INFO0030 : Projet de Programmation
Traitement d'Images par Application de Filtres

B. Donnet, E. Marechal, M. Goffart, E. Wansart

Alerte : Évitez de perdre bêtement des points...

- Nous vous conseillons **vivement** de relire et d'appliquer les **guides de style et de langage**, mis à votre disposition sur eCampus (INFO0030, Sec. Supports pour le Cours Théorique). Cela vous permettra d'éviter de nombreuses erreurs et de perdre des points inutilement.
- Nous vous conseillons de consulter la **grille de cotation** utilisée pour la correction des projets afin d'éviter de perdre bêtement des points. Elle est disponible sur eCampus (INFO0030, Sec. Procédures d'Évaluation).
- Prenez le temps de lire attentivement l'énoncé. La plupart des réponses aux questions que vous vous posez s'y trouvent.
- Votre code sera compilé et testé sur les **machines CANDI qui servent de référence**. La procédure à suivre pour se connecter en SSH aux machines CANDI est disponible sur eCampus. Veillez donc à ce que votre code fonctionne dans cet environnement. Si vous n'avez pas de compte sur ces machines, veuillez contacter Marc Frédéric.
- Votre solution ne pourra pas être générée entièrement ou partiellement à l'aide d'un outil d'Intelligence Artificielle (e.g., ChatGPT, Blackbox, etc.).

Modifications : dans l'énoncé

- Clarification par rapport à la librairie PNM. Voir texte en bleu dans la Section 3.

1 Contexte

Une image peut être décomposée en un tableau de points élémentaires appelés *pixels* (abréviation de *"picture element"*). Comme détaillé dans l'énoncé du premier projet, les images au format `PNM` peuvent être représentées de 3 manières :

- `PBM` (*Portable Bitmap File Format*) pour les images en noir et blanc.
- `PGM` (*Portable Graymap File Format*) pour les images en niveau de gris.
- `PPM` (*Portable Pixmap File Format*) pour les images en couleur.

Les valeurs possibles des pixels dépend du format utilisé pour représenter l'image.

Pour ce projet, nous vous demandons d'écrire un programme qui permet d'appliquer un filtre graphique particulier à une image qui vous est donnée dans un fichier au format `PNM` (*Portable Anymap*). Ce format est bien connu et a fait l'objet du projet précédent. Il vous est donc demandé, dans ce projet, de **réutiliser votre librairie** permettant de manipuler des images `PNM`.

2 Filtres

Dans cette section, nous décrivons les différents filtres que vous devrez appliquer sur des images `PNM`.

(a) Image initiale (b) Filtre de retournement (Sec. 2.1) (c) Filtre monochrome rouge (Sec. 2.2)

(d) Filtre négatif (Sec. 2.3) (e) Filtre gris (Sec. 2.4) (f) Filtre NB (Sec. 2.5)

FIGURE 1 – Application des différents filtres.

2.1 Retournement

Le filtre de *retournement* effectue une rotation de 180° de l'image. La Fig. 1(b) illustre l'application du filtre de retournement sur la Fig. 1(a).

Le filtre de retournement s'applique à n'importe quel format d'image (i.e., `PBM`, `PGM` ou `PPM`).

2.2 Monochrome

Le filtre *monochrome* permet d'obtenir une version monochrome d'une image couleur.

Pour obtenir une version monochrome d'une image, il suffit de transformer chaque couleur de pixel $c = (r, v, b)$ [1] en annulant une des trois composantes, i.e.,

— $c' = (r, 0, 0)$, pour un filtre monochrome rouge.
— $c' = (0, v, 0)$, pour un filtre monochrome vert.
— $c' = (0, 0, b)$, pour un filtre monochrome bleu.

La Fig. 1(c) illustre l'application d'un filtre monochrome rouge sur la Fig. 1(a).

Le filtre monochrome ne s'applique qu'aux images `PPM`.

2.3 Négatif

Le filtre *négatif* permet d'obtenir le négatif d'une image couleur.

La couleur négative d'une couleur $c = (r, v, b)$ est la couleur

$$c' = (255 - r, 255 - v, 255 - b). \quad (1)$$

1. Le triplet (r, v, b) désigne l'intensité de rouge (r), de vert (v) et de bleu (b) d'un pixel.

La Fig. 1(d) illustre l'application d'un filtre négatif sur la Fig. 1(a).
Le filtre négatif ne s'applique qu'aux images `PPM`.

2.4 50 Nuances de Gris

Le filtre *gris* permet de construire une version nuances de gris d'images en couleurs. Une nuance de gris est une couleur obtenue en donnant la même valeur aux trois composantes (rouge, vert, bleu) d'une couleur donnée.

Nous vous proposons deux techniques pour l'application du filtre gris :

1. On attribue la moyenne arithmétique des trois composantes (r, v, b) à une nouvelle couleur, i.e.,

$$c' = int(round(\frac{r+v+b}{3})). \tag{2}$$

2. On applique la formule suivante :

$$c = int(round(0.299 \times r + 0.587 \times v + 0.114 \times b)). \tag{3}$$

La fonction $round()$ arrondi au nombre le plus proche, tandis que la fonction $int()$ retourne la valeur entière associée à un nombre réel. Dans le cas où le résultat de la formule donne un nombre ambigu (par exemple $1,5$), le nombre sera arrondi à l'entier supérieur, c'est à dire 2. La Fig. 1(e) illustre l'application d'un filtre gris (deuxième technique) sur la Fig. 1(a).

Le filtre gris ne s'applique qu'aux images `PPM` et produit une image `PGM`.

2.5 Noir & Blanc

Le filtre *NB* permet de construire une version en noir et blanc seulement (pas de gris intermédiaire) d'images en couleurs.

Le principe consiste à appliquer d'abord le filtre gris (cfr. Sec. 2.4) et transformer, ensuite, ce gris en un noir ou un blanc selon que la nuance de gris est inférieure ou supérieure à un seuil fixé. Plus précisément, si la valeur du pixel est *supérieure ou égale* au seuil, le pixel deviendra un pixel blanc. La Fig. 1(f) illustre l'application d'un filtre NB (avec une valeur de seuil de 115 – le filtre gris a été appliqué avec la deuxième formule) sur la Fig. 1(a).

Le filtre NB s'applique aux images `PPM` et `PGM` et produit une image `PBM`.

3 Énoncé du Projet

Il vous est demandé d'écrire un programme C permettant d'appliquer sur des images `PNM` les filtres expliqués à la Sec. 2.

Votre projet devra :

— Être soumis dans une archive `tar.gz`, appelée `filtres.tar.gz`, via la Plateforme de Soumission. La décompression de votre archive devra fournir tous les fichiers nécessaires **dans le répertoire courant où se situe l'archive**. Vous penserez à joindre tous les codes sources nécessaires. Votre archive doit être chargée sur la Plateforme de Soumission pour le **Mardi 25/03/2025, 08h00** au plus tard.

— Réutiliser la librairie PNM que vous avez réalisé lors du premier projet. Il est évident que l'équipe pédagogique s'attend à ce que vous ayez apporté des **modifications** à votre librairie PNM en fonction du feedback donné. A cette fin, vous nous fournirez le **code source nécessaire de votre librairie** `PNM`.

— Être modulaire, i.e., nous nous attendons à trouver un (ou plusieurs) header(s) et un (ou plusieurs) module(s). Il est évident que votre projet doit contenir un programme utilisant le code écrit.

— Appliquer les principes de la programmation défensive (vérification des préconditions, vérification des mallocs, ...). Pensez à libérer la mémoire allouée en cours de programmation afin d'éviter les fuites de mémoire.

— Être parfaitement documenté. Vous veillerez à documenter correctement chaque header/fonction/-procédure/structure de données que vous définirez. Votre document suivra les principes de l'outil `doxygen`[2].
— Implémenter les fonctions permettant de filtrer une image (cfr. Sec. 2).
— Être validé par une librairie de **tests unitaires**. Pour cela, vous utiliserez l'outil `seatest` vu au cours[3]. N'oubliez pas de joindre, dans votre archive, les fichiers `seatest.c` et `seatest.h`. Ces tests unitaires porteront sur votre librairie `PNM` mais aussi sur certaines fonctionnalités que vous aurez mis en place pour ce projet (à vous de voir lesquelles sont les plus pertinentes).
— Comprendre un `Makefile` permettant au moins de :

1. Compiler vos tests unitaires pour votre librairie `PNM`. L'exécution de la commande

```
$>make pnm_tests
```

doit produire un fichier binaire, appelé `pnm_tests`, permettant de tester votre librairie `PNM`. Ce fichier binaire devra pouvoir être exécuté de la façon suivante :

```
$>./pnm_tests
```

2. Créer la librairie `PNM`, nommée `libpnm.a`, depuis les fichiers sources de votre librairie `PNM` (cfr. votre projet 1 avec les modifications en fonction du feedback fourni). Nous vous demandons de créer une librairie **statique**. La libraire devra pouvoir être créée à l'aide de la commande suivante :

```
$>make pnm_librairie
```

3. Compiler votre projet. L'exécution de la commande

```
$>make filtre
```

doit produire un fichier binaire, exécutable, appelé `filtre`. Attention, lors de la compilation, vous ne pouvez **pas** recompiler les fichiers sources propres à la librairie `PNM`. A la place, vous devez intégrer la librairie `libpnm.a` (cfr. instruction précédente). Donc, la commande `make filtre` doit créer la libraire à l'aide de la règle `pnm_librairie` dans le makefile. `make filtre` doit être la règle exécutée lorsqu'on fait la commande `make`. Le fichier binaire généré, `filtre`, doit pouvoir être exécuté de la façon suivante :

```
$>./filtre -i <image_input> -f <filtre> [-p <param>] -o <image_output>
```

où `<image_input>` est l'image au format `PNM` qu'il faudra manipuler dans le programme. Attention, votre programme doit *impérativement* accepter les chemins absolus et relatif des fichiers (e.g., `/home/user/x/fichier.ppm` ou `../../fichier.ppm`).
`<filtre>` est le filtre à appliquer et `<param>` l'éventuel paramètre associé au filtre. Les valeurs possibles pour l'option `<filtre>` sont les suivantes :

retournement Applique le filtre de retournement (cfr. Sec. 2.1). Il n'y a pas de paramètre associé à ce filtre (d'où le côté optionnel de l'option `-p`).

monochrome Applique le filtre monochrome (cfr. Sec. 2.2). Le paramètre associé peut être soit `r` (application d'un filtre monochrome rouge), `v` (application d'un filtre monochrome vert) ou `b` (application d'un filtre monochrome bleu).

negatif Applique le filtre négatif (cfr. Sec. 2.3). Il n'y a pas de paramètre associé à ce filtre.

gris Applique le filtre gris (cfr. Sec. 2.4). Dans ce cas-ci, le paramètre sera soit 1 (application de la moyenne arithmétique), soit 2 (application de la deuxième formule).

NB Applique le filtre NB (cfr. Sec. 2.5). Dans ce cas-ci, le paramètre sera le seuil (une valeur comprise entre 0 et 255).

Le résultat du filtrage sera placé dans un nouveau fichier `PNM` dont le nom sera indiqué par l'option `-o <image_output>`.

4. Générer de la documentation. L'exécution de la commande

2. Vous veillerez, aussi, à documenter en `doxygen` votre librairie `PNM`.
3. Le code source de `seatest` est disponible sur la page Web du cours.

```
$>make doc
```

doit produire une documentation, au format HTML, dans le sous-répertoire `doc/`.

Pour tester votre code, vous pouvez réutiliser les fichiers `PGM` donnés lors du premier projet en plus des images présentées dans cet énoncé (Fig. 1 – `antilope.ppm`, `antilope_gris.pgm`, `antilope_monochrome.ppm`, `antilope_nb.pbm`, `antilope_negatif.ppm` et `antilope_retournement.ppm`).

Toute question relative au projet peut être posée sur le forum de la page web du cours (eCampus).

Tout projet ne compilant pas se verra attribuer la note de 0/20.

Tout étudiant ne soumettant pas à minima une archive vide se verra attribuer une note d'absence.

Il est impératif de respecter **scrupuleusement** les consignes sous peine de se voir attribuer une note de 0/20 pour non respect de l'énoncé.