Introduction au traitement numérique des images

Traitements radiométriques

L. Beaudoin

http://learning.esiea.fr/

INF4034, 2014 -2015

Plan

Plan: Introduction

Objectifs de cette présentation

- **Rappeler** les notions fondamentales des images numériques et présenter des outils pour les manipuler
- **Utiliser** des informations statistiques d'une image via la notion d'histogramme
- **Comprendre** l'intérêt de la notion de seuillage

Plan: Fondamentaux

Rappels

Une image numérique :

- un tableau 2D de pixels (abrévation de picture element) : c'est l'information spatiale ou géométrique
- chaque pixel donne l'intensité lumineuse reçue : c'est l'information radiométrique

Une vidéo est une succession d'images séparées par un intervalle de temps constant (typiquement 25 à 30 images par seconde)

Image en couleurs

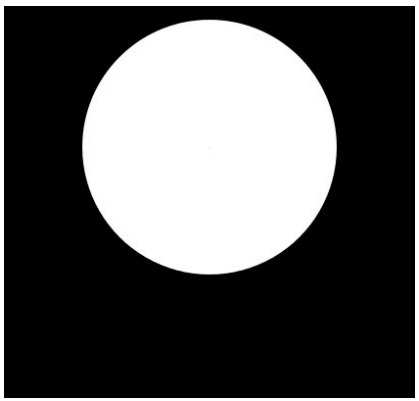
La construction d'une image en couleurs se fait au niveau du couple œil/cerveau qui intègre pour chaque cône d'analyse (pixel) les informations radiométriques provenant de la bande rouge, verte et bleue.

Un pixel d'une image en couleurs est donc composé de 3 valeurs : sa valeur dans le rouge, dans le vert et dans le bleu.

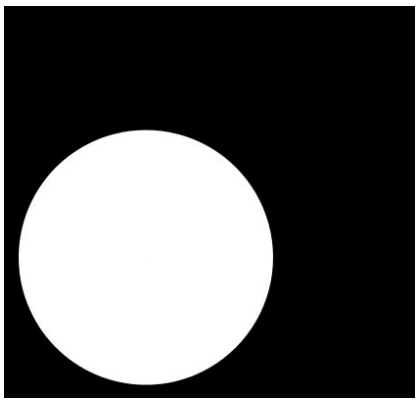
Une image en couleurs est donc composée de 3 images en niveaux de gris : une pour l'information dans le rouge, dans le vert et dans le bleu.

Image en couleurs

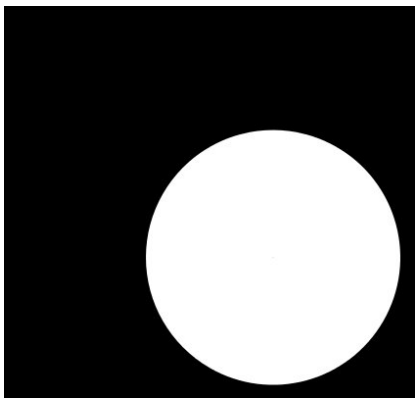
Exemple :



composante rouge



composante verte



composante bleue

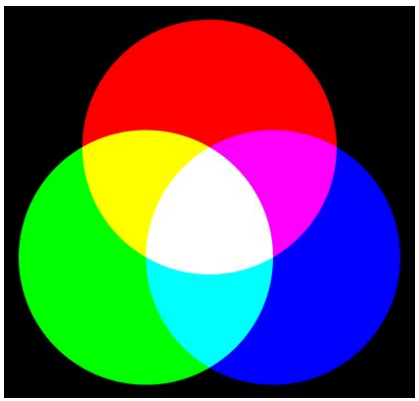


image en couleurs

Fichier image

Les images sont sauvegardées sur un support numérique dans des fichiers qui contiennent :

- la taille de l'image c'est-à-dire le nombre de lignes et de colonnes
- le nombre de plans qui la compose (1 pour une image en niveaux de gris, 3 pour une image en couleurs, 4 pour une image en couleurs avec transparence)
- la profondeur de ces plans (i.e. le nombre de bits par pixel pour décrire la radiométrie, généralement en `unsigned char` soit 8 bits). La profondeur de l'image est la somme des profondeurs des plans qui la compose (24 bits pour une image en couleurs par exemple)
- un "magic number" qui indique comment sont organisées les données (format bmp, raw, tiff, jpeg...)
- les données (pixels)
- éventuellement des données complémentaires (données Exchangeable image file format ou Exif) comme la date, l'heure, les réglages de l'appareil (focale, vitesse d'obturation, ouverture, sensibilité...), la position GPS....

Fichier image

Pourquoi n'existe t'il pas de format de fichier image universel ?

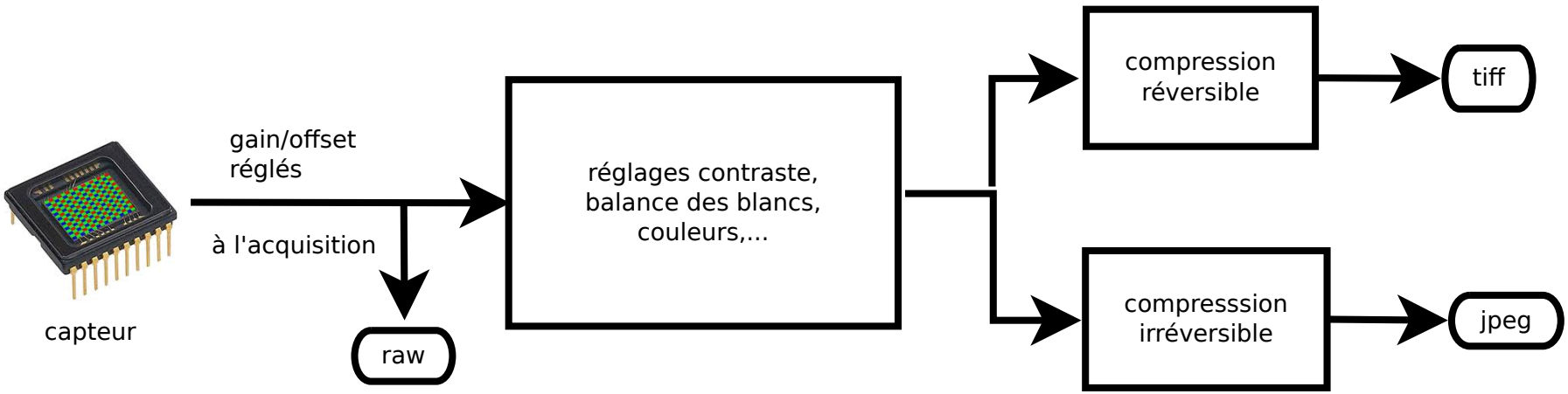
Réponse : parce que les besoins des utilisateurs ne sont pas tous identiques ! Chaque format a ses avantages et inconvénients, il n'y a donc pas de choix universellement optimal !

Parmi les différents formats, on peut distinguer ceux :

- qui ont vocation a être au plus près de la donnée acquise originale (`raw`, `bmp`...). Avantage : données brutes. Inconvénient : grande taille de fichier
- qui optimisent la taille des fichiers (`jpeg`, `tiff`...). Avantage : taille en mémoire optimisée (compressée) donc facilement transférable (web). Inconvénient : la compression entraine une perte de la qualité (finesse) souvent irréversible

Fichier image

En pratique, lors d'une acquisition sur un appareil :



Remarque : il n'y a pas de standardisation du format `raw` ; tous les logiciels ne pourront pas les lire !

Logiciels de traitement d'images

Pour manipuler les fichiers images et interagir avec les pixels, on peut :

1. utiliser des logiciels dédiés libres (gimp) ou non (photoshop)
2. utiliser des langages de haut niveau (matlab, mathématica, IDL)
3. utiliser des bibliothèques de langages de programmation (opencv, gtk...)

Pas de solution universelle :

- Si le traitement est exceptionnel et porte sur un nombre très limité d'images, 1 est adapté
- Si on prototype une chaine, 2 est préférable
- Si on fait un traitement routinier, comme en embarqué, 3 est indiqué

Remarque : attention à la contamination des licences logicielles !

Bibliothèque opencv

Dans le cadre de ce cours, vous serez amené à coder des programmes de traitement d'images (cas 3 du slide précédent). Pour cela, nous utiliserons la bibliothèque de traitement d'image `opencv`, version `C` et en environnement linux.

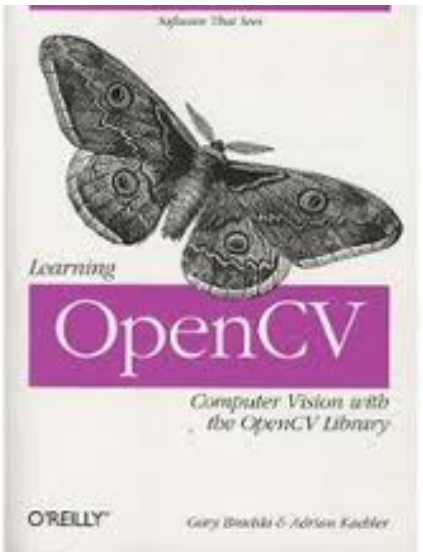
`opencv` est donc une bibliothèque dédiée au traitement d'images (CV pour Computer Vision). Cette bibliothèque est sous licence BSD.

Bibliothèque opencv

La documentation utile :

- `http://docs.opencv.org/`
  Le site officiel, qui héberge notamment la dernière version stable téléchargeable, le wiki officiel, la doc...
- `http://docs.opencv.org/modules/refman.html`
  Le lien du manuel de référence en ligne : à marque-pager tout de suite !

- L'ouvrage de référence (disponible au centre de doc) :
  "Learning OpenCV",
  G. Bradsky and A. Kaehler,
  O'Reilly



Bibliothèque opencv

- `opencv` est né en 1999 dans la société d'Intel pour développer des applications avancées en traitement d'image.
- Constat : chaque universités développaient leur propres bibliothèques à pérénité et efficacité variables et incompatibles entre elles.
- Une idée : créer une bibliothèque à vocation universelle complète, optimisée, interplateformes, ouverte sous licence BSD (applications payantes possibles).

Les objectifs d'`opencv` sont en résumé de :

- promouvoir le traitement d'image,
- aider au développement d'algorithmes de vision en utilisant du code ouvert, optimisé et supporté par une large communauté.

Bibliothèque opencv

Les principaux modules qui composent `opencv` sont :

- `cv` dédié au algorithmes de traitement d'image et de vision,
- `HighGUI` dédié à la réalisation des interfaces Homme-Machine et la gestion des flux entrée-sortie image et vidéo,
- `MLL` (Machine Learning Library) dédié aux applications d'apprentissage (réseaux neuronnaux).

Pourquoi plusieurs modules ? Pour optimiser la place prise par la bibliothèque en mémoire (exemple pas d'interface graphique en embarqué)

Remarque : possibilité d'augmenter les performances de base d'`opencv` en utilisant l'Intel Performance Primitive (IPP) en plus (payant) si utilisation de processeurs de la famille Intel.

Bibliothèque opencv

Les objets `opencv` sont souvent liés entre eux par des notions hiérarchiques.

- Par exemple, le type `IplImage`, qui correspond à une image hérite du type `CvMat` (pour matrice) qui lui-même hérite du type `CvArr` (pour tableau).
- En pratique, cela signifie qu'une fonction dont un paramètre est de type `CvArr` par exemple tolèrera aussi un paramètre de type `CvMat` ou `IplImage`.

Bibliothèque opencv

Quelques objets `opencv` utiles :

structure	contient	représente
`CvPoint`	`int x, y`	un point dans une image
`CvSize`	`int width, height`	la taille d'une image
`CvRect`	`int x, y, width, height`	un extrait rectangulaire d'une image
`CvScalar`	`double val[4]`	un tableau des valeurs colorimétriques d'un point

Bibliothèque opencv

L'objet `IplImage` :

```
typedef struct _IplImage
{    ...
    int  nChannels;
    int  depth;
    int  width;
    int  height;
    char *imageData;
    int  widthStep;
    ...
}
IplImage;
```

- `nChannels` : nombre de plans
- `depth` : profondeur en bits de chaque plan
- `width` : largeur en pixels,
- `height` : hauteur en pixels,
- `imageData` : pointeur sur le début du tableau des données de l'image. (ATTENTION : par défaut, les pixels sont en BGR et de type `char` !)
- `widthStep` : largeur optimisée réelle en octets en mémoire (peut être différente de `width * depth * nChannels`).

Bibliothèque opencv

Quelques fonctions utiles pour charger/sauver :

- `IplImage* cvLoadImage(const char* filename, CV_LOAD_IMAGE_COLOR)`
  charge le fichier image `filename` (`bmp`, `jpeg`...) dans une structure `IplImage*`. `CV_LOAD_IMAGE_COLOR` impose une profondeur de 24 bits dans le format BGR (Blue Green Red) (d'autres options possibles).
- `int cvSaveImage(const char* filename, const CvArr* image)`
  permet de sauver l'image `image` dans le fichier `filename` dont l'extension définira le format de sortie souhaité (`bmp`, `jpeg`...)
- `void cvReleaseImage(IplImage** image)`
  permet de désallouer la place mémoire occupée par l'`IplImage*` nommée `image`.

Bibliothèque opencv

Quelques fonctions utiles pour IHM :

- `int cvNamedWindow(const char* name,CV_WINDOW_AUTOSIZE)`
  permet de créer une fenêtre nommée `name`. `CV_WINDOW_AUTOSIZE` adapte la fenêtre sur la taille de l'image qui sera affichée
- `void cvDestroyWindow(const char* name)`
  désalloue la mémoire réservée par la fenêtre `name`.
- `void cvShowImage(const char* name, const CvArr* image)`
  afficher l'image `image` dans la fenêtre `name`
- `int cvWaitKey(int delay=0)`
  attend que l'on presse une touche du clavier pour continuer le programme (`delay=0` attente infini sinon attente en millisecondes)

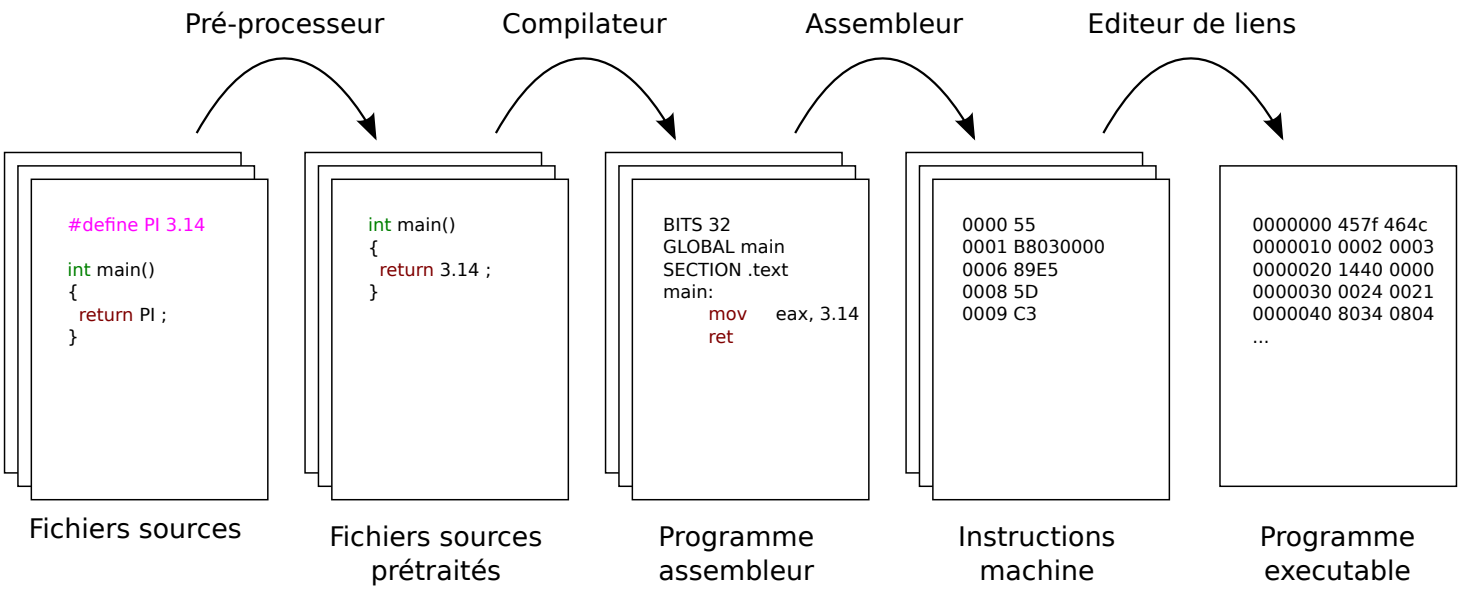
Bibliothèque opencv - code d'affichage d'une image

```
#include <stdio.h>
#include <opencv/highgui.h>

int main(int argc, char** argv)
{
   IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR);
   if (img==NULL){
      printf("Caramba, pas vu pas pris!\n");
      return(1);
      }
   //Partie traitement d'image ici
   cvNamedWindow("Exemple image", CV_WINDOW_AUTOSIZE);
   cvShowImage("Exemple image",img);
   cvWaitKey(0);
   cvReleaseImage(&img);
   cvDestroyWindow("Exemple image");
   return(0);
}
```

Bibliothèque opencv - compilation

Rappel des différentes étapes de la compilation d'un programme C :



Bibliothèque opencv - compilation

- Pour résoudre l'étape pré-processeur, il faut préciser l'endroit où se trouve les headers (`.h`)
- Pour résoudre l'étape édition de liens, il faut préciser le nom des bibliothèques (`.a`, `.so`) et leur localisation
- En pratique, on utilise la commande `pkg-config` encadrée par ` (alt-gr-7) pour préciser ces informations (option `--cflags` pour étape pré-processeur et `--libs` pour l'édition de liens).

Exemple :

```
gcc -Wall toto.c `pkg-config --cflags --libs opencv`
```

Bibliothèque opencv

Rappels

- On manipule un pointeur sur un `IplImage`, donc pour récupérer la valeur d'un champ, il faut utiliser ->

```
int numLine, numCol, nbLines, nbCols;
IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
nbLines = img->height;
nbCols = img->width;
```

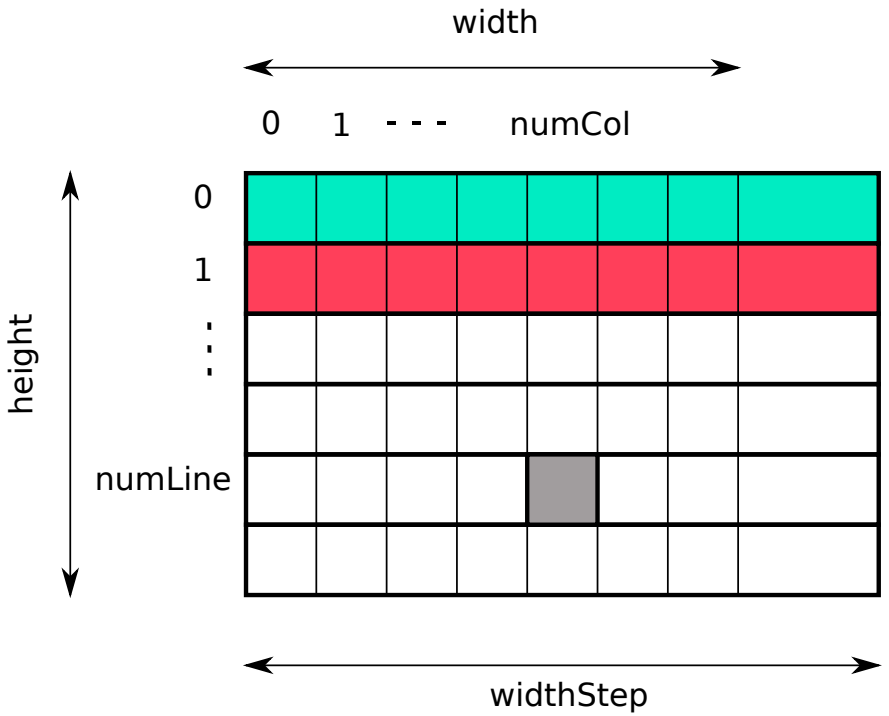
- Les données sont en **`char`** dans l'`IplImage`, il faut donc les retyper en **`unsigned char`** avant de les utiliser

```
valPix = (unsigned char) *(img->imageData + numLine * img->widthStep +
    numCol);
```

Bibliothèque opencv

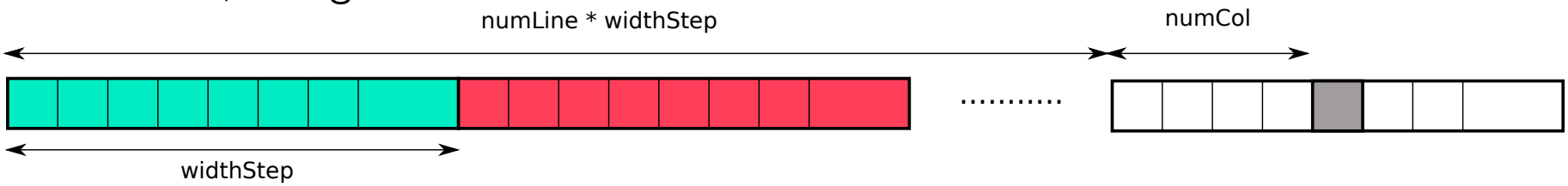
Une image s'organise ainsi :

- les indices lignes/colonnes commencent à 0 en haut à gauche
- les indices s'incrémentent vers la droite et le bas



Bibliothèque opencv

En mémoire, les lignes sont stockées les unes à la suite des autres :



Exemple de code pour récupérer la valeur du pixel d'indices (`numLine`, `numCol`) et l'affecter à la variable `valPix` de type **unsigned char** :

```
valPix = (unsigned char) *(img->imageData + numLine * img->widthStep + numCol);
```

Bibliothèque opencv

Structure d'un code pour parcourir tous les pixels d'une image

```
int numLine, numCol, nbLines, nbCols;
IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
nbLines = img->height;
nbCols = img->width;
for(numLine=0; numLine < nbLines; numLine++){
        for(numCol=0; numCol < nbCols; numCol++){
                /* traitement pixel a pixel */
        }
}
```

Bibliothèque opencv

Exemple de code : calcul des valeurs minimale et maximale d'une image :

```
int numLine, numCol, nbLines, nbCols;
IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
nbLines = img->height;
nbCols = img->width;
unsigned char valPix, min=255, max=0;
for(numLine=0; numLine < nbLines; numLine++){
    for(numCol=0; numCol < nbCols; numCol++){
          valPix = (unsigned char)
                *(img->imageData + numLine*img->widthStep+numCol);
          if(valPix < min)
                min = valPix;
          if(valPix > max)
                max = valPix;
        }
}
printf("min=%d max=%d\n",min, max);
cvReleaseImage(&img);
```

Bibliothèque opencv

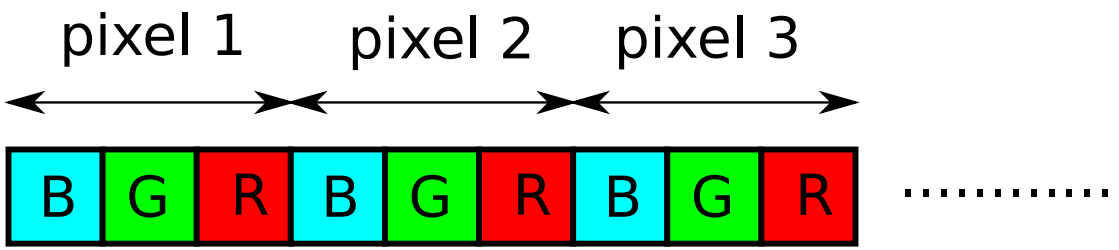
Exemple de code : calcul de la valeur moyenne d'une image :

```
int numLine, numCol, nbLines, nbCols;
IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
nbLines = img->height;
nbCols = img->width;
unsigned char valPix;
double somme = 0., mean;
for(numLine=0; numLine < nbLines; numLine++){
    for(numCol=0; numCol < nbCols; numCol++){
          valPix = (unsigned char)
                *(img->imageData + numLine*img->widthStep+numCol);
          somme = somme + valPix;
        }
}
mean = somme / (nbLines * nbCols);
printf("min=%lf\n",mean);
cvReleaseImage(&img);
```

Bibliothèque opencv

Cas des images en couleurs :

- tout ce que l'on vient de voir sur les images en niveaux de gris (organisation de l'image, parcours...) est identique pour une image en couleurs
- la seule différence est que maintenant chaque pixel n'est plus composé d'une valeur mais de 3 (dans l'ordre Bleu Vert Rouge (BGR) pour `opencv`)



Bibliothèque opencv

Exemple de code : calcul de la valeur moyenne d'une image couleurs :

```
IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR);
nbLines = img->height;
nbCols = img->width;
unsigned char valPixR, valPixG, valPixB, valPixGray;
double somme = 0., mean;
for(numLine=0; numLine < nbLines; numLine++){
    for(numCol=0; numCol < nbCols; numCol++){
       valPixB = (unsigned char)
             *(img->imageData + numLine*img->widthStep+3*numCol);
       valPixG = (unsigned char)
             *(img->imageData + numLine*img->widthStep+3*numCol)+1;
       valPixR = (unsigned char)
             *(img->imageData + numLine*img->widthStep+3*numCol)+2;
       valPixGray = (unsigned char)
             ((valPixB + valPixG + valPixR)/3.);
       somme = somme + valPixGray;
     }
}
mean = somme / (nbLines * nbCols);
```

Plan: Histogramme

Indicateurs statistiques

Problématique :
Comment synthétiser le volume d'informations présent dans une image ?

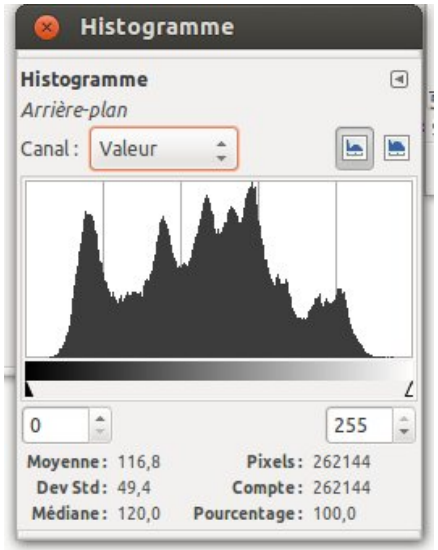
Réponse :
Utiliser des indicateurs statistiques :

- à une valeur : valeur minimale, maximale, moyenne...
- à un nombre limité de valeurs : tableau du nombre de répétitions de chaque valeur radiométrique dans l'image. C'est l'histogramme !

Histogramme

Un histogramme `histo` est donc un tableau 1D dont :

- l'indice `n` est une valeur radiométrique (donc comprise entre 0 et 255)
- `histo[n]` est le nombre de pixels de l'image qui ont la valeur radiométrique `n`



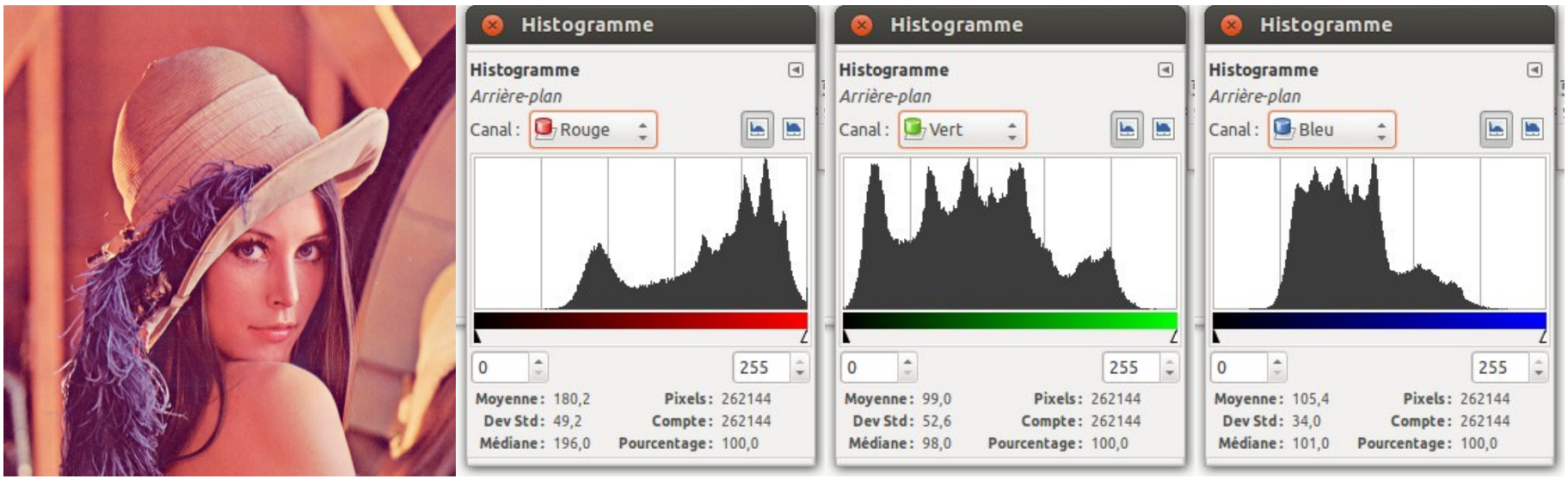
Histogramme

Exemple de code pour une image en niveaux de gris :

```
unsigned char valPix;
int numCase;
long int histo[256];
for(numCase=0; numCase < 256; numCase++) //initialisation histo
    histo[numCase] = 0;
for(numLine=0; numLine < nbLines; numLine++){ //parcours
    for(numCol=0; numCol < nbCols; numCol++){
          valPix = (unsigned char)
                *(img->imageData + numLine*img->widthStep+numCol);
          histo[valPix] = histo[valPix] + 1;
        }
}
```

Histogramme

Dans le cas d'une image en couleurs, on calcule l'histogramme pour chaque plan RVB (donc on a 3 histogramme et pas 1 seul) !



Histogramme

L'information contenue dans l'histogramme permet de retrouver :

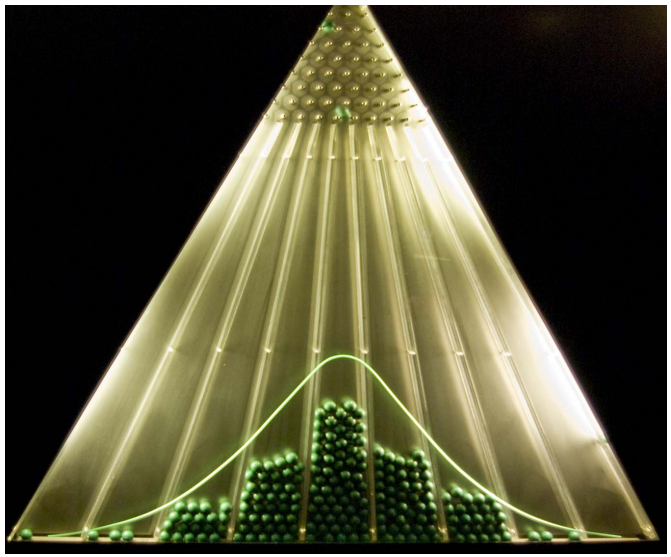
- la valeur minimale et maximale
- la valeur moyenne

```
double mean;
long int weightedSum=0;
int valRadio;
for(valRadio=0; valRadio < 256; valRadio++)
    weightedSum = weightedSum + valRadio * histo[valRadio];
mean = (double) weightedSum / (nbLines * nbCols);
printf("mean = %lf\n", mean);
```

Remarque : d'un point de vue complexité, mieux vaut calculer l'histogramme (1 seul parcours de l'image) puis extraire les valeurs minimale, maximale et moyenne que de faire 3 fois le parcours de l'image !

Petit rappel sur la statistique

La **statistique** est une science de l'aléatoire qui cherche à décrire mathématiquement des phénomènes caractérisés par le hasard et l'incertitude.

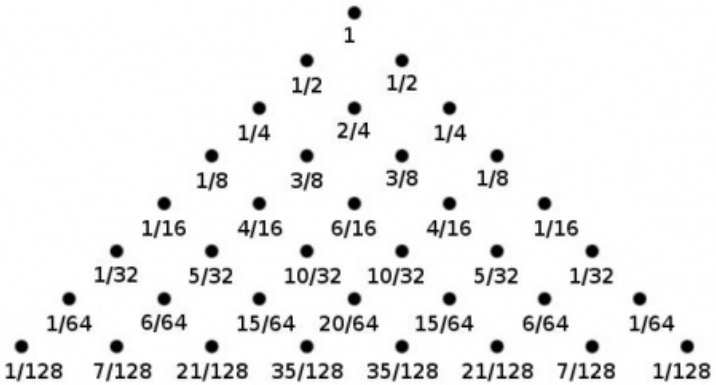


wikipédia

Expérience de la planche de Galton (1889)

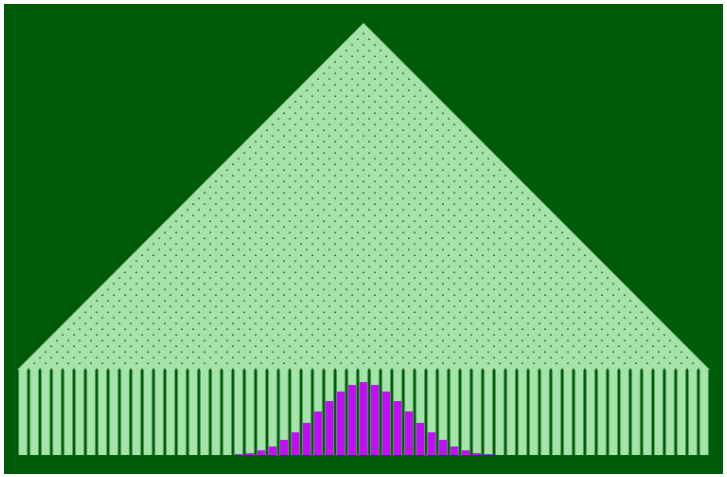
Petit rappel sur la statistique

De l'observation à la théorie : à chaque embranchement, il n'y a que 2 choix possibles.



wikidebrouillard.org

8 niveaux



lmrs.univ-rouen.fr

Plus de 50 niveaux

Remarque : plus le nombre de niveaux augmente, plus on a de cas en sortie (passage du cas discret au continu)

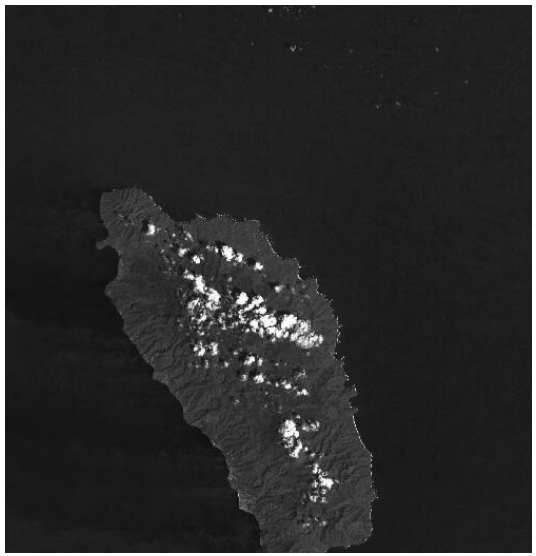
Petit rappel sur la statistique

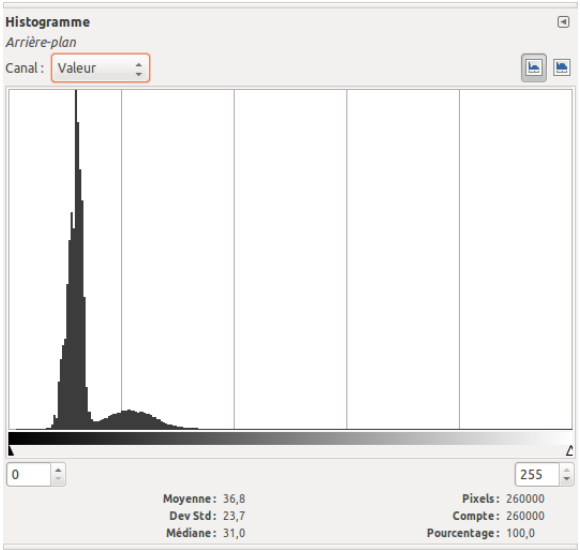
Une **loi de probabilité** est une loi mathématique qui décrit le comportement (pas de prévision !) aléatoire d'un phénomène lié au hasard.

La **densité de probabilité** est une fonction qui permet de représenter une loi de probabilité sous forme d'intégrale. Dans l'exemple de la planche de Galton, c'est la hauteur de chaque colonne. Dit autrement, c'est le nombre de fois où a eu lieu chaque cas en sortie (divisé par le nombre de cas pour que la somme totale soit égale à 1).

Histogramme et statistique

- Un histogramme est une densité de probabilité (au facteur de normalisation par le nombre de pixels de l'image près).
- Les formes ressemblant à des gaussiennes sont probablement liées au même phénomène physique
- Si ces formes se touchent, il deviendra impossible de séparer exactement dans la zone de recouvrement les phénomènes physiques

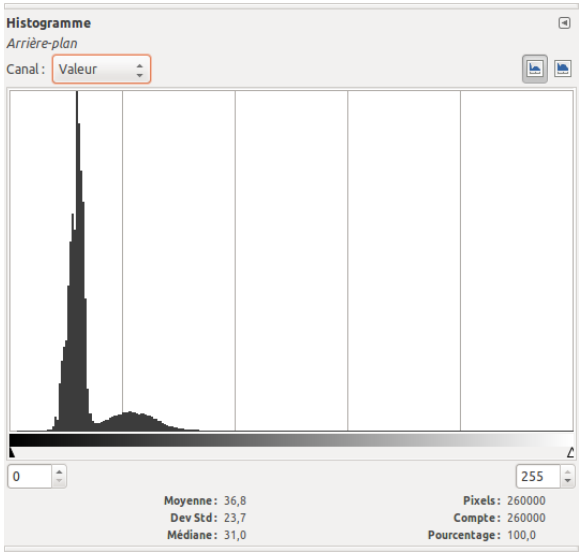




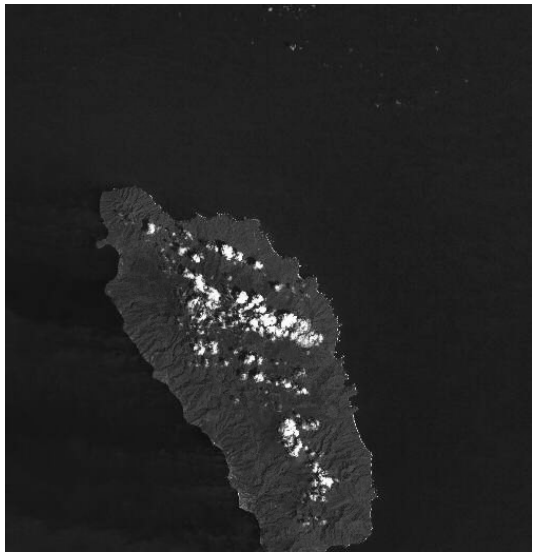
Seuillage binaire

- Un seuillage binaire permet d'obtenir une image composée uniquement de 2 valeurs (utile pour faire des masques des zones d'intérêt par exemple).
- Un seuillage binaire consiste à séparer les pixels de valeur radiométrique supérieure (ou inférieure) à un seuil `threhold` des autres.

```
if(valPix>threshold)
   valPix = 255;
else
   valPix = 0;
```



Histogramme



Original

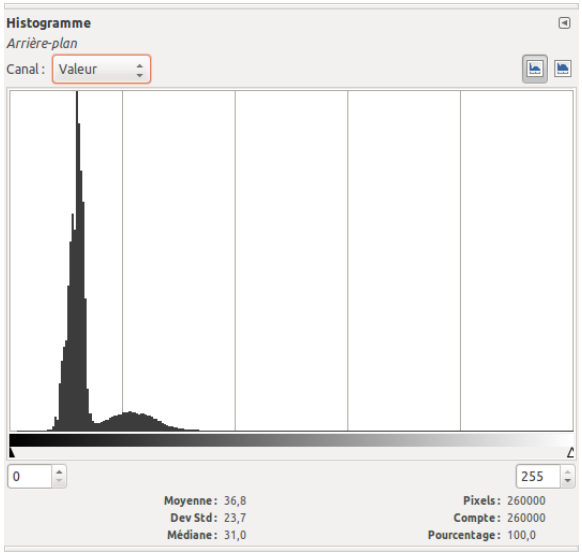
Seuil à 130

Seuillage binaire

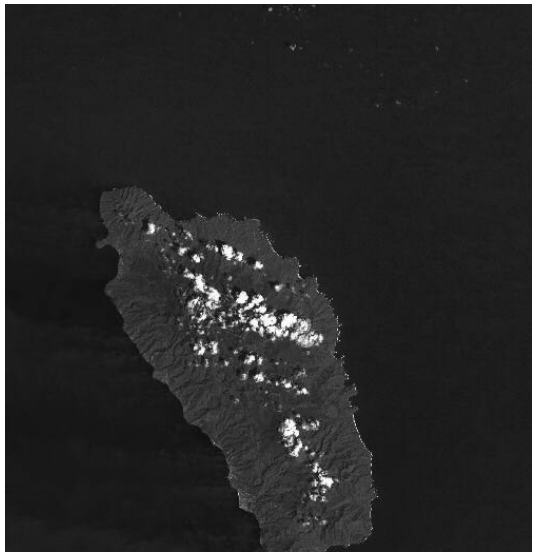
- On peut aussi séparer les valeurs radiométriques entre deux seuils par exemple pour isoler les gaussiennes présentes dans l'histogramme.

```
if(valPix<thresholdMin)
   valPix = 0;
else{
      if(valPix>thresholdMax)
         valPix = 0;
      else
         valPix = 255;
   }
```

- On pourrait aussi ne pas modifier la valeur du pixel lorsque celle-ci est entre les seuils (on a alors plus une image binaire)



Histogramme



Original

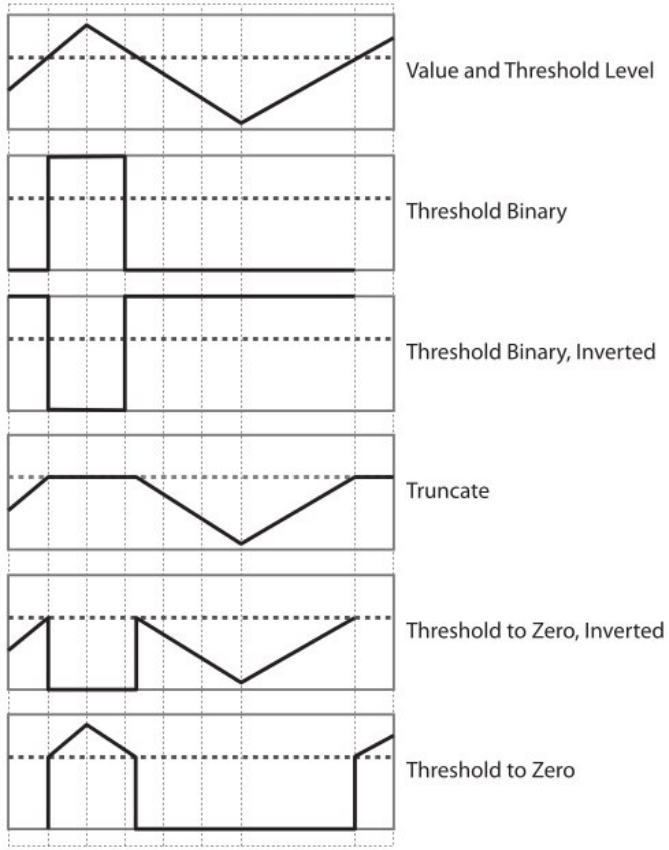
Seuil minimum à 38
Seuil maximum à 130

Seuillage en Opencv

- La fonction utile pour faire du seuillage est :

```
double cvThreshold(CvArr* src, CvArr* dst,
     double threshold, double max_value, int
      threshold_type );
```

- Les différentes possibilités de seuillage selon la valeur de `threshold_type`



Source : Bradsky, O'Reilly