Réduction de palette

L'objectif de ce TP est d'écrire un programme qui, partant d'une image, produise l'image la plus *ressemblante* possible mais n'utilisant qu'un nombre réduit de couleurs différentes. Commençons par préciser le problème.

Note préliminaire

Vous rendrez votre travail sur PROF sous la forme d'une archive contenant :

- un fichier `README` indiquant les auteurs du travail, une description du travail réalisé (ce qui marche, comment vous avez vérifié que cela marche effectivement, etc.) et autres indications utiles,
- votre code (en ocaml, c, ou java) et son `Makefile`,
- toutes les images de tests que vous aurez créées.

En particulier, vous ne rendrez pas les fichiers qui vous sont fournis !

Présentation du problème

Nous allons nous intéresser ici à des images en niveaux de gris[1].

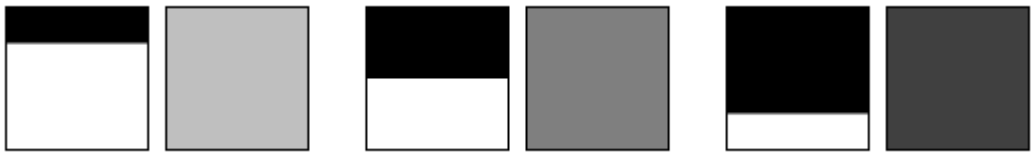
Nous appellerons *palette* d'une image l'ensemble des couleurs (ici, les niveaux de gris) des pixels de l'image.

Pour comparer deux images de mêmes dimensions (et quantifier à quel point elles se ressemblent), nous comparerons chaque pixel de la première image avec le pixel de même position dans la seconde image. Si le pixel a un niveau de gris g_1 dans la première image et un niveau g_2 dans la seconde, nous compterons l'écart $(g_1 - g_2)^2$ entre ces deux pixels. Nous définirons la *distance* entre les deux images comme la somme des écarts entre leurs pixels.

Pour une image donnée, nous appellerons sa réduction à une palette de k gris l'image à distance minimale de l'originale et ayant au plus k gris différents.

Dès lors que les palettes des images initiale et finale sont fixées, toute la transformation de l'image est décidée : pour chaque pixel de l'image initiale, nous allons choisir le gris de la palette finale le plus proche du gris initial. Le problème se ramène donc au choix de la palette de k gris.

La palette finale ne dépend cependant pas que de la palette initiale mais aussi du nombre de pixels ayant les différentes couleurs de la palette initiale, comme l'illustre la figure suivante : la couleur fusionnée doit être plus proche de la majorité des pixels.



3 images en 2 couleurs et leurs versions réduites à 1 couleur

Exemple de réduction de palette

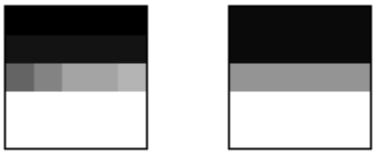
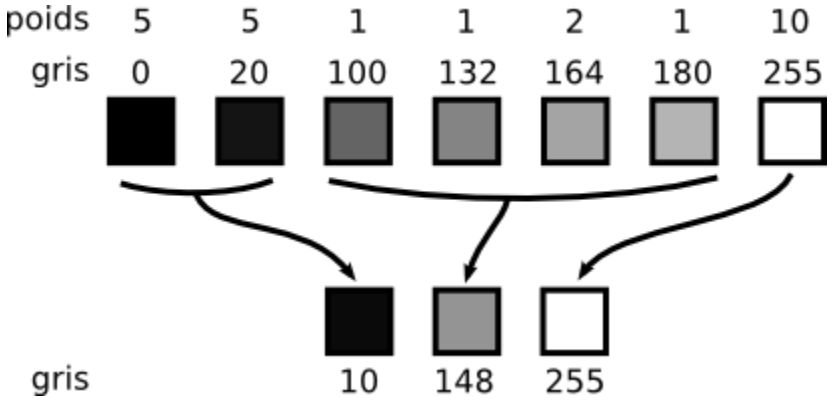


Image jouet et sa réduction à 3 gris

Illustrons la réduction de palette sur l'image de 25 pixels montrée ci-dessus grossie. Cette image utilise seulement 7 niveaux de gris différents. La figure ci-dessous montre comment réduire ces 7 gris à 3.

Le *poids* associé à chaque couleur est le nombre de pixels de cette couleur.



Réduction d'une palette de 7 couleurs à 3 couleurs

Après calcul, la meilleure palette à 3 couleurs a les niveaux de gris 10, 148 et 255. Comme l'indique la figure, tous les pixels ayant une couleur 0 ou 20 ont la couleur 10 après réduction, etc.

La distance entre les deux images peut donc se calculer : $(0-10)^2\times5 + (20-10)^2\times5 + (100-148)^2\times1 + (132-148)^2\times1 + (164-148)^2\times2 + (180-148)^2\times1 + (255-255)^2\times10 = 5096$.

Implémentation

Les différentes fonctions à implémenter demandent beaucoup de soin pour être correctes. Il est indispensable de s'appuyer sur des tests intensifs, à tous les niveaux pour valider le travail.

1. Implémenter les fonctions suivantes.

   - `meilleurGris` calcule, pour un intervalle de la palette, le niveau gris qui donne la distance minimale après fusion ; cette fonction prendra en argument les indices des premier et dernier gris à fusionner. Le « meilleur gris » est la moyenne pondérée des gris fusionnés.

     N.B. : la moyenne n'est pas nécessairement un nombre entier, il faudra l'arrondir à l'entier le plus proche[2].

   - `distanceMin` calcule la distance minimale qu'entraînera la fusion d'un intervalle de la palette (en ne considérant que les pixels ayant ces niveaux de gris).

2. Décomposer la distance entre l'image de départ et celle d'arrivée suite à la réduction de la palette à k couleurs en :
   - la distance due à la fusion d'un intervalle,
   - et la réduction du reste de la palette à $k - 1$ couleurs.

   En déduire une fonction qui calcule la distance minimale due à la réduction de la palette de n à k couleurs.

   En suivant les principes de la programmation dynamique, cet algorithme devra être polynomial (on précisera la complexité).

3. Afin de reconstruire la palette cible, exploiter les informations calculées par la fonction précédente pour identifier les intervalles à fusionner.

4. Finalement, donner une fonction qui prenne un vecteur de pixels, calcule la palette et les poids, et utilise la fonction précédente pour calculer la nouvelle palette et retourne le vecteur de pixels de l'image réduite.

   Finir en fournissant un programme qui charge une image et produise une image ayant (au plus) k niveaux de gris différents, avec k passé en argument.

Les questions précédentes ont supposé implicitement que la palette est triée, c'est-à-dire que les couleurs de la palette vont du noir (ou du gris le plus sombre) au blanc (ou gris le plus clair). Cela implique que nous n'avons fusionné que des couleurs voisines.

5. Justifier pourquoi la distance minimale est atteinte en fusionnant des couleurs voisines (plutôt que des couleurs quelconques).

Utilisation

Une fois le programme de la section précédente écrit, on pourra tester son action sur des images. Comme il existe de très nombreux formats d'image, vous pourrez utiliser le petit utilitaire `convert` d'ImageMagick pour les convertir dans le format qui vous arrangera le plus.

Deux formats sont particulièrement appréciables pour les tests : PGM et Gray. Dans ce dernier format, le fichier contient directement la suite des niveaux de gris en binaire[3] (en particulier, ce format ne contient pas les informations de taille de l'image ; mais l'application qui nous intéresse ici n'en a pas besoin). Par exemple, pour l'image montrée en exemple :

```
convert init-palettes.pgm init-palettes.gray
./reduction < init-palettes.gray > fin-palettes.gray
convert -depth 8 -size 5x5 -compress none fin-palettes.gray fin-palettes.pgm
```

parce que l'image fait 5×5 pixels, avec 8 bits. `-compress none` permet d'avoir une image PGM lisible avec votre éditeur de texte.

On vous fournit quelques images de test, notamment l'image du babouin ci-dessous, souvent utilisée dans la recherche en image. Vous pourrez suivre ces exemples pour concevoir vos tests et mettre au point votre code.

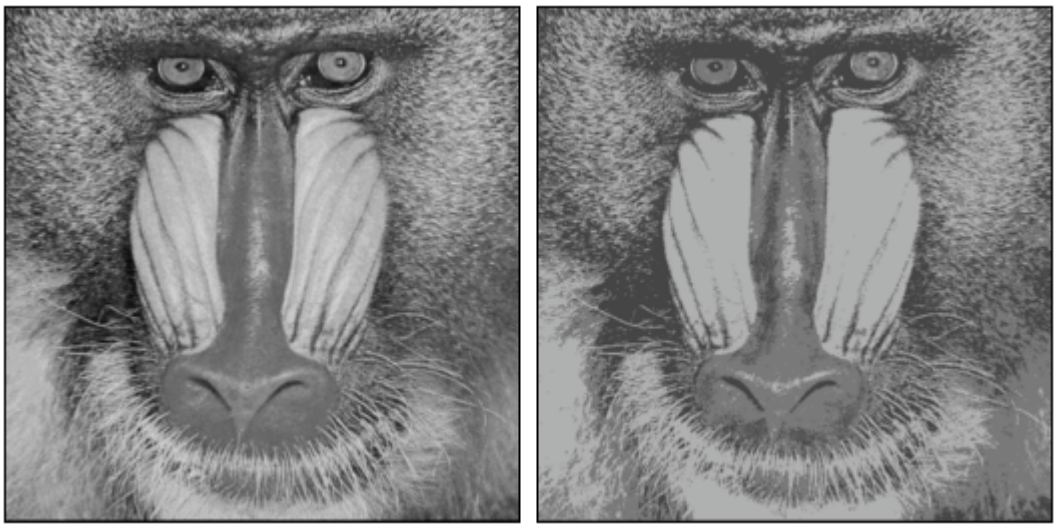


Image standard de test « baboon » et sa réduction à 3 couleurs

Références

Un algorithme un peu moins naïf de réduction de palette est décrit dans l'article Grey level reduction for segmentation, thresholding and binarisation of images based on optimal

partitioning on an interval, Quweider, M.K., Scargle, J.D. and Jackson, B.

---

1. En cas de besoin, reportez-vous à la page wikipédia sur le niveau de gris. La réduction de palette serait tout aussi intéressante pour des images à proprement parler en couleurs (RVB). Considérer uniquement les images en niveaux de gris permet de simplifier le problème.↩

2. Notez que les deux arrondis possibles de $1,5$ (1 et 2, donc) sont aussi bons l'un que l'autre. Dans ce cas, il n'y a pas une unique meilleure solution. Lorsque vous comparerez vos résultats entre binômes, pensez à comparer les distances au cas où les couleurs choisies ne coïncideraient pas.↩

3. Pour l'illustrer par du code C, c'est directement `char image[hauteur*largeur]` tel qu'il serait représenté en mémoire, si chaque pixel est codé sur un octet (`char`).↩

Dernière modification : 29 septembre 2014.