Ingénieur 2000 - $1^{ère}$ année
TP de C

Traitement d'images

Le but de ce TP est d'écrire quelques fonctions de base du traitement d'images. Nous commençons par décrire le format d'image que nous utilisons dans ce TP.

1 Format d'image - Préliminaires

Le format retenu est le format "*raw*" (brut). Elles permettent de stocker des images en niveau de gris à une unique couche. Les valeurs sont codées sur 256 niveaux de gris dans une matrice à *nbL* lignes et *nbC* colonnes et sont comprises en 0 et 255.
Vous disposerez de 3 images différentes, chacune d'elles étant carrée et de 256 pixels de côté :



Trois images sont également disponibles, les mêmes que celles ci-dessus, auxquelles a été ajouté du bruit.

Selon les conventions classiques du traitement d'images :

- Le pixel en haut à gauche est de coordonnées $(0, 0)$
- L'axe X est horizontal et orienté vers la droite
- L'axe Y est vertical et orienté vers le bas
- (j, i) désigne le pixel de la colonne i et de la ligne j

2 Structure et gestion de la mémoire

▷ **Exercice 1**
Dans les fichiers *basesimages.h* et *basesimages.c*, vous définirez une structure permettant de gérer les images codées en niveau de gris sur 256 pixels. Vous définirez également les fonctions

permettant d'allouer et de désallouer la mémoire permettant de stocker une image. Vous pourrez vous inspirer pour cela de la structure déjà définie pour les matrices écrites il y a quelques TP.

▷ **Exercice 2**
Dans les fichiers *ioimages.h* et *ioimages.c*, vous définirez les fonctions de lecture et d'écriture des images.

3 Affichage

▷ **Exercice 3**
Écrivez dans des fichiers *displayimages.h* et *displayimages.c* une fonction permettant d'afficher une image dans une fenêtre à l'aide de la librairie graphique *libMlv*. Pour afficher un pixel gris, il faut passer à la fonction *draw_point* une chaîne de caractères correspondant à une couleur. 100 niveaux de gris différents sont définis, de *gray0* à *gray255*. A partir de la valeur du pixel, vous construirez la chaîne de caractères correspondant au niveau de gris grâce à la fonction *sprintf* (en divisant cette valeur par 2.55).

4 Traitements basiques

Vous écrirez l'ensemble des fonctions de cette partie dans les fichiers *utilsimages.h* et *utilsimages.c*.
▷ **Exercice 4**
Pour jouer sur le contraste des images, on utilise un réétalement des valeurs. Il peut être réalisé selon la formule suivante :

$$new_{value} = new_{min} + \frac{(new_{max} - new_{min})}{(old_{max} - old_{min})} (old_{value} - old_{min})$$

où new_{min} et new_{max} sont les nouvelles valeurs extrêmes de l'image, old_{min} et old_{max} sont les valeurs extrêmes de l'image avant traitement, new_{value} et old_{value} sont respectivement les valeurs avant et après réétalement.
Écrivez une fonction permettant de réaliser le réétalement d'une image en spécifiant les nouvelles bornes. Vous prendrez soin d'écrire des fonctions permettant de calculer les minimum et maximum d'une image.
▷ **Exercice 5**
La détection de contours dans les images est souvent une étape préalable à des traitements de plus haut niveau. Une façon simple de le faire est de calculer les gradients dans les deux directions principales de l'image. On analyse ensuite la norme du gradient en en conservant les maxima. Pour calculer les gradients, on utilise les formules suivantes :

$$\nabla_X(j,i) = \frac{(j,i-1) - (j,i+1)}{2}$$

$$\nabla_Y(j,i) = \frac{(j-1,i) - (j+1,i)}{2}$$

Le norme du gradient se définit comme suit :

$$\|\nabla(j,i)\| == \sqrt{\nabla_X(j,i)^2 + \nabla_Y(j,i)^2}$$

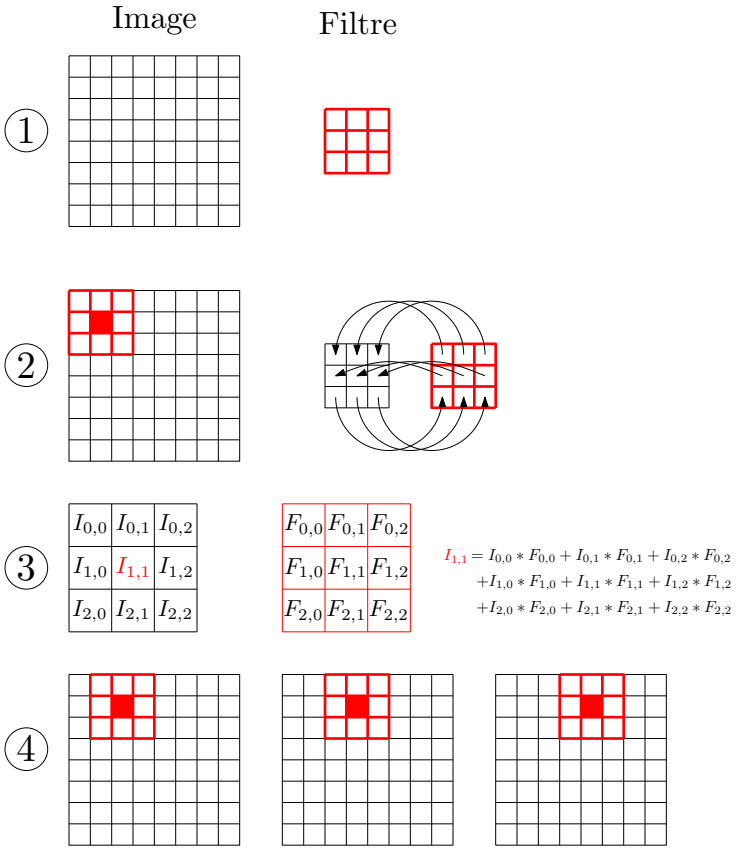
Écrivez trois fonctions permettant de réaliser ces calculs (attention aux bornes ...).

▷ **Exercice 6**

Pour extraire les contours à partir de la norme du gradient, l'opération classique est un seuillage. Cela consiste simplement à construire une image binaire (composée de 0 et de 255) à partir de l'image à traiter et d'un seuil. Si la valeur du pixel de l'image est inférieure au seuil, l'image binaire prend la valeur 0, sinon, elle prend la valeur 255. Écrivez une fonction permettant de réaliser un seuillage.

▷ **Exercice 7**

La convolution est un opérateur mathématique local couramment utilisé en traitement d'images. L'idée de base est qu'une fenêtre de taille finie et de forme connue parcourt l'image. La valeur du pixel en sortie correspond à la somme pondérée des pixels de l'image à traiter multipliés par les valeurs de la fenêtre (cf. l'équation). La fenêtre est appelée le *noyau de convolution* (et pourra être assimilée à une image). Le schéma suivant permet de mieux comprendre le concept de la convolution :



Les deuxième et troisième lignes montrent comment calculer la valeur du pixel de coordonnées $(1,1)$ de l'image I avec une convolution selon le filtre F. Le pixel que l'on calcule est celui qui cor-

respond au centre du filtre. Ainsi, d'une manière générale, un filtre aura des dimensions impaires. La quatrième ligne montre l'évolution de la convolution : on déplace successivement le filtre F sur tous les pixels de l'image où cela est possible.
Écrivez une fonction prenant en entrée deux images, l'une étant l'image à convoluer, l'autre le noyau de convolution, qui permet de calculer la convolution d'une image I par un filtre F.

▷ **Exercice 8**
Le filtre moyenne est classiquement utilisé pour débruiter une image. Il est carré et composé de valeurs identiques dont la somme vaut 1. Écrivez une fonction qui prend en entrée les dimensions du filtre et renvoie un filtre moyenne.
Quelle dimension faut-il prendre pour débruiter les images, tout en gardant une définition convenable ?