

Traitement d'image en Java

A. Gademer

M2SIG 2009-2010

1 Objectifs du TP

Dans ce TP, nous allons coder un certain nombre d'algorithmes classiques de traitement d'image. Le but n'est pas de créer un logiciel professionnel mais de comprendre ce qui se passe à l'intérieur afin de faire faire au logiciel ce que NOUS voulons.

Comme les années précédentes, nous avons fait le choix de ne pas mettre l'accent sur la conception ergonomique de l'IHM (bien que ce sujet soit très important) mais sur l'implémentation correcte des algorithmes.

Nous programmerons dans le langage Java ; vous trouverez de l'aide et des explications sur le fonctionnement de toutes les classes utilisées dans l'API Java 1.5.

<http://java.sun.com/j2se/1.5.0/docs/api/>

2 Le projet

Vous trouverez avec ce document une archive .zip contenant un projet **Netbeans** qui contient le squelette du TP.

1. Téléchargez l'archive
2. Décompressez-la
3. Lancer **Netbeans**
4. Choisissez *File* puis *Open Project...*

Dans le répertoire `src` vous trouverez le package `gui` qui contient quatre fichiers :

- `Application.java` contient la classe principale. C'est elle que l'on exécute (*Run as...*) et c'est celle que vous serez amenée à modifier.
- `ExampleFileFilter.java` sert à afficher les menus de choix de fichier.
- `Picture.java` étend `JPanel`. C'est le composant "Image" de notre IHM.
- `PictureFrame.java` permet d'afficher les images quelle que soit leur taille grâce à des barres défilement.

Les trois dernières classes sont là pour l'IHM et vous n'aurez pas à y toucher. Cependant, si vous êtes un peu curieux, ces classes pourront vous resservir pour développer de nombreuses interfaces.

3 L'IHM

3.1 Lancement

Si vous lancez `Application.java` (*Click Droit : Run as...*) vous devriez voir la Fig 1. L'interface se compose de deux parties. La partie du haut présente trois champs permettant de sélectionner des images ainsi qu'une série de boutons déclenchant des opérations. La partie du bas, vide pour l'instant permet d'afficher le résultat de l'opération. (Fig 2).



FIG. 1 – L’IHM au lancement de l’application.

3.2 Structure de la classe

Comment la classe `Application` permet-elle tout cela ? La classe possède une structure classique pour les programmes avec une interface basée sur `Swing`, l’API d’IHM de Java.

- Elle possède un champ privé nommé `mainPane` de type `JPanel`, le conteneur de base de l’API `Swing` ; `mainPane` **va donc contenir l’ensemble des éléments que l’on désire afficher à l’écran.**
- Elle possède un constructeur par défaut (`Application()`) qui remplit et initialise `mainPane`.
- Elle possède une méthode statique `createAndShowGUI()` qui crée un objet de type `JFrame` (objet fenêtre), crée une nouvelle instance d’`Application` qu’elle associe à la fenêtre puis affiche cette dernière.
- Enfin elle possède la méthode `main(String[])` qui fait de notre classe un exécutable et qui enrobe la méthode `createAndShowGUI()` dans un `thread`.

Cette architecture logicielle est celle que vous trouverez dans la majorité des exemples fournis par Sun. Elle a l’avantage de séparer proprement la définition de l’instanciation et de l’exécution.

3.3 Assemblage des composants

Revenons rapidement à l’initialisation de `mainPane`. Allez à la définition du constructeur `Application()`. Celle-ci peut paraître longue et illisible au départ mais vous verrez qu’elle n’est pas si complexe.

Petit rappel de base : tous les objets visibles dans la fenêtre du programme se nomment des Composants. Nous utilisons ici des étiquettes (`JLabel`), des champs de saisie (`JTextField`), des

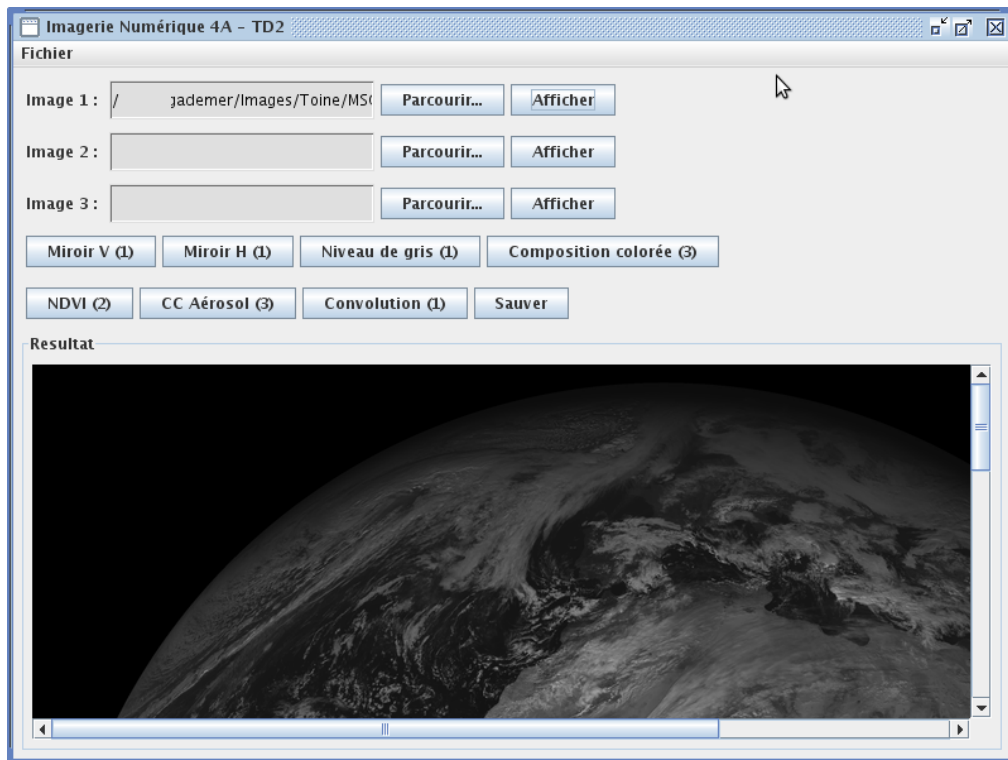


FIG. 2 – L'IHM après l'exécution d'une opération.

boutons (JButton) et un cadre pour notre image résultat (PictureFrame). Ces composants sont organisés dans des boîtes (JPanel) selon différentes organisations (Layout).

Nous utilisons les JTextBoxs pour écrire le chemin vers l'image sélectionnée. Comme c'est un effet visuel, on a désactivé l'édition de ces champs.

```
txtIMG1.setEditable(false);
```

Les JButton sont les sources d'action. On leur associe donc un ActionListener qui n'est autre que la classe Application elle-même

```
buttonIMG1.addActionListener(this);
```

et des ActionCommand qui sont des chaînes de caractères (Ici plusieurs mots clefs séparés par des '_') permettant d'identifier quel est le bouton qui a provoqué l'action.

```
buttonIMG1.setActionCommand("Process_MiroirV");
```

Lorsqu'un JButton est cliqué, la méthode actionPerformed(ActionEvent e) est exécutée.

Dans cette méthode nous découpons la chaîne ActionCommand et nous provoquons la réaction attendue.

En général, c'est le calcul de l'image resultat par l'appel aux méthodes privées de traitement d'image (calculateMiroirV() par exemple),

```
result = calculateMiroirV();
```

puis la mise à jour du cadre d'image,

```
imageResult.getPicture().setImage(result);
```

puis l'actualisation du cadre et

```
imageResult.updateUI();
```

enfin l'affichage du cadre.

```
imageResult.setVisible(true);
```

Toute l'IHM étant en place, il ne reste plus qu'à implémenter les algorithmes !

4 Pour commencer

4.1 Ouvrir une image

Nous allons commencer par observer la fonction `openImage(File fichier)` :

```
BufferedImage openImage(File fichier) {  
    // Vérification du champ IMG1  
  
    if (fichier == null) {  
        JOptionPane.showMessageDialog(null,  
            "Vous devez spécifier un fichier pour l'image.",  
            "Fichier Manquant", JOptionPane.ERROR_MESSAGE);  
    } else {  
  
        // Ouverture du fichier  
  
        try {  
            return ImageIO.read(fichier);  
        } catch (IOException e) {  
            JOptionPane.showMessageDialog(null,  
                "Impossible d'ouvrir le fichier : "  
                + fichierIMG1.getName(), "Erreur de lecture",  
                JOptionPane.ERROR_MESSAGE);  
        }  
    }  
    return null;  
}
```

Que se passe-t-il dans cette fonction ?

Tout d'abord les précautions d'usage : on vérifie que l'utilisateur a bien spécifié un fichier (fichier est non null) avant de cliquer sur ouvrir. Dans le cas contraire on ouvre un pop-up d'erreur prévenant l'utilisateur grâce à la méthode statique `JOptionPane.showMessageDialog(...)`.

Ensuite on ouvre l'image avec la fonction `ImageIO.read(File)` et on retourne le resultat sous la forme d'une `BufferedImage`.

Comme la fonction `ImageIO.read(File)` peut retourner des erreurs de type `IOException` en cas d'absence ou d'incohérence du fichier, il nous faut gérer cette levée d'erreur avec `try { .. } catch(IOException e) { ... }`. Ce qui est dans le bloc `try` est protégé et en cas d'erreur le bloc `catch` est exécuté puis le programme sort directement du `try` et finit son execution (ici il retourne null).

Dans un programme destiné aux utilisateurs, il est essentiel de prévoir ce qui peut avoir été fait de travers. Il est néanmoins dangereux de faire des suppositions sur ce que l'utilisateur à voulu faire. C'est pourquoi toute manipulation erronée doit être sanctionnée par un message d'erreur et l'annulation du traitement.

4.2 Premier algorithme

Pour commencer doucement, la première méthode `BufferedImage calculateMiroirV()` est déjà quasiment implementée.

Nous voulons appliquer un effet de miroir Vertical à l'image 1. C'est à dire inverser sa gauche et sa droite (comme autour d'un axe... vertical).

Si on décrit l'algorithme en pseudo-code :

```
Ouvrir l'image 1
Créer une image résultat de même taille que l'image 1.
Pour chaque colonne
| Pour chaque ligne
| | Prendre le pixel de l'image 1 et l'envoyer
| | de l'autre côté dans l'image resultat.
```

Nous verrons cette histoire d'autre côté plus tard.

Voyons le code que vous avez pour le moment :

```
BufferedImage calculateMiroirV() {
    BufferedImage calculateMiroirV() {
        BufferedImage imgIMG1 = null, result = null;

        // Ouverture du fichier

        imgIMG1 = openImage(fichierIMG1);

        if(imgIMG1 != null) {

            // Initialisation de l'image resultat

            result = new BufferedImage(imgIMG1.getWidth(), imgIMG1.getHeight(),
                BufferedImage.TYPE_INT_RGB);

            // Parcours du tableau des pixels

            for (int col = 0; col < imgIMG1.getWidth(); col++) {
                for (int row = 0; row < imgIMG1.getHeight(); row++) {

                    // Recopie de l'image 1 dans l'image resultat

                    result.setRGB(col, row, imgIMG1.getRGB(col, row));
                }
            }

            return result;
        }
    }
}
```

Tout y est non ?

Regardons de plus près ce qui se passe dans la double boucle :

```
result.setRGB(col, row, imgIMG1.getRGB(col, row));
```

on fait appel ici au deux méthodes essentielles de la classe `BufferedImage`.

Cette classe permet de gérer n'importe quelle image quel que soit son type (enfin presque). Elle nous fournit une méthode pour charger l'image en mémoire (`ImageIO.read(...)`) et deux méthodes pour accéder aux valeurs des pixels.

Quel que soit l'espace colorimétrique des images, `getRGB(...)` et `setRGB(...)` fonctionnent avec le type `IntRGB`. Ce type code l'information couleurs sur des entiers de 32 bits comme indiqué sur le schéma ci-dessous :

```
ARGB => 255,      163,      240,      0
hex =>  F  F  A  3  F  0  0  0
bin => 1111 1111 0110 0011 1111 0000 0000 0000
int  => 4284739584
```

Il faut donc faire attention à ce que retourne la fonction `getRGB(...)` et ce que l'on passe à la fonction `setRGB(...)`.

Dans le cas du miroir Vertical on ne change pas la colorimétrie des pixels, nous n'aurons donc pas de problème de ce côté là, mais que fait-on alors ?

On prend la valeur du pixel de coordonnée (row,col) dans l'image 1 et on la copie dans le pixel de coordonnée (row, col) de l'image resultat... et l'inversion alors ?

Comment réécrire cette ligne pour que l'image résultat soit inversé gauche-droite ?

Aidez-vous de la Figure 3.

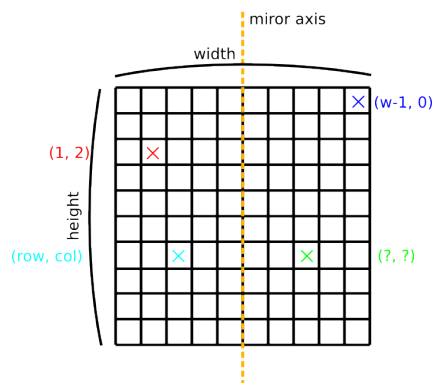


FIG. 3 – Où envoie-t-on le pixel cyan ?

5 Au boulot !

Une fois cette mise en bouche terminée, implémentez les autres fonctions.

5.1 Miroir Horizontal

Inversion des pixels haut-bas. Un jeu d'enfant désormais !

5.2 Quelques explications sur le int RGB

On a dit plus haut que `getRGB(...)` renvoyait un entier 32bits contenant les valeurs des trois canaux R,G,B synthétisées. Comment va-t-on pouvoir extraire la composante Rouge, Verte ou Bleue ?

Pas de panique ! Les quatres fonctions ci-dessous sont déjà implémentée.

```
private int getBlue(int RGB) {
    return RGB & 0xFF;
}

private int getGreen(int RGB) {
    return (RGB >> 8) & 0xFF;
}

private int getRed(int RGB) {
    return (RGB >> 16) & 0xFF;
}

private int makeRGB(int red, int green, int blue) {
    return ((blue & 0xFF) + ((green & 0xFF) << 8)
        + ((red & 0xFF) << 16)) + (0xFF << 24));
}
```

Cela parait du charabia ? Dites-vous que les trois premières permettent de passer de l'int RGB à un int entre 0 et 255 et que la dernière vous permet de régénérer un int RGB a partir de trois nombre entre 0 et 255 !

Si vous voulez vraiment comprendre ce qui se passe, il faut se rappeler des opérateurs binaires » (décalage à droite), « (décalage à gauche), & (et binaire) et | (ou binaire). Les opérateurs de décalage font "glisser" les bits du nombre vers la gauche ou la droite : $8 \gg 1$ donnera la valeur 4.

En mémoire :

```
int => 8   |   (8>>1) |   4
bin => 1000 |   0100   |   0100
```

Les opérateurs logiques s'appliquent bit par bit sur les entiers auxquels ils sont appliqués : $0x34 \& 0xF0$ donnera la valeur $0x30$.

En mémoire :

```
hex => 3   4   |   F   0
bin => 0011 0100 | 1111 0000
```

et donc $0x34 \& 0xF0$:

```
0011 0100
&
1111 0000
-----
0011 0000 => soit 0x30
```

Le canal rouge correspond aux bits 8-15 de notre entier, on fait donc un décalage de 16 vers la droite puis comme ce qu'il y a au-delà (le canal alpha) ne nous intéresse pas, on le supprime par l'opération binaire & avec $0xFF$.

Rappel :

```
ARGB => 255, 163, 240, 0
hex => F F A 3 F 0 0 0
bin => 1111 1111 0110 0011 1111 0000 0000 0000
int => 4284739584
```

En mémoire :

```
rgb (int)    => 4284739584
rgb (bin)    => 1111 1111 0110 0011 1111 0000 0000 0000
(rgb >> 16) => 0000 0000 0000 0000 1111 1111 0110 0011
& 0xFF      ( 0000 0000 0000 0000 0000 0000 1111 1111 )
r (bin)      => 0000 0000 0000 0000 0000 0000 0110 0011
r (int)      => 163
```

Les fonctions ci-dessus devraient vous paraître un peu plus explicites normalement !

5.3 Niveau de gris

On désire ôter l'information couleur de l'image 1. Pour ce faire on peut utiliser plusieurs formules :

$$G = \frac{R+V+B}{3} \quad (1)$$

$$G = 0.3 \times R + 0.59 \times V + 0.11 \times B \quad (2)$$

```
Ouvrir l'image 1
Créer une image résultat de même taille que l'image 1.
Pour chaque colonne
| Pour chaque ligne
| | Extraire les canaux Rouge, Vert et Bleu du pixel de l'image 1
| | Calculer la valeur de niveau de gris correspondante.
| | Ecrire la valeur de l'entier RGB dans le pixel de l'image resultat
```

Implémentez les deux formules. Voyez-vous une différence significative ?

Rappel : en RGB, les niveau de gris sont représenté par les triplets dont les trois composantes sont égales ([0,0,0], [128,128,128], [255, 255, 255]).

5.4 Composition colorée

Implémentez l'algorithme qui prend trois images en niveau de gris en entrée et qui produit une image où la composante bleue est tirée de l'image 1, la composante verte est tirée de l'image 2 et la composante rouge est tirée de l'image 3.

Faites attention de vérifier que les trois images aient la même taille !

Essayez avec les images Meteosat (1 : VIS006, 2 : VIS008, 3 : IR_016). On appelle cette composition colorée Fausse couleur. Pouvez-vous dire pourquoi ?

5.5 Aérosols

On utilise les compositions colorées avec les images satellites pour mettre en évidence des phénomènes que l'on veut observer. En utilisant les différences de valeurs radiométriques dans certains canaux, on peut faire ressortir l'objet de notre étude.

Pour visualiser les aérosols, c'est-à-dire les particules en suspension dans l'atmosphère (poussière, sable) on utilise une composition colorée basée sur les canaux IR_087, IR_108 et IR_120 selon la répartition suivante :

$$\begin{cases} R = IR_{108} - IR_{087} \\ G = IR_{120} - IR_{108} \\ B = IR_{108} \end{cases} \quad (3)$$

Implémentez l'algorithme qui prend trois images en niveau de gris en entrée (1 : IR_087, 2 : IR_108, 3 : IR_120) et qui produit une composition colorée respectant la relation précédente.

Faites attention de vérifier que les trois images aient la même taille !

Attention ! Si l'on soustrait deux nombres compris dans l'intervalle [0..255], on obtient un nombre entre... -255 et 255. Il faut donc penser à rapporter cette valeur dans [0..255].

Il ne faut surtout pas utiliser le modulo, la valeur absolue ou tout traitement séparé des données (si... alors) ! En effet, l'analyse visuelle repose sur la cohérence des données et l'utilisation de fonction non bijective introduirait un repliement des couleurs.

Que faire alors ? Trouvez la fonction affine $ax + b$ qui transforme un interval en l'autre. Pour ce faire il y a une méthode très simple :

- Soit un interval $[m \ M]$ de départ et $[0 \ 255]$ à l'arrivée.
- On uniformise les origines en soustrayant m : $[m \ M] \Rightarrow [0 \ M-m]$
- On divise par le coefficient proportionnel $\frac{255}{M-m}$: $[0 \ M-m] \Rightarrow [0 \ 255]$

A vous de jouer !

5.6 NDVI

Le NDVI (Indice de Végétation basé sur la Différence Normalisé) est une valeur entre -1 et 1 définie comme suit :

$$NDVI = \frac{NIR - RED}{NIR + RED} \quad (4)$$

Implémentez l'algorithme qui prend deux images en niveau de gris en entrée et qui produit une image en niveau de gris correspondant au NDVI.

On prendra 1 :VIS006 = RED et 2 :VIS008 = NIR.

Attention ! (NIR+RED) peut valoir 0, dans ce cas la convention dit que NDVI est saturé à 1.

Attention ! Le NDVI est une valeur réelle appartenant à l'intervalle [-1..1]. Il faut donc la ramener à une valeur entière sur 1 octet [0..255].

Attention ! Le NDVI [0..255] correspond à une composante unique. Comme dans le cas du niveau de gris, utilisez `makeRGB(...)`.

Faites attention de vérifier que les deux images aient la même taille !

Bonus : Les valeurs de végétation caractéristiques se trouvent entre 0,2 et 0,9 de NDVI. Pouvez vous rajouter un test qui permettrait de colorer en vert les pixels répondant à ce critère ?

5.7 Convolution

Dernière étape !

Voyons voir comment appliquer un filtre de convolution à notre image.

On parle de convolution d'une image par une matrice quand :

```

Considérant une matrice N x N.

Pour chaque colonne (sauf pour les N/2 premières et dernières)
| Pour chaque ligne (sauf pour N/2 premières et dernières)
| |
| | Calculer la produit élément par élément de la matrice avec le
| | voisinage du pixel.
| | Ecrire la valeur résultante dans l'image resultat
    
```

Prenons un exemple : soit M notre matrice de convolution N x N et I notre image.

$$M = \begin{pmatrix} m_{(0,0)} & \dots & m_{(N,0)} \\ \dots & m_{(i,j)} & \dots \\ m_{(0,N)} & \dots & m_{(N,N)} \end{pmatrix} \quad (5)$$

$$I = \begin{pmatrix} i_{(0,0)} & \dots & i_{(w,0)} \\ \dots & i_{(col-1,row-1)} & i_{(col,row-1)} & i_{(col+1,row-1)} & \dots \\ \dots & i_{(col-1,row)} & i_{(col,row)} & i_{(col+1,row)} & \dots \\ \dots & i_{(col-1,row+1)} & i_{(col,row+1)} & i_{(col+1,row+1)} & \dots \\ i_{(0,h)} & \dots & \dots & \dots & i_{(w,h)} \end{pmatrix} \quad (6)$$

Pour le pixel de coordonnées (col, row) la multiplication élément par élément donnera :

$$v = \sum_{i=0}^N \sum_{j=0}^N i_{(col-\frac{N}{2}-i,row-\frac{N}{2}-j)} \times m_{(i,j)} \quad (7)$$

Attention ! Comme vous ne pouvez pas savoir à l'avance, quel est l'intervalle des valeurs possible en sortie de la convolution, il vous faudra :

- soit centrer (+128) et saturer votre résultat (en considérant une matrice dont la somme des coefficient fait 0 (intervalle [-V..V])).
 - soit calculer la valeur v_{min} et v_{max} et projeter les valeurs sur [0..255].
- Pour calculer v_{min} et v_{max} , une méthode proposer par Florent Martin et Rémi Jacques Le Seigneur :

```

Initialiser le minimum et le maximum à 0
Pour chaque case de la matrice M
| Si M(i,j) est négatif, ajouter M(i,j) * 255 au minimum
| Si M(i,j) est positif, ajouter M(i,j) * 255 au maximum
    
```

6 Conclusion

Bravo ! Si vous êtes arrivés jusqu'ici, vous êtes maintenant prêts à tester par vous-même tous les algorithmes possibles et imaginables de traitement d'image !

Faites-vous plaisir !