TP 9 – Structures, tableaux dynamiques et images

1. Format d'une image

Dans ce TP nous allons concevoir une application permettant d'effectuer des traitements simples sur une image. Les images manipulées seront stockées au format bitmap, BMP.

Dans un premier temps il nous faudra écrire une fonction permettant de lire un fichier image et de le stocker en mémoire. En informatique une image peut être représentée par un ensemble de points (pixels) ayant chacun 3 composantes : bleu, vert et rouge. Dans notre programme une image sera donc représentée par une structure de données comprenant :

- le nombre de pixels en largeur
- le nombre de pixels en hauteur
- un tableau de dimension 2 de largeur x hauteur cases pour stocker la composante rouge
- un tableau de dimension 2 de largeur x hauteur cases pour stocker la composante vert
- un tableau de dimension 2 de largeur x hauteur cases pour stocker la composante bleu

L'allocation des tableaux se fera dynamiquement en fonction des tailles des images chargées.

Les fichiers BMP peuvent être compliqués à manipuler. Les images dans ce format peuvent être compressées ou non, avoir un nombre de couleur variables, etc...
Nous nous contenterons de décoder les fichiers les plus courants : ils sont non compressés, et chaque pixel est codé par un octet pour le bleu, un octet pour le vert et un octet pour le rouge.
Ils sont organisés avec un en-tête décrivant le fichier, puis un autre décrivant l'image, puis un corps composé des valeurs des 3 couleurs pour chaque pixel.

Voici la structure de l'entête fichier et de l'entête image :

octet 0	**octet 1**	**octet 2**	**octet 3**	**octet 4**	**octet 5**	**octet 6**	**octet 7**
Type de fichier		taille du fichier en octet (LSB en premier)				réservé	
réservé		adresse du premier pixel dans le fichier				taille de ...	
...l'entête image		largeur de l'image (LSB en premier)				hauteur de...	
...l'image		nombre de plans		nombre de bits/pixel		type de	
compression		taile de l'image				résolution horizontale...	
en pixel/mètre		résolution verticale en pixel/mètre				nombre de couleurs...	
...dans la palette		nombre de couleurs importantes					

Par exemple pour une image de 640x480 en 16 millions de couleurs, l'analyse du fichier avec la commande hexdump donne cet affichage :

```
$ hexdump -C chat.bmp|more
00000000  42 4d 36 10 0e 00 00 00  00 00 36 00 00 00 28 00  |BM6.......6...(.|
00000010  00 00 80 02 00 00 e0 01  00 00 01 00 18 00 00 00  |................|
00000020  00 00 00 10 0e 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 ae a3  9f aa 9f 9b a4 9b 97 a4  |................|
00000040  9b 97 a1 9a 97 9b 94 91  94 8f 8e 95 90 8f 92 90  |................|
```

- Type de fichier : 'B' 'M' : Fichier Bitmap Microsoft.
- taille du fichier en octet (LSB en premier) : 0x0E1036 soit $14x16^4 + 1x16^3 + 3x16 + 6 =$ 921654 octets
- réservé : 00 00 00 00
- adresse du premier pixel dans le fichier : 0x36 soit 54 octets
- taille de l'entête image : 0x28 soit 40 octets
- largeur de l'image (LSB en premier) : 0x0280 soit $2x16^2 + 8x16 = 640$ points
- hauteur de l'image : 0x01e0 soit $1x16^2 + 14x16$ soit 480 points
- nombre de plans : 1, l'image n'est pas une superposition de plans
- nombre de bits/pixel : 0x18 = 24 bits soit 1 octet / couleur
- type de compression : 0, donc pas de compression
- taile de l'image : 0X0E1000 soit 921600 pixels

- résolution horizontale en pixel/mètre : 0 donc non renseigné
- résolution verticale en pixel/mètre : 0, donc non renseigné
- nombre de couleurs dans la palette : 0, donc mode « true color ». Chaque couleur est représenté par son intensité
- nombre de couleurs importantes : 0, donc elles le sont toutes...

Ensuite, les points sont rangés les uns derrière les autres, ligne par ligne, le point en bas à gauche en premier.
Donc dans l'exemple, le point en bas à gauche est composé des 3 couleurs ayant les intensités suivantes :

- bleu : 0xAE
- vert: 0xA3
- rouge : 0x9F

Pour afficher les images on pourra se servir de l'utilitaire « eog » et l'appeler par cette fonction dans votre programme :

```
void affiImage (char *nom) {
     char commande [100] ;
     sprintf(commande, "/usr/bin/eog %s &", nom) ;
     system(commande);
}
```

Vous pouvez également vous servir de cet utilitaire pour convertir au format BMP des images au format JPEG. Il suffit de faire « enregistrer sous » et de choisir le format BMP.

2. Prototypes des fonctions à écrire

Vous trouverez dans l'archive différents fichiers que vous devrez compléter :

- **bmp.c / bmp.h** : contient des fonctions de manipulation des formats de fichier BMP
- **image.c / image.h** : contient des fonctions de manipulation des images
- **sapin.bmp / chat.bmp** : exemples de fichiers BMP
- **demo.c :** exemple d'utilisation des fonctions. Vous pourrez dé-commenter les lignes de la fonction *main ( )* au fur et à mesure de votre avancement.
- **Makefile** : le makefile permettant de compiler
- **MakefileBis :** une alternative plus complexe au makefile donné

Cet ensemble de fichiers compile déjà. Vous pouvez le compiler et voir l'affichage d'une image.

Voici les fonctions que vous devez écrire à présent :

`Image` **`newImage`** `(int l, int h) ;`
Retourne une structure de données avec une image vide de largeur *l* et de hauteur *h*. L'allocation mémoire des 3 tableaux de couleurs est faite par cette fonction. *Cette fonction est déjà écrite.*

`Image` **`lireImageBMP`**`(const char *nomFic) ;`
Retourne une image lue à partir d'un fichier de nom *nomFic*. *Cette fonction est déjà écrite.*

`void` **`saveImageBMP`**`(Image im, const char * nomFic) ;`
Sauvegarde sur disque l'image *im* dans un fichier de nom *nomFic*.

`Image` **`filtreNB`** `(Image im) ;`
Retourne une nouvelle image en noir et blanc à partir de l'image *im*. On remplacera les 3 couleurs de chaque pixel par la moyenne des 3 couleurs.


fig 1: Image originale, chat qui dort


fig 2: En noir et blanc

`Image` **`sepia`** `(Image im) ;`
Retourne une nouvelle image façon « sépia » à partir de l'image *im*. Comme pour le noir et blanc, on calculera la moyenne (moy) . Les nouvelles couleurs seront calculées ainsi :
bleu = moy + 8
vert = moy + 26
rouge = moy + 44
On veillera à ce que les couleurs restent dans l'intervalle [0,255].

`Image` **`filtreFlou`** `(Image im) ;`
Retourne une nouvelle image plus floue, mais aussi moins bruitée. Chaque couleur du point sera remplacée par la moyenne des couleurs de ses voisins et de lui même (éventuellement pondéré). En bord d'image on veillera au fait qu'il n'y a pas 8 voisins.

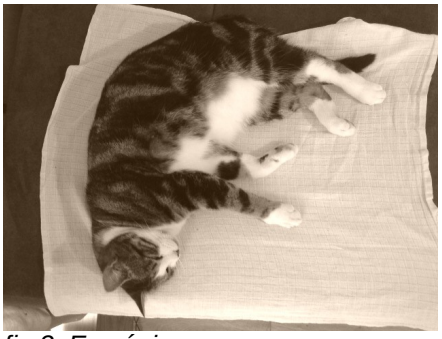


fig 3: En sépia

`Image` **`filtreSeuil`** `(Image im, unsigned char seuil) ;`
Retourne une nouvelle image où chaque point est remplacé soit par la couleur blanche (B=255, V=255, R=255), soit par la couleur noire (B=0, V=0, R=0). Pour chaque point on calcul la moyenne des 3 couleurs. Si la moyenne est supérieure au seuil on remplace le point par la couleur blanche, sinon, on lui affecte la couleur noire.

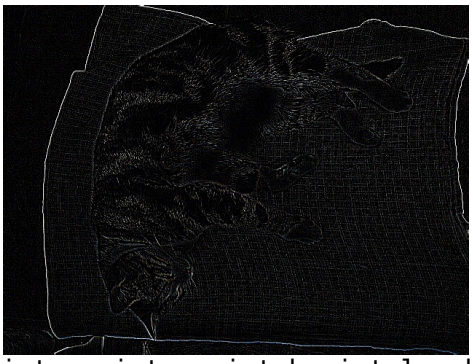

fig 4: Effet de seuillage avec s = 100

```
PixelCouleur filtrePixel3x3 (
     int i,
     int j,
     const Image im,
     unsigned char filtre[ ][3]
) ;
```

Applique un filtre 3x3 sur 1 pt de coordonnées (i,j) de l'image et retourne le pixel et ses 3 couleurs.

`Image` **`filtre3x3`** `(Image im, unsigned char filtre[][3]) ;`
Applique un filtre 3x3 sur l'image et en retourne une nouvelle.
Essayez le filtre 3x3 suivant qui permet la détection de contours = {{-1, -1, -1},{-1, 8, -1}, {-1, -1, -1}} ;

Voici le résultat.



`void` **`rectangle`** `(Image image, int x, int y, int h, int l, char b, char v, char r) ;`
Dessine sur l'image un rectangle plein de couleur (b, v, r) de hauteur h et de largeur l. L'angle supérieur gauche se trouve aux coordonnées (x, y).

`void` **`ligne`** `(Image image, int x0, int y0, int x1, int y1, char b, char v, char r) ;`
Dessine sur l'image un segment de couleur (b, v, r) entre le point de coordonnées (x0, y0) et celui de coordonnée (x1, y1).

(Facultatif)
A l'aide de la fonction rectangle précédente, contruire une image qui sera l'histogramme en niveau de gris d'une image.