Rapport de Projet de Programmation impérative

Implémentation de différents traitements sur des images en niveaux de gris ou en couleur

Steve DIEME

Encadré par Guillaume BUREL

Filière : formation initiale ENSIIE, 1ère année

Année scolaire 2024/2025

Table des matières

Introduction

Ce projet vise à mettre en œuvre divers traitements sur des images en niveaux de gris et en couleur. Ce rapport présente les choix effectués, les défis techniques rencontrés, ainsi que les solutions mises en place.

Les images peuvent être considérées comme des matrices de pixels, chaque pixel ayant une valeur variant de 0 à 255. Dans le cas des images en niveaux de gris, chaque pixel est défini par une seule valeur, tandis que les images en couleur sont composées de trois valeurs représentant les canaux Rouge, Vert et Bleu (R, G, B).
Le projet se divise en deux parties distinctes :

- Manipulation des images : Techniques et opérations pour modifier les images.
- Application de modifications visuelles sur une image : Intégration d'effets ou d'ajustements esthétiques sur les images.

Nous travaillerons avec l'image Lenna_gray.ppm ci-dessous disponible aussi en noir et blanc.

FIGURE 1.1 – Lenna_color.ppm

Fonctionnement du programme

2.1 Structures de données

Les images sont représentées par une structure `picture` définie comme suit :

```
4   typedef unsigned char byte;
5   #define MAX_BYTE 255
6
7   typedef struct {
8       byte* data;
9       int h;
10      int w;
11      int c;
12  } picture;
```

FIGURE 2.1 – Lenna_color.ppm

— `width w` : largeur de l'image.
— `height h` : hauteur de l'image.
— `channels c` : nombre de canaux (1 pour niveaux de gris, 3 pour couleur).
— `data` : pointeur vers un tableau unidimensionnel contenant les pixels.

Un module `pictures.h` (et son fichier picture.c) gère les fonctions de manipulation d'images, et un sous-module `pixels.h` (et son fichier pixel.c) est dédié à l'accès direct aux composantes des pixels.

2.2 Modules et fonctions implémentés

2.2.1 Lecture et écriture des fichiers images

Les fonctions sont :

— `picture read_picture(char *filename);` : lit un fichier image au format PGM ou PPM et retourne une structure `picture`.
Difficulté rencontrée : La vérification du bon fonctionnement de cette fonction était délicate. J'ai décidé d'afficher la structure `picture` à la fin de son exécution pour vérifier la validité des données lues.

— `int write_picture(picture p, char *filename);` : écrit une image dans un fichier, en vérifiant les extensions et les erreurs possibles.
Difficulté rencontrée : Aucun fichier n'était créé à l'issue de son exécution. J'ai rapidement constaté que le problème ne venait pas de cette fonction, mais de la fonction `extfrompath` que j'utilisais incorrectement.

2.2.2 Manipulation des images

— Création d'une image : `picture create_picture(unsigned int width, unsigned int height, unsigned int channels);`

— Nettoyer les données d'une image : `void clean_picture(picture *p);`

— Copie d'une image : `picture copy_picture(picture p);`

— Obtention d'informations sur une image :
  — Indication d'image vide : `int is_empty_picture(picture p);`

  — Indication d'image en niveaux de gris : `int is_gray_picture(picture p);`

  — Indication d'image en couleurs : `int is_color_picture(picture p);`

  — Affichage des infos d'une image : `void info_picture(picture p);`
  Remarque : En vérifiant les informations sur les images, j'ai réalisé que `void info_picture(picture p);` correspondait à ma fonction `read_picture_info`.
— Conversion d'un format à un autre :
  — `picture convert_to_color_picture(picture p);` : convertit une image en niveaux de gris en une image couleur.
  — `picture convert_to_gray_picture(picture p);` : convertit une image couleur en niveaux de gris.
  Ajout : J'ai ajouté une vérification des premiers pixels pour m'assurer que mes fonctions fonctionnaient correctement.
— Séparation et fusion des canaux :
  — `picture *split_picture(picture p);` : sépare les canaux d'une image couleur.

  — `picture merge_picture(picture red, picture green, picture blue);` : fusionne trois canaux en une image couleur.

2.2.3 Manipulation directe des valeurs des pixels

— Eclaircissement d'une image :
  — `picture brighten_picture(picture p, double factor);` : augmente la luminosité des pixels.
  Difficulté rencontrée : J'ai dû m'assurer que les valeurs ne dépassent pas 255 pour éviter les débordements.
— Fonte (vers le bas) des valeurs des pixels d'une image :
  — `picture melt_picture(picture p, int number);` : fond les pixels d'une image.
  Difficulté rencontrée : J'ai pris soin de vérifier que la taille de l'image est supérieure à 2, car chaque pixel a besoin de celui au-dessus de lui. J'ai aussi constaté que le résultat attendu n'était pas le même que dans le cours, ce qui m'a amené à revoir mon approche. Cela a nécessité quelques schémas pour bien visualiser l'opération.

2.2.4 LUT (Look Up Table)

Un module `lut.c/h` implémente des fonctions de transformation des pixels en utilisant des LUTs. Les LUTs permettent d'inverser, normaliser ou réduire les niveaux des pixels.

Un LUT agis comme une fonction sur un pixel.

— `lut* create_lut(int n);` : crée une LUT.

— `void free_lut(lut* lut);` : libère une LUT.

— `picture lut_on_picture(picture p,lut* lut);` : applique la LUT sur l'image. *Difficulté rencontrée :* A l'aide de la difficulté rencontré à la fonction melt_picture j'ai pu mieux comprendre le fonctionnement du tableau unidimensionnel, ce qui a facilité la mise en œuvre de cette fonction.

2.2.5 Re-échantillonnage

Deux méthodes de re-échantillonnage sont définies, c'est sur cette partie du projet que j'ai passé le plus de temps (ainsi que les deux fonctions principales read_picture et write_picture) :

— `picture resize_nearest_neighbor(picture p, int new_width, int new_height);`

Difficulté rencontrée : J'ai rencontré un probleme qui m'a fais perdre beaucoup de temps, j'avais inversé i (parcourt en largeur) et j (en hauteur) dans les boucles ce qui provoquait un dépassement de mon image ce qui ne fournissait pas du tout le résultat attendu.

FIGURE 2.2 – Image en sortie

— `picture resize_bilinear(picture p, int new_width, int new_height);` : utilise une interpolation bilinéaire.Je n'ai malheureusement pas réalisée cette fonction malgré l'aide fourni

Manuel d'utilisation

Ce manuel décrit l'utilisation du programme qui permet de manipuler des images en utilisant des opérations de traitement d'images.

2.3 Compilation

Pour compiler le programme, vous devez d'abord vous assurer que vous avez les fichiers nécessaires dans votre répertoire de travail. Ensuite, compilez le programme en utilisant la ligne de commande suivante :

```
gcc -Wall -Wextra -o my_program main.c picture.c filename.c lut.c
```

Cela générera un exécutable nommé `myprogram` que vous pourrez utiliser avec différentes options.

2.4 Utilisation du programme

Le programme peut être exécuté avec différentes options selon l'opération souhaitée.Chaque commande que vous exécutez suit une structure générale :

```
./my_program <operation> <fichier_entree> <fichier_sortie>
```

```
<operation> : Le type d'opération que vous souhaitez effectuer (par exemple,
convert_color, brighten, melt, etc.).
<fichier_entree> : Le fichier d'image que vous souhaitez traiter.
<fichier_sortie> : Le nom du fichier où le résultat sera enregistré.
```

Le seul moyen que j'ai trouvé de generer toute les images en une seule compilations est de faire une boucle dans le main puis d'entrer toute les executions dans le terminale (certainement pas la meilleur méthode), un copier-coller est disponible ici.

2.4.1 Conversion d'une image en niveaux de gris et en couleur

La commande suivante convertit une image en niveaux de gris en une image couleur.

```
./my_program convert_to_color Lenna_gray.pgm ./Images/Lenna_gray_convert_color.ppm
```

La commande suivante convertit une image couleur en une image en niveaux de gris.

```
./my_program convert_to_gray Lenna_color.ppm ./Images/Lenna_color_convert_gray.pgm
```

2.4.2 Fusionner les canaux de couleur

La commande suivante permet d'extraire les composantes de couleur d'une image.

```
./my_program split Lenna_color.ppm ./Images/Lenna_color_red.pgm ./
   Images/Lenna_color_green.pgm ./Images/Lenna_color_blue.pgm
```

2.4.3 Augmenter la luminosité de l'image

La commande suivante permet d'augmenter la luminosité de l'image, en appliquant un facteur d'augmentation.

```
./my_program brighten Lenna_gray.pgm ./Images/Lenna_gray_brighten.pgm
./my_program brighten Lenna_color.ppm ./Images/Lenna_color_brighten.ppm
```

2.4.4 Faire "fondre" l'image

La commande suivante permet de faire fondre l'image. J'ai décider de modifier le facteur est de retirer le *5 car le temps pour réaliser l'image etait bien trop long

```
./my_program melt Lenna_gray.pgm ./Images/Lenna_gray_melted.pgm
./my_program melt Lenna_color.ppm ./Images/Lenna_color_melted.ppm
```

2.4.5 Inverser les couleurs de l'image

La commande suivante permet d'inverser les couleurs d'une image en niveaux de gris ou en couleur.

```
./my_program inverse Lenna_gray.pgm ./Images/Lenna_gray_inverse.pgm
./my_program inverse Lenna_color.ppm ./Images/Lenna_color_inverse.ppm
```

2.4.6 Dynamique optimale

La commande suivante permet d'optimiser la dynamique d'une image.

```
./my_program normalize_dynamic Lenna_gray.pgm ./Images/Lenna_gray_dynamic.pgm
\subsection{Dynamique optimale des composantes de couleur}
```

La commande suivante permet d'optimiser la dynamique de chaque composante de l'image couleur, puis de les recomposer en une nouvelle image.

```
./my_program split Lenna_color.ppm ./Images/Lenna_color_red.pgm ./
   Images/Lenna_color_green.pgm ./Images/Lenna_color_blue.pgm
./my_program normalize_dynamic ./Images/Lenna_color_red.pgm ./
   Images/Lenna_color_red_dynamic.pgm
./my_program normalize_dynamic ./Images/Lenna_color_green.pgm ./
   Images/Lenna_color_green_dynamic.pgm
./my_program normalize_dynamic ./Images/Lenna_color_blue.pgm ./
   Images/Lenna_color_blue_dynamic.pgm
./my_program merge ./Images/Lenna_color_red_dynamic.pgm ./Images/
   Lenna_color_green_dynamic.pgm ./Images/Lenna_color_blue_dynamic.
   pgm ./Images/Lenna_color_dynamic.ppm
```

2.4.7 Limiter le nombre de niveaux

La commande suivante limite l'image source à 8 niveaux par composante.

```
./my_program levels Lenna_gray.pgm ./Images/Lenna_gray_levels.pgm
./my_program levels Lenna_color.ppm ./Images/Lenna_color_levels.ppm
```

2.4.8 Réduction de taille avec le plus proche voisin

La commande suivante permet de rétrécir l'image source.

```
./my_program resample_nearest Lenna_gray.pgm ./Images/
   Lenna_gray_smaller_nearest.pgm
./my_program resample_nearest Lenna_color.ppm ./Images/
   Lenna_color_smaller_nearest.ppm
```

2.4.9 Réduction de taille avec interpolation bilinéaire **Fonction non réalisée**

La commande suivante permet de rétrécir l'image source.

```
./my_program resample_bilinear Lenna_gray.pgm ./Images/
   Lenna_gray_smaller_bilinear.pgm
./my_program resample_bilinear Lenna_color.ppm ./Images/
   Lenna_color_smaller_bilinear.ppm
```

2.4.10 Agrandissement avec le plus proche voisin

La commande suivante permet d'agrandir l'image source.

```
./my_program resample_larger Lenna_gray.pgm ./Images/
   Lenna_gray_larger_nearest.pgm
./my_program resample_larger Lenna_color.ppm ./Images/
   Lenna_color_larger_nearest.ppm
```

2.4.11 Agrandissement avec interpolation bilinéaire **Fonction non réalisée**

La commande suivante permet d'agrandir l'image source.

```
\begin{verbatim}
./my_program resample_bilinear Lenna_gray.pgm ./Images/
   Lenna_gray_larger_bilinear.pgm
./my_program resample_bilinear Lenna_color.ppm ./Images/
   Lenna_color_larger_bilinear.ppm
```

2.4.12 Différence entre interpolations **Fonction non réalisée**

La commande suivante permet de calculer la différence normalisée entre les deux types d'interpolations.

```
./my_program difference Lenna_gray_larger_nearest.pgm
   Lenna_gray_larger_bilinear.pgm ./Images/Lenna_gray_difference.
   pgm
./my_program difference Lenna_color_larger_nearest.pgm
   Lenna_color_larger_bilinear.pgm ./Images/Lenna_color_difference.
   ppm
```

2.4.13 Produire un masque et appliquer le produit

Pour cette opération, vous aurez besoin d'une image masque.

```
./my_program multiply panda.pgm Lenna_gray.pgm ./Images/Lenna_gray_product.pgm
./my_program multiply panda.ppm Lenna_color.ppm ./Images/Lenna_color_product.ppm
```

2.4.14 Mixture de deux images

La commande suivante permet de mélanger l'image inversée et l'image source en utilisant le masque.

```
./my_program mix ./Images/Lenna_color_inverse.ppm Lenna_color.ppm
   panda.ppm ./Images/Lenna_color_mixture.ppm
./my_program mix ./Images/Lenna_gray_inverse.pgm Lenna_gray.pgm
   panda.pgm ./Images/Lenna_gray_mixture.pgm
```

2.5 Conclusion

Ce projet m'a permis d'approfondir mes connaissances en manipulation d'images et d'acquérir de l'expérience dans le développement de fonctions pratiques en programmation impérative. Les défis rencontrés ont été formatifs et ont renforcé mes compétences en résolution de problèmes.

2.6 Les commandes à entrer dans le terminale après compilation.

`Disponible dans le fichier commands.txt de l'archive tar afin de copier coller`