



Université  
de Lille  
1 SCIENCES  
ET TECHNOLOGIES

Université de Lille  
**MASTER 1 INFORMATIQUE**  
Deuxième semestre



# Traitement d'image

TP n°09

**BARCHID Sami**

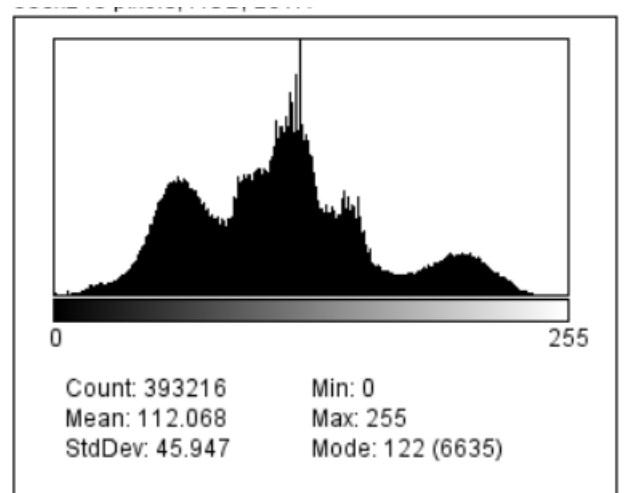
**CARTON Floriane**

# Introduction

---

La matière vue dans ce TP traite de la détection de contours. L'objectif est d'apprendre les notions de contour, points contours et des approches par dérivées premières dans un travail pratique qui consiste à détecter les contours d'une image grâce au seuillage local par hystérésis.

Les manipulations se feront grâce au logiciel « ImageJ » et l'image qui sera le sujet de la totalité des expériences du TP est « **lighthouse** », l'image codée sur 8 bits (niveaux de gris 0 à 255) suivante :



**IMAGE – lighthouse + histogramme**

Ce rapport de TP est divisé en trois parties, définissant les trois étapes dans la détection du contour :

- **Seuillage de la norme d'un gradient** : expérimentations et examination de l'adaptation d'un seuillage global de la forme d'un gradient à la détection des contours de l'image.
- **Détection des maxima locaux de la norme d'un gradient** : représentation des maxima locaux de la norme du gradient calculée précédemment afin d'interpréter les résultats obtenus.
- **Seuillage des maxima locaux par hystérésis** : étape de seuillage local par hystérésis afin d'aboutir à une détection de contours et interprétation des résultats.

# Seuillage de la norme d'un gradient

---

Dans cette partie, l'objectif est de calculer la norme du gradient de l'image pour en interpréter les résultats.

## Mode opératoire

La norme du vecteur gradient de  $f$ , la fonction image du phare, est calculée grâce à la norme des gradients par masque de Sobel.

En effet, on cherche alors à calculer une approximation des dérivées partielles selon l'axe horizontal et vertical par convolution avec les masques de Sobel suivants :

$$H_x^s = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ et } H_y^s = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

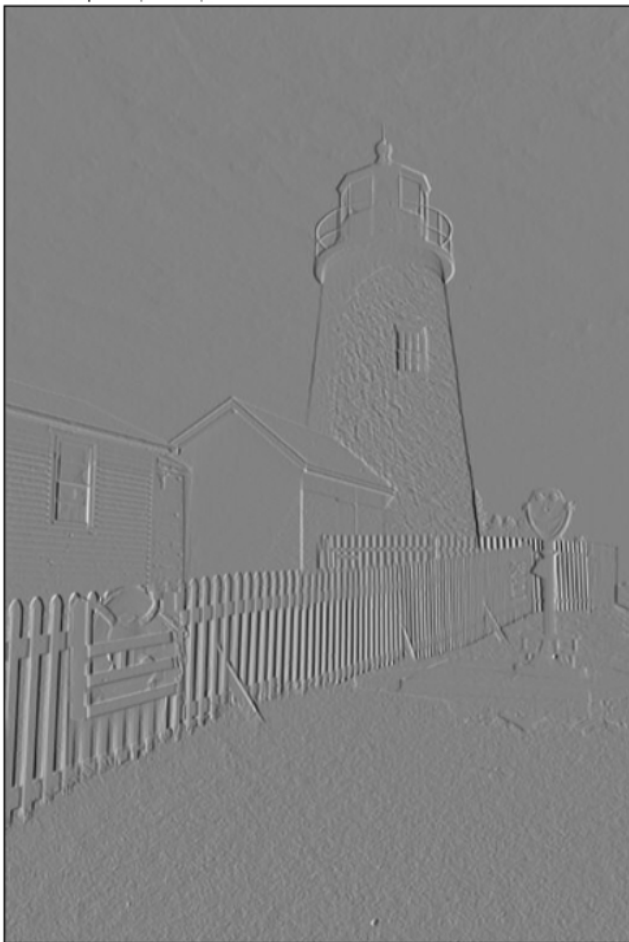
On obtient la formule pour les vecteurs gradients suivante :

$$\vec{\nabla} f(x, y) = \begin{pmatrix} (I * H_x^s)(x, y) \\ (I * H_y^s)(x, y) \end{pmatrix}$$

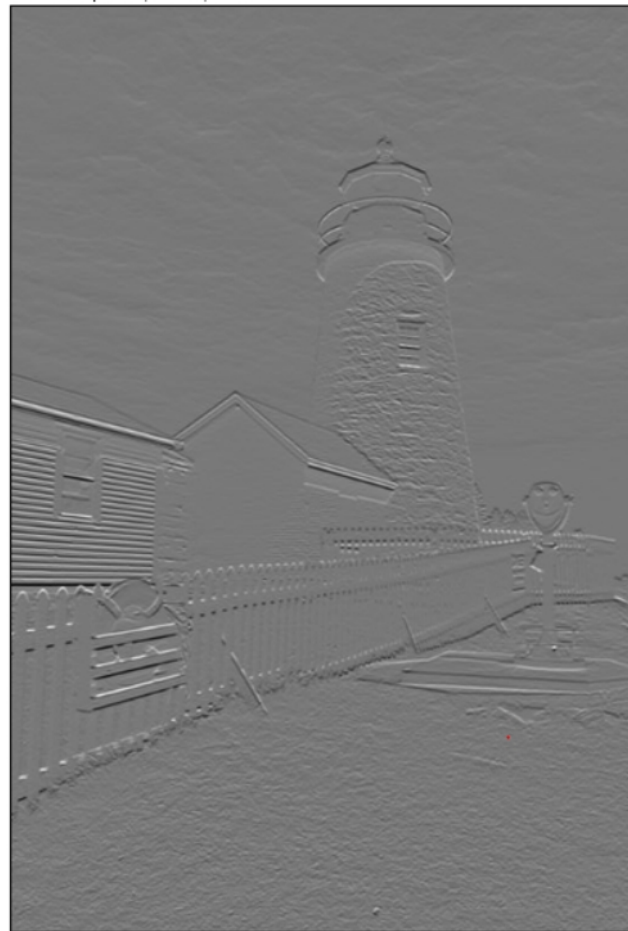
- où  $\vec{\nabla} f(x, y)$  est le vecteur gradient aux coordonnées (x,y).
- $(I * H_x^s)(x, y)$  est l'approximation de la dérivée partielle en x par masque de convolution de Sobel.

- $(I * H_y^s)(x, y)$  est l'approximation de la dérivée partielle en y par masque de convolution de Sobel.

Le calcul de ces dérivées partielles donnent les résultats suivants :



$$(I * H_x^s)(x, y)$$



$$(I * H_y^s)(x, y)$$

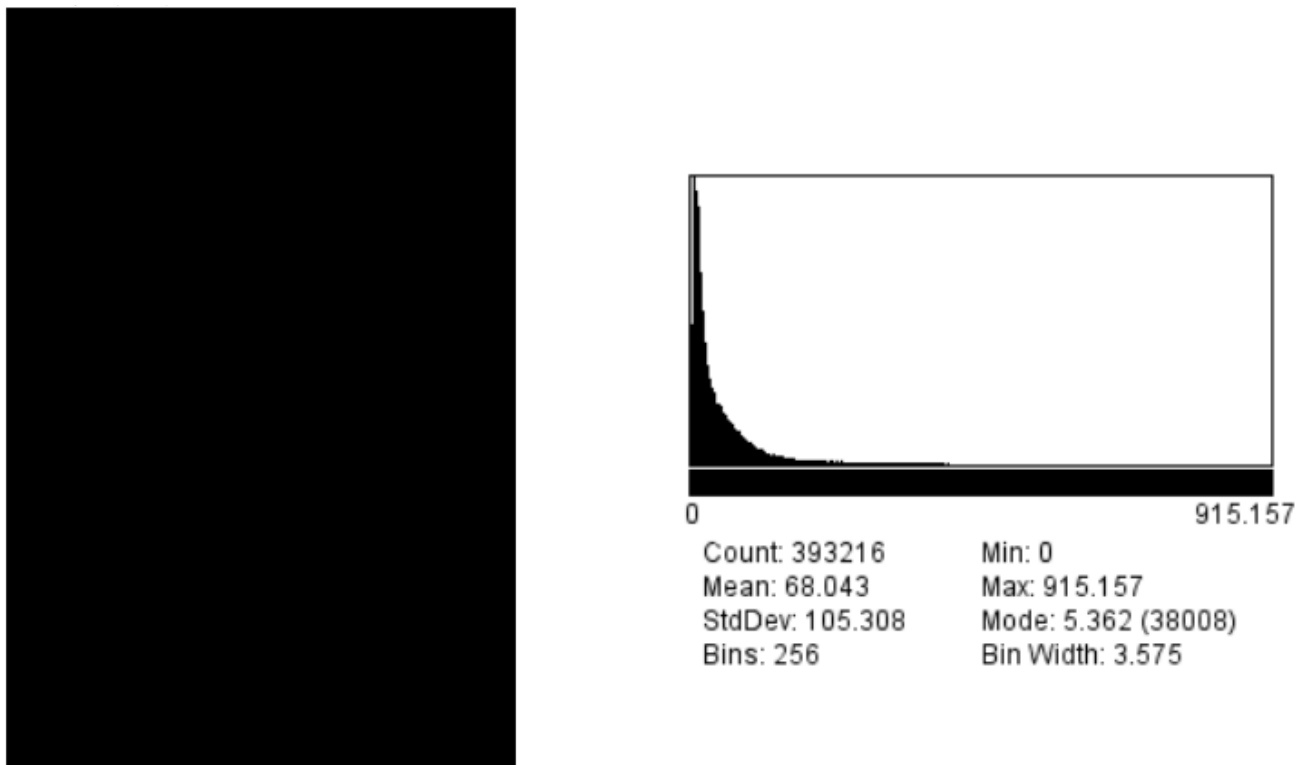
On applique alors la formule de la norme du gradient suivante :

$$\|\vec{\nabla} f(x, y)\| = \sqrt{\left(\frac{\partial f}{\partial x}(x, y)\right)^2 + \left(\frac{\partial f}{\partial y}(x, y)\right)^2}$$

Il est important de noter que, lors du calcul des dérivées partielles et de la norme du gradient, les images doivent être codées sous 32 bits. La raison derrière est que le calcul des dérivées partielles doivent permettre des valeurs positives comme négatives pour obtenir un résultat ayant du sens afin de ne pas tronquer les valeurs qui ne sont pas comprise dans l'intervalle [0,255].

## Application de la norme du gradient

Les manipulations décrites ci-dessus ont été réalisées dans le programme macro ImageJ situé en annexe 1. Le résultat du calcul de la norme du gradient du phare donné par la macro a été le suivant :



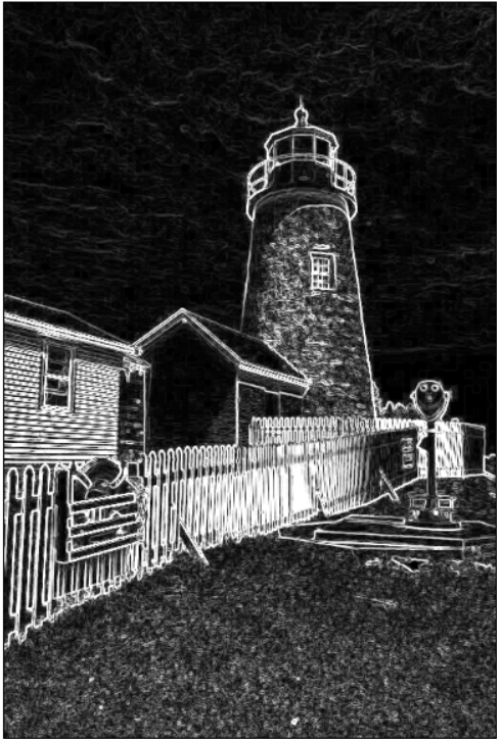

**Norme du gradient de Sobel pour le phare + histogramme**

On remarque plusieurs choses :

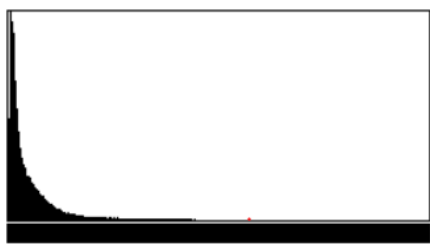
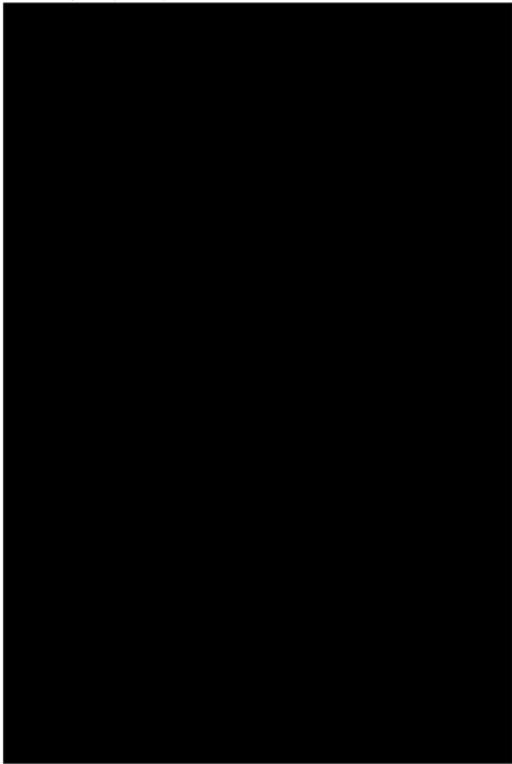
- La valeur minimale de la norme du gradient est 0

- La valeur maximale de la norme du gradient est de 915.157
  - Ces valeurs (en dehors de 0-255) sont dues au calcul sur 32 bits ainsi qu'à l'équation de la norme du gradient.
- La représentation visuelle ici n'est pas représentative puisqu'elle dépend de l'interprétation qu'on lui donne.

L'autre travail que nous pouvons effectuer est de comparer les manipulations faites dans notre macro avec la macro « Find edges » par défaut dans le logiciel ImageJ qui permet également de calculer la norme du gradient d'une image. Le tableau suivant permet de comparer les résultats pour find edges et nos résultats :

Résultat	Visuel	Histogramme
« Find edges »		 <p>Count: 393216      Min: 0  Mean: 68.043      Max: 915.157  StdDev: 105.308      Mode: 5.362 (38008)  Bins: 256      Bin Width: 3.575</p>

Calcul de la norme du gradient de Sobel personnel



Count: 393216	Min: 0
Mean: 68.043	Max: 915.157
StdDev: 105.308	Mode: 5.362 (38008)
Bins: 256	Bin Width: 3.575



Après observation des histogrammes, on remarque aisément que les minimum, maximum, écart-type, etc sont égaux. Ce qui nous laisse déjà émettre l'hypothèse que, malgré la différence visuelle, les deux résultats sont les mêmes.

Afin de le prouver, la soustraction des deux images a été opérées et il s'est avéré que le calcul de la norme du gradient de Sobel était bien le même dans les deux cas.

La transformation à opérer sur l'image issue de notre calcul doit être de réajuster le contraste de l'image de la norme du gradient par rapport à la valeur minimale et maximale de l'image, à savoir entre 0 et 915.157. Ce réajustement nous donne le résultat suivant :



**Norme du gradient de Sobel calculée, réajustée et codée en 8 bits**

## Seuillage global par la norme du gradient

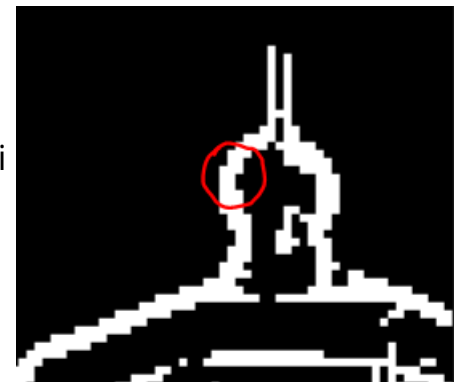
Il est désormais possible d'expérimenter un seuillage global de la norme du gradient afin de détecter les contours de l'image. Pour ce faire, nous appliquons un seuil pour binariser l'image de la norme du gradient de Sobel codée en 8 bits :



**IMAGE – norme du gradient de Sobel calculée, réajustée et codée en 8 bits sur laquelle on a appliqué un seuil de 14.19**

Bien que la plupart des contours aient été retrouvés, il y a plusieurs critiques à émettre :

- Les contours trouvés ne sont pas des contours correctement délimités (les contours ont une épaisseur plus grande que 1 pixel, donnant). Ce qui a pour conséquence d'augmenter la quantité d'information à retenir pour une image et de créer des contours discontinus.



- Les contours qui ne correspondent pas à de fortes variations de l'intensité ne sont pas pris en compte.
- Des endroits à fortes variations d'intensité qui ne sont pas des contours sont comptabilisés comme tel :



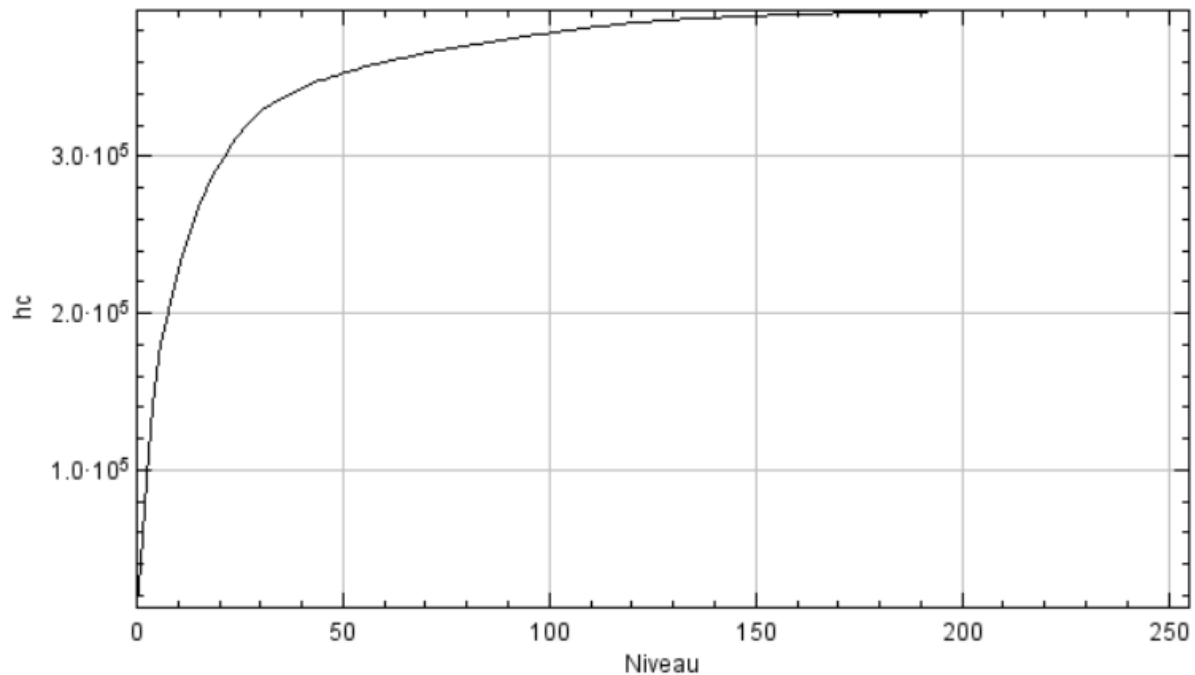
C'est pourquoi il ne peut pas y avoir de seuil global permettant de mettre en évidence les pixels des contours de manière satisfaisante.

## Détermination du seuil par histogramme cumulé

Nous allons maintenant expérimenter la possibilité de trouver un seuil global de manière semi-automatique par l'histogramme cumulé de la norme du gradient.

L'idée est, à partir de l'histogramme cumulé de la norme du gradient, de sélectionner un pourcentage restreint de pixels contours les plus significatifs.

L'histogramme cumulé de la norme du gradient pour l'image du phare est le suivant :



Le mode opératoire ici est de vérifier, pour chaque niveau de gris, si le nombre de pixels de la norme du gradient pour le niveau de gris trouvé est à 80 % du nombre de pixels total (80 % de 393216). De cette manière, tous les pixels ayant le niveau de gris trouvé ou moins seront seuillé en noir et les pixels ayant un niveau de gris plus haut seront seuillé en blanc.

Ce seuillage global permettrait de diminuer de manière plus ou moins automatique les pixels de la norme du gradient qui ne sont pas des contours effectifs.

Pour le niveau de gris 25, on obtient un nombre de 314872 pixels ayant un niveau de gris  $\leq 25$ , à savoir quasiment 80 % de 393216 ( $0.8 \times 393216 = 314572.8$ ).

L'image seuillée par la technique de l'histogramme cumulée est la suivante :



En comparant avec l'image seuillée de la question précédente, on remarque que nous acceptons plus de contours dans des zones de faibles variations d'intensité. Cependant, on remarque aussi que nous avons une plus grande quantité de pixels n'appartenant pas à des contours qui sont comptés comme tel.

Au final, on peut conclure sur le fait les résultats apportés par le seuillage global par la norme du gradient ne donnent pas des résultats assez satisfaisant pour être identifiés comme des contours : les erreurs sont trop fréquentes.

# Détection des maxima locaux de la norme d'un gradient

---

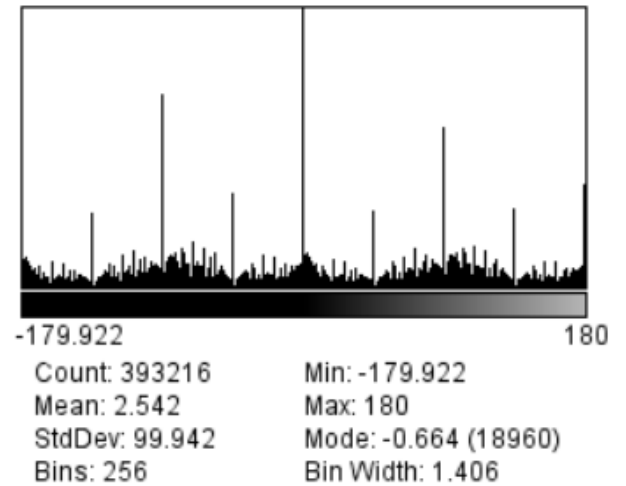
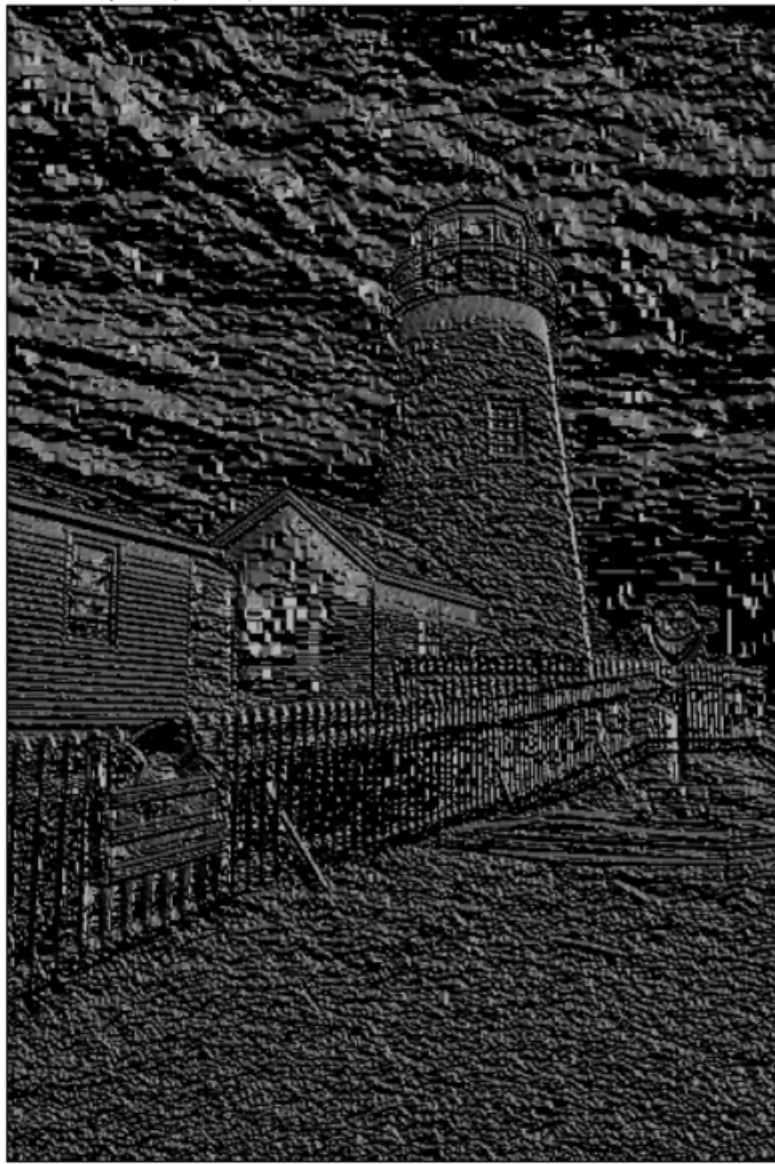
Le but de cette partie du TP est de détecter les maxima locaux de l'image de la norme du gradient calculée dans l'exercice précédent. Il faut, pour ce faire, déterminer les voisins directs  $P_1$  et  $P_2$  qui se trouvent dans la direction du gradient et appliquer la règle suivante :

Si  $\|\vec{\nabla} f(P)\| \geq \|\vec{\nabla} f(P_1)\|$  et  $\|\vec{\nabla} f(P)\| > \|\vec{\nabla} f(P_2)\|$ ,  
**alors  $P$  est un maximum local.**

## Calcul de la direction du gradient

La première chose à faire est de calculer la direction du gradient estimé par les masques de Sobel. L'annexe 2 permet de visualiser la macro ImageJ permettant de calculer cette direction du gradient.

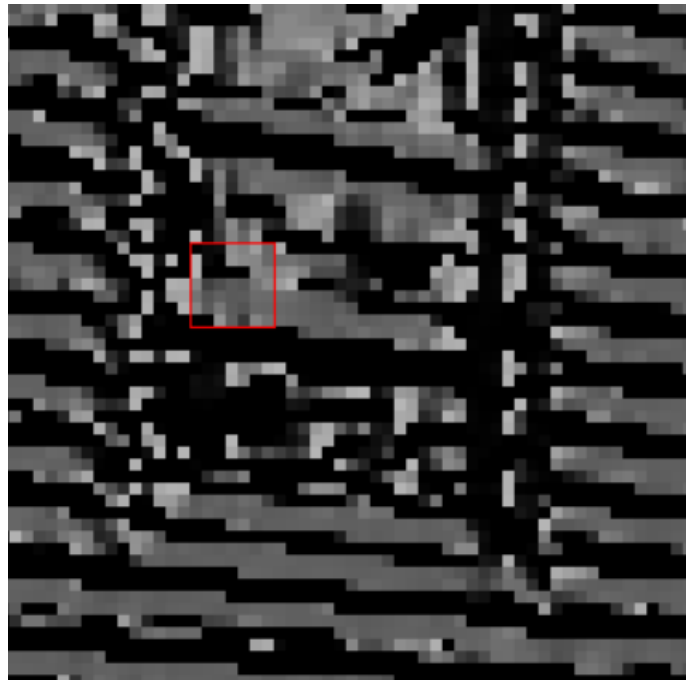
Le résultat du calcul obtenu par la macro créée sur ImageJ est le suivant :



### Direction du gradient du phare + histogramme

- La direction du gradient est toujours perpendiculaire au contour
- La direction minimale du gradient est -179.922 et la direction maximale est 180

Nous pouvons nous assurer de la perpendicularité de la direction du gradient en visualisant les pixels de l'image de la direction du gradient sur un carreau de la fenêtre positionné à l'horizontale :



s	43	44	45	46	47	48	49
399	-179.35	7.62	12.26	142.13	129.09	157.11	101.31
400	179.69	7.72	-5.39	146.31	135.00	133.64	103.17
401	179.22	0.00	-15.46	-90.00	-135.00	132.40	125.91
402	133.34	63.61	77.28	90.00	84.96	115.14	136.85
403	114.48	84.46	96.27	90.00	91.04	106.39	105.73
404	131.42	90.00	118.16	90.64	82.61	102.94	101.31
405	-105.14	-29.05	146.59	93.81	55.12	131.82	112.75

On remarque ici que le contour du barreau de la fenêtre est complètement horizontal. Or, la direction du gradient indique un angle de 90 %, parfaitement perpendiculaire au barreau de la fenêtre.

## Détection des maxima locaux

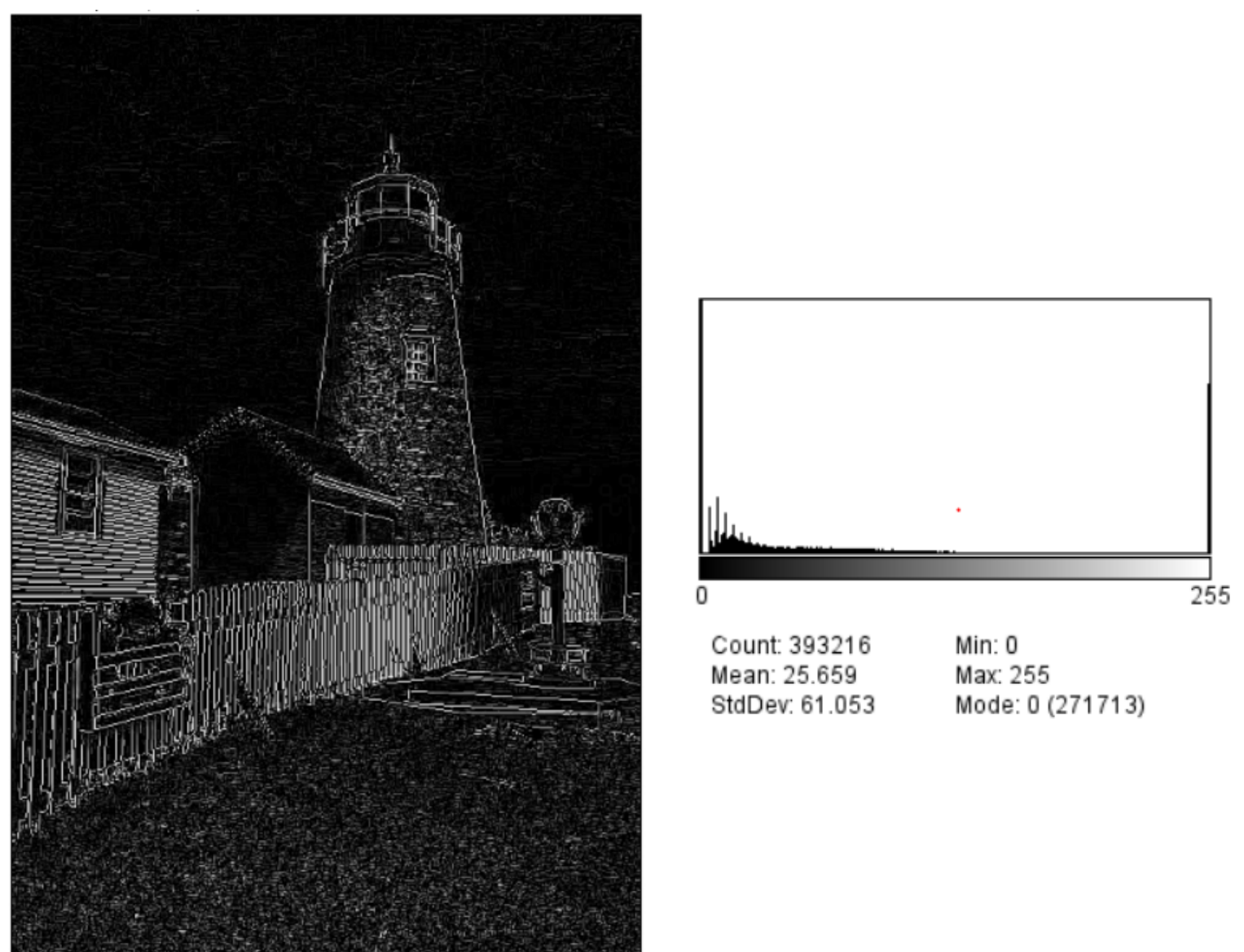
L'annexe 3 (fin de document) montre les instructions macro ImageJ nécessaire à la détection des maxima locaux dans l'image de la norme du gradient grâce à l'utilisation de la norme du gradient.



Le principe était d'appliquer la règle décrite plus tôt tout en définissant, à l'aide de la direction du gradient,  $P_1$  et  $P_2$ , les voisins de  $P$ .

Il est à noter que les voisins ont été déterminés par discrétisation de la direction (= on arrondit la direction du gradient au multiple de  $45^\circ$  le plus proche et  $P_1$  et  $P_2$  sont situés aux opposés de  $P$ ).

On obtient alors l'image des maxima locaux suivante :



**Maxima locaux de la norme du gradient**

Lorsque l'on compare l'image des maxima locaux avec l'image de la norme du gradient obtenue initialement, on remarque qu'une grande partie des pixels « parasites » ont été réduits à 0 grâce à la détection des maxima locaux.

En effet, grâce à la détection des maxima locaux, les pixels à niveau de gris  $> 0$  restants sont des points contours candidats satisfaisant. On observe également que les problèmes d'épaisseurs de contours et de pixels non-contours erronés sont réduits.

# Seuillage des maxima locaux par hystérésis

---

Dans cette partie du TP, nous expérimenterons et analyserons la dernière partie de la détection de contours qui suit les opérations faites auparavant : l'étape de seuillage des maxima locaux par hystérésis.

Le but est de fixer deux seuils au lieu d'un (afin d'améliorer la flexibilité du seuillage).

Les opérations ont été faites grâce à un plugin d'ImageJ en Java. Afin de comprendre comment fonctionne l'algorithme, des commentaires ont été ajoutés sur les instructions du plugin.

Le seuillage des maxima locaux par hystérésis prend en entrée l'image des maxima locaux de la norme du gradient, un seuil bas et un seuil haut afin d'appliquer la règle suivante pour chaque pixel :

- Si le niveau de gris du pixel  $<$  Seuil bas, le pixel n'est pas un contour
- Si le niveau de gris du pixel  $>$  Seuil haut, le pixel est un contour
- Si le niveau de gris du pixel est entre les deux seuils, le pixel est un contour uniquement s'il est situé à côté d'un pixel contour

```

/**
 *
 * @param imNormeG image des maxima locaux (càd des candidats contours)
 * @param seuilBas seuil bas choisi pour refuser des maxima locaux
 * @param seuilHaut seuil haut choisi pour refuser des maxima locaux
 * @return l'image 8bits qui représente l'image seuillée qui affiche les contours.
 */
public ByteProcessor hystIter(ImageProcessor imNormeG, int seuilBas, int seuilHaut) {
    // Récupérer les dimensions de l'image
    int width = imNormeG.getWidth();
    int height = imNormeG.getHeight();

    // Résultat final de la méthode
    ByteProcessor maxLoc = new ByteProcessor(width,height);

    // Liste qui va contenir les coordonnées des pixels qui seront directement pris dans le contour
    List<int[]> highpixels = new ArrayList<int[]>();

    // POUR CHAQUE [pixel]
    for (int y=0; y<height; y++) {
        for (int x=0; x<width; x++) {
            // g est le maximal local récupéré au pixel (x,y)
            int g = imNormeG.getPixel(x, y)&0xFF;

            // SI g < Seuil bas, alors ce n'est pas un contour
            if (g<seuilBas) continue;

            // SI g > seuil haut, alors c'est d'office un contour
            if (g>seuilHaut) {
                maxLoc.set(x,y,255);
                // ajouter dans la liste des points contours déjà reconnus
                highpixels.add(new int[]{x,y});
                continue;
            }

            // SI seuilBas <= g <= seuilHaut, alors le point est contour
            // UNIQUEMENT SI le point est connecté à un autre pixel déjà contour

```

```

        maxLoc.set(x,y,128);
    }
}

int[] dx8 = new int[] {-1, 0, 1,-1, 1,-1, 0, 1};
int[] dy8 = new int[] {-1,-1,-1, 0, 0, 1, 1, 1};

// Tant que je n'ai pas trouvé tous les points contours
while(!highpixels.isEmpty()) {

    // Liste des points contours qui sera utilisée pour trouver les points contours de l'itération
    List<int[]> newhighpixels = new ArrayList<int[]>();

    // POUR CHAQUE pixel du contour
    for(int[] pixel : highpixels) {
        int x=pixel[0], y=pixel[1];

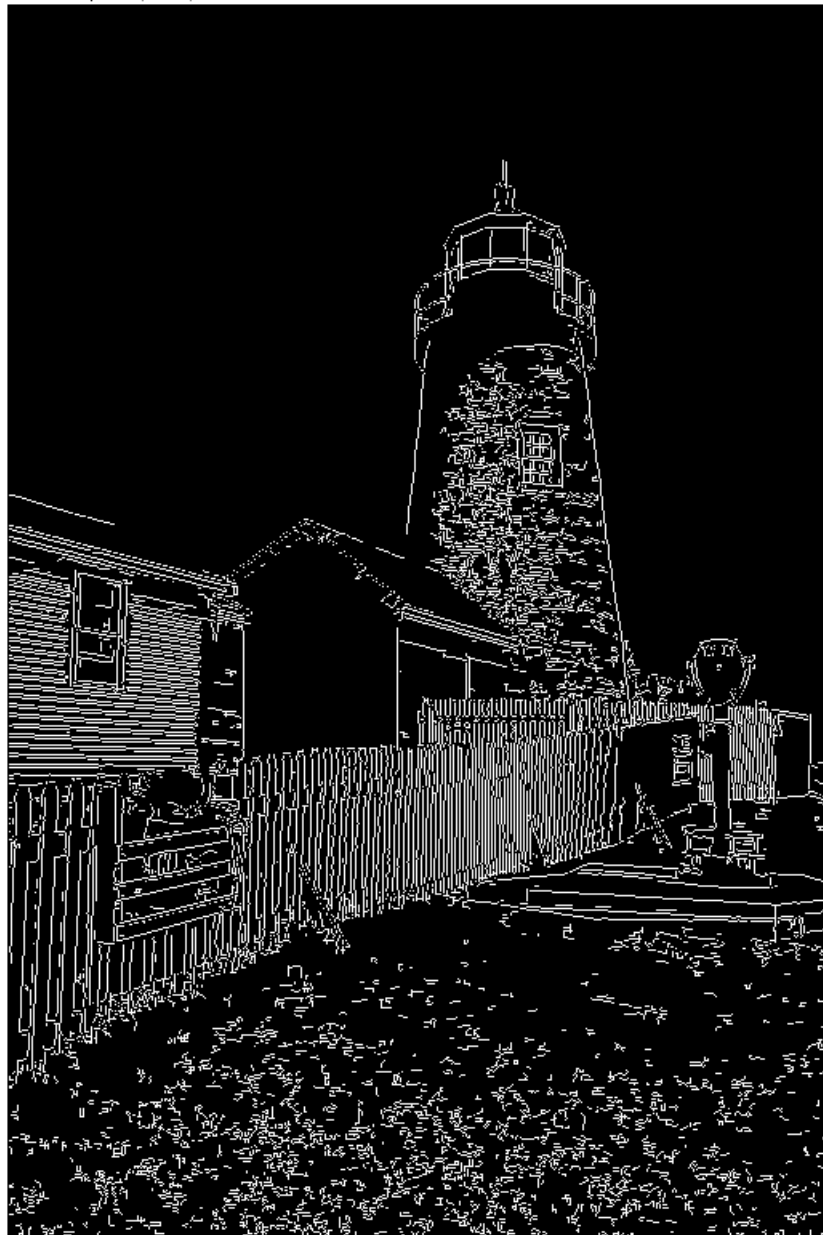
        // Vérifier qu'un pixel voisin en x du pixel de contour est un pixel de contour valide
        // (il a normalement été mis à 128 auparavant lors de la comparaison avec les seuils)
        for(int k=0;k<8;k++) {
            int xk=x+dx8[k], yk=y+dy8[k];
            if (xk<0 || xk>=width) continue;
            if (yk<0 || yk>=height) continue;
            if (maxLoc.get(xk, yk)==128) {
                maxLoc.set(xk, yk, 255);
                newhighpixels.add(new int[]{xk, yk});
            }
        }
    }

    // Les nouveaux pixels de contours sont les pixels de contours trouvés durant l'ancienne itération
    highpixels = newhighpixels;
}

// Binariser l'image : binariser l'image résultante pour
// mettre à les points non-contours
// -----
for (int y=0; y<height; y++) {
    for (int x=0; x<width; x++) {
        if (maxLoc.get(x, y)!=255) maxLoc.set(x,y,0);
    }
}
return maxLoc;
}

```

Nous arrivons au résultat suivant pour un seuil haut = 130 et seuil bas = 70 :



**Seuillage des maxima locaux par hystérésis**

On remarque que, par rapport aux anciennes images, les contours ont une épaisseur de 15 et que, dans la majorité, ces contours sont satisfaisant.

Cependant, en comparant avec les valeurs présentées en cours théoriques, nous nous rendons compte de la différence du rendu. Nous pensons donc qu'il y a eu une erreur de calcul de notre part, notamment dans notre macro pour calculer les maxima locaux. En effet, il se peut qu'une conversion 32bits-8bits n'ait pas été réalisée dans le calcul de la norme du gradient.

## Annexe 1 – Norme du gradient

```
1 // Calcul de la norme du gradient par masque de Sobel
2 //run("Close All");
3 requires("1.41i"); // requis par substring(string, index)
4 setBatchMode(true); // false pour déboguer
5 /***** Création des images *****/
6 sourceImage = getImageID();
7 filename = getTitle();
8 extension = "";
9 if (lastIndexOf(filename, ".") > 0) {
10     extension = substring(filename, lastIndexOf(filename, "."));
11     filename = substring(filename, 0, lastIndexOf(filename, "."));
12 }
13 filenameDerX = filename+"_der_x"+extension; // images des
14 filenameDerY = filename+"_der_y"+extension; // dérivées
15
16 /***** Calcul de la norme du gradient *****/
17 // récupération de la taille de l'image
18 w = getWidth();
19 h = getHeight();
20
21 // Approximation de la dérivée partielle pour (x,y)
22 // -----
23 run("Duplicate...", "title=derivNormX");
24 derivNormX = getImageID();
25 run("32-bit"); // conversion en Float avant calcul des dérivées !!
26 run("Duplicate...", "title=derivX");
27 derivX = getImageID();
28
29 run("Duplicate...", "title=direction-gradient");
30 directionGradient = getImageID();
31
32 run("Duplicate...", "title=derivNormY");
33 derivNormY = getImageID();
34 run("Duplicate...", "title=derivY");
35 derivY = getImageID();
36
37 // Convolution avec le filtre de sobel Hx
38 selectImage(derivNormX);
39 run("Convolve...", "text1=[-1 0 1\n-2 0 2\n-1 0 1\n] normalize");
40 run("Square");
41
42 // Convolution avec le filtre de sobel Hy
```



```
43 selectImage(derivNormY);  
44 run("Convolve...", "text1=[-1 -2 -1\n0 0 0\n1 2 1\n] normalize");  
45 run("Square");  
--
```

## Annexe 2 – Calcul de la direction du gradient

(Note : cette partie de code est la suite directe de la partie de code fournie en annexe 1)

```
56 /**
57  * #####
58  * Calcul de la direction du gradient
59  * #####
60  */
61 // Convolution avec le filtre de Sobel Hx
62 selectImage(derivX);
63 run("Convolve...", "text1=[-1 0 1\n-2 0 2\n-1 0 1\n] normalize");
64
65 // Convolution avec le filtre de Sobel Hy
66 selectImage(derivY);
67 run("Convolve...", "text1=[-1 -2 -1\n0 0 0\n1 2 1\n] normalize");
68
69 // Calcul de la direction du vecteur gradient :
70 for (i = 0; i < w; i++) {
71     for (j = 0; j < h; j++) {
72         // pixel pour dérivée partielle en X
73         selectImage(derivX);
74         px = getPixel(i, j);
75
76         // pixel pour dérivée partielle en Y
77         selectImage(derivY);
78         py = getPixel(i, j);
79
80         // calculer direction pour le pixel
81         selectImage(directionGradient);
82         val = atan2(py, px) * (180/PI); // application de la formule
83         setPixel(i, j, val);
84     }
85 }
86
87 selectImage(derivX);
88 //close();
89 selectImage(derivY);
90 //close();
91
```

## Annexe 3 – détection des maxima locaux

(Note : cette partie de code est la suite directe du code des parties précédentes)

```
192 selectImage(sourceImage);
193 run("Duplicate...", "title=maximaLocaux");
194 maximaLocaux = getImageID();
195
196 for (x = 0; x < w; x++) {
197     for (y = 0; y < h; y++) {
198         // Arrondir la direction du gradient au multiple de 45° Le plus proche
199         selectImage(directionGradient);
200         theta = getPixel(x, y);
201         theta = round((theta / 45)) * 45;
202
203         selectImage(gradNorm);
204         p = getPixel(x, y);
205         p1 = 0;
206         p2 = 0;
207         if(theta == 0 || theta == 180 || theta == -180) {
208             p1 = getPixel(x+1, y);
209             p2 = getPixel(x-1, y);
210         }
211
212         if(theta == 45 || theta == -135) {
213             p1 = getPixel(x+1, y-1);
214             p2 = getPixel(x-1, y+1);
215         }
216
217         if(theta == 90 || theta == -90) {
218             p1 = getPixel(x, y+1);
219             p2 = getPixel(x, y-1);
220         }
221
222         if(theta == 135 || theta == -45) {
223             p1 = getPixel(x-1, y-1);
224             p2 = getPixel(x+1, y+1);
225         }
226
227         selectImage(maximaLocaux);
228         if(p < p1 || p < p2) {
229             setPixel(x, y, 0);
230         } else {
231             setPixel(x, y, p);
232         }
233     }
234 }
```