Introduction à la Programmation

—

Examen final sur machine

—

G1: monasse*(at)*imagine.enpc.fr
G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*onera.fr
G4: antoine.recanati*(at)*inria.fr
G5: bourkia*(at)*imagine.enpc.fr
G6: fadhela.kerdjoudj*(at)*u-pem.fr

—

11/01/17

—

1 Enoncé

1.1 Quantification des couleurs d'une image par algorithmes des k-moyennes

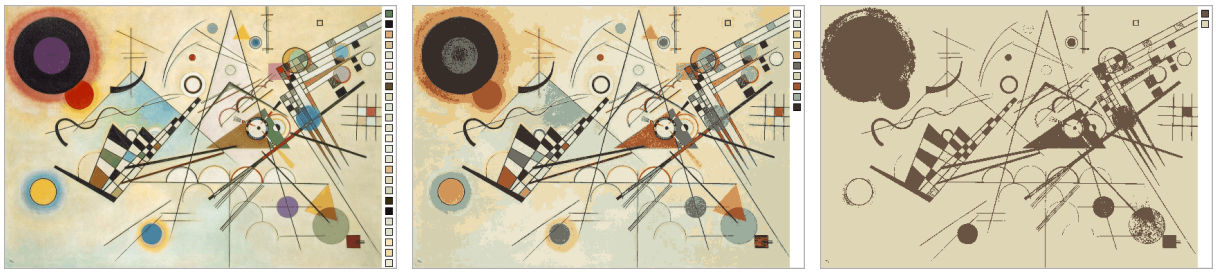
Un problème fréquent en traitement de données est de quantifier ces données en un nombre petit k, défini par l'utilisateur. On l'applique à la quantification des couleurs d'une image : représenter une image en utilisant au plus k couleurs, voir figure.

L'idée est de partitionner les couleurs de l'image, représentées par des points dans l'espace à trois dimensions RGB (rouge/vert/bleu), en k *clusters*. Un point de chaque cluster est plus proche du centroïde (barycentre) de son cluster que du centroïde de tout autre cluster. Ainsi, chaque pixel reçoit la couleur du centroïde du cluster auquel il se rattache.

L'algorithme des k-moyennes (k-means en anglais), très simple de principe, résoud ce problème difficile de façon approchée :

1. Sélectionner au hasard k points distincts parmi les couleurs présentes dans l'image.
2. *Affectation* : pour chaque pixel de l'image, trouver la couleur la plus proche parmi les k (son cluster).
3. *Moyennage* : calculer les centroïdes des clusters, remplacer les k points par ces centroïdes.
4. Si certains points ont changé de cluster à l'étape 2, retourner en 2.
5. Affecter à chaque pixel de l'image la couleur du cluster auquel il est rattaché.

Il est plus important de livrer un code clair (commenté et indenté) et qui compile sans warning, même s'il ne répond pas à toutes les questions. Pour cela, vérifiez à chaque étape que votre programme compile et se lance correctement. Pour les classes que vous définissez, ne mettez en public que le minimum nécessaire. N'oubliez pas de marquer `const` les méthodes qui peuvent l'être.



$k = 200$ $k = 10$ $k = 2$

FIGURE 1 – Quantification en k couleurs d'une image. Avec $k = 200$, il n'y a guère de différence visible avec l'image originale.

Créez un projet Imagine++ avec un fichier contenant le `main`. Prenez le `CMakeLists.txt` d'un projet existant et adaptez-le. Cherchez aussi une image en couleurs pour la copier dans votre dossier.

1.1.1 Affichage de l'image

1. Dans votre fonction `main`, chargez votre image en couleur et affichez-la dans un fenêtre de taille appropriée.

   Votre image pourra être gérée soit sous forme de trois tableaux r, g, b de `byte`, soit sous comme un tableau composite rgb, soit comme un tableau de `Color`.
2. Demandez à l'utilisateur d'entrer le nombre k de clusters. Si l'utilisateur entre 0, on termine le programme, sinon on affiche l'image quantifiée (en suivant les instructions de la suite de l'énoncé) et on repose la question.

1.1.2 Initialisation

3. Dans un fichier séparé, créez la classe `Cluster`. Celle-ci stocke juste la somme des r, g, b des points affectés et leur nombre, mais pas les points eux-mêmes.
4. Créez un constructeur pour cette classe.
5. Créez un opérateur `==` pour la comparaison de deux clusters.
6. Créez une méthode `get` qui renvoie la couleur moyenne du cluster (obtenue en divisant les champs r, g, b par le nombre de points).
7. Créez une méthode `distance` qui renvoie la distance d'une couleur à la couleur moyenne du cluster. On peut se contenter du carré de la distance euclidienne, sans prendre de racine carrée, puisque seul l'ordre importe.
8. Créez une méthode `add` qui ajoute un point au cluster.
9. Créez une fonction `init_kmeans` qui crée les k clusters initiaux en tirant k points *distincts* de l'image. Si l'image est pauvre en couleurs dès le départ, on renonce à avoir k clusters exactement, on se contente de moins : on abandonne au bout de 10000 tirages aléatoires. La fonction `init_kmeans` est donc susceptible de diminuer le nombre k de clusters.

1.1.3 Itérations du k-means

10. Écrire une fonction `find_cluster` qui renvoie l'indice du cluster, parmi le tableau des clusters, de couleur moyenne la plus proche d'une couleur donnée.
11. Écrire une fonction `iter_kmeans` qui crée les nouveaux clusters en réaffectant tous les pixels de l'image. Elle enregistre le nombre de pixels qui changent de cluster. On arrête la boucle au bout de 30 itérations ou lorsque le nombre de pixels qui changent de cluster est inférieur à 2%. Après chaque itération, afficher le pourcentage des pixels changeant de couleur. Par exemple :

    ```
    Entrez le nombre de couleurs (0 pour terminer) : 2
    100%  27%  5%  1%
    Iterations : 4
    ```

    (utilisez `cout << flush` pour afficher immédiatement sans attendre le `endl`)

    Remarque : il est nécessaire d'avoir un tableau auxiliaire des nouveaux clusters en construction à chaque itération.

1.1.4 Affichage

12. Écrire une fonction `quantify` qui renvoie une nouvelle image en ne prenant que les couleurs des centroïdes des clusters.
13. Afficher l'image quantifiée, ainsi que les couleurs utilisées à droite, comme sur la figure.

Important : Quand vous avez terminé, créez une archive du projet *à votre nom et numéro de groupe* en ZIP, RAR, TGZ ou 7z, par exemple `G1_Kandinsky_Wassily_Final.zip`. N'incluez que les fichiers source et le CMakeLists.txt, pas les fichiers binaires créés par Cmake. Ne partez pas avant que le surveillant ne soit passé vous voir pour copier cette archive sur clé USB.