peuvent être vus comme des images en niveaux de gris.

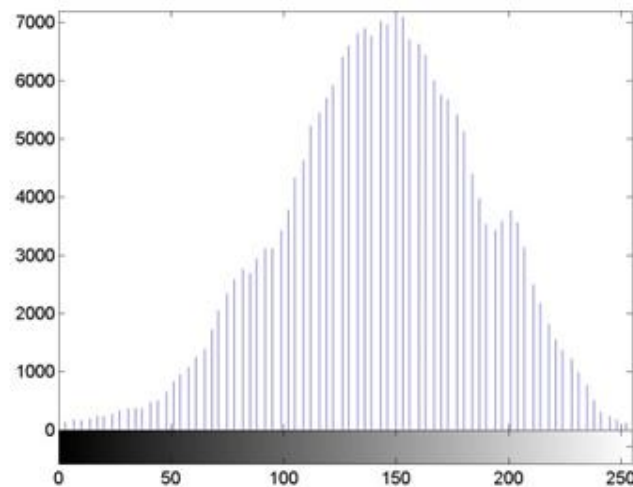
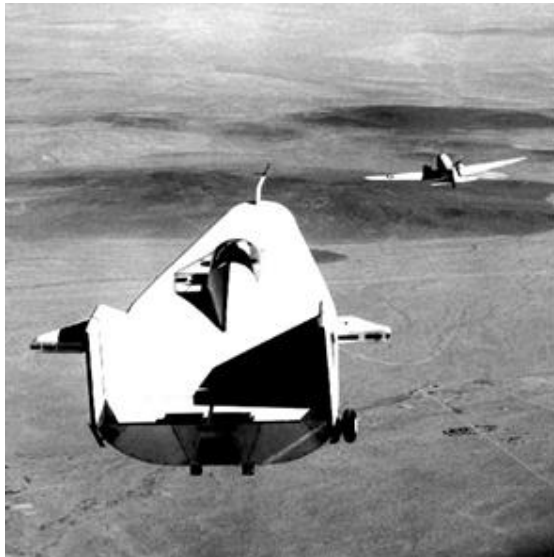
Le traitement des images couleur peut aussi se faire en tenant compte de la nature vectorielle de l'information fournie par chaque pixel. Cette approche globale nécessite de développer des algorithmes spécifiques, et est donc moins souvent représentée dans la palette des outils disponibles.

Extension de dynamique

Cette méthode consiste à utiliser au mieux les niveaux de gris situés dans une plage de valeur donnée, et à les étaler de manière régulière pour couvrir toute la dynamique possible des niveaux de gris. En pratique, la relation entre les niveaux de gris d'entrée et les niveaux de gris de sortie est une fonction affine, arrondie à des valeurs entières entre 0 et 255, que l'on peut calculer à partir des valeurs minimum et maximum des plages de niveaux de gris en entrée :

$$f(g) = 255 \times \frac{g - g_{\min}}{g_{\max} - g_{\min}}$$

Exemple après avoir transformé les niveaux de gris situés entre les valeurs 95 et 170 de l'image de départ en des niveaux de gris situés entre 0 et 255:



Seuillage

Le seuillage (en anglais **threshold**) est sûrement la méthode de segmentation la plus simple.

Elle consiste à tester pour chaque pixel de l'image si sa valeur est supérieure ou inférieure à un certain seuil, et produit une image binaire regroupant les résultats.

Il est aussi possible d'utiliser plusieurs seuils, voire de sélectionner les pixels appartenant à une plage spécifique de niveaux de gris.

Il existe plusieurs méthodes pour détecter de manière automatique la valeur du seuil à appliquer. L'une des plus répandue est la **méthode d'Otsu**. Elle consiste à faire l'hypothèse que l'image contient deux classes, décrites chacune par une partie de l'histogramme de l'image. La qualité du seuillage est quantifiée en mesurant la **variance** des niveaux de gris de chaque classe. On cherche ensuite la valeur de seuil qui minimise les variances des deux classes, et on utilise cette valeur pour binariser l'image.

Seuillage à hystérésis

Le seuillage à hystérésis est une méthode un peu plus perfectionnée, qui permet d'améliorer la détection des contours des zones contrastées, tout en évitant de détecter des zones du fond. Le principe est d'utiliser deux seuils : un seuil haut et un seuil bas. On sélectionne ensuite les pixels au-dessus selon les règles suivantes:

- Inférieur au seuil bas, le point est rejeté ;
- Supérieur au seuil haut, le point est accepté comme formant un contour ;
- Entre le seuil bas et le seuil haut, le point est accepté s'il est connecté à un point déjà accepté.

Classification de pixels

Les méthodes de classification, utiles pour la segmentation d'images couleur, peuvent aussi s'employer pour segmenter des images complexes en niveaux de gris.

Pour discriminer efficacement les différentes classes de pixel, on cherche à décrire chaque pixel par un ensemble de caractéristiques.

En appliquant une large variété de filtres de différente nature à l'image d'origine, on obtient pour chaque pixel autant de valeurs que le nombre de filtres appliqués.

En choisissant des régions représentatives de chaque classe ou catégorie dans l'image, on peut initialiser un classifieur, qui va calculer les caractéristiques de chaque classe.

En appliquant le résultat sur l'image d'origine, et si la famille de filtres a été correctement choisie, il est possible d'obtenir une segmentation d'image qu'il aurait été difficile d'obtenir autrement.

Détection de contours (edge detection)

Segmentation de régions par ligne de partage des eaux

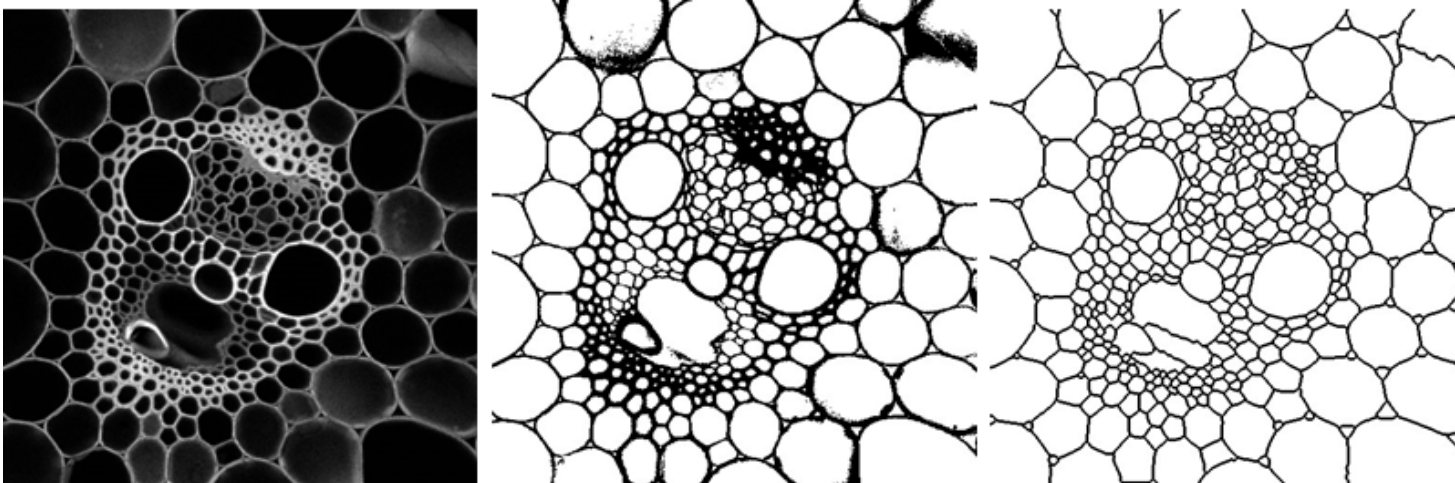
La ligne de partage des eaux est un outil de segmentation issu de la morphologie mathématique.

L'algorithme s'applique à des images en niveaux de gris.

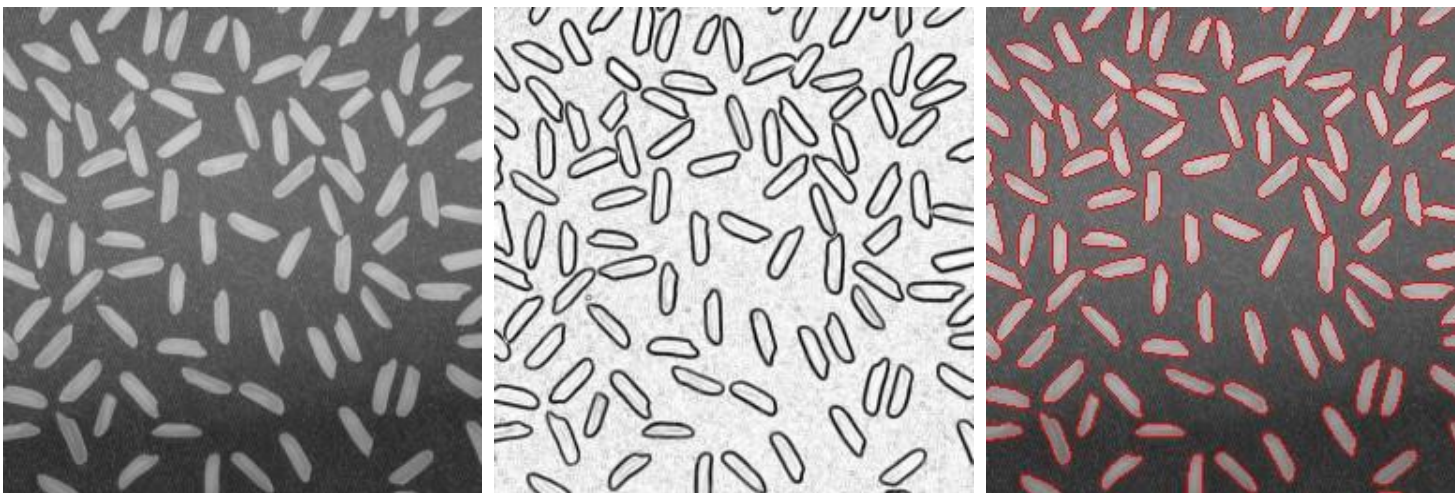
L'image est assimilée à un paysage dont l'altitude de chaque point est donnée par l'intensité du pixel correspondant.

Les pixels de forte intensité (blancs) correspondent à des sommets, tandis que les pixels de faible intensité (noir) correspondent à des fonds de vallées.

Le principe de l'algorithme est de partitionner l'image selon une approche topographique : on recherche les bassins versants, séparés par des lignes de crêtes.



Une image de cellules végétales (à gauche), et le résultat de la segmentation par deux méthodes : seuillage (au centre), et ligne de partage des eaux (à droite).



Application de la ligne de partage des eaux sur le gradient d'une image pour détecter le contour des grains. (a) image d'origine (b) norme du gradient (représenté en inversé) (c) résultat de la ligne de partage des eaux superposée sur l'image d'origine.

Codage des contours

On considère, pour chaque région ou particule de l'image binaire, la succession des pixels qui forment son contour.

Au lieu de stocker l'ensemble des coordonnées de chaque pixel, on code la direction du pixel suivant, en utilisant une valeur entre 0 et 7 (8 directions).

On obtient ainsi le code de Freeman.

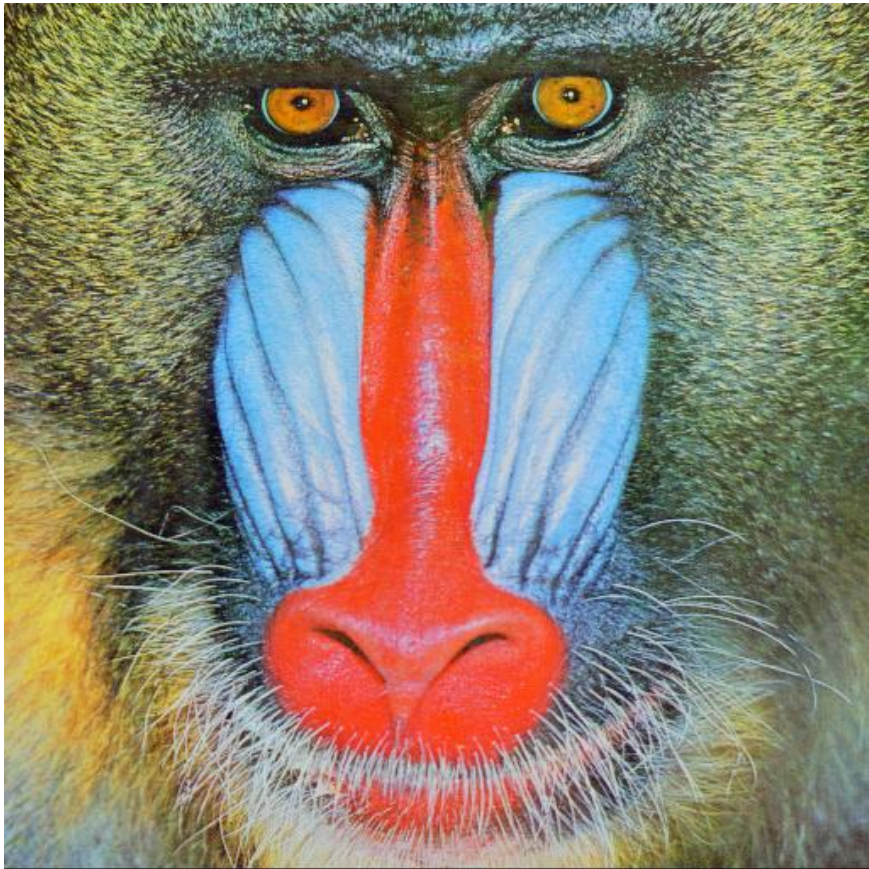
Exercices 2

Le but de ces exercices est à la fois de présenter quelques méthodes de manipulation d'images sans utiliser les fonctions toutes faites d'un module et de le faire d'une façon un peu plus mathématique que d'ordinaire en utilisant numpy et la manipulation mathématique des arrays.

PIL ou pillow nous permettra d'ouvrir les images directement dans un tableau numpy, mais nous n'utiliserons pas ses fonctionnalités de traitement d'images, nous le ferons en codant nous même les fonctions.

Lenna ou Lena est une image test standard largement utilisée dans le domaine du traitement de l'image depuis 1973. Il s'agit d'une partie de la photo du mannequin suédois Lena, prise par le photographe Dwight Hooker, tirée de la page centrale du magazine Playboy de novembre 1972. (<https://fr.wikipedia.org/wiki/Lenna>)

Cette image pose problème au mouvement féministe qui suggère de ne plus l'employer. Aussi pouvez-vous utiliser l'image 4-02-03 issue d'une banque standard, même si elle n'est pas aussi classique pour comparer les effets d'un algorithme de traitement d'image.



Ces images sont disponibles en différents formats (jpg, png, bmp, gif, tif) et en différentes tailles (220 x 220 pixels ou 512 x 512), en couleur et en noir et blanc (BW) dans le répertoire image



Exercice 2.0 (déjà fait ex0.py ou ex3.py)

L'exercice 2.0 vous donne la syntaxe d'ouverture d'une image et de sauvetage de la nouvelle image. En fait, on ouvre l'image dans un tableau numpy nommé image, on récupère ses dimensions pour les afficher, on la transforme (dans cet exercice, on la recopie simplement) et on la sauve sous un autre nom/format.

On peut aussi facilement récupérer les dimensions de l'image avec la fonction shape et stocker dans nb_lignes le nombre de lignes de notre tableau et dans nb_colonnes le nombre de colonnes pour les afficher.

Cet exercice est déjà fait (ex2-1.py), il doit vous permettre de vérifier que votre environnement est correct (bibliothèque PIL ou Pillow) et vous permettre de démarrer plus rapidement

Note:

dans les exercices, nous laisserons le soin à PIL/Pillow d'encoder/décoder les formats d'image sans nous en préoccuper, ce qui nous intéresse, ce sera les modifications/traitements sur l'image.

Vous pourriez bien sûr ouvrir le fichier via python, mais il faudra alors parfaitement connaître la structure des fichiers graphiques ce qui sort du cadre de ce cours.

Exercice 2.2 (déjà fait ex2-2.py)

Chaque pixel est représenté dans le tableau numpy par un triplet (r,v,b) où r, v et b sont des nombres entre 0 et 255 (8 bits de profondeur) représentant la "proportion" de rouge, vert et bleu pour afficher chaque pixel. Un print du tableau affichera la valeur des pixels sous forme de triplets

Dans cet exercice on ouvre une minuscule image d'un damier de couleur rouge, vert et bleu de 3x2 pixels et on affiche le tableau correspondant. Dans ce format, l'ouverture de 6x2.bmp permet de voir clairement la structure.

Les pixels sont repérés dans le tableau par la ligne et la colonne comme dans une matrice. Autrement dit, pour récupérer le pixel tout en bas à gauche de l'image de Lenna, on utilisera `image[511,0]` puisque l'image est de dimension 512 par 512, que la première ligne porte le numéro « 0 », donc la dernière le numéro 511 et que le tableau de réception de l'image s'appelle `image`.

On peut aussi facilement récupérer les dimensions de l'image avec la fonction `shape` et stocker dans `nb_lignes` le nombre de lignes de notre tableau et dans `nb_colonnes` le nombre de colonnes.

Si on effectue une manipulation sur le fichier, on sauvera l'original dans `image_entree.png` (pour que l'original `Lenna.png` reste toujours dispo pour les autres scripts) et l'image modifiée dans `image_sortie.png`

Exercice 2.2 suite

Vous remarquerez que l'on a rajouter la **bibliothèque time** qui nous servira à mesurer le temps mis pour réaliser nos algorithmes de traitement d'image en calculant la différence entre l'heure de début du script et l'heure de fin.

On a aussi ajouté la **bibliothèque tkinter** afin d'afficher l'image de départ et l'image du résultat du traitement, cela vous évite d'avoir à les ouvrir pour comparer. Si vous gardez les noms « image_entree.png » et « image_sortie.png », vous ne devrez rien changer dans la boucle tkinter, l'affichage se fera automatiquement dans une fenêtre graphique en popup. L'affichage des tableaux et textes continuera à se faire dans la console.

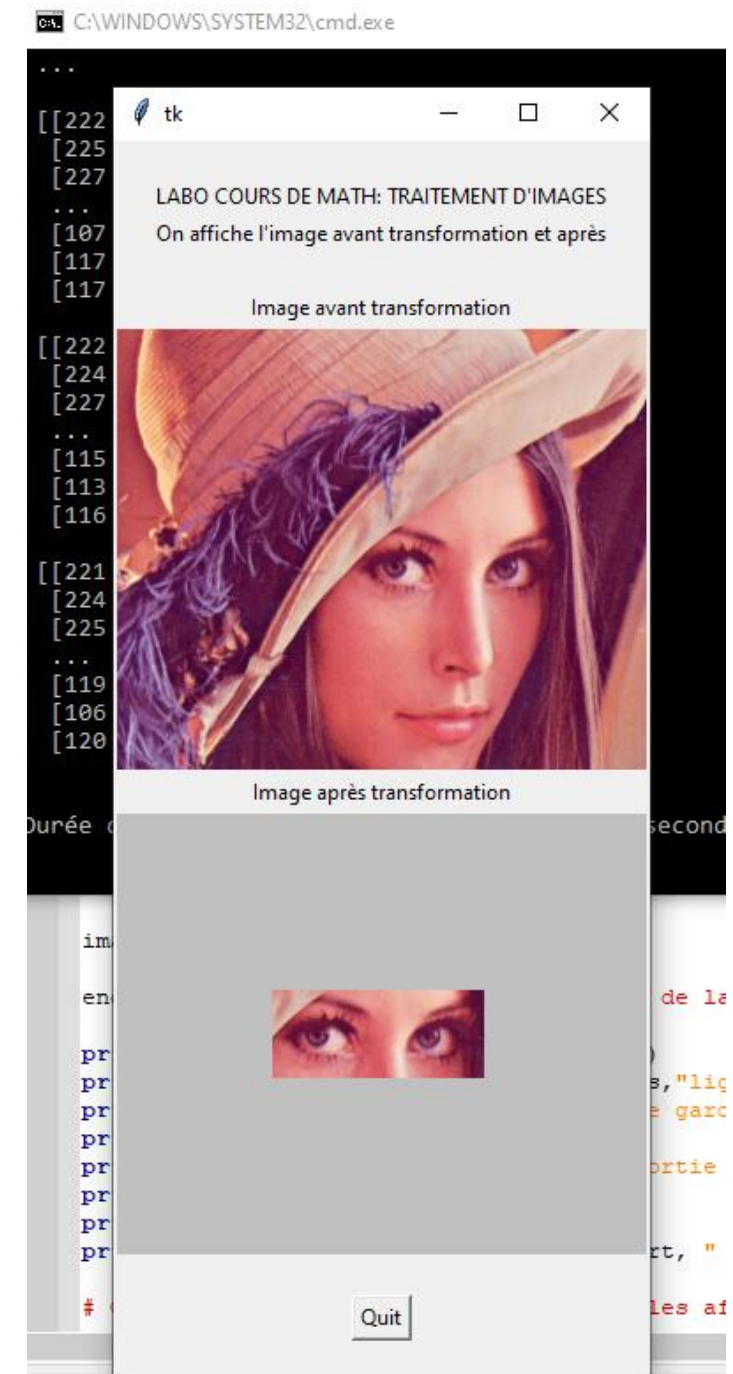


Exercice 2.3 récupérer une partie d'une image.

Pour cela il suffit de ne conserver qu'une partie de notre tableau de données.

C'est très facile avec un tableau numpy : `image[2:6,3:9]` permet de récupérer la zone entre la ligne 2 et 5 et la colonne 3 et 8 (en python on ne prend pas la dernière valeur).

Si on souhaite récupérer le regard de Lenna512, on peut récupérer la zone entre les lignes 240 et 290 et les colonnes 240 et 360.



Exercice 2.4 image miroir horizontale et verticale

Pour retourner une image (inverser gauche-droite: la colonne 0 devient la 511, la 1 devient la 510, la 2, 509), il suffit simplement que sur chaque ligne, on mette dans le pixel situé à la colonne col la valeur du pixel de la colonne $(nb_colonnes-1)-col$ de l'image originale.

Exemple:

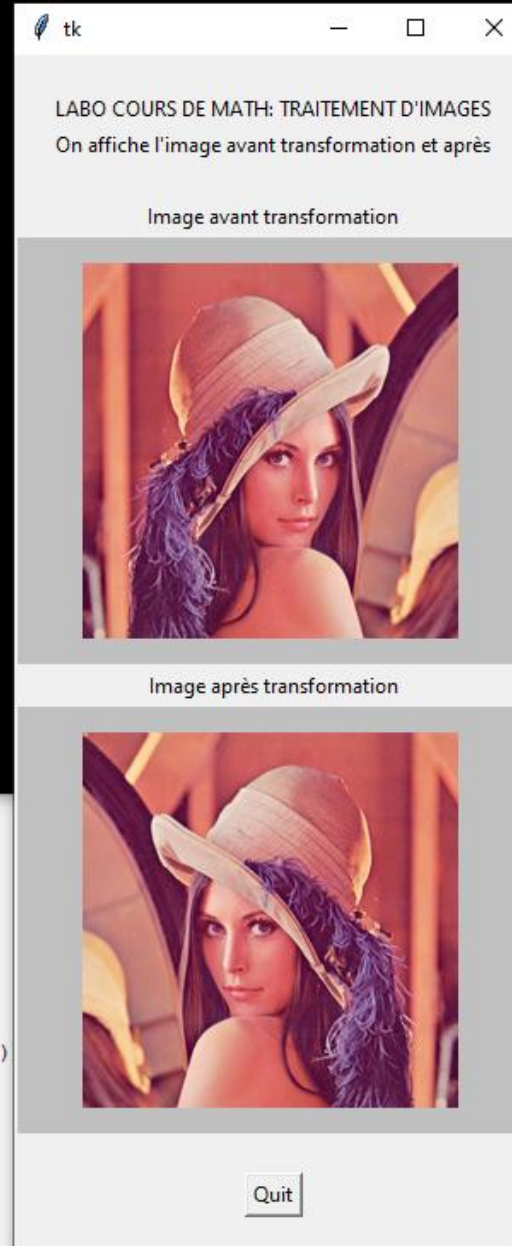
On balaye toutes les lignes et
pour la colonne 0 ($col=0$) on va mettre la valeur du pixel de la colonne $512-1-0=511$
Pour la colonne 1 on mettra la valeur du pixel $512-1-1=510$ etc..
Jusqu'à la colonne 511 avec la valeur du pixel $512-1-511=0$
Et on passe à la ligne suivante.

Pour faire une image upside/down, remplacer ligne par colonne et colonne par ligne dans l'algorithme! Faites les deux!

```

19
20 #-----
21 # Importation des bibliothèques de numpy et de PIL
22 #-----
23 # Utilisation des arrays pour traiter des images
24 import time
25 from tkinter import *
26 from PIL import Image, ImageOps
27 import numpy as np
28
29 # Vous avez ouvert l'image Lenna .png
30 # qui a pour dimensions 220 lignes et 220 colonnes
31
32 # Vous l'avez ensuite inversée gauche/droite (effet miroir)
33 # encodage des fonctions
34 #-----
35 # Durée du traitement: 0.05296754837036133 seconde
36
37 #-----
38 # encodage du programme
39 #-----
40
41 print("Utilisation de numpy et de PIL")
42 print("-----")
43 print("Utilisation des arrays pour traiter des images")
44
45 # On charge l'image et on la convertit en array numpy
46 nom='Lenna'
47 image_entree = Image.open('image_entree.png')
48 image = np.asarray(image_entree)
49 nb_lignes,nb_colonnes,_ = image.shape
50
51 # Traitement de l'image
52 start = time.time()
53
54 image_sortie = np.copy(image)
55 for ligne in range(nb_lignes):
56     for col in range(nb_colonnes):
57         image_sortie[ligne,col] = image[ligne,nb_colonnes-1-col]
58
59 end = time.time() # stocke l'heure de la fin du traitement
60
61 print("\nVous avez ouvert l'image ", nom, ".png")
62 print("qui a pour dimensions ", nb_lignes,"lignes et ", nb_colonnes, " colonnes\n")
63 print("Vous l'avez ensuite inversée gauche/droite (effet miroir)\n")
64 print ("Durée du traitement: ",end - start, " seconde\n")
65
66 # On sauvegarde les images pour pouvoir les afficher
67
68 Image.fromarray(image).save("image_entree.png")
69 Image.fromarray(image_sortie).save("image_sortie.png")
70

```



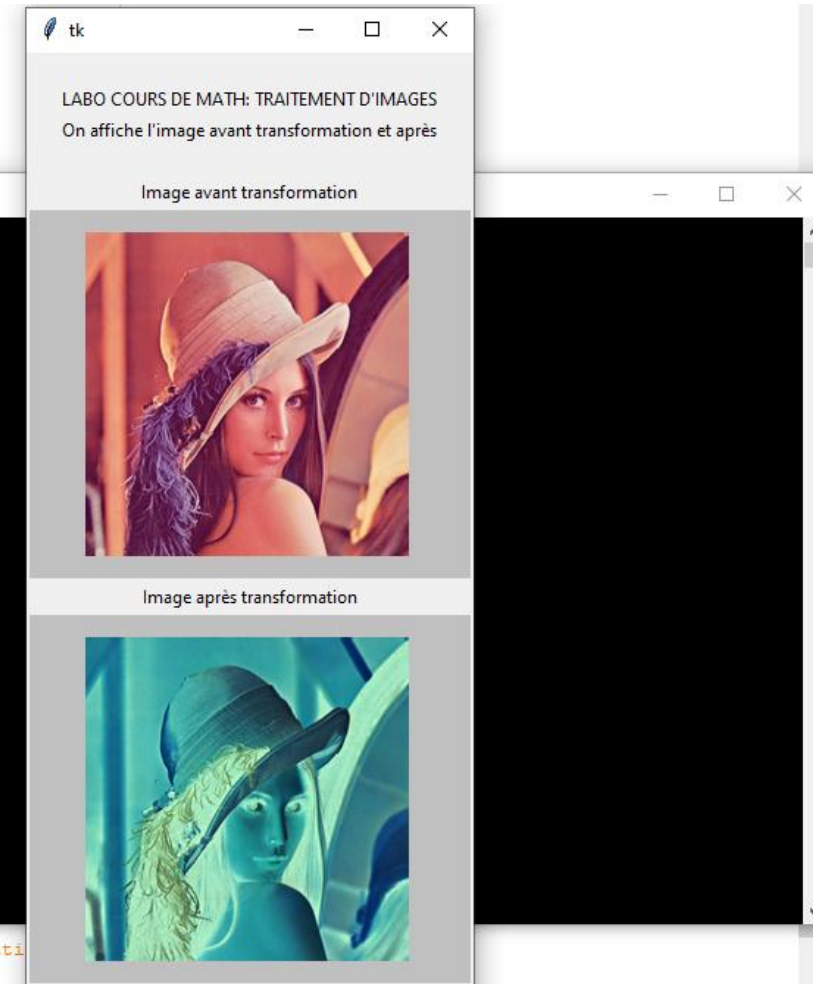
Exercice 2.5 image en négatif

Pour inverser les couleurs (faire un négatif), il suffit de remplacer chaque couleur (le rouge, le vert et le bleu) de valeur v par la valeur $255 - v$ (le complément à 255).

Faites-le :

- sur une image couleur
- sur une image en niveau de gris

```
41 # On charge l'image et on la transforme en tableau
42
43 nom='Lenna'
44 image_entrée = Image.open(nom + ".png")
45 image = np.asarray(image_entrée)
46 nb_lignes,nb_colonnes,_ = image.shape
47
48 # Traitement de l'image
49
50 image_sort = image.copy()
51 for ligne in range(nb_lignes):
52     for colonne in range(nb_colonnes):
53         for couleur in range(3):
54             image_sort[ligne][colonne][couleur] = 255 - image[ligne][colonne][couleur]
55
56 print("\nVous avez ouvert l'image Lenna.png")
57 print("qui a pour dimensions 220 lignes et 220 colonnes")
58 print("Vous l'avez ensuite passé en négatif")
59
60 # On sauvegarde l'image
61 image_sort = Image.fromarray(image_sort)
62 image_sort.save("Lenna_negatif.png")
63
64 # Affichage de l'image avant et après transformation
65
66 #-----
67 # Affichage de l'image avant transformation
68 #-----
69
70 root=Tk()
71
72 empty_line
73 empty_line
74 empty_line
75 empty_line
76 champ_label = Label(root, text="Image avant transformation")
77 champ_label.pack()
78 empty_line
79 empty_line
80
81 champ_label_result = Label(root, text="Image après transformation")
82 champ_label_result.pack()
```



Exercice 2.6 isoler une des 3 couleurs

A partir d'une image, on peut isoler une seule composante de couleur, le rouge par exemple. On peut aussi créer 3 images différentes dans lesquelles on ne garde à chaque fois que la composante rouge, verte ou bleue de chaque pixel. L'image originale étant en fait la superposition des trois.

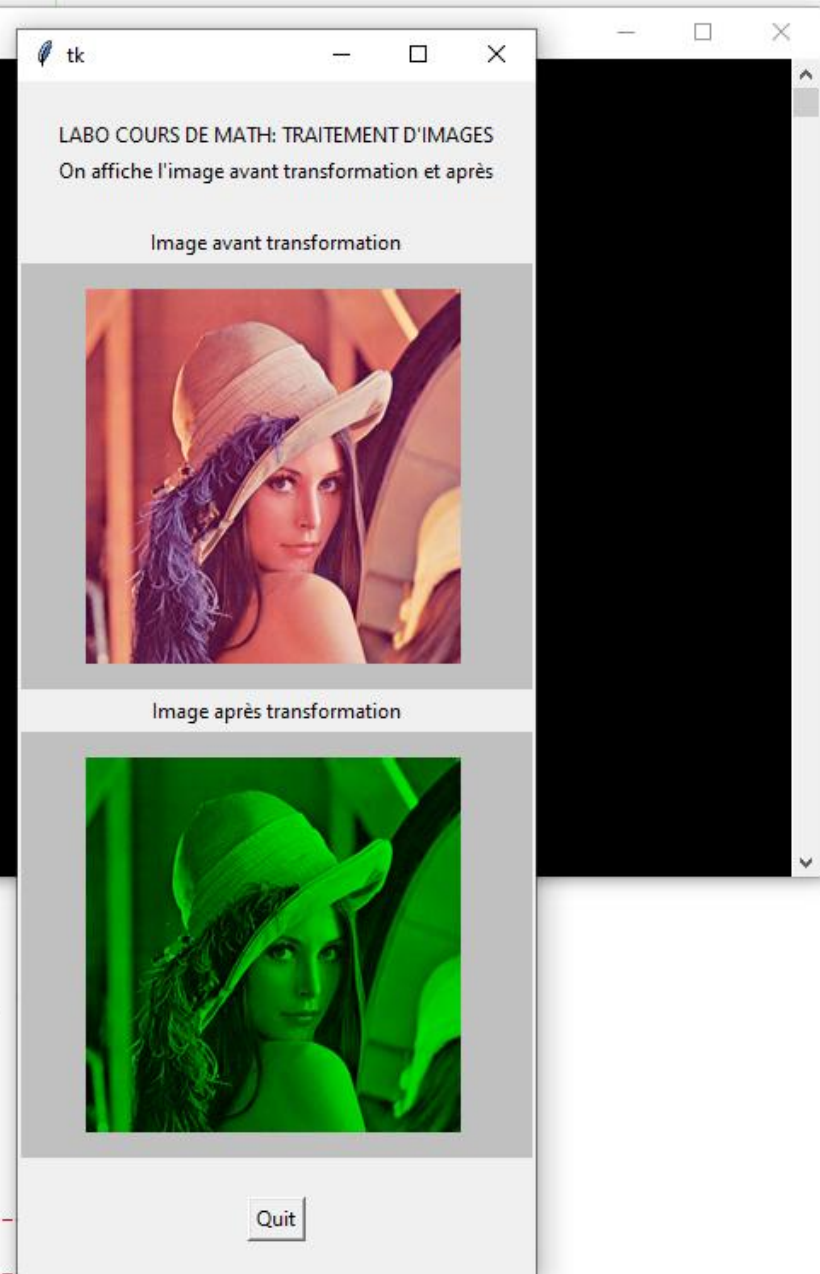
Pour isoler une couleur, il suffit simplement de garder la valeur de la couleur souhaitée et de remplacer les autres par 0. Pour cela, le plus simple est de multiplier le tuple contenant les valeurs des 3 couleurs par un tableau (1,0,0) si vous voulez garder le rouge, (0,1,0) si vous voulez garder le vert et (0,0,1) si vous voulez garder le bleu (le triplet affiche les couleurs dans l'ordre Rouge (R), Vert (V) et Bleu (B)).

Le même algorithme peut servir à modifier la colorimétrie d'une image. On multiplie chaque couleur non plus par 1 ou 0 mais par un réel compris entre 0 et 1. Ainsi (0.8, 1, 0.95) diminuerait le rouge de 20%, garde le vert tel quel et diminue le bleu de 5%. Attention, il faudra régler le problème de multiplication entre integer et float et remettre le résultat final sous forme d'integer compris entre 0 et 255!

```

20 # -----
21 # Importation des bibliothèques
22 # -----
23
24 import Utilisation de numpy et de PIL
25 from t -----
26 from Utilisation des arrays pour traiter des images
27 import
28
29 # ----- Vous avez ouvert l'image Lenna .png
30 # encod qui a pour dimensions 220 lignes et 220 colonnes
31 # -----
32 # Vous avez ensuite isolé la composante Verte de l'image
33 # -----
34 # encod
35 # -----
36
37 print(
38 print(
39 print(
40
41 # On c
42
43 nom='L
44 image_
45 image_
46 nb_lig
47 nb_col
48
49 # Trai
50
51 np.arr
52 image_
53 for li
54     for col in range(nb_colonnes):
55         image_sortie[ligne,col] = image[ligne,col]*[0,1,0]
56
57 print("\nVous avez ouvert l'image ", nom, ".png")
58 print("qui a pour dimensions ", nb_lignes,"lignes et ", nb_colonnes,
59 print("Vous avez ensuite isolé la composante Verte de l'image\n")
60
61 # On sauvegarde les images pour pouvoir les afficher
62
63 Image.fromarray(image).save("image_entree.png")
64 Image.fromarray(image_sortie).save("image_sortie.png")
65
66 # -----
67 # Affichage dans tkinter
68 # -----

```



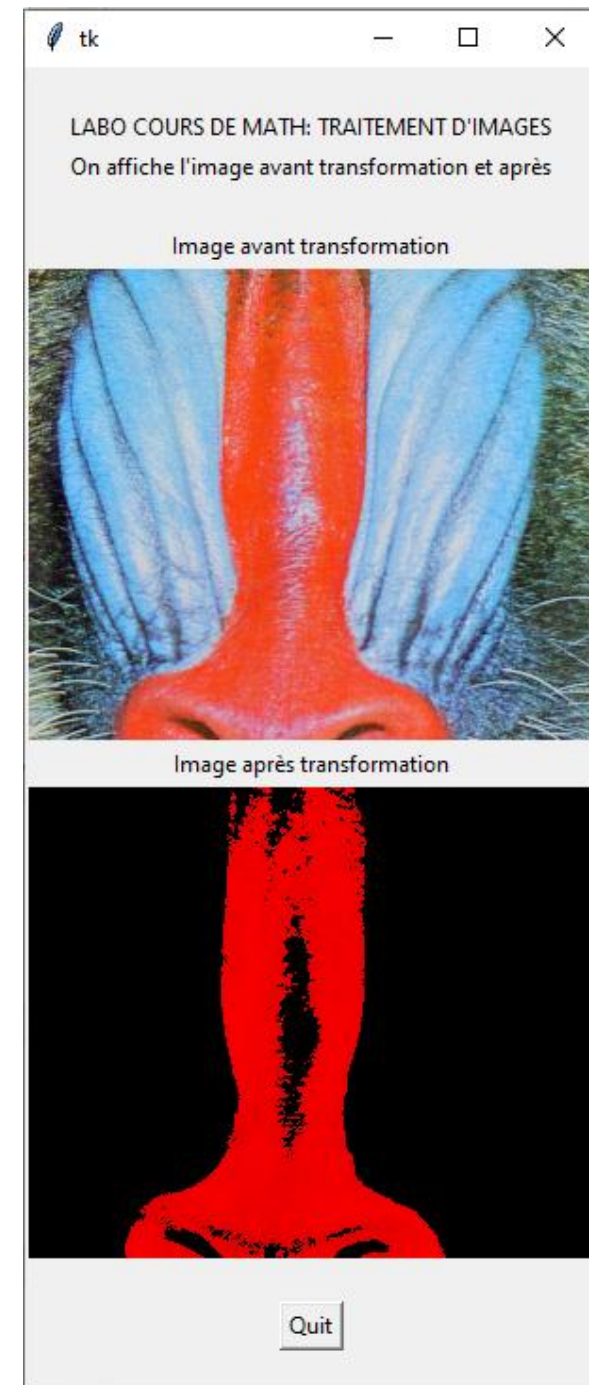
Exercice 2.6b isoler une des 3 couleurs à partir d'un seuil

Souvent dans l'analyse scientifique d'une image, on désire extraire de l'image les zones correspondant à un seuil minimal pour une couleur donnée.

Imaginer une petite modification de votre script qui fasse passer en noir les zones dont la composante rouge est inférieure à un certain seuil et conserve (ou force à 255 selon votre envie) les zones dont la composante rouge dépasse ce seuil.

Fixez le seuil à par exemple 220 et appliquez le filtre à l'image du singe. Vous isolerez très facilement son nez.

Ce principe est la base de nombreux algorithmes de sélection de zone et de mesure (par exemple sélection d'une lésion dans une radiographie permettant sa mesure)



Exercice 2.7 Mettre en nuance de gris

On pourrait imaginer faire la somme des valeurs rouge, vert et bleu et diviser par 3.

Mais dans ce cas, le gris correspondant à un rouge pur, un vert pur ou un bleu pur serait identique et donc l'image manquerait de rendu.

On peut trouver sur Wikipédia la formule suivante pour mettre une photo en nuance de gris :

Pour chaque pixel, on remplace la couleur (r,g,b) par la couleur (luminance, luminance, luminance) où

$$\text{luminance} = 0.2126 * r + 0.7152 * v + 0.0722 * b$$

Dans ce cas un rouge pur (255,0,0) est transformé en (54 ,54 ,54)

un vert pur (0, 255,0) est transformé en (182 , 182 , 182)

un bleu pur (0,0, 255) est transformé en (18 ,18 ,18)

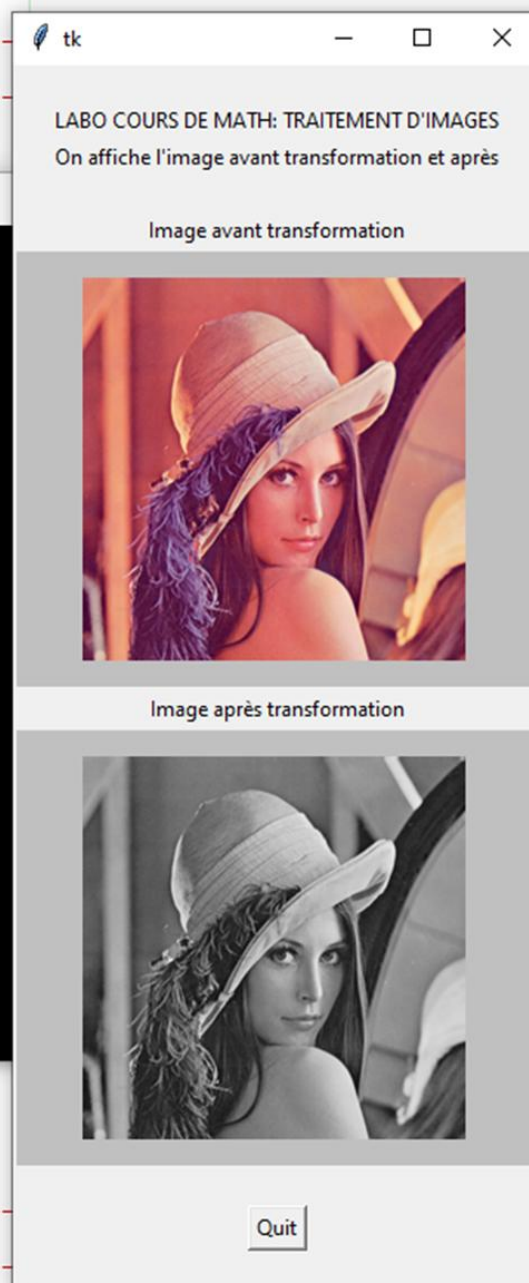
(il faut bien sur normaliser le résultat comme un entier compris entre 0 et 255!)

Vous pouvez facilement tester d'autres valeurs pour les coefficients, notamment 0,33 pour les trois

```

31 # -----
32
33 # -----
34 # encodage du programme principal
35 # -----
36
37 print("Utilisation de numpy et de PIL")
38 pr
39 pr C:\WINDOWS\SYSTEM32\cmd.exe
40 Utilisation de numpy et de PIL
41 # -----
42 Utilisation des arrays pour traiter des images
43 no
44 im
45 im Vous avez ouvert l'image Lenna .png
46 nb qui a pour dimensions 220 lignes et 220 colonnes
47 nb
48 Vous avez transformé cette image en noir et blanc
49 #
50 Durée du traitement: 1.9797797203063965 seconde
51 #
52 #
53 #
54
55 st
56
57 im
58 fo
59
60
61
62
63 en
64
65 pr
66 pr
67 pr
68 pr
69
70 # On sauvegarde les images pour pouvoir les afficher
71
72 Image.fromarray(image).save("image_entree.png")
73 Image.fromarray(image_sortie).save("image_sortie.png")
74
75 # -----
76 # Affichage dans tkinter
77 # -----
78

```



Exercice 2.8 Modifier la luminosité

Pour augmenter la luminosité d'une image, il suffit d'ajouter (ou soustraire) à toutes les valeurs un même nombre sans dépasser la valeur maximale qui est de 255 si on ajoute (ou minimale qui est 0 si on soustrait).

On pourra soit le faire avec une condition, soit utiliser la fonction `min`.

On peut aussi écrire en une ligne la modification en utilise la fonction:

```
np.where(condition,valeur_si_vraie,valeur_si_fausse)
```

où la condition écrite sur un tableau `t` de valeurs s'applique sur chaque valeur directement.

Par exemple, `np.where(t<30,0,t-30)` permet de créer un tableau dans lequel toutes les valeurs de `t` inférieures strictement à 30 sont remplacée par 0 et celles au dessus de 30, on soustrait 30. Cela permettrait de réduire la luminosité de 30 si on l'applique à une image.

Exercice 2.9: mixage de deux images

On peut, à partir de deux images, créer un mélange des deux.

Pour cela, on commence par choisir dans quelle proportion on veut les mélanger (par exemple 60% de la première et donc 40% de la seconde).

Ensuite, il suffit de prendre comme valeur, pour chaque couleur de chaque pixel, 60% de la valeur de la première image + 40% de la valeur de la seconde image en veillant à ne pas dépasser 255 et à ne conserver que la valeur entière.

Ainsi, si par exemple le premier pixel de la première image est de couleur (10,20,30) et le premier pixel de la seconde image est de couleur (100,100,100) alors l'image mélangée sera de couleur $(0.6*10 + 0.4*100, 0.6*20 + 0.4*100, 0.6 * 30 + 0.4*100)$ c'est à dire (46, 52, 58).

C'est assez facile puisque multiplier un tableau numpy par un nombre multiplie chacun de ses termes.

Seul détail technique, il faut penser à transformer le résultat final en tableau à valeur entière inférieure à 255 (en appliquant `np.asarray(..., dtype=np.uint8)` au résultat par exemple).

Exercice 2.10: Fond vert

Une image sert de fond, une autre est prise devant un fond vert.

Le principe est un peu le même qu'avec le fondu de deux images, si ce n'est que dans ce cas:

si le pixel de l'image 1 d'avant plan est vert, on le remplace totalement par le pixel équivalent de l'image 2 de fond souhaitée,

sinon on garde sa valeur initiale.

Utilisez chat-vert.jpg et fond-pour-chat-vert.jpg (768x432)

Exercice 2.11 Modifier le contraste

Pour chaque pixel de couleur (r,v,b) , on peut définir son intensité par la moyenne des valeurs des 3 couleurs c'est à dire $i=(r+v+b)/3$. On a ainsi, si on parcourt tous les pixels de l'image, une intensité minimale i_{\min} et une maximale i_{\max} . L'idée, pour augmenter le contraste, est redimensionner la plage des intensités entre 0 et 255 de manière linéaire. Autrement dit, on veut que le pixel d'intensité i_{\min} devienne d'intensité 0 et celui d'intensité i_{\max} devienne d'intensité 255 et entre les deux, on modifie les valeurs de manière linéaire. Pour cela, un pixel d'intensité i à l'origine sera modifié en un pixel d'intensité normalisée $i_n = 255 * (i - i_{\min}) / (i_{\max} - i_{\min})$ en multipliant chaque couleur par cette valeur. On fera bien attention à ce que le résultat soit un entier inférieur à 255.

Exercice 2.12 Redimensionner une image

Supposons que nous voulions redimensionner notre image qui est de dimension a_0 lignes et b_0 colonnes en une nouvelle dimension a_1 lignes et b_1 colonnes.

On peut noter alors les ratios des transformations selon les lignes et les colonnes :
 $\text{ratio_lignes} = a_0/a_1$ et $\text{ratio_colonnes} = b_0/b_1$.

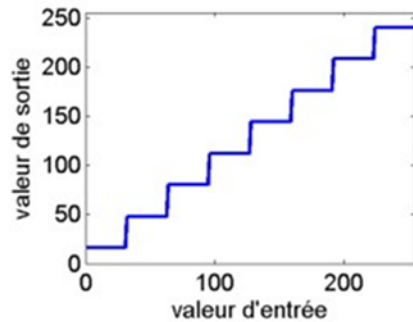
Pour remplir notre nouvelle image, il suffit alors de remplir le pixel situé à la ligne ligne et la colonne col avec les couleurs du pixel de l'image de départ situé à la ligne $\text{int}(\text{ligne} * \text{ratio_lignes})$ et la colonne $\text{int}(\text{col} * \text{ratio_colonnes})$.

Remarque : Si on fait ainsi, lors d'un agrandissement, plusieurs pixels de l'image de départ seront recopiés donnant une impression de gros pixels. Il existe beaucoup de façon d'empêcher ce phénomène en lissant les couleurs par différentes méthodes mais nous ne nous y intéresserons pas ici.

Ecrivez le script suivant pour qu'il affiche une image en sortie de dimension 220 lignes et 220 colonnes de Lenna à partir de celle de 512x512 et comparer avec Lenna220.png.

Exercice 2.13 Re-quantification sur 8 niveaux de gris

Re-quantifier l'image de Lenna ou du singe sur 8 niveau de gris
et sur 8 niveaux en couleur ($3 \times 8 = 512$ couleurs)



8 conditions du type
si $x_1 < \text{valeur} < x_2$, alors valeur = x_3

