TP de Système n° 3 : Manipulation d'images

Le but de ce TP est de manipuler des images au format `ppm P6` . Les noms des fichiers contenant de telles images ont un suffixe `ppm` ; ces fichiers peuvent être visualisés à l'aide de la commande `display`.

Voici un exemple de fichier `ppm` , pour comprendre comment ce format fonctionne :

```
P6          ! format du fichier
2 4         ! largeur et hauteur (nombre de pixels)
15          ! profondeur maximale des couleurs
cccccc      || spécification des couleurs :
cccccc      || 3 caractères par pixel, correspondant
cccccc      || à une combinaison de rouge, vert et bleu
cccccc      || (sans retour à la ligne)
```

Dans ce format, chaque caractère définissant une couleur est codé sur un octet si la profondeur des couleurs est inférieure à 256, sur deux octets sinon. En outre, les caractères ne sont séparés par aucun caractère d'espacement.

Dans la suite du TP, on fait l'hypothèse que la profondeur maximale de couleur de tous les fichiers manipulés est inférieure à 256 : chaque couleur est donc codée sur un octet.

Un fichier d'en-tête vous est fourni :
`http://www.liafa.jussieu.fr/~duchi/Enseignement/SYSTEMES10/image.h`
Vous y trouverez des définitions de macros et de types, ainsi que les spécifications des fonctions à écrire. Vous trouverez également à cette adresse des fichiers `ppm P6` pour tester votre programme.

Déplacement dans le fichier `ppm` Chaque pixel correspond à trois octets. Comme il n'y a pas de caractère d'espacement entre les couleurs, la description des pixels ressemble à un tableau « bidimensionnel - plat ». Le pixel de coordonnées `(i,j)` se situe donc à l'octet `offset+(i*largeur+j)*RGB`, où `offset` est la position du premier pixel dans le fichier `ppm` et `RGB` le nombre de couleurs par pixel (`RGB=3`).

Vous trouverez dans les pages du manuel la description d'outils utiles pour faire ce TP, notament les fonctions `fread`, `fscanf`, `fwrite`.

Exercice :

1. Programmer la fonction `ppm_to_image`.
2. Ajouter la fonction inverse `image_to_ppm`.
3. Écrire un `main` pour tester ces fonctions, en particulier s'assurer que l'identité est conservée (*i.e.* qu'on retrouve bien le même fichier de départ lorqu'on passe par la conversion en *image*). Faire en sorte que le programme prenne le nom du fichier `ppm` en argument sur la ligne de commande.

4. Écrire les fonctions de :
   - miroir horizontal (par rapport à l'axe des abscisses) d'une image,
   - rotation de 90° (on testera aussi avec des images non carrées),
   - symétrie centrale par blocs de 32 32 pixels. L'image est découpée en blocs, et les blocs symétriques par rapport au centre de l'image sont échangés. Découvrir l'image `image_mystere.ppm` (la fonction est son propre inverse).

   Question bonus

5. Écrire une fonction d'extraction de contours. Elle pourra être appelée sur une image en niveaux de gris ou sur une composante. Pour chaque pixel `(i,j)`, d'intensité C (voir tableau), on calcule la norme du gradient en choisissant les dérivées verticale et horizontale de Sobel :

NO	N	NE
O	C	E
SO	S	SE

$$4 \quad D_x = NO + 2 \quad O + SO \quad (NE + 2 \quad E + SE)$$
$$4 \quad D_y = NO + 2 \quad N + NE \quad (SO + 2 \quad S + SE)$$
$$G = \sqrt{D_x^2 + D_y^2}$$

   Le tableau représente un pixel et les 8 pixels qui l'entourent.

   Puis on crée une nouvelle image de même taille, en niveaux de gris, où l'intensité du pixel `(i,j)` est G. Par exemple, pour l'image 1(a), on obtient les contours dessinés dans l'image 1(b) (en négatif).

(a) (b)

FIGURE 1 – Une image et ses contours