MI01 TP6 – Traitement d'image – Deuxième partie.

Ce TP est à réaliser en deux séances. Si vous n'avez pas terminé la conversion de l'image en niveaux de gris, finissez-là, elle est essentielle pour le fonctionnement de l'algorithme de détection de contours.

1 Détection des contours d'une image

1.1 Principe

Les contours caractérisent les frontières des objets à l'intérieur d'une image et sont d'une importance fondamentale en termes de traitement d'images. Ils sont définis par les zones d'une image contenant de fortes variations de contraste – un saut d'intensité entre un pixel et ses voisins. Détecter les contours d'une image réduit de manière significative la quantité de données et élimine l'information inutile, tout en préservant les structures importantes de l'image.

Les algorithmes de détection de contour se divisent principalement en deux catégories : les méthodes basées sur l'analyse du gradient de l'image, et les méthodes basées sur l'analyse de son laplacien. L'algorithme que vous allez utiliser dans ce TP est du premier type.

Les méthodes basées sur le gradient détectent les contours en trouvant les minima et maxima de la dérivée première de l'image. En une dimension, on peut assimiler un contour à une rampe et calculer sa dérivée permet de déterminer son emplacement. Supposons un signal unidimensionnel (Figure 1) qui comprend un contour caractérisé par un saut d'intensité (on peut imaginer qu'il s'agit d'une ligne de l'image).

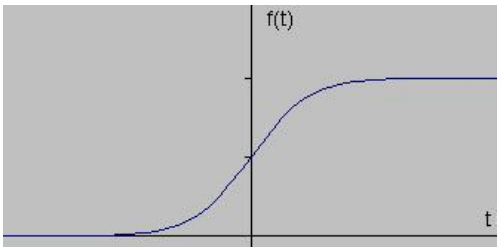

Figure 1 - Saut d'intensité dans un signal unidimensionnel f(t)

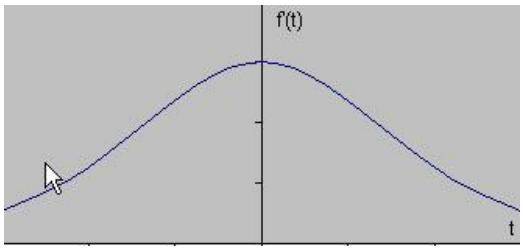

Figure 2 - Dérivée du signal f(t)

Le gradient de ce signal (Figure 2), qui en une dimension est simplement sa dérivée, comporte clairement un maximum situé au centre du contour.

1.2 Opérateur de Sobel

Cette technique de détection peut être facilement étendue en deux dimensions pour traiter des images à condition d'avoir une approximation précise du gradient dans l'image. L'opérateur de Sobel effectue une mesure en deux dimensions du gradient de l'intensité d'une image en niveaux de gris, permettant de trouver la norme absolue du gradient en tout point de l'image. Cet opérateur se présente sous la forme de deux masques de convolution S_x et S_y (Figure 3) qui, appliqués à l'image comme expliqué dans le paragraphe suivant, fournissent pour chaque pixel la norme G_x du gradient dans la direction x (colonnes) et la norme G_y

du gradient dans la direction *y* (lignes). Les masques étant bien plus petits que l'image, on les applique en les décalant d'un pixel à chaque fois pour parcourir toute l'image.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Figure 3 - Masques de convolution de Sobel

La norme *G* du gradient d'un pixel est calculée à partir de sa norme sur *x* et de sa norme sur *y* comme suit :

$$|G| = \sqrt{G_x{}^2 + G_y{}^2}$$

que l'on peut approximer par $|G| = |G_x| + |G_y|$ pour simplifier les calculs.

1.3 Application d'un masque de convolution à une image

Le masque à appliquer glisse le long de l'image de départ pour calculer la valeur d'un pixel de l'image résultante, puis décalé à droite d'un pixel, pour chaque pixel d'une ligne. A la fin de la ligne, le traitement recommence au premier pixel de la ligne suivante et ainsi de suite jusqu'à la fin de l'image. Dans l'exemple suivant, on veut appliquer le masque au pixel a_{22} de l'image de départ.

Image de départ

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}

Masque

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

Image résultante

b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}
b_{31}	b_{32}	b_{33}	b_{34}	b_{35}	b_{36}
b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	b_{46}
b_{51}	b_{52}	b_{53}	b_{54}	b_{55}	b_{56}
b_{61}	b_{62}	b_{63}	b_{64}	b_{65}	b_{66}

La valeur du pixel b_{22} dans l'image résultante est :

$$b_{22} = m_{11}a_{11} + m_{12}a_{12} + m_{13}a_{13} + m_{21}a_{21} + m_{22}a_{22} + m_{23}a_{23} + m_{31}a_{31} + m_{32}a_{32} + m_{33}a_{33}$$

On constate que les pixels de la première et de la dernière ligne, ainsi que de la première colonne et de la dernière colonne ne peuvent être manipulés par un masque de taille 3x3. En effet, quand on place le centre du masque sur un pixel de la première ligne par exemple, une partie du masque se trouve en dehors des limites de l'image de départ, ce qui rend le calcul impossible.

2 Algorithme de traitement

L'algorithme de traitement consiste à parcourir toute l'image source et à calculer pour chaque pixel la valeur *G* du gradient total qui sera stockée dans l'image de destination. Cette valeur est un entier entre 0 et 255, 255 représentant les zones de gradient maximal. Cette valeur représentant aussi l'intensité maximale, l'affichage direct du gradient aboutirait à une image des contours en blanc sur fond noir. Puisqu'on veut les afficher en noir sur fond blanc, il faut inverser l'intensité avant de stocker la valeur dans l'image de destination.

```
Source ← adresse du premier pixel de l'image source
Dest ← adresse du second pixel de la seconde ligne de l'image de destination
POUR chaque ligne i de l'image source FAIRE
        POUR chaque colonne j de la ligne i FAIRE
                Gx ← Convolution du masque Sx avec les 9 pixels à l'adresse Source
                Gy ← Convolution du masque Sy avec les 9 pixels à l'adresse Source
                G ← |Gx|+|Gy|
                G ← 255 – G
                SI G < 0 ALORS G = 0
                Pixel à l'adresse Dest ← G
        FIN POUR
FIN POUR
```

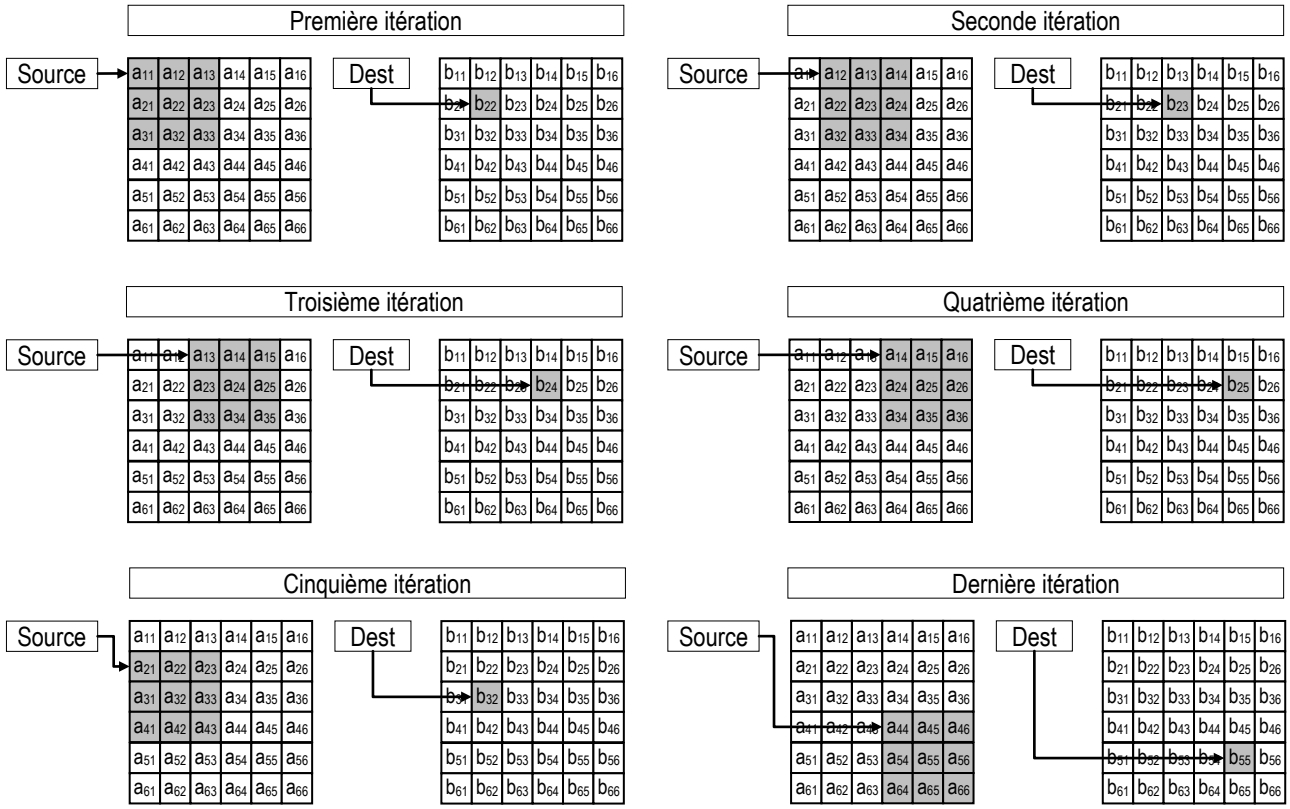

Figure 4 - Exemple d'exécution

3 Travail à réaliser

On suppose que chaque pixel de l'image source contient dans sa composante bleue le niveau de gris associé au pixel, toutes les autres composantes doivent être nulles. Si vous aviez transformé l'image après la conversion en niveaux de gris afin qu'elle s'affiche effectivement en niveau de gris et pas en niveaux de bleu, supprimez cette partie du code pour l'instant (mais conservez-là, on utilisera cette transformation à la fin de la détection de contours).

3.1 Mise en place

Sur le site Moodle de l'UV, téléchargez le fichier du TP (tp_image_p2.zip) et copiez son contenu dans le dossier de projet du TP précédent (conversion en niveaux de gris). Remplacez les fichiers *appli.c*, *appli_r.h* et *appli.rc*. La nouvelle version de l'application Windows contient une implémentation du détecteur de contours de Sobel. Vous implémenterez votre version à la suite de la conversion en niveaux de gris dans le fichier *image.asm*.

N'oubliez pas de nettoyer la solution (menu « Générer » puis « Nettoyer la solution ») après la copie, pour forcer le compilateur à recompiler tous les fichiers modifiés.

3.2 Calcul des adresses source et destination

L'image source est le résultat de la conversion en niveaux de gris de l'image initiale, réalisée dans le TP précédent. Il s'agit donc de l'image *img_temp1*. On stocke le résultat de la détection de contours dans l'image *img_temp2*.

On suppose que le registre ESI contient l'adresse du premier pixel auquel appliquer le masque (Source) et EDI l'adresse du pixel de *img_temp2* dans lequel on veut stocker le résultat. D'autre part, on va stocker dans EBP la taille d'une ligne en pixels.

- Comment faut-il initialiser EBP, ESI et EDI pour tenir compte de ces hypothèses (n'oubliez pas que chaque pixel est stocké dans un mot de 32 bits soit quatre octets) ?
- Quel problème peut poser l'usage de EBP comme registre général dans le calcul ?

Ajouter au sous programme *process_image_asm* les calculs nécessaires.

3.3 Construction de la double itération

Afin d'économiser un registre, le registre ECX va être utilisé pour réaliser à la fois le compteur de lignes i et de colonnes j. On suppose que la partie haute de ECX (16 bits de poids fort) contient le compteur de lignes restant à traiter dans l'image, la partie basse (16 bits de poids faible, accessibles en utilisant le registre CX) contient le compteur de colonnes restant à traiter dans la ligne.

3.3.1 Itération sur les lignes

- A quelle valeur faut-il initialiser le mot de poids fort de ECX (peut-être vaut-il mieux le faire avant de modifier EBP dans la question précédente) ?
- De quelle manière décrémenter seulement les poids forts de ECX ?
- En utilisant une opération logique, comment détecter la condition d'arrêt (il ne reste plus aucune ligne à traiter) ?
  Indication : vous pourrez utiliser l'instruction TEST, qui modifie les drapeaux sans modifier la donnée testée.
- Comment modifier ESI à la fin du traitement d'une ligne pour passer à la ligne suivante ? Même question pour EDI.
- Implémenter cette boucle en assembleur dans le sous-programme *process_image_asm*.

3.3.2 Itération sur les colonnes

- A quelle valeur initialiser CX au début de chaque ligne ?
- Comment doivent évoluer ESI et EDI dans cette boucle ?
- Implémenter cette boucle imbriquée dans la précédente en assembleur.

Pour tester le fonctionnement des deux boucles, modifiez chaque pixel correspondant de l'image source et de l'image destination de telle manière qu'il prenne une couleur identique (par exemple rouge). Quelle devrait être la structure de l'image source et de l'image de destination après cette modification ?

3.4 Calcul du gradient de chaque pixel

Une fois que vous êtes satisfait de la manière dont l'image source et l'image destination sont parcourues dans la double boucle imbriquée que vous venez de mettre en œuvre, il s'agit de réaliser le calcul proprement dit pour chaque pixel.

Supprimez le code de modification de l'image source et de l'image de destination utilisé dans la question précédente !

On stocke la valeur de Gx dans EBX, la valeur de Gy dans EDX.

- En supposant que ESI contienne l'adresse de a_{11} dans l'image source, quelle est l'adresse de a_{12} ? L'adresse de a_{21} ? L'adresse de a_{31} ?
  Indication : on peut utiliser EBP comme index dans les modes d'adressage indexés avec facteur d'échelle. Il faut prendre garde à ce qu'il soit bien utilisé comme index (mov eax, [esi + ebp]) et non comme base (mov eax, [ebp + esi]), car dans ce deuxième cas le déplacement serait pris par rapport à SS et non DS. Attention, EBP contient le nombre de pixels d'une ligne, pas le nombre d'octets correspondant !
- Implémentez un programme ce calcul de *Gx* et de *Gy*. Ne faites que les calculs nécessaires, inutile de chercher à construire un programme de convolution avec des masques quelconques.
- Comment calculer la valeur absolue d'un registre ? Vous pourrez utiliser les sauts conditionnels JS (saut si drapeau de signe SF positionné) et/ou JNS (saut si drapeau de signe SF effacé). Toutes les instructions arithmétiques modifient le drapeau SF pour refléter le signe de leur résultat (SF = 1 si négatif).
- Comment calculer la valeur finale de G avant de le stocker dans l'image de destination ?
- Stockez le résultat dans l'image de destination de telle manière qu'elle s'affiche en niveaux de gris et pas en bleu.

Important : le compilateur Visual C++ s'attend à ce que les registres EBX, ESI et EDI soient préservés par les appels de sous-programmes.