Projet de Programmation

Juliusz Chroboczek

17 novembre 2021

1 Introduction

Le but de ce projet est d'écrire un programme qui lit un fichier contenant une image et écrit une image visuellement semblable dans un format plus efficace, le *format indexé*.

1.1 Formats d'image en mémoire

Une image est stockée en mémoire comme une matrice (un tableau à deux dimensions) de *pixels*, dont chacun représente une couleur.

Représentation des couleurs Du fait de la structure de l'œil des gens normaux [1], les couleurs constituent un espace à trois dimensions. Il existe plusieurs façons de le représenter, mais la plus habituelle est le format RGB, où une couleur est représentée par trois entiers (R, G, B) qui indiquent l'intensité de la lumière rouge, verte et bleue respectivement. Dans ce projet, nous utiliserons une représentation *additive* avec une *profondeur* de 8 bits : R, G et B seront compris entre 0 (pas de lumière) et 255 (intensité maximale de la lumière).

Format RGB Une image au format RGB est un tableau de lignes (figure 1). Chaque ligne est une suite de valeurs $(R_0, G_0, B_0, R_1, G_1, B_1 \ldots)$ dont la longueur est 3 fois la largeur de l'image (3 valeurs représentent un pixel).

R_0	G_0	B_0	R_1	G_1	B_1	$\cdots$
R'_0	G'_0	B'_0	R'_1	G'_1	B'_1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 1 — Une image au format RGB. Chaque case est un octet.

1. Les daltoniens et les tétrachromates « vrais » ont une vision différente de la norme.

Format RGBA Le format RGBA (figure 2) est une extension au format RGB où chaque pixel est représenté par quatre valeurs ; une ligne est une suite $(R_0, G_0, B_0, A_0, R_1, G_1, B_1, A_1 \ldots)$ dont la longueur est 4 fois la largeur de l'image.

Le format RGBA a deux avantages par rapport au format RGB : chaque pixel commence à une adresse qui est un multiple de 4, et la composante A permet de stocker l'information de transparence (voir partie 5 ci-dessous).

R_0	G_0	B_0	A_0	R_1	G_1	B_1	A_1	$\cdots$
R'_0	G'_0	B'_0	A'_0	R'_1	G'_1	B'_1	A'_1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 2 — Une image au format RGBA.

Format indexé Les formats ci-dessus sont efficaces si toutes les couleurs de l'image sont distinctes ; cependant, si l'image contient un petit nombre de couleurs distinctes, elles gâchent de l'espace en stockant les mêmes couleurs plusieurs fois.

Une *palette* est une suite finie de couleurs $P = (C_0, C_1, \ldots)$ toutes distinctes. Étant donnée une palette, une *image indexée* par cette palette (figure 3) est une image qui, au lieu de stocker des couleurs, stocke des indices de la palette. Ainsi, une ligne de l'image contient une suite d'indices $(I_0, I_1 \cdots)$, et elle est interprétée comme la suite de couleurs $C_{I_0}, C_{I_1} \ldots)$.

I_0	I_1	$\cdots$
I'_0	I'_1	$\cdots$
$\vdots$	$\vdots$	$\ddots$

R_0	G_0	B_0
R_1	G_1	B_1
R_2	G_2	B_2
$\vdots$	$\vdots$	$\vdots$

FIGURE 3 — Une image au format indexé avec sa palette.

1.2 Formats d'image sur disque

Il existe de nombreux formats de fichiers qui permettent de stocker une image. Dans ce projet, nous utiliserons le format PNG, qui est un format compressé qui permet de stocker entre autres des images RGB et RGBA, ainsi que des images indexées avec une palette de 256 éléments au plus.

1.3 Code fourni

Je vous fournis une bibliothèque capable de lire et d'écrire les fichiers au format PNG, ainsi qu'un programme capable de convertir des images arbitraires en un format utilisable par ma bibliothèque.

1.3.1 Bibliothèque `pngio`

Je vous fournis une archive contenant deux fichiers `.c` et un fichier `.h`.

Les fichiers `pngio.h` et `pngio.c` implémentent une petite bibliothèque qui permet de lire les images au format RGBA et d'écrire des images au format indexé. Les images RGBA sont représentées par la structure suivante :

```
struct image {
    unsigned char **data;
    int height, width;
};
```

Les champs `height` et `width` indiquent la taille de l'image en pixels. Le champ `data` contient l'image elle-même; c'est un tableau de `height` tableaux dont chacun a une taille de $4 \cdot$ `width` octets.

Les images indexées sont représentées par la structure suivante :

```
struct pal_image {
    unsigned char **data;
    int height, width;
    unsigned char *pal;
    int pal_len;
};
```

Les champs `height`, `width` et `data` sont comme ci-dessus, sauf que chaque élément de `data` est un tableau d'indices et a une taille de `width` octets. Le champ `pal_len` indique la taille de la palette, et le champ `pal` contient la palette sous format RGB; il a une taille de $3 \cdot$ `pal_len` octets.

Le fichier `pngtest.c` contient un petit programme qui utilise la bibliothèque `pngio.c`. Il lit une image au format RGBA puis écrit une image indexée. Vous pouvez vous en servir comme exemple, ou comme source de copier-coller pour votre solution au projet. Vous pouvez le compiler à l'aide de la commande :

```
gcc -Wall pngtest.c pngio.c -lpng
```

1.3.2 Programme de conversion

Je vous fournis un programme nommé `writergba` qui convertit une image au format JPEG, GIF ou PNG en une image en un format utilisable par ma bibliothèque. Pour convertir une image nommée `image.jpeg` en une image `image.png`, faites

```
./writergba image.jpeg image.png
```

2 Conversion simple

Le but de cette partie est d'écrire un programme qui prend deux noms de fichier en ligne de commande. Le premier indique le nom d'un fichier existant au format RGBA; le deuxième indique le nom du fichier indexé à créer.

Votre programme lira le fichier d'entrée, puis calculera l'ensemble des couleurs qui apparaissent dans l'image. Pour cela, il faudra parcourir l'image, et pour chaque pixel, vérifier si la couleur est déjà dans l'ensemble ; si elle n'y est pas, il faudra l'y ajouter. Vous êtes libres du choix de la structure de données qui représente l'ensemble ; vous pourrez par exemple utiliser :

- un *slice* tout bête, et faire des recherches linéaires ;
- un *slice* trié, et faire des recherches par dichotomie ;
- un arbre binaire ;
- un arbre AVL ;
- un B-tree d'arité soigneusement choisie pour que chaque nœud remplisse une ligne de cache [2].

Le choix de la structure de données sera pris en compte lors de l'évaluation de votre projet.

Votre programme affichera ensuite le nombre de couleurs distinctes trouvées ; si ce nombre est strictement supérieur à 256, il affichera un message d'erreur et terminera. Dans le cas contraire, il convertira l'image en une image indexée et l'écrira au format PNG.

3 Réduction à une palette fixée

Le programme que vous avez obtenu produit une image indexée à partir d'une image qui contenait 256 couleurs distinctes au plus. Que peut-on faire pour une image qui contient trop de couleurs ? La solution la plus simple consiste à choisir une palette fixée et à associer à chaque couleur l'élément de la palette qui en est le plus proche.

Étant données deux couleurs $C_1 = (R, G, B)$ et $C_2 = (R', G', B')$, on définit la *distance euclidienne* entre C_1 et C_2 par

$$\mathrm{d}(C_1, C_2) = \sqrt{(R - R')^2 + (G - G')^2 + (B - B')^2}.$$

Votre programme choisira une palette P, puis, pour chaque pixel de l'image d'origine, il déterminera la couleur de P qui est la plus proche (au sens de la distance euclidienne) de la couleur du pixel ; il remplacera la couleur du pixel par cette dernière. Il écrira ensuite une image à palette comme ci-dessus. (Remarquez qu'il n'est pas nécessaire de calculer des racines carrées, on peut comparer les carrés des distances euclidiennes.)

Idéalement, votre programme devrait accepter des options de ligne de commande qui déterminent s'il se comporte comme dans la partie ci dessous ou s'il utilise une palette fixée. Je vous suggère d'implémenter une ou plusieurs parmi les palettes traditionnelles suivantes :

- la palette saturée (8 couleurs) :

$$\{(255i, 255j, 255k) \mid i, j, k \in \{0, 1\}\};$$

- la palette CGA (16 couleurs) :

$$\left\{(\lfloor 127.5i + 127.5l \rfloor, \lfloor 127.5j + 127.5l \rfloor, \lfloor 127.5k + 127.5l \rfloor) \;\middle|\; \begin{array}{l} i, j, k, l \in \{0, 1\} \\ (i, j, k, l) \neq (0, 0, 0, 1) \end{array}\right\}$$
$$\cup \{(192, 192, 192)\};$$

2. Je blague.

– la palette 4-4-4 (64 couleurs) :

$$\{(85i, 85j, 85k) \mid i, j, k = 0 \cdots 3\};$$

– la palette 6-6-6 (216 couleurs) :

$$\{(51i, 51j, 51k) \mid i, j, k = 0 \cdots 5\};$$

– la palette 6-7-6 (252 couleurs), où la composante verte peut en plus prendre la valeur 212 ;
– la palette noir et blanc,

$$\{(0, 0, 0), (255, 255, 255)\};$$

– la palette à 256 niveaux de gris,

$$\{(i, i, i) \mid i = 0 \cdots 255\}.$$

FIGURE 4 — Une image restreinte à la palette saturée, la palette CGA, la palette 4-4-4. L'image d'origine est à droite.

4 Palette dynamique

Plutôt qu'utiliser une palette statique, on peut aussi calculer une palette *ad hoc* pour l'image considérée. La technique la plus simple consiste à prendre les n couleurs les plus utilisées dans l'image (où n est la taille de palette désirée). Pour cela, il faudra d'abord construire un histogramme des couleurs utilisées dans l'image, c'est à dire un ensemble de paires (c, k) où c est une couleur et k est le nombre de fois que la couleur apparaît dans l'image ; il faudra ensuite trier cet histogramme, et se restreindre à ses k premiers éléments.

Idéalement, le paramètre n sera indiqué par une option de ligne de commande.

Il existe de meilleurs algorithmes de quantification ; je ne suis pas spécialiste, mais j'ai entendu parler de l'algorithme *median cut*.

5 Extensions

Toutes les extensions au projet seront les bienvenues. La liste ci-dessous n'est pas exhaustive.

Distance entre couleurs La distance euclidienne dans l'espace RGB est-elle la seule notion de distance sur les couleurs ? Quelles sont les autres notions de distance définies dans la littérature, pouvez-vous en implémenter une, et donner un exemple d'image où votre distance produits des résultats visuellement différents de la distance euclidienne ? Sont-ils meilleurs ?

Algorithme de décimation Quels sont les meilleurs algorithmes pour calculer une palette dynamique ? Je ne suis pas spécialiste, je ne connais que le *median cut*, mais je suis sûr qu'on peut en trouver d'autres.

Transparence La composante *A* des images RGBA contient une information de transparence. Pouvez-vous utiliser cette information pour composer l'image avec une couleur de fond spécifiée par l'utilisateur avant de la convertir au format indexé ?

Mise à l'échelle Pourquoi ne pas réduire les dimensions de l'image tant qu'on y est ? Attention cependant à utiliser le bon algorithme de passage à l'échelle — ils ont des propriétés différentes, et certains créent des artefacts ou rendent les images floues.

Diffusion d'erreurs Floyd-Stenberg ?

6 Modalités de rendu

Vous me soumettres une archive `.tar.gz` contenant :

- le code source ;
- un fichier texte nommé `README` expliquant comment compiler et exécuter votre code ;
- un rapport de quelques pages au format PDF indiquant notamment les choix d'implémentation que vous avez faits, les différences entre votre solution et les algorithmes décrits dans ce document, les extensions que vous aurez implémentées, et toute information qui peut m'aider dans l'évaluation de votre projet (ou tout simplement qui peut m'intéresser).

L'archive devra s'appeler *nom1-nom2*.tar.gz, et s'extraire dans un sous-répertoire *nom1-nom2* du répertoire courant. Par exemple, si vous vous appelez *Arlo Guthrie* et *Janis Joplin*, votre archive devra porter le nom `guthrie-joplin.tar.gz` et son extraction devra créer un répertoire `guthrie-joplin` contenant tous les fichiers que vous me soumettrez.

Vous me soumettrez votre solution par courrier électronique à mon adresse. Le courrier que vous m'enverrez devra impérativement avoir le sujet « `Projet L3 maths` » et l'archive sera en attachement.