# Traitement d'images

## TP2 : Transformée de Fourier et traitements fréquentiels

NGUYEN Van Tho

31 mars 2013

Encadrement : NGUYEN Thi Oanh

## 1 Objective

Ce TP effectue à réaliser la transformée de Fourier des images à l'aide de l'outil openCV et à implémenter quelques traitements fréquentiels.
Dans ce TP, j'implémente le filtre passe-bas, le filtre passe-haut et quelques filtres pour corriger le bruit sinusoïdal :

## 2 Transformée de Fourier

Dans cette partie, le programme prend l'image en argument d'entrée. Le programme transforme cette image en domaine fréquent. La dernier étape, la transformée inverse de Fourier est exécutée pour recréer l'image originale. La commande pour lancer le programme :

```
% ./tp2 <image_name> dft
```

Résultats :

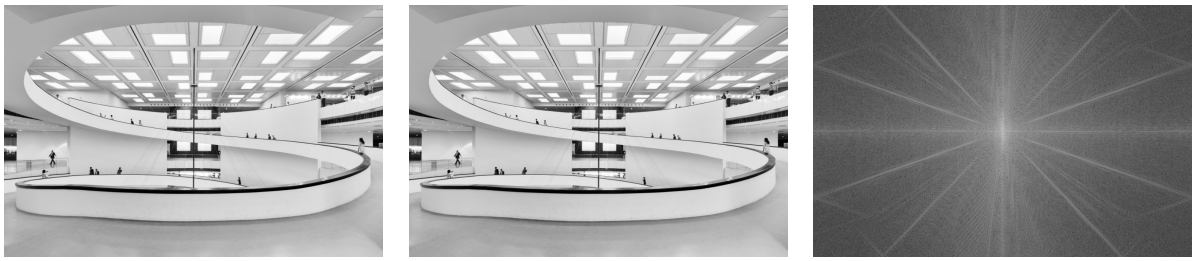

**Figure 1:** *L'image original, le résultat de la transformée inverse et le spectre*

On constate que l'image originale et le résultat de la transformée inverse sont identiques.

Nous allons analyser le spectre de l'image. Au centre du spectre, c'est la composante continue (DC) de l'image qui corresponde à la moitié de niveau de gris de l'image, qui a la fréquence zéro. Les points qui sont proche au centre sont les basses fréquences, Les points qui sont loin du centre sont les hautes fréquences.

On peut voir dans l'image originale, il y a des bords à l'escalier et sur la plafond. À ces bords, le changement de niveau de gris est très fort (haute contraste), celui-ci a la haute fréquence et corresponde aux lignes du centre aux extrémités de l'image.

# 3 Traitements fréquentiels

## 3.1 Filtre passe bas

L'objectif de ce filtre est d'atténuer le composante dont la fréquence haute. En effet, l'image est transformée en domaine fréquent par la transformée de Fourier. Le produit de la transformée de Fourier est un tableau de nombre complexe qui sont présentés par le tableau réel et le tableau imaginaire. Un circle déterminé par l'argument qui est la frontière : les point (de tableau réel et de imaginaire) à l'extérieur du circle sont mis à 0. Après la transformation, une transformation inverse est réalisée pour créer l'image filtrée. Le code pour appliquer le filtre est affiché ci-dessous :

Listing 1: Le filtre passe bas

```
    for(int i = 0; i < radius; i++){
        for(int j = 0; j < radius; j++){
            if(i*i + j*j <= r2){
                lowReal.at<float>(j,i) = imgReal.at<float>(j,i);
                lowImaginary.at<float>(j,i) = imgImaginary.at<float>(j,i)
                    ;

                lowReal.at<float>(j, cols - i - 1) = imgReal.at<float>(j,
                     cols - i - 1);
                lowImaginary.at<float>(j, cols - i - 1) = imgImaginary.at
                    <float>(j, cols - i - 1);

                lowReal.at<float>(rows - j - 1,i) = imgReal.at<float>(
                    rows - j - 1, i);
                lowImaginary.at<float>(rows - j - 1,i) = imgImaginary.at<
                    float>(rows - j - 1, i);

                lowReal.at<float>(rows - j - 1, cols - i - 1) = imgReal.
                    at<float>(rows - j - 1, cols - i - 1);
                lowImaginary.at<float>(rows - j -1, cols - i - 1) =
                    imgImaginary.at<float>(rows - j - 1, cols - i - 1);
            }
        }
    }
   imgReal = lowReal;
   imgImaginary = lowImaginary;
```

### 3.1.1 Expérimentation

La commande pour lancer le programme est affichée ci-dessous. Où radius est un nombre entre 0.0 à 1.0.

```
./tp2 <nom de l'image> lowpass <radius> {préfixe de sortie }
```

J'expérimente plusieurs images avec des différents rayons. Les images ci-dessous sont les résultats :

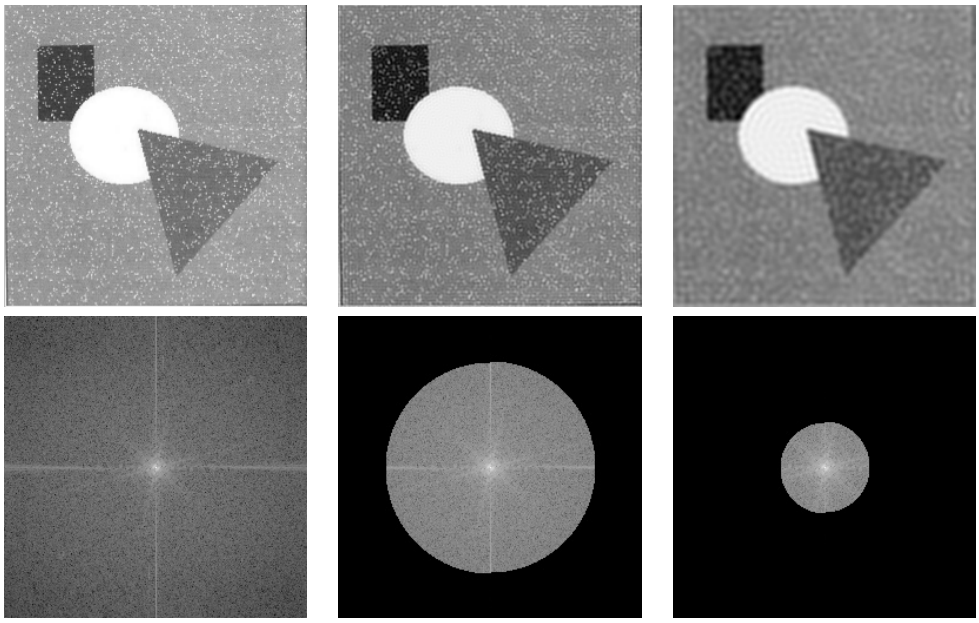

**Figure 2:** *L'image original, le filtre passe bas avec rayon 0.7 et 0.3*

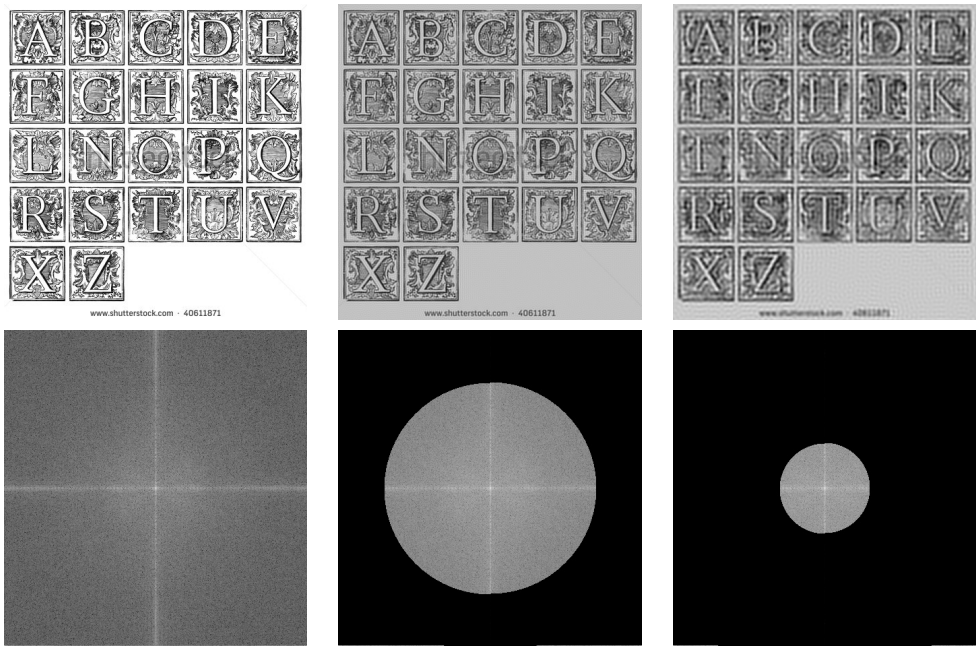

**Figure 3:** *L'image original, le filtre passe bas avec rayon 0.7 et 0.3*

### 3.1.2 Analyse des résultats

À partir des résultats on peut dire que le filtre passe bas lisse l'image, atténue les bruits en haute fréquence. Avec le rayon plus petit, l'image est plus lisse. Cependant, l'image après le traitement se perde quelques détails. Le niveau de la perte est dépendu à la valeur du rayon. Le rayon est plus petit, la perte est plus grande.

## 3.2 Filtre passe haut

L'objectif de ce filtre est d'atténuer les composantes dont la fréquence basse et passer les composantes dont la fréquence haute. Comme le filtre passe bas, on a un cycle pour définir la frontière, mais maintenance, l'intérieur de l'image est mis à 0. Après la transformation, une transformation inverse est réalisée pour créer l'image filtrée. Le code pour appliquer le filtre est affiché ci-dessous :

Listing 2: Le filtre passe haut

```
    for(int i = 0; i <= radius; i++){
        for(int j = 0; j <= radius; j++){
            if(i*i + j*j <= r2){
                imgReal.at<float>(j,i) = 0;
                imgImaginary.at<float>(j,i) = 0;

                imgReal.at<float>(j, cols - i - 1) = 0;
```

```
                imgImaginary.at<float>(j, cols - i - 1) = 0;

                imgReal.at<float>(rows - j - 1, i) = 0;
                imgImaginary.at<float>(rows - j - 1, i) = 0;

                imgReal.at<float>(rows - j - 1, cols - i - 1) = 0;
                imgImaginary.at<float>(rows - j - 1, cols - i - 1) = 0;
            }
        }
    }
```

### 3.2.1 Expérimentation

La commande pour lancer le programme est affichée ci-dessous. Où radius est un nombre entre 0.0 à 1.0.

```
./tp2 <nom de l'image> highpass <radius> {préfixe de sortie }
```

J'expérimente le filtre passe haut avec les images dans la partie ci-dessus. Les images ci-dessous sont les résultats :

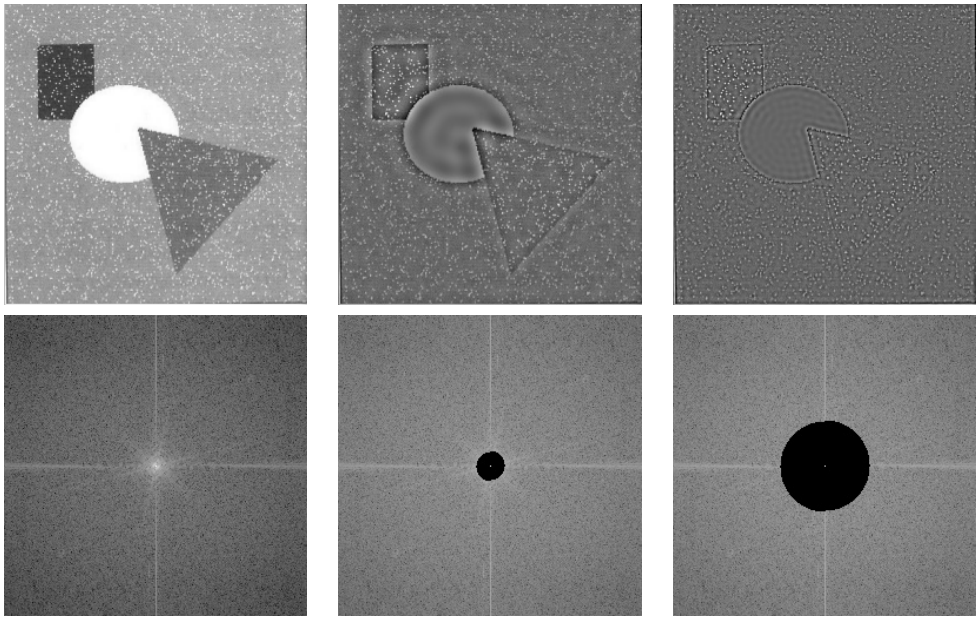

**Figure 4:** *L'image original, le filtre passe haut avec rayon 0.1 et 0.3*

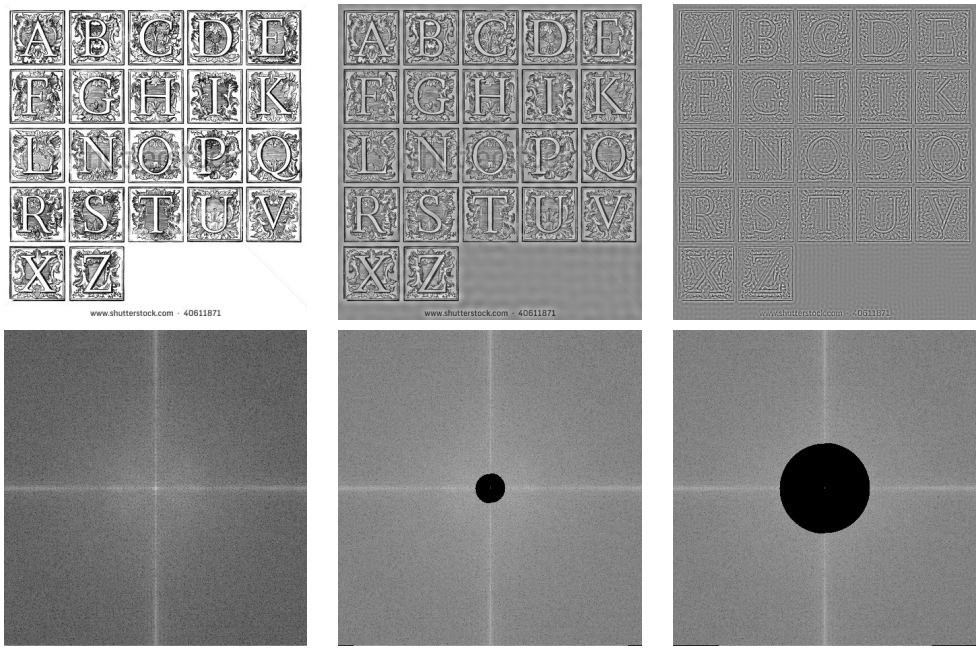

**Figure 5:** *L'image original, le filtre passe haut avec rayon 0.1 et 0.3*

### 3.2.2 Analyse des résultats

Le filtre passe haut a l'effet inverse celui du filtre passe bas. Effectivement, l'image, résultat du filtre est plus aigue (sharp en anglais). Les contours de l'image, qui ont les fréquences hautes sont plus clairs. Avec le rayon plus grand, l'image est plus aigue. Cependant, l'image après le traitement se perde quelques contenus. Le niveau de la perte est dépendu à la valeur du rayon. Le rayon est plus grand, la perte est plus grande.

## 3.3 Filtre du bruit sinusoïdal

Dans cette partie, j'ai utilise plusieurs filtres pour corriger le bruit sinusoïdal.

### 3.3.1 Filtre coupe-bande

Ce filtre est l'ensemble du filtre passe haut et du filtre passe bas. Les composantes qui sont entre deux cycles sont atténuées.

### 3.3.2 Filtre Butterworth

Le filtre est calculé par ce formule :

$$H(u,v) = 1 - 1/(1 + (D - D_0)^2/(w/2)^2)$$

Listing 3: Le filtre Butterworth

```
float h = getRadius(cols, rows, high);
float l = getRadius(cols, rows, low);
float high2 = h*h;
float low2 = l*l;
for(int i = 0; i < cols; i++){
    for(int j = 0; j < rows; j++){
        int dj = (j < rows / 2) ? j : rows - j;
        int di = (i < cols / 2) ? i : cols - i;
        float dist2 = dj*dj + di*di;
        if(dist2 > low2 && dist2 < high2){
            imgReal.at<float>(j,i) = 0;
            imgImaginary.at<float>(j,i) = 0;
        }
    }
}
```

### 3.3.3 Un autre filtre du bruit sinusoïdal

Dans ce filtre, j'ai développé un algorithme pour trouver les points plus clairs dans le spectre (sauf le point DC). Après ces points sont trouvés, des petits cycles noires sont créés autour ces points.

### 3.3.4 Expérimentation

Les commande pour lancer le programme sont affichées ci-dessous.

```
./tp2 <nom de l'image> bandreject <radius1> <radius2>
./tp2 <nom de l'image> butterworth <radius1> <radius2> <puissance>
./tp2 <nom de l'image> sin
```

J'expérimente ces filtre avec les images qui ont le bruit sinusoïdal. Les images ci-dessous sont les résultats :

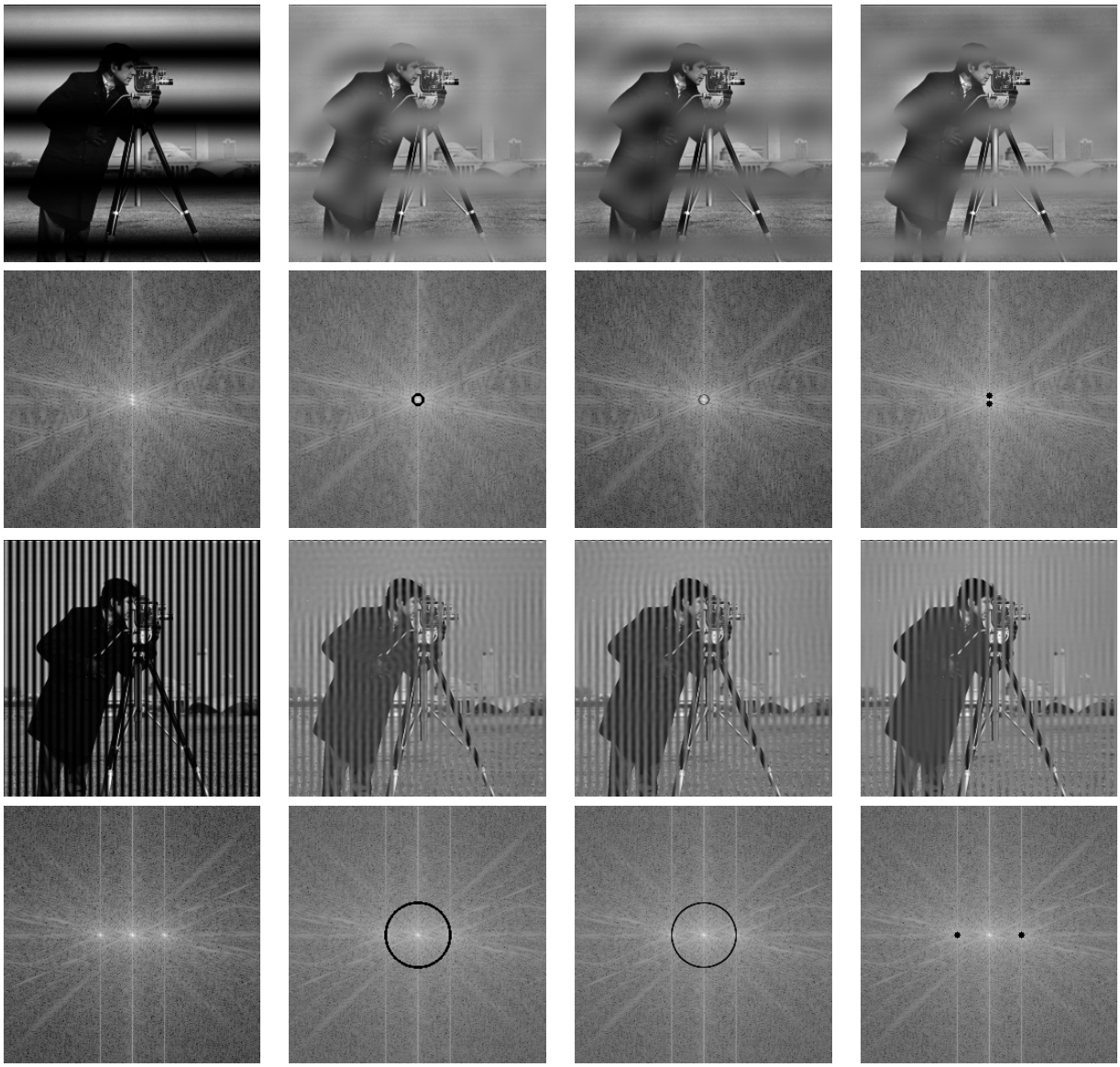

**Figure 6:** *L'image original, le filtre coupe-bande, le filtre Butterworth, et un autre filtre sinusoïdal*

### 3.3.5 Analyse des résultats

Ces filtres peuvent filtrer le bruit sinusoïdal. Cependant le résultat est dépendu de la fréquence du bruit. Dans le premier cas, la fréquence du bruit est très bas, donc quand on applique les filtres à l'image, quelques contenus de l'image sont aussi atténués. Dans le deuxième cas, la fréquence du bruit est plus grande. Le résultat est donc meilleur car le contenu de l'image ont des fréquences plus basses. Dans tout cas, on ne peut pas totalement filtrer le bruit sinusoïdal.

# 4 Conclusions

Dans ce TP j'ai implémenté les fonctions pour calculer la transformée de Fourier et la transformée inverse de Fourier.
Pour traiter des images en domaine fréquent j'ai développé des filtres : le filtre passe bas, le filtre passe haut, le filtre coupe bande, le filtre Butterworth...
Le filtre passe bas peut lisser des images, cependant l'image perde quelques détails, les contours de l'image est plus flous.
Le filtre passe haut peut filtre quelques bruits dont la fréquence basse et pour éclairer les contours. Cependant l'image peut perdre son contenu.
Les filtres sinusoïdal que j'ai implémenté peut atténuer le bruit sinusoïdal à certaine fréquence. Si la fréquence du bruit est plus haute, le résultat est plus bon. Pourtant, ces algorithmes ne peuvent pas totalement atténuer le bruit, donc il faut les améliorer. Cela reste comme un travail futur.