TP1 : Image bitmap et opérations graphiques de base

TP à faire sous PC-Linux par binôme.
Compte-rendu à envoyer au format PDF (par exemple document *Libre Office Writer* exporté au format PDF) à l'adresse `szafran@imag.fr` avant le 4 février 2017 en indiquant comme sujet de l'e-mail :
[L3-Info] - Image - TP1 - *noms du binôme* .

Récupérez le fichier archive `tp1.tar` à l'URL suivante :
http://www-ljk.imag.fr/membres/Nicolas.Szafran/ENSEIGNEMENT/L3
puis décompactez l'archive avec la commande `tar -xvf tp1.tar` .

---

1 - La librairie graphique

Parmi les fichiers se trouve la librairie graphique nommée ***lib_image*** permettant la manipulation et l'affichage d'images bitmap.
Cette librairie est écrite en C++ et s'appuie sur la librairie ***CImg*** développée par David Tschumperlé et dont le site source se trouve à l'adresse suivante http://cimg.eu.

Dans un premier temps, créez la librairie avec la commande `make lib_image`

Ensuite pour créer l'exécutable correspondant à un fichier source nommé *prog*`.cpp`, tapez la commande `make` *prog*

2 - Formats d'image avec valeurs entières

Dans cette partie, vous allez créer puis afficher des images avec différents formats (***ImageIndexee***, ***ImageGris***, ***ImageCouleur***) où les valeurs des couleurs seront définies par des valeurs entières entre 0 et 255 (type `UINT8` de la librairie ***lib_image***).

Pour les différents formats, une image sera définie à partir de ces dimensions (largeur `L` et hauteur `H`) et éventuellement un tableau de pixels `p` de dimension $\texttt{L} \times \texttt{H}$, dans ce cas, le pixel de l'image en abscisse-colonne x ($0 \leq x \leq \texttt{L} - 1$) et ordonnée-ligne y ($0 \leq y \leq \texttt{H} - 1$) aura la valeur `p[x+L*y]`.

Pour les images en niveaux de gris (***ImageGris***), les valeurs des pixels sont de type `UINT8` c'est à dire entiers positifs sur 8 bits (entiers entre 0 et 255).

Pour les images couleur, le type `RGB` composé de 3 valeurs `r`, `g`, `b` de type `UINT8` est utilisé.

Pour les images indexées, les couleurs sont codées dans une table de valeurs `RGB`, les valeurs font références à cette table via des indices (entiers positifs de type `UINT` de la librairie ***lib_image***).

2.1 - Mode 256 niveaux de gris

En mode 256 niveaux de gris, une image est définie par :
- ses dimensions `L` et `H`,
- un tableau `p` de pixels de type `UINT8`.

Prenez connaissance des spécifications de la classe ***ImageGris*** dans le fichier `lib_image.hpp`.

A partir du fichier source image_gris1.cpp, créez l'exécutable, puis exécutez-le :

- une image `I1` de dimensions 3×2 est définie à partir d'un tableau de pixels p puis affichée à l'écran avec le facteur d'échelle 100,
- une image `I2` de dimensions 16×8 représentant 16 niveaux de gris est définie puis affichée à l'écran avec le facteur d'échelle 20, cette image est ensuite sauvegardée dans le fichier nommé `degrade16.png` .

Exercice 1 :

Complétez/modifiez le programme pour :

- créer une image avec un fond noir contenant trois carrés avec différents nuances de gris,
- créer une image de dimensions 256×100 permettant de représenter un dégradé de l'ensemble des 256 niveaux de gris.

Dans votre rapport, mettez les instructions C++ que vous avez ajoutées/modifiées, ainsi que les deux images obtenues.

2.2 - Mode RGB 24 bits

En mode RGB 24 bits, une image est définie par :

- ses dimensions `L` et `H`,
- un tableau `p` de pixels de type `RGB`.

Prenez connaissance des spécifications de la classe ***ImageCouleur***.

A partir du fichier source image_couleur1.cpp, créez l'exécutable puis exécutez-le :

- une image `I1` de dimensions 4×2 est définie à partir d'un tableau de pixels `p` puis affichée à l'écran avec le facteur d'échelle 100,
- une image `I2` de dimensions 256×256 représentant l'ensemble des couleurs RGB avec la composante *Bleu* minimale (égale à 0).

Exercice 2 :

Complétez/modifiez le programme pour :

1. créer une image du drapeau français,
2. en s'inspirant de l'image `I2`, créer cinq autres images, chacune représentant l'ensemble des couleurs en fixant une des 3 composantes RGB à sa valeur minimale (0) ou sa valeur maximale (255) :
   - les couleurs avec la composante *Bleu* fixée à 255,
   - les couleurs avec la composante *Vert* fixée à 0,
   - les couleurs avec la composante *Vert* fixée à 255,
   - les couleurs avec la composante *Rouge* fixée à 0,
   - les couleurs avec la composante *Rouge* fixée à 255.

Dans votre rapport, mettez les instructions C++ que vous avez ajoutées/modifiées, ainsi que les images obtenues.

A partir du fichier source image_composante_rgb.cpp, créez l'exécutable puis exécutez-le : ce programme charge une image couleur à partir du fichier donné en argument, puis crée une image `IR` correspond à la seule composante *Rouge*.

Exercice 3 :

Complétez/modifiez le programme pour afficher deux images supplémentaires correspondant aux deux autres composantes *Vert* et *Bleu*.
Dans votre rapport, mettez les images obtenues pour deux fichiers différents.

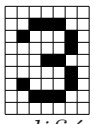
2.3 - Mode indexé

En mode indexé, une image est définie par :
- ses dimensions `L` et `H`,
- un tableau `p` de pixels,
- un table de `N` couleurs `T`.

Prenez connaissance des spécifications de la classe ***ImageIndexee***.

A partir du fichier source `image_indexee1.cpp`, créez l'exécutable puis exécutez-le :
- une image `I1` de dimensions 3×2 avec une table `T` de 3 couleurs est définie puis affichée à l'écran avec le facteur d'échelle 100,
- une image `I2` de dimensions 4×3 en noir en blanc est définie puis affichée à l'écran avec le facteur d'échelle 50.

Exercice 4 :

Complétez/modifiez le programme pour :
- obtenir non pas l'image du drapeau français mais celui de l'Irlande (vert-blanc-orange),
- pour créer une image en noir et blanc representant le caractère 3 : 

Dans votre rapport, mettez les instructions C++ que vous avez ajoutées/modifiées.

3 - Chargement / sauvegarde sur fichier

Il est possible de créer une image à partir d'un fichier image (via une méthode *constructeur*), et aussi de sauvegarder une image dans un fichier.

A partir du fichier source `image_fichier.cpp` créez l'exécutable, puis exécutez-le.
Ce programme crée l'image couleur `I1` à partir du fichier `bastille.ppm`, sauvegarde l'image `I1` dans le fichier nommé `bastille.bmp` (format BMP), affiche la taille du ce fichier, crée une deuxième image couleur `I2` à partir du fichier `bastille.bmp` puis calcule l'écart moyen et l'écart maximum entre les images `I1` et `I2`.

Exercice 5 :

1. Dans le programme, dans la chaîne de caractères `nom2`, remplacez `bastille.bmp` par `bastille.png`, et exécutez-le de nouveau (afin de sauvegarder au format *PNG*).
   Notez la taille du fichier `bastille.png` et s'il y a une différence entre les deux images.
2. en utilisant la méthode `sauvegarder_jpeg`, exporter l'image au format *JPEG* (le nom du fichier doit avoir l'extension `.jpg`) en spécifiant la qualité requise (entier entre 1 et 100).
   Testez les qualités suivantes : 100, 90, 75, 50, 10 et 1.
   Comparez les tailles des différents fichiers JPEG ainsi obtenus et les taux de compression correspondants, ainsi que les qualités visuelles des différentes images ainsi sauvegardées.
3. refaites la question précédente (*sauvegarde au format JPEG*) avec les fichiers suivants :
   - `chat.ppm`
   - `uga.ppm`
   - `alea_couleur.ppm`

Dans votre rapport, faites un tableau récapitualtif de la sauvegarde au format JPEG des 4 images (`bastille`*,* `chat`*,* `uga`*,* `alea_couleur`*) pour les différentes qualités (100,90,75,50,10 et 1), et commentez les résultats obtenus.*

Remarque : si vous souhaitez copier une image d'un de vos programmes dans votre rapport, vous pouvez utiliser les fonctions de sauvegarde.

4 - Formats d'image avec valeurs réelles

Pour créer, stocker, afficher des images, les formats avec codage en valeurs entières sont préférables.

Par contre, dès qu'on souhaite faire des opérations sur des images, il est préférable d'utiliser des formats avec codage en valeurs réelles.

Dans cette partie, on va utiliser les classes ***ImageGrisF***, ***ImageCouleurF***, classes correspondant respectivement aux classes ***ImageGris***, ***ImageCouleur*** mais avec les composantes des couleurs codées avec le type `float` avec la correspondance suivante entre valeur réelle v_f de type `float` et valeur v_e de type `UINT8` :

$$v_e \mapsto v_f = \frac{(\texttt{float})v_e}{255.0} \qquad v_f \mapsto v_e = (\texttt{UINT8})\text{Arrondi}\Big(\text{Min}\big(\text{Max}(v_f \times 255.0, 0.0), 255.0\big)\Big)$$

La classe ***ImageGrisF*** (resp. ***ImageCouleurF***) reprend les mêmes méthodes que la classe ***ImageGris*** (resp. ***ImageCouleur***).

A partir du fichier source `image_gris2.cpp`, créez l'exécutable puis exécutez-le : ce programme reprend le programme `image_gris1.cpp` mais avec des images avec composantes réelles.

Exercice 6 :

Complétez ce programme pour créer et afficher :
- une image `I3` de dimensions 401×401
  avec la valeur de chaque pixel (x, y) égale à $(x + y)/800.0$,
  c'est à dire que les valeurs de l'image seront comprises entre 0.0 et 1.0 ;
- une image `I4` de dimensions 401×401
  avec la valeur de chaque pixel (x, y) égale à $(x + y)/400.0 - 0.5$,
  c'est à dire que les valeurs de l'image seront comprises entre -0.5 et 1.5 ;

Dans votre rapport, copiez les deux images obtenues, ainsi qu'un commentaire.

A partir du fichier source `image_couleur2.cpp`, créez l'exécutable puis exécutez-le : ce programme reprend le programme `image_couleur1.cpp` mais avec des images à composantes réelles.

Exercice 7 :

Complétez ce programme pour créer et afficher des images (toutes de dimensions 401×401) :
- une image `I3R` de type `ImageGrisF` avec la valeur de chaque pixel (x, y) égale à $(200.0 + x - y)/400.0$,
- une image `I3G` de type `ImageGrisF` avec la valeur de chaque pixel (x, y) égale à $(x + y - 200.0)/400.0$,
- une image `I3B` de type `ImageGrisF` avec la valeur de chaque pixel (x, y) égale à $((x - 200.0) \times (x - 200.0) + (y - 200.0) \times (y - 200.0))/40000.0$,
- une image `I3` de type `ImageCouleurF` avec la valeur `RGBF` de chaque pixel (x, y) égale à `(I3R(x,y),I3G(x,y),I3B(x,y))`,
- une image `I4` de type `ImageGrisF` avec la valeur de chaque pixel (x, y) égale à $(\texttt{I3R(x,y)} + \texttt{I3G(x,y)} + \texttt{I3B(x,y)})/3$

Dans votre rapport, mettez les images obtenues.

L'intéret de travailler avec des composantes réelles est de pouvoir convertir des images du mode *RGB* vers un autre mode, par exemple le mode *HSV*.

A partir du fichier source `image_rgb_hsv.cpp`, créez l'exécutable puis exécutez-le :
– dans la fonction `partie1`, une image `I_HSV` est créée en fixant la première composante (teinte H) des pixels avec la teinte *vert* (1/3) et en faisant varier les deux autres composantes (S et V) entre 0.0 et 1.0 puis l'image est convertie au mode RGB (image `I_RGB`) afin d'être affichée. Cela permet de visualiser l'ensemble des couleurs correspondant à la teinte *vert* ($H = 1/3$).

Exercice 8 :

Modifiez/complétez la fonction `partie1` afin de visualiser :

1. une image avec la composante H fixée à 2/3, les deux autres composantes (S et V) variant entre 0.0 et 1.0,
2. une image avec la composante S fixée à 1, les deux autres composantes (H et V) variant entre 0.0 et 1.0,
3. une image avec la composante V fixée à 1, les deux autres composantes (H et S) variant entre 0.0 et 1.0,

Dans votre rapport, mettez les images obtenues.

Modifiez le programme précédent afin d'appeler la fonction `partie2` : dans cette fonction, on charge une image couleur (au format RGB) que l'on convertit au mode *HSV*, on modifie les composantes S et V des pixels en les fixant à la valeur 1.0 puis on repasse au mode *RGB* pour l'afficher.
Cela permet de visualiser les différentes teintes de l'image.

Exercice 9 :

Modifiez/complétez la fonction `partie2` afin de visualiser de manière pertinente :

1. la composante S d'une image en modifiant d'une certaine manière les composantes H et V,
2. la composante V d'une image en modifiant d'une certaine manière les composantes H et S.

Dans votre rapport, le code source de la fonction `partie2`*, les images obtenues pour différents exemples, et un commentaire sur les résultats obtenus.*

5 - Opérations graphiques de base

Le programme `image_trace_segment` contient l'algorithme de tracé de segment (fonction `dessiner_segment`) uniquement dans le cas $x_2 - x_1 \geq y_2 - y_1 \geq 0$.

Exercice 10 :

Compléter/modifier la fonction `dessiner_segment` afin de prendre en compte tous les cas de figures, et compléter la fonction `main` afin de tester toutes les configurations possibles.
Dans votre rapport, mettez le code complet de la fonction `dessiner_segment` *et les différents tests couvrant l'ensemble des cas.*