Toolkit image

Antoine Duquenoy, Antoine Gosse

20 décembre 2016

Sommaire

Introduction

Le projet intitulé “Toolkit Image” consiste à manipuler des images au format ppm. Ce type de fichier est particulièrement adapté à l'usage éducatif, en effet, il est construit d'une manière très simple : chaque pixel de l'image contient la proportion des couleurs primaires (rouge, vert et bleu) et cette ensemble forme la couleur que nous voyons. Bien que ce fichier soit facilement exploitable, il peut vite devenir très lourd de par sa non-compression (une image de 512x512 pèsera 768Ko alors qu'une image de 12Mpx fera 36Mo). A la manière d'un logiciel de traitement d'image, notre programme propose divers opérations sélectionnables à partir d'un menu en ligne de commande. Pour arriver au résultat final, nous avons dû compléter et modifier des parties du code proposé et nous avons fait certains choix afin de satisfaire au mieux notre vision du projet. En plus des fonctionnalités exigées par le cahier des charges, nous avons pris la liberté d'en ajouter d'autres.

Partie 1

Choix de réalisation

1.1 Modifications apportées au code d'origine

La fonction de redimensionnement exige une sauvegarde de l'image dans une résolution inférieure, en l'occurrence en 256x256. En revanche, la fonction de sauvegarde fournie ne le permet pas sans une légère modification. Pour faire face à ce problème, nous avons simplement ajouté deux paramètres qui sont la largeur et la hauteur de l'image. Ensuite, nous avons remarqué que l'en-tête d'une image lorsqu'elle est enregistrée ne permet pas la réutilisation de celle-ci par la suite car la fonction de chargement requiert le format d'en-tête standard du ppm.

1.2 Fonctionnalités pratiques

Les choix que nous allons évoquer dans cette partie concernent exclusivement les fonctionnalités pratiques et non les filtres applicables aux images. Notre menu textuel permet à l'utilisateur de choisir l'image qu'il souhaite modifier en indiquant son nom. Le fichier doit par contre se situer au même emplacement que le programme C et il n'est pas possible de renseigner le chemin complet du fichier. L'utilisateur est ensuite invité à entrer le nom du fichier de sortie sous la forme *NomImage.ppm*. De ce fait, l'image de départ reste intacte et réutilisable. Après avoir appliqué le filtre voulu, il est possible d'en ajouter d'autres de manière successive. Pour permettre un contrôle total de l'image finale, nous avons implémenté un moyen de prévisualisation qui consiste en une sauvegarde intermédiaire. A noter, qu'une fois l'image redimensionnée en 256x256, le programme n'est plus en mesure de modifier l'image et se ferme.

Partie 2

Présentation des fonctionnalités

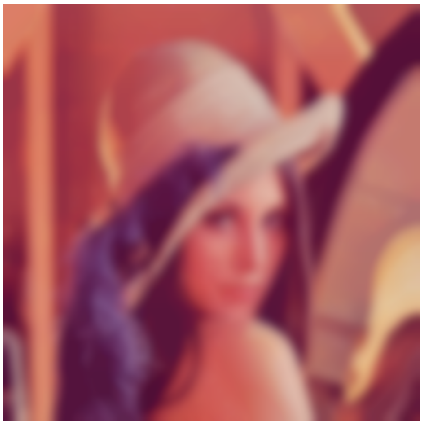
Dupliquer l'image Cette fonction que nous avons nommée "copy" prend en paramètre deux tableaux représentant les images. Le premier tableau est celui de l'image à copier alors que le second, bien souvent vide, est celui de la future image copiée. Pour effectuer cette copie, la fonction parcours tous les pixels, composé de trois couleurs, et recopie les valeurs dans l'autre tableau.

Réduire la taille de l'image en 256x256 La première idée qui nous est venue consistait à supprimer un pixel sur deux de l'image. Cette méthode, bien que fonctionnel et simple à mettre en place, souffre d'un problème de crénelage. Plus communément appelé *aliasing*, ce phénomène est visible par l'apparition de motif en forme d'escalier sur les contours obliques d'une image. Pour remédier à ce problème, il est possible depuis le menu, de sélectionner une réduction avec lissage. Cette option applique un filtre passe-bas (fonction flou) à l'image avant de la réduire. On constate que le résultat est bien meilleur.

Appliquer un flou Cette fonctionnalité est sans doute celle qui nous a demandé le plus de travail pour obtenir un résultat convaincant. L'idée globale consiste à faire une moyenne des pixels sur un voisinage plus ou moins grand. Nous avons procédé par étape pour la réaliser. Premièrement, nous avons décidé de ne pas se soucier de la taille du voisinage et on a donc opté pour une taille d'un pixel autour de celui visé. Avant même de commencer la programmation, nous avons décelé le premier problème : les pixels sur les bords de l'image ne pourront pas faire la moyenne de tous les pixels voisins. Pour faire face à ce problème, nous avons imaginé tous les cas particuliers qui peuvent se présenter. Avec ces éléments en main, la programmation s'est faite sans encombre, et le premier résultat correspondait à ce que nous attendions. Ensuite, grâce à cette base, nous avons appliqué le cahier des charges,

en donnant la possibilité à l'utilisateur d'entrer la taille de la zone de flou. Les conditions déterminées précédemment restent les mêmes, nous avons seulement ajouté une quatrième boucle for qui a pour rôle d'incrémenter la zone autour du pixel visé. Pour information, nous avons choisi d'utiliser une zone carrée pour faire la moyenne. Finalement, nous avons créé une autre fonction qui appelle autant de fois que demandé la fonction de flou pour effectuer plusieurs passes. Pour améliorer davantage notre algorithme, nous avons choisi d'intégrer quatre types de flou. En effet, après quelques recherches, nous nous sommes aperçu que le moyennage ne donnait pas le même poids aux pixels les plus éloignés, autrement dit, les pixels proches de celui visé ont plus d'importance dans le calcul de la moyenne. Voici les coefficients appliqués classés par ordre de rapidité de convergence vers zéro : $\frac{1}{2^x}$, $\frac{1}{x}$, $\frac{1}{ln(x)}$ et 1.

Flou Gaussien : taille 7 / 1 passe



Flou Gaussien : taille 7 / 15 passes

Passer l'image en niveau de gris Il s'agit d'une fonction extra qui permet de passer d'une image couleur à une image en niveau de gris. Pour réaliser ce filtre, il faut savoir qu'un gris (neutre) a ses trois composantes RVB égales. Pour déterminer la nuance de gris, et ainsi avoir un gris plus ou moins foncé, il faut donc faut faire la moyenne des trois valeurs de couleurs et l'appliquer sur les trois composantes.

Passer l'image en noir et blanc Le filtre noir et blanc, également en extra, permet de passer d'une image couleur à une image uniquement composée de noir et de blanc. Notre première fonction se résumait à changer les pixels dont la moyenne était inférieure à 127 en noir et les autres en blanc. Pour l'exemple de Lena, cela fonctionnait très bien, en revanche, pour des images dont la luminosité était soit basse soit haute, on voyait clairement une dominance de blanc ou de noir. Afin de régler ce problème, nous effectuons au préalable une moyenne de tous les pixels de l'image, moyenne qui

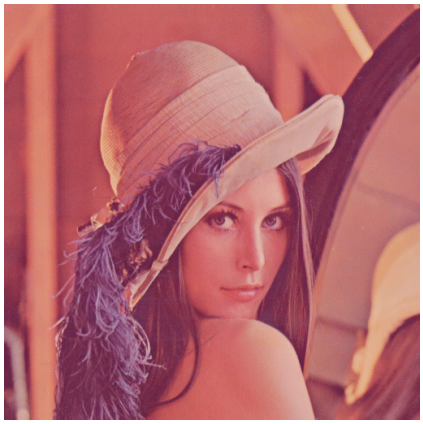
sera ensuite utilisée comme seuil. Ainsi, une image plutôt sombre se verra pourvue malgré tout de détails. Pour laisser plus de libertés à l'utilisateur, il est également possible d'entrer le seuil manuellement.

Niveau de gris

Noir et Blanc : seuil auto

Filtre test Cet ajout devait être le précurseur de nombreuses autres fonctionnalités utilisant ce principe mais pour des contraintes liées aux temps, nous ne pouvons présenter que celle-ci. Concrètement, l'image de sortie correspond à celle d'entrée avec des zones floues. En fait, on commence par générer une image floue et une image noir et blanc de l'image d'origine. Ensuite, quand il s'agit d'un pixel blanc, le pixel de sortie reste de même couleur que celui d'origine mais lorsque le pixel est noir, le pixel de sortie prend la valeur de celui de l'image floue. Le résultat n'a rien de visuellement agréable mais ce principe de masque aurait pu être détourné à des fins plus "artistiques".

Extrapoler l'image Applique un effet d'extrapolation sur l'image.



Extrapolation 0.7

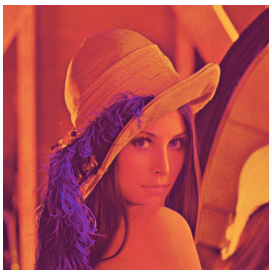
Extrapolation 0.2

Changer la teinte Tout comme les deux fonctions suivantes, cette fonctionnalité devient facile à mettre en œuvre lorsque l'on passe l'image en TSV. En effet, les grandeurs de ce système de gestion de couleurs sont plus évidentes à manipuler que du RVB. Ce filtre demande à l'utilisateur une valeur de teinte entre 0 et 360 puis repasse la matrice TSV en une image RVB.

Changer la saturation Modifie la saturation de l'image. Convertit l'image en TSV, modifie la deuxième valeur "Saturation" puis reconvertit l'image en RVB.

Changer l'intensité lumineuse Modifie l'intensité lumineuse de l'image. Convertit l'image en TSV, modifie la troisième valeur "Luminosité" puis reconvertit l'image en RVB.

Teinte 120



Saturation 70%

Lumière 90%

Appliquer un flou directionnel Ce flou est quasiment similaire à celui vu en (3), si ce n'est que la zone de moyennage est seulement horizontale et non plus carré.

Flou directionnel : taille 21

Agrandir l'image en 1024x1024 L'agrandissement consiste pour un pixel visé à le reproduire quatre fois sur l'image agrandie. C'est comme si l'on formait des plus gros pixels sur l'image de sortie.

Inverser les couleurs Cette fonction extra remplace chaque couleur par son complément. Les composantes du pixel sont remplacées par la soustraction 255 - valeur originale.

Inversion des couleurs

Partie 3

Difficultés rencontrées

Lors de ce projet, nous avons dû faire face à plusieurs difficultés. Tout d'abord, l'en-tête du fichier ppm dans la fonction de sauvegarde de l'image nous a posé problème. En effet, celle-ci ne respectait pas les standards du ppm, il nous était alors impossible de réutiliser une image sur laquelle notre programme avait déjà appliqué un effet (la fonction de chargement ne reconnaissait pas l'image et refusait donc de l'ouvrir). Ceci a été résolu en modifiant l'en-tête des fichiers dans la fonction de sauvegarde de l'image.

Nous avons également rencontré un problème lors de la création de la fonction RVB vers TSV. L'image résultante était noire. Tous les pixels valaient (0,0,0). Nous pensions que cela venait de la fonction en elle-même, nous avons donc utilisé des flottants au lieu d'entiers car contrairement au RVB les valeurs du TSV sont des flottants. Ceci n'a pas suffi à résoudre le problème qui venait en fait des fonctions "Maximum(x,y,z)" et "Minimum(z,y,z)" qui permettent de déterminer respectivement la valeur maximale et minimale entre 3 valeurs x,y et z. Ces fonctions sont directement utilisées dans les fonctions de conversion du RVB vers le TSV, mais comme elles retournaient des entiers, le résultat ne pouvait pas être juste. Pour résoudre ce problème nous avons changé toutes les variables des fonctions Maximum et Minimum vers des flottants, la fonction était ensuite opérationnelle.

Enfin, notre dernier problème fut de bien cibler les types de variables en fonction de la situation, nous n'avions encore jamais eu recours au type "unsigned", nous avons donc dû bien comprendre la manière de l'utiliser.

Conclusion

Durant ce projet nous avons fait plusieurs apprentissages. Tout d'abord ce travail en binôme nous a demandé de l'organisation, il fallait se mettre d'accord sur les conventions à appliquer pour assurer la lisibilité et l'homogénéité du code. Nous avons dû ensuite nous mettre d'accord sur la manière de faire les choses, notamment sur la fonction du flou où plusieurs approches sont possibles. Nous avons également amélioré notre maîtrise du langage C notamment sur l'aspect structurel du code qui dans notre projet est découpé en plusieurs fonctions. Ce programme nous a demandé une attention particulière concernant les types des variables qui n'étaient pas toujours adaptés dans toutes les situations. Finalement, on s'accorde sur le fait que ce projet nous a surtout, au-delà d'appliquer nos connaissances en C, permis de développer notre raisonnement algorithmique en passant par des phases de recherche sur papier au préalable afin d'éviter les mauvaises surprises.