TP6 – Traitement 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_ecriture.ipynb » (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.

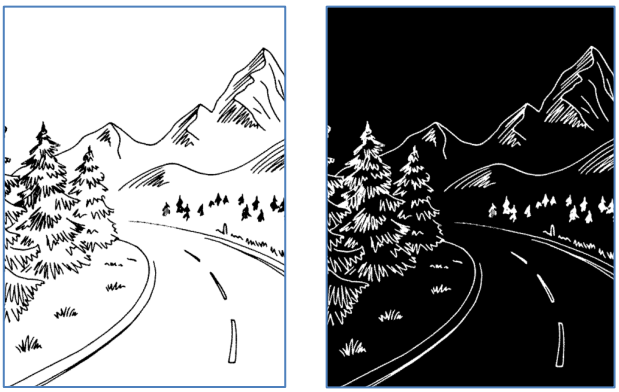
- Lancer le logiciel GIMP pour la mettre au bon format.
- Ouvrir l'image voulue
- La redimensionner si elle est trop grande : « image » ⇒ « Echelle et taille de l'image ».
- Puis « Fichier » ⇒ « Exporter sous » afin de la mettre au format .bpm. (On choisira le format « ASCII »)

Sinon vous pouvez télécharger les image0.bpm et image1.bpm 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-ton 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) ou 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) Voici un algorithme de réduction d'image. On a testé celui-ci pour la valeur 2. Tester le programme pour plusieurs valeurs et vérifier à l'aide de GIMP que la fonction demandée est bien réalisée. Rajouter des annotations au programme ainsi qu'une spécification.

```
def reduction(img, d):
    larg, haut = len(img), len(img[0])
    mat1 = [[0 for j in range(haut)] for i in range(larg//d + 1)]
    for i in range(larg):
        if i % d == 0:
            for j in range(haut):
                mat1[i//d][j] = img[i][j]
    mat2 = [[0 for j in range(haut//d + 1)] for i in range(larg//d + 1)]
    for i in range(larg//d + 1):
        for j in range(haut):
            if j % d == 0:
                mat2[i][j//d] = mat1[i][j]
    return mat2

mat4=lire_fichier_pbm('image0.pbm')
red=reduction(mat4,2)
ecrire_fichier_pbm('image0_red.pbm',red)
```

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

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. On note p1 le pixel d'indice (i, j), p2 le pixel d'indice (n−1− j, i), p3 le pixel d'indice (n−1−i, n−1−j), p4 le pixel d'indice (j, n−1−i). On effectue pour i allant de 0 à n/2 et j allant de 0 à n/2, les permutations (p1, p2, p3, p4) ⇒ (p2, p3, p4, p1).

Une image carrée est découpée en quatre carrés de même taille. Chaque carré est déplacé, (c1, c2, c3, c4) ⇒ (c2, c3, c4, c1) et on effectue la rotation d'un quart de tour de chaque carré par un appel récursif.

- Compléter et annoter la fonction rotation qui suit :

```
def rotation(img, x, y, n):
    if n > 1:
        n = n // 2
        for i in range(x, x+n):
            for j in range(y, y+n):
                temp = img[i][j]
                img[i][j] = img[i][n + j]
                img[i][n + j] = img[n + i][n + j]
                img[n + i][n + j] = img[n + i][j]
                img[n + i][j] = temp
        rotation(              )
        rotation(              )
        rotation(              )
        rotation(              )
    return(img)

mat6=lire_fichier_pbm('image0.pbm')
rot=rotation(mat6,0,0,400)
ecrire_fichier_pbm('image0_rot.pbm',rot)
```

- A l'aide de GIMP et de votre programme. Créez une image carré simpliste et montrer le rôle de x et y.

Exercice 5 : Le photomaton

On considère une image carrée de côté n pixels. La transformation à appliquer pour créer une nouvelle image est décrite pour chaque pixel repéré par un couple d'indices (i, j) (les indices vont de 0 à n − 1) :

- Si i et j sont pairs, le pixel est envoyé sur le pixel d'indice $\left(\frac{i}{2}, \frac{j}{2}\right)$;
- Si i est impair et j pair, le pixel est envoyé sur le pixel d'indice $\left(\frac{n+i}{2}, \frac{j}{2}\right)$;
- Si i est pair j impair, le pixel est envoyé sur le pixel d'indice $\left(\frac{i}{2}, \frac{n+j}{2}\right)$;
- Si i et j sont impairs, le pixel est envoyé sur le pixel d'indice $\left(\frac{n+i}{2}, \frac{n+j}{2}\right)$.

Écrire une fonction récursive photomaton qui prend en paramètres une matrice carrée et un entier correspondant au nombre de répétitions de cette transformation. Tester la fonction avec 3 répétitions par exemple ce qui donnera 64 petites images.