Projet Tutoré : Jouer avec les images

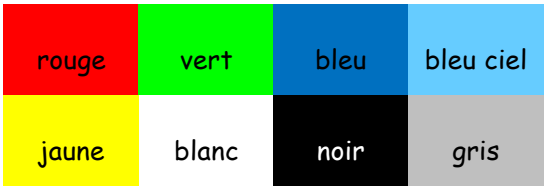
A rendre avant le mardi 13 janvier 2015 à 17h

1. Introduction :

Une image est un ensemble de pixels dont chacun est défini par trois valeurs, que l'on note R, V, et B. Ces valeurs représentent l'intensité des couleurs rouge, vert et bleu et déterminent la couleur du pixel. Elles sont comprises entre 0 et 255. Chaque couleur est associée à un unique triplet (R, V, B). On appelle composante rouge d'un pixel (respectivement verte, bleue), la valeur R (respectivement V, B) dans le triplet (R, V, B).

Dans ce projet, une image sera représentée par trois tableaux 2D, notés tabR, tabV, tabB, tous de même taille (la taille de l'image), et représentant chacun respectivement la composante couleur rouge, verte, ou bleue de l'image. Ainsi, chaque case d'un de ces tableaux contiendra la valeur R du pixel si l'on considère le tableau tabR, la valeur V si l'on considère le tableau tabV, ou bien la valeur B si l'on considère le tableau tabB.

Par exemple, l'image suivante est représentée par les trois tableaux qui suivent, tous des tableaux de taille 4x2, avec chaque carré représentant un pixel. Les données sont stockées ligne par ligne.



tabR

255	0	0	102
255	255	0	191

tabV

0	255	0	204
255	255	0	191

tabB

0	0	255	255
0	255	0	191

On remarque par exemple que le pixel jaune de l'image, en case [1,0], contient du rouge et du vert, mais pas de bleu, puisque la valeur dans la case de même indice des tableaux **tabR** et **tabV** n'est pas nulle, alors qu'elle l'est dans le tableau **tabB**. Le gris par contre comprend à valeurs égales R,V et B, comme le noir, mais à un niveau supérieur (191 au lieu du minimum 0). Et on voit que le bleu ciel a peu de rouge (102), beaucoup de vert (204) et le bleu est au maximum.

2. But du projet :

Le but de ce projet est de vous offrir une initiation au traitement de l'image, tout en développant vos compétences de programmation, surtout en ce qui concerne la manipulation de tableaux à deux dimensions et des sous-programmes dans un premier temps, puis des classes dans un second temps. L'objectif est de pouvoir manipuler une image. Vous allez comprendre comment elle est codée et voir ce qu'il est possible de faire avec. Vous apprendrez entre autres à travailler les couleurs (contraste, inversion de couleurs), à travailler sur la géométrie de l'image (symétrie, rotation) et à appliquer certains filtres, par exemple pour rendre une image floue ou encore pour détecter les contours des objets de l'image.

Nous mettons à votre disposition (sous Dokéos) plusieurs images que vous pourrez utiliser pour ce projet. Elles sont toutes carrées, de taille égale ou inférieure à 150 pixels par 150 pixels, et

codées au format ppm (« *portable pixmap* »). Pour plus de détails sur ce format, vous pourrez lire l'annexe disponible sous Dokéos.

Si vous le souhaitez, vous pouvez également utiliser vos propres images. Pour cela, il faudra les convertir au format .ppm (avec Gimp par exemple) et les redimensionner pour qu'elles ne dépassent pas 150 pixels par coté. Attention lors de la sauvegarde de vos images : il faut qu'elles soient encodées en ASCII et non en binaire. Pour être sûr que l'image est codée en ASCII, vous pourrez l'ouvrir avec un éditeur de texte et vérifier que la première ligne est bien P3 (et non P6).

Vous pourrez visualiser vos images avec Gimp ou XnView, qui sont des visionneurs d'images gratuits et disponibles sur internet, très simples d'installation et d'utilisation.

Sous-Programmes de lecture et écriture d'images :

Les sous-programmes de lecture et écriture des images sous forme de fichier sont donnés dans les fichiers manipFichier.h et manipFichier.cpp.

Les images sont représentées par des tableaux 2D d'entiers, de capacité maximale **MAX** x **MAX**, avec **MAX** une constante du programme définie à 150.

Les deux sous-programmes, **loadPicture** et **writePicture**, permettent de charger et de créer une image..

Les spécifications algorithmiques de ces sous-programmes sont les suivantes :

```
// loadPicture : prend le nom d'un fichier contenant une image au format ppm,
// remplit 3 tableaux 2D avec les données de cette image, et renvoie la taille
// effective de ces tableaux (identique pour les 3 tableaux).
// paramètres (D) image : chaîne de caractères (nom du fichier ppm contenant l'image)
//            (R) tabR, tabV, tabB : tableaux 2D d'entiers de taille MAX
//            (R) taille : entier
void loadPicture(string image, int tabR[][MAX], int tabV[][MAX], int tabB[][MAX],
                    int & taille);

// writePicture : prend 3 tableaux 2D, leur taille effective (identique pour les 3),
// crée une image au format ppm à partir des données de ces tableaux, et l'enregistre
// dans un fichier.
// paramètres (D) tabR, tabV, tabB : tableaux 2D d'entiers de taille MAX
//            (D) taille : entier
//            (D) image : chaîne de caractères (nom du fichier ppm contenant l'image)
void writePicture(const int tabR[][MAX], const int tabV[][MAX],
                    const int tabB[][MAX], int taille, string image);
```

Les appels suivants permettent de récupèrer les données de l'image contenue dans le fichier de votre choix (ici coco.ppm) et d'enregistre une copie de cette image dans le fichier copie.ppm à partir des tableaux ainsi initialisés :

```
string image = "coco.ppm";
loadPicture(image, tabRouge, tabVert, tabBleu, taille);
writePicture(tabRouge, tabVert, tabBleu, taille, "copie.ppm");
```

A vous maintenant !

Partie 1 : Opérations sur les couleurs

Voici les fonctionnalités que votre programme doit permettre de réaliser à partir d'une image contenue dans le fichier « XX ».

1. Création d'un fichier « XX-copie » contenant une copie conforme d'une image.
2. Création d'une image « XX-rouge.ppm » contenant uniquement la composante de couleur rouge de l'image (c'est à dire que les valeurs des composantes V et B valent toutes 0). On créera de même les images « XX-vert.ppm » et « nomXXfic-bleu.ppm » contenant uniquement les composantes vertes et bleues respectivement.
3. Recherche de la présence de la couleur rouge dans une image
4. Création une image «XX-sansRouge.ppm » qui sera une copie de l'image après avoir retiré la couleur rouge, mais seulement si cette image contient du rouge (sinon c'est un traitement inutile !).
5. Création du négatif d'une image. Pour avoir le négatif d'une image, il suffit d'appliquer à chaque pixel la transformation suivante : si **x** est la valeur d'une composante d'un pixel donné, alors **255–x** est la valeur de la composante du pixel du négatif. Complétez votre programme afin de créer le négatif « XX-negatif.ppm » de votre image.
6. Binariser une image. Pour transformer une image couleur en noir et blanc (cette opération s'appelle la binarisation), on fixe un entier SEUIL et une composant. Les composantes de l'image binarisée sont calculées de la façon suivante : si la composante choisie d'un pixel est supérieure au seuil, alors on met les trois composantes RGB du pixel à 255, sinon on les met toutes à 0. Discutez des résultats pour différentes images selon la valeur du SEUIL et la couleur choisie.
7. Modifier la luminosité d'une image. Chaque composante de chaque pixel est multipliée par un même réel strictement positif **a**. Si après cette opération, il y a un pixel dont une composante est supérieure à 255 (valeur maximale pour R, G ou B), alors on mettra cette composante à 255. On obtient ainsi une nouvelle image : « XX-luminosite-a.ppm » Discutez des effets selon la valeur de a.

Partie 2 : Transformations géométriques

8. Créer le symétrique d'une image par rapport à un axe vertical (resp. horizontal). Le symétrique d'une image par rapport à l'axe vertival (resp. horizontal) est l'image obtenue à partir de cette image en mettant un miroir le long d'un côté (bas ou haut) de l'image. Ainsi, la droite se retrouve à gauche et la gauche se retrouve à droite.

   L'exemple ci-dessous illustre cette notion :

Image originale

Symétrique par rapport à l'axe vertical droit

Image originale

Symétrique par rapport à un axe horizontal

Complétez votre programme afin d'obtenir les images «XX-symetrieVerticale.ppm » qui est le symétrique par rapport à un axe vertical de votre image d'origine et «XX-symetrieHorizontale.ppm » qui en est le symétrique horizontal.

9. Rotation à 90° : la rotation à 90° d'une image est l'image obtenue en tournant d'un quart de tour par rapport au sens inverse des aiguilles d'une montre l'image d'origine.

   L'exemple ci-dessous illustre cette notion :

Image originale

Image après rotation à 90°

Complétez votre programme afin d'obtenir l'image «rotation.ppm » qui est la rotation à 90° de votre image d'origine.

Partie 3 : Application de filtres

L'application d'un filtre à une image permet de modéliser bon nombre de phénomènes comme le fait de rendre une image floue (le floutage) ou encore la détection de contours.

Appliquer un filtre à une image consiste à modifier chaque pixel en fonction de ses voisins. Un filtre est représenté sous la forme d'un tableau de taille **n x n**, où **n** est un entier positif, le plus souvent égal à 3. L'origine du filtre étant la position centrale du tableau et correspondant au pixel que l'on veut modifier, un filtre 3x3 signifie que chaque pixel sera modifié en fonction de ses 8 voisins immédiats.

A titre d'exemple, considérons le filtre suivant :

1/4	0	0
0	1/4	1/4
0	1/4	0

Ce filtre modifie un pixel de la manière suivante : Si entrée(i,j) désigne le pixel en position (i,j) sur l'image de départ et sortie(i,j) le pixel en position (i,j) en sortie, alors :

sortie(i,j) = **1/4**(entrée (i-1, j-1)) + **1/4**(entrée(i, j)) + **1/4**(entrée(i+1, j)) + **1/4**(entrée(i, j+1)).

Autrement dit, le pixel en sortie est la moyenne entière des valeurs de lui même, de son voisin en haut à gauche, de celui directement à droite et de son voisin du dessous.

Par exemple, si le 1er tableau ci-dessous désigne la composante rouge d'une image, alors le 2ème représente la composante rouge de l'image après application du filtre. La valeur 3 de la case en indice [1,1] devient 77 par exemple car 0/4 + 3/4 + 178/4 + 124/4 = 77.

0	0	255	198
2	3	178	96
128	127	6	7
132	12	49	à

0	0	0	0
0	77	70	0
0	36	16	0
0	0	0	0

Les bords n'étant pas traités par un tel filtre, les pixels sur le contour du tableau sont mis à 0 sur chaque composante.

Compléter votre programme pour permettre les fonctionnalités suivantes :

10. Créer l'image «XX-filtre1.ppm » après application du filtre suivant à l'image XX :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

On fera attention à ne pas appliquer ce filtre aux bords de l'image, et donc, à initialiser toutes les valeurs aux bords à 0. Ce filtre fait la moyenne de tous les pixels voisins. Selon vous, à quoi correspond ce filtre ? Que fait-il à l'image ?

11. Créer une image «XX-filtre2.ppm » qui sera le résultat de votre image d'origine après application du filtre A donné ci-dessous, ainsi qu'une image «XX-filtre3.ppm » qui sera le résultat de votre image d'origine après application du filtre B :

1	0	-1
2	0	-2
1	0	-1

filtre A

1	2	1
0	0	0
-1	-2	-1

filtre B

On fera bien attention à ce que les valeurs des composantes des pixels restent entre 0 et 255 : si la valeur obtenue est négative, on la mettra à 0 et si elle est supérieure à 255, on la mettra à 255.

Ces deux filtres ont en fait des effets complémentaires sur une image. L'un est dit directionnel en x (selon l'axe des abscisses) et l'autre directionnel en y (selon l'axe des ordonnées). On les appelle les filtres de Sobel. Il est possible de les combiner de la manière suivante :

$$\mathrm{Sortie}(i,j) = \sqrt{\mathrm{Filtre2}^2 + \mathrm{Filtre3}^2}$$

12. Créer une image « filtre4.ppm » qui combine l'effet de ces deux filtres. Selon vous, à quoi correspond ce filtre ? Que fait-il sur l'image ?

Partie 4 : Bonus

En matière de traitement d'images, les possibilités sont multiples ! En vous documentant, vous trouverez facilement d'autres algorithmes, d'autres traitements à mettre en place.
Alors n'hésitez pas : rien n'est plus motivant qu'un challenge qu'on se donne à soi-même !

Travail à rendre : avant le 13/01/2015 à 17h :

Ce projet de traitement d'images représente un travail plus important que les mini-projets. Commencez très vite par une réunion de travail en groupe (de nos jours, les réunion de travail peuvent même se pratiquer à distance !). Vous pourrez ainsi identifier les fonctionnalités à réaliser dans un premier temps, et le squelette du programme principal afin de vous répartir le travail.

Dans sa version finale, ce projet devra être réalisé **avec des classes**.

- Afin d'avancer dans la programmation sans attendre d'être assez avancés sur les classes en TD/TP, vous devez commencer par en faire **une version sans classe**. Vous veillerez à un découpage intéressant en sous-programmes (qui sera justifié dans le dossier).
- Version avec les classes : à commencer dès que le TP12 a été traité.

Vous devez rendre (mail + impression du dossier) à l'enseignant qui corrigera votre projet une archive contenant :

- L'ensemble des fichiers constituant le programme. Les noms des programmeurs de chaque partie doivent apparaître obligatoirement en commentaire, en tête de chaque fichier.
- Un dossier comprenant, au minimum :
  - Une présentation de votre projet

- La liste des fonctionnalités que permet votre programme.
- Une discussion sur la mise en place des classes et une description des classes utilisées.
- Si nécessaire, les explications sur certains algorithmes de manipulation.
- Les explications détaillées des fonctionnalités non proposées dans le sujet.
- Les réponses aux éléments de discussion évoqués pour certains traitements.
- Une conclusion dans laquelle seront indiqués au moins les éléments suivants :
  - Organisation du travail dans le temps et répartition du travail entre les membres du groupe,
  - Avantage/inconvénient de la programmation objet (votre expérience pas le cours),
  - Avancement plus ou moins complet du travail, les difficultés rencontrées,
  - Les extensions éventuelles apportées par rapport au sujet.
  - Les améliorations envisageables,
  - Une conclusion personnelle sur ce que le projet vous a apporté.

Soutenances :

La présentation du travail réalisé fera l'objet d'une soutenance lors de la semaine finale du S1 (après les DS). Chaque étudiant exposera une partie du travail. Il vous appartient de choisir ce que vous souhaitez présenter et comment vous vous répartissez les présentations pour valoriser à la fois votre travail personnel et votre travail de groupe (5mn par personne). La soutenance se terminera par une démonstration (5 mn pas plus).

Evaluation :

L'évaluation du projet tiendra compte non seulement de la réalisation mais aussi la qualité du travail en équipe (mise en valeur en particulier lors de la soutenance).

Ainsi, les éléments qui participeront à l'évaluation seront notamment :

- Qualité du travail en équipe
- Contenu du dossier
- Qualité de la programmation
- Soutenance