Premiers pas

Les exercices à réaliser sont situés dans la base de code que vous récupérez sur Moodle. Lisez bien le readme du dépôt pour comprendre comment l'utiliser. La majorité des fonctions demandées existent déjà dans OpenCV : **le but n'est pas d'utiliser les fonctions d'OpenCV mais de les coder vous même !** Nous utiliserons donc uniquement les conteneurs de base d'OpenCV et les fontions d'entrée/sortie.

Important

Au cours de ce chapitre, vous compléterez le fichier ``tpHistogram.cpp``.

Introduction à OpenCV

Sous OpenCV, une image est représentée par la classe Mat (pour Matrix). Comme le nom le suggère, il s'agit d'un tableau à 2 dimensions. La mémoire des objets de type `Mat` est gérée automatiquement avec des *smart pointer* : vous n'aurez normalement pas à vous en soucier. Le type des éléments stockés dans une image est indiqué par une constante de la forme `CV_8UC3` qui se lit *8 bit Unsigned Channel 3* (`CV_` est juste un raccourci pour OpenCV...) : donc une image couleur (3 canneaux RGB) dont les valeurs sont de type unsigned char ([0, 255]).

Important

Dans la suite on utilisera essentiellement deux types d'images :

- `CV_32FC1` ou plus simplement `CV_32F` : les images en niveaux de gris (1 cannal) codées en nombre flottant. Dans ce cas la convention est d'avoir les valeurs des pixels comprises entre 0 (noir) et 1 (blanc)
- `CV_8UC1` ou plus simplement `CV_8U` : les images en niveaux de gris (1 cannal) codées sur 1 octet non signé ([0, 255]). Dans ce cas il faudra faire attention aux dépassements de capacités (`255 + 1 == 0` sur 1 octet...)

Il existe de nombreuses façons de créer des objets de type `Mat`, notamment :

```
int height = 12;
int width = 21;
Mat image = Mat::zeros(height, width, CV_8UC1); // image initialisée à zéro
Mat copie = image.clone(); // copie d'une image existante
```

L'accès aux éléments d'une image peut se faire en récupérant un pointeur sur les données brutes ou avec la fonction générique `at` qui retourne une référence sur un pixel :

```
image.at<uchar>(3,2) = 7; // modifie la valeur à la position: ligne 3, colonne 2
uchar v = image.at<uchar>(0,0); // lit la valeur à la position (0,0);
```

La taille d'une image peut être connue avec les champs `cols` (nombre de colonnes, donc largeur de l'image) et `rows` (nombre de lignes, donc hauteur de l'image):

```
int total = 0;
for(int y = 0; y < image.rows; y++)
        for(int x = 0; x < image.cols; x++)
                total += image.at<uchar>(y, x);
```

La classe `Mat` permet de *vectoriser* certaines opérations par surcharge des opérateurs de base, ce qui permet d'éviter d'écrire des boucles `for` :

```
image = image + copie; // réalise l'addition des 2 images (somme matricielle)
image += 3; // ajoute la valeur 3 à la valeur de chaque pixel de l'image.
```

Inverser une image

Exercice 1 : Inverser une image

On souhaite définir une fonction d'inversion des niveau de gris donnée par la transformation inv définie par $\forall x \in E, inv(x) = 1 - x$ (les blancs deviennent noirs et inversement).

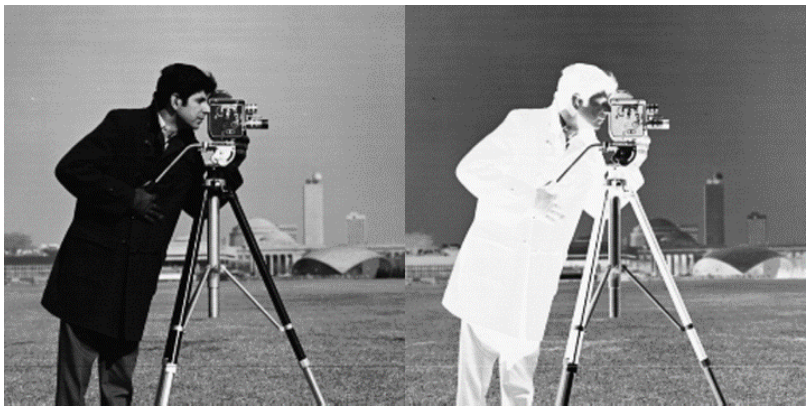

Image du cameraman à gauche et image inversée à droite.

1. Dans votre projet, complétez la fonction `inverse` du fichier `tpHistogram.cpp`. Cette fonction prend en entrée une image f et doit retourner une nouvelle image f_{inv}.
2. Exécutez la commande `make` à la racine du projet pour compiler. Un exécutable appelé `inverse` est généré dans le dossier `bin`.
3. Exécutez la commande `./inverse --help` pour voir les options possibles. **Ceci fonctionnera avec tous les exécutables générés dans les TPs.**
4. Exécutez la commande `./inverse -S` pour exécuter la commande avec l'option `-S` pour afficher le résultat. **Ceci fonctionnera avec tous les exécutables générés dans les TPs.**
5. Exécutez la commande `./test -P inverse -S` pour effectuer un test unitaire sur votre programme. En cas de problème, le programme affiche la carte des différences par rapport au résultat attendu. **Ceci fonctionnera avec tous les exécutables générés dans les TPs.**

Exercice 2 : Couleur

1. Chargez une image couleur.

2. Générez l'image négative de l'image couleur (traitement indépendant sur chaque canal).

Autres traitements

Exercice 3 : Dessiner

1. Créer une image noire de 128x128
2. Dessiner une ligne horizontale de 100 pixels de longueur
3. Dessiner une ligne oblique à 45°
4. Dessiner une ligne entre deux pixels dont on donnera les coordonnées
5. Dessiner une ligne oblique à 30°
6. Dessiner un rectangle sur la même image
7. Créer une image en dégradés de niveaux de gris

Exercice 4 : Inversion aléatoire

1. Coder une fonction qui prend en entrée une image et qui retourne une inversion aléatoire des pixels de cette image. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant cette inversion.

Exercice 5 : Extrema d'une image

1. Coder une fonction qui prend en entrée une image et qui affiche les valeurs minimales et maximales des niveaux de gris des pixels de l'image.

Exercice 6 : Couleur vers niveau de gris

1. Coder une fonction qui prend en entrée une image couleur et qui retourne une transformée de cette image en niveaux de gris. Pour un pixel p(x,y) de couleur (r,g,b), lui affecter le niveau de gris v où v = (r + g + b)/3. La fonction ne devra pas modifier l'image fournie en entrée mais créer une nouvelle image contenant la transformée.