INFORMATIQUE - TP n°5:
Images matricielles

Introduction

Une image matricielle est une image constituée d'une matrice de pixels. En règle générale, une image matricielle est constituée d'un entête donnant diverses informations (taille de la matrice, nombre de couleurs, date de création,...) et d'une matrice codant chacun de ses pixels. Des techniques de compression de données sont ensuite utilisées pour réduire la taille de l'image. Cette compression peut être effectuée avec ou sans perte.

À la différence des images matricielles, une image vectorielle est composée d'objets géométriques (segments, courbes de Bézier, polygones, ...), définis chacun par différents attributs (forme, position, couleur, ...). Les images vectorielles sont particulièrement adaptée au changement d'échelle. Par contre une image matricielle est plus adaptée à la photographie ou aux écrans.

Nous ne nous intéresserons qu'aux images matricielles les plus simples.
Le Portable PixMap file format (PPM), le Portable GrayMap file format (PGM) et le Portable BitMap file format (PBM) permettent de coder des images en noir et blanc, des images en niveau de gris ou des images en couleurs. Ces trois formats sont minimalistes dans le sens ou peu d'informations complémentaires se trouvent dans l'entête et aucune compression n'est effectuée. Les fichiers dans un des trois format précédent contiennent dans l'ordre les informations suivantes.

- le nombre magique du format (deux octets) : il indique le type de format (PBM, PGM, ou PPM) et la variante (binaire ou ASCII) ;
- la largeur de l'image (nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII) ;
- la hauteur de l'image (idem) ;
- les pixels de l'image : en partant du haut à gauche de l'image et en codant les pixels ligne par ligne.

Pour visionner les images codées dans les trois formats précédents on pourra utiliser un visionneur d'image supportant ces formats. Si aucun ne se trouve sur votre ordinateur, utiliser un convertisseur en ligne pour transformer les images en des images JPG.

1 Images en niveau de gris

Récupérer l'image `image_gris.pgm` et l'ouvrir avec un éditeur de texte (si votre éditeur de texte n'affiche pas plusieurs ligne, changer d'éditeur).

Les deux premières lignes indiquent que l'image est en niveau de gris (grâce au nombre magique P2), qu'elle mesure 22 colonnes par 7 lignes. Finalement le nombre 15 indique la valeur maximale utilisée pour coder les niveaux de gris (ainsi un pixel blanc sera représenté par le nombre 15).

Nous allons représenter la matrice des pixels d'une telle image par une liste de n listes de longueur m, où n et m sont des entiers strictement positifs. Par exemple la matrice

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \\ 0 & 1 & 3 \\ 0 & 4 & 5 \end{pmatrix}$$

sera codée par $[\,[1,2,0]\,,\,[1,3,0]\,,\,[0,1,3]\,,\,[0,4,5]\,]$.

Question 1. Commençons par implémenter quelques fonctions permettant de travailler en python sur des tableaux de nombres.

1. Créer la fonction `affichemat(A)` qui affiche dans la console la matrice A « ligne par ligne ». Par exemple avec la matrice A précédente,

```
A = [ [1,2,0] , [1,3,0] , [0,1,3] , [0,4,5] ]
affichemat(A)
>>> [1,2,0]
>>> [1,3,0]
>>> [0,1,3]
>>> [0,4,5]
```

2. Implémenter la fonction `creemat(n,m)` qui crée la matrice nulle de taille $n \times m$. *L'image correspondante est donc noire.*
3. Qu'est censé faire le code suivant ?

```
A=creemat(4,6)
A[0][0]=1
affichemat(A)
```

   Vérifier ainsi que votre fonction `creemat(n,m)` marche correctement, sinon reprendre la question précédente.
4. Implémenter la fonction `recopiemat(A)` qui prend en arguments une matrice carrée et qui crée une nouvelle matrice qui en est la copie. On devra notamment tester la fonction avec le code

```
A=[[1,1,1] , [1,1,1], [1,1,1]]
B=recopiemat(A)
B[1][1]=42
affichemat(A) # A n'est pas censée avoir été modifiée
```

Question 2. Compléter le code suivant permettant de récupérer la matrice des pixels de l'image `nom` et la valeur maximale utilisée pour coder les niveaux de gris.

```
def recuperation_image_gris(nom):
    file = open(nom,'r')
    file.readline() # sauter la première ligne

    ligne= file.readline() # lecture de la seconde ligne
    ligne=À COMPLETER # séparer les données au niveau de l'espace
    nbligne = À COMPLETER # nombre de lignes de l'image
    nbcol = À COMPLETER   # nombre de colonnes de l'image

    valeurmax  = int(file.readline()) # valeur maximale pour le niveaux de gris

    for ligne in file.readlines(): # pour toutes les lignes suivantes du fichier
        À COMPLETER

    file.close()
    return matrice,valeurmax
```

Question 3. Le code suivant permet de créer un fichier appelé `nom` (ou de le remplacer si celui-ci existe déjà) qui sera une image en niveau de gris dont le tableau de pixels est celui fournit par `matrice` et dont la valeur maximale utilisée pour coder les niveaux de gris est `valeurmax`.

```
def creation_image_gris(nom,valeurmax,matrice):
    file = open(nom,'w')
    file.write("P2\n")
    n = len(matrice)
    m = len(matrice[0])
    file.write("%d %d\n" %(m,n)) # ecrire les dimensions
    file.write("%d\n" %valeurmax)

    for ligne in matrice:
        for k in ligne:
            file.write(str(k))
            file.write(" ")
        file.write("\n")

    file.close()
```

Comprendre et commenter le code

Pour afficher l'image définie par une matrice on peut soit utiliser le code précédent et créer un fichier .pgm ou on peut utiliser matplotlib :

```
import matplotlib.pyplot as plt
plt.imshow(A,cmap=plt.get_cmap("gray")) # A est la matrice des pixels
plt.show()
```

Avec cette dernière méthode, vous remarquerez que l'échelle des pixels s'adapte automatiquement aux valeurs minimales et maximales présentent dans A. On peut utiliser les paramètres optionnels `vmin` et `vmax` si il faut préciser la plage des valeurs utilisées.

Question 4. Écrire la fonction `inverser_gris(matrice,valeurmax)` prennant en entrée une image et renvoyant une nouvelle image où les « couleurs » auront été inversées. Par exemple un pixel blanc (codé par valeurmax) doit devenir noir (codé par 0).

Question 5. Écrire la fonction `moyenne_couleur(matrice)` renvoyant la moyenne de la valeur des pixels de l'image. Cette valeur s'appelle la luminance de l'image.

Question 6.

1. Écrire la fonction `symetrie_verticale_gris(matrice)` réalisant une symétrie verticale de l'image.
2. Écrire la fonction `symetrie_horizontale_gris(matrice)` réalisant une symétrie horizontale de l'image.

2 Images en couleurs

Récupérer l'image `image_couleur.ppm` et l'ouvrir avec un éditeur de texte.

Les deux premières lignes indiquent que l'image est en couleurs (grâce au nombre magique P3), qu'elle mesure 22 colonnes par 7 lignes. Finalement, le nombre 255 indique la valeur maximale utilisée pour coder les couleurs (grâce à trois nombres codant le rouge, le vert, le bleu).

On représentera une image en python grâce à une matrice de pixels où chaque pixel sera une liste de trois valeurs représentant le niveau de rouge, vert, bleu du pixel. Une matrice étant une liste de liste, une image seradonc une liste de n listes de longueur m contenant des listes de 3 entiers.

Question 7.

1. Écrire la fonction `recuperation_image(nom)` permettant de récupérer les valeurs des pixels à partir d'une image.
2. Écrire la fonction `creation_image` permettant de créer un fichier dont le nom est donné en paramètre de la fonction et qui est une image en couleurs.
3. Écrire une fonction permettant de créer une « image » noire de taille donnée. Il faut donc créer une matrice dont les éléments sont `[0,0,0]`.
4. Écrire une fonction permettant de copier une « image ».

Question 8. Pour utiliser matplotlib et afficher directement l'image avec python, il faut transformer les valeurs des pixels en nombre flottant entre 0 et 1. Écrire la fonction `convertir(A,N)` prenant en paramètre une matrice A dont les éléments sont des listes à 3 éléments d'entiers entre 0 et N et renvoyant une matrice dont les éléments sont des listes à 3 éléments d'entiers entre 0 et 1 en utilisant la conversion $x \mapsto \dfrac{x}{N}$.

Pour afficher l'image A où les couleurs sont codées entre 0 et 255 on pourra par exemple faire

```
import matplotlib.pyplot as plt
B=convertir(A,255)
plt.imshow(B)
plt.show()
```

Question 9.

1. Écrire des fonctions `diminuer_resolution_1` et `diminuer_resolution_2` permettant de diviser par deux la résolution : étant donnée une « image » elle renvoie une « image » contenant deux fois moins de pixels dans chaque direction (i.e. le nombre de lignes et de colonnes est divisé par 2). Deux méthodes sont à programmer et comparer :
   (a) Supprimer des pixels en ne travaillant qu'une ligne sur deux et qu'une colonne sur deux. Par exemple on peut ne récupérer que les pixels dont l'indice des lignes et des colonnes est pair.
   (b) Remplacer les carrés de 4 pixels par leur valeur moyenne.
2. Écrire la fonction `augmenter_resolution` permettant de multiplier par deux la résolution : étant donnée une « image » elle renvoie une « image » contenant deux fois plus de pixels dans chaque direction. Pour cela on remplacera un pixel par un carré de 4 pixels de même valeur.

Question 10. Écrire la fonction `lissage` qui permet de lisser une image. Pour cela, on remplace les valeurs des couleurs d'un pixel par la moyenne des valeurs de cette couleur de ce pixel et des 8 huit pixels l'avoisinant. *Attention aux bords de l'image.* Cette opération permet également de flouter une image. En particulier on peut l'utiliser à la fin de la fonction `augmenter_resolution` pour améliorer le résultat.

Question 11. *(bonus)* En pratique, pour lisser une image, on obtient de meilleur résultat en utilisant la médiane au lieu de la moyenne. On rappelle que dans une liste de $2n+1$ valeurs rangées dans l'ordre croissant, la médiane est $n+1$-ième valeur (si la liste possède $2n$ valeurs alors la médiale est la moyenne des n-ième et $n+1$-ième valeurs). Écrire la fonction `lissage2` utilisant la médiane.

La recherche de contours est une opération difficile. Nous ne présentons qu'un algorithme simple mais des algorithmes plus efficaces (à la fois en qualité et en rapidité) existent. De plus, dans le cadre de ce TP, l'étape finale est laissée à l'œil humain car elle est moins évidente.

Tout d'abord, définir ce qu'est un contour n'est pas évident. Nous nous limiterons aux contours produits par de brusques changements de couleurs. L'idée est de calculer le gradient des couleurs (i.e. la différence de couleur existant entre deux pixels) et de trouver la localisation des extremum locaux de la norme du gradient (i.e. quand la différence de couleur est importante).

On considère le pixel (i,j)

	$(i,j+1)$	
$(i-1,j)$	(i,j)	$(i+1,j)$
	$(i,j-1)$	

Au point (i,j), le gradient de couleur suivant x noté $G_x(i,j)$ et celui suivant y noté $G_y(i,j)$ sont donnés par les formules

$$G_x(i,j) = (i+1,j) - (i-1,j), \qquad G_y(i,j) = (i,j+1) - (i,j-1)$$

où il faut évidement remplacer les coordonnées des pixels par la valeur des pixels et répéter cela pour les trois couleurs.

Pour chaque couleur, il faut alors calculer la norme du gradient qui est donnée par la formule $\sqrt{G_x(i,j)^2 + G_y(i,j)^2}$. On prend alors le maximum des normes des trois gradients de couleurs au pixel (i, j).

L'intérêt de ce qui précède est que si les pixels voisins de (i, j) ont couleurs très proches (les valeurs sont très proches) alors la norme du gradient sera proche de 0. Au contraire si les pixels sont très différents, la norme du gradient sera élevée. Un pixel se trouvant sur le bord d'un contour a de forte chance d'avoir des pixels voisins de couleurs éloignées et donc la norme du gradient en ce pixel sera élevée.

Question 12. Écrire la fonction `gradient` renvoyant la matrice constituée des maximum des normes des gradients

La matrice ainsi obtenue peut être interprétée comme une image en niveau de gris.

- Les pixels noirs de cette image proviennent de pixels où la norme du gradient de couleur est peu importante.
- Les pixels blancs de cette image proviennent de pixels où la norme du gradient de couleur est très importante et donc des contours.

Nous laissons alors l'œil humain déterminer les contours.

Question 13. Programmer une fonction permettant d'obtenir l'image en niveau de gris des contours.

Le problème de la méthode précédente est l'apparition de faux contours qu'il faut donc filtrer.

Le gradient suivant x faisant ressortir essentiellement les contours verticaux (dans la direction orthogonale), il faut filtrer suivant cette même direction. On remplace donc la valeur $G_x(i, j)$ par la moyenne des valeurs de $G_x(i, j-1)$, $G_x(i, j)$, $G_x(i, j+1)$. De même on remplace $G_y(i, j)$ par la moyenne des valeurs de $G_y(i-1, j)$, $G_y(i, j)$, $G_y(i+1, j)$.

Question 14. Programmer une fonction permettant d'obtenir l'image en niveau de gris des contours en effectuant le lissage précédent.