ImageJ - TP 4
À vos palettes !

Exercice 4.1 *ImageJ et couleurs indexées*

Le but de cet exercice est de comprendre comment imageJ traite les couleurs indexées :
— Sous imageJ, ouvrir une image couleur au format GIF.
— Regardez son type (`Image->Type`). Que cela signifie-t-il ?
— ImageJ appelle LUT (Look-Up Table) les palettes. Affichez la LUT de l'image ouverte. Modifiez-la. Que pouvez-vous en déduire ?

Prendre l'image `fleur.png`, la passer en mode 8-bits. Que se passe-t-il ?
Quel mode faut-il choisir ? À quoi correspondent les modes 16 bits et 32 bits ?

Exercice 4.2 *Quantification de couleurs*

L'objectif de cet exercice est de voir comment réduire le nombre de couleurs d'une image RGB à 256.

1. En utilisant la méthode `couleurVersNg` du TP précédent, proposez un plugin qui transforme une image en niveaux de gris. Sauvegardez cette image. Transformez maintenant l'image couleur d'origine en niveau de gris par l'interface d'imageJ (cf. exercice précédent). Comparez les tailles des images sauvegardées. Que constatez-vous ?
2. Écrire un plugin qui extrait la palette (LUT) d'une image en couleurs indexées et la sauvegarde dans un fichier. Ecrire un autre plugin le plugin pour appliquer une palette sauvegardée à une image en couleurs indexées.
3. Proposez une méthode qui crée une palette "standard" c'est-à-dire une palette qui pourrait s'appliquer à toutes les images. Il s'agira d'écrire un algorithme qui partagera le cube RGB en 256 cases (rappel : on perçoit mieux le vert, puis le rouge que le bleu, cf. les coefficients de la méthode `couleurVersNg`) et qui choisira un représentant de chaque case en couleur de référence. Sauvegarder cette palette.
4. Écrire un plugin qui réduit le nombre de couleurs d'une image RGB à 256 en appliquant la palette que vous venez de créer. La couleur RGB de chaque pixel devra être remplacée par la couleur de la palette la plus proche (cf du TP précédent).
5. *Pour aller plus loin...*
   Il s'agit maintenant de créer une palette personnalisée pour chaque image. Pour cela, on doit trouver les 256 couleurs les plus représentatives d'une image. Il faut donc découper le cube RGB en fonction de ces couleurs. Un algorithme classique, appelé `k-means`, permet de découper un espace en k parties (appelées clusters), centrées autour des valeurs les plus représentatives. Cet algorithme est le suivant :
   (a) choisir le nombre de clusters (k),
   (b) créer k points (qui vont être les représentants des clusters ou centroïdes),
   (c) initialiser au hasard la position de ces k centroïdes,
   (d) calculer la distance des points de l'espace (pour nous, des couleurs) à ces k centroïdes,
   (e) affecter chaque couleur au cluster du plus proche centroïde,
   (f) pour chaque cluster, recalculer les positions des centroïdes par moyennage des points contenus dans celui-ci,

(g) recommencer au point d tant qu'il reste des changements dans les positions des centroïdes.

Le fichier `kmeans.java` contient une implémentation de cet algorithme (trouvée sur internet sur le site `stackoverflow`) dont vous pouvez vous servir librement.