IMAC
INGÉNIEUR
ESIPE

Projet C

IMAC 1

—2018 - 2019—

Traitement d'images

Ce projet vous permettra d'aborder quelques notions clés utilisées par les logiciels classiques de traitement d'images comme Photoshop ou Gimp, indispensables de nos jours pour l'édition multimédia. Ce projet, à réaliser par binôme, a donc pour but de vous familiariser avec d'une part, quelques outils indispensables du traitement d'images : les LUT, l'histogramme, et d'autre part, l'implémentation en C des images.

---

1 Positionnement du problème

1.1 Introduction

Un logiciel de traitement d'images possède toujours quelques éléments clefs. Pour effectuer des modifications pertinentes à l'image, on exploite fréquemment l'histogramme de l'image finale. Cet histogramme apporte des informations importantes sur l'image comme son niveau de contraste et son exposition. Des Look Up Table (LUT) permettent ensuite de modifier l'histogramme de cette image, permettant ainsi la réalisation "d'effets".

Le but de ce projet est d'implémenter en C ces éléments fondamentaux des logiciels de traitement d'images. des types de données abstraits : File et Liste.

1.2 Image

Une image est composée de pixels (picture element) répartis sous forme de tableau. On attribue à chaque pixel d'une image en niveau de gris, une valeur codée en binaire sur n

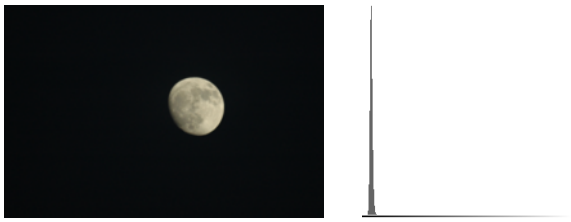

Figure 1: image sombre

Figure 2: image claire

bits. S'il s'agit d'une image couleur, chaque pixel a trois composantes (ou trois valeurs) : une composante rouge, une composante verte et une composante bleue, chacune codée sur n bits. A partir de ces trois composantes, il est possible de représenter plus ou moins bien la quasi-totalité du spectre visible.

Dans le cadre de ce projet, les images seront en couleurs et chaque niveau de gris sera codé sur un octet soit huit bits. Les valeurs de niveaux de gris varieront donc de 0 à $2^8 - 1 = 255$. Pour le stockage dans des fichiers, nous utiliserons le format PPM détaillé dans le paragraphe 4.1. Une des difficulté de ce projet sera la gestion distincte des 3 canaux de couleur rouge, vert et bleu. En effet, les LUT que nous verrons au paragraphe 1.4 ne s'applique que sur une *image en niveau de gris.*

1.3 Histogramme

1.3.1 Définition

Un histogramme est un ensemble de données statistiques permettant de représenter la distribution des intensités lumineuses des pixels d'une image, c'est-à-dire le nombre de pixels pour chaque niveau de gris. Pour une image en couleur, il est possible soit de faire un histogramme par composante (rouge, vert et bleue), soit de faire l'histogramme de la moyenne des trois composantes pour chaque pixel.

1.3.2 Intérêt

Un histogramme donne des informations sur la distribution des couleurs dans une image. Il permet par exemple de savoir si une image est plutôt sombre lorsque les piques se trouvent plutôt vers la gauche (figure 1). Pour une image plutôt claire, les piques de l'histogramme se trouveront vers la droite (figure 2).

Si la plupart des niveaux de gris sont situés vers le milieu de l'histogramme, cela signifie que l'image n'est pas très contrastée (figure 3). Enfin, si les niveaux de gris sont situés sur les extrèmes de l'histogramme, cela peut laisser penser que le contraste est trop fort (figure 4).

1.3.3 construction d'un histogramme

Pour constuire un histogramme, il faut d'abord créer un tableau H de taille k où k est le nombre de niveaux de gris représentables sur une image. Les valeurs de H doivent être

initialisées à 0. Il suffit alors de parcourir chaque pixel de l'image et d'ajouter 1 dans la case correspondant au niveau de gris du pixel traité.

1.4 Look-Up Table (LUT)

Une *look-up table* est un tableau permettant d'effectuer des transformations sur les couleurs d'une image. A chaque niveau de gris défini dans l'image de départ est associé un niveau de gris d'arrivée. Une *look-up table* n'est donc pas nécessairement bijective et deux niveaux de gris de départ peuvent se voir attribuer le même niveau de gris d'arrivé. D'ailleurs, le nombre de niveaux de gris définis dans l'image de départ n'est pas forcément égal à celui défini dans l'image d'arrivée. L'utilisation de *look-up tables* est courante dans les logiciels de traitement d'images car elle permet de combiner plusieurs opérations sur une image sans la déteriorer au cours du traitement.

Dans notre projet les LUT seront utilisées pour appliquer des effets à l'image. En effet, l'application d'une LUT sur une image peut permettre d'augmenter ou de diminuer la luminosité, d'augmenter ou de diminuer le contraste, faire un effet sépia ... De plus, combiner n LUT est très rapide puisqu'il suffit de combiner l'effet des n tables au lieu d'appliquer n fois les LUT sur n images. Le gain de temps est considérable. Ainsi, dans notre application, la combinaison de plusieurs LUT devra être calculée dans une seule LUT ... finale.

2 Travail demandé

Le but du projet est d'implémenter en langage `C` un petit programme de manipulation d'images. Il devra se conforter aux exigences suivantes :

2.1 Structure du projet

Dans ce projet complexe, il est hors de question de n'utiliser qu'un seul fichier contenant toutes les fonctions nécessaires à l'application ! Il vous faut donc séparer les traitements en différents fichiers `.c` et `.h`. La découpe des fichiers est laissée à votre appréciation mais doit être logique. Afin de compiler le projet, un `Makefile` devra être réalisé.

Note importante : Tout `#include "unfichier.c"` ou tout rendu de projet sans `Makefile` entrainera une note sanction (probablement directement un 0).

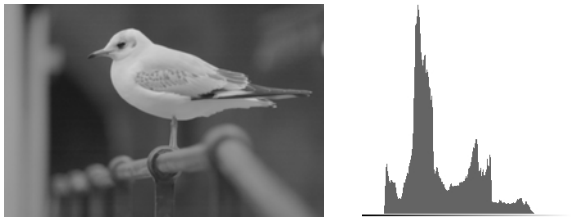

Figure 3: pas assez de contraste

Figure 4: contraste trop fort

Le programme sera donné sous la forme d'un exécutable nommé `minigimp`. Il prendra en ligne de commande plusieurs arguments, dont une et une seule image (voir section 2.3.3). Toutes les options ainsi que le nom de l'image à traiter seront spécifiées sur la ligne de commande. Votre programme devra compiler et fonctionner sous Linux et plus spécifiquement sur les machines de la fac.

Enfin votre projet sera mis dans un repertoire nommé aux noms des auteurs. Dans celui ci vous suivrez la structure suivante :

```
Nom1Nom2/
  \-- bin/
  \-- src/
  \-- images/
  \-- doc/
  \-- Makefile
```

Le répertoire `bin` contient l'exécutable `minigimp`. Le répertoire `src` contient les fichiers d'entête `.h` et les fichiers sources `.c` de votre programme. Le répertoire `images` contient tous les `.ppm`. Vous pouvez considérer que toutes les images qu'utilisera votre application sont et doivent être incluses dans ce répertoire. Le répertoire `doc` contiendra toute la documentation, fichier README, rapport... Enfin un `Makefile` permettera de compiler `minigimp`.

2.2 Structures de données

Sur ce projet, vous **devez** utiliser les structures de données suivantes :

- structure image contenant un tableau de pixels (RGB) ainsi que les dimensions de l'image.
- tableaux contenant des LUT.

2.3 Spécifications demandées

Votre programme devra tout d'abord respecter les contraintes énoncées aux paragraphes 2.1 et 2.2. De plus, il devra s'efforcer d'implémenter les spécifications décrites dans les sous paragraphes suivants.

2.3.1 Coté Image

L'application doit

1. Charger une image PPM,
2. Sauvegarder l'image finale PPM.

Pour le chargement, il se fait en ligne de commande.

La sauvegarde de l'image finale se fait en partant de l'image initiale sur laquelle on applique l'ensemble des LUT. On obtient alors l'image finale. Celle-ci est ensuite sauvegardée dans le répertoire `images`. Le nom de l'image finale est soit fixée à l'avance, soit demandée à l'utilisateur.

2.3.2 Coté LUT

L'application devra pouvoir ajouter des LUT qui sont listées ci-dessous. Chacune de ces LUT possède un code indiqué entre parenthèses.

- augmentation de luminosité (ADDLUM), dépend d'un paramètre,
- diminution de luminosité (DIMLUM), dépend d'un paramètre,
- augmentation du contraste (ADDCON), dépend d'un paramètre,
- diminution du contraste (DIMLUM), dépend d'un paramètre,
- inversion de la couleur (INVERT),
- effet sépia (SEPIA), peut dépendre d'un ou plusieurs paramètres.

L'ajout d'une LUT se fera toujours en fin de liste de LUT. L'application d'une LUT à une image est détaillée dans le paragraphe 1.4.

2.3.3 Coté IHM

En ce qui concerne l'exécution de votre programme sur la ligne de commande, celle ci devra être de la forme :
`$ minigimp mon_image.ppm [-h] [-histo] [<code_lut>[_<param1>]*]* [-o image_sortie.ppm]`
L'application prend donc obligatoirement une image qui sera l'image source. De plus, votre application peut insérer une ou plusieurs LUT. Les crochets `[]` indiquent un bloc. Le caractère `*` signifie que le bloc précédent peut être répété aucune, une ou plusieurs fois. Enfin `<code_lut>` doit être remplacé (y compris les `<` et `>`) par le code de la LUT (voir section 2.3.2). L'option `-histo` demande au programme de générer l'histogramme de votre image. Cet histogramme peut être soit affiché sur le terminal (pas terrible), soit être sauvegardé dans une image (il faut alors construire cette image). Enfin l'option `-o` permet de spécifier l'image de sortie qui sera l'image initiale sur laquelle s'applique l'ensemble des LUT.
Par exemple :
`$ ./minigimp vacances.ppm SEPIA 5 ADDLUM 20 -o super-vacances.ppm`
appliquera à l'image `$vacances.ppm` les LUT `$SEPIA` avec le paramètre '5' et `$ADDLUM` avec le paramètre 20, puis sauvegardera le résultat dans le fichier `$super-vacances.ppm`.

2.4 Rapport

Vous devrez justifier le choix de vos méthodes dans un rapport d'environ 5-10 pages (pas plus). Ce rapport est fait pour montrer ce que vous avez compris de votre projet. Il doit montrer que vous savez prendre du recul par rapport à votre travail. Voici quelques règles à suivre pour mieux orienter son contenu :

- Ne perdez pas de temps à réexpliquer le sujet du projet, l'enseignant le connait déjà, faites seulement un bref résumé de quelques lignes. De manière plus générale, ne détaillez pas des méthodes déjà expliquées dans l'énoncé à moins que vous les ayez modifiées.
- Un rapport sert surtout à montrer comment vous avez fait face aux problèmes (d'ordre algorithmique). Certains problèmes sont connus (on en parle dans l'énoncé), d'autres sont imprévus. Montrez que vous les avez remarqués et compris. Donner la liste des solutions à ce problème et indiquez votre choix. Justifiez votre choix (vous avez le droit de dire que c'est la méthode la plus facile à coder).
- Il ne doit figurer aucune ligne de code dans votre rapport. Un rapport n'est pas un listing de votre programme où vous détaillez chaque fonction. Vous devez par contre détaillez vos structures de données et mettre du pseudocode pour expliquer vos choix algorithmiques. Il est autorisé d'utiliser des "racourcis" tels que "`initialiser le tableau tab à 0`" plutôt que de détailler la boucle faisant la même chose.
- N'hésitez pas à mettre des images dans votre rapport pour illustrer vos propos et vos résultats.
- Enfin, il est **très important** de faire la liste de ce que vous avez fait, de ce qui fonctionne correctement et de la liste des disfonctionnements. Précisez quand il s'agit d'options que vous avez rajoutées en plus de ce qui était demandé.

3 Notation

Le projet sera évalué en considérant les éléments suivants :

- le fonctionnement du programme (fait-il ce qui est demandé?).
- le rapport (la moitié de la note, ne le négligez pas).
- la qualité du code (nom des variables et fonctions, commentaires).

4 Documentation technique

4.1 **fichier**.*ppm*

Il s'agit d'un format d'images reconnu par divers éditeurs d'images comme *gimp* ou *Photoshop*. Il en existe plusieurs variantes : couleurs/niveaux de gris, ASCII/binaire. Le format le mieux

adapté à notre problème est le format couleur en binaire. Le fichier commencera donc avec l'en-tête correspondant.

Un pixel en couleur est défini par 3 nombres : un pour "la quantité" de rouge (R), un pour "la quantité" de vert (V) et un pour "la quantité" de bleu (B), noté RVB (RGB en anglais). Nous coderons ces quantités par des `unsigned char`. Un `unsigned char` est codé sur 8 bits, soit un octet. Ces nombres vont donc de 0 à 255.

Revenons au format *ppm*. Le premier nombre correspond au "numéro de variante" choisi, dans notre cas *P*6. Sur la ligne suivante, les deux nombres correspondent à la *largeur* et la *hauteur* de l'image. Enfin, le dernier nombre correspont à la valeur maximale que l'on puisse rencontrer (ici 255). Des commentaires peuvent être insérés dans l'en-tête, une ligne de commentaires commence par un #.
Chaque pixel sera écrit dans le fichier en procédant ligne par ligne, de gauche à doite, en commençant par le haut. Les pixels seront écrits sous forme de 3 octets écrits les uns à la suite des autres (RVB).

Exemple:

```
P6
#commentaires eventuels
128 256
255
^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y^@y
^@y^@y^@y ...
```

N'hésitez pas à discuter avec votre enseignant pour avoir plus de détails.

5 Options & Bonus

Il est conseillé de rajouter à votre programme quelques bonus et options :

- Sauvegarder sous la forme d'une image PPM, l'histogramme de l'image initiale et finale,
- Implémenter d'autres LUT,
- Faire un historique,
- Gérer les calques,
- ...

6 Conseils et remarques

- Ce sujet de projet constitue un cahier des charges de l'application. Tout changement à propos des spécifications du projet doit être validée par l'enseignant.
- Le temps qui vous est imparti n'est pas de trop pour réaliser l'application. N'attendez pas le dernier moment pour commencer à coder.
- Il est très important que vous réfléchissiez **avant de commencer à coder** aux principaux modules, algorithmes et aux principales structures de données que vous utiliserez pour votre application. Il faut également que vous vous répartissiez le travail et que vous déterminiez les tâches à réaliser en priorité.
- Ne rédigez pas le rapport à la dernière minute sinon il sera baclé et cela se sent toujours.
- N'oubliez pas de **tester** votre application à chaque spécification implémentée. Il est impensable de tout coder puis de tout vérifier après. Pour les tests, confectionnez vous tout d'abord de petites images (taille 5 par 5 par exemple) extrêmement simple avec 1 seul calque et pas de LUT.
- votre enseignant est là pour vous aider. Si vous ne comprenez pas un algorithme ou avez des difficultés sur un point, n'attendez pas la soutenance pour nous en parler. Passez plutôt voir votre enseignant.
- Ce document peut changer. Je vous dirais lorsque de nouvelles versions dûes à d'inévitables précisions ou corrections apparaitront.

Bon courage.