LABO 8 – TRAITEMENT D'IMAGE
Informatique 1

But du laboratoire (4 heures)

1. Dans ce laboratoire, vous allez travailler sur un programme permettant de faire du traitement d'image. Le but est de travailler avec les tableaux de différentes dimensions.
2. La durée estimée pour réaliser ce laboratoire est de **quatre périodes**. Si vous n'avez pas terminé dans le temps imparti, vous êtes invités à le terminer en dehors des heures de cours.
3. Vous pouvez trouver cette donnée sous forme électronique se trouve sur le site web du cours (http://inf1.begincoding.net/). Vous y trouverez également le corrigé de ce labo à la fin de celui-ci.

Partie 1 – Introduction

Pour cette première partie, nous vous proposons un exercice de base avec les tableaux. Pour ce faire :

1. Créez une classe `TestArray`.
2. Écrivez une fonction prenant un tableau de `double` en argument et retournant la moyenne de ce tableau.
3. Testez votre code avec un tableau contenant les nombres de 1 à 1000. La moyenne doit être de 500.5.

Partie 2 – Traitement d'image

Dans cette partie, vous allez réaliser des filtres d'images comme le font des programmes comme Adobe Photoshop par exemple. Pour simplifier un peu, nous allons travailler avec des images en noir et blanc uniquement (la couleur demande simplement de refaire trois fois les mêmes opérations). ***Notez que valeurs d'un pixel en noir et blanc va de 0 (pour le noir) à 255 (pour le blanc).***

Tâche 1

1. Créez un nouveau projet `Labo8` puis téléchargez le fichier `image_processing.zip` qui se trouve sur le cours à la page du laboratoire et extrayez-en le contenu dans le répertoire `src` du projet.
2. Examinez le contenu de la classe `ImageProcessing` et observez comment une image se trouvant dans le dossier images est affichée à l'aide de la classe `ImageGraphics`.
3. Observez le prototype des méthodes *publiques* offertes dans la classe `ImageGraphics`, principalement les méthodes commençant par *`get`* et *`set`*.

Tâche 2

1. Le code qui vous a été remis n'est pas complet. En effet, la méthode `duplicate` est censée recopier les pixels de l'image `org` dans l'image `cpy`.
2. Notez que pour extraire tous les pixels d'une image (convertie en niveau de gris) sous forme de tableau, la classe `ImageGraphics` vous met à disposition la méthode `getPixelsBW()`. Sa contrepartie est la méthode `setPixelsBW` qui permet de modifier l'image affichée à l'aide d'un tableau.
3. Complétez le code de la méthode duplicate pour que les deux fenêtres contiennent la même image là où se trouve le commentaire `Write your code hereunder`.

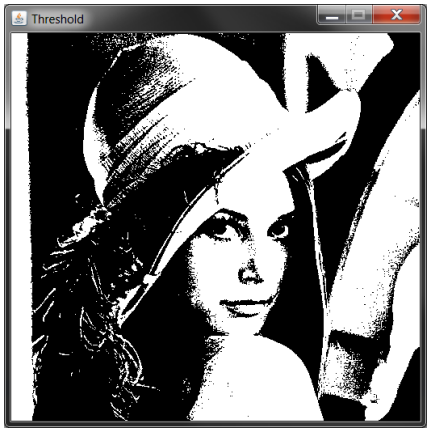
Partie 3 – Fonction de seuillage d'une image

Vous allez maintenant réaliser votre premier filtre, à savoir un seuillage. L'idée de ce filtre est simple : on prend une valeur de seuil (nommé *t* ci-après). Pour tous les pixels de l'image, si la valeur du pixel de l'image est supérieure à *t* on remplace la valeur par blanc et autrement le pixel est mis à noir.

Cette méthode est la plus simple à implémenter et elle permet parfois de séparer les pixels du fond de ceux de l'avant plan.

Tâche 3

1. En vous basant sur la méthode duplicate, implémentez une méthode prenant un tableau d'entiers correspondant à l'image et qui fasse le seuil de celle-ci. Votre méthode devra également avoir la valeur de seuil comme paramètre.
2. Testez votre méthode sur l'image *lena.bmp* avec un seuil de 100. Vous devriez obtenir un résultat comme suit :



Partie 4 – Floutage

Vous allez réaliser une première fonction appelée `mean` qui va appliquer un filtre moyenneur à l'image. Ce filtre a pour but de flouter l'image et d'enlever ainsi ses aspérités.

	x-2	x-1	x	x+1	x+2
y-2					
y-1		128	127	100	
y		255	240	234	
y+1		123	230	84	
y+2					

Figure 1 : Filtre moyenneur

Le principe est le suivant, la valeur d'un pixel est remplacée par la moyenne du carré de 3 par 3 pixels centré sur le pixel. Si on veut calculer la nouvelle valeur du pixel situé à la position (x,y) selon la Figure 1, sa nouvelle valeur sera la moyenne des 9 valeurs affichées. Elle vaut donc pour cet exemple:

$$\frac{128+127+100+255+240+234+123+230+84}{9}=169$$

Tâche 4.1

1. Implémentez une nouvelle fonction appelée `mean` qui prend un *tableau d'entiers* en paramètre et qui retourne le tableau d'entiers transformés, comme la fonction `duplicate`.
2. En vous inspirant du code de `duplicate`, implémenter cette fonction `mean` qui va remplacer la valeur de tous les pixels de l'image résultantes par la moyenne du carré de 3 par 3 pixels dont il fait partie.
3. Faites attention aux bordures de l'image ! Pour simplifier, ne modifiez la première et la dernière ligne ainsi que la première et la dernière colonne.

4. Tester votre fonction en l'appliquant sur l'image `imageProcessing.jpg` et observer comme ce filtre va lisser l'image.

Tâche 4.2 / [Exercice optionnel]

Nous allons réaliser maintenant une nouvelle version de la fonction `mean` qui permet de préciser la taille du domaine sur lequel est calculée la moyenne. Comme paramètre nous pourrons préciser le nombre de lignes de pixels environnants qui sont pris en considération. Pour une largeur de 1, nous prendrions un pixel tout autour, ce qui correspond à la version précédente de la fonction `mean` (carré de 3 par 3 pixels). Une largeur de 4 correspondrait donc à un carré de 9 par 9 pixels.

1. Faites une copie de votre fonction `mean`.
2. Modifiez la liste des paramètres pour qu'elle accepte un paramètre de type `int` qui correspond à la largeur prise en considération.
3. Modifiez le corps de cette nouvelle fonction `mean` afin que le calcul de la moyenne se fasse sur le bon nombre de pixels.
4. Essayez votre nouvelle fonction `mean` avec plusieurs valeurs de largeur du filtre et observez le résultat sur différentes images.

Partie 5 – Détection de contours

Dans cette partie, nous allons cherchez à détecter les contours dans une image. Un contour se définit principalement par une grande variation d'intensité entre deux parties proches (donc une dérivée importante).Pour ceux qui sont intéressés par le domaine, vous pouvez lire la description faite par Wikipedia[1].

La dérivée doit se calculer selon les deux axes. Le calcul est très simple: la dérivée selon x du pixel situé en (x,y) vaut la valeur du pixel de droite (x+1,y) moins la valeur du pixel de gauche (x-1,y). Dans le cas de la Figure 1, la dérivée selon x vaut D_x=234-255=-21.

De même, on peut calculer la dérivée selon y. Elle correspond au pixel du dessous (x,y+1) moins le pixel (x,y-1) du dessus. Dans le cas de la Figure 1, la dérivée selon y vaut D_y = 230-127 = 103.

La norme de la dérivée est calculée selon le théorème de Pythagore :

$$D = \sqrt{D_x^{\,2} + D_y^{\,2}}$$

Dans notre exemple, la norme de la dérivée vaut:

$$D = \sqrt{(-21)^2 + 103^2} = 105.1$$

Tâche 5

1. Implémentez la dérivée comme les autres fonctions (pour chaque pixel de l'image)

2. Testez votre fonction, notamment sur l'image *rice.jpg* qui doit donner un résultat comme suit :

[1] http://fr.wikipedia.org/wiki/Détection_de_contours

Partie 6 – Filtre de Sobel

Une autre méthode pour extraire les contours à l'intérieur d'une image est d'utiliser l'algorithme de Sobel.[2] Cette méthode est très similaire à celle de la dérivée, mais un peu plus compliquée et donne de meilleurs résultats. La Figure 2 représente les poids qui doivent être appliqués aux pixels environnants pour calculer le filtre de Sobel selon x.

	x-2	x-1	x	x+1	x+2
y-2					
y-1		-1		1	
y		-2		2	
y+1		-1		1	
y+2					

	x-2	x-1	x	x+1	x+2
y-2					
y-1		-1	-2	-1	
y					
y+1		1	2	1	
y+2					

Figure 2 : Poids du filtre de Sobel selon x (à gauche) et selon y (à droite)

Pour l'exemple de la Figure 1, la valeur du filtre de Sobel selon x vaudrait:

$$S_x = 100 + 2 \cdot 234 + 84 - 128 - 2 \cdot 255 - 123 = -109$$

De même la valeur du filtre de Sobel selon y vaudrait:

$$S_y = 123 + 2 \cdot 230 + 84 - 128 - 2 \cdot 127 - 100 = 185$$

Comme auparavant, la norme du filtre de Sobel se calcule selon Pythagore et vaut pour cet exemple:

$$S = \sqrt{S_x^{\,2} + S_y^{\,2}} = \sqrt{(-109)^2 + 185^2} = 214.7$$

Comme cette valeur peut être bien plus grande que 255, il vaut certaines fois la peine d'ajouter un facteur correctif qui multiplie la valeur de S pour la rendre plus petite (ou plus grande).

Tâche 6

1. Implémentez une nouvelle fonction appelée `sobel` qui prend un *tableau de pixels* et un `double` (*intensityFactor*) en paramètres et qui retourne le tableau de pixels modifié.
2. Implémentez le corps de cette fonction en calculant la norme du filtre de Sobel pour chacun des pixels de l'image d'entrée.
3. Essayez cette nouvelle fonction sur les images fournies et observez le résultat. Faites aussi varier la valeur de facteur d'intensité. Dans le cas de l'image de la rivière, une valeur de 0.2 est intéressante.

[2] http://fr.wikipedia.org/wiki/Algorithme_de_Sobel