Lecture et Ecriture d'Images numériques (Java, C++)

Alban Gaignard, alban.gaignard@cnrs.fr

11 février 2014

1 Généralités sur l'image numérique

Une **image numérique**[1] résulte en général d'une acquisition à partir d'un **capteur** (microscope, appareil photo, imageur médical, satellite, radar, télescope, radio-télescope, etc.). Les **signaux** originaux (continus) sont **numérisés** au cours des processus d'**échantillonnage** et de **quantification** pour produire des signaux discrets (en nombre fini) que l'on peut afficher par exemple sur un écran d'ordinateur. Une image numérique est donc constituée de chacun de ces signaux que l'on appelle **pixels** (compression de *picture element*). Un pixel peut être de **dimension** 1 (1D) si l'on souhaite représenter une valeur entière sur une échelle de gris (photo noir et blanc) ou peut être de plusieurs dimensions pour représenter une information de couleur (composantes rouge, vert et bleu par exemple). De même une image peut comporter plusieurs dimensions en fonction de ce que l'on mesure ou souhaite représenter (2D pour une représentation graphique plane, 3D pour l'information de volume, 2D+t (=3D) pour une séquence vidéo, 3D+t pour l'imagerie cardiaque, etc.).

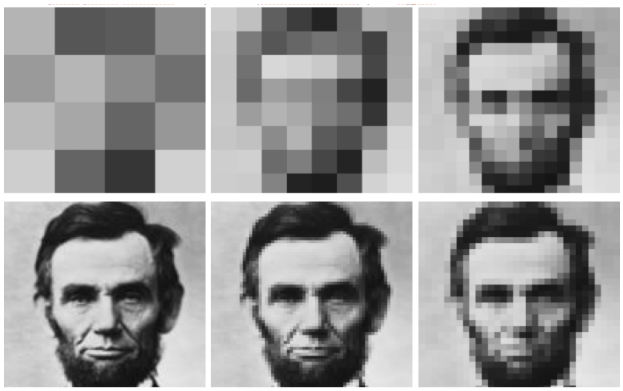

Figure 1 – Différents niveaux de détail pour différents niveaux d'échantillonage

1. à partir du cours de Diane Lingrand et Joël Leroux

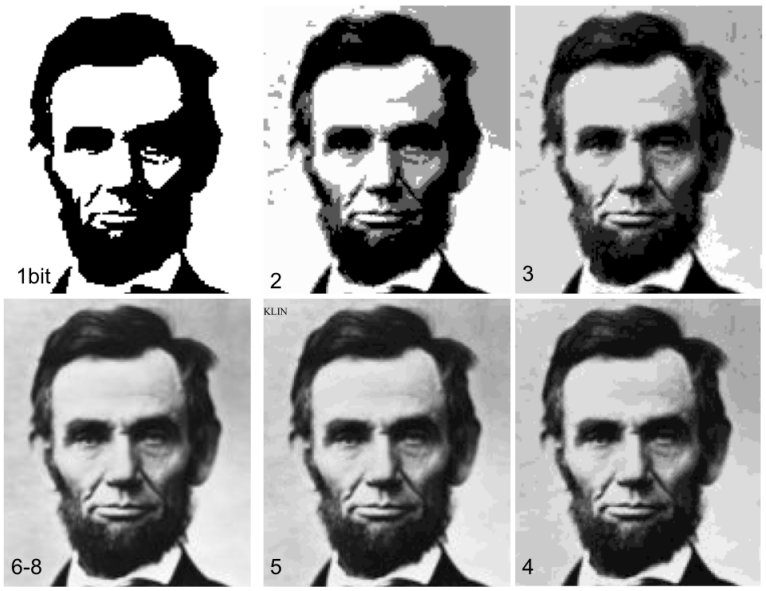

Figure 2 – Différents niveaux de gris pour différents niveaux de quantification

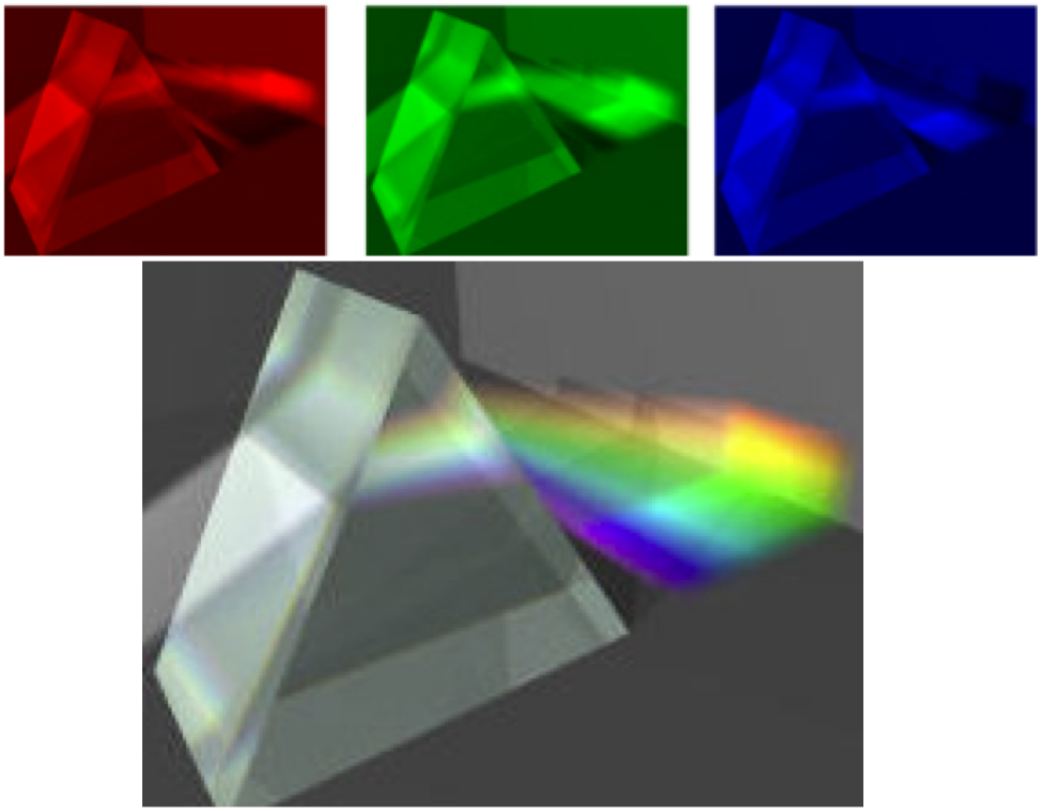

Figure 3 – Composantes {R,V,B} pour une image couleur

La quantité d'information contenue dans une image peut être relativement importante. Voici quelques ordres de grandeur (hors compression) :

couleur	3 composantes
quantification de l'intensité	8 bits par composante
pixels	plusieurs millions (10^6, Mb) par image
page d'écriture	$40 * 60$ caractères ($20 * 10^3$, Kb)
photo	Dizaines de millions de bits
film (1h30)	16,2 Tb ($16,2 * 10^{12}$ bits)

16,2 Tb = 8bits (intensité)*3(couleur)*5.10^6(taille de l'image)*25(images par seconde)*(3600+1800)(une heure et demi)

Vous allez au cours des exercices suivants aborder les aspects informatiques nécessaires à la navigation dans les pixels d'une image, et plus généralement la lecture et l'écriture d'une image avec les langages Java et C++.

2 Notion d'histogramme

Les histogrammes donnent une représentation graphique à une dimension (1D) de la distribution des pixels d'une image en fonction de leurs intensités. Ils permettent d'avoir une signature de l'image avec relativement peu d'information (256 valeurs entières pour une image en niveaux de gris codés sur 8bits).

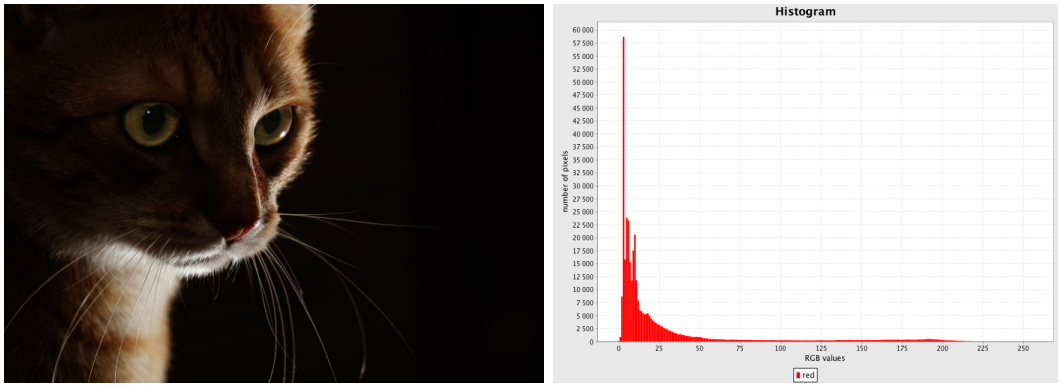

Figure 4 – Image couleur, et l'histogramme correspondant à sa couche rouge.

Sur cet histogramme, vous constaterez un décalage vers la gauche, qui s'explique par la quantité importante de pixels d'intensité faible ("foncés").

3 Manipulation avec le langage Java

3.1 Présentation

Nous allons nous appuyer sur deux classes incluses dans les bibliothèques standard de Java. Il s'agit des classes `javax.imageio.ImageIO` et `java.awt.image.BufferedImage`.

La classe `BufferedImage` est une classe de haut niveau issue de l'API Java2D. Elle offre des primitives d'accès aux données (pixels) de l'image en passant par l'objet `Raster` (retourné par la méthode `getData()`) . Il est ainsi possible d'accéder directement à un pixel particulier, ou bien à la totalité des pixels contenus. Une `BufferedImage` permet également d'accéder au modèle de couleur (classe `ColorModel`) en passant par sa méthode `getColorModel()` afin d'obtenir l'interprétation "couleur" des données de l'image.

La classe `ImageIO` est une classe statique fournissant un certain nombre de services relatifs aux lectures/écritures des images. Les formats supportés par défaut sont GIF, PNG, JPEG, BMP, et WBMP. Les principales méthodes sont une série de `read(..)` et `write(..)` ainsi que `getReaderFormatNames()` et `getWriterFormatNames()` pour obtenir des informations sur les formats pris en charge.

Vous trouverez plus d'information au travers des pointeurs suivants :
- Documentation `BufferedImage`
  http ://java.sun.com/javase/6/docs/api/java/awt/image/BufferedImage.html
- Documentation `ImageIO`
  http ://java.sun.com/javase/6/docs/api/javax/imageio/ImageIO.html
- Utilisation de la classe `BufferedImage`
  http ://www.particle.kth.se/ lindsey/JavaCourse (chapitre 11)
- Exemples d'utilisation de la classe `ImageIO`
  http ://www.exampledepot.com/egs/javax.imageio/pkg.html

3.1.1 Lecture d'une image

Le code ci-dessous réalise la lecture d'une image et l'affichage des composantes rouge, vert et bleu de ses pixels :

```
1  File fileSelected = new File("image_file_path");
2  System.out.println("file to be opened :" + fileSelected);
3  try {
4    BufferedImage bin = ImageIO.read(fileSelected);
5
6    // print each rgb values of pixels
7    int w = bin.getWidth();
8    int h = bin.getHeight();
9    for (int j = 0; j < h; j++) {
10     for (int i = 0; i < w; i++) {
11       Color c = new Color(bin.getRGB(i, j));
12       System.out.println("[R,G,B] = ["+c.getRed()+","+c.getGreen()+","+c.getBlue()
             +"]");
13     }
14   }
15 } catch (IOException e) {
16   System.out.println("Unable to open file " + fileSelected);
17   System.out.println(e.getStackTrace());
18 }
```

La lecture se fait effectivement en ligne 4. Attention, il est obligatoire de positionner cette instruction dans un bloc `try-catch`. En effet si la lecture échoue, une exception de type `IOException` est levée. Il faut donc pouvoir l'intercepter (lignes 15 à 18).

Le parcours est réalisé avec deux boucles `for` imbriquées (lignes 9 à 14). L'image est d'abord parcourue par lignes, puis par colonnes. On évalue alors pour chaque coordonnée `[ligne, colonne]` la valeur des composantes rouge, vert et bleu en utilisant un objet de type `Color`. A noter que l'image renvoie un entier qui contient un codage des trois composantes. Cet entier est utilisé dans le constructeur de l'objet `Color`.

3.1.2 Ecriture d'une image

le code suivant réalise l'écriture d'une image :

```
1 try {
2   BufferedImage bi = getMyImage(); // retrieve image
3   File outputfile = new File("saved.png");
4   ImageIO.write(bi, "png", outputfile);
5 } catch (IOException e)
6   System.out.println("Unable to write file")
7   System.out.println(e.getStackTrace());
8 }
```

De la même manière que pour la lecture d'une image, nous allons passer par la classe `ImageIO` et sa méthode `write()`. L'écriture est effectuée en ligne 4 et pour les même raisons que précédemment, est placée dans un bloc `try-catch`. Parmi ses paramètres, le format d'écriture est indiqué par une chaîne de caractère correspondant à un des formats supportés.

3.2 En pratique

La compilation du code se fait par l'outil *Maven*, similaire à *Ant*, *Make*, *Cmake*, ou encore *Qmake*. Le fichier `pom.xml` contient les directives de construction du projet (compilation, packaging, exécution). *Maven* a l'avantage d'avoir une syntaxe concise, et de gérer les dépendance. Il suffit alors de lancer la commande `mvn install` dans le répertoire ou se trouve le fichier `pom.xml` pour construire le projet. Les environnements de développement tels que *Netbeans* ou *Eclipse* gèrent nativement les projets *Maven*.

Exercice 1 *Ecrire une classe Java qui comporte une méthode* `load` *qui retourne, à partir d'un chemin vers un fichier passé en paramètre, un objet de type* `BufferedImage`.

Exercice 2 *Compléter cette classe avec une méthode* `dump` *qui parcourt l'ensemble des pixels d'une* `BufferedImage` *passée en paramètre et affiche sur la sortie standard, les valeurs des composantes rouge, vert et bleu, d'un pixel sur dix. Ecrire dans une méthode* `main` *le chargement de l'image ainsi que l'affichage des informations sur les pixels.*

Exercice 3 *Compléter cette classe avec une méthode* `save` *qui réalise, à partir d'une image, d'un format, et d'un chemin de fichier passés en paramètres, l'écriture de l'image. Compléter également la méthode main pour effectuer une conversion de format et finalement valider l'enregistrement d'une image.*

Exercice 4 *Compléter cette classe avec une méthode* `histogram` *qui imprime sur la sortie standard l'histogramme d'une des couches couleurs d'une image. Comparez les histogrammes obtenus avec des images claires et foncées.*

Vous pourrez vous aider des étapes suivantes :

1. pour une couche couleur, sommer les pixels de même intensité dans un tableau ;
2. normaliser ce tableau pour avoir des valeurs d'histogramme comprises entre 0 et max_{new} (10 par exemple) afin d'avoir une représentation textuelle plus compacte.

$$H_{norm}(i) = H(i) * \frac{max_{new}}{max_{old}}$$

3. parcourir l'histogramme pour l'imprimer sur la sortie standard (`System.out`), attention au sens de lecture de l'histogramme.

vous devez obtenir quelque chose comme ça :

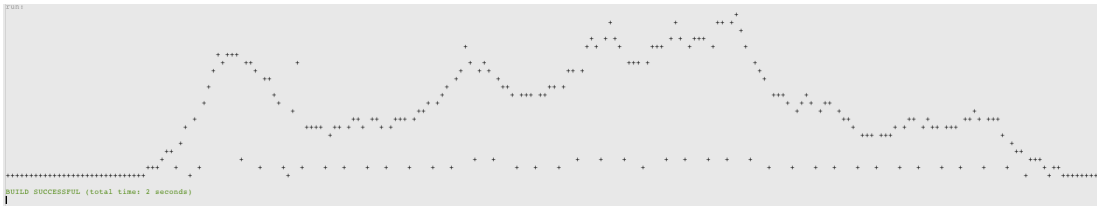

Figure 5 – Histogramme imprimé sous forme textuelle

4 C++ CImg

4.1 Présentation

CImg (http ://cimg.sourceforge.net/) est une bibliothèque C++ libre (licence CeCILL) dédiée au traitement d'images. Cette bibliothèque utilise la généricité de C++ (les *templates*) et est portable. Elle est de plus très simple d'utilisation : il suffit d'inclure le fichier "CImg.h" dans un code C++. Attention cependant, il est nécessaire d'avoir sur son système d'exploitation "ImageMagick" (http ://www.imagemagick.org) pour réaliser des conversions de format, et les fichiers headers de X11 (contenus par exemple dans les packages Ubuntu "libX11-dev" et "libxrandr-dev").

CImg est constitué de trois classes principales :
- CImg<T> : classe dédiée à la manipulation d'images de une à quatre dimensions, chaque scalaire étant de type générique T ;
- CImgList<T> : classe de manipulation de séquence d'images ;
- CImgDisplay : classe d'affichage d'images. Il est possible de capturer les interactions de l'utilisateur via la souris ou le clavier.

Cette bibliothèque fournit également des facilités pour itérer sur les pixels d'une image. Il devient donc inutile d'imbriquer des boucles `for` comme dans l'exemple en Java. On peut directement utiliser l'instruction suivante :

```
1 // Equivalent to for(y..){for(x..){..}}
2 cimg_forXY(image,x,y){ image(x,y) = .. ;}
```

La documentation de ces classes est disponible à cette adresse :
http ://cimg.sourceforge.net/reference/index.html

4.1.1 Lecture d'une image

Le code source suivant réalise le chargement d'une image codée sur un octet en passant par un constructeur (`CImg (const char *const filename)`).

```
1 CImg<unsigned char> image = CImg<>("path_to_input_file");
```

Il est possible d'instancier un objet `CImg`, et de charger dans un second temps une image. La méthode `load` est alors utilisée (`CImg< T > & load (const char *const filename)`).

```
1 CImg<unsigned char> image = CImg<>();
2 image.load(file_i);
```

Finalement, l'image est affichée via l'objet `CImgDisplay`. Afin de conserver l'image "à l'écran", on écrit une boucle d'attente :

```
CImgDisplay main_disp(image,"Input Image");

while (!main_disp.is_closed &&
   !main_disp.is_keyESC &&
   !main_disp.is_keyQ) {
    cimg::wait(20);
}
```

4.1.2 Ecriture d'une image

Le code source suivant enregistre l'image (`image`) au format BMP, à l'endroit spécifié en paramètres. D'autres formats sont disponibles et les méthodes sont décrites sur la documentation en ligne.

```
image.save_bmp("path_to_output_file");
```

Le code suivant est une méthode similaire, le format de l'image est déterminé par le suffixe du fichier. Attention, pour prendre en charge la majorité des formats d'image, la bibliothèque ImageMagick doit être installée.

```
image.save("path_to_output_file.jpg");
```

4.2 En pratique

Voici le corps d'un programme C++ minimaliste constitué d'une méthode `main` et de l'inclusion des fichiers d'en-tête `CImg` et `iostream` :

```
#include "CImg.h"
#include <iostream>
using namespace cimg_library;

int main(int argc,char **argv) {
  return 0;
}
```

La compilation se fait soit avec le `Makefile` fournit par la bibliothèque CImg, soit avec la ligne suivante (pour linux) :

```
g++ -o hello_word.exe hello_world.cpp -O2 -L/usr/X11R6/lib -lm -lpthread -lX11
```

Dans le cas ou vous utilisez le `Makefile` fournit, il vous faut l'éditer et spécifier la liste des fichiers sources à compiler.

Exercice 5 *Utiliser ce patron de programme pour charger une image et l'afficher. Vous vous appuierez sur le constructeur ou bien une des méthodes de chargement.*

Exercice 6 *Compléter votre programme en parcourant la totalité des pixels de l'images et en affichant la valeur de chacune des composantes rouge, vert et bleu. Utiliser l'opérateur* (x, y, k) *où* $k = \{0, 1, 2\}$ *;* 0 *pour la composante rouge,* 1 *pour le vert, et* 2 *pour le bleu.*

Exercice 7 *Finalement, sauvegarder l'image dans un format différent de celui de l'image d'entrée.*

Exercice 8 *En "bonus", afficher l'histogramme d'une image en utilisant la fonction CImg prédéfinie.*