Travaux pratiques #5 — Images PGM

Introduction

Lena la guerrière.

Les images PGM sont parmi les plus simples à gérer, à la fois dans leur structure et dans la simplicité de leur format de fichier. Une image PGM est une image en *niveaux de gris*, c'est-à-dire composée de pixels pouvant prendre des valeurs comprises entre 0 et `VALEUR_MAX` $\leqslant 65536$.

Le format PGM. Les fichiers PGM sont tous composés de la même manière [1] :

- Le nombre magique du format (deux octects), prenant valeur dans P_i, $i \in \{1, \ldots, 6\}$.
- Un caractère d'espacement (espace, tabulation, nouvelle ligne)
- Les largeur et hauteur de l'image, écrites explicitement sous forme d'un nombre en caractères ASCII.
- Un caractère d'espacement (espace, tabulation, nouvelle ligne).
- La valeur maximale de l'image, écrite explicitement sous forme d'un nombre en caractères ASCII.
- Un caractère d'espacement (espace, tabulation, nouvelle ligne).
- Les données de l'image : succession des valeurs associées à chaque pixel. Les images sont codées ligne par ligne en partant du haut, et de gauche à droite.
- Il est également possible d'insérer des commentaires dans ces fichiers, démarrant par le symbole #.

Le nombre magique

Le nombre magique correspond aux différents formats d'images portables (PBM, PGM et PPM, respectivement image en noir et blanc, niveaux de gris et couleur RGB), ainsi qu'au mode d'enregistrement de leur fichier (ASCII ou binaire).

	PBM	PGM	PPM
ASCII	P1	P2	P3
Binaire	P4	P5	P6

La structure `image_pgm`. La première étape de ce TP consiste à définir une structure `image_pgm`, permettant de gérer (et de traiter) ces images par la suite.

La création de cette structure est libre, et doit permettre d'enregistrer toutes les informations nécessaires à la gestion d'une image au format PGM Binaire (`P5`) [2].

Un exemple d'image PGM au format ASCII est donné ci-dessous (au format ASCII les lignes ont $\leqslant 70$ caractères) :

1. https://fr.wikipedia.org/wiki/Portable_pixmap#PGM
2. Il est également possible d'élargir légèrement la structure, en s'autorisant à gérer n'importe quel format d'image portable.

Une image PGM au format ASCII

```
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
```

La gestion d'images PGM

Le premier objectif de ce TP est d'écrire une librairie permettant de gérer les images PGM.

$\mathcal{Q}_1$. *Créer* une image PGM en fonction de paramètres définis par l'utilisateur. Il s'agit là uniquement de créer la *structure* `image_pgm` contenant les informations nécessaires au traitement de l'image. *Attention au nombre magique !*

$\mathcal{Q}_2$. *Lire* une image PGM depuis un fichier. La lecture devra prendre en compte le format de fichier défini précédemment, et retourner une *structure* `image_pgm` contenant les informations nécessaires au traitement de l'image.

$\mathcal{Q}_3$. *Écrire* une image PGM dans un fichier. Il faudra à nouveau prendre soin de respecter le format décrit précédemment.

La librairie — objectifs

Il sera évidemment -probablement- nécessaire de définir des fonctions intermédiaires pour la réalisation de cette librairie. À nouveau, la structure des fichiers et des fonctions est laissée libre.

Traitements

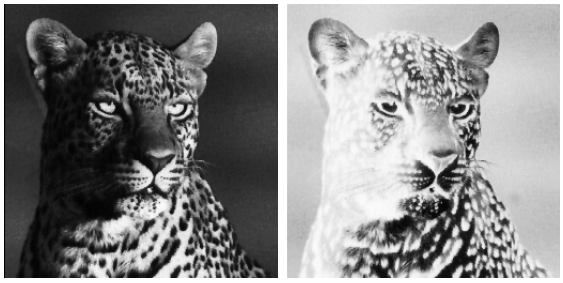
Gradients et détection de contours

Le traitement d'images est un domaine de l'informatique particulièrement riche, dans lequel de nombreux algorithmes peuvent être implémentés. Deux des plus simples sont le filtre *négatif* et la détection de contours.

Dans le premier cas, il s'agit s'implement de remplacer chaque pixel par son complémentaire, défini en fonction de la valeur maximum que peuvent prendre les pixels.

Le filtre négatif

Si l'on considère une valeur maximale de 255, un pixel de valeur 10 aura une valeur 245 dans l'image sur laquelle le filtre négatif est appliquée.



Dans le second cas, de nombreux *masques* peuvent être appliqués, permettant d'obtenir une image où les contours de l'image originale sont mis en avant. L'un des masques les plus simples est le masque horizontal, consistant à remplacer la valeur de chaque pixel $p_{i,j}$ en fonction de la valeur de son pixel suivant :

$$p_{i,j} = p_{i+1,j} - p_{i,j}$$

On notera que le masque vertical peut être défini de manière similaire.

$$p_{i,j} = p_{i,j} - p_{i,j-1}$$

Les gradients — vertical et horizontal

$\mathcal{Q}_4$. Ecrire des fonctions `negatif` et `gradient`, retournant respectivement une image après application du filtre négatif et des gradients.

La question

Pour permettre une meilleure lisibilité de votre code, il est demandé d'implémenter ces fonctions dans un fichier différent de celui contenant la librairie `PGM`.

Traitements génériques

Pour un filtre avec toi.

Les deux filtres précédents correspondent à des fonctions appliquées à l'image. Il est évidemment possible d'imaginer vouloir utiliser d'autres filtres, qui seront également appliqués à l'image. Ces opérations possèdent de nombreuses étapes similaires : ouverture de l'image originale, enregistrement de l'image filtrée, etc. Afin d'éviter une duplication de code trop importante, il serait bon d'avoir une fonction `filtre` prenant en paramètre l'image à traiter **et** le filtre à appliquer.

La généricité. Il est possible de passer une fonction en *paramètre* d'une fonction (avec ses propres paramètres), et de l'instancier au moment de l'appel à la fonction.

$\mathcal{Q}_5$. Ecrire une fonction `filtrer` prenant en paramètre une image à filtrer, le filtre à appliquer (sous la forme de *pointeur de fonction*) et le nom de l'image de sortie. Un prototype de fonction possible est :

```
void filtrer(struct image i, void(*filtre)(struct image i), char* image_de_sortie)
```

La question

Aller plus loin

Il est possible d'envisager d'autres méthodes de détection de contour (voir ici) ou encore de réaliser des méthodes de seuillage (voir ici).