

# **Analyse, acquisition et traitement d'image**

## **TP1**

Aurélien CHEMIER 10908892 et Romane LHOMME 11006689

27 octobre 2014

## 1 Introduction

Dans le cadre de ce projet d'analyse d'image, nous devons implémenter une méthode de détection de contours, à l'aide des filtres vus en cours. Ensuite, une étape de traitement et d'affinage des contours sera effectuée afin d'avoir un meilleur résultat. Ce rapport a pour but de présenter et d'expliquer nos choix lors de ces différentes étapes.

## 2 Technologies utilisées

Ce programme de détection de contour a été fait en C++ qui est un langage objet que nous avons l'habitude d'utiliser.

Pour manipuler les images, nous avons choisi d'utiliser OpenCV, une bibliothèque graphique spécialisée dans le traitement d'image en temps réel. Cette librairie a plusieurs avantages :

- Elle est gratuite.
- Elle permet une gestion simple des images (lecture, écriture, sauvegarde...).

L'utilisation de notre programme se fait en ligne de commande.

Tous nos tests ont été fait sur l'image "Lena", un classique du traitement d'image.



FIGURE 1 – Lena

## 3 Détection des contours

L'étape de détection des contours se fait uniquement sur des images en niveaux de gris.

Dans un premier temps, on procède au calcul du vecteur gradient en chaque point de l'image. La méthode demandée consiste à appliquer des opérateurs (ou masques) de convolution (tableau  $M \times M$ ).

Pour chaque Pixel de l'image, on fait la somme du produit des pixels voisins avec la case du filtre correspondante, comme le montre le code suivant.

Listing 1 – Utilisation d'un masque de convolution 3x3

```
for (i = 0; i < 3; ++i)
{
    for (j = 0; j < 3; ++j)
    {
        gradient += p[i][j] * Filtre[i][j];
    }
}
```

Le pixel sur lequel s'applique le filtre se situe au milieu de celui ci, ici en position 1,1.

Le gradient est ensuite normalisé entre 0 et 255 et devient la valeur du pixel courant dans l'image filtrée.

Tous les filtres appliqués dans ce TP sont de dimension 3x3. [H] Différents filtres peuvent être utilisés :

- Prewitt

$$\text{horizontal} \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{vertical} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

— Sobel

$$\text{horizontal} \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{vertical} \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

— Kirsch

$$\text{horizontal} \begin{pmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{pmatrix} \quad \text{vertical} \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{pmatrix}$$

— D'autres masques peuvent être utilisés.

Ces différents filtres appliqués à Lena donnent :

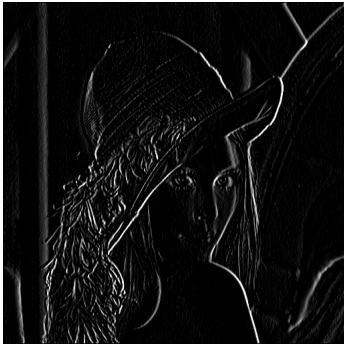


FIGURE 2 – Prewitt Horizontal

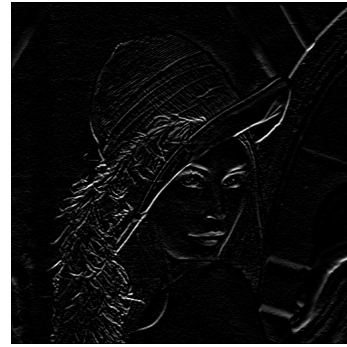


FIGURE 3 – Prewitt Vertical

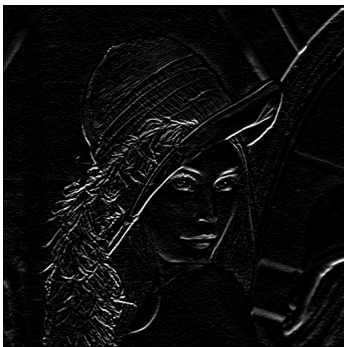


FIGURE 4 – Sobel Horizontal

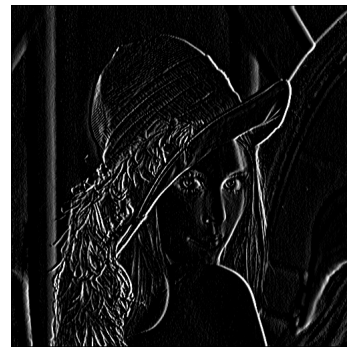


FIGURE 5 – Sobel Vertical



FIGURE 6 – Kirsch Horizontal



FIGURE 7 – Kirsch Vertical

La complexité de cette méthode est de l'ordre

$$w \times h$$

avec :

- $w$  la largeur de l'image,
- $h$  la hauteur de l'image.

### 3.1 Filtrage bidirectionnel

Le filtrage bidirectionnel consiste à appliquer deux masque de convolutions sur la même image. La valeur du gradient devient :

$$G = \sqrt{GV^2 + GH^2}$$

Avec

- $G$  le gradient du filtre bidirectionnel,
- $GV$  le gradient du filtre vertical,
- $GH$  le gradient du filtre horizontal.



FIGURE 8 – Prewitt Bidirectionnel

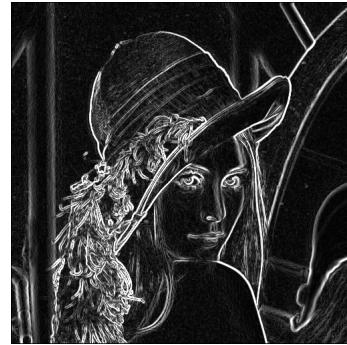


FIGURE 9 – Sobel Bidirectionnel



FIGURE 10 – Kirsh Bidirectionnel

L'avantage de cette méthode est que seul 2 filtres sont nécessaires pour calculer le gradient en 1 point. Cependant elle peut être plus sensible au bruit que la méthode multidirectionnelle.

Comme les deux filtres sont calculé sur le même de l'image, la complexité est également de

$$w \times h$$

.

### 3.2 Filtrage multidirectionnel

Pour le calcul du filtre multidirectionnel deux masques diagonaux sont rajoutés, voici ceux du filtre de Prewitt :

$$\text{Diagonal gauche} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix} \quad \text{Diagonal droite} \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix}$$

Ces filtres donnent :



FIGURE 11 – Prewitt Diagonal Gauche

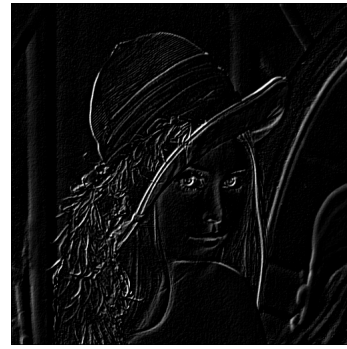


FIGURE 12 – Prewitt Diagonal Droite

Le filtre multidirectionnel calcule donc un filtre bidirectionnel avec les filtres diagonaux correspondants.

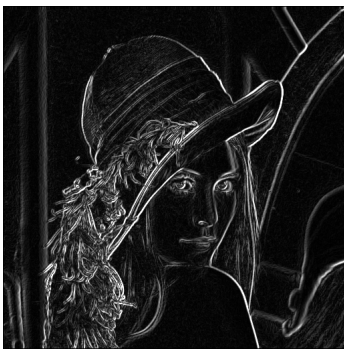


FIGURE 13 – Prewitt Multidirectionnel

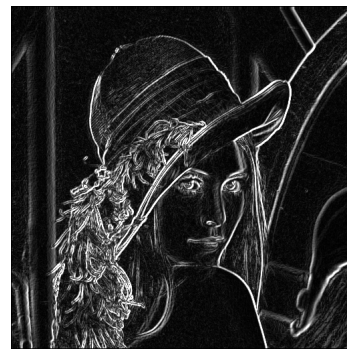


FIGURE 14 – Sobel Multidirectionnel



FIGURE 15 – Kirsh Multidirectionnel

Le calcul des masques se faisant toujours en un seul passage sur l'image, la complexité ne change pas.

L'utilisation de plus de filtre permet de réduire le bruit sur la detection des contours. Néanmoins, ce bruit est toujours présent, c'est pourquoi il faut passer à l'étape du seuillage.

## 4 Seuillage

Le seuillage consiste à "filtrer" les pixels de l'image filtrer en fonction d'un seuil. Si la valeur du pixel courant est inférieur au seuil, elle est passée à 0, sinon elle prend la valeur maximale (ici 255).

Celui ci peut être déterminer de plusieurs manière.

## 4.1 Seuillage fixe

Le seuil  $S$  est fixé et est commun à toute l'image. néanmoins il faut choisir une bonne valeur pour avoir un résultat correct.

Les exemples ci-dessous ont été calculé sur un filtre de Prewitt multidirectionnel.

FIGURE 16 –  $S = 20$ FIGURE 17 –  $S = 50$ FIGURE 18 –  $S = 70$ FIGURE 19 –  $S = 100$ FIGURE 20 –  $S = 150$ FIGURE 21 –  $S = 200$ 

Pour des seuils très bas, on remarque bien que le bruit est important (figure 16). Au contraire, dans le cas de seuil élevé, certain contours sont effacés et l'image perd en précision (figure 20 et 21).

La complexité de cette méthode est de

$$w \times h$$

## 4.2 Seuillage global

Nous venons de voir que le choix du seuil est crucial dans le seuillage de l'image. Il est rapidement contraignant de chercher "à la main" le seuil optimal pour une image. Le seuillage global permet de déterminer un seuil pour une image : il calcule la valeur moyenne des gradients de l'image et prend cette moyenne comme seuil.



FIGURE 22 – Seuillage Global de Prewitt multidirectionnel

La complexité est plus grande car on effectue deux parcours de l'image :

$$2 \times w \times x$$

L'image contenant beaucoup de nuances de gris comme contours, le résultat contient beaucoup de "bruit" dans les contours.

### 4.3 Seuillage local

Le seuillage local calcule la moyenne des pixels voisins au pixel courant. Une fois cette moyenne calculé, on la compare avec le pixel courant.



FIGURE 23 – Seuillage Local de Prewitt multidirectionnel

Le résultat est bruité mais celui-ci a une précision supérieur à celle du seuil global. En effet, on retrouve toute les courbes intacts dans l'image seuillée.

La complexité est plus élevée a cause du calcul systématique de la moyenne locale :

$$l^2 \times w \times x$$

avec  $l$  la taille de la zone où est calculé la moyenne.

### 4.4 Seuillage par Hysteresis

Dans le cas d'un seuillage par hysteresis, on utilise deux seuil, un seuil Bas  $S_b$  et un seuil haut  $S_h$ .

```
Si Pixel_courant <  $S_b$  alors Pixel_courant = 0
Sinon Si Pixel_courant >  $S_h$  alors Pixel_courant = 255
Sinon Si un voisin de Pixel_courant = 225 alors Pixel_courant = 255
    Sinon Pixel_courant = 255
```



FIGURE 24 – Seuillage Hysteresis de Prewitt multidirectionnel ( $S_b = 44$ ,  $S_h = 60$ )

Cet algorithme permet de réduire le bruit et les trous dans les contours et donne d'excellents résultats.

La difficulté de cette méthode est dans le choix des valeurs de  $S_b$  et  $S_h$ . Les mêmes problèmes que le seuillage fixe peuvent se produire. (voir 4.1).

La complexité de cette algorithme est de

$$n \times w \times h$$

avec  $n$  le nombre de passage sur l'image pour contrôler les pixel entre deux valeurs.

Une fois, les contours bien définis par le seuillage, il reste encore une étape : l'affinage.

**5 Affinage**

**6 Conclusion**