École d'ingénieur ESIPE-IMAC
Université Paris-Est Marne La Vallée
Boulevard Copernic
77420 Champs-sur-Marne

RAPPORT DE PROJET

Programmation C

Traitement d'images

Par **Elise RITOUX** (TD2) et **Marion VILA** (TD2)
Année universitaire : 2015/2016
IMAC 1

Sommaire

Introduction

Le projet consiste à réaliser un mini logiciel de traitement d'images, à l'instar de Photoshop, en utilisant une bibliothèque graphique mise à disposition. Nous avons fait une programmation multi-fichiers pour faciliter le travail de développement en équipe.

Analyse fonctionnelle générale

Lorsque le programme démarre, il demande à l'utilisateur de rentrer l'image qu'il souhaite afficher. Cette action ouvrira la fenêtre graphique, et déterminera la largeur et hauteur des calques qui suivront.
Un menu apparaît ensuite, dans la console et dans l'IHM, pour indiquer à l'utilisateur quelles actions il peut réaliser sur cette image. Ces actions s'effectuent sur le calque affiché, qui sera nommé calque courant.

rouge +	annuler
rouge -	sav
vert +	histo
vert -	infos
bleu +	opacite
bleu -	melange
sepia	supr calque
noir/blanc	ajout img
inversion	ajout calque
contraste +	img finale
contraste -	lum +
supr LUT	lum -

Au niveau des calques, on peut rajouter un calque vierge, supprimer le calque courant, modifier l'opacité et le mélange du calque, c'est à dire définir si c'est une addition ou une multiplication. L'utilisateur peut également afficher les informations de tous les calques, ainsi que se déplacer entre eux au moyen des flèches directionnelles.
L'utilisateur peut également afficher l'histogramme, le programme crée alors un nouveau calque sur lequel est affiché l'image dont on affiche l'histogramme ainsi que l'histogramme qui devient le calque courant. Le calque dont on affiche les informations devient le calque référent. L'histogramme s'actualise en fonction des actions sur les LUT.
Au niveau des LUT, l'utilisateur peut augmenter ou diminuer la luminosité, le contraste, le taux de rouge, de vert ou de bleu, mettre l'image en sépia, en noir et blanc et inverser les couleurs. Il peut aussi supprimer une ou plusieurs LUT.
L'utilisateur peut également rajouter une nouvelle image sur un nouveau calque, mais celle-ci doit avoir la même hauteur et largeur que la première image chargée. Enfin, il peut afficher l'image finale et avoir la possibilité de la sauvegarder dans le dossier "/images".

Une dernière touche permet de quitter le programme.

Il est également possible de revenir en arrière grâce à la gestion d'un historique, cependant cet historique n'est pas capable de régénérer un calque qui aurait été supprimé par l'utilisateur. Il est donc possible d'annuler les déplacements sur les calques, les ajouts et suppressions de LUTs, l'ajout d'un nouveau calque et les modifications d'opacité ou de mélange.

Structures

Nous allons maintenant présenter les structures des principales fonctionnalités du programme.

Calques : Une liste doublement chaînée de calques permet de naviguer entre eux. Chaque structure de calque contient une image, un lien vers le calque précédent, un lien vers le calque suivant, un paramètre d'opacité (nombre flottant), le type d'opération de mélange, pour savoir si c'est l'addition (0) ou la multiplication (1), le numéro du calque (la liste commence à 0), une liste de LUT et un indice montrant si le calque est un calque d'information (si oui, les pixels blancs sont invisibles)

Images : Une image est composée d'un numéro de variante, P5 pour les images en noir/blanc, P6 pour les images couleurs, d'une hauteur, d'une largeur, d'un tableau de pixels, et d'une valeur maximale.

LUT : Les LUT sont reliées par une liste chaînée. Elles sont composées de trois tableaux pour le rouge, le vert et le bleu et un indice permettant d'identifier la LUT, cela permet de rétablir une LUT qui a été supprimée. Il est aussi utile de savoir quelle modification apporte la LUT afin de déterminer si elle s'applique sur une image en noir et blanc ou non.

Historique : L'historique est constitué d'un int, représentant une action a rétablir, d'un float stockant la valeur précédente et d'un pointeur vers l'action précédente.

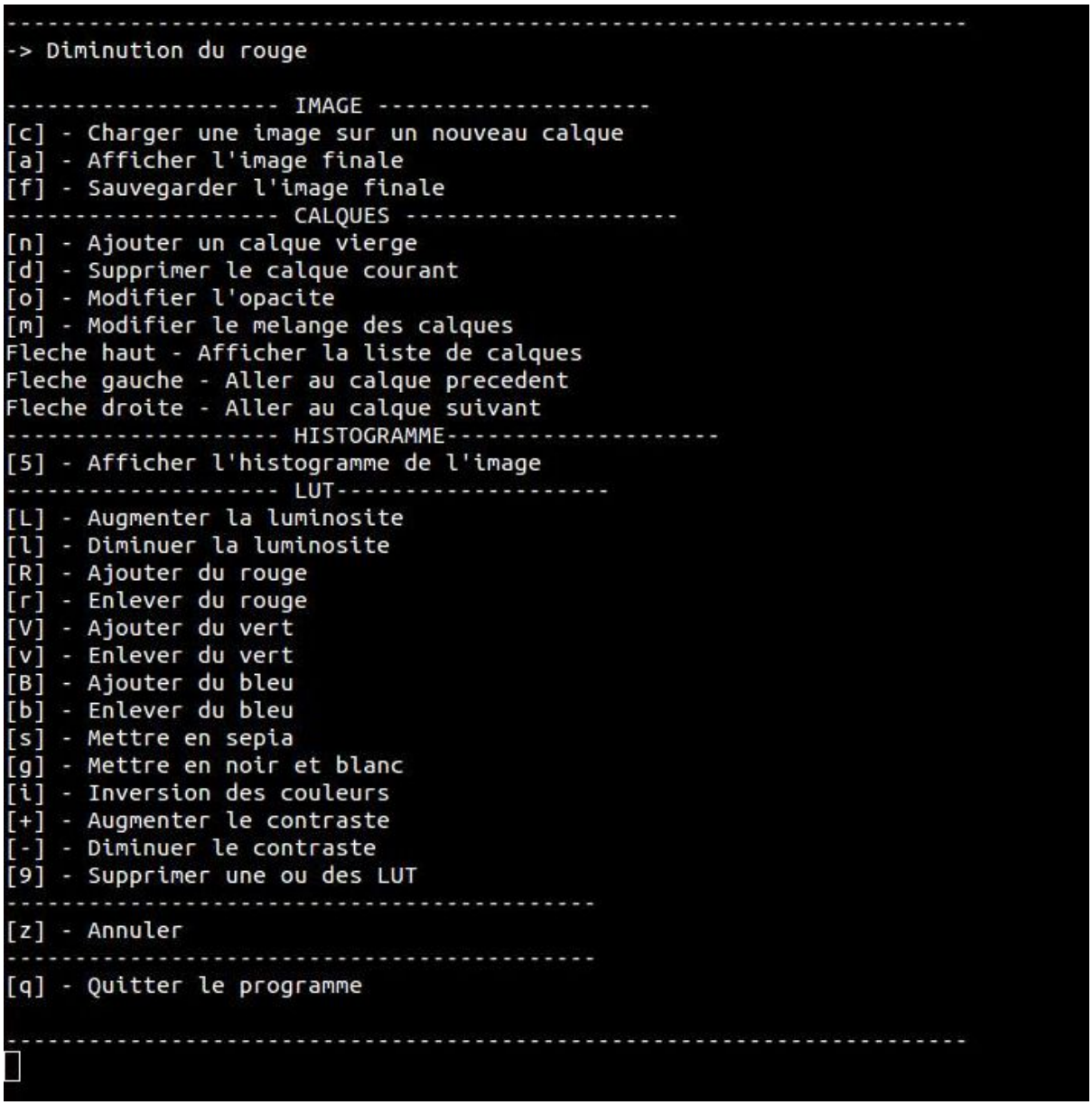
Actions gérées par l'historique :

1 : ajout d'un nouveau calque
2 : changement d'opacité
3 : changement de mélange
4 : ajout d'une LUT
5 : suppression d'une LUT
6 : navigation du calque à gauche
7 : navigation du calque à droite

Pour annuler la suppression d'une LUT, l'action est identifiée par la valeur 5 et l'identifiant de la LUT en question est stockée dans valeurPrecedente.

Analyse détaillée

Dans la fonction principale, nommée programme(), nous utilisons les fonctions de la bibliothèque fournie pour faire l'affichage. Après que la première image soit chargée dans le programme, son tableau de pixels est retourné et envoyé à la fonction initGLIMAGIMP_IHM, qui permettra de l'afficher tout en laissant de la place pour l'interface graphique. A partir de ce moment, l'utilisateur peut effectuer plusieurs actions, à l'aide des fonctions suivantes : fixeFonctionClavier, fixeFonctionClavierSpecial, fixeFonctionDessin et fixeFonctionClicSouris.



Nous allons détailler les fonctions :

POUR LES CALQUES

Calque* ajoutPremierCalque(Image* image, float opacite, int melange);
→ Ajoute le premier calque en début de programme, et alloue la mémoire nécessaire.

Calque* ajoutCalqueVierge(listeCalque listeCalque);
→ Ajoute un calque vierge multiplicatif d'opacité nulle, rempli de blanc, et l'insère a la fin de la liste chaînée. Son numéro est calculé en parcourant la liste de calques. Il devient le calque courant.

Calque* naviguerDansCalque(Calque *calqueCourant, int direction);
→ Cette fonction permet de naviguer dans les calques, c'est à dire de changer le calque courant affiché. La flèche gauche emmène sur le calque précédent, la flèche droite sur le calque suivant, cette information est transmise grâce à la variable direction. Le programme avertit l'utilisateur s'il essaye d'aller sur un calque qui n'existe pas, et ne le déplace pas.

void opaciteCalque(Calque* calque, float opacite);
→ Elle modifie le paramètre d'opacité du calque envoyé.

void melangeCalque(Calque* calque, int melange);
→ Elle modifie la fonction de mélange du calque envoyé, soit addition, soit ajout

int supprimeCalqueCourant(Calque** premierCalque, Calque* calque);
→ Elle supprime le calque envoyé (qui est le calque courant), en fonction de sa position dans la liste chaînée. La suppression est impossible s'il ne reste qu'un seul calque. Elle décrémente également le numéro des autres calques, qui suivent le calque supprimé. L'indice renvoyé permettra de définir le prochain calque courant. Le calque **premierCalque sert à supprimer le calque courant, si celui-ci est le premier calque, afin de ne pas perdre la liste chaînée.

listeCalque supprimeListeCalque(listeCalque listeCalques);
→ Supprime la liste de calque, utilisée en fin de programme.

Image* creationImageFinale(Calque* premierCalque);
→ Cette fonction va créer l'image finale, en prenant en compte tous les calques de la liste chaînée. Elle alloue de la place pour l'image, crée un tableau de pixels selon le premier calque en appliquant les LUT. Elle parcourt ensuite les prochains calques de la liste à l'aide d'un calque temporaire, en continuant d'appliquer les LUT et en utilisant une formule selon si le mélange est addition ou multiplication. Elle retourne finalement l'image.

Formule pour l'addition : $C^f_n = C^f_{n-1} + \alpha * C_n$
Formule pour la multiplication : $C^f_n = (1 - \alpha) * C^f_{n-1} + \alpha * C_n$
α représente l'opacité, C_n le calque temporaire, C^f_{n-1} les calques déjà faits.

Dans la fonction programme(), une image temporaire est utilisée pour recevoir l'image finale, c'est elle qui sera affichée avec la fonction actualiseImage(). C'est pour cela que l'appui sur une autre touche l'enlève.

int afficheListeCalques(listeCalque listeCalques, Calque* calqueInitial);

→ Affiche la liste et le nombre de calques. Elle utilise la fonction infosCalque pour afficher les infos de chacun.

void infosCalque(Calque *calqueCourant, int indiceCalqueInitial);
→ Donne les infos du calque envoyé, comme son numéro, son opacité, son mélange et précise si c'est le calque courant.

unsigned char* creeImageAvecInfo(Calque *calque_courant, Image *img);
→ Si le calque courant contient des informations graphique à afficher (historique), il récupère l'image du calque précédent à laquelle il ajoute les infos du calque courant.

POUR LES IMAGES

int saisieUtilisateurImage(Image* img, int initialisation, unsigned int largeurInitial, unsigned int hauteurInitial);
→ Cette fonction demande à l'utilisateur de saisir le nom d'une image, et change le texte affiché si ce n'est pas la première fois qu'il charge une image. L'utilisateur peut annuler cette action en tapant "e". La fonction chargerImage() est ensuite appelée.
L'image occupera le premier calque la première fois qu'elle est appelée, et un nouveau calque en fin de liste par la suite.

int chargerImage(Image* img, char* nom, int initialisation, unsigned int largeurInitial, unsigned int hauteurInitial);
→ Cette fonction charge une image à partir d'un fichier .ppm. Elle vérifie d'abord que le fichier existe (c'est à dire que l'utilisateur ait rentré un bon nom d'image), puis charge dans l'ordre les informations existantes dans le fichier .ppm, comme le numéro de variante, la hauteur et largeur de l'image (si l'utilisateur a déjà chargé une image, et que ces informations sont différentes de l'image initiale, alors l'opération est annulée), et récupère les pixels dans un tableau tabPixel. On vide ensuite le buffer et on ferme le fichier.

int sauvegarderImage(Image* img, char* nomImg);
→ Cette fonction sauvegarde l'image en ouvrant un fichier avec les droits d'écriture, et écrit dans l'ordre le numéro de variante, la hauteur, la largeur, les pixels à partir du tableau tabPixel, et ferme ensuite le fichier, nommé selon la chaîne de caractères nomImg.
Le nom est fixé par l'utilisateur.

int retourneImage(Image *img, unsigned char *tab);
→ Cette fonction retourne le tableau de pixels d'une image, pour qu'elle soit affichée à l'endroit dans la fenêtre graphique.

int libererImage(Image* img);
→ Cette fonction libère la mémoire occupée par une image

POUR L'HISTOGRAMME

int couleurHisto();

→ Propose à l'utilisateur de choisir quelle couleur va représenter l'histogramme (le rouge, le vert, le bleu ou alors sa globalité)

float* creationHisto(Calque *calqueCourant, unsigned char *tabHisto, int couleur);
→ Analyse l'image et retourne un tableau indiquant le nombre de pixels pour chacune des 256 valeurs possibles.

Image *afficherHistogramme (unsigned char *tabHisto, int taille);
→ Retourne une image de l'histogramme

void integreHistoSurImage (Image *histo, Calque *calqueCourant, int couleur);
→ Intègre sur le calque courant l'image de l'histogramme sur fond blanc et de la couleur adapté à la couleur analysée

POUR LES LUT

void ajoutLUT(LUT** listeLUT);
→ Ajoute une lut (LUT_1) à listeLUT.

unsigned char* appliqueLUT(Image *img, LUT **liste_lut);
→ Applique une LUT a une image (LUT_2). Pour le calque courant on applique toutes les LUT une à une. Tant qu'il reste des LUT à appliquer, on associe a chaque pixel de l'image la valeur que la LUT actuelle associe pour cette valeur.

char supprimerDerniereLUT(LUT **listeLUT);
→ Supprime la dernière LUT (LUT_3), renvoie le type de la LUT qui vient d'être supprimée.

void supprimerLUT(LUT **listeLUT, int rep, listeHistorique *historique);
→ Supprime le nombre de LUT souhaité par l'utilisateur et ajoute chaque LUT supprimées à l'historique des actions faites par l'utilisateur.

void supprimerTousLUTsansHistorique(LUT **listeLUT);
→ Supprime tous les LUT d'un calque sans les ajouter à l'historique. C'est utile après avoir appliqué définitivement les LUT au calque, il est nécessaire de supprimer la liste des LUT associée pour que les modifications ne soient pas effectuées deux fois, sans que ces suppressions de LUT ne soient ajoutées à l'historique.

void augmenteLuminosite(LUT** listeLUT, char id);
→ Augmentation de luminosité (ADDLUM) du calque courant. Les valeurs de chaque pixel sont incrémentées de +10. Si elles dépassent 255, elles sont écrêtées à ce seuil. On applique à la LUT créée l'identification fournie en paramètre ('L') afin de pouvoir ensuite la rétablir en cas de suppression.

void diminueLuminosite(LUT** listeLUT, char id);

→ Diminution de luminosité (DIMLUM) du calque courant. On applique le même principe que pour augmenterLuminosite() sauf que les valeurs sont décrémentées de 10 et seuillées à 0.

void augmenteContraste(LUT** listeLUT, float histo[], char id);
→ Augmentation de contraste (ADDCON) du calque courant. Grâce à l'histogramme on repère les valeurs minimales et maximales des pixels de l'image. La moitié supérieure de ses pixels sont incrémentées de 5, tandis que la moitié inférieure est décrémentée de 5.

void diminueContraste(LUT** listeLUT, float histo[], char id);
→ Diminution de contraste (DIMCON) du calque courant. On applique le même principe que pour augmenteContraste mais en inversant les deux moitiés : les pixels de la moitié supérieure sont décrémentées de 5, les autres incrémentés.

void inversionCouleur(LUT** listeLUT, char id);
→ Inversion de couleur (INVERT) du calque courant. On associe à chaque valeur de pixels la valeur maximale possible (255) - la valeur du pixel. Ainsi pour un pixel noir (valeur 0), la valeur associée sera 255, soit un pixel blanc.

void sepia(LUT** listeLUT, char id);
→ Effet Sepia (SEPIA) sur le calque courant. On conserve les valeurs des pixels rouges, mais on divise par deux celles associées aux pixels vert et par quinze celles des pixels bleus.

void gris(LUT** listeLUT, char id);
→ Effet de noir et blanc sur l'image. On associe à chaque pixel leur propre valeur, on n'utilise que le tableau de LUT tabLutR. On saura qu'il s'agit d'une image grise grâce à son identification id et on appliquera donc les modifications sur la moyenne des valeurs r, v et b. Après application de la LUT, les valeurs pour r, v et b ont la même valeur, celle de leur moyenne.

void augmenterRouge(LUT** listeLUT);
void augmenterVert(LUT** listeLUT);
void augmenterBleu(LUT** listeLUT);

void diminuerRouge(LUT** listeLUT);
void diminuerVert(LUT** listeLUT);
void diminuerBleu(LUT** listeLUT);

→ Ajoute ou diminue la teneur en rouge, vert ou bleu de l'image. On incrémente les valeurs des pixels de +5 ou -5.

POUR L'HISTORIQUE

void ajoutListeHistorique(int action, float valeur, listeHistorique liste);
→ Ajoute une action à la pile de l'historique

void annuler (listeHistorique historique, Calque **premierCalque, Calque **calque_courant, Calque **calqueReferent, listeCalque *liste_calques, int *histogramme, unsigned char *tabHisto, Image** imgAffichage);

→ Annule les actions.

- Si un calque a été ajouté, on le supprime.
- Si on a modifié l'opacité, on met la variable opacité du calque à la valeur stockée dans valeurPrécédente
- Si on a modifié le mélange des calques, on la rétablit
- Si on a ajouté une LUT, on la supprime
- Si on a supprimé une LUT, on la rétablie via l'identification stockée dans valeurPrécédente
- Si on a navigué entre les calques, on navigue en sens inverse

Conclusion

Niveau de réalisation demandé et atteint

L'application peut charger une image .ppm (IM_1) et la sauvegarder (IM_2).
On peut ajouter un calque vierge (CAL_1), naviguer dans les calques (CAL_2), modifier le paramètre d'opacité d'un calque (CAL_3), modifier la fonction de mélange du calque (addition/ajout) (CAL_4) et supprimer le calque courant (CAL_5).
L'utilisateur peut ajouter une LUT (LUT_1), appliquer une LUT a une image (LUT_2), supprimer la dernière ou plusieurs LUT (LUT_3). On peut augmenter la luminosité (ADDLUM), la diminuer (DIMLUM), augmenter le contraste (ADDCON), le diminuer (DIMLUM), inverser les couleurs (INVERT) et mettre un effet sépia (SEPIA).

Bonus

Nous avons rajouté quelques bonus au programme :

- Historique pour certaines fonctions
- LUT pour mettre en noir et blanc
- LUT pour gérer augmenter ou diminuer la quantité de rouge, vert ou bleu
- IHM pour la quasi totalité des touches

Bugs non résolus

Malheureusement, nous n'avons pas réussi à régler certains problèmes :

- L'IHM bugge pour les images trop grandes, mais seulement si l'écran de l'ordinateur est trop petit
- Bugs (de segmentation entre autre) changeants et aléatoires selon les machines où le programme est testé
- L'histogramme ne s'affiche pas bien pour les très petites images (128 pixels)
- Il reste un warning par rapport à une variable de l'histogramme

- L’affichage de l’image finale fait fusionner les calques de temps en temps avec leur LUT

A améliorer

Plusieurs points d’améliorations sont possibles, comme régler les bugs rencontrés, et permettre une interface de dessin.