ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

Fondements de l'Informatique

Projet - Réduction de couleurs

Encadrant de TP : BRUN Luc

Table des matières

1 Introduction

L'objectif de ce projet est, étant donné une image possedant potentiellement 256^3 couleurs différentes, de la représenter à l'aide d'un nombre fixe $K < 256^3$ couleurs. Pour cela, on utilisera un algorithme de **quantification par popularité** qui consiste à remplacer chaque couleur par la couleur de plus haute fréquence qui lui est la plus proche.

Ce travail présente brièvement[1] l'ensemble des structures et fonctions créees dans ce but, en détaille les algorithmes[2] ou en propose des alternatives ou optimisations.

2 Organisation du répertoire de travail

3 Calcul d'histogramme couleur

Un histogramme couleur d'une image I permet d'associer une couleur (R, G, B) à sa fréquence d'apparition f dans I.
Un représentation par un tableau $256 \times 256 \times 256$ prendrait trop de place en mémoire. Par

[1] La description des fonctions et leurs commentaires étant présents dans les sources.
[2] Décrits en énoncé

ailleurs, une représentation de $\mathcal{H}$ par une liste de liste de liste permettrait d'optimiser spatialement son occupation mémoire, mais augmenterait significativement la complexité des algorithmes de construction et de traitement de l'histogramme du fait des parcours de liste.
Un bon compromis serait donc de représenter $\mathcal{H}$ par un tableau à deux dimensions, donc chaque élément (R, G) pointe vers une liste contenant les valeurs de B.

3.1 Manipulation de listes

L'algorithme d'insertion d'une cellule dans une liste, de complexité linéaire à été vu en cours. La seule différence est que lorqu'on trouve la cellule, on augmente simplement son champ `freq` .

3.2 Construction, initialisation et destruction de l'histogramme

3.2.1 Construction de l'histogramme

```
histo create_histo()
```

Alloue un tableau à deux dimensions en mémoire et initialise chaque cellule à `NULL`et le renvoie en sortie.

3.2.2 Initialisation à partir d'une image

```
void init_histo(histo,image)
```

Initialise l'histogramme à partir d'une image donnée.

3.2.3 Destruction

```
void delete_histo(histo)
```

Libère l'histogramme de la mémoire.

3.3 Interrogation de l'histogramme

Une fois l'histogramme $\mathcal{H}$ crée, on a besoin d'un outil qui nous permet de parcourir facilement l'ensemble des couleurs présentes dans l'histogramme : on va utiliser un itérateur, qui pointe sur un cellule et qu'on pourra facilement modifier pour qu'il pointe sur la couleur suivante.

On le définit comme étant une structure privée du module `histo`
Néanmoins, on peut constater qu'il pourrait être utile de pouvoir utiliser cette structure en dehors du module, comme par exemple dans le module quantification. En effet, lorsque l'on compare la fréquence de la couleur contenu dans la cellule pointée par l'itérateur pour savoir s'il faut l'insérer dans la liste des K couleurs de plus haute fréquence, on fait appel à une fonction qui nous donne la fréquence par un parcours de la liste (R, G) alors que cette même fréquence est contenue dans la cellule pointée directement par l'itérateur : il faudrait donc plutôt créer une fonction `give_freq_histo_iter()` plutôt qui renvoie directement le champs `freq` de la cellule pointée par l'itérateur.

3.3.1 Récupération de la fréquence d'une couleur

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

Renvoie la fréquence associée à la couleur (R, G, B) à partir de l'histogramme h.

3.3.2 Création et positionnement de l'itérateur

```
histo_iter create_histo_iter(histo h)
```

Alloue en mémoire l'itérateur et le place à la première entrée non nulle de l'histogramme h.

3.3.3 Repositionnement de l'itérateur

```
histo_iter start_histo_iter(histo h)
```

Alloue en mémoire l'itérateur et le place à la première entrée non nulle de l'histogramme h.

3.3.4 Couleur de l'itérateur

```
void give_color_histo_iter(histo_iter iter, int *c)
```

Copie la couleur de l'itérateur `iter` dans le tableau pointé par `c` .

3.3.5 Destruction de l'itérateur

```
void delete_histo_iter(histo_iter)
```

Désallour la mémoire associée à un itérateur.

4 Implémentation de ppmhist

Le programme `ppmhist` utilise le travail réalisé précédemment pour afficher les différentes couleurs de l'image ainsi que leur luminosité et leur fréquence d'apparition. Il se structure suivant les étapes suivantes :

E1. Chargement de l'image

E2. Positionnement du pointeur courant au début de l'image

E3. Création de l'histogramme

E4. Initialisation de l'histogramme à partir de l'image

E5. Création de l'itérateur

E6. Initialisation de l'itérateur

E7. Affichage de la couleur pointée (et ses caractéristiques) par l'itérateur courant

E8. Passage à l'itérateur suivant

Repeter E7,E8 Tant Que l'itérateur courant est non nul.

5 Quantification par popularité

Pour cette partie, on définit préalablement une nouvelle structure de liste dont chaque cellule contient un tableau codant une couleur (R, G, B), la fréquence correspondante et un pointeur sur la cellule suivante.

5.0.6 Quantification

```
void quantification(histo h,int *tab,int K)
```

E1. On initialise une liste de longueur K contenant la liste des K premières couleurs de l'image.

E2. On ajoute chaque cellule restante de l'histogramme parcouru à l'aide de l'itérateur à la liste si cette cellule est de fréquence plus grande que la couleur pointée par la tête de la liste (qui est par construction de fréquence minimale car la liste est triée par ordre croissant). À la fin de cette étape, on dispose dans une liste des K couleurs de plus haute fréquence.

E3. On convertit cette liste[3] en tableau de taille $3 \times K$.

5.0.7 Couleur la plus proche

```
closest_color(int* tab,int K,int R,int G,int B,int*output)
```

Pour une couleur (R, G, B) donnée, renvoie la couleur (R', G', B') appartenant au tableau `tab` la plus proche de (R, G, B), *i.e* celle minimisant la quantité $\sqrt{(R'-R)^2+(G'-G)^2+(B'-B)^2}$ [4]

Il s'agit d'un simple parcours d'un tableau de longueur $3K$, donc en complexité linéaire. On peut améliorer cette complexité en réalisant une recherche par dichotomie.

[3] La structure de liste permet temporairement de pouvoir faire des insertions sans avoir à réaliser de décalage

[4] Ou son carré, ce qui revient au même et qui est moins coûteux en temps de calcul.

5.0.8 Mapping

```
void mapping(image input,image output,int* tab,int K)
```

Cette fonction parcours simplement l'image d'entrée et modifie la couleur de chaque pixel par celle qui lui est la plus proche (voir fonction précédente) dans la liste des K couleurs de plus haute fréquence.

6 Quelques résultats

(a) Image d'origine (b) 1024 couleurs (c) 128 couleurs

Figure 1: Tests pour l'image `fleurs.ppm`

(a) Image d'origine (b) 1024 couleurs

(a) 128 couleurs