IMAC 1 - Projet Algorithmique en C- 2015-2016

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 a donc pour but de vous familiariser avec d'une part, quelques outils indispensables du traitement d'images : les calques,les LUT, les histogrammes, l'historique, et d'autre part, l'implémentation en C des images.
Le programme utilisera une bibliothèque graphique mise à votre disposition.
Ce projet sera à réaliser en binôme.

1 Positionnement du problème

1.1 Introduction

Un logiciel de traitement d'images possède toujours quelques éléments clefs :

- Tout d'abord, les **calques** permettent, comme leur contrepartie réelle, d'ajouter différentes couches de couleur sur un canevas. Chaque calque a la même dimension, et la même résolution. Ces calques se combinent pour créér l'image finale.
- D'autre part, pour effectuer des modications pertinentes sur 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, dans le but de réaliser des effets.
- Enfin un bon logiciel d'édition d'image offre toujours la possibilité de **défaire et de refaire les actions de l'utilisateur.**

Le but de ce projet est d'implémenter en C ces quatre éléments fondamentaux des logiciels de traitement d'images. Cette implémentation utilisera les types de donneés *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 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-totalite 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 detaillé dans le paragraphe 3.1.
Une des difficultés 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 1.5 ne s'applique que sur une *image en niveaux de gris.*

1.3 Les calques

Dans un logiciel de traitement d'images, les calques peuvent se voir comme un ensemble de couches de couleur que l'on additionne ou mélange entre elles pour obtenir l'image finale. Il n'est ainsi pas rare de trouver des images contenant plus d'une centaine de calques. En règle générale, les calques possèdent exactement la même dimension en largeur et hauteur (ce sera le cas pour notre projet).
Chaque calque représente une image en couleur. De plus, chaque calque definit la manière dont il se mélange avec le calque situé en dessous de lui. Pour le calque initial, on peut considérer qu'il existe un calque "virtuel", complètement blanc, situé en dessous de lui. De plus, chaque calque dispose d'un parametre d'opacité , compris entre 0 et 1 et qui définit la transparence de ce calque (1 = opacité totale / 0 = transparence totale).
Dans le cadre de ce projet, chaque calque sera donc une image couleur de la même dimension que l'image initiale. Chaque calque aura son paramètre d'opacité α. De plus, chaque calque définira s'il s'additionne ou se multiplie avec le calque situé en dessous de lui.
Dans le cadre de l'addition, si C^f_{n-1} et C_n sont deux pixels de même position sur le calque courant n et le calque du dessous n - 1, alors la couleur C^f_n de l'image finale (jusqu'au calque n) se calcule par la formule :

$$C^f_n = C^f_{n-1} + \alpha * C_n$$

Notez bien que C^f_{n-1} est bien la couleur finale calculée pour le calque n - 1.

Dans le cas de la multiplication, la formule devient :

$$C^f_n = (1 - \alpha) * C^f_{n-1} + \alpha * C_n$$

1.4 Histogramme

1.4.1 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.

1.4.2 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 pics se trouvent plutôt vers la gauche (figure 1). Pour une image plutôt claire, les pics 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 tres contrastée (figure 3). Enfin, si les niveaux de gris sont situés sur les extrêmes de l'histogramme, cela peut laisser penser que le contraste est trop fort (figure 4).

Figure 1: image sombre

Figure 2: image claire

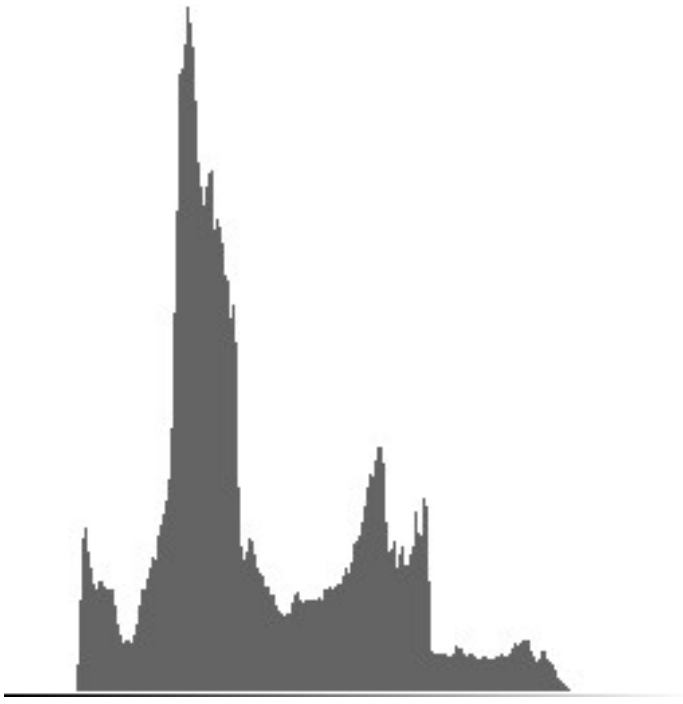

Figure 3: pas assez de contraste

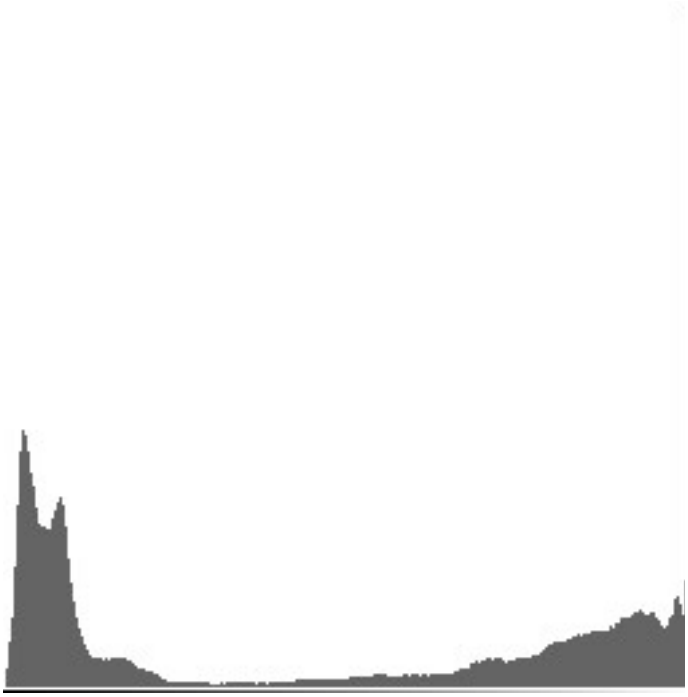

Figure 4: contraste trop fort

1.4.3 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 différents. Les valeurs de H doivent être initialisées a 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.

1.5 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 necessairement bijective et deux niveaux de gris de départ peuvent se voir attribuer le même niveau de gris d'arrivée. D'ailleurs, le nombre de niveaux de gris défini 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 détériorer 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 sepia...

De plus, combiner *n* LUT est très rapide puisqu'il sufft 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, chaque calque se verra attribué une liste de LUT.

1.6 Historique des modifications

La gestion de l'historique des modifications est loin d' être évidente. La méthode la plus efficace consiste à conserver les données sources, et enregistrer toutes les modications faites par l'utilisateur. L'idée ici est que si l'on est suffisamment rapide pour reproduire l'ensemble des modications en temps réel, alors on peut, à la volée, présenter le résultat final à l'utilisateur.

Dans notre cas, on peut effectivement faire une telle supposition. Nous devrons donc prendre bien soin de conserver, pour chaque calque, l'image source, et, toujours pour chaque calque, la liste des traitements (qui est la liste des LUT). Si l'utilisateur souhaite voir le résultat final, il faudra le calculer à la volée (éventuellement ce résultat peut être stocké temporairement - il est valable tant qu'aucune modication ne s'est produite). Les modications doivent être enregistrées. Pour ce faire, il faut énumérer toutes les modications possibles, ainsi que les paramètres associés. Un ensemble de ces éléments peut se trouver aux paragraphes 2.3.5 et 3.2.

2 Travail demandé

Le but du projet est d'implémenter en langage C un petit programme de traitement 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.
Le programme sera donné sous la forme d'un exécutable nommé *imagimp*. Il pourra prendre, sur la ligne de commandes, plusieurs arguments, dont une et une seule image (voir section 2.3.4). Toutes les options ainsi que le nom de l'image à traiter seront specifiées soit sur la ligne de commande, soit via l'IHM. Votre programme devra compiler et fonctionner sous Linux et plus spécifiquement sur les machines de l'Université.
Enfin votre projet sera mis dans un répertoire nommé aux noms des 2 binômes.
Dans celui ci vous suivrez la structure suivante :

```
Nom1_Nom2/
    bin/
    doc/
    images/
    lib/                    les librairies utilisées
    src/
    Makefile
```

Le répertoire *bin* contient l'executable *imagimp*.
Le répertoire *src* contient les fichiers d'entête .h et les fichiers sources .c de votre programme.
Le répertoire *lib* (optionnel) contiendra les fichiers *.a* ou *.so* de vos bibliothèques ainsi que tout le nécessaire pour les compiler :

- un *Makefile*
- un répertoire *include* contenant les .h
- un repertoire *src* contenant les .c.

Le répertoire images contient tous les .ppm. Vous pouvez considerer que toutes les images qu'utilisera votre application sont et doivent être incluses dans ce repertoire.
Le répertoire *doc* contiendra toute la documentation :

- fichier *README*

rapport

...

Enfin un *Makefile* permettra de compiler imagimp.

2.2 Structures de données

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

a) Pour les calques, une liste doublement chaînée.

Voici les informations importantes que devrait avoir votre structure de données (attention, ce n'est pas exhaustif) :

calque précédent et suivant

image source

paramètre d'opacité,

type d'opération de mélange,

liste de LUT...

b) Pour l'ensemble des LUT, une liste (circulaire, simplement ou doublement chaînee à votre guise) par calque.

Voici les informations importantes que devrait avoir votre structure de données (attention, ce n'est pas exhaustif) :

LUT suivante

tableau représentant la LUT courante

information concernant les canaux...

c) Pour l'historique, une pile.

d) Notez que vous risquez d'avoir besoin de structures de données pour d'autres représentations informatiques comme les images, l'historique, les canaux...

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. Chaque spécication est munie d'un code indiqué entre parenthèses.

2.3.1 Côté Image

L'application doit

1. Charger une image PPM (IM_1),
2. Sauvegarder l'image finale PPM (IM_2).

Pour le chargement, il se fait soit en ligne de commande, soit via l'IHM. Si l'image est chargée en ligne de commande (au lancement du programme donc), alors l'image occupe le premier layer. Si elle est chargée via l'IHM, elle écrase l'image du calque courant.

La sauvegarde de l'image finale se fait en partant de l'image du calque 0. A chaque calque, on applique tout

d'abord l'ensemble des LUT sur l'image courante du calque. Puis on réalise l'operation de mélange avec le calque précédent. On avance ensuite d'un cran jusqu'au dernier calque où l'on obtient alors l'image finale. Celle-ci est ensuite sauvegardee dans le répertoire images. Le nom de l'image finale est soit fixée à l'avance, soit demandée a l'utilisateur.

2.3.2 Côté Calque

1. Ajouter un calque vierge (CAL_1),
2. Naviguer dans les calques (CAL_2),
3. Modfier le paramètre d'opacité d'un calque (CAL_3),
4. Modifer la fonction de mélange du calque (addition/ajout) (CAL_4),
5. Supprimer le calque courant (CAL_5).

L'ajout d'un calque vierge est l'insertion d'un calque multiplicatif d'opacité nulle et dont l'image est une image remplie de pixel blanc pur. Cette insertion s'effectue a la fin de la liste des calques.

A tout moment, l'utilisateur doit savoir dans quel calque il se trouve. L'application devra permettre à l'utilisateur de naviguer entre les calques. De fait, le calque sélectionné est celui ou l'utilisateur se trouve.

La modication du paramètre d'opacité se fera sur le calque courant.

De même la modification de la fonction de mélange se fera sur le calque courant. A moins

que vous n'implémentiez de nouveaux modes de mélange, il s'agit d'un simple switch entre mélange additif et soustractif.

La suppression est impossible s'il ne reste plus qu'un calque.

2.3.3 Côté LUT

1. Ajouter une LUT (LUT_1),
2. Appliquer une LUT a une image (LUT_2),
3. Supprimer la derniere LUT (LUT_3).

Les LUT que l'application devra pouvoir ajouter sont listées ci-dessous. Chacune de ces modifications possède un code indiqué entre parentheses.

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 parametre

inversion de la couleur (INVERT),

effet sepia (SEPIA), peut dépendre d'un ou plusieurs paramètres.

L'ajout d'une LUT se fera toujours en fin de liste de LUT pour un calque donné. L'ajout se fait sur le calque courant.

L'application d'une LUT à une image est detaillee dans le paragraphe 1.5.

La suppression d'une LUT est effectuée a la fin de la liste. Elle se fait sur le calque courant.

2.3.4 Côté IHM

L'application s'appuiera sur une bibliothèque permettant d'acher une image. Elle permettra les actions suivantes :

1. Changer de mode de vue (IHM_1)
2. Permettre le chargement d'une premiere image en ligne de commande (IHM_2)
3. Permettre l'application d'effets sur ce premier calque en ligne de commande (IHM_3)
4. Sortir de l'application (IHM_4).

Votre application devra avoir deux modes de visualisation.

Le premier mode permet de voir l'image source du calque selectionné.

Le second permet de voir l'image finale.

En ce qui concerne l'exécution de votre programme sur la ligne de commande, celle ci devra être de la forme :

$ imagimp mon_image.ppm [<code_lut>[_<param1>]*]*

L'application prend donc obligatoirement une image qui sera mise sur le premier calque.

De plus, votre application peut insérer, directement, sur ce calque, une ou plusieurs LUT.

Les crochets [] indique un bloc. Le caractere * 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 2.3.3).

Votre application devra permettre à l'utilisateur d'en sortir, en appuyant sur une touche ou via l'IHM. Cette sortie devra être propre (désallocation de la memoire et vérification avec valgrind).

2.3.5 Côté historique

1. Achiffer l'historique des modifications,
2. Annuler la dernière opération si possible.

L'affichage de l'historique des modifications se fera dans le terminal. A minima la liste des codes doit être affichées. L'ensemble des opérations à enregistrer est :

IM_1,

CAL_1, CAL_3, CAL_4, CAL_5,

LUT_1, LUT_3

En fait IM_2 et LUT_2 correspondent à la génération de l'image finale et ne sont donc pas des opérations proprement dites (de même pour le changement de mode de vue).

La gestion de l'historique se fait par une pile. On ne peut donc défaire que la derniere action réalisée.

Les actions annulables sont : CAL_1, CAL_3, CAL_4, LUT_1, LUT_3.

Tout ce qui a été fait sur la ligne de commandes n'est, à priori, pas annulable. Si la dernière action n'etait pas annulable alors l'annulation est inopérante.

3 Documentation technique

3.1 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'entête correspondant.
Un pixel en couleur est défini par 3 nombres :

un pour "la quantite" de rouge (R)
un pour "la quantite" de vert (V)
un pour "la quantite" 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 a 255.
Format ppm.

Le premier nombre correspond au "numero de variante" choisi, dans notre cas P5 pour les images en noir et blanc et P6 pour celles en couleurs.
Sur la ligne suivante, les deux nombres correspondent à la largeur et à la hauteur de l'image.
Le dernier nombre correspond à la valeur maximale que l'on puisse rencontrer (ici 255).
Des commentaires peuvent être inserés dans l'en-tête.
Une ligne de commentaires commence par un #.
Les lignes suivantes représentent la liste des pixels, à commencer par la première ligne de l'image. Un nombre par pixel pour les images N&B ou grises, trois nombres par pixels pour les images couleur.

Exemple:

P6
#commentaires éventuels
128 256
255
3 67 34 233 22 88 255 0 0 …....

3.2 Quelques éléments sur l'historique

La gestion de l'historique peut être assez difficile. Ainsi pour la gestion des LUT, il est important de savoir exactement quel type de LUT a été inséré (ou supprimé). La nomenclature présente dans 2.3 n'est pas à priori suffisante pour gérer une liste d'undo/redo.
Chaque modification doit donc être référencée dans la pile des modications. Cela implique, non seulement de savoir quel type d'action a été realisé, mais également avec quels paramètres, quelles LUTs etc...
Votre structure de données devra refléter cette contrainte.
Par exemple, on peut tout à fait stocker, en plus, dans un élément de l'historique l'opération inverse de celle représentée par cet élément. Cela peut permettre de gérer par exemple les suppressions.

4 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
- Faire un redo (stockage des opérations annulées)
- Autoriser le undo pour les opérations IM_1 et CAL_5
- Implémenter d'autres LUT
- Navigation sur les LUT d'un calque, avec ajout et suppression de la LUT courante
- Mini application de dessin (suivant développement de la bibliothèque graphique)

  …

5 Conseils et remarques

Ce sujet de projet constitue un cahier des charges de l'application.
Le temps imparti n'est pas de trop pour réaliser l'application.
Ne pas attendre le dernier moment pour commencer à coder.

Ne pas perdre trop de temps sur la gestion de la saisie des actions utilisateurs (scanf), c'est souvent un point délicat et ce n'est pas sur ce point que portera l'évaluation du projet.
Ne pas hésiter à contourner l'obstacle en mettant "en dur" (c.a.d. directement dans le code) les éléments à faire saisir a l'utilisateur (par exemple les noms de fichier).
Il est aussi possible d'utiliser la ligne de commandes.

Avant de commencer à coder, il est très important de réfléchir aux principaux modules, algorithmes et aux principales structures de données à utiliser pour l' application . Il faut également répartir le travail et déterminer les tâches à réaliser en priorité.

Ne pas rédiger le rapport à la dernière minute sinon il sera bâclé et cela se sent toujours.

Le projet est à faire par binôme. Il est impératif que chacun d'entre vous travaille sur une partie et non pas tous en même temps sinon vous n'aurez pas le temps de tout faire.

Développer vos propres fonctions sur les listes et les piles.

Ne pas oublier de tester l' application à chaque spécification implémentée. Il est impensable de tout coder puis de tout vérifier après.
Pour les tests, créer tout d'abord de petites images (taille 5 par 5 par exemple) extrêmement simples avec 1 seul calque et pas de LUT.