Projet – Traitement d'images

Le but de ce projet est d'implémenter une fonctionnalité simple de traitement d'images : la détection de contours d'une image. Le code sera réalisé en assembleur MIPS, et executé grâce à l'émulateur `Mars`. Le projet est à réaliser en binômes, et à rendre avant le mardi 9 avril 2013 à minuit en respectant les spécifications données dans la section 1.

L'archive `projet.tar.gz` contient :
– le présent document,
– deux images de test `lena.bmp` et `lena256.bmp`,
– l'archive java `Mars.jar`.

1 Spécifications

Votre code, qui sera commenté, prendra la forme d'un fichier `.s`, exécutable grâce à l'émulateur `Mars`. Lors de son exécution, il demandera à l'utilisateur de rentrer une chaîne de caractère désignant une image au format bitmap (`.bmp`). Le résultat de l'éxécution sera une nouvelle image nommée `MonimageContour.bmp`, correspondant au résultat de l'application du filtre de Sobel (voir la section 4, et la figure 1) à l'image `Monimage.bmp`. Lors de l'éxécution du processus, les données nécessaires au traitement (l'image, ses attributs), seront stockées dans le tas.

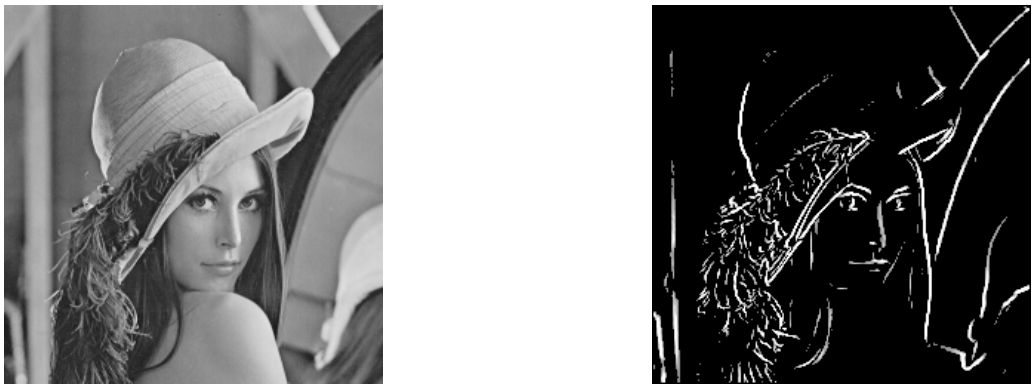


FIGURE 1 – À gauche une image en 256 niveaux de gris, à droite le résultat de l'application du filtre de Sobel sur cette image.

Les images admissibles en entrée seront au moins des images .bmp à 256 niveaux de gris. On pourra, en bonus, gérer des images à 2 ou 16 niveaux de gris, voir à couleurs réelles. En bonus encore, on pourra implémenter d'autres filtres de traitement d'image (par exemple les filtres passe haut ou passe bas).

Le rendu prendra la forme d'une archive nommée `Nom1_Nom2.tar.gz`, qui contiendra : le fichier `.s`, l'émulateur `Mars`, quelques images de test, et éventuellement un fichier `readme.txt` expliquant les fonctionnalités supplémentaires implémentées.

Cette archive sera déposée avant le mardi 9 avril 2013 à minuit sur la plateforme `Moodle`.

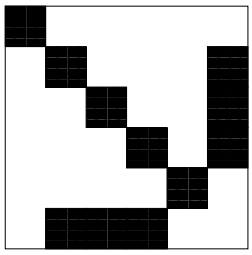
La suite du document présente le format bitmap (section 2), le filtre de Sobel (section 4), et quelques aides sur les appels systèmes disponibles avec l'émulateur `Mars` (section 3). Les questions posées par la suite peuvent être ignorées, elles constitueront cependant un point de départ si vous êtes bloqués. (Il n'est pas demandé de réponse à ces questions dans le matériel rendu.)

2 Images .bmp

Le format .bmp stocke une image sous la forme de la liste de ses pixels, plus précisémment la liste des couleurs de ses pixels. Il permet de représenter :

– des images en couleurs réelles : la couleur de chaque pixel est alors codée sur 3 octets (les composantes bleues, vertes et rouges de la couleur),
– des images à 256, (respectivement 16, 2) couleurs indexées : la couleur d'un pixel est alors un nombre entre 0 et 255 (resp. 15, 1), donc codé sur 1 octet (resp. 4 bits, 1 bit). Chacun de ces nombres représente l'index d'une couleur, et la table d'association index-couleur, appellée palette de couleur, est donné sous forme d'une liste, indexée par l'index de la couleur, dans l'entête du fichier.

L'image à stocker est vue comme un tableau à 2 dimensions de couleurs, chaque case étant associée à un pixel de l'image, et contenant la valeur (couleur réelle ou index) de la couleur de ce pixel. La numérotation des lignes de ce tableau commence à la ligne la plus basse de l'image, et celle des colonnes à la colonne la plus à gauche de l'image. Le fichier .bmp de l'image contient la liste du contenu des éléments de ce tableau, en commençant par l'élément $(0, 0)$, puis $(0, 1)$, ... jusqu'à $(0, M-1)$, où M est le nombre de pixels horizontal de l'image ; puis $(1, 0)$, ..., jusqu'à l'élément $(N-1, M-1)$.



1	0	0	0	0	0	5
0	1	0	0	0	1	4
0	0	1	0	0	1	3
0	0	0	1	0	1	2
0	0	0	0	1	0	1
0	1	1	1	0	0	0
0	1	2	3	4	5	

Palette de couleur : (0x00, 0x00, 0x00), (0xFF, 0xFF, 0xFF)
Image stockée : 0,1,1,1,0,0, 0,0,0,0,1,0, 0,0,0,1,0,1, 0,0,1,0,0,1, 0,1,0,0,0,1, 1,0,0,0,0,0

FIGURE 2 – À gauche une image en 2 couleurs, au centre le tableau des index des couleurs, à droite la palette de couleur, et la liste des index des pixels de l'image.

La figure 2 illustre le codage d'une image à 2 couleurs (noir et blanc) de 6 par 6 pixels.

2.1 Structure globale d'un fichier .bmp

Un fichier .bmp contient les quatre parties suivantes, dans l'ordre :

1. l'entête du fichier, sur 14 octets,
2. l'entête de l'image, sur 40 octets,
3. éventuellement la palette de couleurs, sur $4 \times c$ octets où c est le nombre de couleurs, si l'image n'est pas en couleurs réelles,
4. l'image elle-même, sur $N \times M \times log_2(c)$ octets.

On trouvera sur internet, et notamment à l'adresse :
`http://www.commentcamarche.net/contents/video/format-bmp.php3`
une description détaillée du contenu et de la structuration de chaque partie.

2.2 Quelques questions

Vous pouvez utiliser l'utilitaire `hd` (hexdump) pour explorer le contenu d'un fichier .bmp. Ouvrez le fichier `lena.bmp` fourni dans l'archive avec la commande `hd lena.bmp | more`, pour répondre aux questions suivantes :

1. Quel est le mode de représentation des couleurs de cette image ? Couleurs réelles ? 256 (ou 16, ou 2) couleurs ?
2. À quel offset (décalage depuis le début du fichier, compté en octets) se trouve la palette de couleurs ? Que contient-elle ?
3. `lena.bmp` est une image en niveaux de gris. Quelle est la relation entre l'index représentant la couleur d'un pixel et les composantes (bleues, vertes et rouges) de la couleur représentée par cet index ?
4. Vous devrez stocker l'image (liste des indexes de ses pixels) dans le tas du processus. Comment, grâce aux données des entêtes (du fichier et de l'image) connaître la taille en octets que l'image occupera en mémoire ?
5. L'index de la couleur du pixel $(1, 1)$ se trouve dans le fichier à l'offset 0x0637. À quelles adresses se trouvent :

- son voisin de gauche, son voisin de droite ? (pixels $(1,0)$, $(1,2)$),
- son voisin du haut, son voisin du bas ? (pixels $(2,1)$, $(0,1)$),
- ses voisins diagonaux ? (pixels $(0,0)$, $(0,2)$, $(2,0)$, $(2,2)$).

6. Pouvez-vous généraliser ? Étant donné l'offset du pixel (i,j), comment retrouver l'offset du pixel $(i+k, j+l)$? Quels sont les informations, en plus de la liste des indexes des couleurs de pixels, que vous devrez stocker pour appliquer ces formules ?

3 Mars et appels systèmes

L'émulateur Mars se lance à partir de l'archive java Mars.jar grâce à la commande `java -jar Mars.jar`.

On peut effectuer des appels système grâce au mot clé `syscall`. La liste complète des appels système disponibles ainsi que leur utilisation est décrite à l'adresse :
`http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html`
On notera que pour ouvrir un fichier, (appel système 13), l'adresse du fichier à passer est l'adresse depuis le répertoire duquel a été lancé `Mars`.

On notera également l'appel système 9 qui alloue de réserver de la mémoire dans le tas du processus. Le code assembleur MIPS suivant alloue 4 octets de mémoire dans le tas : la taille mémoire, en octet est passé dans le registre `$a0`, et après l'appel système, l'adresse de la mémoire allouée se trouve dans le registre `$v0`.

```
li $a0 4 #on demande 4 octets
li $v0 9 #9 pour appel système sbrk
syscall
sw $v0 4($sp) #$v0 contient l'adresse de la mémoire réservée dans le tas
```

Enfin, l'émulateur `Mars` peut se révéler assez lent pour effectuer de gros traitement. N'hésitez pas à tester votre code en développement sur de très petites images, sous peine de perdre beaucoup de temps !

4 Filtre de détection de contours

On propose d'implémenter le filtre de Sobel pour des images à niveaux de gris, qui permet de détecter les brusques changements de couleur horizontaux et verticaux d'une image, en calculant deux images intermédiaires : une dont chaque pixel contient le gradient horizontal en le pixel de même coordonnées de l'image originale, l'autre dont chaque pixel contient le gradient vertical. Le seuillage de la somme des valeurs absolues de ces deux images donnera l'image des contours.

Des détails sur le filtre de Sobel peuvent être trouvés à cette adresse :
`http://fr.wikipedia.org/wiki/Filtre_de_Sobel`, et on se propose ici de détailler différentes étapes du filtrage.

On considère à présent une image de $N \times M$ pixels à niveaux de gris comme une matrice de N lignes et M colonnes d'entiers compris entre 0 et 255. Les lignes (respectivement les colonnes) d'une telle matrice sont numérotées de 0 à $N-1$ (resp. $M-1$). Le filtre de Sobel procède par convolution de cette matrice avec deux petites matrices 3*3 appelées masques. On Détaille à présent le produit de convolution d'une matrice par un masque. Dans la suite, si A est une matrice, $A(i,j)$ désigne l'élément i,j de A.

4.1 Produit de convolution de matrices

Soit A la matrice $N \times M$ d'une image en niveaux de gris, et F une matrice 3×3 d'entiers (masque). On appelle convolution de A par F, et on la note $A * F$, l'opération dont le résultat est la matrice d'entiers G de taille $N \times M$, définie par
– $G(i,j) = 0$ si $i = 0$ ou $i = N-1$ ou $j = 0$ ou $j = M-1$,
– $G(i,j) = \sum_{k=0}^{k=2}(\sum_{l=0}^{l=2} F(k,l) \times A(i-1+k, j-1+l))$ sinon.
Soient les matrices :

$$F_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \text{ et } F_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

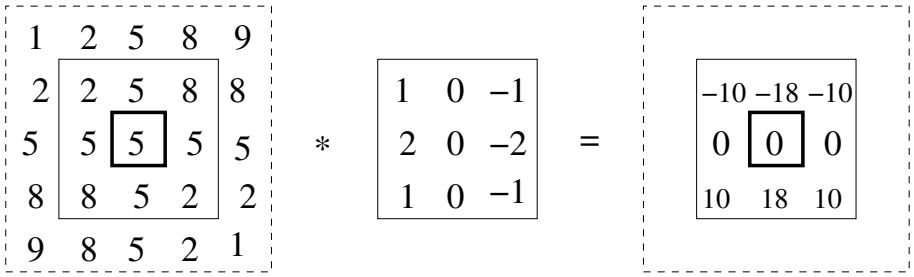


FIGURE 3 – Le produit de convolution de deux matrices.

On propose de détailler la convolution de la matrice A de taille 5*5 donnée à gauche de la figure 3 par F_x. On appelle G le résultat de l'opération.

- Pour $i = 0$ et $0 \leq j \leq 5$, $G(i, j) = 0$.
- Pour $i = 1$ et $j = 0$, $G_(i, j) = 0$.
- Pour $i = 1$ et $j = 1$, $G_(i, j) = F_x(0,0) * A(0,0) + F_x(0,1) * A(0,1) + F_x(0,2) * A(0,2) + F_x(1,0) * A(1,0) + F_x(1,1) * A(1,1) + F_x(1,2) * A(1,2) + F_x(2,0) * A(2,0) + F_x(2,1) * A(2,1) + F_x(2,2) * A(2,2) = 1 \times 9 + 0 \times 8 + (-1) \times 5 + 2 \times 8 + 0 \times 8 + (-2) \times 5 + 1 \times 5 + 0 \times 5 + (-1) \times 5 = 10$.
- ...
- Pour $i = 2$ et $j = 2$, $G_(i, j) = F_x(0,0) * A(1,1) + F_x(0,1) * A(1,2) + F_x(0,2) * A(1,3) + F_x(1,0) * A(2,1) + F_x(1,1) * A(2,2) + F_x(1,2) * A(2,3) + F_x(2,0) * A(3,1) + F_x(2,1) * A(3,2) + F_x(2,2) * A(3,3) = 2 \times 1 + 5 \times 0 + 8 \times (-1) + 5 \times 2 + 5 \times 0 + 5 \times (-2) + 8 \times 1 + 5 \times 0 + 2 \times (-1) = 0$. C'est cette étape qui est illustrée sur la figure 3.

4.2 Filtre de Sobel

On note $A * F_x = G_x$, et $A * F_y = G_y$. G_x et G_y sont des matrices d'entiers de taille $N \times M$, mais ne correspondent pas à des images, car elle contiennent des entiers non-nécessairement compris entre 0 et 255. En fait, l'élément à la ligne i et à la colonne j de G_x (respectivement G_y) correspond au gradient suivant la direction horizontale (resp. verticale) de l'image A.

Afin d'obtenir à partir de G_x (resp. G_y) des images portant une information de contour horizontaux (resp. verticaux), on lui applique deux opérations :

1. passage à la valeur absolue : on définit la matrice $|G_x|$ (resp. $|G_y|$) dont l' élément (i, j) est la valeur absolue de l'élément (i, j) de G_x (resp. G_y),
2. seuillage : on définit la matrice G'_x (resp. G'_y), qui ramène à 255 chaque élément de $|G_x|$ (resp. $|G_y|$) supérieur ou égal à 255.

G'_x et G'_y sont alors des images portant des informations de contour de l'image A : pour un pixel (i, j), $G'_x(i, j)$ (respectivement $G'_y(i, j)$) représente la composante horizontale (resp. verticale) de la "vitesse" à laquelle A change de couleur autour du pixel (i, j). Les contours sont les pixels de l'image où cette vitesse est supérieure à un certain seuil. Pour ne garder dans G'_x (resp. G'_y) que les contours, on effectue un second seuillage, qui ramène à 0 tous les éléments de G'_x (resp. G'_y) inférieurs à un certain seuil s. Vous ferez des essais avec différentes valeurs (entre 0 et 255) pour déterminer une bonne valeur de s.

Pour obtenir une image synthétisant les informations de G'_x et G'_y, on peut additionner $|G_x|$ et $|G_y|$ puis effectuer un nouveau seuillage sur le résultat.

4.3 Quelques questions

1. Mis à part la convolution de matrice, toutes les opérations matricielles utilisées sont des opérations sur un seul élément. (Par exemple, l'élément (i, j) de $|G_x|$ est déterminé uniquement par l'élément (i, j) de $|G_x|$.) Est-il alors possible d'effectuer toutes les opérations en un seul parcours des éléments de l'image A ?
2. De quels opérations élémentaires aurez-vous besoin pour réaliser ces opérations ?
3. Considérez le fichier `lena.bmp`, et le fichier `lenaContour.bmp` résultat de l'application du filtre de Sobel. Quels sont les points communs entre ces deux fichiers ?