POLYTECH PARIS-SUD

**Et3 - Tronc commun**

Programmation C

# TP Noté

## 1 Objectifs

Ce projet individuel vous permettra de mettre en pratique les connaissances acquises dans le module de programmation en C. Le but est d'implanter quelques outils fondamentaux de traitement d'images : lecture et écriture d'une image en niveaux de gris puis calcul de l'histogramme de l'image, de son négatif, puis exemple simple d'application d'un filtre numérique.

*Rappel :* Séparez de manière appropriée vos fonctions et structures dans des fichiers *.c et *.h qui eux-mêmes ont un nom logique.

## 2 Contexte

Une image numérique est une matrice de pixels, chacun de ces pixels codant une intensité lumineuse entre 0 (pas de lumière) et 255 (pixel allumé entièrement). Cette intensité est généralement codée sur 8 bits. La position de chaque pixel n'est pas codée, l'image (dont on connaît les dimensions) étant simplement représentée par une succession de pixels en mémoire.

Un format simple de stockage d'image est le format PGM ASCII. Ce format est structuré sur la base suivante :

- le nombre magique du format (deux octets) : il est fixé à la valeur 'P2'
- un caractère d'espacement (espace, tabulation, nouvelle ligne) ;
- la largeur de l'image (nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII) ;
- un caractère d'espacement ;
- la hauteur de l'image (idem) ;
- un caractère d'espacement ;
- la valeur maximale utilisée pour coder les niveaux de gris, que l'on notera $v_{max}$ ;
- un caractère d'espacement ;
- les données de l'image : succession des valeurs associées à chaque pixel ; l'image est codée ligne par ligne en partant du haut, et chaque ligne est codée de gauche à droite ;
- toutes les lignes commençant par un croisillon # sont ignorées (lignes de commentaires).

Par exemple, voici une image au format PGM :

```
P2
# Affiche le mot "FEEP" (exemple de la page principale de Netpbm à propos de PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Note : Vous pourrez tester sur le “FEEP” ou sur des plus gros fichiers tels que : PengBrew ou Lena. Ensuite, vous pouvez lire vos fichiers *.pgm avec GIMP ou Irfanview, par exemple pour évaluer le rendu et vérifier le résultat.

# 3 Lecture/Écriture d’image

1. Implantez une structure `image_t` qui contienne les informations nécessaires au stockage des pixels issus d’une image PGM ASCII : hauteur, largeur et $v_{max}$ sous forme entiers, ainsi qu’un pointeur vers les données des pixels stockées comme des entiers non signés 8 bits.
2. Implantez une fonction `read_image` qui charge un fichier texte contenant la représentation au format PGM d’une image et renvoie une instance de `image_t` contenant les données. Pensez à vérifier que le mode est bien P2.

3. Implantez une fonction `write_image` qui sauvegarde dans un fichier au format PGM l’image contenue dans une instance de `image_t`.

Vous implanterez aussi toutes fonctions auxiliaires que vous jugerez nécessaires.

# 4 Image inverse

L’inverse d’une image est définie comme une image où chaque pixel contient le complément à $v_{max}$ de la valeur du pixel d’origine. Ainsi, les pixels valant $n$ sont transformés en pixels valant $v_{max} - n$.

4. Implantez une fonction `inverse_image` qui prend en paramètre une `image_t` et renvoie une `image_t` contenant l’inverse de l’image d’entrée.
   Il faudra que l’on puisse passer le nom des fichiers par la ligne de compilation.

Vérifiez le résultat en permettant de sauvegarder l’image obtenu grâce à votre fonction `write_image`.

# 5 Histogramme d’une image

En imagerie numérique, l’histogramme représente la distribution des intensités de l’image. Pour une image en niveaux de gris, l’histogramme est défini comme une fonction qui associe à chaque valeur d’intensité le nombre de pixels prenant cette valeur. La détermination de l’histogramme est donc réalisée en comptant le nombre de pixels pour chaque intensité de l’image. On effectue parfois une quantification, qui regroupe plusieurs valeurs d’intensité en une seule classe, ce qui peut permettre de mieux visualiser la distribution des intensités de l’image. Définissez le nombre de classes comme vous le souhaitez. (Attention, le faire en fonction du nombre de valeurs possibles paraît une bonne idée !)

5. Implantez une fonction `histogram_image` qui prenne en paramètre une `image_t` et renvoie une structure contenant l’histogramme de l’image d’entrée. Pour ce faire, définissez et implantez une structure capable de stocker les valeurs de l’histogramme.

# 6 Filtre numérique RIF

En traitement du signal, un filtre à réponse impulsionnelle finie ou filtre RIF (en anglais Finite Impulse Response filter ou FIR filter) est un filtre dont la réponse impulsionnelle est de durée finie. De façon générale le filtre à réponse impulsionnelle finie est décrit par la combinaison linéaire suivante, où

$$x[i]_{1 \leq i \leq n}$$

représente les valeurs du signal d'entrée et

$$y[i]_{1 \leq i \leq n}$$

les valeurs du signal de sortie :

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + .... + b_N \cdot x[n-N]$$

En utilisant le symbole de sommation, l'équation peut être réécrite de la façon suivante :

$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k] y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k]$$

Au niveau de l'implantation, un tel filtre se code en utilisant une boucle sur tous les pixels de l'image qui va travailler sur un voisinage de taille fixe pour produire le résultat filtré. Par exemple, le FIR 1D (monodimensionnel) qui moyenne les pixels d'une image peut s'écrire comme suit :

```
for(int y=0;y<hauteur;y++){
  for(int x=1;x<largeur-1;x++){
    sortie[y][x] = (entree[y][x-1] +entree[y][x] +entree[y][x+1])/3;
  }
}
```

On dira que ce filtre est défini par **la matrice de convolution** $\frac{1}{3}[111]$.

Pourquoi les bornes de la boucle sur `x` ne démarrent-elles pas à 1 ? Si le filtre était plus large, c'est-à-dire que le nombre de points dans le voisinage augmente, comment calculeriez-vous le point de départ et d'arrivée de la boucle ?

6. Implantez deux fonctions `filter_border`$_x$ et `filter_border`$_y$ qui prennent en entrée une image et renvoient l'image contenant les contours verticaux et horizontaux de l'image. Pour ce faire, vous utiliserez les filtres suivants qui approximent les bords en calculant les dérivées spatiales de l'image comme suit. Soit $\mathbf{A}$ l'image source, $\mathbf{G_x}$ et $\mathbf{G_y}$ les deux images qui en chaque point contiennent respectivement des approximations de la dérivée horizontale et de la dérivée verticale de chaque point. Ces images sont calculées comme suit :

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$$

# 7 Rendu

## 7.1 Détails pratiques

- Il s'agit d'un travail individuel.
- *Date de rendu* : mercredi 15 février 2017, minuit. **-3pts/j** de retard.
- Envoyer par mail à votre chargé de TP (cf. adresses e-mail ci-dessous) les fichiers suivants dans un dossier compressé (format .zip, tar.gz ou autre) :
  1. vos fichiers *.c et les fichiers *.h associés ;
  2. un fichier nommé Readme.txt dans lequel la ligne de compilation/exécution est fournie, ou un fichier Makefile pour ceux qui préfèrent ;
  3. Un rapport d'une page maximum (police standard de taille 11, marges raisonnables) pour répondre aux questions du sujet et pour expliquer ce que vous avez fait, mettre en valeur ce dont vous êtes fiers, les difficultés que vous avez éprouvées, etc.
- Le nom du dossier compressé doit être de la forme `nom_prenom_numéroGroupe` (par exemple "Dupont_Pierre_5.zip")
- L'objet de votre mail doit être de la forme `[et3-tp-c] nom prénom numéroGroupe` (par exemple "[et3-tp-c] Dupont Pierre 5")
- ATTENTION : un TP ne sera considéré comme rendu qu'une fois que son destinataire vous aura répondu en vous confirmant par e-mail la bonne réception de votre rendu.
  Tout non-respect de ces consignes simples sera naturellement sanctionné.

## 7.2 A qui envoyer votre TP ?

Le rendu est a envoyer à un chargé différent de votre chargé de TP. Ci-dessous, le numéro des groupes et le mai du chargé de TP à qui envoyer :

| **Groupe n°** | **A envoyer à :** |
|---|---|
| 1 | laveau@lri.fr |
| 2 | victor.antoine.estrade@gmail.com |
| 3 | jennifer.vandoni@gmail.com |
| 4 | rachel.bawden@limsi.fr |
| 5 | arnaud.ferre@u-psud.fr |

## 7.3 Critères de notation

- Pour le code, seront pris en compte :
  1. La modularisation de votre code (est-ce que vous avez séparé le code en fonctions minimales ? est-ce que les fichiers *.c et *.h sont correctement distingués ?) ;
  2. La présence de commentaires ;
  3. L'indentation ;
  4. L'utilisation de noms adéquats pour les variables et les fonctions ;
  5. La bonne compilation de votre code ;
  6. L'utilisation de structures adaptées aux tâches à effectuer et aux données à manipuler ;
- Projet : Jusqu'à où vous êtes allés ?
- Le Rapport doit :
  1. Faire 1 page maximum ;
  2. Être concis et clair ;
  3. Ne pas contenir de fautes d'orthographe (utilisez les correcteurs orthographiques si besoin) ;
  4. Ne pas contenir de code source.

Le TP noté comptera pour 30% de la note finale.