Introduction au Traitement d'Images

Séance 2

Isabelle Debled-Rennesson
debled@loria.fr

Contenu

- Points abordés :
  - Codage des images
  - Histogrammes
    - Transformations linéaires
    - Transformations linéaires avec saturation
    - Transformations linéaires par morceaux
    - Transformations non-linéaires
    - Égalisation de l'histogramme
    - **Opérations sur les intensités de deux images : opérations logiques, addition et soustraction**

TP : Réalisation de plugins pour ImageJ

Opérations sur les images

Sources : Anne Vialard, Alain Boucher

Opérateurs logiques

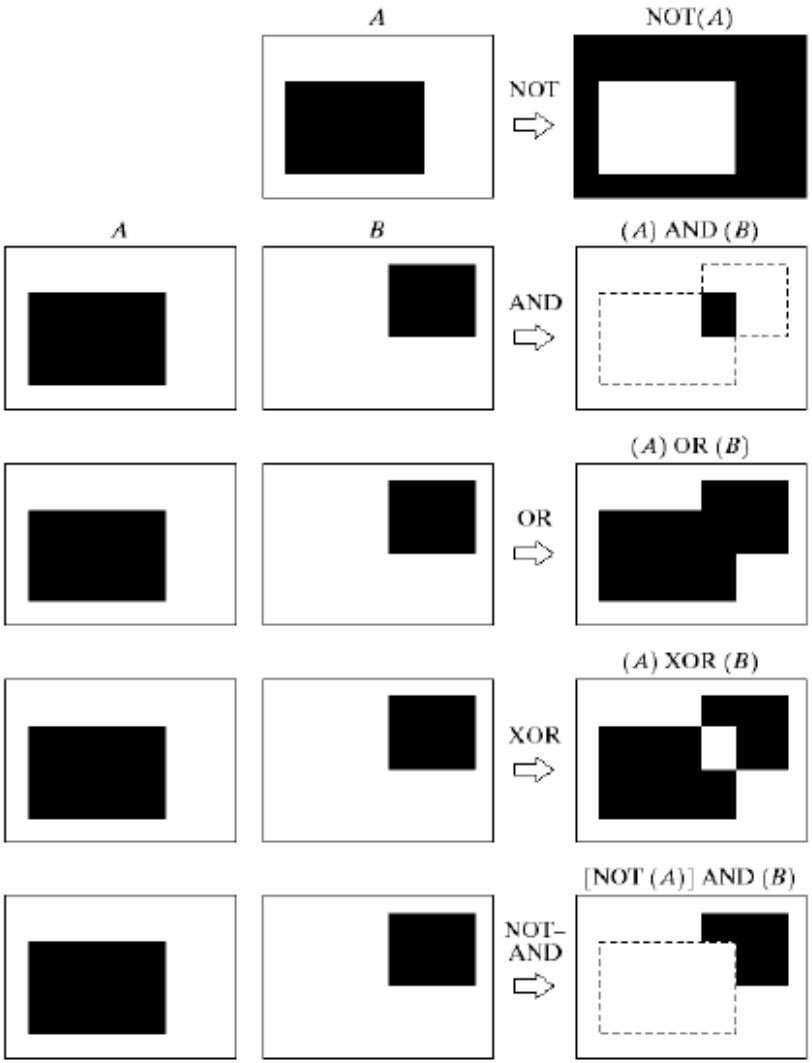


FIGURE 9.3 Some logic operations between binary images. Black represents binary 1s and white binary 0s in this example.

Source : Gonzalez and Woods. Digital Image Processing. Prentice-Hall, 2002.

Addition d'images

- L'addition pixel à pixel de deux images I et J est définie par :

$$A(x,y) = \text{Min}\ (\ I(x,y) + J(x,y)\ ,\ 255)$$

- L'addition de deux images peut permettre :
  - De diminuer le bruit d'une vue dans une série d'images
  - D'augmenter la luminance en additionnant une image avec elle-même

Source : Eric Favier. L'analyse et le traitement des images. ENISE.

Soustraction d'images

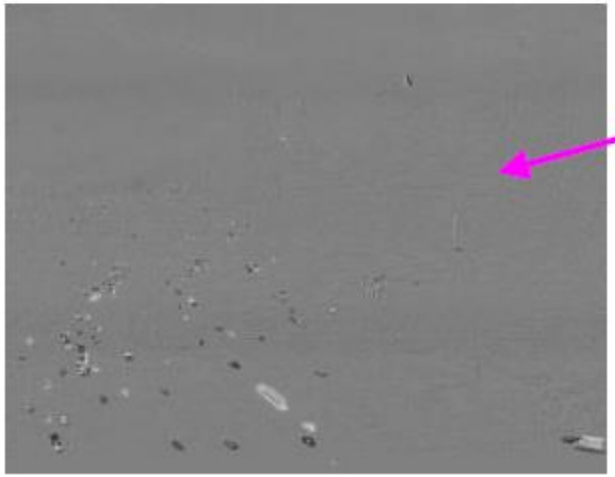
- La soustraction pixel à pixel de deux images I et J est définie par :

$$S(x,y) = Max (I(x,y) - J(x,y) , 0)$$

- La soustraction d'images peut permettre
  - La détection de défauts
  - La détection de mouvements

Soustraction d'images

Images prises à T et T + Δt



Résultat de la soustraction

Détection des changements

Multiplication d'images

- La multiplication d'une image I par un nombre est définie par :

$$M(x,y) = \text{Min}\ (\ I(x,y) * nb\ ,\ 255)$$

- La multiplication d'images peut permettre d'améliorer le contraste et la luminosité

Multiplication d'images

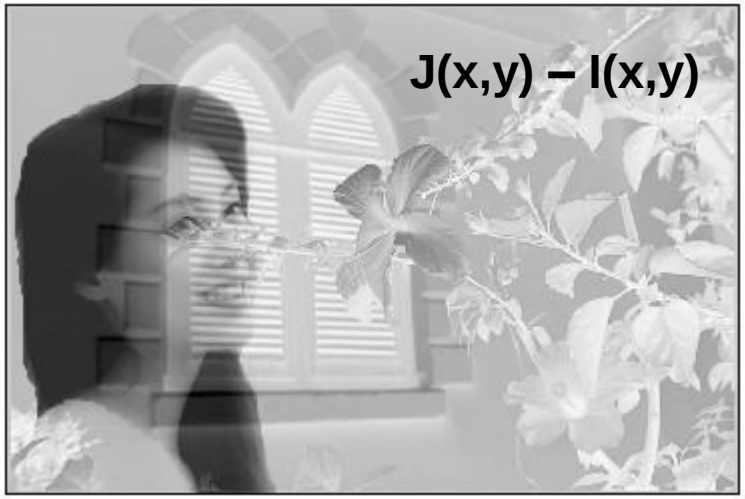
Source : Eric Favier. L'analyse et le traitement des images. ENISE.

Des exemples

I(x,y)

J(x,y)

0,5*I(x,y)+0,5*J(x,y)

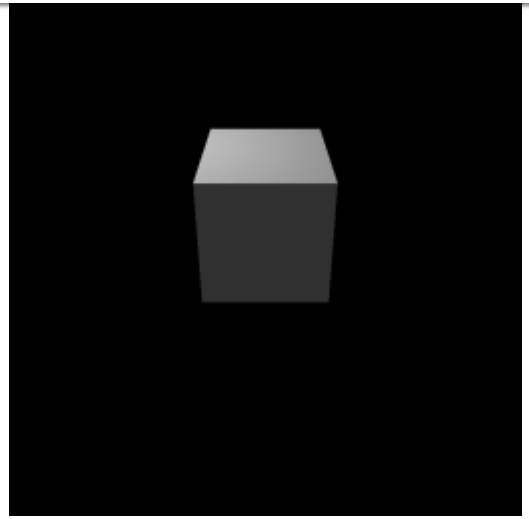


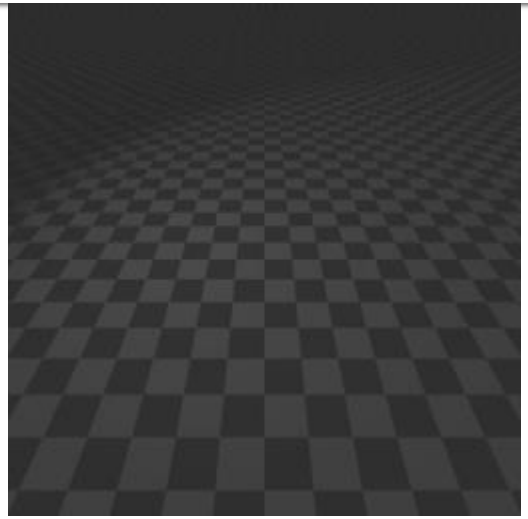
J(x,y) – I(x,y)

I(x,y) – J(x,y) ?

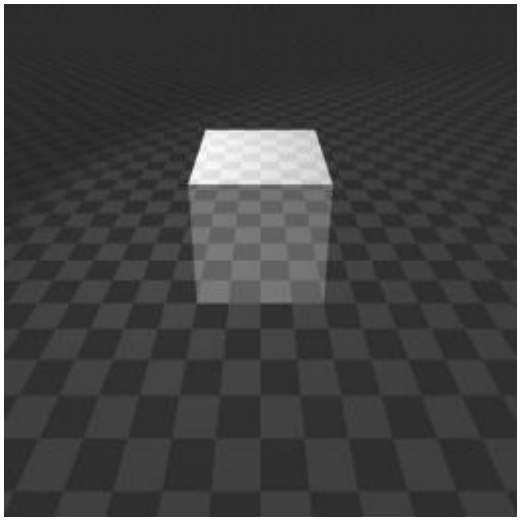
Des exemples

I(x,y)

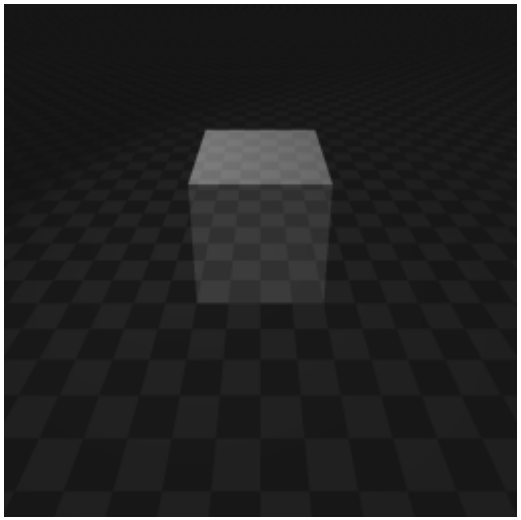




J(x,y)



I(x,y)+ J(x,y)



0,5*I(x,y)+0,5*J(x,y)

ImageJ

Sources pour cette partie : Olivier Losson

Généralités

- Développé par le *National Institutes of Health* : http://rsb.info.nih.gov/ij/
- Logiciel libre, écrit en Java, dédié au traitement d'images
  - conseillé : JRE > 1.6
  - multi-plateformes, multi-threaded, applet ou application autonome
  - sources disponibles et architecture ouverte (extensibilité par *plugins* en Java)
- communauté très active, surtout en imagerie biomédicale
- **Fonctionnalités en analyse d'images**
  - Lit/écrit de nombreux formats d'images et vidéos
  - manipulation des piles d'images (« images 3D »)
  - Possibilités étendues d'analyse des images en standard
    - outils d'analyse : histogrammes, profils, ...
    - opérations ponctuelles et de voisinage, morphologiques, FFT, segmentation, ...

Plugins

- **Définition**
  - Module écrit en Java (classe) permettant d'étendre les fonctionnalités d'ImageJ.
  - Utilise les classes de l'API ImageJ (http://rsbweb.nih.gov/ij/developer/api/).
  - Des plugins sont pré-installés dans <ij>/plugins.
  - Tout plugin doit contenir un **underscore** (_) dans son nom.

- **Développement**
  - Écrire un fichier .java contenant un underscore dans son nom ;
  - une classe de même nom, **implémentant l'interface PlugIn** ou **PlugInFilter**.
  - Compiler ce fichier (***Plugins/Compile and Run...***) pour générer le fichier .class.

Classes principales de l'API -1-

- **ij.ImagePlus**
  - Représente une fenêtre contenant une image et permet d'interagir avec elle.
  - Ses éléments sont accessibles par les méthodes d'instance, par exemple :
    - ImageWindow **getWindow()** : la fenêtre elle-même
    - ImageProcessor **getProcessor()** : le processeur traitant les données de l'image
    - Roi **getRoi()** : la région d'intérêt courante (zone sélectionnée)
    - ImageCanvas **getCanvas()** : le « canevas » utilisé pour représenter l'image dans la fenêtre (rectangle, facteur de zoom, ...) et en traiter les événements
    - ColorModel **getColorModel()** : le modèle couleur (ou la LUT) de représentation de l'image
    - FileInfo **getFileInfo()** : le fichier image

Classes principales de l'API -2-

- **ij.process.ImageProcessor**
  - Classe abstraite permettant de traiter ou convertir une image.
  - Ses sous-classes sont adaptées aux données contenues : ByteProcessor (et BinaryProcessor), ShortProcessor, FloatProcessor, ColorProcessor.
  - *Rem.*: l'image traitée n'est pas forcément affichée à l'écran.
- **ij.ImageStack**
- **ij.gui.Roi**

Deux types de plugins

- **Deux interfaces définissant deux types de plugins**
  - Un plugin implémentant l'interface **PlugIn** ne nécessite pas d'image en entrée.
  - Un plugin implémentant l'interface **PlugInFilter** nécessite une image en entrée.

Deux types de plugins

- **Deux interfaces définissant deux types de plugins**
  - Un plugin implémentant l'interface **PlugIn** ne nécessite pas d'image en entrée.
    - 1 seule méthode appelée, qui implémente ce que réalise effectivement le plugin :

      void run(java.lang.String arg)

Deux types de plugins

- **Deux interfaces définissant deux types de plugins**
  - Un plugin implémentant l'interface **PlugInFilter** nécessite une image en entrée.
    - la première méthode appelée initialise le plugin :

      int setup(java.lang.String arg, ImagePlus imp)
      - imp est l'image (*i.e.* la fenêtre contenant l'image) sur laquelle travaille le plugin
      - retourne ce que le plugin attend en entrée (type(s) d'image, région d'intérêt, ...)

    - la seconde méthode implémente ce que réalise effectivement le plugin :

      void run(ImageProcessor ip)
      - ip est l'image (*i.e.* le processeur accédant aux pixels) sur laquelle travaille le plugin
      - la ***fenêtre*** contenant l'image n'est accessible dans run() que si elle a été préalablement stockée dans une variable d'instance à l'exécution de setup()

Détails sur PluginFilter

La méthode setup() retourne une combinaison des constantes (int)

- Types d'images : le plugin traite ...
  - DOES_8G : des images en niveaux de gris sur 8 bits (entiers positifs)
  - DOES_16 : des images en niveaux de gris sur 16 bits (entiers positifs)
  - DOES_32 : des images en niveaux de gris sur 32 bits (flottants signés)
  - DOES_RGB : des images couleur RGB
  - DOES_8C : des images couleur indexées (256 couleurs)
  - DOES_ALL : des images de tous les types précédents
  - DOES_STACK : toutes les images d'une pile (applique run() sur chaque image)
- Type d'action : le plugin ...
  - DONE : effectue seulement son initialisation setup(), sans appeler la méthode run()
  - NO_CHANGES : n'effectue aucune modification sur les valeurs des pixels
  - NO_UNDO : ne nécessite pas que son action puisse être annulée

Classes principales de l'API -2-

La méthode setup() retourne une combinaison des constantes (int)

- Paramètres d'entrées supplémentaires : le plugin ...
  - STACK_REQUIRED : exige en entrée une pile d'images
  - ROI_REQUIRED : exige qu'une région d'intérêt (*RoI*) soit sélectionnée
  - SUPPORTS_MASKING : demande à ImageJ de rétablir, pour les *RoI* non rectangulaires, la partie de l'image comprise dans la boîte englobante mais à l'extérieur de la *RoI*

Exemple 1 : Image_Inverter.java

- Tester ce plugin dans ImageJ
  - Lancer ImageJ (→ création du répertoire .ImageJ)
  - Copier le fichier depuis arche dans le répertoire .imagej/plugins/ de votre home directory
  - Ouvrir une image dans imageJ (par exemple bridge avec open/samples)
  - Faire Plugins / Compile and Run
- Ouvrir le fichier et consulter le code

Accès aux pixels -1-

- **Accès ponctuel à un pixel dans un ImageProcessor**
  - Avec vérification des coordonnées : **int getPixel(int *x*, int *y*)**
    - Si *x* ou *y* hors limites, retourne 0 (et putPixel(*x*,*y*) n'a pas d'effet).
  - Sans vérification des coordonnées (plus rapide) : **int get(int *x*, int *y*)**
  - Accès par coordonnée unique : **int get(int *pix_index*)**
    - utile si les coordonnées ne sont pas utilisées, *ex*. pour des opérations ponctuelles
    - 0 ≤ *pix_index* ≤ **getPixelCount()**
  - Accès en écriture :
    - void **putPixel(int *x*, int *y*, int *value*)**
    - void **set(int *pix_index*, int *value*)**

Acces aux pixels -2-

- **Accès global aux pixels d'un ImageProcessor**
  - Accès aux pixels dans un tableau 1D : **Object getPixels()**
    - Nécessite une conversion, car le type du tableau dépend du type de processeur :

```
int[] pixels = (int[]) anImageProcessor.getPixels();
```

    - Le tableau retourné est **mono dimensionnel** :
      - Plus performant qu'un accès ponctuel (*ex.* getPixel()) à chacun des pixels.
  - Accès aux pixels dans un tableau 2D
    - Pour un {Byte|Short|Color}Processor : **int[][] getIntArray()** // coords : [x][y]

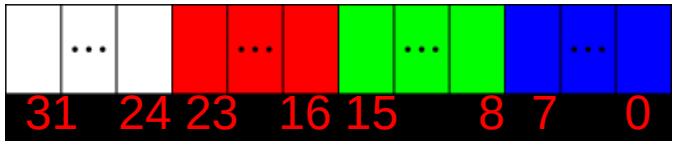
Acces aux pixels -3-

- **Élimination du bit de signe**
  - **Problème**
    - Les méthodes d'accès (ponctuel ou global) aux pixels retournent des valeurs entières **signées** (byte, short, int) ; *ex*. byte Î [-128..+127].
    - Or le plus souvent, on souhaite des valeurs **positives** de luminance ([0..255], etc.).
  - **Solution** : conversion en entier (int) **non signé** (positif) par **masquage**.
    - Exemple pour une image ip en niveaux de gris sur 8 bits (8G)

```
byte[] pixels=(byte[]) ip.getPixels();
...
int grey=pixels[i] & oxFF; // pixels[i] & oxFFFF pour short[]
```

    - Exemple pour une image ip couleur sur 32 bits (RGB)

```
int[] pixels=(int[]) ip.getPixels();
...
int r = (pixels[i] & oxFFoooo)>>16; // rouge
int g = (pixels[i] & oxooFFoo)>>8;    // vert
int b = (pixels[i] & oxooooFF);       // bleu
```



Exemple 2 : Color_Counter.java

- Tester ce plugin après ouverture d'une image
- Examiner le code

Documentation ImageJ

- Documentation des classes ImageJ
  - http://rsbweb.nih.gov/ij/developer/api/
- Tutoriel sur les plugins ImageJ
  - http ://www.imagingbook.com/fileadmin/goodies/ijtutorial/tutorial171.pdf