Rapport de Projet C
Traitement d'images Mini-gimp

Code source disponible sur [github]

Guillaume Haerinck
Nicolas Lienart
2018 - 2019

Sommaire

Résumé des fonctionnalités

Mentionnées par le sujet

Implémentés	Dysfonctionnelles	Non implémentés
- Ajout de contraste - Diminution de contraste - Inversion de couleur - Ajout de luminosité - Diminution de luminosité - Sepia - Histogramme final - Histogramme de départ	Aucune	Aucune

Supplémentaires

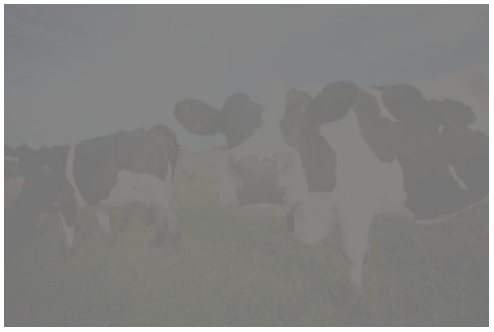
Fonctionnelles	Améliorables
- Image en niveau de gris - Matrice de convolution statique (Contour, Flou Gaussien, Emboss) - Ajout de contraste par fonction sinusoïdale - Égalisation de contraste par histogramme - Affichage de l'Histogramme dans le terminal - Matrice de convolution dynamique par dégradé radial (flou par moyenne) - Transformations (Flip vertical ou horizontal)	- Le flou radial (échelons parfois visibles) - Convolution statique (bordures et taille de matrices pour EMBOSS, EDGE et KBLUR)

Exemple de commandes

```
./bin/minigimp ./images/car.ppm -h EDGE BLUR 25 -o ./output.ppm
```

```
./bin/minigimp ./images/flower.ppm -h SEPIA INVERT ADDCON 50
```

```
./bin/minigimp ./images/no-contrast-cow.ppm -h HISTEQ
```



Identification des tâches

> Il est facile de se faire noyer sous les fonctionnalités et les besoins d'un projet. C'est pour cela que très tôt, nous l'avons découpé en 5 étapes à compléter dans l'ordre pour suivre notre avancée efficacement.

1 - L'interaction homme-machine

Réduite à son état minimal avec l'entrée en ligne de commande, il a cependant fallu prendre en compte la spécification des consignes. La contrainte principale étant la gestion d'un nombre de filtres indéfini, et dans l'ordre où ils sont entrés. Rien qu'une boucle et quelques "if" n'auraient su régler.

Nous avons pensé à créer une interface avec GTK où IMGUI une fois l'étape "Bonus" atteinte, mais nous avons ensuite compris que l'intérêt du sujet résidait dans la manipulation d'image.

2 - La lecture et le stockage d'un fichier

C'est l'étape qui soulève les premières questions d'architecture. Quels formats de PPM autoriser ? Sous quelle forme stocker ces données ? Comment y accéder de façon sécurisée ? Comment gérer les erreurs ?

Comme indiqué plus bas, la réponse à ces questions réside pour beaucoup dans le concept d'**abstraction**. L'idée est d'avoir une interface simple d'utilisation, portable et sécurisée, et dont le coeur s'occupe de modifier les données brutes.

3 - L'application d'effets sur un fichier

Avant d'utiliser les LUTs, comprendre comment manipuler une image était primordial. Les filtres d'inversion, d'ajout et de diminution de luminosité ont été faits très rapidement, ce qui nous a permis d'affiner les fonctions "**getter/setter**" de l'image pour éviter d'accéder à des zones mémoires non allouées.

La suite demandait un travail de recherche et de documentation pour utiliser les formules du contraste et de sépia, qu'il est possible d'implémenter de nombreuses façons. Nous avons sélectionné la formule de sépia conseillée par Microsoft. Pour le contraste nous avons fait plusieurs versions laissant une certaine flexibilité à l'utilisateur.

4 - La compréhension et l'utilisation des LUTs

La gestion des LUTs a été amorcée par la conception de l'histogramme de l'image. Simple à utiliser, il a cependant fallu un temps d'adaptation pour bien comprendre comment implémenter ces fonctionnalités.

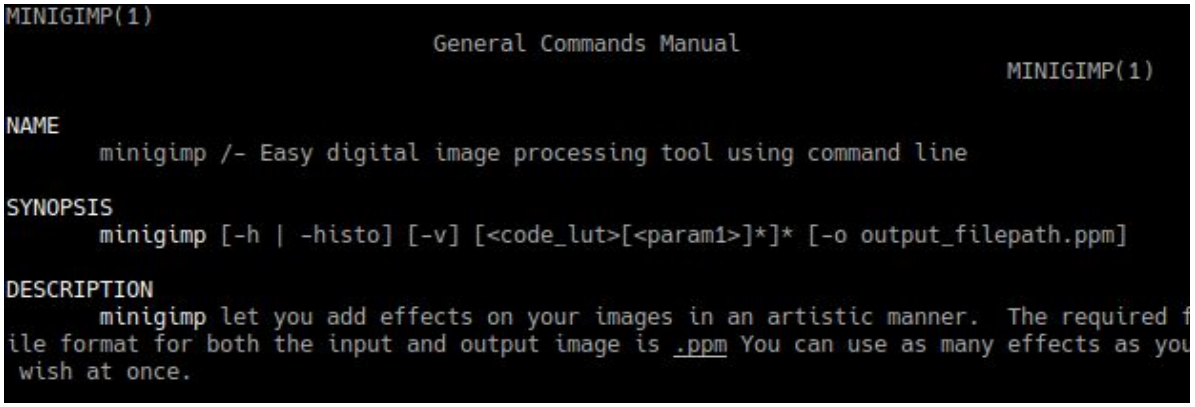
À partir du moment où l'on a compris que chaque index du tableau correspondait à la luminosité d'un pixel, l'adaptation des effets existant sur les LUTs a été assez rapide (à l'exception du Sépia un peu plus complexe à gérer, abordé dans la résolution de problème).

5 - Les bonus

Nous avons concentré nos efforts sur l'ajout d'effets, en particulier la gestion de matrice de convolution, que nous avons implémentée à la fois de façon statique et de façon dynamique. Nous avons trouvé de nombreux articles pour nous aider dans notre démarche, et l'application, bien qu'encore améliorable, est parfaitement fonctionnelle.

Côté interface, nous nous sommes intéressés par curiosité à la gestion des couleurs dans la console. Nous avons donc décidé de créer un titre en "ASCII art" pour notre programme. Ce dernier a été généré via l'outil de Patrick Gillespie, puis nous l'avons découpé en morceau afin de permettre son affichage et sa colorisation. Nous utilisons aussi la couleur pour afficher certains success.

Le programme dispose également d'une référence gitbook, disponible avec la commande "gitbook serve doc" (voir le README.md pour les dépendances), ainsi que de son manuel pouvant être consulté via la commande "man ./minigimp.man". Nous avons rédigé ce manuel en utilisant le formatage GNU troff (groff). Une version PDF du fichier est consultable en ligne.



Les premières lignes du man

Difficultés rencontrées et solutions apportées

> Nous n'avons pas rencontré de problèmes majeurs ayant bloqué notre avancée. Cependant les étapes ci-dessous ont été moins fluides que les autres.

Les commentaires du format P6

Le format P6 autorise dans sa spécification l'ajout de commentaire (précédé par un '#') à n'importe quel endroit de son header. Par exemple voici un header P6 valide :

```
P6
1024 # the image width
788 # the image height
# A comment
```

C'est une contrainte à gérer en plus, car il ne suffit plus de lire les données dans l'ordre. Il faut tester le premier caractère de chaque jeu de donnée, et s'il s'agit d'un '#', aller à la ligne pour lire la suite. Un algorithme plutôt trivial, encore faut-il y penser, et ce n'est pas aidé par la gestion contraignante des strings dans le langage C.

Makefile automatisé.

Face à l'ajout de nombreux fichiers dans le projet, la modification du makefile allait être une tâche pénible et répétitive. Nous avons donc voulu automatiser la détection de fichiers sources dans un dossier par le makefile.

Il est vrai que nous disposons aussi d'un fichier cmake qui gère cela à l'aide de la fonction "file(GLOB_RECURSE …)", mais par goût du challenge, nous avons cherché à reproduire ce comportement nativement.

À force de tutoriels vidéos, de recherches sur les forums et sur la documentation officielle, nous sommes parvenus à nos fins. L'astuce consiste à récupérer dans une variable tous les chemins relatifs des fichiers .c dans le dossier src, et tous les sous-dossiers via une wildcard. Pour les dépendances on utilise la même

démarche. On compile les différents éléments en .o et on termine par l'édition des liens.

Un inconvénient est que chaque objet dépend de toutes les dépendances définies plus haut. Nous ne sommes pas parvenus à gérer le choix des headers de façon 100% automatique.

Ce fichier très simple à faire évoluer, il est réutilisable pour des projets futurs.

Structure de données et Abstraction

L'allocation de données pendant l'exécution ainsi que la manipulation de tableau et autres pointeurs sont des **zones particulièrement à risque en développement**. Pour éviter les traditionnelles erreurs "Stack Overflow" ou "Memory access violation", nous avons fait le choix très tôt d'englober nos fichiers dans des "classes" abstraites, pour manipuler les données sans risque.

L'idée est la suivante : on fournit un 'struct' pour chaque type de données ayant besoin d'être stockées. Avec cet objet est disponible un ensemble de fonctions de création, de destruction, de lecture et d'application/modification. Les types de paramètres de ces fonctions ne sont pas contraignants pour faire face à la diversité d'utilisation.

void	**img_setPixelChannel** (**ImacImg** * img, unsigned int x, unsigned int y, int value, enum **img_Channel** c) Set the value of a color of a pixel.
void	**img_setPixelChannels** (**ImacImg** * img, unsigned int x, unsigned int y, int value) Set each colors of the pixel with the value.

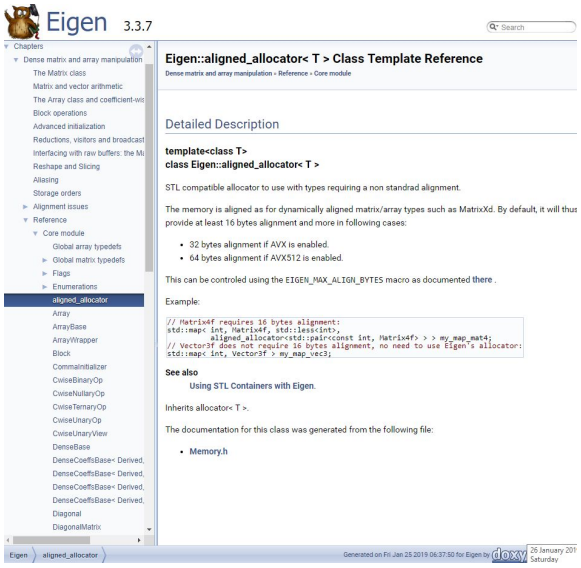
Extrait de fonctions publiques du struct ImacImg

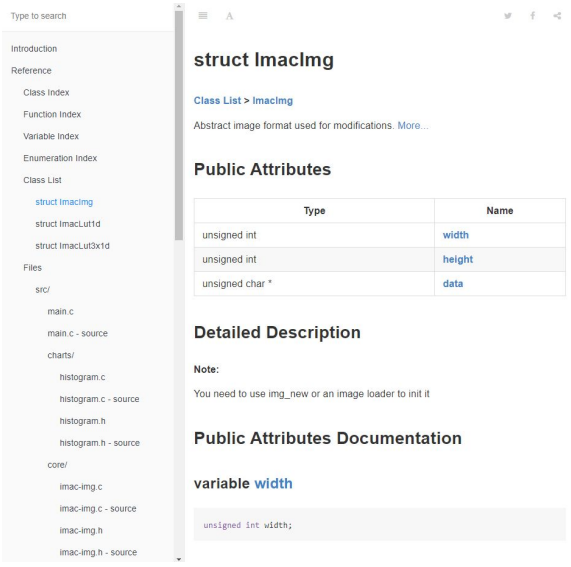
On remarque que **la valeur est un "int" et peut donc être négative**, alors que l'image est stockée en "unsigned char" (0-255). Chacune de ces fonctions valide et modifie les données passées en paramètre pour les faire correspondre au type de la donnée stockée. S'il y a un problème, un breakpoint se déclenche avec un message d'erreur explicite et l'exécution se stoppe.

En résumé, on favorise la portabilité et la sécurité face à la vitesse d'exécution.

Documentation

Nous avons adopté très tôt la notation de commentaire **Doxygen** sur l'ensemble de nos fonctions, pour ainsi avoir la documentation de notre code source générée automatiquement. Bien que cet outil permettra de générer un site à partir de ces commentaires, nous avons décidé d'utiliser Gitbook, un outil plus moderne pour visualiser la documentation.





Référence Doxygen Référence gitbook

Pour ce faire, Doxygen génère des données au format xml, ces données sont finalement récupérées par une bibliothèque et traduites au format Markdown pour être lu par Gitbook.

L'utilisation du Sépia

Lorsque nous sommes arrivés à la création de la fonction sépia, nous nous sommes posé la question suivante :

"Comment faire varier les canaux couleur les uns par rapport aux autres ?"

Nous avons déduit qu'il nous faudrait un LUT contenant 16 millions de données pour correspondre à toutes les combinaisons possibles. Après avoir lu plusieurs articles au sujet des LUT 3D nous nous sommes orientés vers l'utilisation d'un LUT 3x1D qui contient des paramètres différents sur les 3 canaux de couleur.

Problème, l'image est en couleur or la fonction de Sépia est exprimée en fonction des canaux R, V et B, ainsi nous convertissons les données des pixels en valeurs de

gris via une moyenne des trois composantes. Finalement les résultats étaient peu convaincants, car les images étaient sombres, nous avons réglé cela en convertissant notre image en niveaux de gris via la méthode de luminosité, qui applique une contribution différente aux trois luminophores.

Le débogage

Nous travaillions avec 3 compilateurs différents pour tester notre code, à savoir GCC, MINGW, et MSVC. Ces compilateurs gèrent tous l'indication de breakpoint différemment, alors que beaucoup de nos fonctions vont en déclencher un lorsqu'une vérification rate. L'utilisation d'assertion est certes plus portable, mais interrompt l'exécution ce qui n'est pas une alternative viable en développement.

Nous avons donc défini le macro DEBUG_BREAK, qui se transforme en raise(SIGTRAP) sur les compilateurs de type Linux, et en __debug_break() avec MSVC. Cet ajout nous a fait gagner un temps précieux, car nous connaissions dès lors le moment et les valeurs qui bloquent quand il y avait une erreur, cela sans avoir à relancer le programme.

Multiples effets de convolution

Les effets de convolutions prennent une image à analyser et affichent le résultat du calcul sur une autre image en sortie. Mais l'utilisation de plusieurs filtres à la suite supprimait le résultat de l'ancienne convolution.

Nous avons donc modifié notre code pour toujours utiliser un pointeur sur l'image principale et un pointeur sur une image temporaire. L'idée est d'intervertir la valeur de ces pointeurs à la fin de chaque effet de convolution. Ainsi l'image principale devient le résultat de sortie, et peut donc être passée sur une nouvelle convolution.

Conclusion

Ce que le projet nous a appris

Nous avons été amené à comprendre en profondeur certains aspects de la programmation en langage C. Beaucoup de contraintes avec lesquelles nous avons dû composer nous serviront plus tard à identifier des problèmes dans des langages de plus haut niveau.

Un autre très gros apport fut la découverte de nombreuses notions de traitement de l'image. Ce projet nous a poussés à faire des recherches, et nous sommes fiers de comprendre ce qui se passe derrière des logiciels comme Photoshop ou Gimp. Dorénavant nous appliquerons des effets en sachant comment ils fonctionnent, et nous saurons comment mieux les utiliser. Ce projet fut aussi l'occasion d'appliquer concrètement des notions mathématiques, jusqu'ici obscures à nos yeux.

Plus généralement nous avons appris à manipuler des fichiers, gérer la mémoire de l'ordinateur et amélioré nos pratiques de développeur face à la rigueur du langage.

Nicolas

Ce projet m'a aussi initié davantage à l'éditeur VIM et je suis ravi de pouvoir développer sur un logiciel généralement installé par défaut est dont la consommation en ressource est négligeable. Je suis aussi très satisfait d'avoir appris à créer des Makefiles, leur utilisation étant très versatile.

Guillaume

Ce projet a démystifié beaucoup de choses qui semblaient obscures et qui sont finalement assez simples à mettre en place et à réutiliser. En faisant des recherches sur la création d'un effet, on tombe sur d'autres articles intéressants et on continue à s'aventurer plus loin. J'ai pris goût au processing d'image, et j'ai hâte d'attaquer ce type d'opération avec les shaders.

ANNEXE

Détail des fonctionnalités supplémentaires

Matrice de convolution

Flou par moyenne à matrice automatique (BLUR)
Création d'une matrice dynamique calculant la moyenne des pixels avoisinants selon une certaine intensité donnée par l'utilisateur en paramètre.

Détection du contour (EDGE)
Mets en valeur la ligne qui apparaît dans la transition entre deux couleurs. Les pixels les plus proches d'une transition sont passés en blanc alors que le reste de l'image est noirci.

Estampage (EMBOSS)
Passage d'une image en nuance de gris, où les zones à faible contraste sont neutres alors que les contours sont marqués.

Flou Gaussien (GBLUR)
Application d'un très léger flou gaussien sur l'image. Une amélioration souhaitable est la création d'une matrice de convolution plus grande que celle actuelle (3x3) selon l'entrée de l'utilisateur.

Flou via dégradé radial (VBLUR)
Cela consiste en la création d'un point de flou au centre de l'image, d'une forme ronde et d'une intensité progressive du centre vers les côtés.

Ayant vu Pierre travailler sur un effet de vignettage, nous avons décidé de l'utiliser en corrélation avec le filtre de convolution de flou pour créer un effet de mise en évidence d'une partie de l'image. Jules nous a facilité la tâche en nous rappelant d'utiliser le théorème de Pythagore.

Histogramme

Histogramme pré- et post-effets (-h)
Génération facultative des histogrammes sous la forme d'images PPM, l'un présente la répartition des couleurs de l'image en entrée et l'autre de celle en sortie.

Histogramme multicolore et monochrome
Nous avons fait en sorte de faire apparaître les histogrammes des trois canaux de couleurs sous l'histogramme de moyenne afin que l'utilisateur final puisse mieux juger de la répartition de la luminance. Nous avons aussi créé une fonction pour l'afficher seulement pour un canal donné.

Histogramme avant/après dans le terminal (-v)
Cela permet à l'utilisateur d'avoir une idée générale de l'allure de l'histogramme avant et après application des effets sans exporter d'image supplémentaire. Les deux histogrammes sont affichés côte à côte.

L'affichage dans la console d'un tel graphique s'est avéré assez compliqué à réaliser à cause de la mise à l'échelle de l'affichage pour pouvoir occuper correctement l'espace restreint de la console.

Nous avons pensé l'intégralité de cet algorithme sur papier puis dans un programme à part avant de l'implémenter dans le projet. De plus cela fait appel à quelques astuces de printf( ) pour la gestion de l'alignement des caractères.

Contraste

Égalisation du contraste (HISTEQ)
Il s'agit là d'un effet pas du tout flexible, bien que très intéressant. Il permet adaptation histogramme afin de balancer le contraste sur toutes les plages de luminances. Cela permet de faire apparaître de nombreux détails sur une image peu contrastée, l'effet est assez impressionnant.

Le procédé utilise une fonction de répartition (discrète) de la luminance et une notion de probabilité qui est la fonction de masse. On obtient un résultat équilibré sans prise de tête pour l'utilisateur.

Voici les formules utilisées qui proviennent de ce document. Nous remercions Jules pour nous avoir apporté des précisions sur la formule.

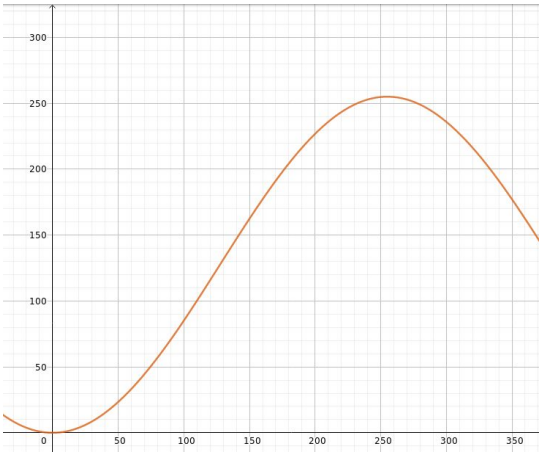
$$p_n = \frac{\text{number of pixels with intensity } n}{\text{total number of pixels}} \qquad n = 0, 1, ..., L-1.$$

$$T(k) = \text{floor}((L-1)\sum_{n=0}^{k} p_n).$$

Contraste sinusoïdal (SINCON)
Après avoir échangé avec Jules à propos du contraste, il nous a indiqué avoir utilisé la courbe du sinus pour réaliser sa fonction. Nous avons tout de suite compris ce qu'il voulait dire. Avec Élisa et Yoann nous nous sommes lancés dans la construction d'une fonction adaptée à notre besoin. Nous avons ainsi fini par créer ce qu'on cherchait et notre fonction f(x) représentant la nouvelle valeur de notre x dans le LUT

$$f(x) = \frac{\left(\sin\left(\frac{\pi}{255}\ x + \frac{3\pi}{2}\right) + 1\right) \cdot 255}{2}$$

Cette formule est néanmoins peu flexible par rapport à la méthode de contraste que nous avons implémenté et ne permet pas de variations très affinées. Son avantage et de ne pas écraser le détail de l'image.



Transformations

Miroir horizontal (FLIP_H), miroir vertical (FLIP_V)
Ces deux effets proviennent de notre réflexion lors du partiel de programmation. Nous intervertissons les pixels selon l'axe de symétrie, qui est respectivement la ligne du milieu et la colonne du milieu.