



Université
de Lille
1 SCIENCES
ET TECHNOLOGIES

Université de Lille
MASTER 1 INFORMATIQUE
Deuxième semestre



**FACULTÉ
DES SCIENCES ET
TECHNOLOGIES**
Département Informatique

Traitement d'image

TP n°10

BARCHID Sami

CARTON Floriane

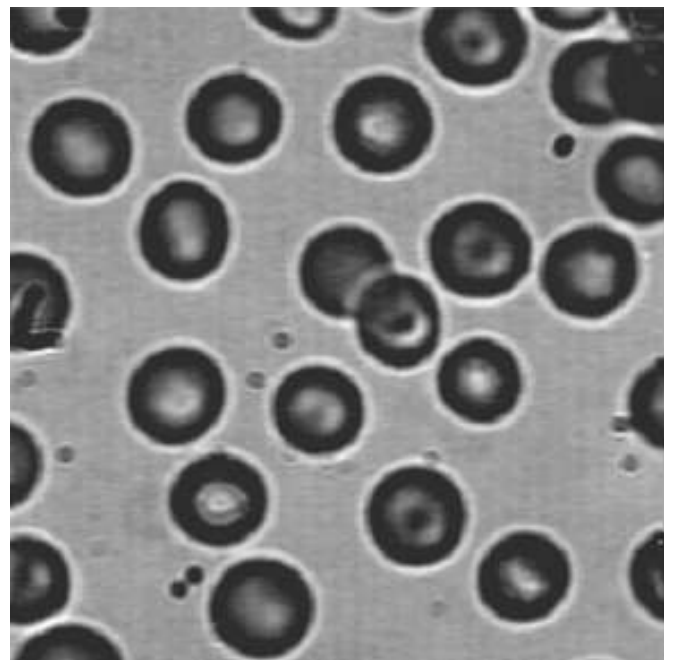
Introduction

La matière vue dans ce TP traite de la détection de contours en utilisant une approche par dérivée seconde de l'image. Dans ce rapport seront testées et interprétées diverses expériences portant sur les étapes de la détection de contour par approche du second ordre.

Les manipulations se feront grâce au logiciel « ImageJ » avec la programmation de plugins java. Deux images (en niveaux de gris, codées sur 8 bits) seront utilisées lors de nos expériences : « spores », l'image principale, et « phare ».



Phare



Spores

Pour effectuer les opérations en lien avec les expériences à faire dans ce TP, nous ajouterons des instructions aux archives du plugin « Laplacien » fourni pour ce TP.

Ce rapport de TP est divisé en 3 parties,

- **Calcul du Laplacien** : expérimentations et interprétations sur le calcul du Laplacien de l'image par convolution.
- **Seuillage des passages par 0 du Laplacien** : expérimentations et interprétations sur le seuillage des passages par 0 du Laplacien.
- **Utilisation du filtre LoG et détection multi-échelles** : expérimentations sur l'utilisation du filtre LoG, de l'utilisation de la norme du gradient afin de seuiller les passages par 0 et comparaison des résultats obtenus grâce à la détection multi-échelle.

Calcul du Laplacien

Cette partie a pour but de calculer et afficher, au moyen du langage de programmation de plugins Java d'ImageJ, le Laplacien de l'image « Spores » par convolution.

Pour ce faire, nous allons rajouter deux opérations au code déjà existant du plugin Laplacien :

- Le calcul par convolution du Laplacien de l'image d'entrée dans une image 32 bits (afin de pouvoir travailler avec des valeurs négatives, puisque nous travaillons avec des dérivées secondes).
- L'affichage du résultat.

Ces deux opérations sont représentées par les instructions suivantes :

```
// Calcul du laplacien de l'image par convolution
fpLaplacian.convolve(MASQUES_LAPLACIENS3x3[filtre], 3, 3);

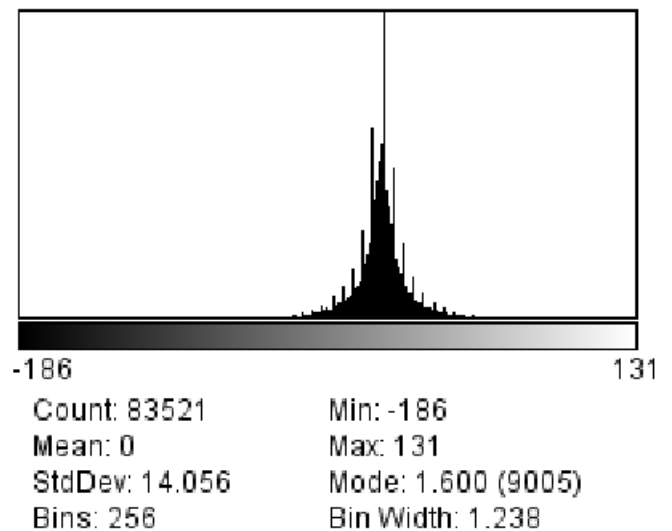
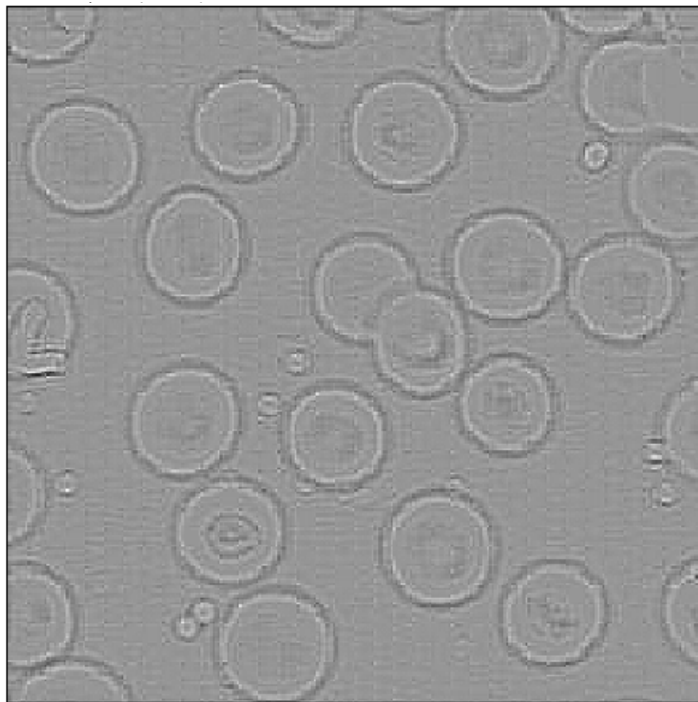
//Affichage du résultat
ImagePlus newImg = new ImagePlus("Résultat du lissage par masque laplacien", fpLaplacian);
newImg.show();
```

Le masque de convolution utilisé pour calculer le Laplacien est le suivant :

0	+1	0
+1	-4	+1
0	+1	0

Résultats sur l'image des spores

Le résultat du calcul du Laplacien sur les spores est le suivant :

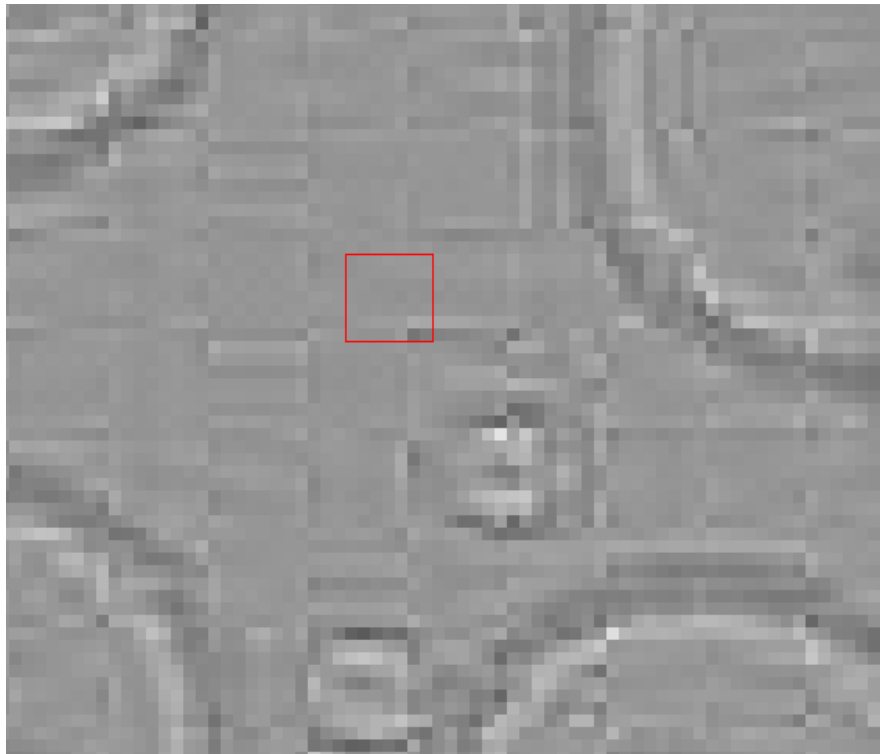


Laplacien des spores par convolution

Nous obtenons une image possédant des valeurs négatives et positives comprises entre $[-186, 131]$. De plus, sur l'image même, nous remarquons plusieurs choses :

- Les pixels du fond (qui ne sont donc pas des contours) possèdent des valeurs proches de 0. Cependant, on remarque aussi des valeurs plus éloignées de 0 qui sont dues à un bruit sur l'image d'origine (le fond n'est pas uniforme sur l'image des spores).
- Les pixels des contours des objets montrent une variation fortement marquée de pixels négatifs à positifs (avec, toujours, des valeurs éloignées de 0).

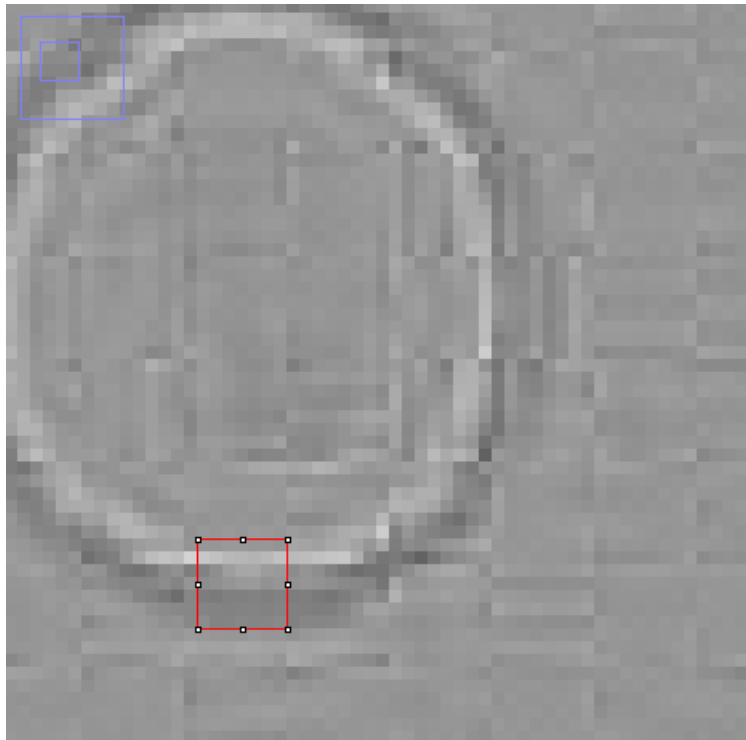
Nous pouvons observer ces deux remarques grâce à des zooms sur l'image du Laplacien avec une analyse des valeurs des pixels :



Prefs	107	108	109	110	111	112	113
130	-4.00	1.00	2.00	6.00	4.00	5.00	3.00
131	-4.00	-1.00	0.00	0.00	3.00	4.00	5.00
132	-1.00	-1.00	-4.00	-3.00	-3.00	-3.00	-5.00
133	-1.00	-2.00	-2.00	-3.00	-7.00	-1.00	-4.00
134	2.00	-2.00	1.00	-2.00	0.00	-2.00	1.00
135	1.00	5.00	9.00	9.00	8.00	26.00	22.00
136	-2.00	-5.00	-4.00	-5.00	8.00	-41.00	-28.00

Zoom sur le fond de l'image du Laplacien des spores

Ici, nous constatons que le fond de l'image possède bel et bien des valeurs proches de 0. Cependant, vers le coin inférieur droit du périmètre d'analyse des valeurs des pixels, on voit des valeurs bien moins proches de 0 voire des fortes variations de nombres négatifs et positifs éloignés de 0. Ces pixels dont les valeurs varient fortement seront problématiques dans la détection des points contours, car ce bruit risque d'être interprété comme du contour du fait de cette forte variation.

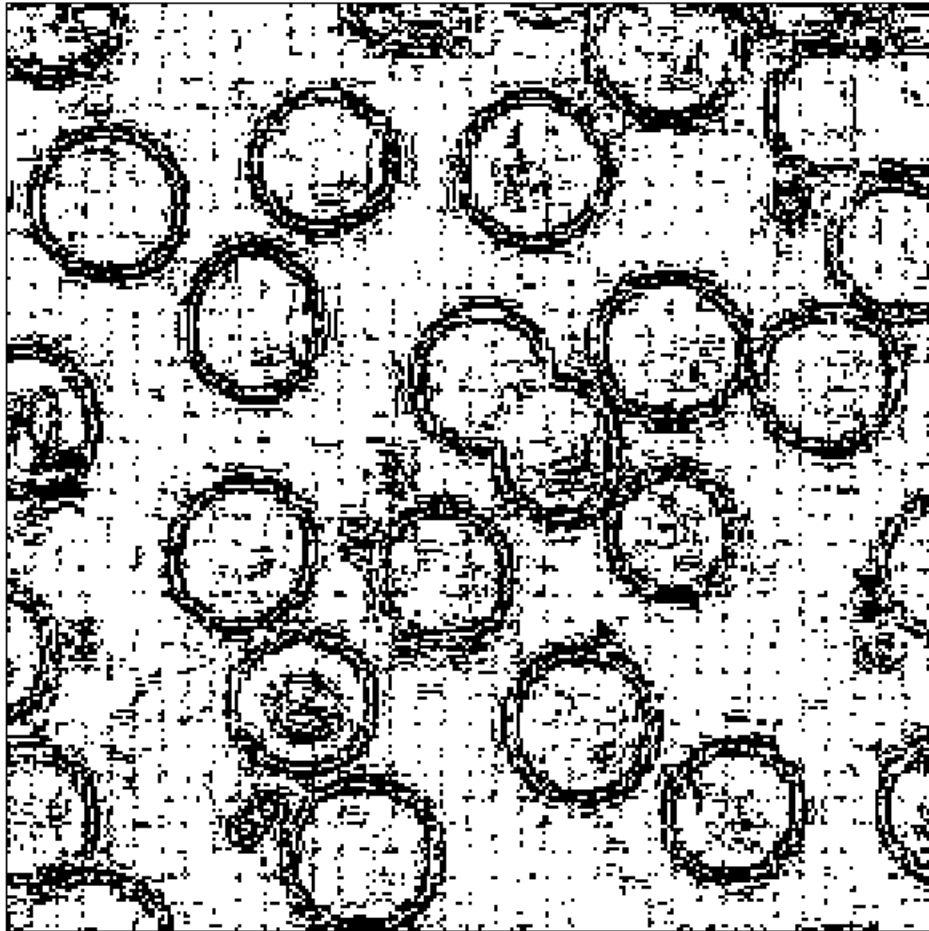


Prefs	73	74	75	76	77	78	79
118	2.00	1.00	4.00	4.00	2.00	2.00	-2.00
119	61.00	54.00	43.00	40.00	37.00	43.00	54.00
120	-25.00	-10.00	11.00	20.00	21.00	8.00	6.00
121	-11.00	-7.00	-1.00	4.00	5.00	-5.00	0.00
122	-31.00	-33.00	-27.00	-27.00	-24.00	-28.00	-24.00
123	-22.00	-23.00	-25.00	-23.00	-22.00	-25.00	-22.00
124	-12.00	-23.00	-22.00	-18.00	-21.00	-20.00	-16.00

Zoom sur un contour du Laplacien des spores

Comme on peut le voir sur ce zoom, les contours sont constitués d'un voisinage de pixels qui présentent une variation de pixels négatifs et positifs (donc, un passage par 0).

Pour appuyer nos observations, nous avons seuillé l'image du Laplacien des spores en mettant en évidence les pixels auxquels le laplacien a une valeur proche de 0.



Mise en évidence des pixels entre $[-10,10]$ dans le Laplacien de l'image des spores

Dans l'image ci-dessus, les pixels dont le laplacien a une valeur située entre -10 et 10 sont mis en blanc, le reste est en noir.

On voit donc que les contours sont bien en noir et les pixels du fond sont en majorité en blanc.

Cependant, on remarque aussi la présence du bruit dans le fond de l'image. On peut donc conclure que le Laplacien est très sensible au bruit. De plus, l'utilisation directe des valeurs nulle du Laplacien pour détecter les points contours ne fournirait pas un résultat satisfaisant car, comme les points contours sont détectés lors d'un passage par 0, il faut que ce passage soit bien marqué pour être pertinent. D'où l'importance de

rajouter la notion d'un seuil pour lequel un passage par 0 peut être réellement interprété comme un point contour.

Seuillage des passages par 0 du Laplacien

Le but de cette partie du TP est de comprendre l'avantage de l'utilisation d'un seuil pour la détection des passages par 0 du Laplacien.

Mode opératoire

Le mode opératoire du seuillage des passages par 0 à partir du Laplacien est le suivant :

Pour chaque pixel,

- Déterminer le voisinage $n \times n$ du pixel (Pour ce TP, ce sera 3×3)
- Imposer qu'il se trouve au moins une valeur positive et une valeur négative dans ce voisinage
- Imposer un seuil S et vérifier qu'il existe au moins un voisin du pixel $voisin > S$ et $voisin \leq -S$

Les instructions java ajoutées dans le plugin ImageJ pour implémenter l'algorithme du seuillage des passages par 0 du Laplacien sont décrites ci-dessous.

```

/**
 * Détection des passages par 0 du Laplacien (algo min<-seuil && max>seuil)
 *
 * @param imLaplacien Image du Laplacien (ImageProcessor 32 bits)
 * @param seuil Seuil sur les valeurs du Laplacien
 * @return imZeros Carte des passages par 0 (image binaire)
 */
public ByteProcessor laplacienZero(ImageProcessor imLaplacien, Float seuil) {

    int width = imLaplacien.getWidth();
    int height = imLaplacien.getHeight();

    // Image binaire résultat des points contours après seuillage
    ByteProcessor imZeros = new ByteProcessor(width, height);

    /* à compléter */
    // POUR CHAQUE pixel
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            float pixel = Float.intBitsToFloat(imLaplacien.getPixel(i, j));

            float minimal = Float.MAX_VALUE; // représente le voisin minimal
            float maximal = Float.MIN_VALUE; // représente le voisin maximal
            // Parcourir le voisinage (3x3) du pixel
            for (int x = i - 1; x <= i + 1; x++) {
                if (x < 0 || x >= width) {
                    continue;
                }
            }
        }
    }
}

```

```

    for (int y = j - 1; y <= j + 1; y++) {
        if (y < 0 || y >= height) {
            continue;
        }

        float voisin = Float.intBitsToFloat(imLaplacien.getPixel(x, y));
        minimal = minimal > voisin ? voisin : minimal;
        maximal = maximal < voisin ? voisin : maximal;
    }
}

```

```

// Assurer un changement de polarité dans le voisinage
// Vérifier que minimal est négatif
// Vérifier que maximal est positif
if (minimal >= 0 || maximal < 0) {
    imZeros.set(i, j, 0);
    continue;
}

```

```

// Imposition du seuil
if (minimal < -seuil && maximal > seuil) {
    imZeros.set(i, j, 255);
} else {
    IJ.log("mini = " + minimal + ", maxi = " + maximal + ", pixel=" + pixel);
    imZeros.set(i, j, 0);
}

```

```

}

```

```

}

```

```

return imZeros;

```

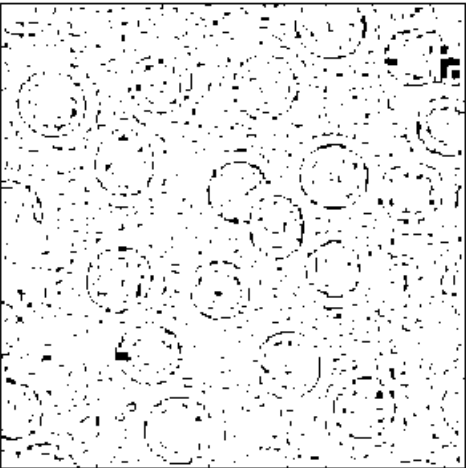
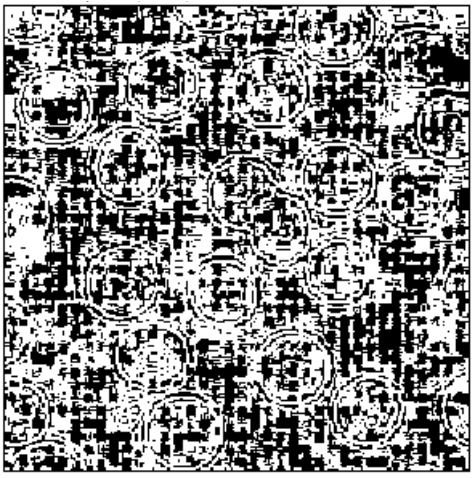
```

}

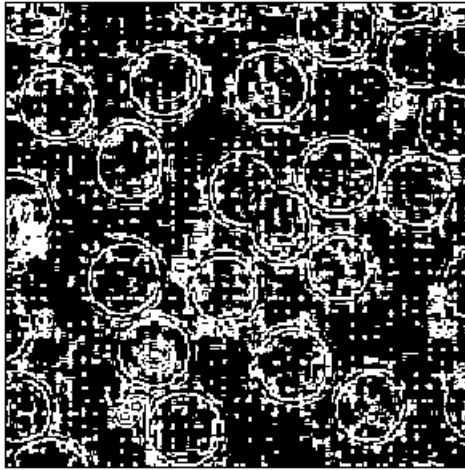
```

Seuillage manuel

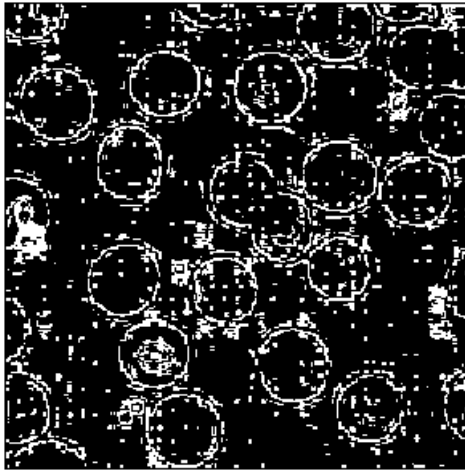
Nous avons ensuite expérimenté ce seuillage des passages par 0 du Laplacien de l'image des spores en définissant manuellement différents seuils pour comprendre ses effets. Le tableau suivant montre les résultats obtenus après le seuillage des passages par 0 en utilisant un seuil défini.

Seuil	Résultat
0 (Pas de seuil)	
5	

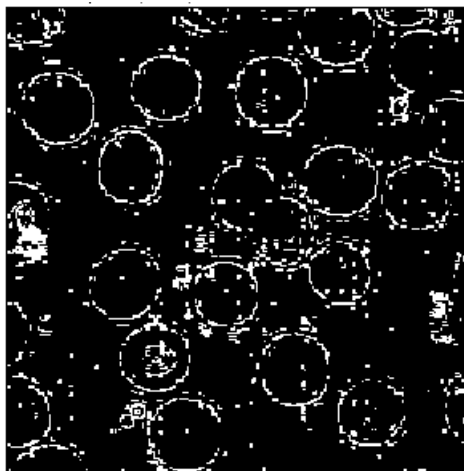
10



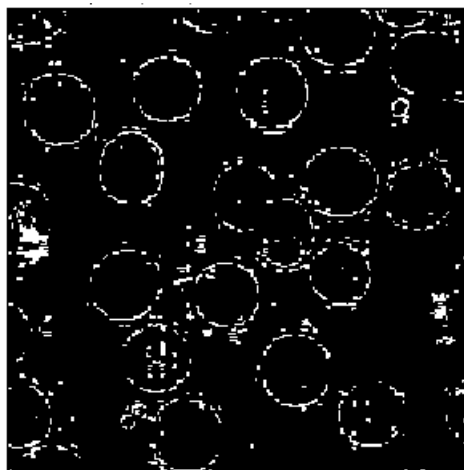
15



20



25



On remarque que, pour les valeurs de seuil les plus faibles, le bruit influence énormément la détermination des points contours. Cependant, plus le seuillage est élevé, moins on détecte de vrais points contours. On peut remarquer cela dans l'image du seuil à 25 où mêmes les contours principaux des spores ne sont pas pris en compte en entier.

Ici, parmi tous les tests effectués, le résultat obtenu avec un seuil de 15 semble être le plus pertinent. En effet, le bruit, bien que présent, a été un peu réduit tandis que les réels contours de l'image sont bien représentés.

On notera tout de même que le résultat conserve des défauts (contours épais, bruit, contours non-fermés, ...).

Seuillage automatique

La prochaine étape est de trouver le seuil le plus optimal pour l'image du phare, afin de comprendre comment trouver, de manière automatique, un seuil convenable.

Parmi les seuils testés, le résultat le plus convaincant pour le seuillage du Laplacien du phare est $S=34$, qui donne le résultat suivant :



Seuillage des passages par 0 du Laplacien des phares en utilisant un seuil $S = 34$

Ce résultat donne des contours fortement épais mais permet de limiter au mieux le bruit et de rendre quand même visibles les contours les moins marqués.

Lorsque nous analysons les seuils trouvés pour les spores et pour le phare, nous nous rendons compte que les seuils trouvés sont très proches des valeurs correspondant à l'écart-type du Laplacien.

Les valeurs de l'écart-type du Laplacien pour Spores et pour Phare sont les suivantes :

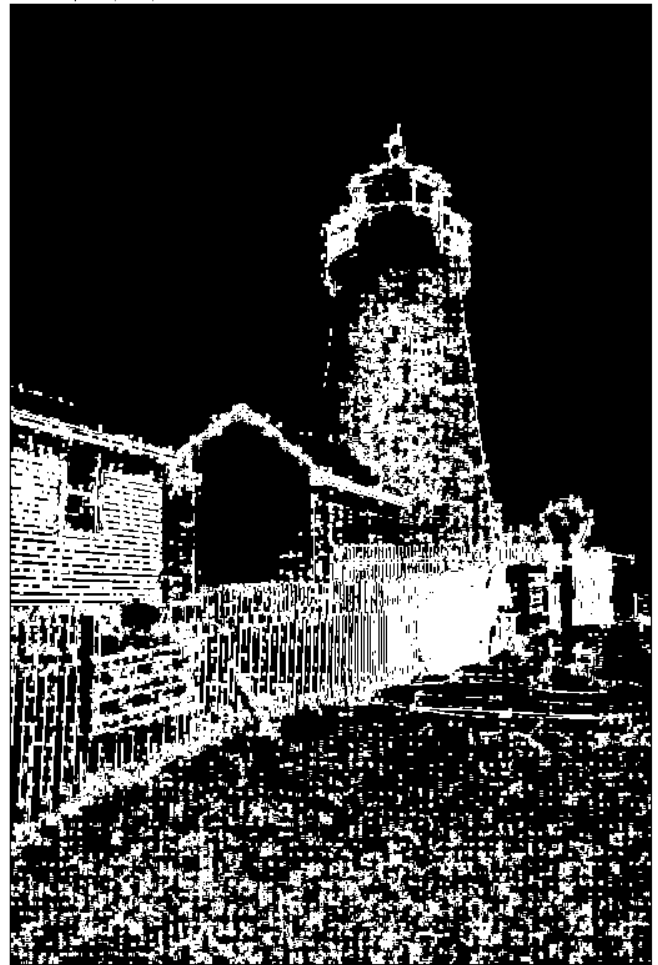
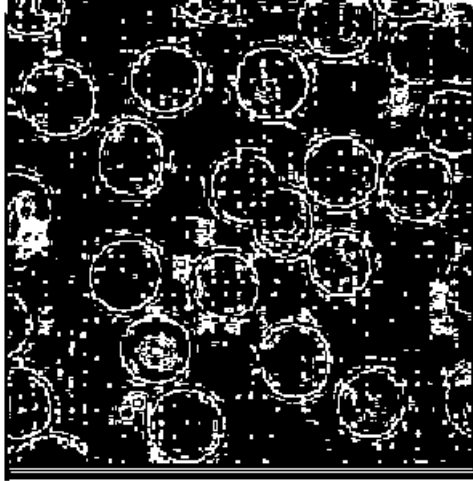
- $\sigma_{\text{laplacien de phare}} = 34.644$
- $\sigma_{\text{laplacien de spores}} = 14.056$

Le fait que les valeurs des écart-types des Laplacien font de bons seuils s'explique car, comme nous essayons de mesurer les passages par 0 les plus significatifs, nous limitons les passages par 0 par une valeur représentant la dispersion des valeurs des pixels.

Ainsi, pour calculer le seuil de manière automatique, nous devons utiliser l'instruction suivante :

```
// Le seuil calculé automatiquement est l'écart-type du Laplacien  
seuillage = this.laplacienZero(fpLaplacian, (float) fpLaplacian.getStatistics().stdDev);
```

Les résultats que nous obtenons alors pour phare et spore sont proches de ceux que nous avons déjà obtenus auparavant :



Seuillage automatique des passages par 0 du Laplacien pour spores et phare

Utilisation du filtre LoG et détection multi-échelles

Dans cette partie du TP, nous expérimenterons l'utilisation d'un lissage gaussien préalable à l'application du Laplacien en interprétant les résultats obtenus lorsque nous changeons les paramètres. Cette implémentation du lissage gaussien avant l'application du Laplacien est nommé l'opérateur « laplacien de gaussien » (Laplacian of Gaussian ou « LoG » en anglais).

Génération de masques pour le LoG

L'inconvénient du calcul de l'image du Laplacien par convolution dans la détection de contours est que le Laplacien est sensible au bruit. La manière de limiter cette sensibilité au bruit est de lisser l'image avec un lissage gaussien avant d'appliquer le Laplacien. Ce procédé est nommé l'opérateur LoG.

Dans l'élaboration de notre plugin ImageJ, nous disposons d'une méthode permettant de calculer le masque de convolution de l'opérateur LoG selon deux paramètres : une taille de masque et un σ .

Les deux paramètres doivent respecter des règles :

- Le masque du LoG doit être de taille impaire afin que le pixel qui bénéficie de l'opération de convolution soit au centre du masque.
- La taille du masque doit être de $6 \times \sigma$ (en arrondissant à l'unité supérieur pour garantir une taille impaire). Un filtre gaussien plus grand pour le même σ est possible mais ne présente pas de bénéfice en plus (et demande plus de puissance de calcul). D'un

autre côté, un masque de taille trop limité ne permettrait pas d'effectuer un lissage correct et donc de limiter le bruit.

Dans le plugin ImageJ, la génération du masque de LoG est implémentée de la manière suivante :

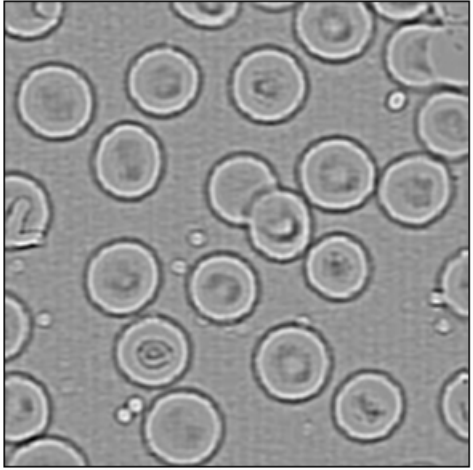
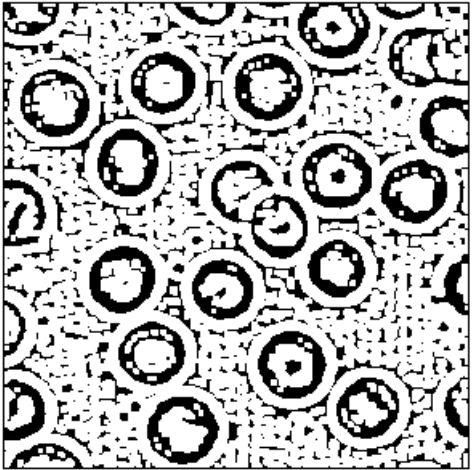
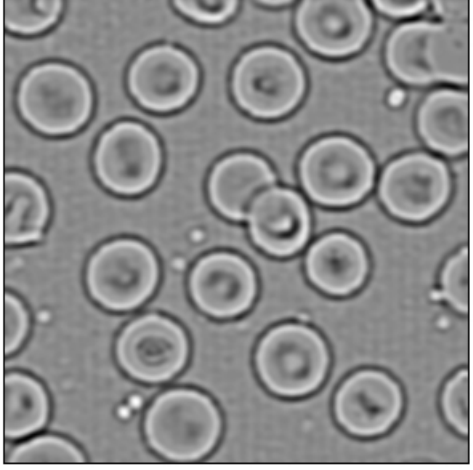
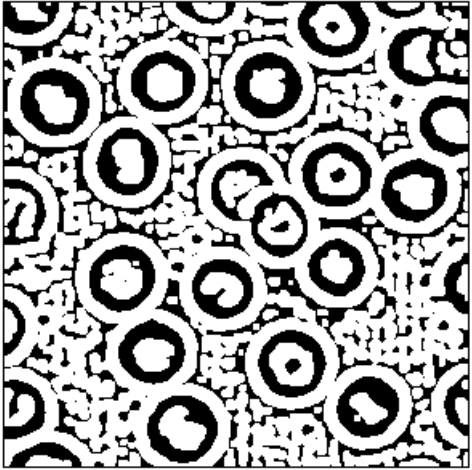
```
float[] masque = null;
// La taille du masque de LoG doit être de 6*sigma
int tailleMasque = Math.round(6 * sigma);
// La taille du masque doit être impaire
if (tailleMasque % 2 == 0) {
    tailleMasque++;
}

if (sigma > 0) {
    masque = masqueLoG(tailleMasque, sigma);
}
```

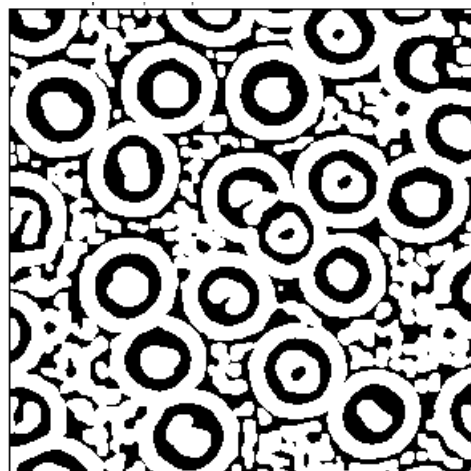
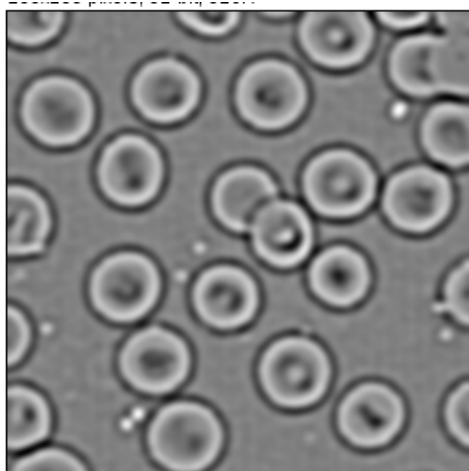
Ensuite, pour calculer le LoG de l'image, il suffit de faire l'opération de convolution de l'image avec le masque LoG trouvé.

```
// Calcul du LoG par convolution avec le masque
Convolver conv = new Convolver();
conv.setNormalize(false);
conv.convolve(fpLaplacian, masque, tailleMasque, tailleMasque);
ImagePlus newImg = new ImagePlus("LoG [Sigma=" + sigma + "]", fpLaplacian);
newImg.show();
```

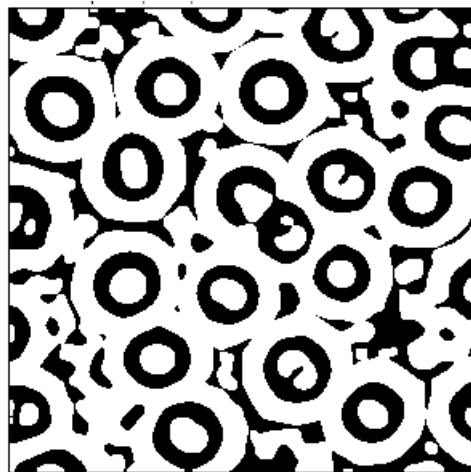
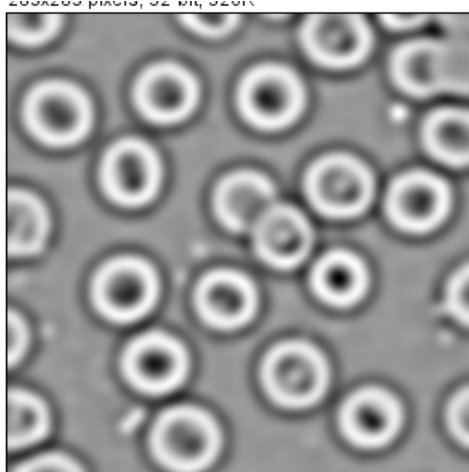
Nous pouvons calculer le LoG en faisant varier le sigma du lissage gaussien et mettre en évidence les passages par 0 du LoG avec la méthode que nous avons vu auparavant. Le tableau suivant montre le sigma choisi, le LoG obtenu et l'image seuillée des passages par 0 pour l'image des spores.

σ	LoG	Passages par 0
1.4		
2.2		

3.0



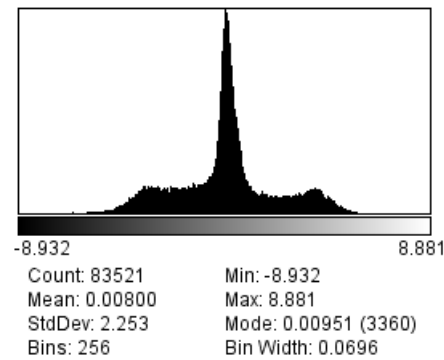
5,0



Nous remarquons alors plusieurs choses :

- Les points contours trouvés forment des contours qui sont bien trop épais, ce qui en fait un résultat non-pertinent, malgré le fait que l'on voit que les contours sont bien situés.
- Plus le σ est élevé, plus le LoG est une représentation lissée de l'image d'origine.

- Lorsque le σ est petit (et donc que le lissage gaussien était limité), la mise en évidence des passages par 0 montre que la détection des points contours est plus sensible au bruit mais permet de détecter les contours des objets moins importants.
- Lorsque le σ est élevé, l'image du LoG est plus floue et la mise en évidence des passages par 0 permet de détecter des points contours des objets principaux de l'image. Cependant, on remarque que ce lissage important a aussi créé des imperfections dans les contours. En effet, certains points contours sont mal placés par rapport à leur emplacement réel et on remarque que certains contours fusionnent pour donner un contour erroné.
- Contrairement à la mise en évidence des passages par 0 du Laplacien que l'on a fait dans la partie précédente, ici, les contours définis sont fermés. Nous pouvons faire l'analogie de ce phénomène avec celui des contours de même altitude dans les cartes de topographie des reliefs.
- L'histogramme d'un des opérateurs LoG donne la répartition dont on peut voir un exemple sur l'image ci-dessous. On remarque que l'histogramme présente une forme de « chapeau mexicain ». Nous pouvons émettre l'hypothèse que les pixels qui ne font pas partie du contour se retrouvent sur la pointe du chapeau. Si on désire mettre un seuil pour la mise en évidence des passages par 0 d'après le LoG, nous pouvons émettre l'hypothèse que trouver un seuil permettant d'occulter les pixels faisant partie de la pointe du chapeau permettrait de trouver des points contours pertinents.



Exemple d'histogramme d'une image d'un LoG pour la photo des spores.

Détection multi-échelles

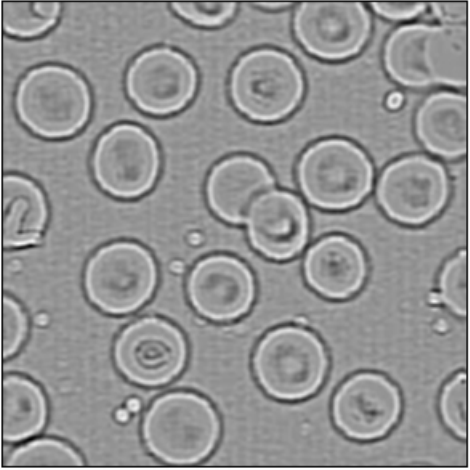
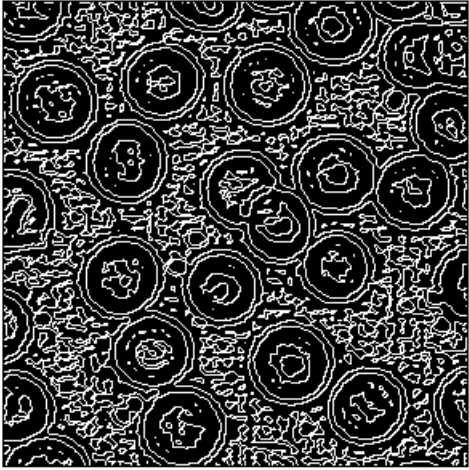
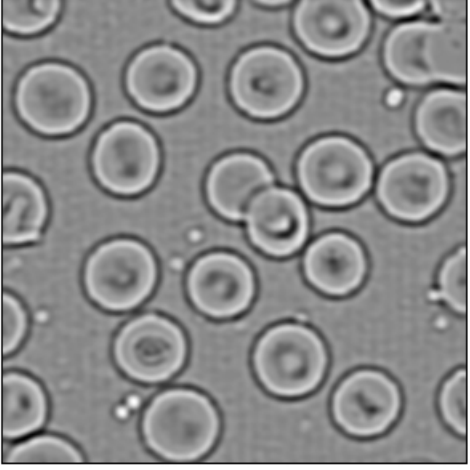
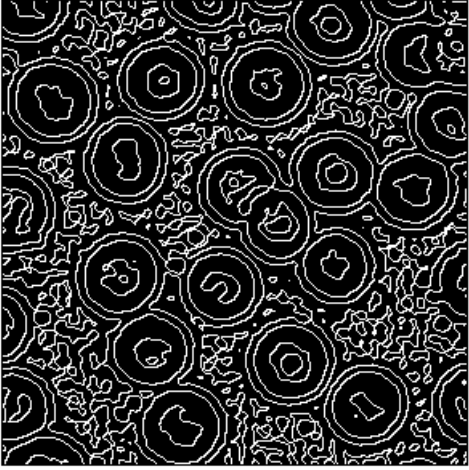
La détection des passages par 0 du Laplacien que nous avons implémentée auparavant a pour inconvénient que les contours générés sont épais. Nous allons donc remplacer le contenu cette détection des points contours par l'autre algorithme ci-dessous et refaire les expériences pour les mêmes valeurs de σ .

```

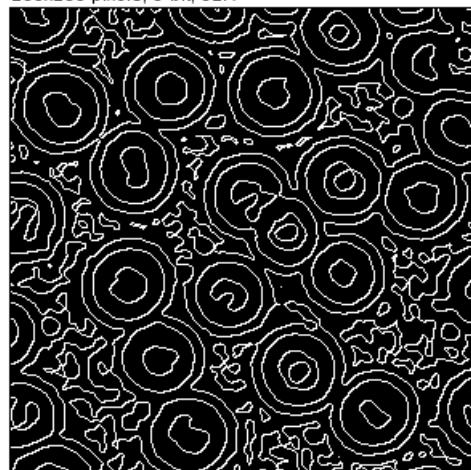
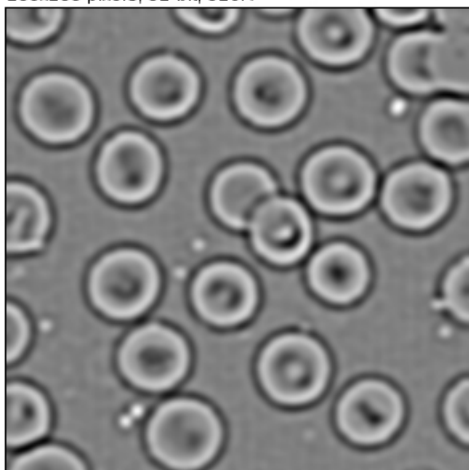
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        // Pour chaque pixel (x,y),
        float pixelC = imLaplacien.getPixelValue(x, y); // Pixel central
        float pixelD = imLaplacien.getPixelValue(x + 1, y); // Pixel droit
        float pixelB = imLaplacien.getPixelValue(x, y + 1); // Pixel bas
        float pixelBD = imLaplacien.getPixelValue(x + 1, y + 1); // Pixel bas droit
        // Détection des transitions Horizontales
        if (pixelC < -seuil && pixelD > seuil) { // Transition horizontale -|+
            imZeros.set(x, y, 255);
        }
        if (pixelC > seuil && pixelD < -seuil) { // Transition horizontale +|-
            imZeros.set(x + 1, y, 255);
        }
        // Détection des transitions Verticales
        if (pixelC < -seuil && pixelB > seuil) { // Transition verticale -|+
            imZeros.set(x, y, 255);
        }
        if (pixelC > seuil && pixelB < -seuil) { // Transition verticale +|-
            imZeros.set(x, y + 1, 255);
        }
        // Détection des transitions Diagonales
        if (pixelC < -seuil && pixelBD > seuil) { // Transition diagonale -|+
            imZeros.set(x, y, 255);
        }
        if (pixelC > seuil && pixelBD < -seuil) { // Transition diagonale +|-
            imZeros.set(x + 1, y + 1, 255);
        }
    }
}

```

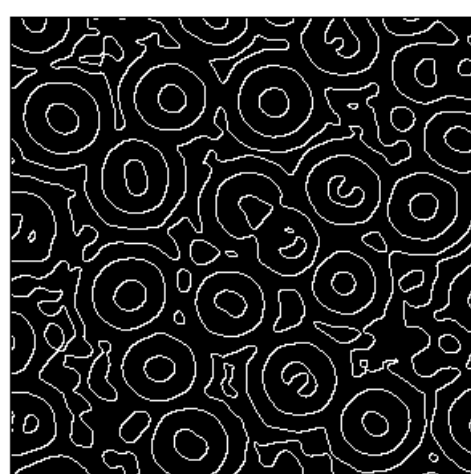
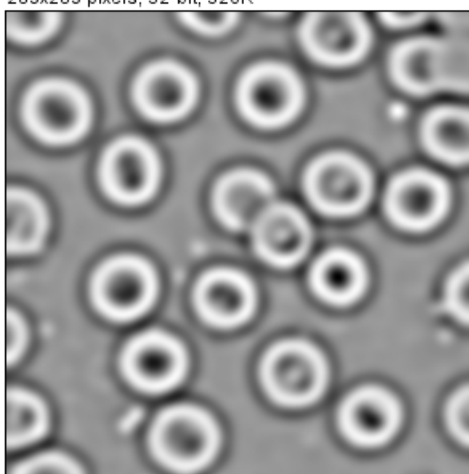
Le tableau suivant nous fourni les résultats :

σ	LoG	Passages par 0
1.4		
2.2		

3.0



5,0



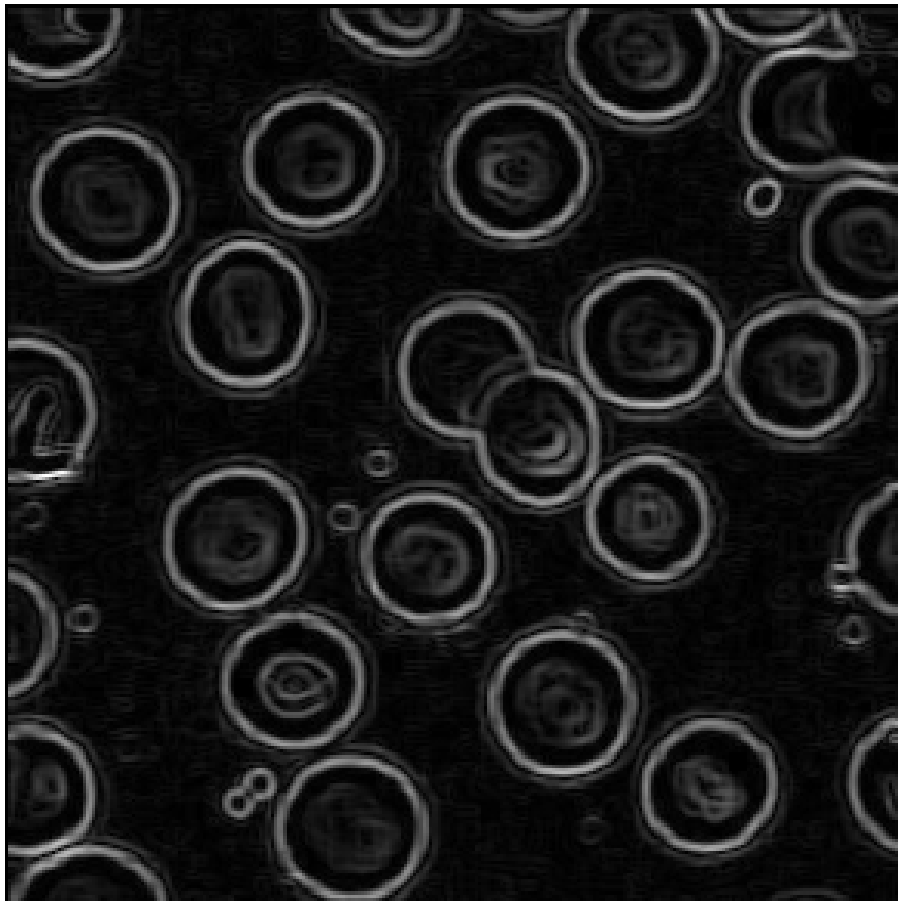
Nous remarquons tout de suite que les contours sont largement moins épais qu'auparavant, ce qui donne un résultat plus pertinent. De plus, les contours formés par les points contours n'ont qu'un seul pixel d'épaisseur.

Utilisation de la norme du gradient

Nous pouvons utiliser la norme du gradient de l'image pour seuiller les passages par 0 du LoG. Pour ce faire, nous devons effectuer plusieurs étapes :

- Calculer l'image (binaire) des passages par 0 du LoG de l'image des spores. Dans notre expérience, nous choisirons l'image des spores avec un LoG où $\sigma=1.4$ et un seuil $S=0$ pour la détection des passages par 0.
- Calculer la norme du gradient de l'image des spores
- Masquer l'image de la norme du gradient des spores avec celle des passages par 0 obtenu lors de la première étape (on effectue une opération AND).
- Appliquer un seuil sur l'image résultante (ici, ce sera un seuillage simple)

Dans les opérations effectuées, nous obtenons les images suivantes :



Norme du gradient obtenu par convolution des masques de Sobel

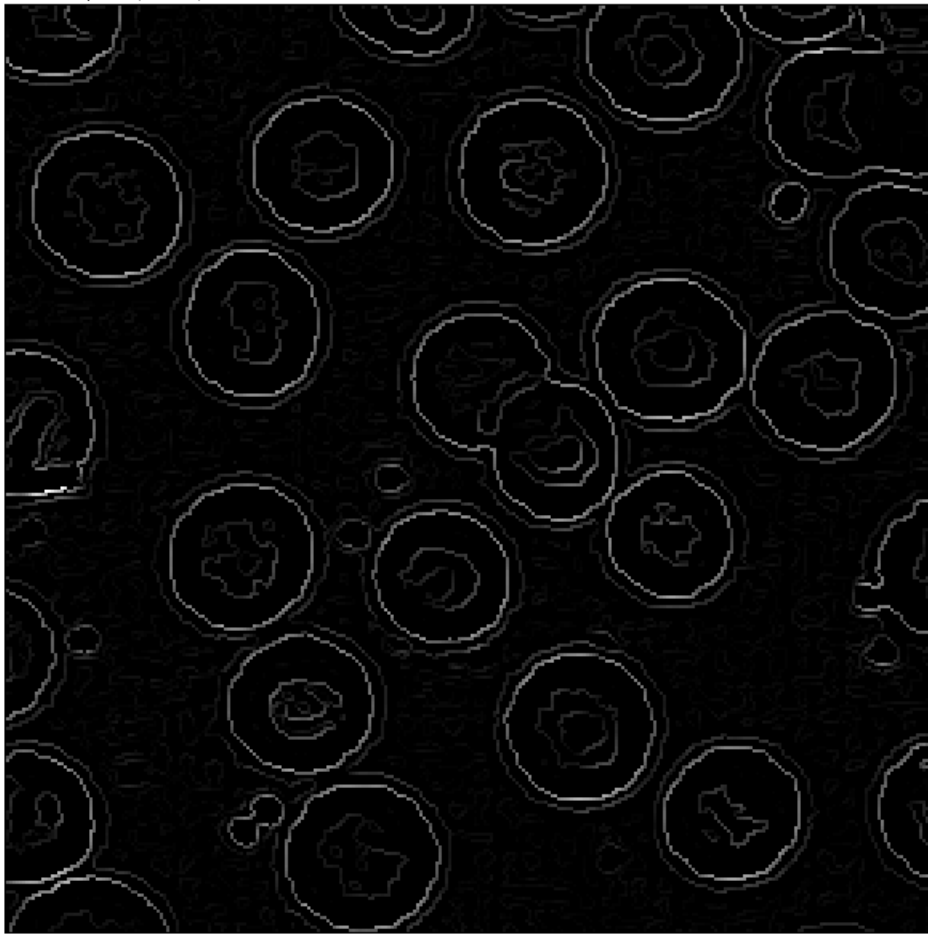
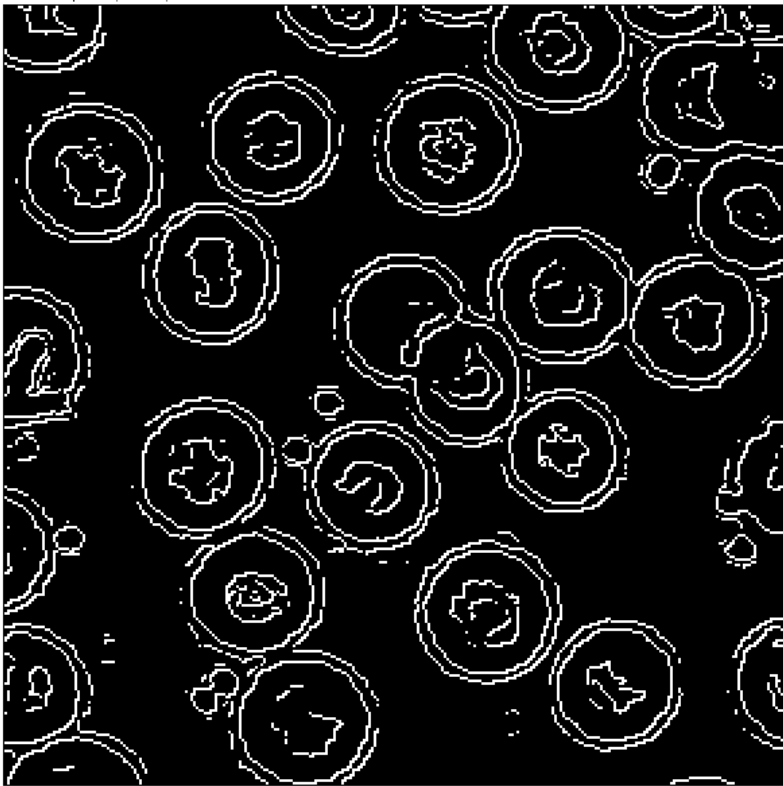
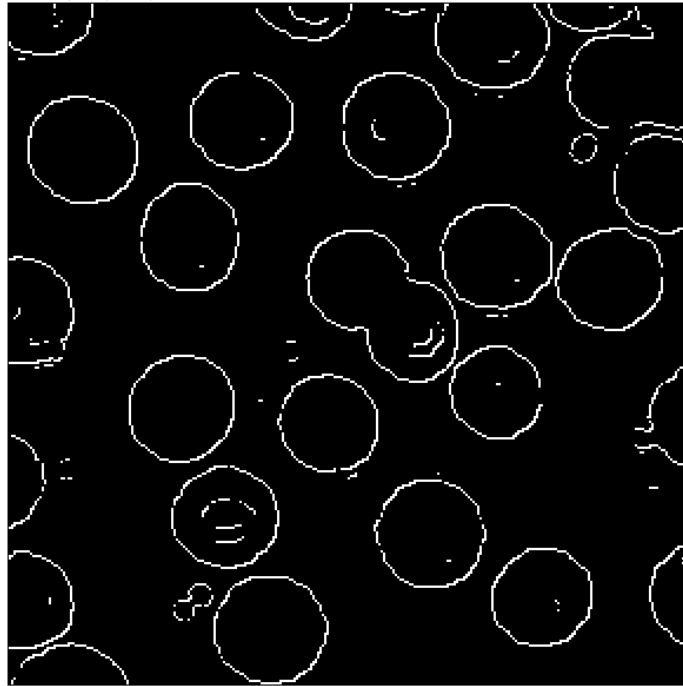


Image de la norme du gradient masquée par l'image des passages par 0 du LoG

Nous pouvons alors appliquer des seuils différents à l'image de la norme du gradient masquée par l'image des passages par 0 du LoG :

Seuil	Résultat
8.66	

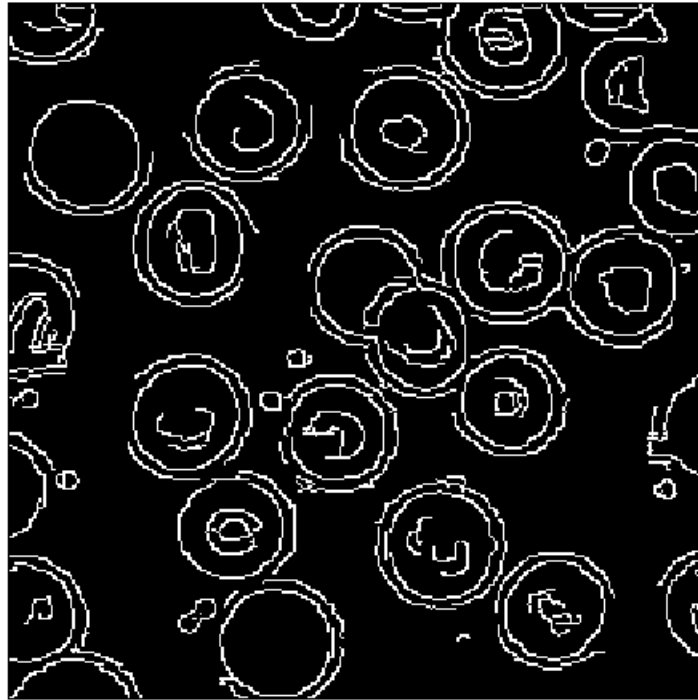
28.61



L'intérêt du seuillage opéré ici est que l'on peut encore choisir de détecter des contours pour des objets faisant partie du détail ou pour les objets de plus grande importance.

Le désavantage du seuillage par la norme du gradient est le fait que les contours ne sont désormais plus fermés.

Nous pouvons comparer ces deux résultats au résultat des points contours obtenus grâce à la méthode de Canny (suppression des non maxima locaux) avec un seuillage par hystérésis de seuil bas et haut respectivement à 8 et 20.



Méthode de Canny avec seuillage par hystérésis de seuils 8 (bas) et 22 (haut)

En comparant les images obtenues avec le Laplacien et l'image obtenue grâce à la méthode de Canny, nous pouvons voir plusieurs choses :

- Les points contours trouvés par la méthode de Canny donnent des contours mieux situés que ceux obtenus grâce au LoG, du fait que la méthode de Canny est moins sensible au bruit que le Laplacien. En effet, comme il faut appliquer un lissage plus fort sur le Laplacien pour réduire le bruit, les contours sont un peu moins bien situés.
- Il y a des points contours trouvés par le LoG qui n'ont pas été trouvés par la méthode de Canny (et vice versa). Ce qui nous montre qu'il n'y a pas de méthode parfaite.
- Les deux approches nécessitent d'ajuster des paramètres, ce qui peut rajouter de la complexité.

Annexe : source du plugin

Laplacien

Méthode principale « run »

```
// Méthode principale du plugin
public void run(ImageProcessor ip) {

    // Affichage de la fenêtre de configuration
    if (!showDialog())
        return;

    // Titre et extension de l'image source
    String titre = imp.getTitle();
    String extension = "";
    int index = titre.lastIndexOf('.');
    if (index > 0)
        extension = titre.substring(index);
    else
        index = titre.length();
    titre = titre.substring(0, index);

    // Génération d'un masque LoG de taille impaire (exercice 3)
    /* à compléter */
    float[] masque = null;
    // La taille du masque de LoG doit être de 6*sigma
    int tailleMasque = Math.round(6 * sigma);
    // La taille du masque doit être impaire
    if (tailleMasque % 2 == 0) {
        tailleMasque++;
    }

    if (sigma > 0) {
        masque = masqueLoG(tailleMasque, sigma);
    }

    // Calcul et représentation du Laplacien (exercice 1, puis à modifier dans le 3)
    FloatProcessor fplLaplacian = (FloatProcessor) (ip.duplicate().convertToFloat());
    /* à compléter */
    if (sigma <= 0) {
        // Calcul du laplacien de l'image par convolution
        fplLaplacian.convolve(MASQUES_LAPLACIENS3x3[filtre], 3, 3);
    }
}
```

```

    // Affichage du résultat
    ImagePlus newImg = new ImagePlus("Résultat du lissage par masque laplacien", fpLaplacian);
    newImg.show();
} else {
    // Calcul du LoG par convolution avec le masque
    Convolver conv = new Convolver();
    conv.setNormalize(false);
    conv.convolve(fpLaplacian, masque, tailleMasque, tailleMasque);
    ImagePlus newImg = new ImagePlus("LoG [Sigma=" + sigma + "]", fpLaplacian);
    newImg.show();
}

// Détection et affichage des passages par 0 du Laplacien par seuillage du
// Laplacien (exercice 2)
if (seuillageZeroCross) {
    /* à compléter */
    ByteProcessor seuillage = null;
    String title = "Seuillage [S=";
    // SI [recherche automatique du seuil]
    if (seuilZeroCrossAuto) {
        // Le seuil calculé automatiquement est l'écart-type du Laplacien
        seuillage = this.laplacienZero(fpLaplacian, (float) fpLaplacian.getStatistics().stdDev);
        title += fpLaplacian.getStatistics().stdDev + "];";
    }
    // SINON
    else {
        seuillage = this.laplacienZero(fpLaplacian, seuilZeroCross);
        title += seuilZeroCross + "];";
    }
    ImagePlus imgSeuil = new ImagePlus(title, seuillage);
    imgSeuil.show();
}
}

```

Méthode laplacienZero (implémentation finale)

```

public ByteProcessor laplacienZero(ImageProcessor imLaplacien, Float seuil) {

    int width = imLaplacien.getWidth();
    int height = imLaplacien.getHeight();

    // Image binaire résultat des points contours après seuillage
    ByteProcessor imZeros = new ByteProcessor(width, height);

    /* à compléter */
    // POUR CHAQUE pixel

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            // Pour chaque pixel (x,y),
            float pixelC = imLaplacien.getPixelValue(x, y); // Pixel central
            float pixelD = imLaplacien.getPixelValue(x + 1, y); // Pixel droit
            float pixelB = imLaplacien.getPixelValue(x, y + 1); // Pixel bas
            float pixelBD = imLaplacien.getPixelValue(x + 1, y + 1); // Pixel bas droit
            // Détection des transitions Horizontales
            if (pixelC < -seuil && pixelD > seuil) { // Transition horizontale -|+
                imZeros.set(x, y, 255);
            }
            if (pixelC > seuil && pixelD < -seuil) { // Transition horizontale +|-
                imZeros.set(x + 1, y, 255);
            }
            // Détection des transitions Verticales
            if (pixelC < -seuil && pixelB > seuil) { // Transition verticale -|+
                imZeros.set(x, y, 255);
            }
            if (pixelC > seuil && pixelB < -seuil) { // Transition verticale +|-
                imZeros.set(x, y + 1, 255);
            }
            // Détection des transitions Diagonales
            if (pixelC < -seuil && pixelBD > seuil) { // Transition diagonale -|+
                imZeros.set(x, y, 255);
            }
            if (pixelC > seuil && pixelBD < -seuil) { // Transition diagonale +|-
                imZeros.set(x + 1, y + 1, 255);
            }
        }
    }

    return imZeros;
}

```

Méthode laplacienZero (implémentation de la partie 2)

```
public ByteProcessor laplacienZero(ImageProcessor imLaplacien, Float seuil) {

    int width = imLaplacien.getWidth();
    int height = imLaplacien.getHeight();

    // Image binaire résultat des points contours après seuillage
    ByteProcessor imZeros = new ByteProcessor(width, height);

    /* à compléter */
    // POUR CHAQUE pixel
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            float pixel = imLaplacien.getPixelValue(i, j);

            float minimal = Float.MAX_VALUE; // représente le voisin minimal
            float maximal = Float.MIN_VALUE; // représente le voisin maximal
            // Parcourir le voisinage (3x3) du pixel
            for (int x = i - 1; x <= i + 1; x++) {
                if (x < 0 || x >= width) {
                    continue;
                }

                for (int y = j - 1; y <= j + 1; y++) {
                    if (y < 0 || y >= height) {
                        continue;
                    }
                    float voisin = imLaplacien.getPixelValue(x, y);
                    minimal = minimal > voisin ? voisin : minimal;
                    maximal = maximal < voisin ? voisin : maximal;
                }
            }

            // Imposition du seuil
            if (minimal < -seuil && maximal > seuil) {
                imZeros.set(i, j, 255);

            } else {
                //
                IJ.log("mini = " + minimal + ", maxi = " + maximal + ", pixel=" + pixel);
                imZeros.set(i, j, 0);
            }
        }
    }

    return imZeros;
}
```