Manipulation d'images au format ppm

Objectifs :
- manipulation des images au format `pnm` ;
- utilisation de la bibliothèque `bcl` ;
- écriture des premiers programmes en `C` de manipulation d'images.

1 Le format d'image ppm

1. À l'aide du manuel en ligne, étudier le format d'image `ppm`.
2. **Petite révision :** écrire un programme en `C` permettant de construire une image de couleur au format `ppm`/`ASCII`. L'exécution se fera de la manière suivante :

   ```
   $> print_ppm r g b w h > my_ppm.ppm
   ```

   où $r, g, b \in [0, 255]$ et w, h sont des valeurs entières et correspondent, respectivement, à la largeur et la hauteur de l'image en sortie. Voici un squelette de code à utiliser et à modifier :

   ```
   /**
    * @author Vinh-Thong Ta <ta@labri.fr>
    * @file print_ppm.c
    * @brief print to standard output a color ppm file
    */
   #include <stdlib.h>
   #include <stdio.h>

   static void process(int r, int g, int b, int w, int h){
     /* etc */
   }
   void usage (char *s){
     char* use = "Usage: %s <r[0,255]> <g[0,255]> <b[0,255}> <width> <height>\n";
     fprintf(stderr,use,s);
     exit(EXIT_FAILURE);
   }
   #define param 5
   int main(int argc, char *argv[]){
     if (argc != (param+1)) usage(argv[0]);
     /* etc */
     process(r, g, b, w, h);
     return EXIT_SUCCESS;
   }
   ```

   Consignes :
   - Compiler avec la ligne de commande suivante :
     `$> gcc -Wall -Wextra -std=c99 print_ppm.c -o print_ppm`
     où `-Wall -Wextra -std=c99` active les warnings et mets le compilateur sur le standard `c99` et `-o` *name* fabrique l'exécutable en lui donnant le nom *name*.
   - Par exemple, la commande suivante : `$> print_ppm 0 255 255 100 200 > my_ppm.ppm` fabrique une image de taille 100×200 et de couleur cyan.
   - Visualiser le résultat.

2 Installation de la bibliothèque `bcl`

Cet exercice a pour but d'installer et de tester les fonctions de manipulation d'image `ppm` de la bibliothèque libre `bcl`.

Consignes :

- Créez un répertoire `project` et décompresser le fichier `src.tgz` Vérifier qu'on obtient la hiérarchie :

```
project +-- src  +-- bcl +-- Makefile
                 |       |-- bcl.c
                 |       |-- bcl.h
                 |       |-- pnm.c
                 |       |-- pnm.h
                 |       |-- ...
                 |-- td-pnm +-- Makefile
                            |-- test1.c
                            |-- td-pnm.pdf
                            |-- lena_color.ppm
```

- l'archive contient : un exemple de `Makefile`, un exemple de programme (`test1.c`) utilisant `bcl`, une image couleur de test (`lena_color.ppm`) et une version `pdf` de l'énoncé du TD.
- Étudier le fichier `Makefile` afin de comprendre comment s'effectue l'installation de la bibliothèque `bcl`.
- Installer la bibliothèque (`make install` depuis le répertoire `bcl`). Vérifier le contenu des répertoires `include` et `lib` créés lors de l'installation.
- Étudier le code du module `pnm` afin de comprendre son fonctionnement.

3 Interface du module pnm

Le module `pnm` (écrit en langage `C`) définit un type `pnm` permettant de lire des images aux formats `ppm`, `pgm`, et `pbm`, d'en manipuler le contenu, et de le sauvegarder au format `ppm`/`RAW`. Une fois chargée, l'image comporte 3 canaux, quelque soit son format initial.

Gestion des fichiers et objets.

- `pnm pnm_load(char *path)` : chargement d'un fichier au format PNM et instanciation d'un objet de type `pnm` initialisé avec l'image chargée.
- `void pnm_save(pnm self, pnmType type, char *path)` : écriture de l'image d'un objet `pnm` dans un fichier.
- `pnm pnm_new(int width, int height, pnmType type)` : création d'un objet `pnm` de largeur `width` et hauteur `height`.
- `void pnm_free(pnm self)` : libération d'un objet `pnm`.
- **N.B. :** Le seul format (`pmnType`) actuellement supporté (paramètre `type`) est le format `ppm`/`RAW` : `PnmRawPpm`.

Obtention des caractéristiques.

- `unsigned short pnm_maxval` : niveau maximal par canal.
- `int pnm_get_width(pnm self)`
- `int pnm_get_height(pnm self)`

Accès aux pixels de l'image.

- `unsigned short *pnm_get_image(pnm self)` : retourne une référence vers le tampon de l'image (suite d'entiers `short` $r_1 g_1 b_1 r_2 g_2 b_2 r_3 g_3 b_3 ...$)
- `int pnm_offset(pnm self, int line, int column)` : retourne le décalage du pixel de l'image de l'objet `pnm` `self` situé en ligne `line` et colonne `column`.
  Exemple d'accès à la composante rouge du pixel (i, j) :

```
unsigned short *image = pnm_get_image(pnm_image);
unsigned short *p = image + pnm_offset(pnm_image, i, j);
```

- `unsigned short pnm_get_component(pnm self, int i, int j, pnmChannel channel)` : obtention d'un canal d'un pixel.
- `void pnm_set_component(pnm self, int i, int j, pnmChannel channel, unsigned short v)` : modification d'un canal d'un pixel avec `pnmChannel` $\in$ {`PnmRed`, `PnmGreen`, `PnmBlue`}

Manipulation des canaux R, G et B.

- `unsigned short *pnm_get_channel(pnm self, unsigned short *buffer, pnmChannel channel)` : extraction d'un canal et recopie du canal dans le tampon `buffer` ; si la valeur du paramètre `buffer` est le pointeur `NULL`, la mémoire destinée à recevoir le canal est allouée dynamiquement.
- `void pnm_set_channel(pnm self, unsigned short *buffer, pnmChannel channel)` : modification d'un canal de l'image d'un objet `pnm`.

4 Quelques programmes utilisant le module `pnm`

1. Créer un répertoire `bin` au même niveau que `src`, `include`, et `lib`.
2. Étudier, compiler (avec le `Makefile` fourni) et exécuter le programme `test1`.
3. Ajouter votre programme `print_ppm.c` dans le répertoire `td-pnm`. Mettre à jour le `Makefile` de manière à pouvoir compiler votre programe. Compiler, tester que votre programme fonctionne toujours.

En utilisant le module `pnm` écrire les programmes `C` suivants. **N'oublier pas de mettre à jour le `Makefile`** pour l'ensemble des programmes demandés.

4.1 `extract_subimage.c`

Écrire le programme `extract_subimage x0 y0 w h ims imd` qui permet d'extraire une sous image de l'image source `ims` à partir du pixel positionné en `x0`, `y0` de taille `w`×`h`. L'image résultat est sauvegardée dans `imd`.

4.2 `extract_channel.c`

Écrire le programme `extract_channel num ims imd` qui permet d'extraire le numéro de canal `num` de l'image couleur source `ims` et le sauve dans l'image résultat `imd`.

4.3 `gray2color.c`

Écrire le programme `gray2color ims0 ims1 ims2 imd` qui permet de composer une image couleur (sauvegardée dans `imd`) à partir de trois images en niveaux de gris (`ims0 ims1 ims2`).

4.4 `color2mean.c`

Écrire le programme `color2mean ims imd` qui de permet de convertir une image couleur (`ims`) en une image en niveau de gris (`imd`) où chaque pixel de l'image résultat est la moyenne des trois canaux de l'image couleur.

4.5 `normalize.c`

Écrire le programme `normalize min max ims imd` qui de permet de normaliser les valeurs des pixels de l'image source `ims` dans la nouvelle image `imd` en utilisant le nouvel intervalle $[\min, \max]$ et en appliquant la fonction de normalisation suivante

$$I'(x,y) = \frac{\max - \min}{\mathrm{Max}(I) - \mathrm{Min}(I)} \times I(x,y) + \frac{\min \times \mathrm{Max}(I) - \max \times \mathrm{Min}(I)}{\mathrm{Max}(I) - \mathrm{Min}(I)} \tag{1}$$

où I' est l'image résultat, $\mathrm{Min}(I)$ et $\mathrm{Max}(I)$ correspondant respectivement aux valeurs minimale et maximale de l'image source I.