Documentation IMAGE J

INSTALLATION :
ImageJ est installé dans une des machines virtuelles fournies par le DI. ImageJ est normalement installé dans le réperetoire Mes_Documents/ImageJ

Pour l'installer sur votre PC: http://imagej.nih.gov/ij/
User guide officiel : https://imagej.nih.gov/ij/docs/guide/146.html

GUIDE D'UTILISATION ET FORUM :

- Help>Documentation>Image J User guide
- Help>List Archive

http://imagejdocu.tudor.lu
http://imagej.net/Welcome

OUVERTURES IMAGES :

- Faire glisser les images sur la barre d'outils d'imageJ
- ATTENTION : toujours penser à dupliquer l'image avant de la travailler, afin de ne pas perdre l'originale :
    - Image>Duplicate
    - Sinon File>Revert pour revenir à l'image initiale (ne marche pas avec toutes les opérations effectuées)

DEFINITION IMAGE
Format image (Image>Type)

- Image binaire : noir et blanc
- 8 bits : 256 niveaux de gris (chaque pixel de l'image a une intensité comprise entre 0 et 255)
- 12 bits : 4096 niveaux de gris
- 16 bits : 65536 niveaux de gris
- Image RGB (red, green, blue) : 8 bits dans chaque couleurs, soit 24 bits
- Stack : 3 dimensions (x ; y ; z) ou (x ; y ; temps)
- Hyperstack : plus de trois dimension

BARRE D'OUTILS

- Lorsque la souris est sur l'image : il est indiqué (en bas à gauche de la barre d'outils) la position en x, y, ainsi que le niveau de gris (intensité) du pixel pointé (des trois niveaux de gris si l'image est RGB)

- **Les outils de sélections : du rectangle au point**
  - Modifier leurs propriétés : double clics sur l'icône
  - Les garder en mémoire : Analyse>tools>ROI Manager

- **ROI Manager** (ouvre une fenêtre spécifique) :
  - Garde en mémoire les sélections,
  - Permet de faire des mesures sur une ou plusieurs sélections
  - Pour les intégrer à l'image : onglet Flatten

REDUIRE LE BRUIT - FILTRES

- **Mean Filter** : remplace chaque pixel dans l'image par une moyenne des intensités des pixels voisins
  - Process>Filters>Mean
- **Gaussian Blur Filter** : même principe que Mean Filter mais cette fois le procédé mathématique suit une distribution Gaussienne
  - Process>Filters>Gaussian Blur
- **Median Filter** : enlever le bruit de fond aléatoire créer par le détecteur (bruit dit poivre et sel)
  - Process>Filters>Median
- **Minimum and Maximum Filter** : le filtre Min augmente les régions de faibles intensités, le filtre Max augmente les régions de fortes intensités
  - Process>Filters>Minimum or Maximum

SATURATION DE L'IMAGE

- Image>Lookup Table>HiLo : l'intensité minimale (=0) en bleu ; l'intensité maximale (=255 pour image 8 bits) en rouge
- Pour créer une LUT (Lookup Table) et donner son propre code couleurs aux différentes intensités :
  - Image>Color>Edit LUT

SELECTIONS D'OBJETS :

- Image>Adjust>Threshold : permet de séparer les objets entre eux en fonction de leur intensité en niveaux de gris, ou les objets avec le bruit de fond (la case à droite règle la couleur de ce qui est sélectionné; le graphique est le nombre de pixel (y) sur l'intensité (x)
- Analyse>Analyse particles : discriminer les particules en fonctions de leur taille, de leur circularité, possibilité d'exclure ceux qui sont aux bords (exclude on edges), les ajouter au ROI Manager pour effectuer des mesures dessus et les compter…

COMPTAGE D'OBJETS :

- Analyse particle (vu précedement) : automatique
- Points selection : compter manuellement les objets en cliquant dessus avec l'option multi point tool, et les enregistrer sur le ROI Manager
- Magic Wand Tool : une fois le seuil (=threshold) réglé, on clique sur les objets avec cet outil et il va définir le contour de chaque objet sélectionné. Pour avoir le comptage il faut enregistrer chaque sélection dans le ROI Manager

MACROS ET PLUGINS :

Développement de méthodes d'analyse d'images « à façon » :

- Help>Plugins ou Macros
- Les macros et plugins sont à glisser directement sur la barre d'outils de imageJ. Une fenêtre d'enregistrement s'ouvre alors pour les plugins
- Pour les macros une fenêtre de commande de la macro s'ouvre, il suffit de faire save as pour l'enregistrer
- La mise à jours des plugins et macros ajoutés s'effectue par
    - Help>Refresh menu
    - Ou fermer et rouvrir image J

UTILISATION DE L'OUTIL MACRO :

- Enregistrer une succession de commandes pour éviter de les refaire une à une sur chaque image
- Plugins>Macros>Record : une fenêtre s'ouvre et enregistre toutes les commandes que vous choisissez, quand vous avez terminé cliquez sur Create. La fenêtre de la macro s'ouvre et vous pouvez l'enregistrer en format « .txt ». Pour l'ouvrir Plugins>Macros>Run et aller la chercher dans le dossier Macros

LES PLUGINS IMAGEJ

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/).

Installation

- Télécharger l'un des nombreux plugins sur http://rsbweb.nih.gov/ij/plugins.
- Copier simplement le fichier .class ou .jar dans (un sous-répertoire de) <ij>/plugins.
- Redémarrer ImageJ pour que l'installation soit prise en compte (*cf*. menu *Plugins*)

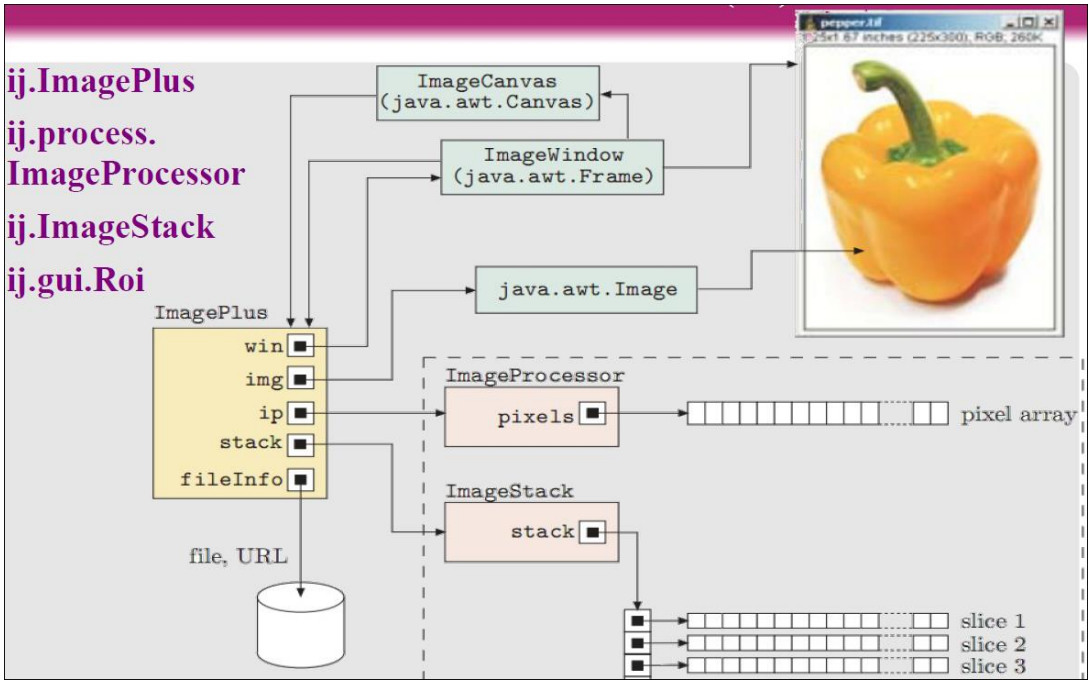
Remarques

- 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 fondamentalesdamentales de l'API (1/2)



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
- ImageStack getStack() : la pile d'images (éventuellement réduite à 1 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
- Overlay getOverlay() : les éléments superposés à l'image
- FileInfo getFileInfo() : le fichier image

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.

2 interfaces définissant 2 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
- Il en existe d'autres (cf examples)

Plugin implémentant PlugIn

1 seule méthode appelée, qui implémente ce que réalise effectivement le plugin :
void run(java.lang.String arg)

Plugin implémentant PlugInFilter

- 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
    setup() 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()

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
- 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*

Premier exemple : Inversion d'une image

```
// Importation des paquets nécessaires
import ij.*; // pour classes ImagePlus et IJ
import ij.plugin.filter.PlugInFilter; // pour interface PlugInFilter
import ij.process.*; // pour classe ImageProcessor
import java.awt.*; // pour classe Rectangle

// Nom de la classe = nom du fichier. Implémente l'interface PlugInFilter
public class Image_Inverter implements PlugInFilter {
	public void run(ImageProcessor ip) {
		Rectangle r = ip.getRoi(); // Région d'intérêt sélectionnée (r.x=r.y=0 si aucune)
		for (int y=r.y; y<(r.y+r.height); y++)
			for (int x=r.x; x<(r.x+r.width); x++)
				ip.set(x, y, ~ip.get(x,y)); // Complément bit à bit des valeurs des pixels
	}
	public int setup(String arg, ImagePlus imp) {
		if (IJ.versionLessThan("1.37j")) // Requiert la version 1.37j d'ImageJ
			return DONE; // Ne pas appeler la méthode run()
		else // Accepte tous types d'images, piles d'images et RoIs, même non rectangulaires
			return DOES_8G+DOES_RGB+DOES_STACKS+SUPPORTS_MASKING;
	}
}

// Version 2 : image complete
public class Image_Inverter implements PlugInFilter {
	public void run(ImageProcessor ip){
		int width = ip.getWidth();  int height = ip.getHeight();
		for (int x = 0; x < width; x++)
			for(int y = 0; y < height; y++)
				ip.putPixel(x, y, 255- ip.getPixel(x, y));
	}
	public int setup(String arg, ImagePlus imp){
		if (arg.equals("about")){
			IJ.showMessage("Inversion de l'image");
			return DONE;
		}
		return DOES_8G;
	}
}
```

Accès ponctuel à un pixel dans un ImageProcessor

La classe ImageProcessor possède notamment les méthodes suivantes :

- getWidth() qui retourne la largeur de l'image.
- getHeight() qui retourne la hauteur de l'image.
- getPixel(x, y) qui retourne un entier correspondant à la valeur du pixel de coordonnées (x, y).
- putPixel(x, y, val qui permet de donner une nouvelle valeur au pixel (x, y).
- getPixels qui permet de récupérer toutes les valeurs de l'image dans un tableau mono-dimensionnel (tableau de byte pour une image en niveaux de gris, tableau d'int pour une image couleur). En modifiant les valeurs de ce tableau, on modifie directement l'image.

Détails : Avec vérification des coordonnées :

Pour ColorProcessor :

- int getPixel(int *x*, int *y*) retourne un **entier** sur 4 octets
  l'entier stocke les 4 composantes dans l'ordre x**RGB**
- int[] getPixel(int *x*, int *y*, int[] *iArray*)
  l'entier stocke les 3 composantes dans l'ordre **RGB R** alors en [0]
  Exemple :
```
int [] c = new int [3];
c = ip.getPixel (x, y, c);
```

Pour FloatProcessor :

- int getPixel(int *x*, int *y*) retourne un entier
  l'entier *p* contient les bits correspondant au flottant lu et doit être converti avec Float.intBitsToFloat(*p*).
- float getPixelValue(int *x*, int *y*) qui retourne un float

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*)
- pour FloatProcessor, utiliser anImageProcessor.getf(*pix_index*).

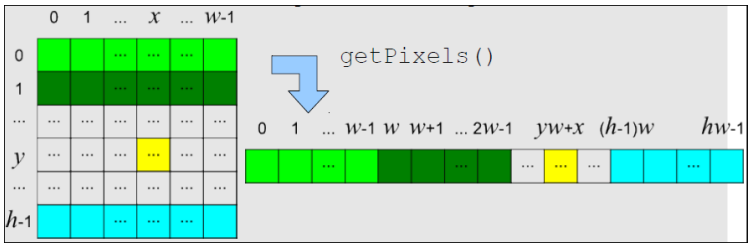
Accès en écriture :

- void putPixel(int *x*, int *y*, int *value*), putPixelValue(int *x*, int *y*, double *value*)
- void set(int *pix_index*, int *value*), setf(int *pix_index*, float *value*)

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 :
  if (anImageProcessor instanceof ColorProcessor)
  int[] pixels = (int[]) anImageProcessor.getPixels();
- Plus performant qu'un accès ponctuel (getPixel()) à chacun des pixels
- Cas part. d'un ColorProcessor :
  void getRGB (byte[] *R,* byte[] *G,* byte[] B)



Accès aux pixels dans un tableau 2D

- Pour un {Byte|Short|Color}Processor : int[][] getIntArray() // coords : [x][y]
- Pour un FloatProcessor : float[][] getFloatArray()

Problème : Élimination du bit de signe

Les méthodes d'accès (ponctuel ou global) aux pixels retournent des valeurs entières **signées** (byte, short, int) ; *ex*. byte [-128..+127].
Or le plus souvent, on souhaite des valeurs **positives** de luminance [0..255]

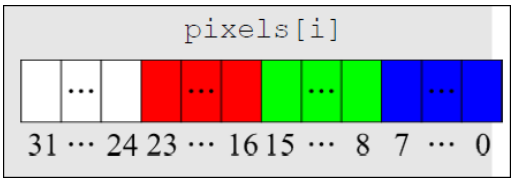
Solution : conversion en entier (int) **non signé** (positif) par **masquage**.

- Exemple pour une image ip en niveaux de gris sur 8 bits (8G) :

```
byte[] pixels = ( byte[] ) ip.getPixels();
...
int grey=pixels[i] & 0xFF;
newPixels[i] = (byte)grey;          // exemple de calcul d'une nouvelle image 8G
```

- Exemple pour une image ip couleur 32b (RGB)

```
int[] pixels=(int[]) ip.getPixels();
    int r=(pixels[i] & 0xFF0000)>>16; // rouge
    int g=(pixels[i] & 0x00FF00)>>8; // vert
    int b=(pixels[i] & 0x0000FF); // bleu
```



Exemple 2 : Supprimer une couleur

```
public class Suppression_Rouge_ implements PlugInFilter {
    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")){
            IJ.showMessage("Suppression du canal rouge");
            return DONE;
        }
        return DOES_RGB;
    }
    public void run(ImageProcessor ip) {
        int w = ip.getWidth();
        int h = ip.getHeight();
        ImagePlus res = NewImage.createRGBImage ("Sans rouge", w, h, 1,
                NewImage.FILL_BLACK);
        ImageProcessor ipRes = res.getProcessor();
        int[] pixels =  (int[]) ip.getPixels();
        int[] pixelsRes = (int[]) ipRes.getPixels();
        for (int i=0; i<w*h; i++){
            int c = pixels[i];
            int r = (c&0xff0000)>>16;
          int g = (c&0x00ff00) >>8;
          int b = (c&0x0000ff);
          r = 0;
          pixelsRes[i] = (r << 16) + (g << 8) + b;
        }
        res.show();
        res.updateAndDraw();
    }
}
```

Exemple 2bis : Compter le nombre de couleurs différentes présentes dans une image

```
// Importation des paquets (non détaillée)
public class Color_Counter implements PlugInFilter {
	ImagePlus imp;
	int colors;
	static int MAX_COLORS = 256*256*256;
	int[] counts = new int[MAX_COLORS]; // Histogramme des couleurs

	public int setup(String arg, ImagePlus imp) {
		this.imp = imp; // Rem.: pas utilisé dans run()
		return DOES_RGB+NO_UNDO+NO_CHANGES; // Traite les images couleurs ; pas d'annulation
	}

	public void run(ImageProcessor ip) {
		int[] pixels = (int[])ip.getPixels();
		for (int i=0; i<pixels.length; i++)
			counts[pixels[i]&0xffffff]++; // Masquage nécessaire car pixels[i] est signé
		for (int i=0; i<MAX_COLORS; i++) { // Comptage des couleurs différentes
			if (counts[i]>0) colors++;
		}

		IJ.log("Couleurs différentes : "+colors);

		if (colors<=64) { // Affichage des couleurs trouvées dans fenêtre Log (si < de 64)
			IJ.log("Counts");
			for (int i=0; i<MAX_COLORS; i++) {
			if (counts[i]>0) IJ.log(" "+Integer.toHexString(i)+": "+counts[i]);
			}
		}
	}
}
```

Exemple 3 : Lancer des commandes ImageJ dans un plugins

Voir menu ImageJ :

- Help>Examples>Java
- Plugins>Utilities>Findcommands
- Plugins>Shortcups>ListCommands

```
public class My_Plugin implements PlugIn {

    public void run(String arg) {
        ImagePlus imp = IJ.getImage();
        IJ.run(imp, "Invert", "");
        IJ.wait(1000);
        IJ.run(imp, "Invert", "");
    }
```

```
public class Filter_Plugin implements PlugInFilter {
    ImagePlus imp;
    public int setup(String arg, ImagePlus imp) {
        this.imp = imp;
        return DOES_ALL;
    }
    public void run(ImageProcessor ip) {
        ip.invert();
    }
}
```

PLUGINS créé dans un JAR

- Plugins can go in the default package or a class with a package declaration.
- Be sure to include an '_' in the filename of the jar file!
- Create a 'plugins.config' file as part of your jar.
    - This tells ImageJ where to locate your plugin in it's menus.
    - The whitespace separating each element in a line must be a space (\s) character and not a tab (\t).
    - Exemple de fichier plugins.config

    This is a comment (empty lines are also ignored)
    This line will add "Blob" to the "New" submenu of the "File" menu.
    Clicking on "Blob" will call the plugin class "my.test.Test"
    File>New, "Blob", my.test.Test

- On Windows, use the jar utility included with the Java Software Development Kit (SDK) from Sun:
- Open a command line window
- Change (cd) to the directory containing the plugin(s) and plugins.config file
- Type "C:\jdk1.5.0\bin\jar cvf Name_of_Package.jar *" (assumes the SDK is installed in C:\jdk1.5.0)

Tutoriels sur les Plugins ImageJ

- Tutoriels : « ***Programmation ImageJ*** » (avec intro à Java), http://imagej.nih.gov/ij/docs/examples
- Podlasov, E. Aggenko, ***Working and Development with ImageJ – A student***
- ***reference***, 2003 : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.4181&rep=rep1&type=pdf
- W. Bailer, ***Writing ImageJ PlugIns – A Tutorial***, 2006 et W. Burger, M. J. Burge, ***ImageJ Short Reference***, 2007 : http://imagingbook.com/imagej-tutorial/
- ressources par J. Ross, dont « ***Using and Writing Macros in ImageJ*** », 2007 http://tinyurl.com/oblr5s2

Annexe A : Encore des exemples de plugins

Convolution 5x5

```
import ij.*;
import ij.process.*;
import ij.gui.*;
import ij.plugin.filter.*;

public class Filtre_Moyen implements PlugInFilter {

    public void run(ImageProcessor ip){
            int w = ip.getWidth();
            int h = ip.getHeight();
            ImagePlus result = NewImage.createByteImage ("Filtrage", w, h, 1, NewImage.FILL_BLACK);
            ImageProcessor ipr = result.getProcessor();

            int [][] masque = {{1, 1, 1, 1, 1}, {1, 1, 1, 1, 1}, {1, 1, 1, 1, 1},{1, 1, 1, 1, 1},{1, 1, 1, 1, 1}};
            int n = 2; // taille du demi-masque ou n=masque.length/2

            int somme_coef = 0;
            for (int u = -n; u <= n; u++)
                    for(int v = -n; v <= n; v++)
                            somme_coef += masque[u+n][v+n];

            for(int y = n; y < h-n; y++)
                    for (int x = n; x < w-n; x++) {
                            int s = 0;
                            for (int u = -n; u <= n; u++)
                                    for(int v = -n; v <= n; v++)
                                            // calcul de la somme ponderee
                                            s += ip.getPixel(x+u,y+v)*masque[u+n][v+n];

                            ipr.putPixel(x, y, s/somme_coef);
                    }
            result.show();
            result.updateAndDraw();
```

Retailler une image => avec Dialog box

```
import ij.*;
import ij.process.*;
import ij.gui.*;
import ij.plugin.filter.*;

public class Retailler implements PlugInFilter {

    public void run(ImageProcessor ip){
            // dialogue permettant de fixer la valeur du ratio .
            GenericDialog gd = new GenericDialog( "Facteur de taille",
                            IJ.getInstance() );
            gd.addSlider( "ratio", 0.0, 10.0, 2.0 );
            gd.showDialog();
            if ( gd.wasCanceled() ) {
                    IJ.error( "PlugIn cancelled" );
                    return;
            }
            double ratio = (double) gd.getNextNumber();


            int w = ip.getWidth();
            int h = ip.getHeight();
            int w_new = (int) (w*ratio);
            int h_new = (int) (h*ratio);
            ImagePlus result = NewImage.createByteImage ("Après retaillage",
                            w_new, h_new, 1, NewImage.FILL_BLACK);
            ImageProcessor ipr = result.getProcessor();

//          parcours de l'image résultat
            for(int y = 0; y < h_new; y++)
                    for (int x = 0; x < w_new; x++) {
                            // recherche du pt correspondant dans l'image initiale
                            int xx = (int)(x/ratio);
                            int yy = (int)(y/ratio);

                            ipr.putPixel(x,  y,  ip.getPixel(xx,  yy));
                    }
            result.show();
            result.updateAndDraw();
    }


    public int setup(String arg, ImagePlus imp){
            if (arg.equals("about")){
                    IJ.showMessage("Retaille l'image suivant le même facteur en x et en y");
                    return DONE;
            }
            return DOES_8G;
    }
}
```