

Yohann Tendero¹

http://perso.telecom-paristech.fr/~ytendero/igr_201.html

- ▶ Illustrations du traitement d'image ;

Example 1 : Contrast



Example 1 : Contrast



Example 2 : Blur



Example 2 : Blur



Example 3 : Inpainting



Example 3 : Inpainting



Example 4 : Denoising



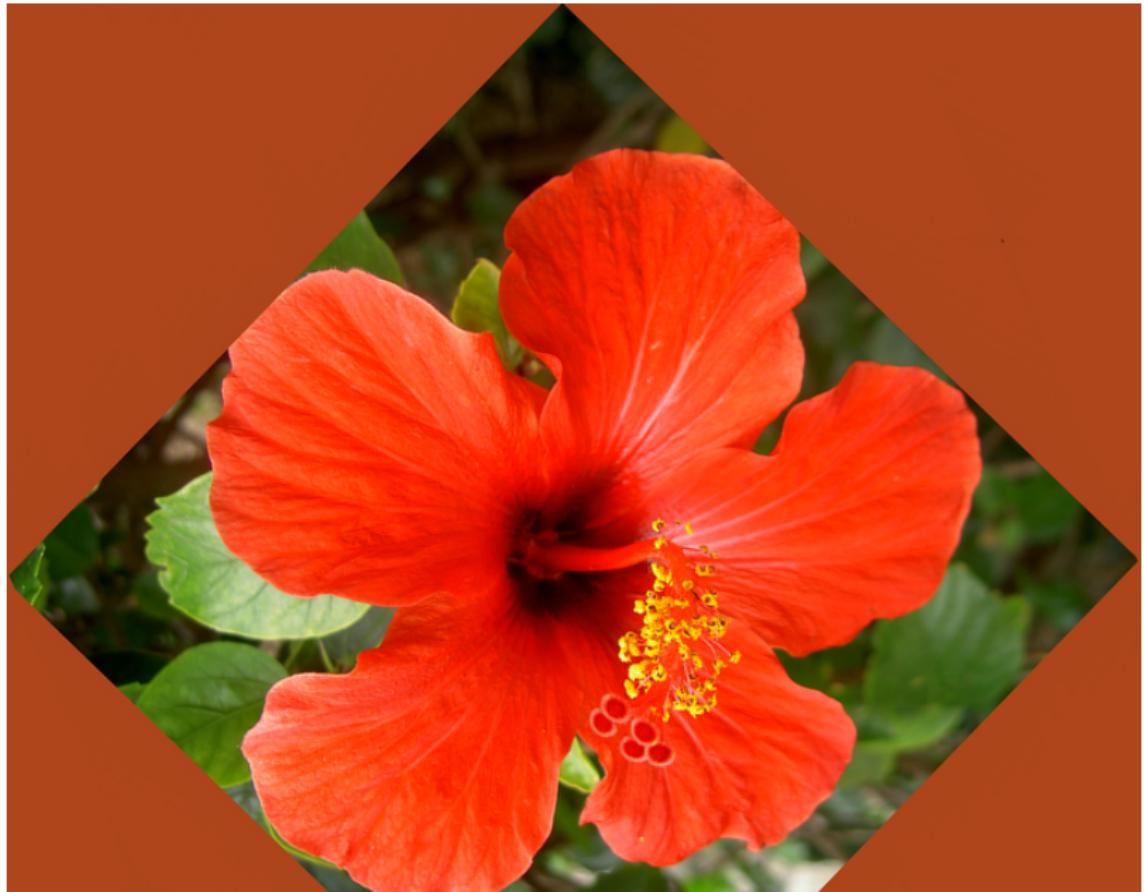
Example 4 : Denoising



Example 5 : Rotation



Example 5 : Rotation

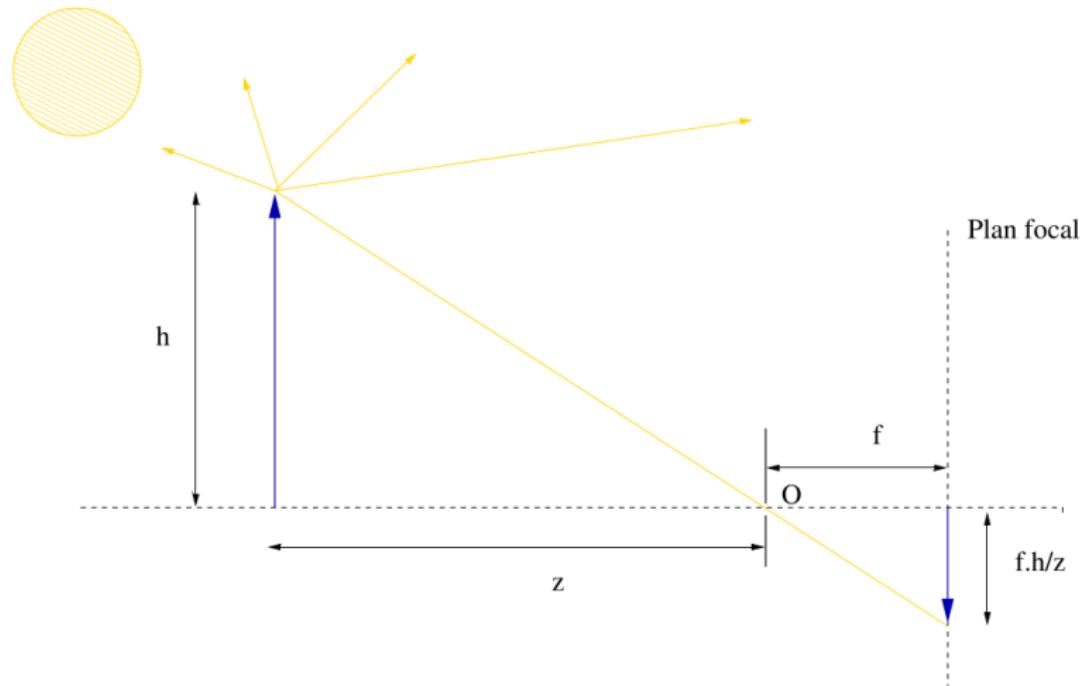


- ▶ Courte introduction au plus simple des modèles de formation d'images (sténopé ou pinhole en anglais)

Nicéphore Niépce, plus ancienne photo conservée, 1827.



Pinhole (ou *camera obscura*), 1D



- ▶ Photo \approx une projection de la scène 3D sur un capteur 2D.
Points cachés ; **occlusion**.
- ▶ Sur le plan focal se trouve un maillage de photo-récepteurs : le "capteur"
- ▶ Photo-récepteur \approx compteur de photons
- ▶ Le photo-récepteur (i,j) estime $u(i,j)$: l'intensité lumineuse au pixel (i,j) qui est stockée.

Théorème fondamental de la photographie

Nature quantique de l'émission photonique : photons émis comme suivant une variable aléatoire de Poisson, $\lambda(i,j)$ l'intensité d'émission photonique (photons/secondes) à la position (i,j) sur le capteur.

Theorem

Soit $\Delta t > 0$ le temps d'exposition, et $\lambda(i,j) > 0$ l'émission photonique au pixel (i,j) . La valeur du pixel peut être n'importe quelle réalisation de

$$u(i,j) \sim \frac{P\left(\int_0^{\Delta t} \lambda(i,j) dt\right)}{\Delta t} 2.$$

2. La notation $X \sim Y$ signifie a pour loi.

Théorème fondamental de la photographie

Nature quantique de l'émission photonique : photons émis comme suivant une variable aléatoire de Poisson, $\lambda(i,j)$ l'intensité d'émission photonique (photons/secondes) à la position (i,j) sur le capteur.

Theorem

Soit $\Delta t > 0$ le temps d'exposition, et $\lambda(i,j) > 0$ l'émission photonique au pixel (i,j) . La valeur du pixel peut être n'importe quelle réalisation de

$$u(i,j) \sim \frac{P\left(\int_0^{\Delta t} \lambda(i,j) dt\right)}{\Delta t}^2. \text{ On a } \mathbb{E}(u(i,j)) = \lambda(i,j) \text{ et}$$

$\text{var}(u(i,j)) = \frac{\lambda(i,j)}{\Delta t} \xrightarrow{\Delta t \rightarrow +\infty} 0$. Le rapport signal à bruit est

$$\frac{\mathbb{E}(u(i,j))}{\sqrt{\text{var}(u(i,j))}} = \sqrt{\lambda(i,j)\Delta t}.$$

2. La notation $X \sim Y$ signifie a pour loi.

Critiques

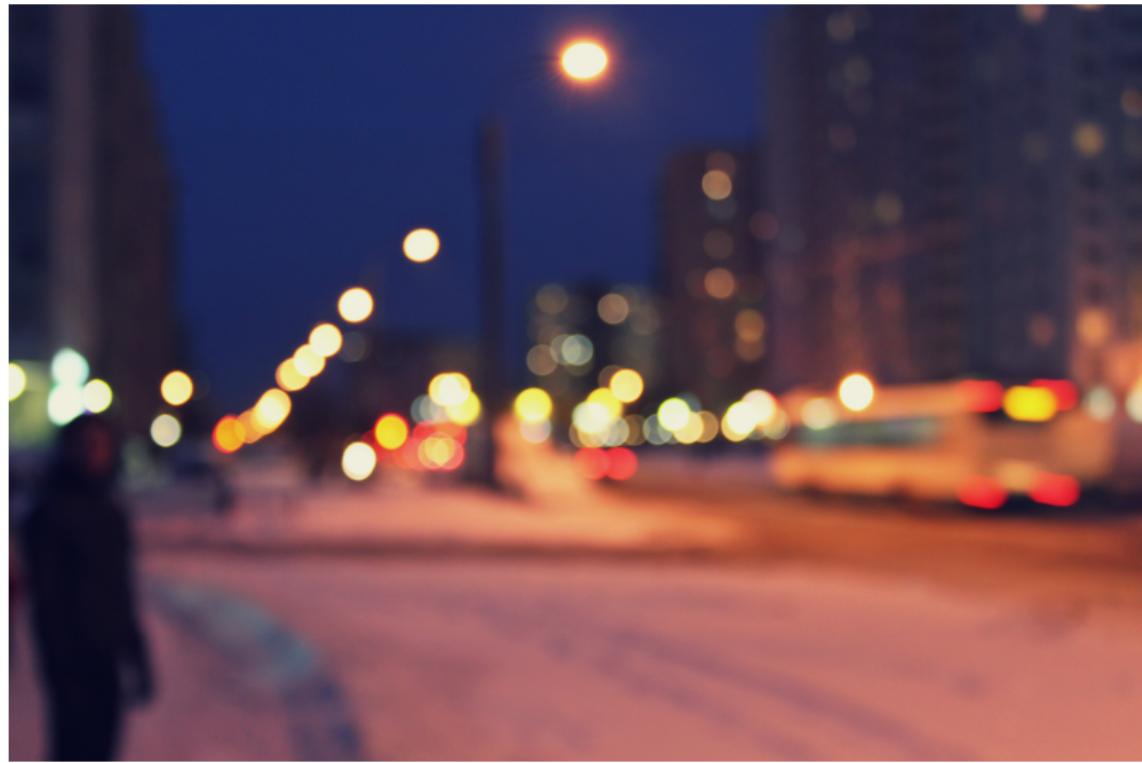
Nous avons négligé

- ▶ l'ouverture (diamètre de la lentille) => Du flou, des distorsions géométriques (lignes droites sont incurvées), vignettage ("coins sombres")
- ▶ La quantification ($u(i,j)$ peut être dans $[0, +\infty)$, pour l'enregistrer il faut le quantifier p. ex dans $\{0, \dots, 255\}$. Peut être vue comme une source de bruit.)
- ▶ Le démosaickage (appareil classique : observe 1 seule des 3 couleurs pour (i,j))
- ▶ L'effet du capteur (échantillonnage)

Ne pas oublier :

- ▶ Le "bruit" ; une image observée est toujours bruitée. Pour tester un algorithme, p. ex., de défloutage ajoutez du bruit Gaussien à votre image avec d'appliquer la méthode pour avoir une idée de sa robustesse au bruit.

Flous



Distorsions géométriques



Vignettage



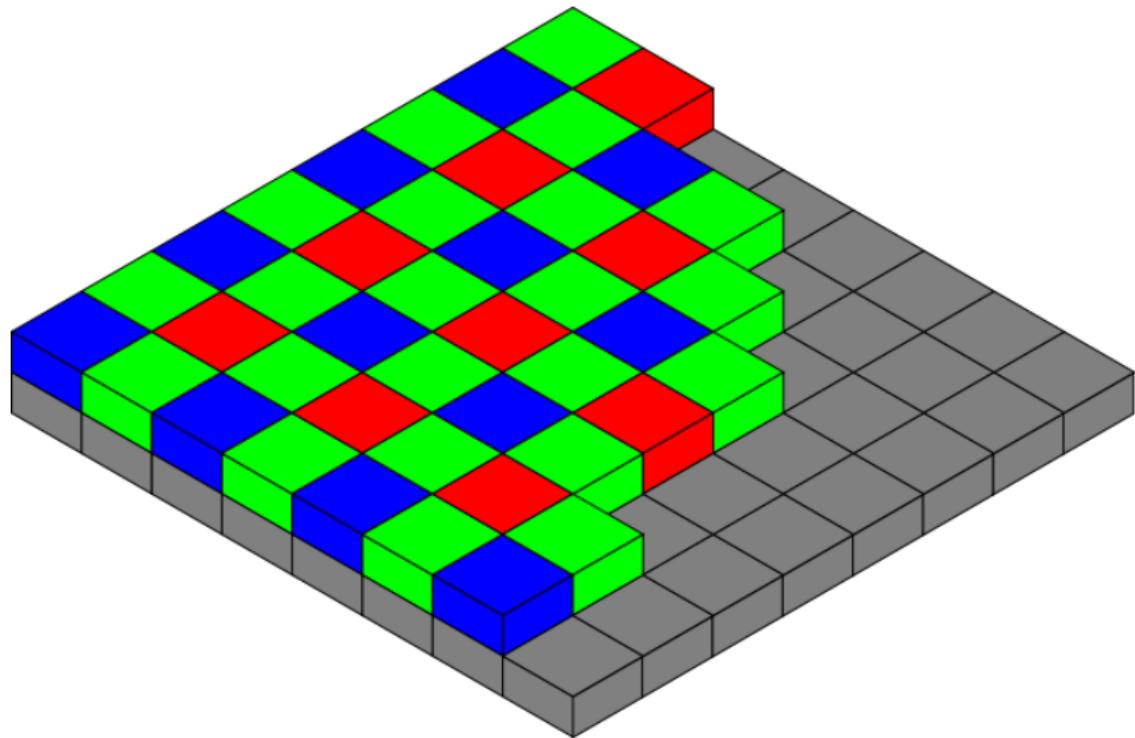
Quantification



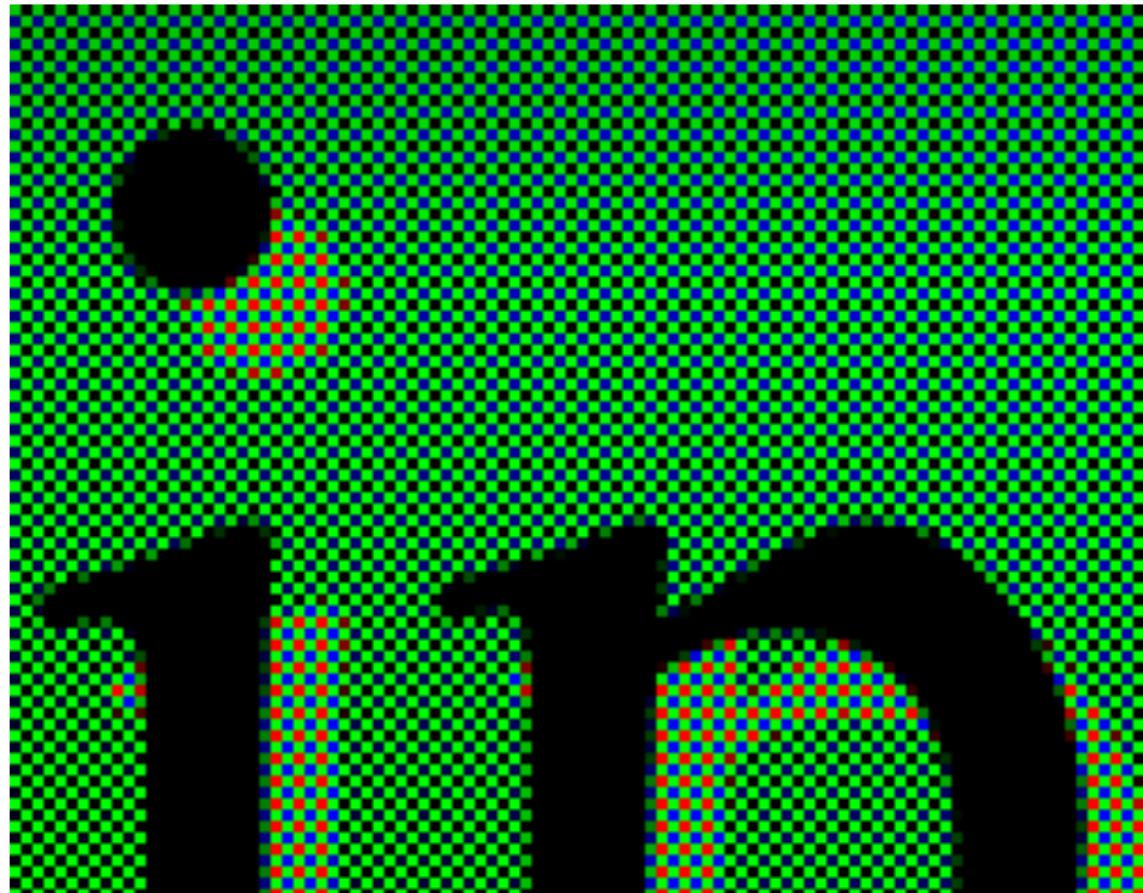
Quantification



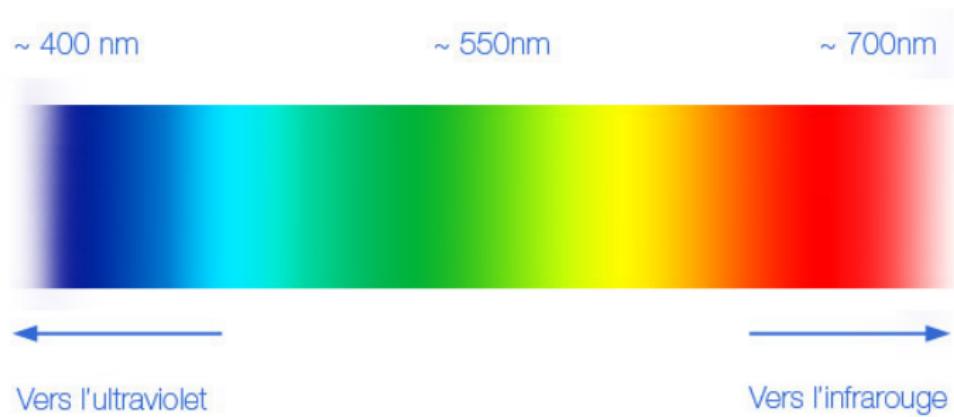
Demosaickage



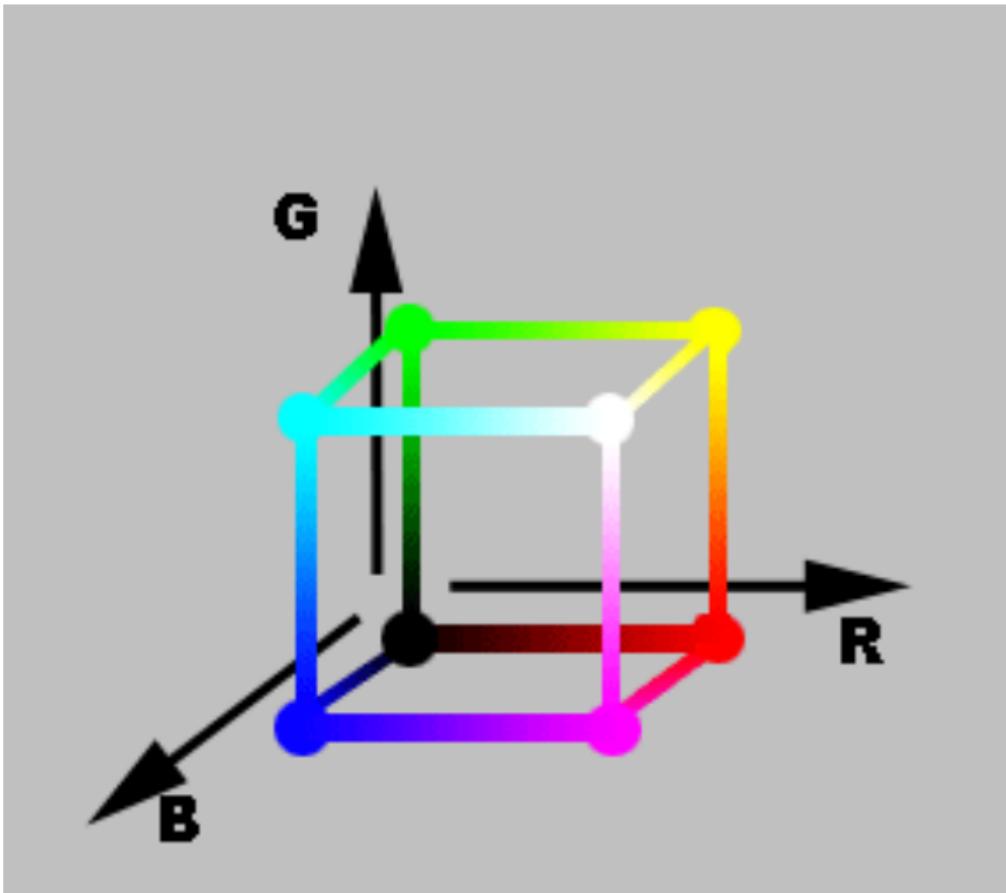
Demosaickage



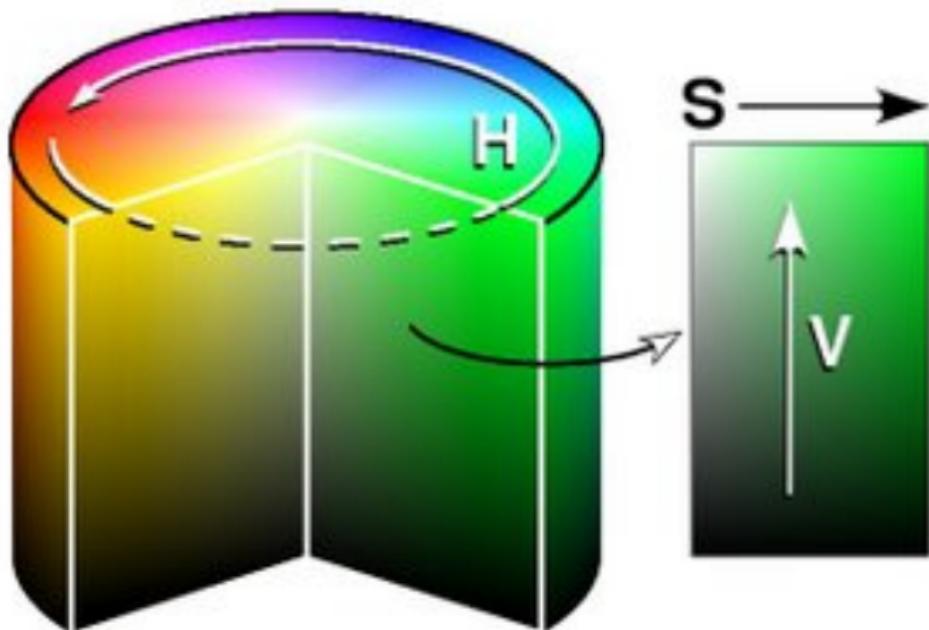
Espaces couleurs



Espaces couleurs : RGB



Espaces couleurs : HSV



Espaces couleurs

$$H = \begin{cases} \text{undefined}, & \text{si } MAX = MIN \\ 60 \times \frac{G-B}{MAX-MIN} + 0, & \text{si } MAX = R \\ & \text{et } G \geq B \\ 60 \times \frac{G-B}{MAX-MIN} + 360, & \text{si } MAX = R \\ & \text{et } G < B \\ 60 \times \frac{B-R}{MAX-MIN} + 120, & \text{si } MAX = G \\ 60 \times \frac{R-G}{MAX-MIN} + 240, & \text{si } MAX = B \end{cases}$$

$H \in [0, 360]$

$$S = \begin{cases} 0, & \text{si } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{sinon} \end{cases}$$

$S, V, R, G, B \in [0, 1]$

$MAX = MAX(R, G, B)$

$MIN = MIN(R, G, B)$

$$V = MAX$$

Espaces couleurs

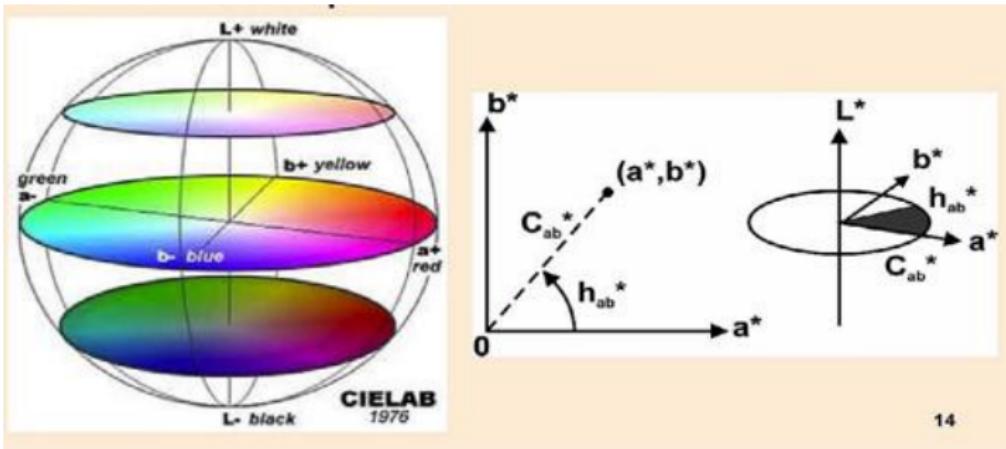


Gauche : image originale

Centre : diminution de 20% de la saturation

Droite : augmentation de 20% de la saturation

Espaces couleurs : LAB



14.

Shannon-Whittaker : comment échantillonner une image ?

On dit que f est à bande-limité si ($f \in L^1 \cup L^2$) satisfait : $\hat{f}(\xi) = 0$ pour tous ξ t.q. $|\xi| > K$.

Soit $f \in L^1 [-\pi, \pi]$ bande limitée. Alors on a $f(x) = \sum_{n=-\infty}^{+\infty} f(n)\text{sinc}(x - n)$ pour tout $x \in \mathbb{R}$

avec $\text{sinc}(x) := \frac{\sin(\pi x)}{\pi x}$ On observe des échantillons $\dots, f(-1), f(0), f(1), \dots$ (ie. une suite) et la formule nous dit comment récupérer toute la fonction f . En d'autres termes de $f(n) = g(n)$ pour tout $n \in \mathbb{Z}$ on déduit que $f = g$.

Pour une image cela donne : (on doit alors supposer que \hat{f} est supporté par $[-\pi, \pi]^2$)

$$f(x, y) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} f(m, n)\text{sinc}(x - m)\text{sinc}(y - n) \quad \forall (x, y) \in \mathbb{R}^2$$

Discussion autour de la validité pratique, de la chaîne d'échantillonnage idéale, de comment obtenir la coupure fréquentielle.

- convolution discrète, périodique et filtres
- convolution et TFD : TFD, filtrage linéaire. Formules classiques (translations, zoom, convolution et Fourier, notion de fréquence dans une image)
- utilisation de la FFT avec Matlab et quelques applications : filtre passe-haut, passe-bas, réhaussement.
- réduction des artefacts dûs aux effets de bords avec la FFT (TCD implicite)

Le **filtrage linéaire** correspond à convoler le **signal** observé u par une fonction, appelée **réponse impulsionnelle** g .

Il est donc crucial de savoir comment calculer efficacement des convolutions. On distinguera deux types de convolutions discrètes

1 la convolution de signaux définis sur \mathbb{Z} ou \mathbb{Z}^2 (formellement)

$$u * g(n) = \sum_{m \in \mathbb{Z}} u(n - m)g(m)$$

$$u * g(m, n) = \sum_{k \in \mathbb{Z}} \sum_{l \in \mathbb{Z}} u(m - k, n - l)g(k, l)$$

Le **filtrage linéaire** correspond à convoler le **signal** observé u par une fonction, appelée **réponse impulsionnelle** g .

Il est donc crucial de savoir comment calculer efficacement des convolutions. On distinguera deux types de convolutions discrètes

1 la convolution de signaux définis sur \mathbb{Z} ou \mathbb{Z}^2 (formellement)

$$u * g(n) = \sum_{m \in \mathbb{Z}} u(n - m)g(m)$$

$$u * g(m, n) = \sum_{k \in \mathbb{Z}} \sum_{l \in \mathbb{Z}} u(m - k, n - l)g(k, l)$$

Pour des signaux discrets finis, ces formules demandent d'étendre suffisamment u et/ou g sinon le résultat n'est **pas défini aux bords**.

De plus, dès que le support de g est un peu grand cela devient très long à calculer. (Le support de u est en général très grand.)

En matlab ce sont les commandes **conv**, **conv2**. Les paramètres permettent de choisir les conditions aux bords. Si on choisit des conditions périodiques cela devient une **convolution circulaire**.

2 la convolution de signaux périodiques (dite circulaire pour les signaux discrets).

On suppose que u et g sont définis sur $\{0, \dots, M - 1\}$.
(Discussion sur g défini que $\{0, \dots, N - 1\}$ et $N < M$.)

$$u *_{per} g(n) = \sum_{m=0}^{M-1} u((n - m) \text{mod}(M))g(m)$$

En matlab c'est la commande `cconv` (pour circular convolution).
(Pour une raison que j'ignore `cconv2` ne semble pas exister pas en matlab.)

Rappel : g s'appelle réponse impulsionnelle. \hat{g} est la réponse fréquentielle.
Il y a un moyen rapide de calculer les convolutions circulaires.

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \dots, N - 1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \dots, N - 1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \dots, N - 1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc. La suite u est implicitement vue comme un **polynôme trigonométrique**.

Transformée de Fourier Discrète (TFD)

Soit u une suite finie définie sur $\{0, \dots, N - 1\}$. La TFD (ou DFT en anglais) de u est

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \dots, N - 1\}.$$

On a (formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \dots, N - 1\}. \quad (1)$$

Si on applique (1) pour $n = N$ on retrouve $u(0)$, etc. La suite u est implicitement vue comme un **polynôme trigonométrique**. Elle est se prolonge naturellement à tout \mathbb{Z} par périodicité. On a en plus (Formule convolution/TFD)

$$\widehat{u * g}_{per} = \hat{u}\hat{g}.$$

Les transformée de Fourier discrètes (TFD) se calculent rapidement (en $N \log(N)$), par les algorithmes de "Fast Fourier Transform" (FFT).

Commandes matlab :

- `fft` (`fft2` pour la 2D) pour la TFD (2D-DFT)
- `ifft` (`ifft2` pour la 2D) pour la TFD inverse (2D-DFT inverse)
- Les commandes `fftshift/ifftshift` pour ré-ordonner (i pour la transformation inverse) les termes, si besoin.)

Il faut se souvenir de (Formule convolution/TFD)

$$\widehat{u}_{per} * \widehat{g} = \widehat{u}\widehat{g}.$$

Algorithme pour calculer une convolution périodique :

On se souvient que $DFT(u *_{per} g)(k) = DFT(u)(k)DFT(g)(k)$, pour tout k .

Entrée : Signal u , réponse impulsionale g .

- 1) Calculer la TFD de u et de g . Si u et g n'ont pas la même taille (souvent g est "trop court") prolonger g par des zéros avant de calculer sa TFD³.
- 2) Multiplier point à point : $\hat{u}(0)\hat{g}(0), \dots, \hat{u}(N-1)\hat{g}(N-1)$
(En matlab c'est `.*`; si u est v ont même longueur $u.*v$ calcule le produit point à point. `*` seul est utilisé pour les produits matriciels.)
- 3) Calculer la TFD-inverse du résultat.
- 4) (Optionnel : ne garder que la partie réelle.)

3. La formule de $*_{per}$ nous dit qu'il faut les ajouter après les valeurs connues de g , sinon on va translater le résultat. Ex $g=[1 -1] \Rightarrow g=[1 -1 0 \dots 0]$

Algorithme pour calculer une dé-convolution (p. ex. enlever du flou) :

On se souvient toujours que $DFT(u *_{per} g)(k) = DFT(u)(k)DFT(g)(k)$, pour tout k .

Cette formule nous donne aussi la procédure pour dé-convoler, et les conditions pour que ce soit possible : Si $DFT(g)(k)$ ne s'annule pas alors le filtre est **inversible**.

Si v est connu ou a été estimé on applique : Entrée : Signal convolé u , réponse impulsionnelle g .

- 1) Calculer la TFD de u et de g . Si u et g n'ont pas la même taille (souvent g est "trop court") prolonger g par des zéros avant de calculer sa TFD⁴.
- 2) Diviser point à point : $\hat{u}(0)/\hat{g}(0), \dots, \hat{u}(N-1)/\hat{g}(N-1)$
(En matlab c'est "./.".)
- 3) Calculer la TFD-inverse du résultat.
- 4) (Optionnel : ne garder que la partie réelle.)

4. La formule de $*_{per}$ nous dit qu'il faut les ajouter après les valeurs connues de g , sinon on va translater le résultat. Ex $g=[1 -1] \Rightarrow g=[1 -1 0 \dots 0]$

En pratique, si pour certains k la quantité $|DFT(g)(k)|$ est très petite, le résultat ne va pas être bon. Car nous multiplions le bruit énormément (on va le multiplier par $\frac{1}{|DFT(g)(k)|}$)

Pour les filtres non-inversibles on va préférer un filtre de type [Wiener](#). Ce filtre est défini directement par sa [réponse fréquentielle](#) :

$$\frac{\text{conj}(DFT(g)(k))}{\lambda + \text{abs}(DFT(g)(k))^2}$$

λ est un paramètre qui contrôle la "quantité de déconvolution".

$\lambda = 0$ correspond à une division et ne fonctionne que pour les filtres inversibles.

Quand λ croît on déconvole de plus en plus partiellement.

(Voir plus bas pour des exemples sur des images.)

Transformée de Fourier Discrète (TFD)

(formule d'inversion)

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} n} \text{ pour } n \in \{0, \dots, N-1\}.$$

Comme tout polynôme trigonométrique, sa définition s'étend à tout \mathbb{R} : On calcule :

$$\hat{u}(k) := \sum_{n=0}^{N-1} u(n) e^{-2i\pi \frac{k}{N} n} \text{ pour } k \in \{0, \dots, N-1\},$$

et on a la **formule d'interpolation**

$$\mathcal{I}u(x) := \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N} x}, \text{ pour } x \in \mathbb{R}.$$

On a évidemment $\mathcal{I}u(x) = u(n)$ pour tout $x \in \{0, \dots, N-1\}$.

On peut donc translater u arbitrairement, par exemple d'un demi-pixel.

Ce modèle continu pour u coïncide avec le théorème de Shannon : si nous observons un signal périodique et bande-limitee il serait exact.

Nous comparerons cette méthode avec un modèle affine par morceaux pour une rotation d'image.

Algorithme pour translater, formule de translation, cas 1D

Translater un signal permet p. ex. de recaler des images ou de synchroniser deux sons.

$$\begin{aligned}\mathcal{I}u(x - \frac{1}{2}) &:= \frac{1}{N} \sum_{k=0}^{N-1} \hat{u}(k) e^{2i\pi \frac{k}{N}(x - \frac{1}{2})}, \forall x \in \mathbb{R}. \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \left(\hat{u}(k) e^{-2i\pi \frac{k}{N} \frac{1}{2}} \right) e^{2i\pi \frac{k}{N} x}\end{aligned}$$

Donc pour translater un signal u de τ pixel(s) :

- 1) On calcule la FTD de u
- 2) On multiplie $\hat{u}(0)$ par $e^{-2i\pi \frac{0}{N}\tau}$, $\hat{u}(1)$ par $e^{-2i\pi \frac{1}{N}\tau}$, ..., $\hat{u}(N-1)$ par $e^{-2i\pi \frac{N-1}{N}\tau}$.
- 3) On calcule la FTD inverse.
- 4) (Optionnel ne garder que la partie réelle)

Cette transformation est clairement inversible (car nous n'avons jamais multiplié par 0).

Transformée de Fourier Discrète (2D-TFD)

Soit u une image finie définie sur $\{0, \dots, M - 1\} \times \{0, \dots, N - 1\}$. LA TFD de u est

$$\hat{u}(k, l) := \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} u(n, m) e^{-2i\pi \frac{k}{N} n} e^{-2i\pi \frac{l}{M} m} \quad \forall (k, l) \in \{0, \dots, M - 1\} \times \{0, \dots, N - 1\}$$

Cette transformation est séparable. Elle consiste à calculer la TFD des lignes, puis la TFD des colonnes du résultat ou le contraire : d'abord les colonnes puis les lignes (l'ordre de sommation n'a pas d'importance).

On a la formule d'inversion : $\forall (m, n) \in \{0, \dots, M - 1\} \times \{0, \dots, N - 1\}$:

$$u(m, n) := \frac{1}{N} \frac{1}{M} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \hat{u}(k, l) e^{2i\pi \frac{l}{N} n} e^{2i\pi \frac{k}{M} m}.$$

Pour les mêmes raisons que le cas 1D on a une formule d'interpolation

$$\mathcal{I}u(x, y) := \frac{1}{N} \frac{1}{M} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \hat{u}(k, l) e^{2i\pi \frac{l}{N}x} e^{2i\pi \frac{k}{M}y}.$$

qui permet d'étendre le signal à tout \mathbb{R}^2 .

Exemple d'applications : translation d'images.

En Matlab cela donne (convention/stokage des fréquences)

```
I= double(imread('house.png'))/255;
I=mean(I,3); %grayscale conversion
DFT2d_I=fft2(I); % compute DTF
[M , N]=size(I); % image size to get frequency grid.

%%%% WITH LOOPs %%%
% X translation
for k=1:floor(N/2)
    for l=1:M
        phase(l,k)=exp(-2*1i*pi*(Tx*(k-1)/N));
    end;
end;
for k=floor(N/2)+1:N
    for l=1:M
        phase(l,k)=exp(-2*1i*pi*(Tx*(k-1-N)/N));
    end;
end;
DFT2d_I_translated1=DFT2d_I.*phase;

% here DFT2d_I_translated1 contains the 2D-DFT of the image translated
% in x, we do the same for y
for k=1:N
    for l=1:floor(M/2)
        phase(l,k)=exp(-2*1i*pi*(Ty*(l-1)/M));
    end;
end;
for k=1:N
    for l=floor(M/2)+1:M
        phase(l,k)=exp(-2*1i*pi*(Ty*(l-1-M)/M));
    end;
```

Algorithme pour zoomer "in" un signal : zero padding.

Idée : le signal est un polynôme.

La TFD a comme sortie autant de point que le nombre de coefficients (degré) du polynôme.

Algorithme pour zoomer "in" un signal : zero padding.

Idée : le signal est un polynôme.

La TFD a comme sortie autant de point que le nombre de coefficients (degré) du polynôme.

On va ajouter des fréquences plus hautes, avec comme valeur zero.

Equivalent à remplacer $X + X^2 - X^4 + X^5$ par

$X + X^2 - X^4 + X^5 + 0X^6 + 0X^7 + 0X^8 + \dots$

Cela ne change donc pas le polynôme, juste le nombre de points d'évaluation.
(Voir plus loin pour une image et la liste des commandes complètes.)

Algorithme pour calculer une convolution périodique :

Entrée : Signal u , réponse impulsionnelle g .

- 1) Calculer la TFD de u et de g .
- 2) Multiplier point à point : `fft2(u).*fft2(g)`
- 3) Calculer la TFD-inverse du résultat.
- 4) (Optionnel : ne garder que la partie réelle.)

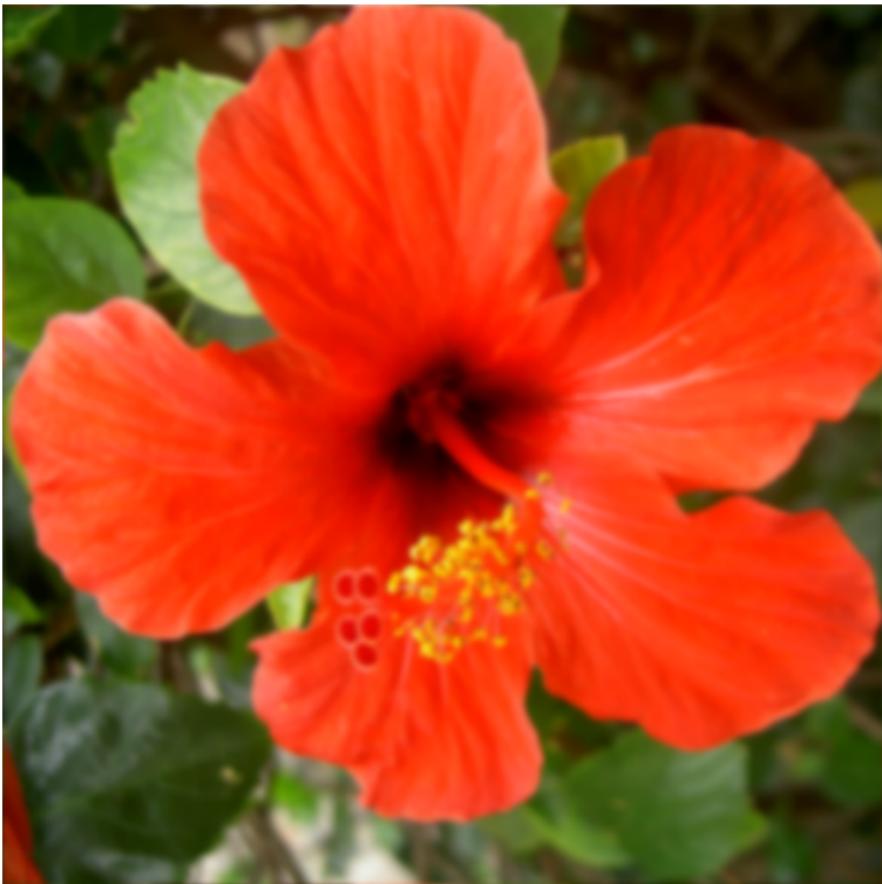
```
I=imread('house.png');  
I=double(mean(I,3)/255);  
g=ones(10,10)/100;  
produit=fft2(I).*fft2(g,size(I,1),size(I,2))5;  
output=real(ifft2(produit));  
figure ;imshow(output);
```

5. `fft2(signal, taille en x, taille en y)` permet de prolonger le signal par des zeros directement.





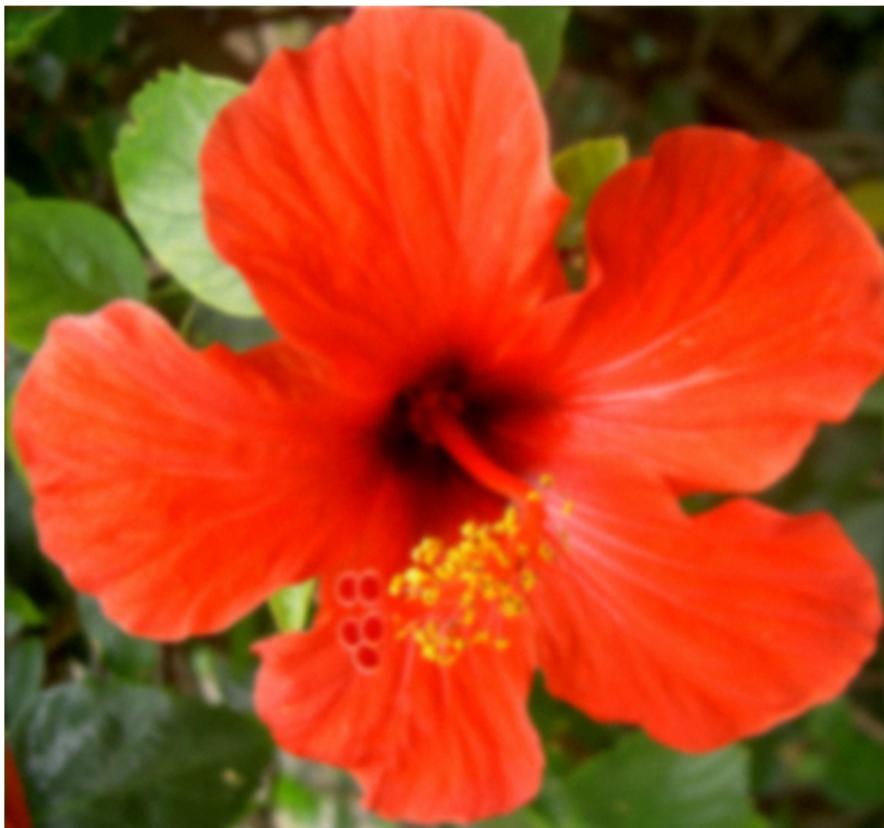
Déconvolution Gaussienne (sans bruit) : image observée



Déconvolution Gaussienne (sans bruit) : image déconvolée,
noyau connu



Déconvolution Gaussienne (avec bruit, $\mathcal{N}(0, 0.01)$) : image observée. Image a ses valeurs dans $[0, 1]$



Déconvolution Gaussienne avec bruit, $\mathcal{N}(0, 0.01)$) :
déconvolée par division



Déconvolution Gaussienne avec bruit, $\mathcal{N}(0, 0.01)$) :
déconvolée par Wiener



Algorithme pour zoomer-in une image

C'est le même principe qu'en 1D. On garde les coefficients de l'image d'origine et on complète par des zeros.

```
DFT2d_I_padded=zeros(floor(M*zoom_factor),floor(N*zoom_factor),nb_color_channels);  
DFT2d_I_padded(1 :floor(M/2),1 :floor(N/2), :)=DFT2d_I(1 :floor(M/2),1 :floor(N/2), :);  
DFT2d_I_padded(1 :floor(M/2),end :-1 :end-  
floor(N/2)+1, :)=DFT2d_I(1 :floor(M/2),end :-1 :floor(N/2)+1, :);  
DFT2d_I_padded(end :-1 :end-floor(M/2)+1,1 :floor(N/2), :)=DFT2d_I(end :-  
1 :floor(M/2)+1,1 :floor(N/2), :);  
DFT2d_I_padded(end :-1 :end-floor(M/2)+1,end :-1 :end-  
floor(N/2)+1, :)=DFT2d_I(end :-1 :floor(M/2)+1,end :-1 :floor(N/2)+1, :);  
DFT2d_I_padded=DFT2d_I_padded*floor(M*zoom_factor)*floor(N*zoom_factor)/(M*N);  
I_zoomed=ifft2(DFT2d_I_padded);  
I_zoomed=real(I_zoomed);
```

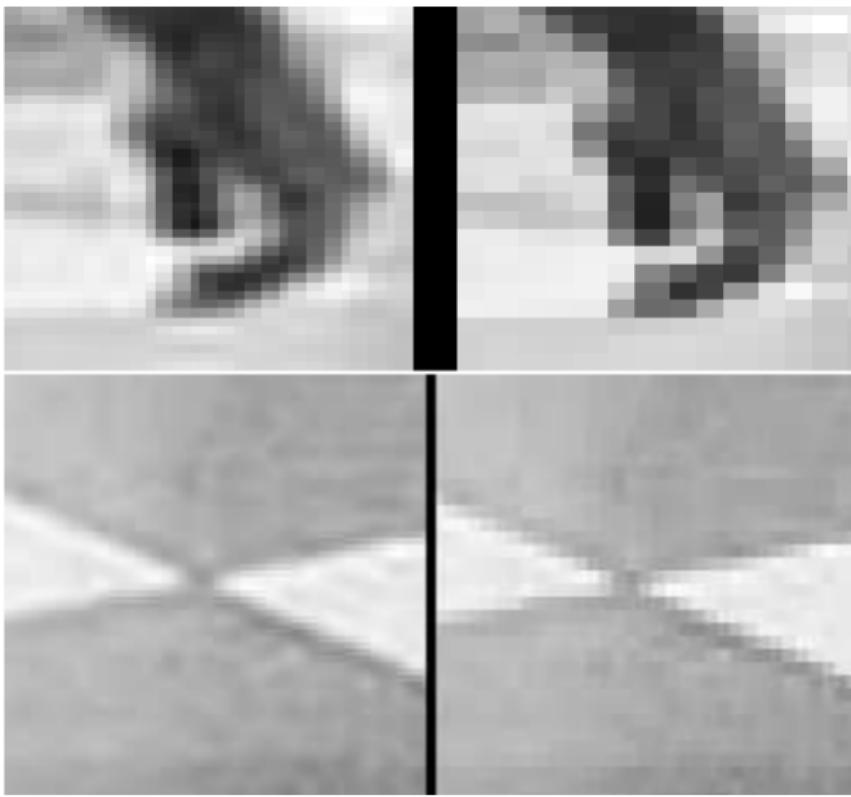
Original



Zoom X4 : padding



Zoom par réPLICATION des pixels



Rotation

Nous voudrions ré-échantillonner u aux points

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix},$$

pour $(m, n) \in \{0, \dots, M-1\} \times \{0, \dots, N-1\}$. La formule

$$\mathcal{I}u(x, y) := \frac{1}{N} \frac{1}{M} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \hat{u}(k, l) e^{2i\pi \frac{x}{N} n} e^{2i\pi \frac{y}{M} m}.$$

conduit à effectuer MN sommes pour évaluer un point de l'image tournée => $(MN)^2$ sommes au total.

Cela rend la méthode directe inopérante en pratique.

Algorithme pour calculer une rotation d'image

Cette méthode est due L. Yaroslavsky. Elle repose sur la remarque suivante

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix},$$

qui est valide pour $\theta \neq \pm\pi$ modulo 2π . Si $\theta = \pm\pi$ modulo 2π , il suffit de retourner l'image (ou de la laisser telle quelle).

Regardons ce qu'il se passe pour la première matrice pour
 $(m, n) \in \{0, \dots, M - 1\} \times \{0, \dots, N - 1\}$:

$$\begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m - n \tan(\frac{\theta}{2}) \\ n \end{pmatrix}$$

Regardons ce qu'il se passe pour la première matrice pour
 $(m, n) \in \{0, \dots, M - 1\} \times \{0, \dots, N - 1\}$:

$$\begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix} = \begin{pmatrix} m - n \tan(\frac{\theta}{2}) \\ n \end{pmatrix}$$

=> C'est une translation suivant le premier indice. Les colonnes ne bougent pas. La première ligne est translatée de 0, la seconde de $-\tan(\frac{\theta}{2})$, la troisième de $-2\tan(\frac{\theta}{2})$, la suivante de $-3\tan(\frac{\theta}{2})$, etc.

=> La seconde matrice correspond à translater les lignes de manière similaire.

=> La troisième translate les colonnes à nouveau.

Algorithme pour calculer une rotation d'image

Placer l'image au centre d'une plus grande image en complétant par des zéros ou une autre constante

(Sinon par périodisation un bord qui sort ré-entre de l'autre coté.)

Utiliser l'algorithme pour translater un signal (les lignes/colonnes) avec les valeurs de translations données par

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix}.$$

Rotation avec la méthode de Yaroslavsky (16 fois $\frac{\pi}{4}$)



Rotation avec interpolation bilinéaire



Image originale



Rotation avec la méthode de Yaroslavsky



Zoom out : original



Zoom out : direct decimation : Aliasing



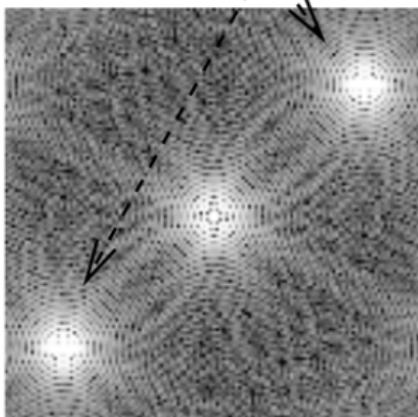
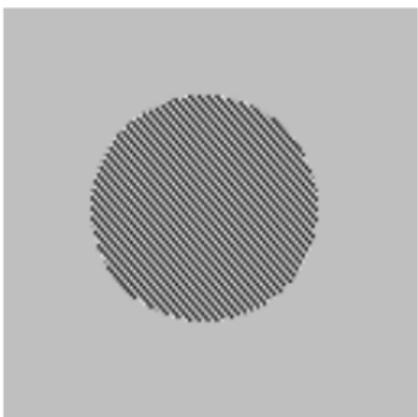
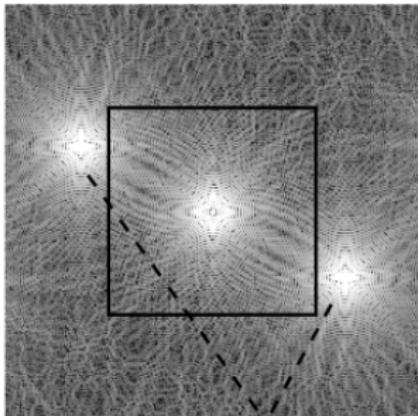
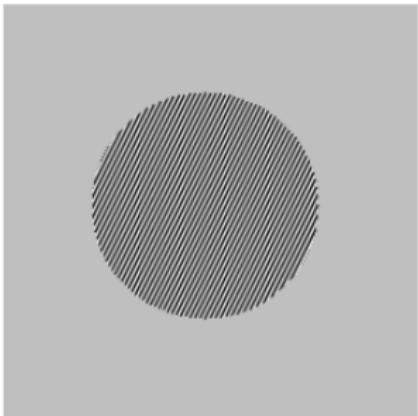
Zoom out (2) : original



Zoom out :(2) direct decimation : Aliasing



Explication du phénomène de repliement spectral (Aliasing)



Algorithme pour de-zoomer

Pour éviter le repliement, il faut d'abord tuer les hautes fréquences.
Nous pourrions les mettre brutalement à zéro, mais cela provoque du "ringing".



Algorithme pour de-zoomer

Pour éviter le repliement, il faut d'abord tuer les hautes fréquences.

Nous pourrions les mettre brutalement à zéro, mais cela provoque du "ringing". Nos yeux ont l'habitude de voir des images avec des spectres tendant vers 0 aux bords.

Algorithme :

- 1) Convoler l'image avec une Gaussienne d'écart type $0.8\sqrt{zoom_factor^2 - 1}$;⁶
- 2) Sous-échantillonner : Ne retenir qu'une valeur tous les $zoom_factor$ dans les deux directions.

Dans la littérature on peut voir $0.8zoom_factor$, cela provient du fait que $\sqrt{zoom_factor^2 - 1} \approx zoom_factor^2$ pour $zoom_factor^2 \gg 1$.

6. Pour la justification de ce facteur voir Morel, J. M., & Yu, G. (2008). "On the consistency of the sift method." equation (7)

Zoom out (1) : original



Zoom out (1) : après convolution par une Gaussienne



Zoom out (2) : original



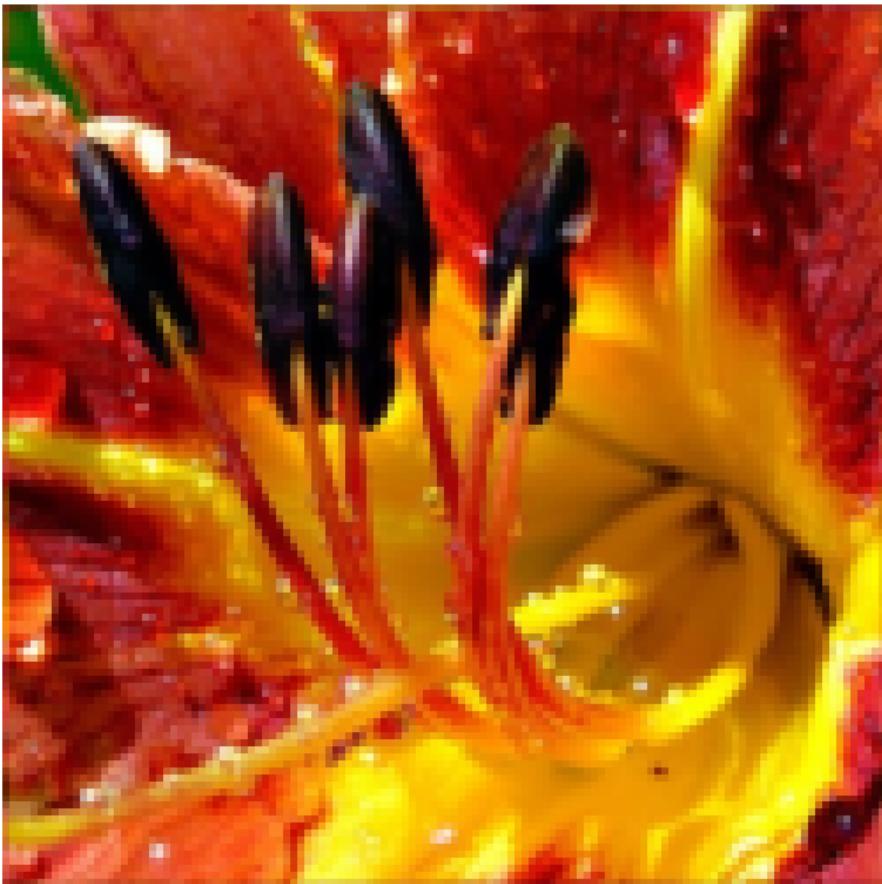
Zoom out (2) : après convolution par une Gaussienne



Zoom out (3) : original



Zoom out (3) : après convolution par une Gaussienne



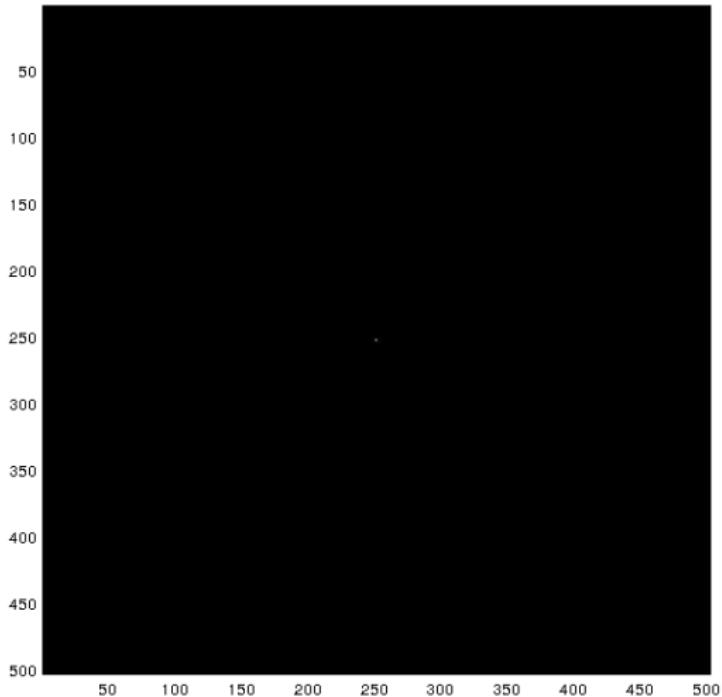
Visualiser le module d'une TFD

Algorithme :

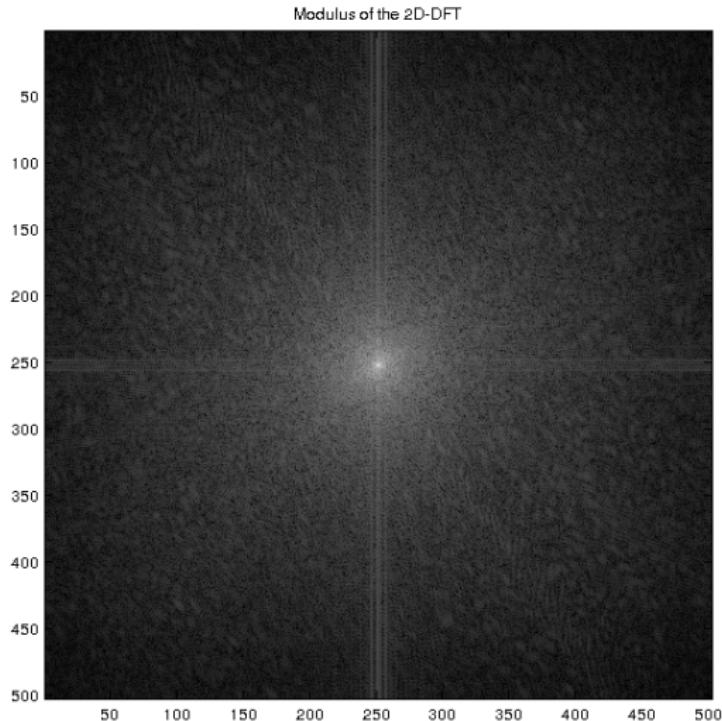
- 1) Calculer la TFD-2D : `fft2_I=fft2(I);`
- 2) Extraire le module (commande abs en matlab) `modulus=fftshift(abs(fft2_I));`
- 3) Ajouter 1 et prendre le log : `modulus=log(modulus+1);`
- 4) Changer la dynamique : mettre le minimum à zero et le max à 1 :
`modulus=modulus-min(modulus(:));`
`modulus=modulus/max(modulus(:));`
- 5) Afficher : `imshow(modulus)`⁷

7. Imshow affiche des images dont la valeur est dans $[0, 1]$. Les valeurs ≤ 0 sont toutes noires, les valeurs ≥ 1 sont toutes blanches.

Module de la TFD d'une image



Module de la TFD d'une image : echelle Log



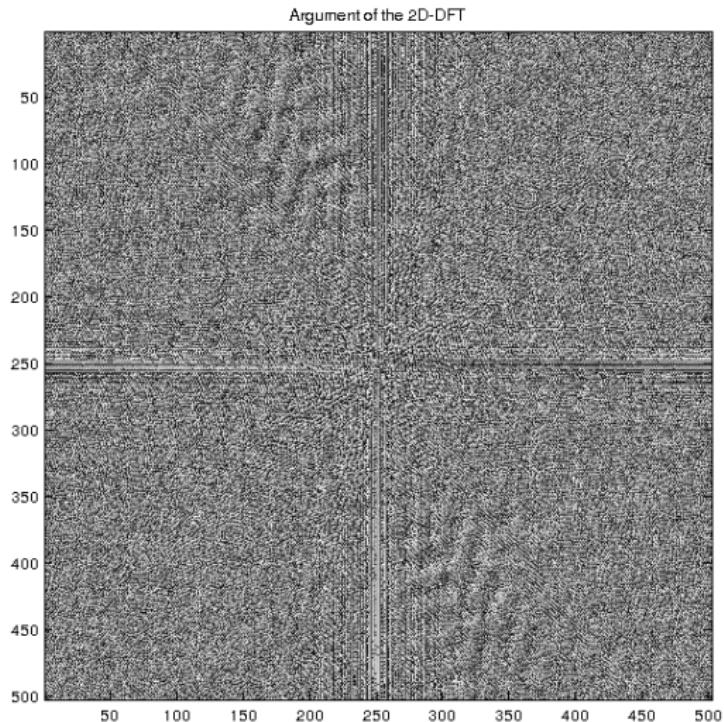
Visualiser la phase d'une TFD

Algorithme :

- 1) Calculer la TFD-2D : `fft2_I=fft2(I);`
- 2) Extraire la phase (commande angle en matlab) `angle=fftshift(fft2_I);`
- 3) Changer la dynamique : mettre le minimum à zero et le max à 1 :
`angle=angle-min(angle(:));`
`angle=angle/max(angle(:));`
- 4) Afficher : `imshow(modulus)`⁸

8. Imshow affiche des images dont la valeur est dans $[0, 1]$. Les valeurs ≤ 0 sont toutes noires, les valeurs ≥ 1 sont toutes blanches.

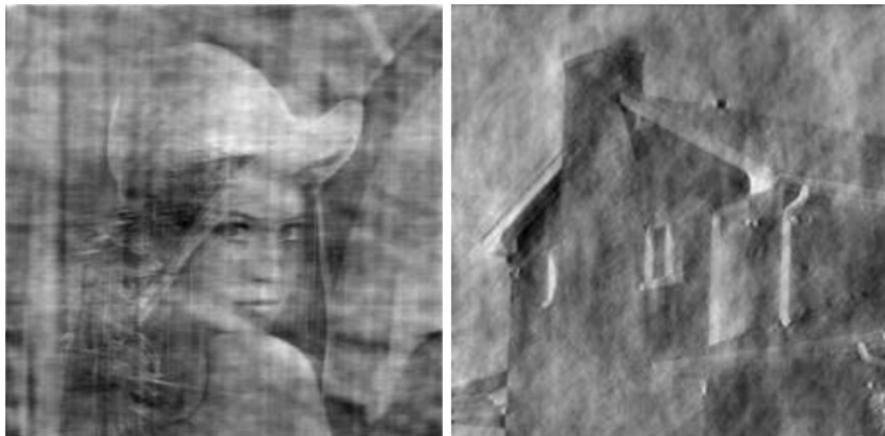
Phase de la TFD d'une image



Echange phase/module



Echange phase/module



L'image de gauche à le module de la maison et la phase de Lena. L'image de droite à le module de Lena et la phase de la maison. => Les structures géométriques sont essentiellement stockées dans la phase.

Pour calculer le gradient en x et en y d'une image (utile p. ex. dans les détecteurs de contours)

```
I= double(imread('lena.bmp'))/255 ;9
grad_x=I(1 :end-1, :) - I(2 :end, :);
grad_y=I( :,1 :end-1) - I( :,2 :end);
(C'est la formule " $\frac{f(x+h)-f(x)}{h}$ ")
```

Pour obtenir la magnitude :

```
grad_mag=sqrt(grad_x( :,1 :end-1).^2+grad_y(1 :end-1, :)^2);
(On tronque l'égèrement à cause des effets de bords.)
```

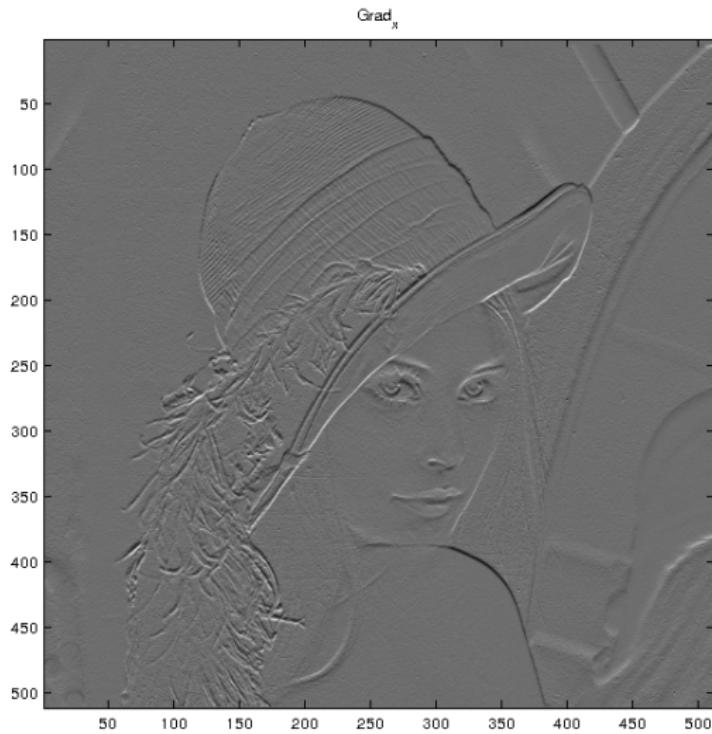
9. Commande pour lire une image, 255 pour passer de 8-bits image à des valeurs dans 0,...,255 à [0, 1], double pour changer le type et avoir accès aux nombres à virgules.

Image

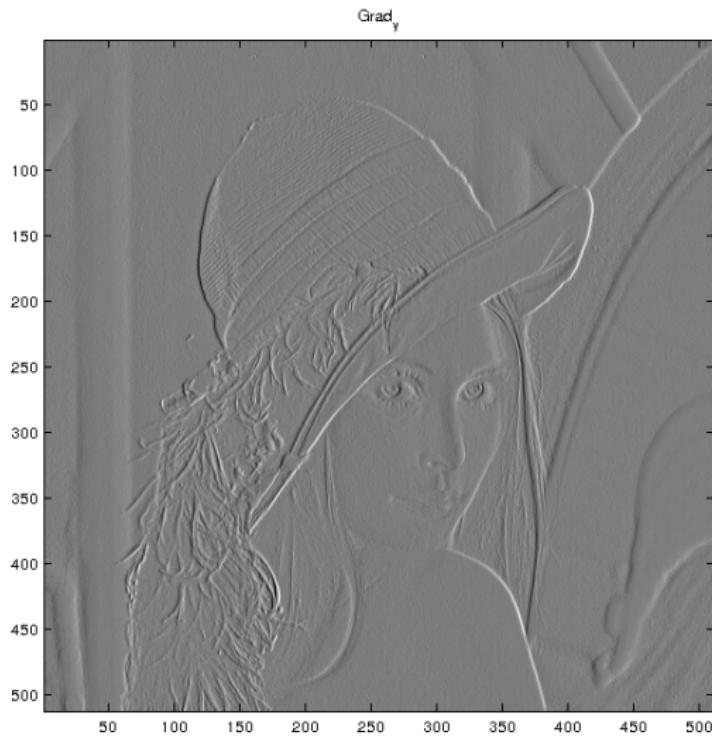
Image I



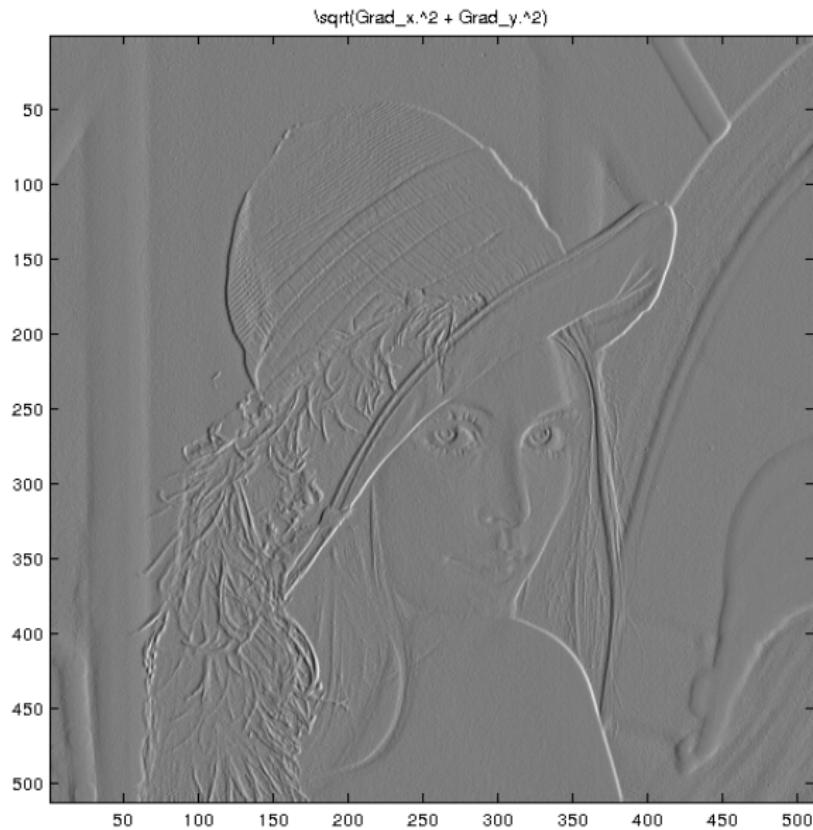
Grad x



Grad y



Module du gradient



Laplacien



Reductions des effets de bords.

Tous les calculs par TFD supposent l'image périodique.

Ce n'est en général pas vrai.

On peut "forcer" les choses en répliquant l'image.

(Faire un dessin)

Cela est équivalent à utiliser des TCD (Transformée en cosinus discret)

Tous les traitements que nous avons vu s'adaptent facilement.

En matlab

```
J( :, :, :) = [I(1 :end, 1 :end, :) I(1 :end, end :-1 :1, :) ; I(end :-1 :1, 1 :end, :)  
I(end :-1 :1, end :-1 :1, :)];
```

original



periodisee par symmetrie miroir



zoom par padding direct



zoom par TCD implicite



References

Quelques référence :

H. Maitre, Le traitement des images.

S. Mallat, A Wavelet Tour of Signal Processing.

G. Blanchet et M. Charbit, Traitement numérique du signal, Simulation sous Matlab
Pour du Fourier :

C. Gasquet et P. Witomski. Analyse de Fourier et applications.

J.M. Bony. Cours d'analyse. Théorie des distributions et analyse de Fourier : théorie des distributions et analyse de Fourier.

La rotation :

Convolution-Based Interpolation for Fast, High-Quality Rotation of Images. M. Unser, P. Thévenaz, and L. Yaroslavsky

Démosaickage :

Self-similarity Driven Demosaicking, A. Buades, B. Coll, J.-M. Morel and C. Sbert

Denoising :

Non-Local Means Denoising, A. Buades, B. Coll, J.-M. Morel

Edges :

LSD : a Line Segment Detector, R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall

A Review of Classic Edge Detectors, H. Spontón, and J. Cardelino

Le site web du journal IPOL : <http://www.ipol.im/>