

# TP7 – Traitement d'images

## A – Sans bibliothèques d'images

Afin de réaliser quelques traitement d'images on utilisera le logiciel Gimp que vous pouvez télécharger sur ce lien : <https://www.gimp.org/downloads/> et d'un code nommé « TP\_lecture\_écriture.ipynb » (alias BC) (<https://pcjoffre.fr/informatique-en-pcsi2/>) qui permettra de lire ou d'enregistrer des images sous python aux formats ppm, pbm ou pgm.

### Exercice 1 : Inversion noir et blanc

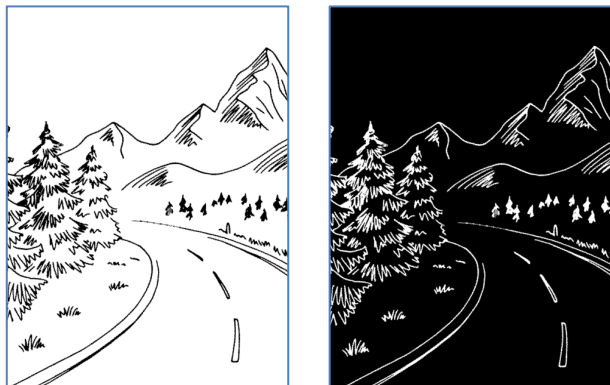
Pour commencer choisissez une image en noir et blanc de votre choix.

- Lancer le logiciel GIMP pour la mettre au bon format.
  - Ouvrir l'image voulue
  - La redimensionner si elle est trop grande : « image »  $\Rightarrow$  « Echelle et taille de l'image ».
  - Puis « Fichier »  $\Rightarrow$  « Exporter sous » afin de la mettre au format .pbm. (On choisira le format « ASCII »)
- Sinon vous pouvez télécharger les image0.pbm et image1.pbm sur le site.

- a) Recopier le corps du code BC dans votre programme puis écrire une fonction « inverser\_nb » qui inverse les teintes noires en blanc et vice-versa.

On appellera l'image et la fonction à l'aide de l'algorithme suivant puis sous GIMP on lira l'image.

```
mat=lire_fichier_pbm('image0.pbm')
ai=inverser_nb(mat)
ecrire_fichier_pbm('image0_inv.pbm',ai)
```



Avant/Après par la fonction « inverser\_nb ».

- b) Quel problème rencontre-t-on avec l'image1. Pourquoi ?

### Exercice 2 : Symétrie

On envisage une symétrie axiale avec un axe vertical. Il convient donc d'échanger chaque pixel d'indice  $(i,j)$  avec le pixel d'indice  $(i,p-1-j)$  pour  $j$  allant de 0 à  $p/2$  où  $p$  représente le nombre de colonnes dans l'image.

- a) Ecrire une fonction `sym_vert` qui prend en paramètre une matrice représentant une image au format pbm et modifie la matrice afin d'effectuer une symétrie d'axe vertical.
- b) Ecrire une fonction `sym_hor` qui fait de même mais en tant que symétrie horizontale. Il faudra inverser les lignes  $i$  et  $(haut-1-i)$  où `haut` représente le nombre de lignes de l'image.

### Exercice 3 : Réduction/Agrandissement

On dispose d'une image de taille  $(n,p)$  et on souhaite la réduire. Nous supposons que la réduction consiste à diviser la longueur et la largeur par un nombre entier  $d$ . C'est le cas le plus simple et on envisage de garder une ligne sur  $d$  lignes et une colonne sur  $d$  colonnes.

- a) Ecrire une fonction `reduction(img,d)` qui permet de réduire l'image d'un facteur  $d$ . Tester le programme avec Gimp et un facteur  $d = 2$ .
- b) Sur le même principe, réaliser un algorithme d'agrandissement d'image. Il suffira de copier  $d$  fois chaque colonne,

et d fois chaque ligne.

#### Exercice 4 : Rotation (image carrée à prévoir)

##### a) Forme récursive

Pour une image représentée par une matrice de pixel, une rotation présente de nombreuses difficultés. Celles-ci proviennent principalement du fait qu'un pixel a des coordonnées entières. La suite se limite donc à des images carrées avec une rotation d'un quart de tour. Une rotation d'un quart de tour consiste à permuter les pixels des quatre quadrants.

```
def rotation_recursive(img, x, y, n):
    """
    Rotation récursive par blocs (type quadrants)
    img : image (matrice)
    x, y : coin supérieur gauche du bloc
    n : taille du bloc (doit être carré)
    """
    #On découpe le quadrant ainsi
    #A B
    #C D
    if n <= 1:
        return
    n2 = n // 2
    # échange des 4 quadrants
    for i in range(n2):
        for j in range(n2):
            temp = img[
                x + i][y + j] # A (Vu les variations de i et j on décrit le quadrant A)
            img[x + i][y + j] = img[
                x + i + n2][y + j] # C -> A
            img[x + i + n2][y + j] = img[
                x + i][y + j + n2] # D -> C
            img[x + i][y + j + n2] = img[
                x + i + n2][y + j] # B -> D
            img[x + i + n2][y + j + n2] = temp # A -> B
    # récursion sur les 4 sous-blocs
    rotation_recursive(img, x, y, n2) #Récursion sur A
    rotation_recursive(img, x, y + n2, n2) #Récursion sur C
    rotation_recursive(img, x + n2, y, n2) #Récursion sur B
    rotation_recursive(img, x + n2, y + n2, n2) #Récursion sur D
```

Voici un exemple d'algorithme où on a découpé les quadrants en A,B (ligne1) et C,D (ligne2). Complétez l'algorithme pour que celui-ci fasse une rotation des quatre quadrants. Vérifiez-le sur une image carrée.

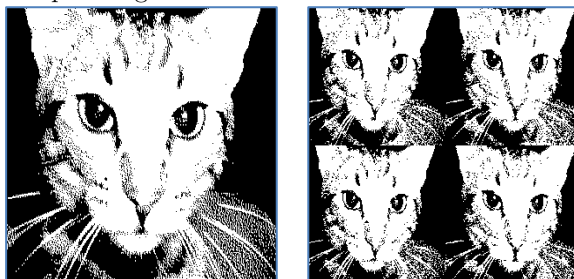
##### b) Jeu d'indices

En manipulant bien les indices, on peut aussi faire un programme non récursif de rotation. Ecrire une fonction *rotation(img)* qui remplit cette tâche.

#### Exercice 5 : Le photomaton

On considère une image carrée de côté n pixels. On désire conserver la même taille mais on veut 4 mini-images à la place.

Écrire une fonction non récursive photomaton qui remplit ce rôle. On n'a pas perdu de pixels dans l'image finale mais juste récupéré un pixel sur 4 dans chaque image.



## B – Avec bibliothèques d'images

On désire à l'aide de python convertir une image couleur en niveau de gris à l'aide de plusieurs algorithmes. Dans cet exercice on va utiliser la bibliothèque « PIL » dont les bases sont données ci-dessous.

Le module PIL fournit le sous-module Image :

- `img1=Image.open('chatc.jpg')` : permet d'ouvrir et d'identifier le fichier image donné (« .jpg », « .png »,...)
- `Image.size` : permet d'extraire la taille de l'image (largeur, hauteur)
- `Image.new(mode,size)` : alloue une image matricielle, de type couleur si `mode = 'RGB'`. Sa taille est donnée par `size =(n,p)` où n est la largeur et p la hauteur de l'image.
- `img1.getpixel((j,i))` : permet de récupérer les valeurs des canaux : r,v,b du pixel de coordonnées (j,i)

- `img_gris1.putpixel((j,i),(r,v,b))` : permet de remplir un pixel de coordonnées (j,i) avec la synthèse « rvb » ou « rgb ». (Le nom `img_gris1` fait référence au programme « gris1 » demandé.
- `a.save('name.jpg')` : permet de sauvegarder l'image matricielle `a` dans le fichier « name.jpg »
- `a.show('name.jpg')` : permet de visualiser l'image dans gimp, paint,...(la première fois il vous laisse choisir votre logiciel préféré.

## Exercice 6 : Utilisation de PIL

a) Écrire une fonction `gris1(img)` qui prend en paramètre une image au format « .jpg ». Cette fonction crée une nouvelle image en niveau de gris, la sauvegarde et l'affiche. Pour calculer l'intensité de gris on calcule la moyenne « m » des intensités de rouge, de vert et de bleu. Un pixel (r,v,b) devient un pixel (m,m,m). Voici une partie du code de la fonction « gris1 », recopier le code puis compléter les lignes manquantes. On utilisera des images jpg.

```
from PIL import Image

def gris1(img):
    img1=Image.open(img)
    n,p=img1.size
    img1_gris1=Image.new("RGB",(n, p))
    for i in range(p):
        for j in range(n):

            img1_gris1.save('chatg.jpg')
            img1_gris1.show()

gris1('chatc.jpg')
```

- b) Écrire une fonction `gris2(img)` qui prend en paramètre une image au format « .jpg ». Cette fonction crée une nouvelle image en niveau de gris, la sauvegarde et l'affiche. Pour calculer l'intensité de gris on calcule la moyenne « m » par :

$$m = 0,2126 \times r + 0,7152 \times v + 0,0722 \times b$$

Conclure brièvement sur les différences entre les deux images obtenues.

- c) Écrire une fonction `nb(img,seuil)` qui prend en paramètre une image au format « .jpg » et un nombre entier naturel nommée seuil. La fonction crée une image en noir et blanc, la sauvegarde et l'affiche. Pour cela reprendre la fonction « gris2 » et si la valeur est supérieure à « seuil », le pixel devient (255,255,255), sinon (0,0,0).

## Exercice 7 : Utilisation de matplotlib.image

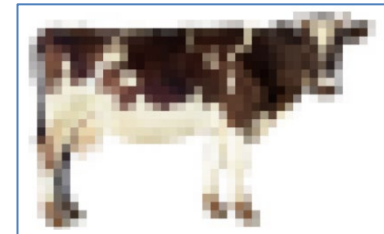
- a) On va utiliser cette bibliothèque pour reproduire notre réduction d'image. Pour cela complétez le programme suivant puis testez le sur une image jpg.

```
1 import numpy as np
2 import matplotlib.image as mpimg
3 import matplotlib.pyplot as plt
4
5 def reduction_mpimg(e,s,echelle):
6     """
7     Fonction qui réduit la taille d'une image.
8     Parameters :
9     - e : chemin de l'image originale
10    - s : chemin où sauvegarder l'image réduite
11    - echelle : facteur de réduction (0.5 = moitié, 0.25 = 1/4, etc.)
12    """
13    # Charger l'image depuis un fichier, img est un tableau numpy (pixels de l'image)
14    img = mpimg.imread(e)
15    # Réduction de l'image
16    # On garde 1 pixel sur N (approximation simple)
17    # Exemple : echelle=0.5 → on garde 1 pixel sur 2
18    mat = img[
19        ,
20    ]
21    # Sauvegarder l'image réduite dans un fichier
22    mpimg.imsave(s,mat)
23    # On retourne l'image réduite (tableau numpy)
24    return mat
25 s=reduction_mpimg('chatc.jpg','chatg_red.jpg',0.2)
26 plt.imshow(s)
27 plt.axis('off')
28 plt.show()
```

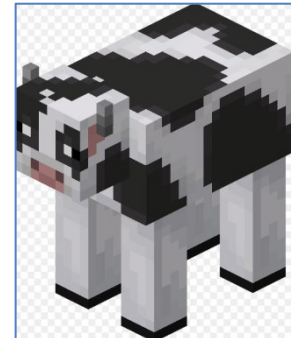
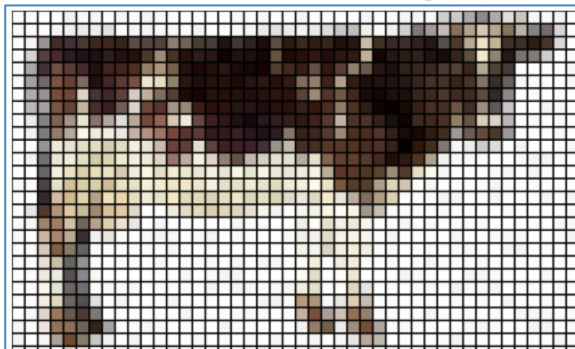
- b) On va maintenant créer une fonction *pixellisation(image,n)* qui va faire la moyenne des couleurs sur les blocs alentours. Par exemple si n vaut trois, on aura une moyenne des couleurs faites sur un bloc 3 par 3. Et on remplit ce bloc par la moyenne des couleurs. Pour cela compléter le code suivant.

```

1 import numpy as np
2 import matplotlib.image as mpimg
3 import matplotlib.pyplot as plt
4
5 def pixellisation(image,n):
6     """
7     Transforme une image en version pixellisée.
8     """
9     image_lue = mpimg.imread(image)
10    img = np.array(image_lue) # Conversion en numpy
11    l, c = img.shape[:2] #On récupère la taille de l'image
12    out = np.zeros_like(img) # On prépare l'image de sortie (copie) - exactement de même taille
13    for i in range(0, l, n): # on parcourt les blocs n x n
14        for j in range(0, c, n):
15            ->
16            ->
17            ->|
18
19    return out
20
21 # pixellisation
22 img_pix = pixellisation('vache.jpg', 15)
23 image_lue = mpimg.imread('vache.jpg')
24 # affichage
25 plt.figure(1)
26 plt.axis("off")
27 plt.imshow(image_lue)
28 plt.figure(2)
29 plt.axis("off")
30 plt.imshow(img_pix)
31 plt.show()
    
```



- c) Améliorer votre programme pour essayer de vous rapprocher au maximum d'un effet Minecraft. Pour cela on créera la fonction *def minecraft(image,n,noise)*: qui rajoutera à la moyenne une valeur entière aléatoire. De plus on rajoutera des cadres noirs autour de chaque bloc.



- d) Essayer de vous rapprocher au mieux du résultat de droite en utilisant des palettes de couleurs, une image de départ isométrique, un lissage par bloc...