Licence Informatique
Université Bordeaux I

Initiation à la Programmation
Projet 2002 - 2003

À la réalisation de fonctionnalités superflues on privilégiera la qualité du code (clarté et concision du code) et la qualité technique (compilation sans avertissement, gestion aboutie de la mémoire).

1 Introduction

L'image, sous forme de matrice de pixels[1] d'intensité variable, constitue la matière de ce projet.

Nous travaillerons exclusivement des images de format PGM[2], format assez répandu et pris en compte par des logiciels de visualisation comme `xv`. Dans ce format, l'intensité lumineuse d'un pixel prend sa valeur parmi 256 nuances de gris, ce qui peut se coder sur 8 bits. Pour les entrées/sorties d'image, nous utiliserons un module programmé par M. P. Chassignet.

L'objectif de ce projet est de proposer un environnement de programmation de transformation d'image au moyen de filtre. Ce projet comporte trois parties :

1. l'écriture de filtres ;
2. la mise en œuvre de structures de données facilitant la programmation de filtres ;
3. l'utilisation de filtres à des fins de stéganographie.

L'ensemble des squelettes des sources nécessaires à la réalisation de ce projet se trouvent à `~wacren/IP/projet`. Il est fortement recommandé d'utiliser les structures de données proposées. Le projet est à réaliser en monôme ou binôme.

2 Écriture de filtres

2.1 Définitions et exemple

Un filtre d'image est une fonction mathématique qui, appliquée à chaque pixel d'une image, transforme celle-ci. Pour une image de résolution *largeur* $\times$ *hauteur*, on note $p_{i,j}$ le pixel de coordonnées $\{i, j\}$. L'intensité d'un pixel p est notée $I(p)$. On sait que $0 \leq I(p) \leq 255$.

Le filtre `Moyenne` agit de la façon suivante. Pour tout pixel p de l'image, on considère le carré connexe de 9 pixels dont le centre est p (on dira que la hauteur et la largeur de ce filtre sont égales à 3). Ce filtre met à jour l'intensité du pixel p en lui affectant comme intensité la moyenne des intensités des pixels du carré.

178	182	230
148	230	230
136	146	149

FIG. 1 – L'intensité du pixel central sera après application du filtre `Moyenne` égale à 181.

[1]contraction de *picture elements*

[2]pour *portable graymap file format*, voir `http ://netpbm.sourceforge.net/doc/pgm.html` ou faire `man pgm` pour plus d'informations

Mathématiquement, on aimerait ainsi écrire la nouvelle intensité du pixel :

$$\frac{\sum\limits_{-1\leq k\leq 1, -1\leq l\leq 1} I(p_{i+k,j+l})}{9}$$

Et donc, à première vue, la programmation de ce filtre pourrait se faire à l'aide d'une fonction

`unsigned char FiltreIntensitePixelVoisin(size_t i, size_t j);`

qui permettrait d'accéder à l'intensité des pixels voisins *relativement* à un pixel préalablement fixé. Cependant cette nouvelle intensité est en fait égale à :

$$\frac{\sum\limits_{-1\leq k\leq 1, -1\leq l\leq 1} J(p_{i+k,j+l})}{\sum\limits_{-1\leq k\leq 1, -1\leq l\leq 1} C(p_{i+k,j+l})}$$

où

$$\begin{cases} C(p_{k,l}) = J(p_{k,l}) = 0 & \text{si } k < 0 \text{ ou } k \geq \text{largeur ou } l < 0 \text{ ou } l \geq \text{hauteur}, \\ C(p_{k,l}) = 1, J(p_{k,l}) = I(p_{k,l}) & \text{sinon.} \end{cases}$$

Cette formulation peut paraître compliquée ; cependant, il faut éviter que les pixels hors de l'image ne faussent le calcul. Cette difficulté se reporte naturellement dans la traduction du filtre en langage C. Aussi, de façon générale, on considère qu'une image est encadrée par une bande fictive de largeur infinie de pixels dont on fixera l'intensité arbitrairement mais toujours de façon uniforme sur l'ensemble de ces pixels dits *hors image*. Cela se programme en introduisant un troisième paramètre de type `int` qui sera utilisé comme valeur par défaut :

`int FiltreIntensitePixelVoisin(size_t i, size_t j, int IntensiteHorsImage)`

On pourra distinguer les pixels hors image des autres pixels, en attribuant une valeur exclue de $[0, 255]$ au paramètre `IntensiteHorsImage`. Le filtre moyenne s'écrit alors :

```
unsigned char
Moyenne11 (void *Parametre)
{
  int i, j, tmp;
  int somme = 0;
  int cardinal = 0;

  for (i = -1; i <= 1; i++)
    for (j = -1; j <= 1; j++)
      {
        tmp = FiltreIntensitePixelVoisin (i, j, PixelHorsImage);
        if (tmp != PixelHorsImage)
          {                    /* Tir cadre */
            cardinal++;
            somme += (unsigned char) tmp;
          }
      }

  return (unsigned char) (somme / cardinal);
}
```

et l'application de ce filtre à une image est réalisé par l'appel suivant :

`FiltrerImage(Moyenne11, image, NULL,3,3);`

2.2 Travail demandé

L'objectif de cette première partie est la réalisation d'une bibliothèque de filtres et d'un programme `filtrer.c` dont le format d'appel sera, par exemple, le suivant :

`filtrer -nom_du_filtre fichier_en_entree.pgm fichier_en_sortie.pgm`

Pour compiler vos programmes vous utiliserez le fichier `filtre.o`. Les filtres seront :

1. `PerturbationAleatoire` qui, pour deux paramètres entiers `a` et `b`, consiste à mettre à jour l'intensité d'un pixel en lui affectant une valeur aléatoire comprise entre 0 et 255 avec une probabilité $\frac{a}{b}$ et ne pas mettre à jour l'intensité de ce pixel avec une probabilité $1 - \frac{a}{b}$;
2. `Contraction` qui, pour un paramètre réel `facteur`, consiste à mettre à jour l'intensité d'un pixel en la multipliant par `facteur` ;
3. `Laplacien`[3] qui permet d'extraire les contours des objets en mesurant la différence entre un point et la moyenne de ses voisins ; la nouvelle intensité d'un pixel $p_{i,j}$ est donnée par la formule :
$$\frac{|4.I(p_{i,j}) - (I(p_{i+1,j}) + I(p_{i-1,j}) + I(p_{i,j-1}) + I(p_{i,j+1}))|}{4}$$
L'intensité des pixels hors image sera pour ce filtre fixée à 128.
4. `Median` qui consiste à mettre à jour l'intensité d'un pixel p en lui affectant l'intensité médiane des intensités des pixels dans l'image composant le carré de 9 pixels connexes de centre p. Naturellement, les intensités fixées des pixels hors image ne doivent pas être prises en compte dans le calcul de cette intensité médiane. Sur l'exemple de la figure 1, la mise dans l'ordre des intensités donne
$$\{136, 146, 148, 149, \quad 178, \quad 182, 230, 230, 230\}$$
et donc la valeur médiane de ces intensités est 178 qui sera la nouvelle intensité du pixel central après application du filtre médian.
Si maintenant, on s'intéresse au pixel du coin supérieur droit de l'image, seuls les 4 pixels du coin supérieur droit sont à prendre en compte et la mise dans l'ordre de leurs intensités donne
$$\{148, 178, 182, 230\}.$$
Dans le cas d'un nombre pair $2n$ de pixels dans l'image, on choisira comme intensité médiane la n ème intensité et donc, pour cet exemple, la nouvelle intensité affectée au coin supérieur droit sera 178.
Dans le codage en langage C de ce filtre, vous utiliserez *obligatoirement* la fonction `qsort`.

On pourra modifier une image grâce au filtre de perturbation aléatoire et déterminer expérimentalement si le filtre médian corrige ou non mieux l'image perturbée que le filtre moyenne. On pourra aussi, pour s'amuser, inventer d'autres filtres.

3 Structures de données adaptées aux filtres

3.1 Contexte d'exécution

Le travail désigné de la fonction `FiltrerImage` est d'appliquer une fonction `Filtre` à l'ensemble des pixels d'une image. Comme nous l'avons vu, les filtres sont programmés à l'aide des deux fonctions `FiltreIntensitePixelCourant` et `FiltreIntensitePixelVoisin`. Le rôle de la fonction `FiltrerImage` est donc, pour chaque pixel de l'image de :

[3] de l'astronome - mathématicien - physicien - chimiste Pierre-Simon Laplace, mars 1749 - 5 mars 1827 ; Laplace fût un porte-drapeau du déterminisme mécaniste ; faire `man laplace` pour plus d'information.

1. Définir le contexte d'exécution de l'appel à la fonction `Filtre` pour le pixel à traiter ;
2. Exécuter la fonction `Filtre` dans ce contexte.

Le contexte d'exécution définit donc le comportement des fonctions `FiltreIntensitePixelCourant()` et `FiltreIntensitePixelVoisin()`, ; il s'agit d'un ensemble de variables globales :

- `ImageCourante`[4] désigne l'image en cours de traitement ;
- `PixelCourant` désigne le pixel en cours de traitement ;
- `Tampon` désigne une copie, éventuellement partielle, de l'image avant sa transformation.

L'application de filtres de hauteur ou de largeur supérieure à 2 nécessite la copie, au moins partielle, de l'image en cours : le filtre utilise l'intensité initiale de chaque pixel d'une part, et, d'autre part, mémorise les valeurs fraîchement calculées.

3.2 Préliminaires : une version gourmande

Dans un premier temps, il s'agit d'écrire une première version relativement simple du module filtre en plaçant dans le tampon une copie (complète) des pixels de l'image en cours de traitement.

1. Définir la structure de la variable `Tampon`.
2. Compléter les fonctions d'initialisation et de nettoyage des variables globales – on utilisera la fonction `memcpy()` pour recopier l'image en cours.
3. Écrire les fonctions `FiltreIntensitePixelVoisin()` et `FiltreIntensitePixelCourant()`.
4. Écrire la fonction `FiltrerImage()`.

3.3 Travail demandé

Le but de cette partie est de proposer une structure de données en général plus économe en mémoire que la simple duplication de l'image initiale.

Supposons que la hauteur de la matrice du filtre soit égale à $2.k+1$; en particulier cela signifie que les pixels des lignes déjà traitées et situées à une distance supérieure à k du pixel en cours de traitement n'ont plus d'influence sur le calcul. Il suffit[5] donc de mémoriser, en plus de la ligne en cours, la valeur initiale des k dernières lignes calculées, à la seule condition de traiter l'image ligne après ligne.

Optimiser l'occupation de la mémoire de votre programme utilisant la variable tampon suivante :

```
static struct {
  size_t nblignes;
  unsigned char **ligne;
  unsigned char *pixels;
} Tampon;
```

On veillera à éviter les recopies multiples d'une même ligne et l'utilisation du modulo, opération fort coûteuse en temps de calcul.

Enfin, l'utilisation de variables globales causent inévitablement des problèmes. Ici, il s'agit de l'impossibilité de traiter des appels dits *ré-entrant* : le fait de pouvoir utiliser de façon parallèle (de façon *multi-threadée*, comme disent les étudiants de Maîtrise) ou de façon (indirectement) récursive le module filtre. Pour éviter cet écueil, on pourra :

- soit interdire (via la macro `assert`) l'exécution ré-entrante de la fonction `FiltrerImage()`.
- soit autoriser les appels ré-entrant en supprimant les variables globales – il s'agit alors de définir un objet filtre en-capsulant les données actuellement globales et de redéfinir l'ensemble des modules...

[4] dans le texte source on a renommé cette variable `ImaCour` pour une meilleure lisibilité des boucles for.
[5] mais ce n'est pas nécessaire...

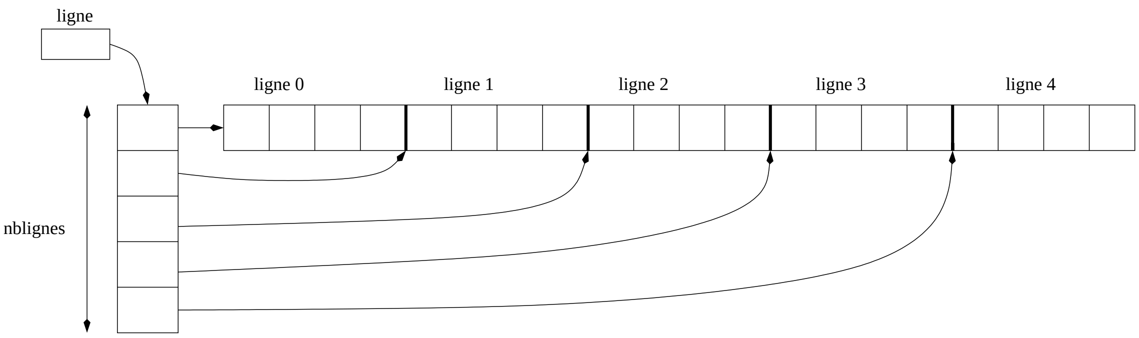

FIG. 2 – Structure de la variable Tampon

4 Application à la stéganographie

On veut maintenant, à des fins stéganographiques[6] chères à Georges Sand et Alfred de Musset, cacher un texte dans une image. Une technique simple de camouflage consiste à disperser les bits du message sur les bits de poids faible des premiers pixels de l'image. Ainsi, sur 8 pixels, un caractère (1 octet) du message pourra être codé.

FIG. 3 – Edouardo, Dussaux, Christian et le sujet d'un examen de Janvier 2003. Distinguez-vous ce dernier ?

Proposer un filtre réalisant le codage d'un message dans une image et un autre pour le décodage. On pourra admettre, dans un premier temps seulement, que le nombre de caractères du message est connu.

[6] du grec $\sigma\theta\epsilon\gamma\alpha\nu o$, couvert et $\gamma\rho\alpha\varphi\epsilon\iota\nu$, écriture

Écrire le programme de camouflage correspondant.

Mettre en évidence l'existence d'un message caché dans les premiers pixels d'une image demeure assez aisé notamment si ses premiers pixels sont d'égale intensité. Proposer un programme C de signature d'image basé sur une technique de camouflage plus difficile à repérer.

5 Modalités Pratiques

Trois séances de travaux dirigés seront consacrées à ce projet. La date butoir de remise du projet est fixée au lundi 20/01/2003, 10h00.

Un dossier est à remettre sous les formes papier (à déposer au secrétariat de la LMI) et électronique (courrier contenant un unique fichier compressé). Ce dossier comprendra :
- les sources du projet (correctement présentées) ;
- un fichier makefile ;
- une démonstration basée sur des images et un script de démonstration `demo` paramétré par le nom du logiciel de visualisation (par exemple `demo xv`) ;
- un rapport contenant la description des fonctionnalités traitées et la liste exacte des points non traités, un compte rendu des difficultés rencontrées et une conclusion générale sur le projet.

Vous mentionnerez les auteurs des codes sources éventuellement utilisés et vous citerez les noms de vos *proches collaborateurs.*

Pour information voici le résultat de la commande wc avant et après destruction des quelques lignes demandées :

Avant

```
[corrige]$ wc *.c
    126     348    2709 filtre.c
    121     381    2597 libfiltre.c
     80     194    1615 filtrer.c
    127     311    2530 signature.c  // filtre de camouflage
```

Après

```
     78     138    1092 filtre.c
     69     177    1142 libfiltre.c
```

Il y a donc de l'ordre de 350 lignes de code à écrire.