

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from this bar, containing the date.

21/10/2013

Compression des images numériques

Compte rendu TD1 : transformée en
ondelettes, filtre de Haar

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Guénon Marie et Favreau Jean-Dominique
VIM / MASTER SSTIM

Table des matières

Préambule :	2
Librairies :	3
Filtre de Haar, Analyse :	4
Méthode directe :	4
Filtrage	6
Reconstitution, Synthèse :	7
Inversion directe	7
Filtrage	Erreur ! Signet non défini.
Récurtivité	9
Analyse	9
Synthèse	10
Annexes :	Erreur ! Signet non défini.

But : réaliser une transformée en ondelettes (analyse / synthèse) au moyen du filtre de Haar.

Préambule :

Après une première approche avec Scilab, nous avons préféré utiliser le langage de programmation C++ pour implémenter ce projet. En effet, ce langage nous permet d'avoir une approche simplifiée du traitement d'image ainsi que des calculs optimisés : les calculs matriciels et les itérateurs sont déjà implémentés et optimisés pour le genre de calculs que nous allons devoir effectuer. Nous obtiendrons donc ainsi de meilleurs résultats, autant au niveau qualitatif, qu'au niveau de la rapidité d'obtention.

Librairies :

De manière native, C++ ne sait pas charger d'image. Nous avons donc du installer la librairie :

OpenCv2

```
#include<opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/core/core_c.h>
#include <opencv2/highgui/highgui.hpp>
```

Cette librairie permet de charger, afficher et traiter les images.

```
cv::Mat input = cv::imread("lena.bmp",CV_LOAD_IMAGE_COLOR);
cv::namedWindow("input", CV_WINDOW_NORMAL );
cv::imshow("input",br);
```



Filtre de Haar, Analyse :

Méthode directe :

Une fois l'image à traiter chargée, nous allons la traiter. Pour ce faire, nous allons la parcourir par bloc de quatre pixels, que l'on nommera $\begin{pmatrix} x & y \\ z & t \end{pmatrix}$. On applique alors les quatre filtres de Haar à chaque point considéré de notre image de départ :

- Moyenne des pixels avoisinants, on applique le filtre : $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ et on obtient l'image résultante :

$$I_0 = x + y + z + t$$

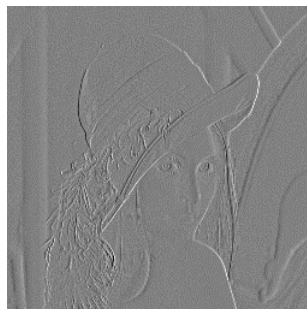
```
*ptr img0++ = val11 + val12 + val21 + val22;
```



- Gradient X, on applique le filtre : $\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$ et on obtient l'image résultante :

$$I_1 = x - y + z - t$$

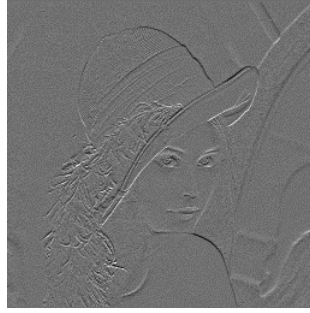
```
*ptr img1++ = val11 - val12 + val21 - val22;
```



- Gradient Y, on applique le filtre : $\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$ et on obtient l'image résultante :

$$I_2 = x + y - z - t$$

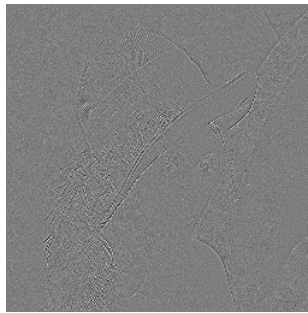
```
*ptr img2++ = val11 + val12 - val21 - val22;
```



- « Gradient diagonal », on applique le filtre : $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ et on obtient l'image résultante :

$$I_3 = x - y - z + t$$

```
*ptr img3++ = val11 - val12 - val21 + val22;
```



Ce qui donne, une fois tous les filtres réunis ensemble, on obtient¹ :



¹ Les images qui sont affichées ici ont été retouchées pour pouvoir être visualisable. En effet, une fois les filtres appliqués, les valeurs obtenues ne sont pas des images et doivent être traduites et étirées entre 0 et 255.

Filtrage

Avant de filtrer l'image, nous devons lui appliquer un traitement préalable. En effet, pour éviter les effets de bords et les problèmes aux limites, nous devons insérer une ligne et une colonne supplémentaire à la fin de l'image. Pour cela, nous insérons par miroir (copie du pixel précédent) ces lignes à droite et en bas de l'image.

Après quoi nous appliquons les quatre filtres vus précédemment mais sur l'intégralité de l'image :

- Moyenne des pixels avoisinants $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$:

$$I_0 = x + y + z + t$$

```
*ptr_img0++ = val11 + val12 + val21 + val22;
```

- Gradient X $\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$:

$$I_1 = x - y + z - t$$

```
*ptr_img1++ = val11 - val12 + val21 - val22;
```

- Gradient Y $\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$:

$$I_2 = x + y - z - t$$

```
*ptr_img2++ = val11 + val12 - val21 - val22;
```

- « Gradient diagonal » $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$:

$$I_3 = x - y - z + t$$

```
*ptr_img3++ = val11 - val12 - val21 + val22;
```

Pour finir, nous effectuons le sous échantillonnage sur les quatre filtres obtenus : nous prenons un pixel sur deux en fonction du filtrage effectué : Nous considérons ici que le numéro du premier pixel est (0,0)

- Moyenne des pixels avoisinants $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$:

On prend les colonnes et les lignes paires.

- Gradient X $\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$:

On prend les colonnes impaires et les lignes paires.

- Gradient Y $\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$:

On prend les colonnes paires et les lignes impaires.

- « Gradient diagonal » $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$:

On prend les colonnes et les lignes impaires.

```
for(int row=0; row<rows2;row++)
{
    const int row_x2 = row<<1;
    const int row_x2p1 = row_x2+1;
    ptr_img0 = img0[row_x2];
    ptr_img1 = img1[row_x2];
    ptr_img1++;
    ptr_img2 = img2[row_x2p1];
    ptr_img3 = img3[row_x2p1];
    ptr_img3++;
    for(int j=0; j<cols2; j++)
    {
        *ptr_dst0++=*ptr_img0;
        ptr_img0+=2;
        *ptr_dst1++=*ptr_img1;
        ptr_img1+=2;
        *ptr_dst2++=*ptr_img2;
        ptr_img2+=2;
        *ptr_dst3++=*ptr_img3;
        ptr_img3+=2;
    }
}
```

Reconstitution, Synthèse :

Inversion directe

Comme vu précédemment, nous avons une image de départ et nous l'avons parcouru par blocs de quatre pixels que nous avons nommé $\begin{pmatrix} x & y \\ z & t \end{pmatrix}$. Cette image a été analysée en quatre images grâce aux filtres décrits ci-dessus :

$$\begin{cases} I_0 = x + y + z + t \\ I_1 = x - y + z - t \\ I_2 = x + y - z - t \\ I_3 = x - y - z + t \end{cases}$$

Pour reconstituer l'image de départ, Il suffit d'inverser le système présenté ici et nous obtenons les formules suivantes :

$$\begin{cases} x = \frac{I_0 + I_1 + (I_2 + I_3)}{4} \\ y = \frac{I_0 + I_2 - (I_1 + I_3)}{4} \\ z = \frac{I_0 + I_1 - (I_2 + I_3)}{4} \\ t = \frac{I_0 + I_3 - (I_1 + I_2)}{4} \end{cases}$$

```
*ptr11 = (val11 + val12 + val21 + val22)>>2;
*ptr12 = (val11 + val21 - (val12 + val22))>>2;
*ptr21 = (val11 + val12 - (val21 + val22))>>2;
*ptr22 = (val11 + val22 - (val21 + val12))>>2;
```

En appliquant ces formules sur les filtres obtenus précédemment, on obtient à gauche l'image d'origine et à droite l'image reconstruite :



Synthèse :

De la même manière que dans l'analyse de l'image, avant de filtrer l'image, nous devons lui appliquer un traitement préalable. En effet, pour éviter les effets de bords et les problèmes aux limites, nous devons insérer une ligne et une colonne supplémentaire à la fin de l'image. Pour cela, nous insérons par miroir (copie du pixel précédent) ces lignes à droite et en bas de l'image.

Après quoi, on sur-échantillonne les quatre images. Nous ajoutons des pixels noirs là où on les avait enlevés précédemment :

```
void sur_ech_v(const cv::Mat_<T>& src, cv::Mat_<T>& dst, bool hf)
{
    const int rows = src.rows;
    const int cols = src.cols;
    const int rowsx2 = rows<<1;
    const int size = cols * sizeof(T);
    dst = cv::Mat_<T>(rowsx2,cols);
    T* ptr_dst = dst[0];
    if(!hf)
    {
        for(int row=0; row<rows; row++)
        {
            memcpy(ptr_dst, src[row], size);
            ptr_dst+=cols;
            memset(ptr_dst,0,size);
            ptr_dst+=cols;
        }
    }
    else
    {
        for(int row=0; row<rows; row++)
        {
            memset(ptr_dst,0,size);
            ptr_dst+=cols;
            memcpy(ptr_dst, src[row], size);
            ptr_dst+=cols;
        }
    }
}
```

Sur-échantillonnage vertical

Ensuite, nous appliquons les filtres :

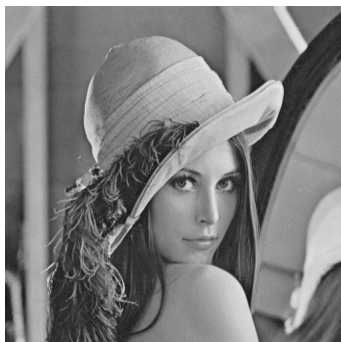
- Filtre basse fréquence (1 1)

```
*ptr_dst+=((*ptr_src1++)+(*ptr_src2++))>>2;
```

- Filtre haute fréquence (1 -1)


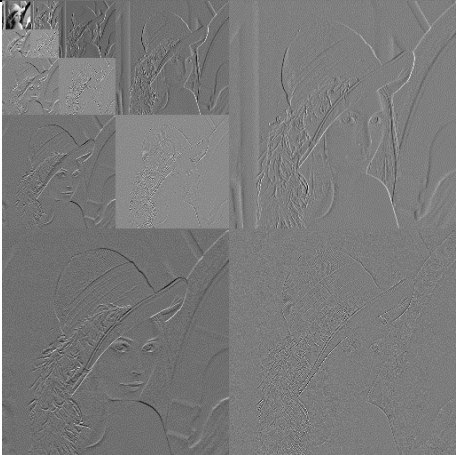
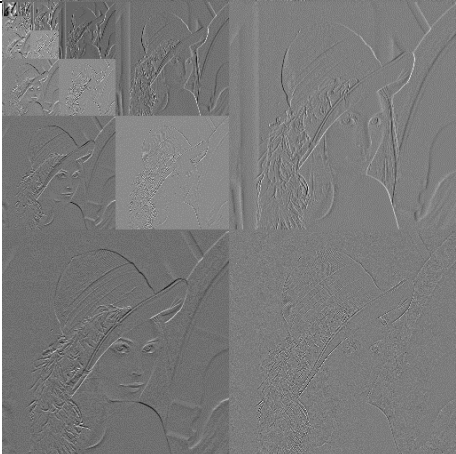
```
*ptr_dst+=((*ptr_src1++)-(*ptr_src2++))>>2;
```

En sommant les résultats obtenus, on obtient l'image d'origine à et à droite l'image reconstruite :



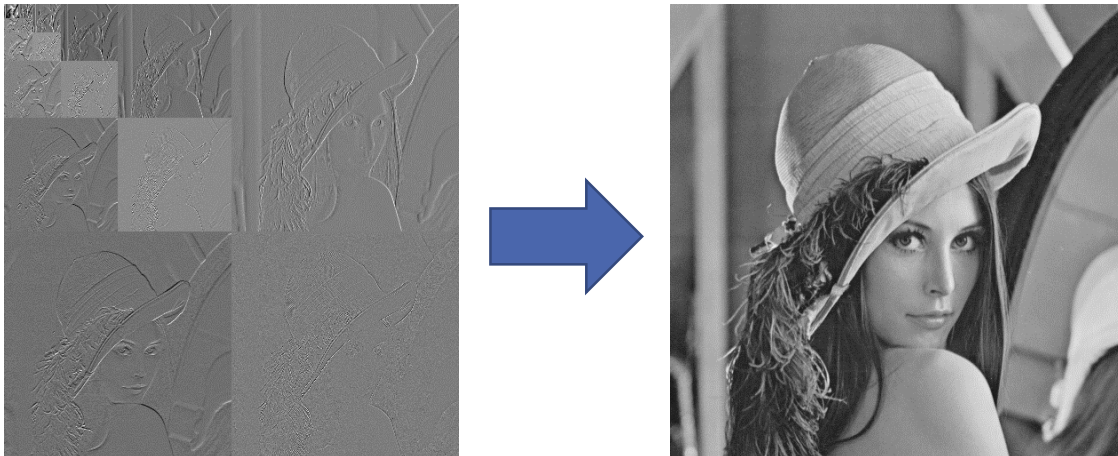
Récurtivité

Analyse

Nombre d'itération	Résultat obtenu	Nombre d'itération	Résultat obtenu
0		3	
1		4	
2		5	

Synthèse

Nous avons appliqué cinq fois notre synthétiseur (méthode directe) sur l'image obtenu après analyse et nous avons eu le résultat suivant :



Pour vérifier la cohérence de nos résultats, nous avons de plus effectué un calcul de distance entre les deux images (norme de la différence) et nous avons constaté que celle-ci était nulle. Et donc nous pouvons dire que nos résultats sont bons.

A droit image originale, à gauche image obtenue après analyse et synthèse :

