ICC – Rapport du projet de programmation ColoReduce

Matthias Kockisch (SCIPER : 303000), 10.12.2018

Analyse du code

Les déclarations se font directement dans la fonction main, mais des fonctions s'occupent de l'initialisation du nombre de couleurs réduit, du nombre de filtrages, du vector contenant les couleurs de filtrage, de celui contenant les seuils et de l'Image image0 contenant les valeurs RVB du fichier d'entrée, car ces éléments nécessitent une vérification de leurs valeurs, qui mène à un message d'erreur et l'arrêt intentionné du programme en cas d'entrées inconvenables. Le type Image correspond à un vector 2D contenant des arrays de 3 entiers chacun, pour permettre de contenir séparément les valeurs RVB de chaque pixel de l'image originale en fonction de leur ligne et de leur colonne. Le char P3 est utilisé pour séparer l'ensemble « P3 » du fichier d'entrée du reste des variables, et ainsi éviter d'assigner la valeur 3 à la variable suivante et de décaler les entrées. P3 est réutilisé pour le fichier sortie, ce qui n'est néanmoins pas nécessaire vu que le contenu de ce char est le même pour chaque fichier d'entrée.

Pour le seuillage, le programme utilise le type ImageS (« image seuillée »), qui est un vector 2D contenant des entiers. On déclare une ImageS de nom images0 et de dimensions nbL et nbC qui contiendra un indice par pixel de l'image d'origine. images0 est remplie par la fonction seuil_images, qui calcule la somme des carrés des valeurs RVB puis l'intensité normalisée de chaque pixel ; l'intensité normalisée est ensuite comparée aux seuils en ordre croissant jusqu'à avoir atteint le premier seuil supérieur à l'intensité normalisée du pixel. La valeur de la place du seuil dans le vector seuil sera copiée dans images0 à l'endroit correspondant au pixel traité. La deuxième condition « s<nbR » de la boucle while de comparaison a été ajoutée pour éviter une boucle infinie lors du seuillage d'un pixel parfaitement blanc, en assignant le dernier seuil à un tel pixel, ce qui ne respecte plus la première condition « in>=seuil[s] ».

Le filtrage réutilise le type ImageS pour déclarer une image destination images1. Pour chaque pixel hormis les bords, la fonction prefiltre construit un array à partir des 8 pixels voisins et lance la fonction filtre avec cet array comme entrée. Cette fonction compte le nombre d'apparition des indices de 3 des pixels de l'array en sautant les indices égaux parmi ceux-ci, puis sort le premier indice apparaissant 6 fois, ou 0 s'il n'y en a pas. Il est en effet seulement nécessaaire de compter le nomre d'apparition des valeurs de trois pixels, même si ces valeurs sont égales. La valeur sortie par filtre est ensuite rangée dans l'emplacement respectif de images1. Afin de pouvoir recommencer le filtrage avec la nouvelle ImageS, images1 est finalement copiée dans images0. Si images0 n'avait pas été copiée dans images1 au tout début de prefiltre, cette dernière instruction de prefiltre remplacerait les bords de images0, non filtrés, par des zéros ; à chaque nouveau filtrage, les pixels colorés voisins à ces bords noirs deviendraient noirs et ainsi de suite, et l'image finale aurait donc un bord noir de largeur le nombre de filtrages. Finalement, la fonction bords_noirs, qui remplace les indices des cases en début et fin de lignes et de colonnes par des zéros, est exécutée.

La dernière étape est chargée de construire le fichier de sortie : l'affichage du header du .ppm se trouve directement dans le main, puis la fonction cout_RVB sort les valeurs RVB correspondantes aux indices des pixels dans l'ImageS filtrée ; à ce stade, les valeurs dans images0 et images1 sont identiques. Un retour à la ligne est ajouté en fin du fichier pour avoir une similitude parfaite entre ce fichier et la sortie du programme demo.

Pseudocode de la fonction prefiltre

Entrées :

- nbF (nombre de filtrages)
- nbL (nombre de lignes de l'image)
- nbC (nombre de colonnes de l'image)
- images0 (vector 2D de dimensions nbL et nbC contenant les indices des pixels de l'image)
- images1 (vector 2D de dimensions nbL et nbC)

Sortie : images0 (vector 2D de dimensions nbL et nbC contenant les indices des pixels de l'image après tous les filtrages)

images1 ← images0

Pour f de 0 à nbF-1 :

    Pour i de 1 à nbL-2 :

        Pour j de 1 à nbC-2 :

            Initialisation d'un array tf de taille 8 contenant tous les indices des pixels voisins du pixel [i][j] dans images0

            images1[i][j] ← fonction filtre avec l'array tf en entrée

    images0 ← images1

Sortir images0

Pseudocode de la fonction filtre

Entrée : tf (array de taille 8 contenant tous les indices des pixels voisins d'un pixel de images0)

Sortie : l'élément de tf présent au moins 6 fois dans tf **ou** 0 (nouvel indice du pixel [i][j])

Initialisation des entiers c et d avec la valeur 0

Faire (

    d ← 0

    Faire c ← c+1 tant que la valeur c de tf est égale à la valeur à l'emplacement c+1 dans tf

    Pour e de 0 à 7 :

        Si la valeur c de tf est égale à la valeur e de tf, alors d ← d+1

) tant que c est inférieur à 3 et d est inférieur à 6

Si d est supérieur à 5, sortir la valeur à l'emplacement c dans tf

Sinon sortir 0

Complexité du programme

Ordre de complexité des fonctions :

- cin_nbR : complexité indépendante des valeurs des entrées
- cin_cf : O(nbR)
- cin_seuil : O(nbR)
- cin_nbF : complexité indépendante des valeurs des entrées
- cin_image : O(nbL*nbC)
- seuil_images : O(nbL*nbC*nbR)
- prefiltre : O(nbF*nbL*nbC)
- filtre : complexité indépendante des valeurs des entrées
- bords_noirs : O(nbL+nbC)
- cout_RVB : O(nbL*nbC)
- fonctions error : complexité indépendante des valeurs des entrées
- main (sans les autres fonctions) : complexité indépendante des valeurs des entrées

Ordre de complexité du programme :

- O(nbL*nbC*nbR) si nbR>nbF
- O(nbF*nbL*nbC) si nbR<nbF