Annexe 2
Exemples de mini-projets sur les images numériques

A La stéganographie

1 Préambule

La stéganographie consiste à dissimuler un message dans un autre. C'est une idée ancienne , qui a connu de maintes déclinaisons, telles que les acrostiches de George Sand, mais il s'agit plus précisément ici de voir comment on peut cacher une image dans une autre, en dissolvant en quelque sorte ses pixels dans ceux de l'image d'origine.

2 Pré-requis

Dans l'écriture décimale d'un nombre, les chiffres ont un poids croissant de la droite vers la gauche :

Dans le cas de 79, le 7 représente $7x10^1$ alors que le 9 représente $9x10^0$.

C'est le même principe en binaire : le premier 1 (à gauche) de 11000101 représente $1x2^7$ alors que celui de droite représente $1x2^0$.

Les conversions décimal-binaire et binaire-décimal ne sont pas du tout indispensables ici, mais on peut utiliser la calculatrice si besoin, ou l'interface Python Shell :

```
>>> print 0b11000101
197
```

```
>>> print bin(197)
0b11000101
```

3 Première étape

Si l'on met à zéro les 2, 3, voire 4 bits de poids faible (ceux de droite) des composantes RBV d'un pixel, l'image est-elle toujours de qualité acceptable ? L'idée étant de pouvoir les utiliser pour insérer une autre image.

Mettre à zéro les deux bits de droite consiste à faire un ET logique avec 11111100, c'est à dire 252.

Etc. Dans le cas des 4 bits de droite, on fait un ET avec 11110000, c'est à dire 240.

En utilisant les deux boucles imbriquées vues précédemment, on obtient assez facilement le résultat voulu.

On peut éventuellement en rester là, pour la programmation, selon le niveau du groupe auquel on s'adresse.

Exemple de programme en Python :

```
from PIL import Image
im1 = Image.open("T:\Seville.png")
L,H = im1.size
im2 = Image.new("RGB",(L,H))
for y in range(H):
    for x in range(L):
         p = im1.getpixel((x,y))
         r = p[0]&240              # mise à zéro des 4 bits de droite de la composante rouge
         v = p[1]&240              # idem pour la composante verte
         b = p[2]&240              # idem pour la composante bleue
         im2.putpixel((x,y),(r,v,b))
im2.save("T:\Seville_RAZ_4bits_pfaible.png")
im2.show()
```

4 Seconde étape (la stéganographie proprement dite)

On peut envisager d'insérer une image 2 dans une image 1 de la façon suivante.

Considérons le pixel (i,j) de chaque image (images de même taille, 512 x 512 pixels ou autre).

La composante bleue de ce pixel vaut par exemple **0010**1111 pour l'image 1.

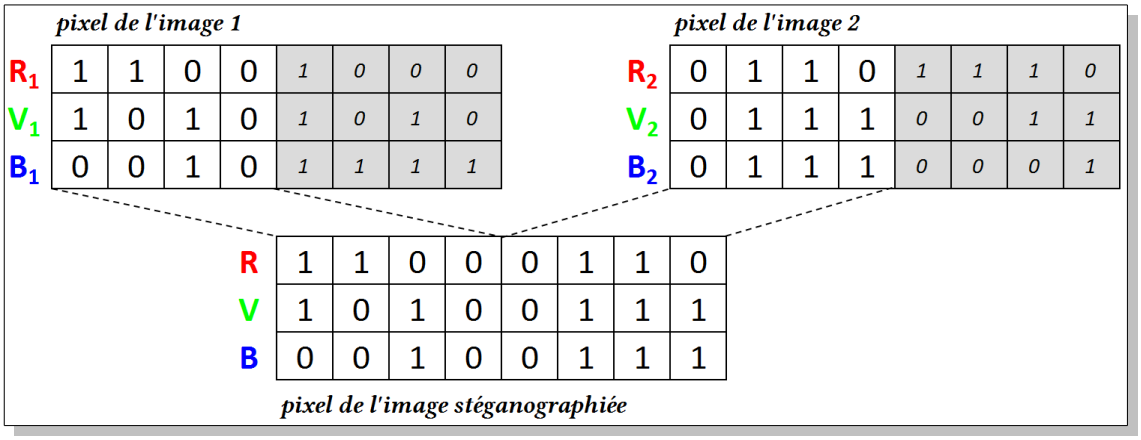
On met à 0 les 4 bits de droite (cf. étape 1) : on obtient **0010**0000.

La composante bleue de ce pixel vaut par exemple **0111**0001 pour l'image 2.

On fait un décalage de 4 bits vers la droite : 0000**0111**. En Python, les 4 bits de gauche sont alors mis à zéro, un ET logique avec 00001111 c'est à dire 15 n'est donc pas nécessaire ensuite.

La composante bleue du pixel (i,j) de l'image stéganographiée est obtenue par un OU entre les deux précédentes : **00100111**.

On fait de même pour les composantes rouge et verte.



L'image 2 étant codée dans les bits de poids faibles, elle devrait ne plus être visible … Est-ce bien sûr ?

Il reste à s'assurer du résultat et à discuter de la pertinence de la démarche : influence du nombre de bits utilisés pour insérer l'image 2.

4, c'est beaucoup, mais avec 2, c'est plus compliqué, et peut-on dans ce cas insérer une image de même taille ?

On peut aussi discuter de la possibilité de détecter une image stéganographiée avec la méthode proposée ici par rapport à l'original de cet image : comparaison des composantes RVB pixel par pixel par exemple, … ce qui peut être testé par programmation.

Exemple de programme en Python :

```
from PIL import Image
im1 = Image.open("T:\Seville.png")
L,H = im1.size
im2 = Image.open("T:\rennes.png")
im3 = Image.new("RGB",(L,H))          # image destination (image sténographiée, « vide » pour
l'instant)
for y in range(H):
  for x in range(L):
      p1 = im1.getpixel((x,y))        # acquisition du pixel (x,y) de l'image 1
      r1 = p1[0]&240                  # et mise à zéro des 4 bits de droite R,V, et B
      v1 = p1[1]&240
      b1 = p1[2]&240
      p2 = im2.getpixel((x,y))        # acquisition du pixel (x,y) de l'image 2
      r2 = p2[0]>>4                   # et décalage de 4 bits vers la droite R,V, et B
      v2 = p2[1]>>4
      b2 = p2[2]>>4
      r = r1|r2                       # insertion du pixel de l'image 2 dans celui de l'image 1
      v = v1|v2
      b = b1|b2
      im3.putpixel((x,y),(r,v,b))     # écriture de l'image sténographiée pixel par pixel
im3.save("T:\Seville_stegano.png")
im3.show()
```

Quelques résultats ...

Image 1

Image 2

Image 1 avec les 4 bits de poids faible R, V, et B, à 0

Image 2 avec les 4 bits de poids faible R, V, et B, à 0

Image 2 cachée dans l'image 1

Image 1 cachée dans l'image 2

Avec l'algorithme rudimentaire proposé ici , on devine l'image cachée sur les zones uniformes (uniquement, mais …).

L'extraction d'une image cachée par ce procédé n'est pas plus compliquée; elle est même plus simple : il s'agit de faire un décalage de 4 bits vers la gauche sur chaque composante R,V,B : les 4 bits de poids faible deviennent ceux de poids fort. En Python, les 4 bits de droite sont alors mis à 0, mais attention, on passe d'un mot de 8 bits à un mot de 12 bits : le décalage allonge la longueur du mot. Il faut donc ignorer les bits de rang 8 à 11 : un ET logique avec 11111111 (& 255) suffit.

B Création et exploitation de l'histogramme d'une image numérique

Afin de simplifier l'étude, on se propose de calculer l'histogramme d'une image en niveaux de gris. Cet histogramme peut servir entre autres au repérage de zones significatives dans l'image (voir paragraphe 6), ou à la reconnaissance optique de caractères, évoquée au paragraphe 7.

L'histogramme représente dans ce cas le nombre N de pixels correspondant à chacun des 256 niveaux de gris (la grandeur n_gris en abscisse est la valeur du niveau de gris, de 0 à 255).

Objectif : on veut obtenir un fichier du type .csv comportant deux colonnes de 256 valeurs (n_gris, et N), dont on pourra ensuite donner une représentation sous forme d'histogramme à l'aide d'OpenOffice Calc. La conversion d'une image couleur en niveaux de gris peut être intégrée au programme à fournir.

A titre de prolongement, on pourra s'interroger sur l'usage d'un tel histogramme, et demander de le fournir sous forme logarithmique par exemple.

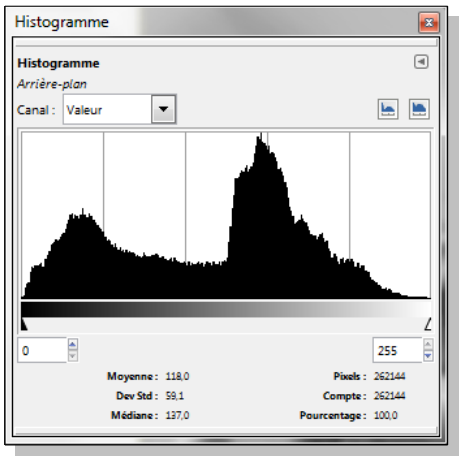
La production directe de l'histogramme sous forme d'image (sans passer par l'étape fichier .csv) peut aussi être intéressante, si le niveau du groupe d'élèves le permet.

1 Obtention de l'histogramme d'une image dans GIMP

Obtention d'une image en niveaux de gris : menu Image > Mode > Niveaux de gris

Obtention de l'histogramme : menu Couleurs > Informations > Histogramme

ou Fenêtres > Fenêtres ancrables > Histogramme



L'histogramme fournit ici plusieurs informations complémentaires, dont le calcul peut être intégré ou non au projet :

- Pixels : 262144 — *image 512 x 512 pixels ici*
- Moyenne : 118 — *moyenne des niveaux de gris sur tous les pixels*
- Médiane : 137
- Déviation standard : 59,1 — *écart-type*

2 Pré-requis

Pré-requis acquis lors des précédentes activités ou projets

Double boucle for pour parcourir tous les pixels d'une image bitmap.

Calcul du niveau de gris à partir des composantes R, V, et B d'un pixel en couleurs.

Pré-requis à acquérir

Structure de type tableau de valeurs pour stocker le nombre de pixels pour chacun des 256 niveaux de gris.

Création et écriture dans un fichier .csv

3 Exemple de programme en Python

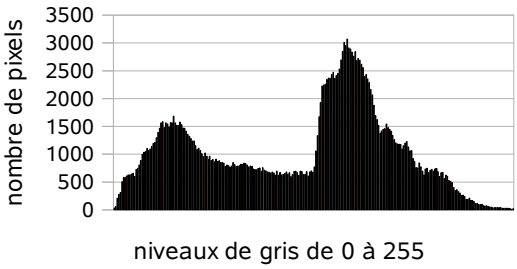
```
from PIL import Image
import csv                                  # pour pouvoir exporter les résultats dans un fichier .csv
im = Image.open("T:\Seville.png")
L,H = im.size
HIST = list()                               # HIST[i] contiendra le nombre de pixels correspondant
                                            # au niveau de gris i  (0 ≤ i ≤ 255)
fichier = open("T:\Seville_histo_niv_gris.csv","wb")
f = csv.writer(fichier)
f.writerow(["niveau_gris","nb_pixels"])     # écriture de la ligne titres du fichier
for i in range(256):
    HIST.append(0)                          # initialisation des HIST[i] à 0
for y in range(H):
    print y
    for x in range(L):
        p = im.getpixel((x,y))
        n_gris = (p[0]+p[1]+p[2])/3         # calcul du niveau de gris de chaque pixel
        HIST[n_gris] = HIST[n_gris]+1       # mise à jour du nombre de pixels
                                            # correspondants
for i in range(256):
    f.writerow([i,HIST[i]])                 # écriture des 256 résultats
fichier.close()
```

À noter que le séparateur utilisé est ici la virgule, donc il faut en tenir compte lorsqu'on ouvre ensuite le fichier .csv dans OpenOffice.org (ou Libreoffice) Calc.

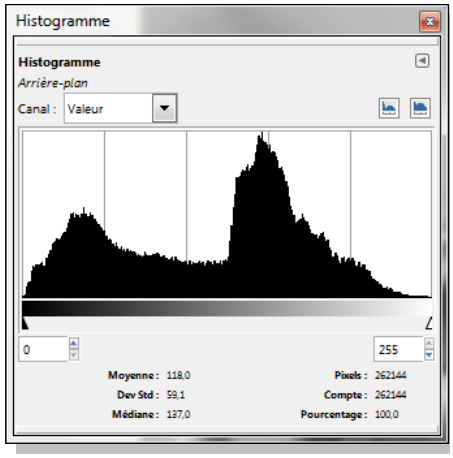
4 Résultats obtenus et comparaison avec ce que donne GIMP

	A	B
1	niveau_gris	nb_pixels
2	0	35
3	1	58
4	2	213
5	3	278
6	4	310
7	5	503
251	...	...
252	249	31
253	250	28
254	251	26
255	252	29
256	253	24
257	254	12
258	255	20
259		
260		262144

Tableau de valeurs (fichier .csv)



Histogramme des niveaux de gris obtenu dans Open Office Calc à partir du fichier .csv généré par le programme en Python



Histogramme donné par GIMP

5 Tracé de l'histogramme

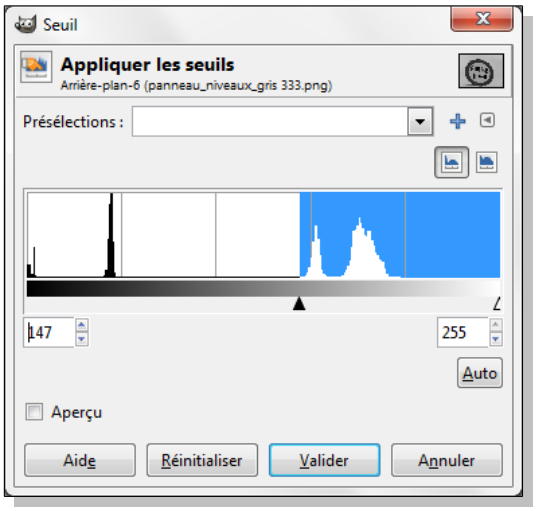
En modifiant légèrement le programme précédent, on peut créer directement dans Python une image de l'histogramme.

```
from PIL import Image, ImageDraw

im = Image.open("T:\Seville.png")
L,H = im.size

histo = Image.new("RGB",(256,140),"white")        # largeur = 256 px car 256 niveaux de gris en
                                                  # abscisse
                                                  # hauteur = 140 px car 1 px ↔ 1 % en ordonnée
                                                  # plus la place pour le titre
dessine = ImageDraw.Draw(histo)
dessine.text((10,10),"histogramme",fill=(0,0,255))  # le texte « histogramme » débute
                                                  # au point de coordonnées (10,10), et sa couleur
                                                  # est ici bleue : (R,V,B) = (0,0,255)
HIST = list()
max = 0                                           # maximum de l'histogramme

for i in range(256):
    HIST.append(0)

for y in range(H):
    print y
    for x in range(L):
        p = im.getpixel((x,y))
        n_gris = (p[0]+p[1]+p[2])/3
        HIST[n_gris] = HIST[n_gris]+1
        if HIST[n_gris]>max:
            max = HIST[n_gris]                    # mise à jour du maximum

for i in range(256):
    HIST[i] = int(100*HIST[i]/max)                # histogramme représenté en % du max
                                                  # voir remarque[1] ci-dessous
    dessine.line([(i,140),(i,140-HIST[i])],width=1,fill=(0,0,0))

histo.show()
histo.save("T:\Seville_histogramme_niveaux_gris.png")
```

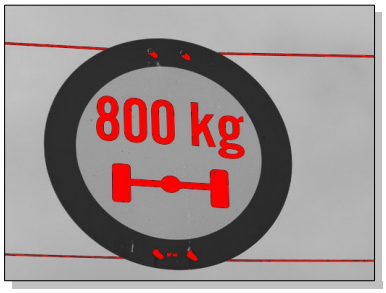
6 Exemple d'exploitation de l'histogramme : définition de seuils puis segmentation de l'image

À partir d'une image dont l'histogramme des niveaux de gris présente des pics bien séparés, on peut noter (ou détecter par programmation) les intervalles de niveaux de gris correspondant à chaque pic, et attribuer une couleur à chaque intervalle.

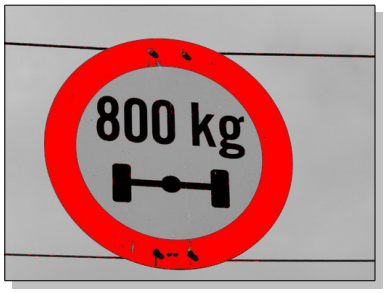
1 dessine.line a pour paramètres les coordonnées (x,y) des deux extrémités du segment à tracer, puis la largeur de ce segment, ici 1 pixel, et enfin la couleur (R,V,B).



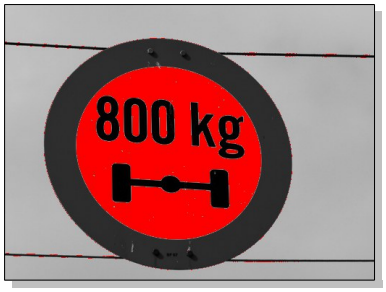
Dans l'exemple ci-dessus, les niveaux vont de 0 à 30 pour le premier pic, de 38 à 49 pour le deuxième, de 147 à 167 pour le troisième, et de 167 à 201 pour le quatrième (menu Couleurs > Seuil … de GIMP; utiliser le curseur pour les afficher). L'algorithme consiste donc à créer une image en niveaux de gris où on lit le niveau de chaque pixel. Suivant l'intervalle dans lequel il se situe, on lui attribue une couleur (il faut créer une nouvelle image destination).



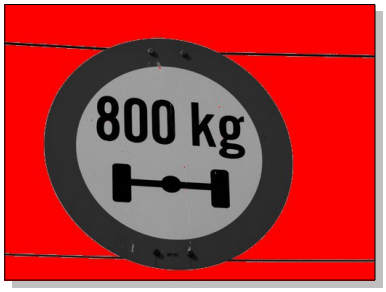
Gris de 0 à 30 remplacé par du rouge



Gris de 38 à 49 remplacé par du rouge

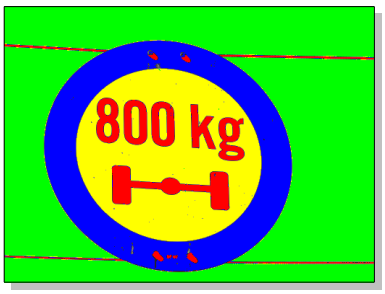


Gris de 147 à 167 remplacé par du rouge



Gris de 167 à 201 remplacé par du rouge

On peut évidemment avec ce procédé attribuer une couleur différente à chaque partie de l'image :



7 Histogramme de projection et exploitation de cet histogramme

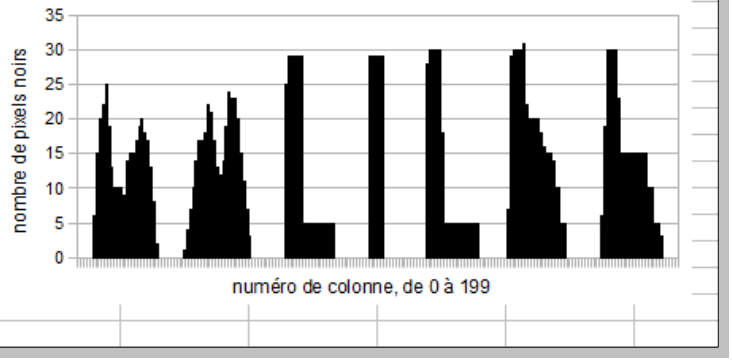
Exemple : on veut pouvoir isoler chaque caractère de l'image de gauche. On transforme tout d'abord cette image en noir et blanc en appliquant un seuil puis on inverse ici le noir et le blanc (image du milieu). Pour obtenir l'histogramme de projection en x (projection sur un axe horizontal), on balaie l'image colonne par colonne, et pour chaque colonne, on compte le nombre de pixels noirs. Avec notre exemple, on obtient l'histogramme ci-dessous à droite :

À partir de là, on connaît la valeur de x qui correspond au début et à la fin de chaque caractère.[2]

En appliquant le même algorithme pour l'histogramme de projection en y, on est ensuite capable de délimiter chaque caractère et d'extraire la sous-image correspondant à chacun d'entre eux. L'intérêt serait ensuite de réaliser une reconnaissance automatique de chaque caractère, à l'aide d'un réseau de neurones par exemple, mais cela devient un peu plus compliqué … En tous cas, la reconnaissance optique des caractères repose partiellement sur ce principe de segmentation. Le cas des images mal orientées (tournées) se traite en faisant tourner progressivement la direction de projection jusqu'à obtenir une bonne séparation[3].

Pour l'implémentation en Python, on peut choisir comme pour l'histogramme d'une image en niveaux de gris, de tracer directement l'histogramme (le programme produit une image, voir ci-dessus), ou bien d'écrire les résultats dans un fichier .csv exploitable dans OpenOffice Calc :

	A	B	C	D	E	F	G	H	I
1	colonne	nb_pixels							
2	0	0							
3	1	0							
4	2	0							
5	3	0							
6	4	0							
7	5	6							
8	6	15							
9	7	20							
10	8	22							
11	9	25							
12	10	19							
13	11	13							
14	12	10							

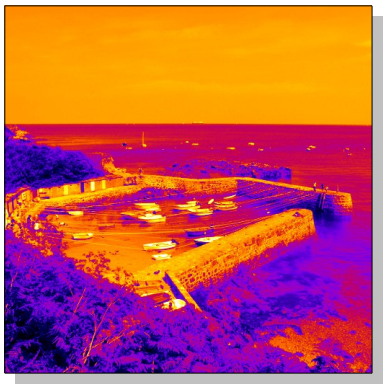


Remarque : l'image « Galilée » utilisée ici a une largeur de 200 pixels, les colonnes sont donc numérotées de 0 à 199.

C Image en pseudo-couleurs

À la différence d'une image en fausses couleurs, où on associe par exemple à trois canaux de luminance enregistrés dans ou hors du domaine visible, le rouge, le vert, et le bleu, une image en pseudo-couleurs est obtenue à partir d'un seul canal de luminance. C'est le cas d'un certain nombre de caméras infrarouge.

2 Curieusement, l'histogramme par projection verticale permet de discriminer notablement certaines lettres …
3 C'est-à-dire, un histogramme le plus « contrasté » possible.



Partant d'une image en niveaux de gris, à chaque niveau de luminance (de 0 à 255 a priori), on associe un triplet (R,V,B) à l'aide d'une table de correspondance appelée look-up table (LUT). La correspondance peut également être établie par calcul, mais une table évite de refaire les calculs pour chaque pixel.

L'intérêt des pseudo-couleurs est d'aider l'œil (humain) à détecter rapidement des zones intéressantes dans l'image.

Plusieurs exemples de LUT se trouvent à cette adresse :
http://www.portanum.com/dossiersTechniques/plugin1.html

En cliquant sur le lien « télécharger le fichier » du paragraphe « Fausses couleurs » mais qui concerne bien les pseudo-couleurs, on obtient plusieurs fichiers texte après décompression.

Si on ouvre le fichier LUT_fire par exemple, on obtient ceci :

Index	Red	Green	Blue
0	0	0	0
1	0	0	7
2	0	0	15
3	0	0	22
4	0	0	30
5	0	0	38
6	0	0	45
7	0	0	53

.....

95	193	0	96
96	195	0	93
97	196	1	89
98	198	3	85
99	199	5	82
100	201	7	78
101	202	8	74

.....

L'index varie de 0 à 255, il représente le niveau de luminance, et les valeurs Red, Green, Blue varient également dans cet intervalle. Il faut donc pouvoir lire et exploiter le fichier texte.

En Python, le bloc suivant permet de lire le fichier ligne par ligne et de stocker chaque ligne dans une liste :

```
tableau = list()                    # ou tableau = []
f = open("T:\LUT_fire.txt","r")     # ouverture du fichier en lecture : r pour read
for li in f:                        # pour chaque ligne du fichier
 s = li.strip("\n\r")               # on supprime les caractères de fin de ligne
 l = s.split("\t")                  # pour chaque ligne, on supprime la tabulation séparant les colonnes
 tableau.append(l)                  # ajout de cette ligne du fichier à la liste
f.close()                           # fermeture du fichier
```

Le niveau de bleu correspondant à une luminance de niveau 4 (voir extrait de la LUT ci-dessus) s'obtient ainsi :

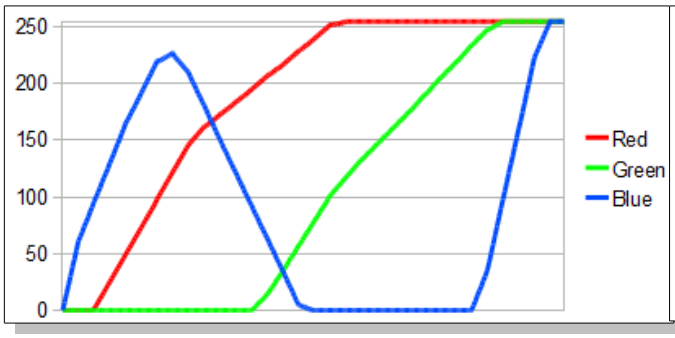
```
b = tableau[5][3]
```

car les lignes et les colonnes sont numérotées à partir de 0, et la ligne titres (Index, …) doit être prise en compte, ou alors, il faut préalablement la supprimer dans le fichier.

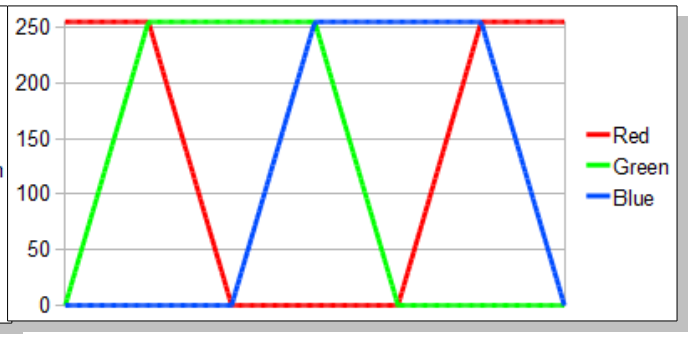
Un exercice préalable peut être proposé aux élèves pour comprendre comment accéder aux données de la liste.

À titre de prolongement, on peut envisager de créer une nouvelle LUT en commençant par examiner comment

varient les niveaux R,V,B en fonction de la luminance (0 à 255) dans les fichiers téléchargés :



LUT « fire »



LUT « spectrum »

Des équations de droites suffisent donc pour établir une correspondance entre niveau de gris et rouge, vert, ou bleu.

Dans la LUT spectrum par exemple, le bleu varie linéairement de 0 à 255 pour les niveaux de gris compris entre 85 et 128, donc b = 255*(gris-85)/(128-85) sur cet intervalle.

Et ainsi de suite …

Crédits photographiques

Les photographies sont la propriété de l'auteur.