TP 6 – 7
Traitement d'image
Deuxième Partie

Table des matières

Partie 3.2

r10 représente le nombre de pixels sur une ligne. Au départ, cette valeur était stockée dans *rcx* mais *rcx* contient maintenant 0 puisqu'on l'a utilisé comme compteur décrémenté. Afin de compenser cela, nous utilisons *r11* comme registre tampon avant de manipuler *rcx*. *r11* contient donc le contenu original de *rcx*, en début de programme, (donc le nombre de pixels par ligne). On stocke donc le contenu de *r11* dans *r10* avant de se lancer dans la suite de l'algorithme.

Afin de faire fonctionner l'algorithme, nous devons travailler sur une image en niveaux de bleu. Une simple modification de l'algorithme de niveau de gris est réalisée : garder à 0 les niveaux de rouge et de vert de chaque pixel. Cette image en niveaux de bleu (stockée dans *img_temp1*, donc dans *r9*) nous sert maintenant d'image source pour le reste de l'algorithme. Nous la déplaçons donc dans le registre *r8* à l'aide d'un *mov r8, r9*.

Afin d'avoir dans *r9* l'adresse du premier pixel de *img_temp2*, on va le chercher dans la pile. On rappelle que cette donnée avait été stocké après le *@return* et le shadow space à *rsp+40*. Un *mov r9, [rsp+40]* suffit donc. Cependant, on remarque que le premier pixel à traiter de *img_temp2* est situé aux coordonnées (2,2). On ajoute donc à *r9* la taille d'une ligne et celle d'un pixel pour arriver à ces coordonnées à partir des coordonnées initiales (1,1).

Partie 3.3

Question 3.3.1

Afin d'afficher les contours de l'image, nous avons besoin de réaliser sur chaque pixels une convolution de deux matrices 3x3. L'une des matrices étant l'opérateur de Sobel, l'autre étant le voisinage du pixel manipulé. Cela implique que nous ne pouvons pas manipuler les pixels de la première et dernière ligne, ni les pixels de la première et dernière colonne. Ce qui signifie que dans nos boucles, au lieu de parcourir toute une ligne, nous allons parcourir toute une ligne moins 2. D'où la présence du *sub rdx, 2* (*rdx* contenant au départ le nombre de lignes de l'image).

Pour passer à la ligne suivante avec *r8*, on ajoute donc la taille de 2 pixels, à savoir 8 puisque les 2 derniers pixels ne sont pas traités. On procède de la même manière pour *r9* malgré le décalage, cette fois les 2 pixels non-traités sont en fin de ligne actuelle et en début de ligne suivante. Nous utilisons l'instruction *add r8, 8* ou *add r9, 8*.

Question 3.3.2

Tout comme le nombre de colonne, le nombre de ligne est décrémenté par 2 puisque qu'on ne peut pas non plus traiter les 2 pixels en bordure. On affecte donc la valeur de *r10* à *rcx* qu'on décrémente de 2 ensuite avec un *mov rcx, r10* et un *sub rcx, 2*.

Cette fois, *r8* et *r9* s'incrémente de la taille d'un pixel à chaque itération pour pouvoir passer au pixel suivant. Nous utilisons donc un *add r8, 4* ou un *add r9, 4*.

Partie 3.4

L'affichage des contours de l'image se réalise en calculant le gradient de couleurs suivant l'axe X et suivant l'axe Y. Comme dit précédemment, nous devons effectuer une convolution de la matrice de Sobel par la matrice 3x3 représentant le voisinage du pixel manipulé. On obtient les pixels adjacent sau pixels à traiter par de simples calculs en fonction de *r8* :

- pixel haut gauche a11 : [r8]
- pixel haut milieu a12 : [r8 + 4]
- pixel haut droit a13 : [r8 + 8]
- pixel gauche a21: [r8 + 4*r10]
- pixel droit a23: [r8 + 4*r10 + 8]
- pixel bas gauche a31 : [r8 + 8*r10]
- pixel bas milieu a32 : [r8 + 8*r10 + 4]
- pixel bas droite a33 : [r8 + 8*r10 + 8]

Nous ne réalisons pas de fonction réalisant une convolution. En effet, la matrice de sobel est toujours la même tout au long du déroulement de l'algorithme. A la place, nous réalisons l'opération directement en nous basant sur la formule donnée dans l'énoncé.

Nous effectuons successivement le calcul suivant l'axe X et suivant l'axe Y afin d'obtenir Gx et Gy. Pour obtenir la valeur absolue de ces données, des sauts inconditionnels sont indispensable. On inspecte le signe de la donnée, si elle est négative, on utilise la mnémonique neg.

La valeur finale du gradient est finalement obtenue par la relation : $G = |Gx| + |Gy|$. Par un simple add entre Gx et et Gy, on obtient donc G. On utilise ensuite l'algorithme de saturation donné dans l'énoncé pour obtenir la valeur finale de G.

Cette valeur G est ensuite placée à l'adresse *[r9]*, c'est à dire à l'adresse du pixel manipulé dans *img_temp2*. G est placé sur les bits de niveaux de rouge, de vert et de bleu du pixel à l'aide de 3 décalages vers la gauche et sommes successives afin d'obtenir une image finale en niveaux de gris. L'algorithme est terminé.