# TP 2

MANDADJIEV Nicolas - FONKOUA Tambue Nelson

## Exercice 1

### Question 1

Si on prend Rmin = 1, Rmax = 100 et un pas de 2, les valeurs possibles

de r sont 1, 3, 5,... 99. Il y aura 49 valeurs possibles. On retrouve ce résultat en calculant (Rmax - Rmin) / pas - Rmin = 100 - 1) / 2 = 49 en valeur entière

Si on prend change le pas à 0.5, on aura

(100 - 1)/ 0.5  = 200 - 1 = 198

### Question 2

Avec r on peut avoir 99 valeurs possibles

Avec c on peut avoir 99 valeurs possibles

Avec rad on peut avoir (100√2- 5) / 1 = 136 en valeurs entières

Pour avoir le nombre de coordonnées possibles, on multiplie le nombre de valeurs possible de chaque paramètre:

99 * 99 * 136 = 1 332 936

### Question 3

Les indices du tableux commencent à 1

acc(1,1,1) correspond au cercle de coordonnées (Rmin, Cmin, RADmin) => (1, 1, 5)

On trouve la valeur grace a la formule suivante

valeur = valMin + pas * (index -1)

avec index l'index correspondant du tableau acc

acc(10, 7, 30) correspond au cercle de cordonnnées (1 + 9 * 1, 1 + 6 * 1, 5 + 1 * 29) => (10, 7, 34)

### Question 4

On cherche la case telle que

40 = Rmin + Rdelta * (i - 1) = Rmin + Rdelta * i - Rdelta

40 = Cmin + Cdelta * (j - 1) = Cmin + Cdelta * j - Cdelta

13 = Radmin + Raddelta * (k - 1) = Radmin + Raddelta * k - Raddelta

donc

i = (40 - Rmin + Rdelta) / Rdelta = 40

j = 40

k = (13 - RadMin + Raddelta) / Raddelta = 9

## Exercice 2

Pour la détection des contours, nous avons d'abord appliqué (optionnellement) le filtre Gaussian, auquel il faut lui donner les paramètres requis.

Ensuite, Nous avons appliqué le filtre Sobel:

Nous avons calculé le gradient en X, puis en Y, et les avons utilisé pour détecter les contours avec l'algorithme de Canny.

L'accumulateur doit stocker un entier pour chaque (r, c, rad) qui sont tous entier, c'est donc un tableau 3 dimensions d'entiers. Pour représenter cela, nous avons choisi d'utiliser un vector<vector<vector<uint>>>.

Nous avons eu un problème lorsqu'il fallait calculer les maximums locaux (QY):

Nous avons fait 3 boucles imbriqué pour parcourir toutes les valeurs de notre accumulateur, et dans cette boucle on appelait une fonction qui ajouterait dans une liste le point si il est un maximum local. On passait en argument à cette fonction la liste des maximums, l'accumulateur et les index. Le problème est qu'on passait l'accumulateur par valeur, il était donc copié à chaque itération. Notre programme était extrêmement lent, mais une fois le problème identifié, nous avons passé l'accumulateur par pointeur et le problème était résolu.

Une fois les cercles extraits, nous les avons affichés en rouge pour pouvoir bien les identifier

## Exercice 3

Pour four.png, le programme prend environ 1 162 673 125 tick counts.

La complexité est de l'ordre de N^4 car lorsqu'on calcule le rayon pour chaque pixel (en considérons toutes les valeurs (r,c) possibles), on a  4 boucles imbriqué: 2 concernant les pixels (x,y), 2 autres concernant toutes les valeurs possibles (r,c). 4 boucles imbriquées forment donc une complexité N^4. Les autres étapes peuvent être ignorées car elles ont une complexité inférieure à N^4 (moins de 4 boucles imbriqués), et sont donc négligeables.

Sur mon ordinateur, le calcul des contours pour four.png prend à peu près 0.1 seconde.

On sait que pour une image de 100 pixels, le temps de calcul est 0.1.

Pour une image 6 fois plus grande, le temps de calcul sera 6^4 fois plus grand que celui pour l'image de 100 pixels. Donc il serait de 1289,6s, soit environ 2.16 minutes