Dans tout le TP, on considère des images numériques $a[i,j]$ de taille $m \times n$ (i.e. $i \in \{0, 2, \ldots, m-1\}$, $j \in \{0, 2, \ldots, n-1\}$), à 256 niveaux de gris (i.e. $a[i,j] \in \{0, 1, ..., 255\}$). On supposera que m et n sont pairs. Commencez par récupérer les images et les programmes comme d'habitude pour les TP d'OASIS (voir site pédagogique).

Vous pouvez aussi utiliser vos propres images pour le TP. Evitez dans un premier temps les images *carte.tif*, *montreuil.tif*, *strawmat.tif*, *skin.tif* pour des raisons qui apparaîtront claires plus loin.

Pour lire une image sous python (OASIS), tapez

```
a=np.float32(skio.imread('image.tif'))
```

la variable a est alors une matrice de valeurs de type float mais allant de 0 à 255 [1]. Si vous utilisez une image couleur, veillez à la convertir en niveaux de gris en tapant `a=a.mean(axis=2)`. Visualisez a comme une image (les valeurs correspondent à des niveaux de gris, 0 est le noir et 255 le blanc) :

```
affiche(a) ; #affiche une image dans le logiciel GIMP
```

Dans la suite, il vous est demandé de répondre aux questions en caractères gras suivant la lettre Q. Le compte-rendu est à rendre à vos encadrants à la fin de la séance.

1 Observation du spectre d'images synthétiques

Dans cette partie, on observe et explique les spectres d'images synthétiques simples.

Sinusoïde horizontale

```
x= np.arange(0,256).reshape((-1,1)); y= 2*pi*(8/256)*np.ones((1,256));
a= sin(x*y) # @ marcherait aussi, ici on utilise le broadcast de python
```

Puis visualisez son spectre (on pourra écrire une fonction python qui le spectre d'une image) :

```
A=abs(fft2(a)); A=fftshift(A); affiche(A); # fft2 calcule la TFD bi-dimensionnelle
```

Remarque : la fonction *fftshift* sert a placer les basses fréquences au centre de l'image. Dans le cas d'images de taille paire, elle effectue une permutation entre le quart haut-gauche et le quart bas-droit de l'image, ainsi qu'une permutation entre quarts haut-droit et bas-gauche.

Visualisez le spectre dans les cas suivants :

Sinusoïde verticale

1. Beaucoup d'opérations sont interdites sur les variables de type **int8** (par exemple fft2 ne fonctionne pas dessus). C'est pourquoi nous avons ajouté la commande np.float32 pour avoir un tableau de flottants dans a et non pas un tableau de **int8**.

```
xx= 2*pi*(24/256)*np.ones((256,1)); yy= np.arange(0,256).reshape((1,-1));
a= sin(xx*yy);
```

Variante

```
x3=2*pi*((24+1/2)/256)*np.ones((256,1));
a=sin(x3*yy);
```

Sinusoïde diagonale

```
a=sin(x*y+xx*yy);
```

Q 1.1 : Dans tous les cas, expliquez avec précision les caractéristiques du spectre observé : nombre de points lumineux, disposition, distance entre les points. Pour cela on visualisera l'image d'origine et on utilisera le curseur de gimp pour connaître les positions des points lumineux.

On s'intéresse ensuite à des images synthétiques contenant des "contours" :

Un carré

```
a=np.zeros((256,256));a[62:191,62:191]=1;
```

Une sinusoïde fenêtrée

En partant du carré précédent, et des indices x et y précédemment définis

```
a=a*sin(x*y);
```

Q 1.2 : Expliquez les deux spectres observés.

2 Echantillonnage et transformée de Fourier

À présent, on observe le spectre d'images naturelles. Chargez l'image *maison.tif* dans la variable a puis visualisez le logarithme de son spectre.

```
A=fft2(a); A=fftshift(A);
affiche(log(0.1+abs(A)));# la constante 0.1 est là pour éviter log(0)
```

La fonction logarithme permet de visualiser une TFD, même lorsque la dynamique de la TFD est très grande.

Q 2.1 : A quoi correspondent les structures horizontales, verticales et obliques observées dans la TFD de cette image ?

Répétez l'expérience avec l'image *carte.gif* ou *montreuil.tif*.

Q 2.2 : Quelles structures supplémentaires observez-vous ?

Sous-echantillonnez l'image a dans le cas de l'image *carte.gif*, d'un facteur 2 (i.e. gardez un pixel sur 2 dans chaque direction).

```
(m,n)=a.shape
b=a[0:m:2,0:n:2];
affiche(b);
```

Q 2.3 : Expliquez précisément comment se caractérise le phénomène de repliement de spectre sur cette image.

Effectuez la même expérience en filtrant préalablement l'image, de manière à atténuer ses hautes fréquences. Par exemple,

```
(m,n)=a.shape
A=fftshift(fft2(a))
B=np.zeros((m,n),dtype=np.complex128);
B[m//4:3*m//4+1,n//4:3*n//4+1]=A[m//4:3*m//4+1,n//4:3*n//4+1]
b=ifft2(ifftshift(B));
affiche(real(b));
```

Remarque : bien que la transformée inverse de B soit réelle, on utilise la fonction *real* pour convertir le type de b (en raison d'erreurs d'arrondi, une partie imaginaire très faible apparaît).

Q 2.4 : À quelle fréquence de coupure a opéré notre filtrage passe bas ? Le résultat est-il meilleur ? pourquoi ? Quel autre type de défaut est apparu ?

On peut se débarrasser de ce défaut en utilisant un filtrage moyenneur avant de sous-échantillonner (au lieu du filtre passe-bas parfait) :

```
c=conv2(a,np.ones((3,3))/9,'same'); #conv2 effectue la convolution bidimensionnelle
affiche(c[::2,::2])
```

3 Importance de la phase de la TFD selon le type d'image

Ici nous allons illustrer le fait que, suivant le type d'image, c'est plutôt la phase ou le module d'une image qui porte l'information la plus pertinente. On se donne un couple d'images (de même taille) a et b et on échange leurs phases de TFD par la suite d'instructions suivantes :

```
A=fft2(a); B=fft2(b);
Ap=exp(i*angle(A))*abs(B); #module de la TFD de b et phase de A
Bp=exp(i*angle(B))*abs(A);
ap=real(ifft2(Ap)); affiche(ap,titre='module_b_phase_a');
bp=real(ifft2(Bp)); affiche(bp,titre='module_a_phase_b');
```

Faites cette expérience pour deux couples d'images :
— a est l'image chapeau et b l'image maison.
— a est l'image papierpeint et b est l'image bois.

Q 3.1 : Quelles conclusions tirez-vous sur la pertinence de la phase et du module suivant que l'image est une texture ou géométrique ?

Selon le même principe, nous pouvons développer une méthode de synthèse de texture. Le but de la synthèse de texture est de générer à volonté des images qui aient le même aspect visuel qu'une texture exemple donnée, mais qui ne soient pas de simples copies de l'exemple. De telles méthodes ont de nombreuses applications dans le domaine des environnements virtuels ou des jeux vidéo.

A partir d'une texture exemple a, on peut très simplement obtenir une nouvelle texture b en lui imposant le spectre de a et les phases d'un bruit blanc gaussien [2]. C'est à dire

```
a=charge_image('bois.tif')
(m,n)=a.shape
b=np.random.randn(m,n)
A=fft2(a)
B=fft2(b)
C=abs(A)*exp(i*angle(B))
c=real(ifft2(C))
affiche(c, titre='synthese')
affiche(a,titre='originale')
```

Si vous utilisez une autre réalisation de bruit blanc, vous obtiendrez une image visuellement similaire mais non identique à la première. Vous pouvez tester une version en ligne d'une approche semblable de la synthèse ici : <http://www.ipol.im/pub/art/2011/ggm_rpn/>

4 Filtrage passe-haut et images

Les contours d'une image, c'est à dire la trace des bords des objets, est l'une des informations les plus utilisées par notre système visuel, et l'extraction de ces contours est une opération de base du traitement des images. Ces contours correspondent à des discontinuités et une méthode simple mais illustrative pour les mettre en évidence consiste à appliquer les filtres suivants, passe-haut :

```
fx=np.asarray([[0 ,0, 0],[-1 ,1, 0],[ 0 ,0, 0]])
fy=fx.T
gx=conv2(a,fx) # conv2 effectue une convolution bi-dimensionnelle
gy=conv2(a,fy)
g=(gx**2+gy**2)**0.5
affiche(g)
```

l'image g est la une approximation discrète du module du gradient d'une image.

2. un bruit blanc est une image obtenue en tirant une variable aléatoire en chaque pixel, indépendamment d'un pixel à l'autre

Appliquez cette procédure à plusieurs images et observez le résultat. L'inconvénient d'une telle méthode est sa forte sensibilité au bruit et aux textures des images. On procède donc en général à une régularisation préalable de l'image. Effectuez l'opération précédente après avoir convolué l'image par le filtre **fm** (filtre moyenneur), comme suit :

```
fm=np.ones((3,3))/9 ; b=conv2(a,fm) ;gx=conv2(b,fx) ;gy=conv2(b,fy) ;
```

Q 4.1 : Montrez que le filtrage proposé est équivalent à l'application de `gx=conv2(a,hx) ; gy=conv2(a,hy) ;` **pour deux RI hx et hy que l'n précisera.**

Unsharp Masking Les photographes qui traitent leurs images sur ordinateur utilisent le terme "Unsharp Masking" pour désigner un filtre qui permet de donner plus de netteté à une image. C'est un filtre très répandu et très utilisé. Schématiquement, un flou consiste en un étalement de la lumière en une tache. Une modélisation possible du flou est l'évolution de l'image comme le ferait une distribution de température suivant l'équation de la chaleur, c'est-à-dire qu'à chaque pas de temps l'image, pour être floutée, se voit augmentée une proportion de son laplacien. Le filtre Unsharp correspond au renversement de cette évolution en retranchant à l'image une portion de son laplacien.

```
a=charge_image('flou.jpg');
laplace=np.ones((3,3)); laplace[::2,::2]=0; laplace[1,1]=-4; #filtre laplacien discret
dirac=np.zeros((3,3)); dirac[1,1]=1; #un dirac sur un support 3x3
alpha=3#taux de mélange
b=conv2(a,dirac-alpha*laplace);
affiche(a,titre='orig',normalise=False);
affiche(b,titre='net',normalise=False); #certaines valeurs vont dépasser 255 car la RI n'est
```

Vous pouvez augmenter le taux de mélange pour obtenir des résultats plus nets, mais une limite du procédé est le réhaussement indésirable du bruit.

5 Super-résolution

La super-résolution est une technique qui permet de reconstruire une image de haute résolution à partir d'images basses résolution. Nous en présentons une version simplifiée sous forme de tutoriel. La première étape est la modélisation de l'appareil photographique.

Q5.1 : Examinez la fonction appareil_photo et dire ce qu'elle fait ? À quoi correspondent les paramètres dx et dy ? Si on note u l'entrée de la fonction et v la sortie, Π la fonction qui sous-échantillonne une image de 2 en 2. $T(u, dy, dx)$ la fonction qui translate une image de dy, dx. Exprimer v en fonction u en utilisant ces opérateurs (et la convolution par un noyau qu'il faut déterminer).

On utilise **appareil_photo** pour prendre plusieurs images d'une scène inconnue. Le but du reste de ce TP est de trouver cette image. Vous disposez pour cela dans le fichier **donnees_super.pickle**[3] de divers sorties du programme **appareil_photo**. La taille d'un pixel d'appareil photographique est de l'ordre de $10\mu m$ et de ce fait il est impossible de contrôler

3. La manière de le charger est dans le scratch.

précisément la position d'une photographie par rapport à une autre. On est obligé de trouver le décalage entre les photographies après les avoir prises.

Q5.2 : Examinez la fonction trouve_decalage et donner la formule mathématique de la fonction dont elle renvoie la position du maximum (en fonction des deux images zoomées im1 et im2) .

Q5.3 : Examinez la fonction recalage qui crée une grande image à partir des images acquises *capte1...4* **sous forme d'une mosaïque. Après avoir lancé trouve_decalage(capte[1.. donner la formule mathématique qui lie la sortie de recalage avec l'image de départ.**

Visualisez la sortie de **recalage**. Arrivez-vous à lire le texte qui se trouve en haut à gauche ? Utilisez la fonction **defloute** avec la sortie de recalage et en visualiser le résultat (attention défloute a besoin de modification par le résultat de Q5.1).