Projet Fondements de l'informatique

Luc Brun & Eric Ziad

1 Informations générales

Début du projet : Semaine du 4 décembre

Remise du projet : Semaine du 14 janvier

Nombre de personnes : le projet est effectué en binôme.

Documents à remettre : Un rapport écrit de 10 pages (max) avec une description des méthodes algorithmiques et des structures de données utilisées ainsi qu'un jeu d'essais. Ce document doit être remis à votre encadrant de TP de C.

Un module permettant de charger/sauvegarder une image vous sera fournie dans le répertoire /home/PROJET_FONDEMENTS. Le format des images traitées est le format ppm dans lequel chaque pixel de l'image est codé par 3 octets consécutifs codant les valeurs de rouge, vert et bleu.

2 Calcul d'histogramme couleur

L'histogramme couleur d'une image code pour chaque triplé (R,G,B) le nombre de pixels de l'image de cette couleur. Un tel histogramme pourrait être stocké à l'aide d'un tableau 3D de taille 256^3. Toutefois ce type de stockage induit un gaspillage de mémoire pour les images de taille usuelles (Calculez en effet, la taille maximum requise pour stocker l'histogramme d'une image 256×256).

On se propose donc de coder l'histogramme comme un tableau $2D$ de pointeurs sur des listes. Une telle représentation est fournie par un tableau *histo* de taille 256×256 dont chaque entrée $histo[R][G]$ est un pointeur sur une liste simplement chaînée de valeurs de B (voir Fig 1). Chaque cellule de cette liste peut être codée par la structure suivante :

```
typedef struct cell * cell;
struct
{
  unsigned char B;
  int           freq;
  cell          next;
} cell;
```

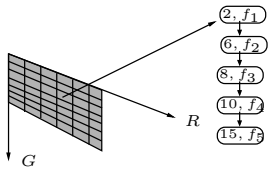

FIGURE 1 – Représentation de l'histogramme $2D$

Cette structure est une structure privée du module histo (elle doit donc être déclarée dans histogram.c).

Une fois l'histogramme construit à partir d'une image, la présence d'une cellule cell dans une liste $histo[R][G]$ signifie que l'image contient cell.freq pixels de couleur $(R, G, cell.B)$.

2.1 Manipulation de listes

Définissez les fonctions suivantes dans le fichier histo.c.

1. `cell create_cell(int B,cell next)`
   Alloue dynamiquement une cellule en initialisant les champs B et next. Le champ freq est initialisé à 1.
2. `cell insert_cell(cell head,int B)`
   Cette fonction :
   (a) ajoute une cellule contenant B avec la fréquence 1 si B n'est pas présent dans la liste head.
   (b) incrémente la fréquence de la cellule contenant B si celle ci existe.
3. `cell delete_list(cell list)`
   Supprime toute les cellules d'une liste (en libérant la mémoire).

ATTENTION ! L'insertion des valeurs de B dans la liste doit produire des listes triées par ordre croissant.

2.2 Construction, initialisation, destruction de l'histogramme

Le type histo sera défini comme un pointeur sur cell dans histogram.h. Définissez les fonctions suivantes :

1. `histo create_histo()`
   Cette fonction alloue dynamiquement un tableau de taille 256×256 dont chaque entrée est initialisée au pointeur NULL.
2. `void init_histo(histo,image)`
   Cette fonction très importante initialise un histogramme à partir d'une image. Supposons que le tableau de listes se nomme histo, pour chaque pixel de l'image de couleur (R, G, B) cette fonction :

(a) Crée une liste à l'entrée $histo[R][G]$ si celle-ci est vide. Cette liste contient la valeur de B avec une fréquence égale à 1,

(b) ajoute une cellule contenant B et la fréquence 1 si B n'est pas présent dans $histo[R][G]$,

(c) incrémente la fréquence de la cellule contenant B si celle ci existe.

Les insertions de valeurs de B doivent être effectuées de façon a préserver un ordre croissant de valeurs de B.

3. `void delete_histo(histo)`
   Comme son nom l'indique cette fonction parcours l'ensemble des cellules du tableau histo et efface toutes les listes allouées. Elle supprime également le tableau histo.

2.3 Interrogation de l'histogramme

Définissez la fonction :

```
int give_freq_histo(histo h,int R,int G,int B)
```

qui renvoie le nombre de pixels de couleur (R, G, B) dans l'image associée à histo. Ce nombre est égal à la fréquence de la cellule de valeur B dans histo[R][G] si celle ci existe et 0 sinon.

On considère la structure suivante :

```
typedef struct histo_iter * histo_iter;
struct
{
 int R,G;
 cell current;
} histo_iter;
```

Cette structure est également une sous structure privée du module histo et doit donc être déclaré dans histogram.c. Elle correspond à ce que l'on appelle un itérateur (d'où son nom) et va nous permettre de parcourir l'ensemble des couleurs de l'histogramme.

Définissez les fonctions suivantes :

1. `histo_iter create_histo_iter()` Alloue une structure de type `histo_iter`. Les champs R et G correspondent aux indices de la première entrée `histo[R][G]` non nulle, le champ current est positionné sur la première cellule de `histo[R][G]`. Un assert doit arrêter le programme si toutes les cellules du tableau histo sont nulles.
2. `void start_histo_iter(histo_iter)`
   Positionne l'itérateur à la position qu'il avait à sa création (voir fonction précédente).

3. `boolean next_histo_iter(histo_iter)`
   Cette fonction passe à l'élément suivant de la liste de B si on est pas positionné en fin de liste (R et G restent alors inchangés). Sinon, R et G correspondent à la prochaine liste `histo[R][G]` non vide et current est positionné à la première cellule de cette liste. La fonction renvoie vrai si une telle liste existe ou si on est pas en fin de liste et faux sinon. L'ordre suivi est l'ordre classique de lecture ligne à ligne du coin en haut à gauche au coin en bas à droite.
   le type boolean est défini dans le .h par :
   ```
   typedef enum {false,true} boolean;
   ```
4. `void give_color_histo_iter(histo_iter,int*)`
   Renvoie la couleur courante de l'itérateur. Le second argument est un tableau de 3 entiers.
5. `int give_freq_histo_iter(histo_iter)`
   Renvoie la fréquence de la couleur courante de l'itérateur.
6. `void delete_histo_iter(histo_iter)`
   Supprime la structure.

3 Implémentation de ppmhist

Définissez un programme qui prend en paramètre (sur la ligne de commande) un fichier ppm et affiche la liste des n-uplets (R,G,B,lum,nb occurrence). R, G et B correspondent aux pixels de l'image, lum est défini par $(R+G+B)/3$ et nb occurrence représente le nombre de pixels de l'image de couleur (R, G, B).

4 Quantification par popularité

On désire afficher l'image avec un nombre fixe (K) de couleurs tout en préservant autant que possible la qualité de celle ci. Pour cela on va concevoir ce que l'on appelle un algorithme de quantification par popularité. L'objet de cet algorithme est d'afficher une image avec ses K couleurs de plus grande fréquence. Définir dans un fichier quantification.c les fonctions suivantes :

1. `void quantification(histo h,int* tab,int K)`
   Cette fonction prend en paramètre un histogramme et renvoi un tableau de taille $K \times 3$ contenant les K couleurs de plus haute fréquence dans l'image. Cette fonction :
   (a) initialise une liste triée par ordre croissant d'occurrence contenant les K première couleurs de l'histogramme. Soit $\{f_1, \ldots, f_K\}$ avec $f_1 > f_2 > f_K$ les nombre d'occurrences des couleurs en question.
   (b) Pour chaque cellule (R, G, B, f) rencontrée en parcourant les couleurs restantes de l'histogramme. Soit f le nombre d'occurrence de la couleur courante :

— Si $f < f_K$ ne rien faire et passer à la couleur suivante
— sinon mette à jour la liste des K couleurs de plus haute fréquence.

(c) Une fois l'histogramme entièrement parcouru transférer la liste dans un tableau.

2. `void mapping(image input,image output,int* tab,int K)`
   Cette fonction crée une image couleur output de la même taille que l'image d'entrée input. Pour chaque pixel de l'image d'entrée, cette fonction écrit dans l'image de sortie la couleur dans la table tab la plus proche de celle-ci (on pourra définir une fonction à cet effet).
   Question Optionnelle : Calculer la complexité du calcul du plus proche voisin et améliorer celle ci en vous basant sur le cours en bas de page [1]

Écrivez un programme qui enchaîne les fonctions précédentes pour créer un fichier correspondant à l'image output de la question précédente. Ce programme prend le nombre K et l'image d'entrée input en paramètres.

4.1 Question Optionnelle

Cette question n'est pas obligatoire mais peut rapporter des points si elle est bien traitée.

Le programme précédent ne fournit généralement pas des résultats de très bonne qualité. Ceci est essentiellement due à la dispersion des couleurs dans une image.

1. Proposez une modification du module histogramme permettant de considérer que deux triplés (R,G,B) et (R',G',B') sont égaux si (en division entière) : $R/2 = R'/2, G = G', B/4 = B'/4$.
2. Cette nouvelle égalité modifie bien évidement les fréquences et donc l'algorithme de quantification. Une cellule ne représente plus une couleur mais un ensemble de couleurs dont il convient de choisir un représentant. Modifier la fonction mapping pour que l'affichage de l'image de sortie utilise ces représentant(a définir).

5 Remarques

1. L'objet du rapport n'est pas de dupliquer les commentaires (qui doivent être présent dans le code) mais d'expliquer les différents choix possibles pour chaque problème et les avantages, inconvénients de la méthode retenue.
2. L'originalité et l'efficacité des solutions algorithmiques (montrées dans le rapport) ainsi que la qualité du code seront des éléments déterminant de la note. Chaque binôme devra faire une démonstration de son projet.

1. http ://www.greyc.ensicaen.fr/∼luc/ENSEIGNEMENT/COURS/tr_img_couleur.ps.gz, section 6.4.3

3. Des images sont disponibles sous /home/public/PROJET_FONDEMENTS avec le module image.