T.P. numéro VII

Tableaux et manipulation d'images « bitmap »

Ce T.P. va faire intervenir les notions suivantes :
– lecture/écriture de fichiers binaires ;
– images bitmap ;
– tableaux de données.

1 Fichiers binaires

Les informations stockées sur le disque bien que toujours de type binaire (on code des 0 et des 1, en fait, on oriente des moments magnétiques), peuvent l'être sous plusieurs formes.
Ainsi, imaginons que l'on veuille écrire sur le disque le nombre 15792 ; plusieurs options se présentent alors.
– On peut écrire que 15792, est le caractère `'1'` suivi du caractère `'5'`, lui même suivi du caractère `'7'`, puis `'9'` et enfin `'2'` soit 5 caractères donc 5 octets. Si ce fichier est édité avec un éditeur de texte, on pourra lire le texte 15792. Le fichier contiendra donc la représentation binaire du caractère `'1'`, puis des caractères `'5'`, `'7'`, `'9'` et `'2'`. C'est ce que l'on appelle la représentation ASCII, pour laquelle à chaque caractère est associé un code tenant sur un octet. Les fichiers ASCII sont lisibles par un éditeur de texte.
– On peut aussi écrire que 15792 est un entier dont la représentation binaire est 0011110110110000, elle occupe 2 octets. L'édition du fichier ne permet pas de lire en clair 15792 puisqu'il ne contient pas le texte 15792 mais cette valeur codée en binaire.

L'avantage que l'on peut tirer du binaire est assez évident dans cet exemple, c'est un grand gain de place. En revanche, si on veut que l'utilisateur puisse lire le fichier de sortie, il faudra ecrire un fichier ASCII.

Lecture et écriture dans les fichiers binaires :
La lecture et l'écriture dans les fichiers binaires se font avec les fonctions `fread()` et `fwrite()`. Ces deux fonctions prennent la même séquence d'arguments, par exemple

```
fread(tab, taille ,nombre,fp)
```

où
– tab est un tableau qui contient les données à lire ou à écrire.
– taille est la taille en octets des éléments à lire ou à écrire. Dans ce T.P., les éléments à lire seront de type **unsigned char**, donc de taille 1 octet.
– nombre est le nombre d'éléments à lire ou à écrire.
– fp est un pointeur sur le fichier dans lequel on veut lire ou écrire (pointeur de type FILE∗).

2 Images : Fichiers bitmap

Le format de stockage des images dans l'ordinateur peut prendre un grand nombre de formes, de part la technique de description de l'image (graphiques bitmap, graphiques vectoriels,...) ou du codage final de celle-ci (compression ou nom de l'information).

La manière la plus simple de coder une image est de la décrire point par point, c'est le codage de type bitmap, l'image est donc décrite comme une matrice de points (pixels), la valeur de chacun des points correspondant à la couleur de celui-ci. Ainsi, l'image peut être stockée sous la forme d'un tableau à deux indices tab[i][j]
Attention : le premier indice (i) représente la ligne, le deuxième indice (j) représente la colone, et l'élément tab [0][0] correspond au pixel inférieur gauche de l'image.
En réalité, le terme de couleur est abusif, puisque dans ce type de codage seul un nombre réduit de couleurs est accessible, c'est ce qu'on appelle la palette. En effet, l'impression visuelle donnée par une image peut être retrouvée en utilisant un nombre réduit de couleurs différentes. Définir la palette revient donc à attribuer à chacune de ces couleurs indispensables (variables d'une image à l'autre) un numéro (on écrit un tableau de couleurs). C'est un de ces numéros qui va être attribué à chacun des pixels de l'image.
Ainsi, pour décrire une image en noir et blanc, on attribuera la valeur 0 à la couleur blanche, la valeur 1 à la couleur noire (ou l'inverse). Et on attribuera à chaque pixel de l'image la valeur 0 ou 1 selon qu'il est blanc ou noir.
Imaginons que l'on veuille décrire la même image mais en rouge et blanc maintenant, il suffira d'attribuer la valeur 0 à la couleur blanche, la valeur 1 à la couleur rouge. On aura alors strictement le même codage pour les deux images. Afin de distinguer celles-ci, il suffira de rajouter en entête du fichier la description de la palette.
Les exercices qui suivent vont utiliser l'image comme prétexte à la manipulation de tableaux et matrices.

3 Manipulation d'images en niveaux de gris

Télécharger le fichier bitmap einstein.bmp. On pourra visualiser cette image à l'écran en utilisant la commande

```
eog einstein.bmp
```

Cette image est un bitmap en « niveau de gris » codé sur 8 bits. Cela veut dire que la palette en entête du fichier définit 256 couleurs différentes représentés par les entiers de 0 à 255 (valeur maximale pour 8 bits) ; le 0 correspondant à la couleur noire, le 255 correspondant à la couleur blanche, et les valeurs intermédiaires correspondant à des gris plus ou moins clairs.
Les entiers codés sur 8 bits (1 octet) correspondent au type **unsigned char**. Le tableau codant l'image sera donc un tableau de type **unsigned char**.

Exercice VII.1 *Entrées-sorties*

Télécharger le code source (incomplet) du programme bitmap.c .

1. L'image einstein.bmp à pour dimensions 296×296 pixels. Sachant que chacun de ces pixels sont codées sur un octet, et qu'on peut connaître la taille (en octets) d'un fichier par la commande

   ```
   ls -l fichier
   ```

   déterminer la taille de l'entete du fichier einstein.bmp. Modifier la définition du paramètre `ENTETE` dans le code bitmap.c (ligne 4) en conséquence.
2. Compléter la fonction LitImage() pour qu'elle lise de façon séquentielle dans le fichier pointé par `fp` : l'entête du fichier bitmap qui sera stockée dans le tableau `head`, puis la description de l'image elle-même qui sera stockée dans le tableau `tab`.
3. Compléter la fonction EcritImage() pour qu'elle écrive de façon séquentielle dans le fichier pointé par `fp` : l'entête du fichier bitmap stockée dans le tableau `head`, puis la description de l'image elle-même stockée dans le tableau `tab`.
4. Compiler, exécuter et vérifier que le fichier de sortie est correct.

Ce code ainsi complété servira de point de départ à tous les exercices suivants.

Exercice VII.2 *Noir et blanc*

Le terme « noir et blanc » est abusif pour décrire l'image einstein.bmp car elle contient beaucoup de niveaux de gris différents et pas uniquement du noir et du blanc.

1. Écrire la fonction NoirEtBlanc() qui transforme effectivement l'image en noir et blanc : les niveaux de gris inférieurs au gris moyen devenant noirs, les autres devenant blancs.
   Appeler la fonction dans le programme principal, compiler, exécuter et visualiser le résultat.
2. Écrire la fonction Negatif () qui échange les niveaux de gris 0, 1, ... 255 en 255, 254 , ... 0.
   Appeler la fonction dans le programme principal, compiler, exécuter et visualiser le résultat.

Exercice VII.3 *Effet miroir*

– L'effet miroir vertical consiste à observer l'image obtenue dans un miroir plaçé verticalement sur un coté de l'image de départ : ce qui était à gauche passe à droite et inversement.
– L'effet miroir horizontal consiste à observer l'image obtenue dans un miroir plaçé horizontalement au dessus ou en dessous de l'image de départ : ce qui était en haut passe en bas et inversement.

1. Écrire la fonction Miroir () qui réalise l'effet miroir horizontal ou vertical suivant la valeur d'un paramètre qui lui est passé en argument.
   Après avoir demandé à l'utilisateur le type d'effet miroir souhaité, appeler la fonction dans le programme principal et compiler.
2. Exécuter et visualiser le résultat de chacune de ces opérations individuellement, puis des deux à la suite. Conclusion ?

Exercice VII.4 *Effet mosaïque*

Principe : on considère l'image zone par zone et on remplace tous les niveaux de gris des pixels de la zone par la moyenne des niveaux de gris des pixels de celle-ci. En pratique, on considérera ici des zones carrées de 8×8 pixels.

1. Écrire la fonction Mosaique() qui réalise cet effet :
   (a) passer successivement dans chacune de ces zones à l'aide de boucles **for** ;
   (b) dans chacune de ces zones, faire la moyenne des niveaux de gris des 64 pixels ;
   (c) donner ce niveau de gris moyenné à tous les pixels de la zone.
2. Appeler la fonction dans le programme principal, compiler, exécuter et visualiser le résultat.

Exercice VII.5 *Effet flou*

Principe : on remplace le niveau de gris de chaque pixel par la moyenne des niveaux de gris des pixels voisins (plus lui même). En pratique ici, on faira la moyenne dans un carré de 7×7 pixels centré sur le pixel concerné.
Cet effet nécessite de sauver le résultat dans un tableau intermédiaire puisque pour chaque point il faut calculer la moyenne sur les niveaux de gris voisins dans l'image avant transformation. De plus, pour les pixels proches du bord de l'image, on devra moyenner sur des carrés incomplets.

1. Écrire la fonction Floutage () qui réalise cet effet :
   (a) passer successivement sur toutes les coordonnées (i, j) des pixels de l'image ;

(b) pour chacune de ces coordonnées, additionner les niveaux de gris des pixels *contenus dans les limites de l'image* du carré de 7×7 centré sur (i, j) et compter le nombre de pixels ainsi traités ;

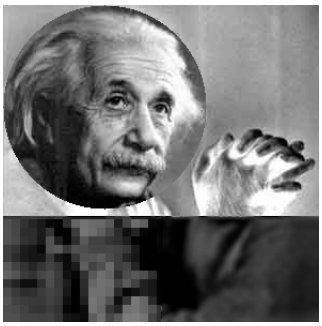
(c) stocker dans le tableau intermédiaire aux coordonnées (i, j) la moyenne des niveaux de gris additionnés précédemment ;

(d) une fois traitées toutes les coordonnées (i, j) recopier le tableau intermédiaire dans le tableau de départ.

2. Appeler la fonction dans le programme principal, compiler, exécuter et visualiser le résultat.

Exercice VII.6 *Soyons fou ...*

Les fonctions définies dans les exercices précédents agissent sur tous les pixels, donc sur l'image dans son ensemble. On peut modifier ces fonctions pour ne les faire agir que sur une portion de l'image en utilisant des structures **if** pour sélectionner par leurs coordonnées les pixels sur lesquels on veut agir. On peut ensuite combiner toutes ces fonctions à la suite pour obtenir par exemple :



Par exemple, le cercle centré sur le visage à pour coordonnées :

$$(i - 200)^2 + (j - 95)^2 = 95^2$$

où i est l'indice de la ligne et j l'indice de la colonne.

1. Combinez les effets des exercices précédents selon votre inspiration ...

Listing des codes à télécharger

Code bitmap.c

```
#include <stdio.h>
#include <stdlib.h>

#define ENTETE 0 /* à remplacer */
#define TAILLE 296


int LitImage(FILE *fp, unsigned char head[ENTETE],
             unsigned char tab[TAILLE][TAILLE]) {
  /* à compléter */
}

int EcritImage(FILE *fp, unsigned char head[ENTETE],
             unsigned char tab[TAILLE][TAILLE]) {
  /* à compléter */
}


int main() {
  FILE *fp;
  char fichier[30];
  unsigned char head[ENTETE];
  unsigned char tab[TAILLE][TAILLE];

  printf("Nom du fichier en entrée ?\n");
  scanf("%s",fichier);
  fp = fopen(fichier,"r") ;
  LitImage(fp,head,tab);
  fclose(fp);

/*---  Traitement de l'image  ---*/

  printf("Nom du fichier en sortie ?\n");
  scanf("%s",fichier);
  fp = fopen(fichier,"w+") ;
  EcritImage(fp,head,tab);
  fclose(fp);
}
```