

# INTRODUCTION AU TRAITEMENT D'IMAGES

*(enseignement intégré)*

2020-2021  
**Rémi Giraud**  
*remi.giraud@enseirb-matmeca.fr*

---



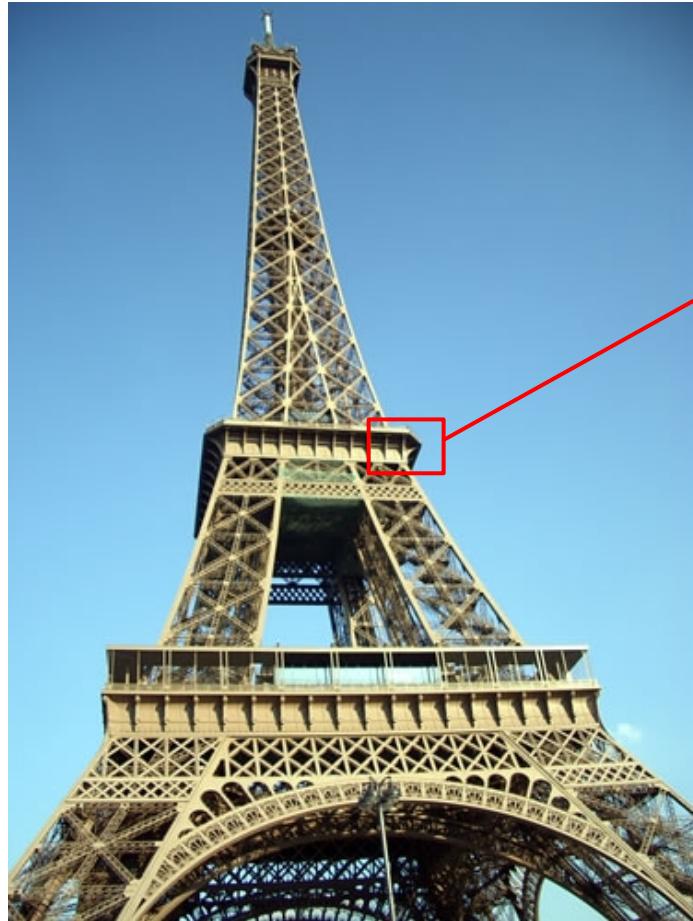
# Plan

---

- Image couleur
  - Format/affichage
  - Visualisation
  - Synthèse
- Espaces couleur caractéristiques
  - Espace YCbCr
  - Applications : compression, esquisse
- Traitements
  - Filtrage linéaire
  - Applications : Anonymisation, débruitage, recouvrement fréquentiel
  - Détection de contours
  - Applications : réhaussement de contraste

# Image numérique (1/4)

---

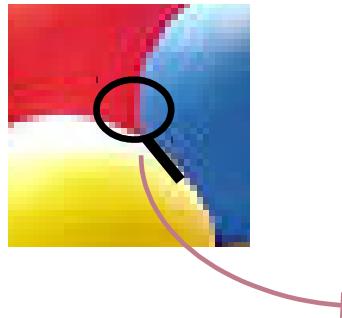


**Image numérique**  
=

**Matrice de pixels « colorés »**

- Dimensions (nombre de pixels)
- Coordonnées (position du pixel)
- Valeur (couleur du pixel)

# Image numérique (2/4)



R=212 G=16 B=40	R=205 G=65 B=112	R=103 G=120 B=176	R=62 G=127 B=193
R=201 G=26 B=43	R=197 G=69 B=94	R=154 G=106 B=148	R=98 G=117 B=186
R=192 G=101 B=106	R=138 G=59 B=80	R=127 G=96 B=137	R=97 G=129 B=188
R=255 G=250 B=250	R=230 G=192 B=213	R=140 G=118 B=156	R=73 G=97 B=145
R=250 G=248 B=251	R=255 G=248 B=255	R=255 G=246 B=255	R=182 G=176 B=210

Intensité vectorielle

## « Vraies » couleurs

- Composante rouge (R)
- Composante verte (V)
- Composante bleue (B)

212	205	103	62
201	197	154	98
192	138	127	97
255	230	140	73
250	255	255	182

Matrice R

40	112	176	193
43	94	148	186
106	80	137	188
250	213	156	145
251	255	255	210

Matrice B



16	65	120	127
26	69	106	117
101	59	96	129
250	192	118	97
248	248	246	176

Matrice V

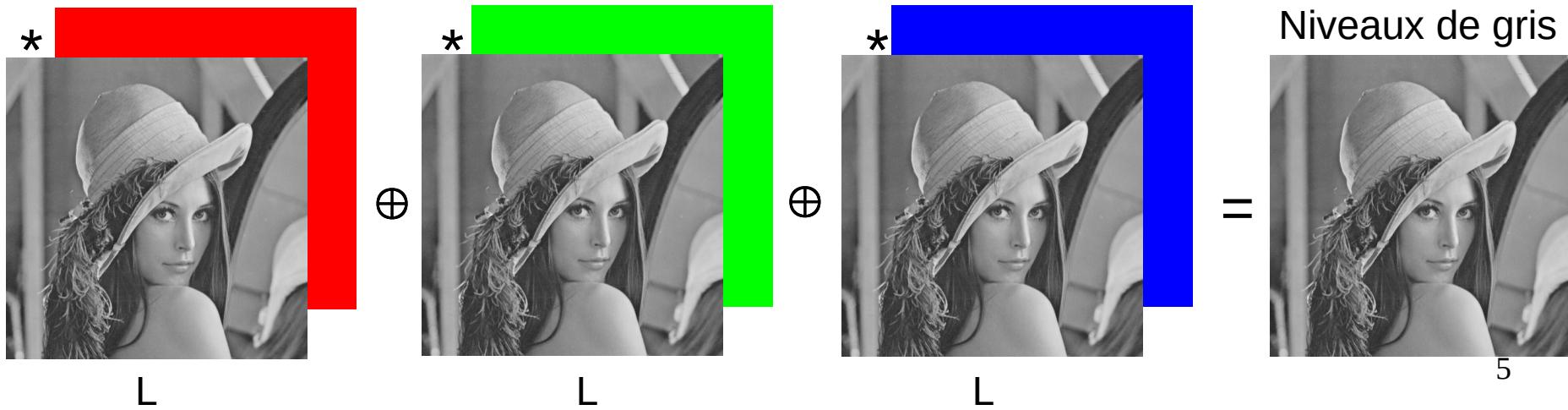
Format usuel :  
entiers 0 à 255

# Image numérique (3/4)

- Image couleur : Trois canaux d'intensité associés à une couleur primaire :



- Image naturelle en « niveaux de gris » : souvent la luminosité :  $L = (R+G+B)/3$

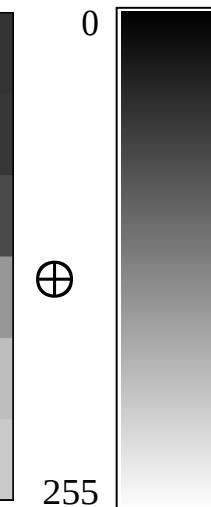


# Image numérique (4/4)



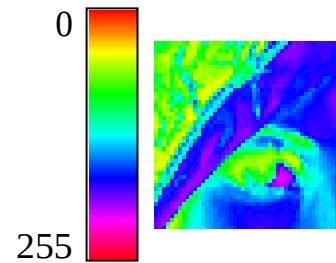
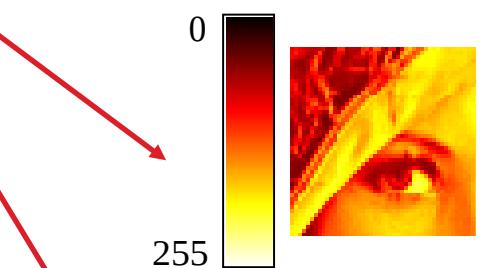
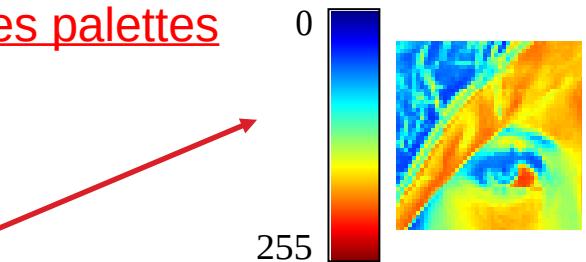
60	64	62	62	55	51
59	73	98	90	66	54
72	105	167	170	120	74
88	91	188	202	184	150
103	77	191	205	203	190
75	131	208	209	207	202

Intensité scalaire



Palette  
(niveaux de gris)

Autres palettes

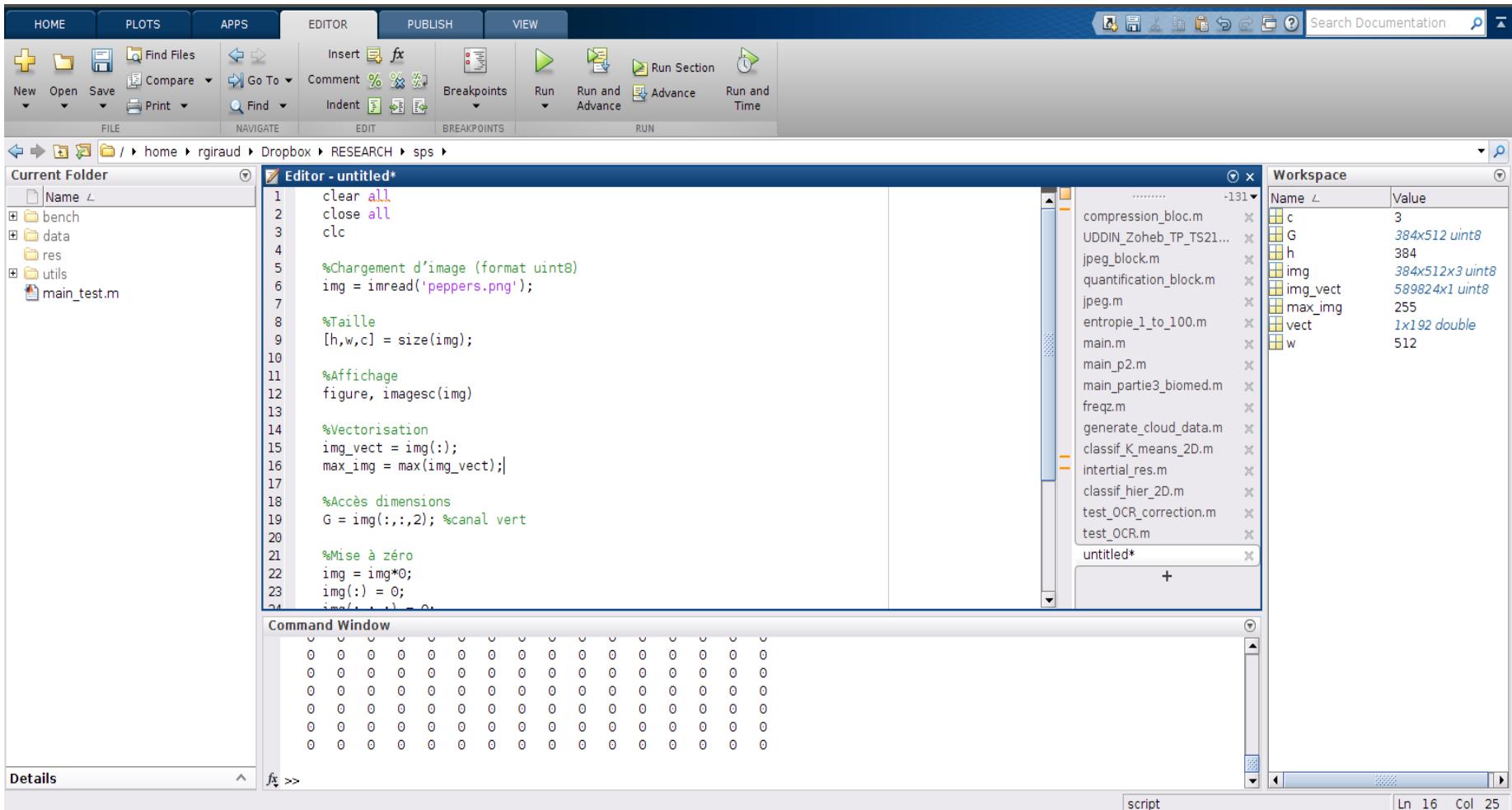


Couleurs indexées ou « fausses » couleurs

- Scalaire (index de table)
- Palette (table de correspondance couleur)

# Rappels Matlab (1/2)

- Environnement de développement



# Rappels Matlab (2/2)

- Conventions

x=1 y=1	x=2 y=1	x=3 y=1	x=4 y=1	x=5 y=1
x=1 y=2	x=2 y=2	x=3 y=2	x=4 y=2	x=5 y=2
x=1 y=3	x=2 y=3	x=3 y=3	x=4 y=3	x=5 y=3
x=1 y=4	x=2 y=4	x=3 y=4	x=4 y=4	x=5 y=4

Accès à une image RGB

=

Accès à un tableau tridimensionnel

`im(ligne,colonne,canal)`

=

`im(y,x,canal)`

~~`im(x,y,canal)`~~

# Lecture et affichage « vraies couleurs »

Images disponibles : <http://rgiraud.vvv.enseirb-matmeca.fr/teaching/>

```
>> A = imread('bdx.jpg');
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	538x808x3	1304112	uint8	

```
>> figure,imshow(A)
```

hauteur

largeur

« vraies » couleurs



# Lecture et affichage « fausses couleurs »

---

```
>> [A,palette] = imread('lena.bmp');
>> whos
  Name          Size            Bytes   Class      Attributes
  A              512x512        262144  uint8
  palette       256x3           6144   double
```

```
>> figure,imshow(A,colormap(jet))
>> figure,imshow(A,palette)
```



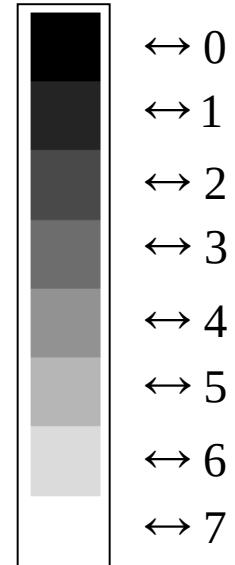
# Palette couleurs ?

```
>> gray(8)
```

```
ans =
```

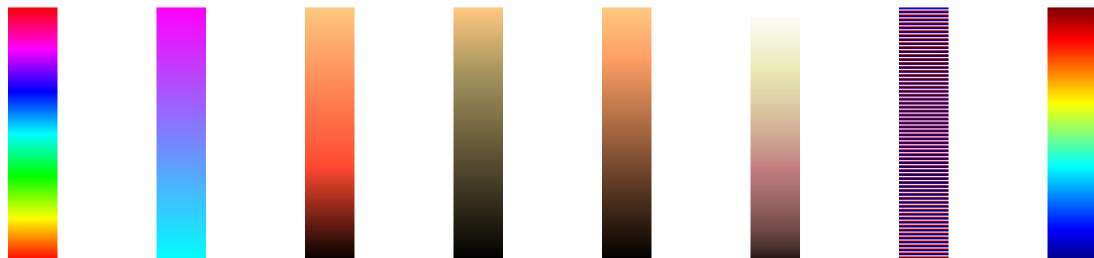
```
    0         0         0  
0.1429   0.1429   0.1429  
0.2857   0.2857   0.2857  
0.4286   0.4286   0.4286  
0.5714   0.5714   0.5714  
0.7143   0.7143   0.7143  
0.8571   0.8571   0.8571  
1.0000   1.0000   1.0000
```

R, G, B



Autres palettes : hsv, cool, hot, bone, copper, pink, flag, jet, ...

64 niveaux par défaut



# Méthodologie (1/4)

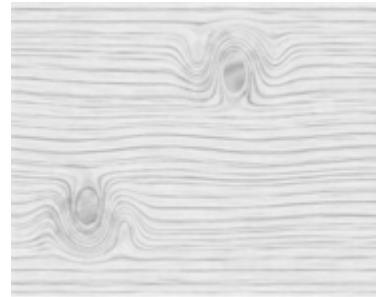
---

- Bilan des caractéristiques de l'image
  - Dimensions spatiales
  - Format numérique
  - Intervalles d'intensité
  - Codage
    - « vraies couleurs »
    - couleurs indexées
- Identification de la fonction d'affichage
  - Contexte
    - de fidélité
    - « informationnel »
    - de comparaison
  - Choix : palette, ratio L/H, intervalles d'intensité, etc.

# Méthodologie (2/4)

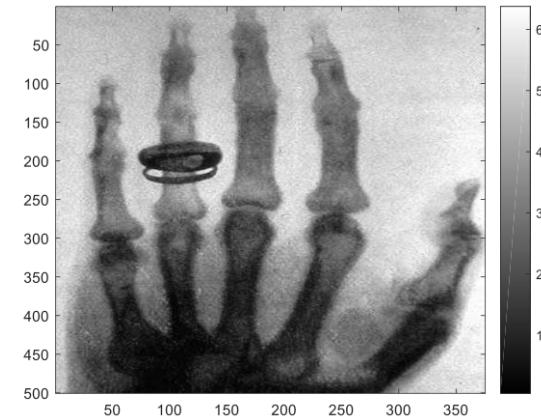
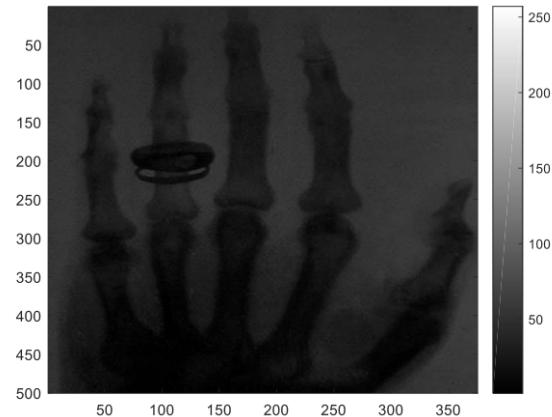
---

- Contexte de « fidélité »
  - Respect des intensités initiales
  - Fonctions Matlab : `imshow`/`imagesc`

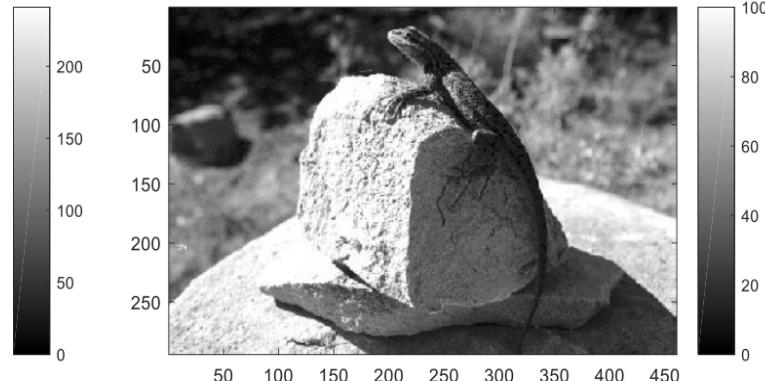
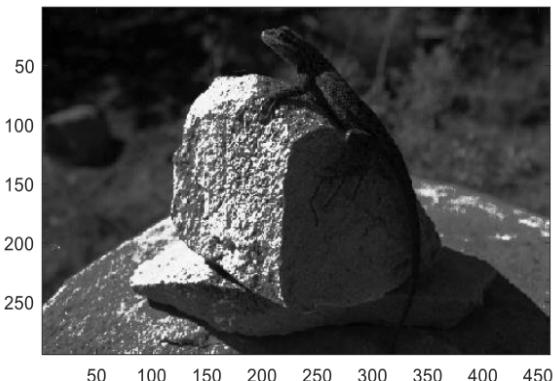


# Méthodologie (3/4)

- Contexte « informationnel » (couleurs indexées)
  - Étalement automatique des intensités (`imageds`)



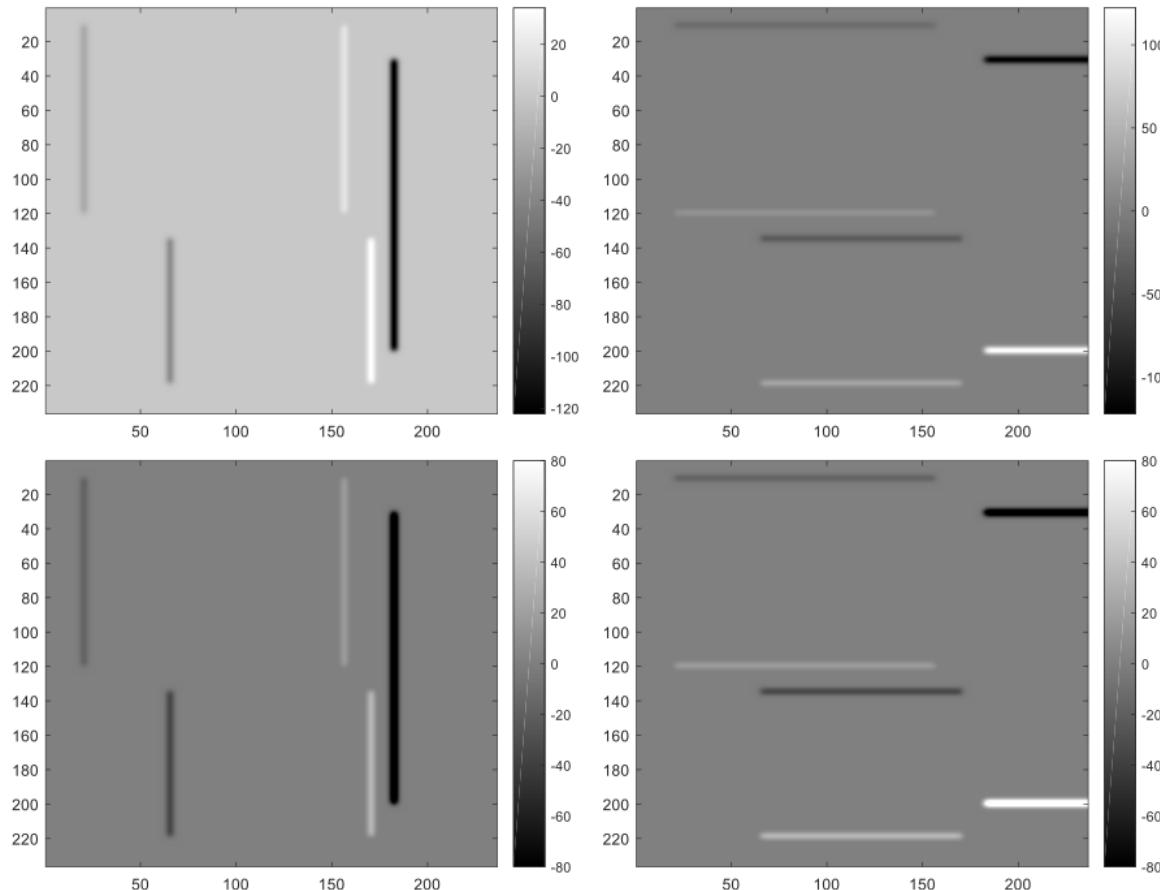
- Étalement contrôlé des intensités (`imageds(..., [ ])`)



# Méthodologie (4/4)

---

- Contexte de comparaison (couleurs indexées)
  - Étalement contrôlé et identique des intensités



# Mémo Matlab

- Commandes de bases :

```
%Chargement d'image (format uint8)
img = imread('img.png');
[h,w,c] = size(img) ← affichés dans la console

%Exemple d'affichage
figure, imagesc(img)

%Vectorisation
img_vect = img(:);
max_img = max(img_vect);

%Accès dimensions
G = img(:, :, 2); %canal vert

%Mise à zéro
img = zeros(h, w, c); %ones() existe aussi
img = img*0;

%Sous échantillonnage
img = imread('img.png');
figure,
imagesc(img(1:2:h, 1:4:end, :))

%Création d'une matrice
mat = [1 1; 2 2; 3 3] %de taille 3x2

%Produits vecteurs/matriciels
vect = (1:2:11); % (début: (pas):fin)
vect_2 = vect.*vect; %terme à terme
vect_x = vect*vect'; %produit matriciel
                                ↑ transposée
```

- Bonnes pratiques :

- Se placer dans un dossier dédié
- Toujours écrire dans un script
- Surveiller le workspace pour voir l'état et surtout la taille des variables
- Consulter l'aide MATLAB pour utiliser une fonction
- Ne pas relancer tout le script à chaque fois

- Exécution :

- Tout le script : **F5** ou <Run>
- Par section : **ctrl+enter** ou <Run section>

```
[h, w, c] = size(img);
```

```
%% Affichage
figure, imagesc(img)
```

```
%% Vectorisation
img_vect = img(:);
```

- Par sélection : **F9**

```
[h, w, c] = size(img);
```

```
%Affichage
figure, imagesc(img)
```

# Exercice : Affichage d'images

---

- Trouver une façon satisfaisante d'afficher les images de *challenge.mat*  
Fonctions Matlab : `imshow`, `imagesc`, `imagesc(..., [])` (lire la doc.)

Nom	Donnée	Problématique
Baboon	$I = s.A$	Format numérique
Formula	$I = s.B$	Ratio L / H
Radio	$I = s.C$	Intervalle d'intensité
Chess	$I = s.D$	Palette continue
World map	$I = s.E$	Palette discrète

```
clear  
close all  
  
s = load('challenge.mat');  
I = s.A; %Baboon
```

**Image**  $(h, w, c)$   
hauteur largeur canaux

Type ?

*uint8*  
(par défaut)

Intensité entière des valeurs entre  
[[ 0, I\_max = 255 ]]

*double*

Intensité supposée des valeurs entre  
[ 0, I\_max = 1 ]

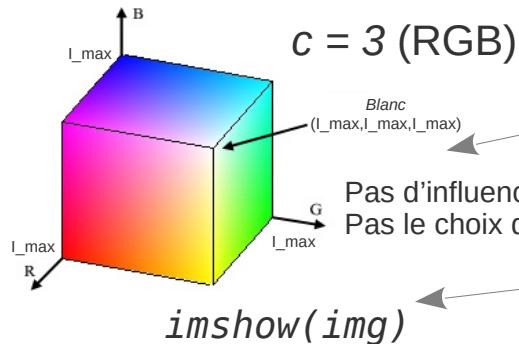
++ Possibilité de faire des calculs sur les intensités sans perdre d'information

Nombre de canaux (c) ?

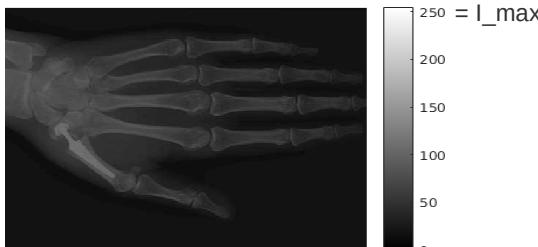
*c = 1 (?)*

Modifier la palette :  
*colormap('#palette\_name(N)')*

Fonction et palette ?



*imshow(img)*  
Palette par défaut : *gray(256)*  
N=256 fixe le niveau de détails de la palette



- conserve les proportions initiales ( $h \times w$ )
- pas d'axes d'échelle
- ne gère que des palettes niveaux de gris

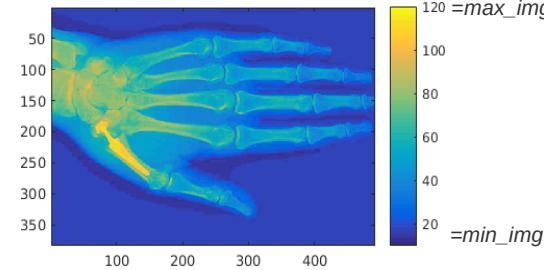
→ Pour afficher facilement des images couleurs ou niveaux de gris standards

Contexte de « fidélité »

*imagesc(img)*

Palette par défaut : *parula(64)*

Palette ajustée aux limites de l'image  
(*min\_img, max\_img*) sur N=64 niveaux



- + visualisation automatique du contenu
- plus possible de comparer deux images de manière équivalente

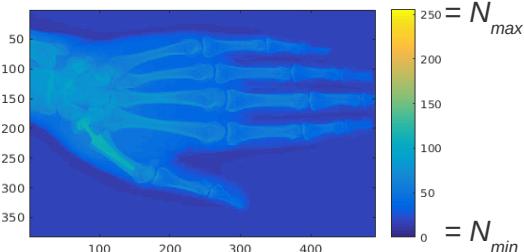
→ Pour travailler facilement sur tout type d'image

Contexte informationnel

*imagesc(img, [N\_min N\_max])*

Palette par défaut : *parula(64)*

Palette de N niveaux répartis uniformément entre  $N_{min}$  et  $N_{max}$



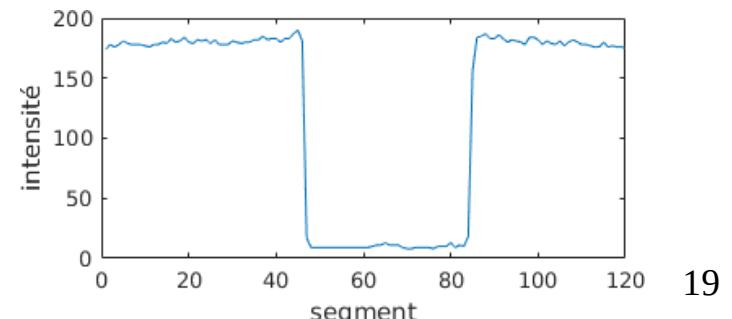
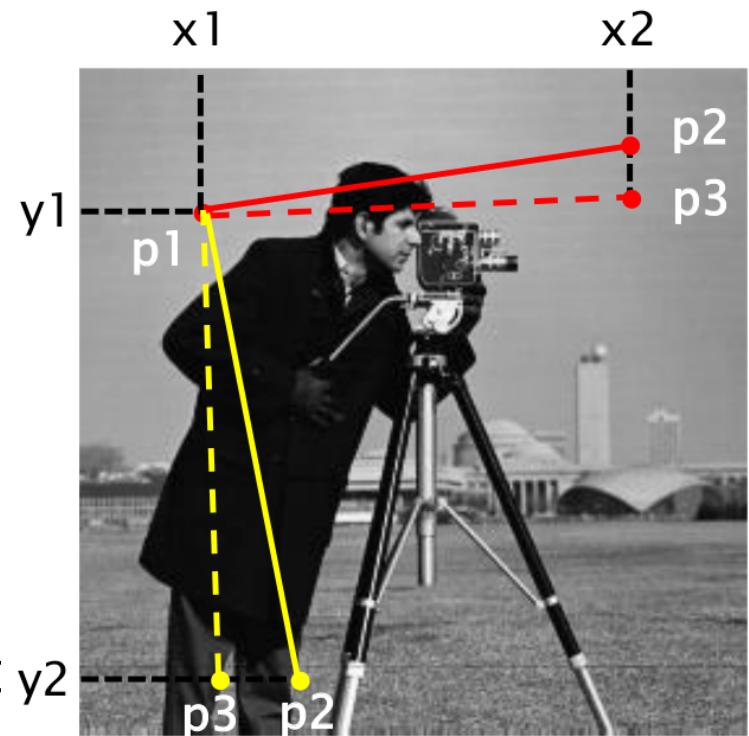
- besoin de définir une palette pertinente
- + possible de comparer deux images de manière équivalente

→ Pour comparer des images en fausses couleurs avec la même échelle

Contexte de comparaison

# Exercice : Interaction

- Affichage d'une image  
(par exemple *cameraman.tif*)
- Définir un segment horizontal ou vertical par 2 clics (ginput)
- Aligner les points en préférant le plus large segment  
`if(abs(x1-x2)>abs(y1-y2))`
- Extraire le profil 1D correspondant  
`I(y1,x1:x2)` ou `I(y1:y2,x1)`
- Afficher le profil 1D (plot)
- Superposer les profils R, G, B d'une image couleur

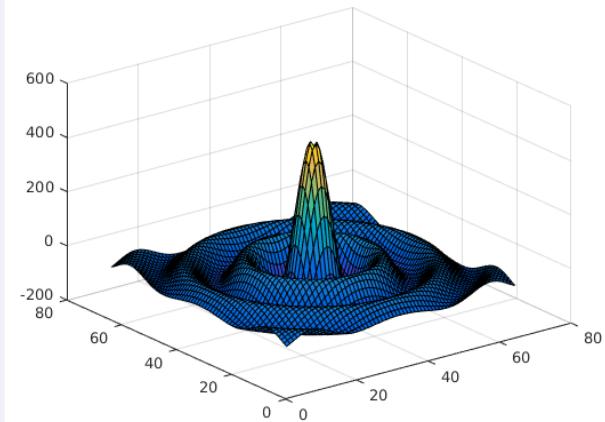
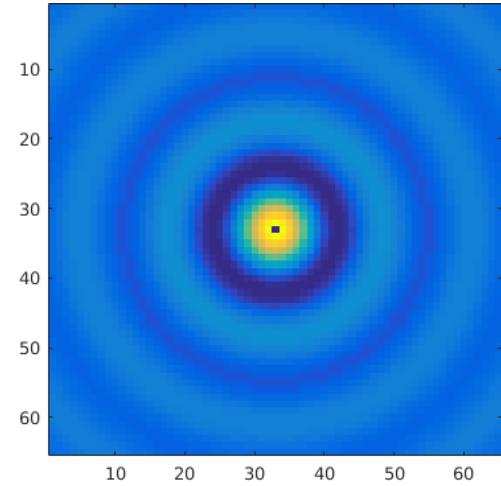


# Synthèse analytique en « couleurs indexées »

---

```
clear, close all, clc
%solution longue (8 lignes)
x = -34:34; y = -32:32;
img1 = zeros(length(y),length(x));
for i=1:length(y)
    for j=1:length(x)
        r = sqrt(x(j)^2+y(i)^2);
        img1(i,j) = 1000*sin(r/2)/r;
    end
end
figure, imagesc(img1), axis square
figure, surf(img1)

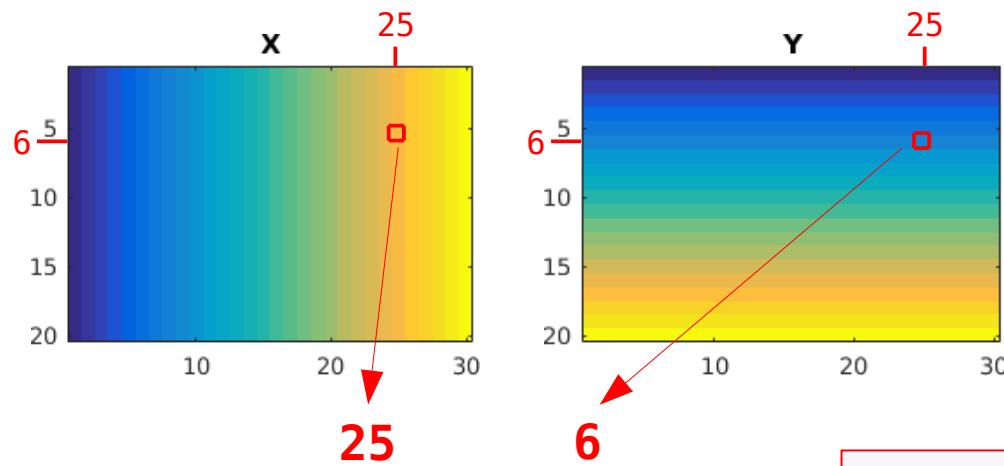
%solution courte (3 lignes)
[X,Y]=meshgrid(-34:34,-32:32);
R=(X.^2+Y.^2).^.5;
img2=1000*sin(R/2)./R;
figure, imagesc(img2), axis square
figure, surf(img2)
```



# Fonctionnement de meshgrid

But : Se passer des boucles de parcours H/L

Exemple : `[X, Y] = meshgrid(1:30, 1:20);`



Crée deux matrices de taille 20x30 qui stockent en valeur pour chaque pixel le numéro de colonne (X) et le numéro de ligne (Y)

```
clear, close all, clc  
  
x_vect = -10:10;  
y_vect = -20:20;  
[X, Y] = meshgrid(x_vect, y_vect);  
I = X.^2 + Y.^2;
```

Opérateur terme à terme sur chaque élément de la matrice

```
clear, close all, clc  
  
x_v = -10:10;  
y_v = -20:20;  
I = zeros(length(y_v), length(x_v));  
for i=1:length(y_v)  
    for j=1:length(x_v)  
        I(i,j) = y_v(i)^2 + x_v(j)^2;  
    end  
end
```

# Synthèse en « vraies couleurs »

---

```
clear, close all
```

```
Nx=25;
```

```
Ny=25;
```

```
[X,Y]=meshgrid(0:Nx-1,0:Ny-1);
```

```
R=(1-mod(X,2)).*(1-mod(Y,2));
```

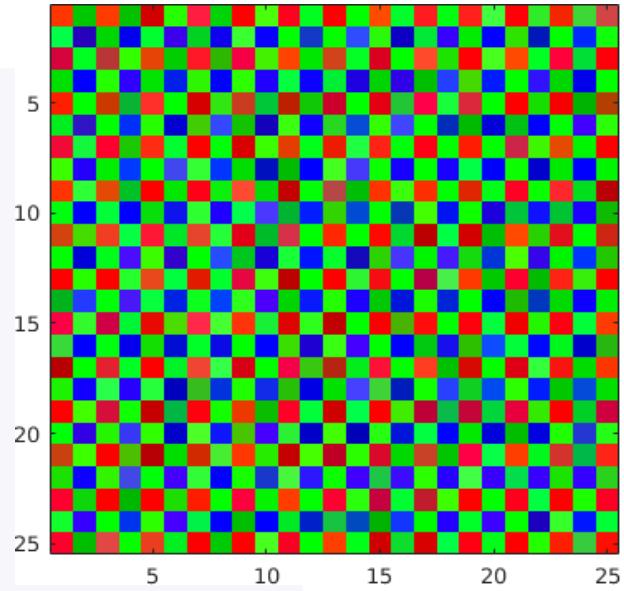
```
G=mod(X+Y,2);
```

```
B=(1-mod(X+1,2)).*(1-mod(Y+1,2));
```

```
I=cat(3,R,G,B)+0.6*(rand(Ny,Nx,3)-0.5);
```

```
I=max(min(I,ones(Ny,Nx,3)),zeros(Ny,Nx,3));
```

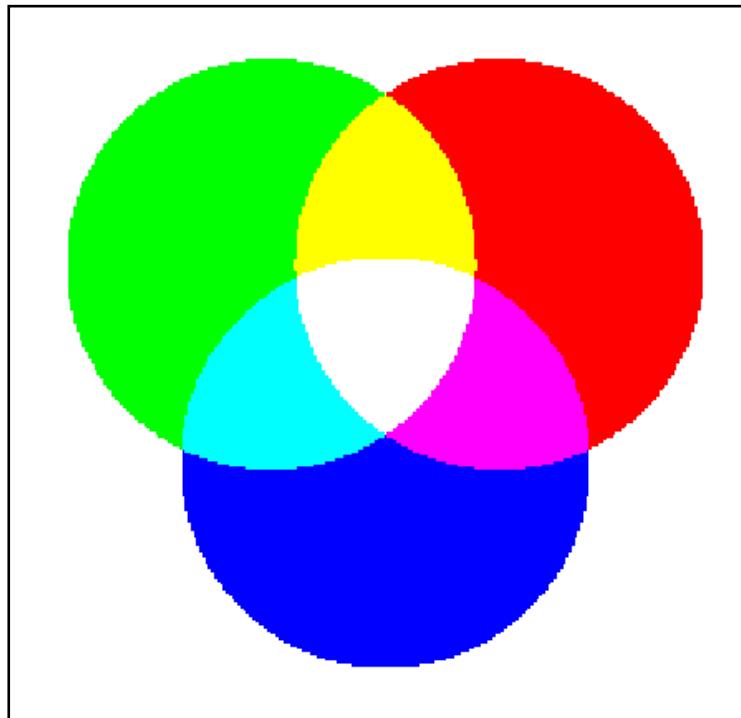
```
figure, image(I), axis image
```



# Exercice : Synthèse additive

---

- Écrire une fonction disk.m qui génère cette image :
    - En « vraies couleurs »
    - En couleurs indexées
- La largeur et l'espacement des cercles seront des paramètres



# Exercice : Synthèse additive dynamique

---

- Remplissez le programme *p1\_disks\_video.m* qui alterne les couleurs des cercles pour créer une vidéo de :
  - 100 images
  - 5 images/seconde (option FrameRate)
  - Sans compression

Fonctions utiles :

- `VideoWriter`, `writeVideo`, `open`, `close`  
(Bien lire les documentations)

# Exercice : Synthèse additive dynamique

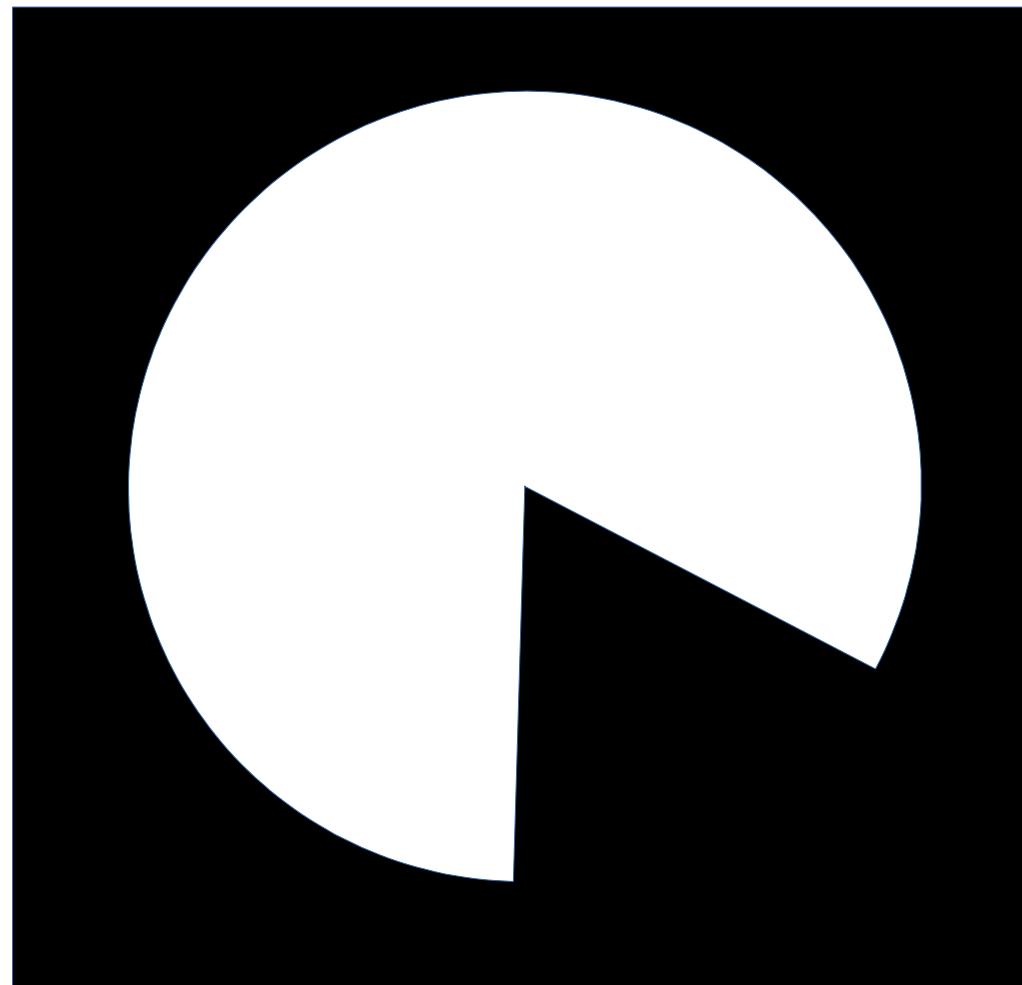
```
clear all, close all, clc  
  
Size=255;  
r=70;  
dist=45;  
[R,G,B] = p1_ex_disks(size,r,dist);  
  
%Conversion en uint8 [0, 255]  
C1=uint8(R*255);  
C2=uint8(G*255);  
C3=uint8(B*255);  
  
%Créer l'objet vidéo avec Videowriter  
% v = ...  
  
%Définir le framerate  
...  
  
%Ouvrir de l'objet vidéo  
open(v)
```

```
for n=1:100  
  
    q=mod(n, 3);  
  
    switch q  
        case 0 %C1=R-C2=G-C3=B  
            img = cat(3,C1,C2,C3);  
        case 1 %C3,C1,C2  
            ...  
        case 2 %C2,C3,C1  
            ...  
    end  
  
    %Ajout du frame dans la vidéo  
    ...  
  
End  
  
%Fermeture de l'objet vidéo  
close(v)
```

# Exercice : Timer

---

- Créer un timer animé sous forme de camembert progressif

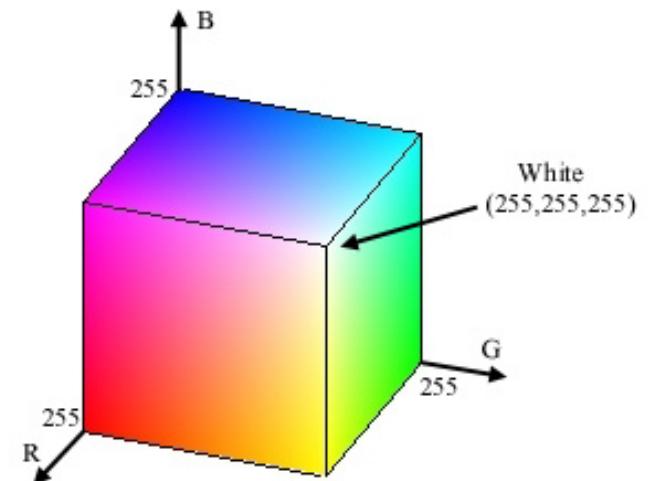
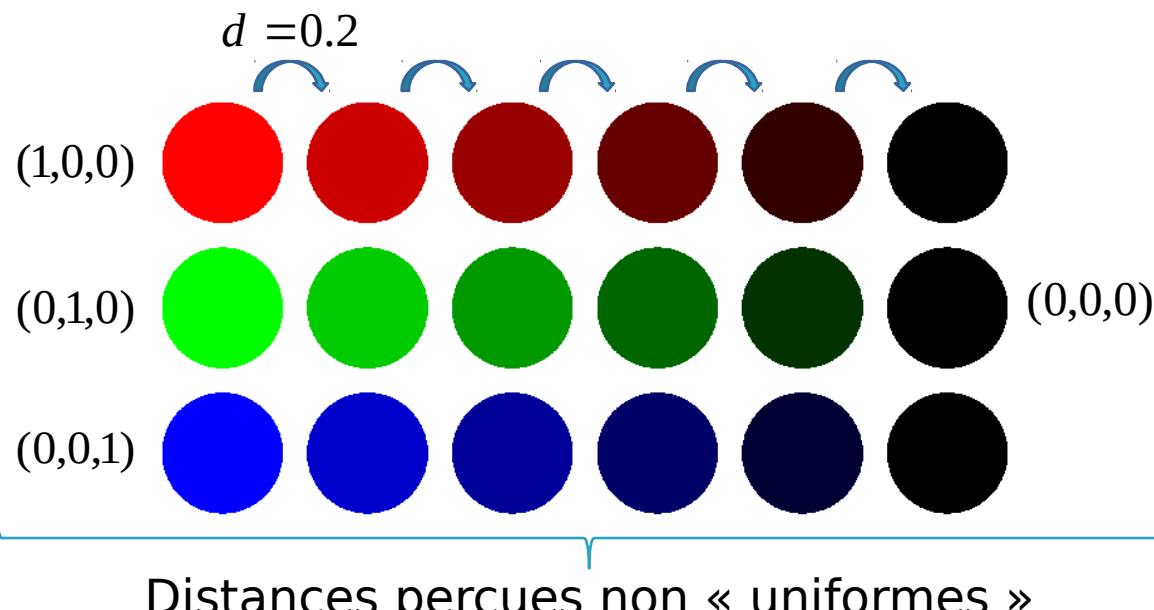


# Plan

---

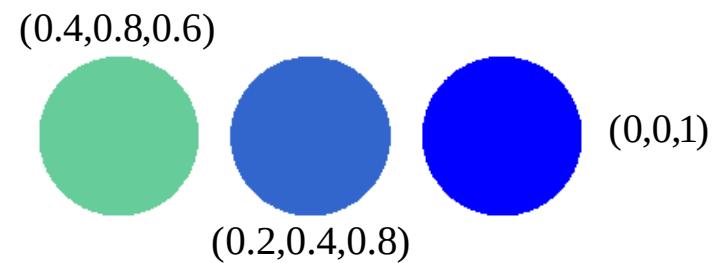
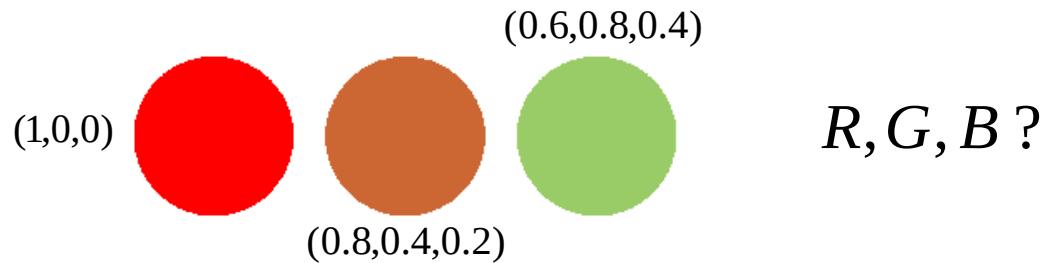
- Image couleur
  - Format/affichage
  - Visualisation
  - Synthèse
- Espaces couleur caractéristiques
  - Espace YCbCr
  - Applications : compression, esquisse
- Traitements
  - Filtrage linéaire
  - Applications : Anonymisation, débruitage, recouvrement fréquentiel
  - Détection de contours
  - Applications : réhaussement de contraste

# Espace chromatique RGB



$$L = \frac{R + V + B}{3}$$

luminance  
(intensité moyenne)

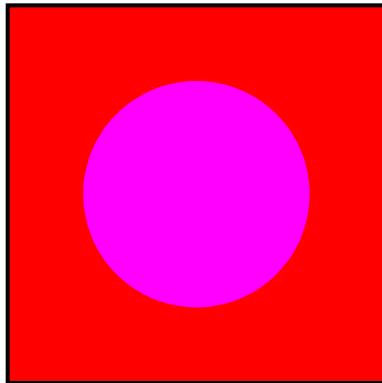


# Luminosité vs Chrominance

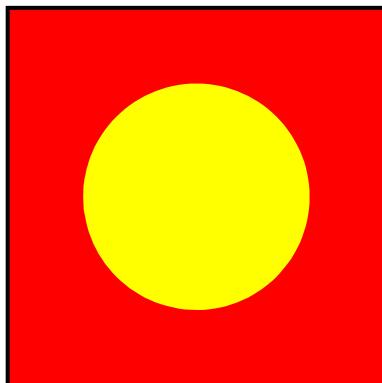
---



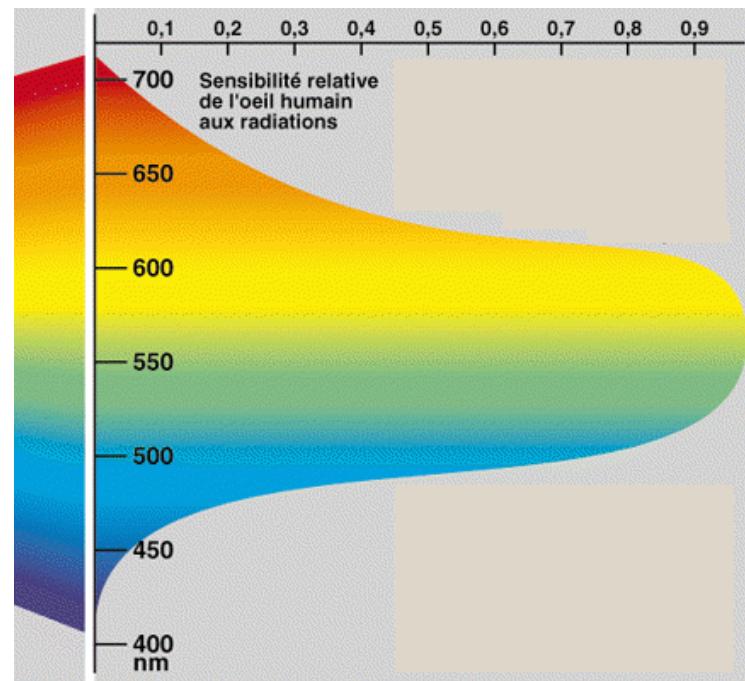
# Luminance « perceptuelle »



Rouge/Bleu



Rouge/Vert



$$Y = 0.299 R + 0.587 V + 0.114 B$$

pondération plus faible

# Espace colorimétrique YCbCr

---

Espace décomposant Luminance **Y** et canaux couleurs **Cb Cr**  
Utilisé par ex. pour la compression et la transmission hertzienne

```
clear all
close all

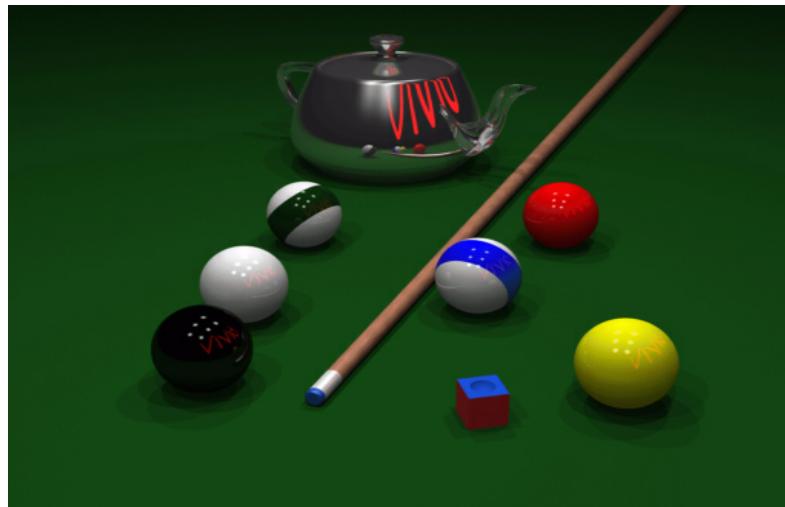
I=double(imread('pool.tif'));
R=I(:,:,1);
G=I(:,:,2);
B=I(:,:,3);

Y=0.299*R+0.587*G+0.114*B;
Cb=0.564*(B-Y)+128;
Cr=0.713*(R-Y)+128;
L=(R+G+B)/3;

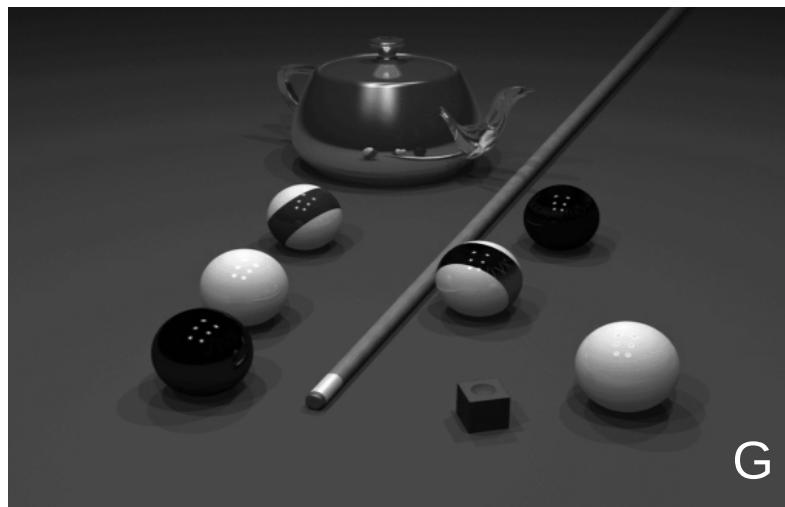
figure, imshow(uint8(I)); title('I');
figure, imshow(uint8(L)); title('L');
figure, imshow(uint8(Y)); title('Y');
```

# Décomposition en canaux RGB

---



R



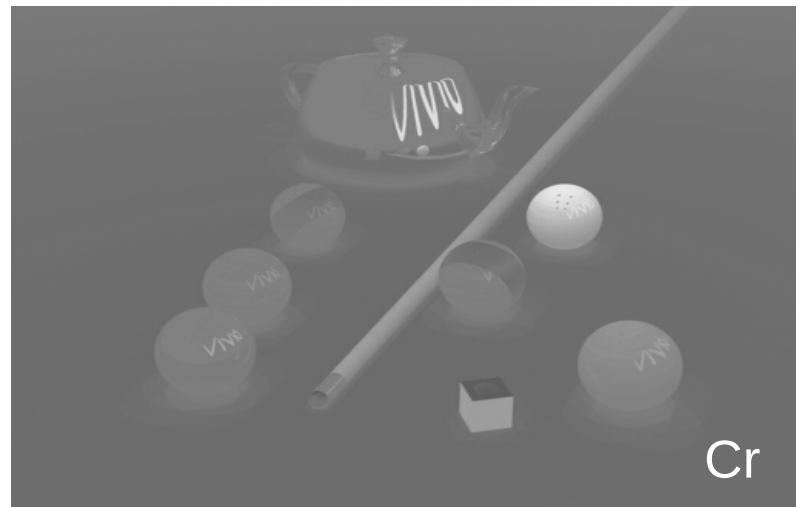
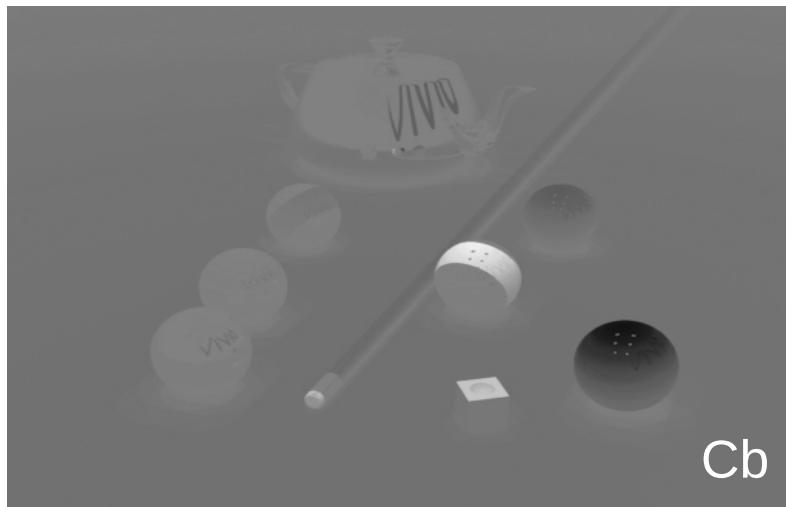
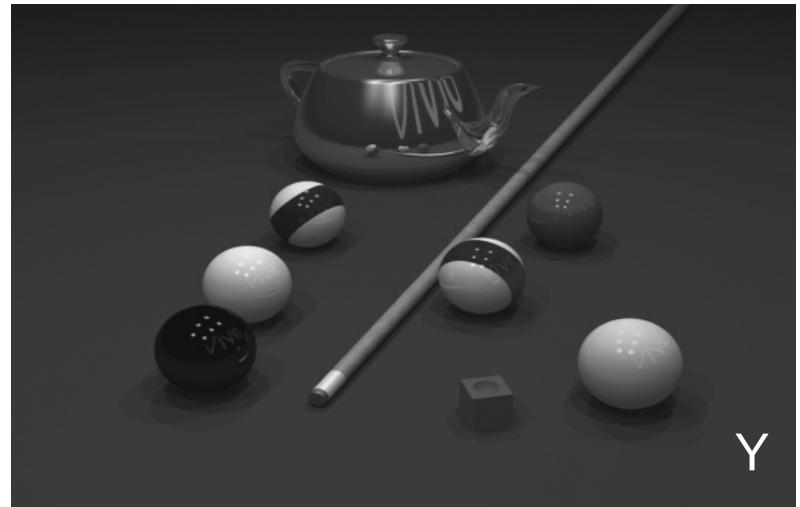
G



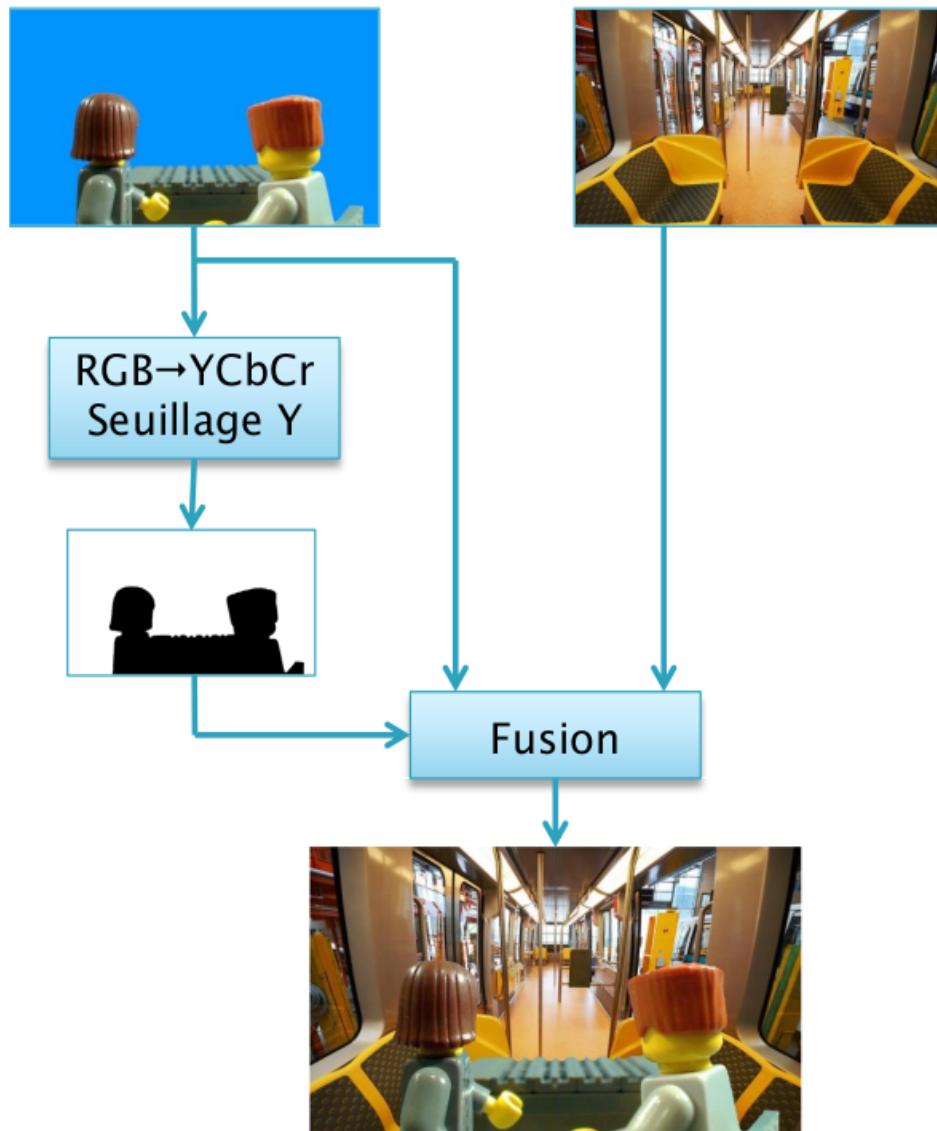
B

# Décomposition en canaux YCbCr

---



# Chroma-Keying



→ Implémentation en TP

# Exercice : Compression YCbCr

---

- Convertir l'image *pool.tif* en YCbCr
- Compresser uniquement en taille les canaux de chrominance CbCr (*imresize*) par un facteur  $r$  qu'on fera varier
- Reconstruire l'image en agrandissant les canaux à leur taille initiale
- Comparer le résultat avec l'image initiale
- Quantifier le gain en taille mémoire



Image initiale



Image compressée  $r = 0.5$

# Exercice : Effet Pencil Sketch

---

- Calculer une carte de contours C de l'image *home.jpg* (edge)
- Convertir l'image en YCbCr, puis modifier le canal Y :

$$Y \leftarrow (255 - \alpha) \times (1 - C) + \beta$$

avec  $\alpha, \beta \in [0, 255]^2$ , des paramètres à régler manuellement

- Repasser en RGB pour obtenir un résultat d'esquisse :



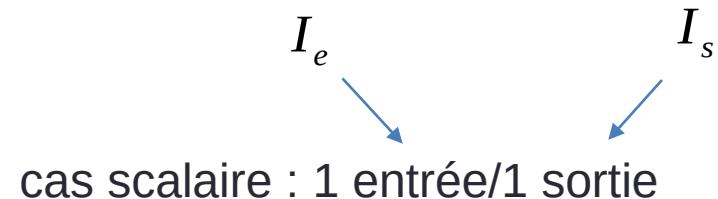
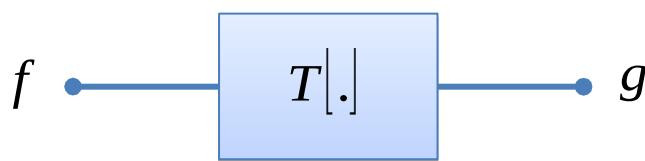
# Plan

---

- Image couleur
  - Format/affichage
  - Visualisation
  - Synthèse
- Espaces couleur caractéristiques
  - Espace YCbCr
  - Applications : compression, esquisse
- Traitements
  - Filtrage linéaire/non linéaire
  - Applications : Anonymisation, débruitage, recouvrement fréquentiel
  - Détection de contours
  - Applications : réhaussement de contraste

# Filtrage linéaire

## Système linéaire



L'entrée et la sortie d'un système (filtre) linéaire  
 $T$  sont reliées par un produit de convolution

Un filtre linéaire est entièrement caractérisé  
par sa réponse impulsionnelle

$$h(t) = T[\delta(t)]$$

impulsion

## Filtre linéaire RIF

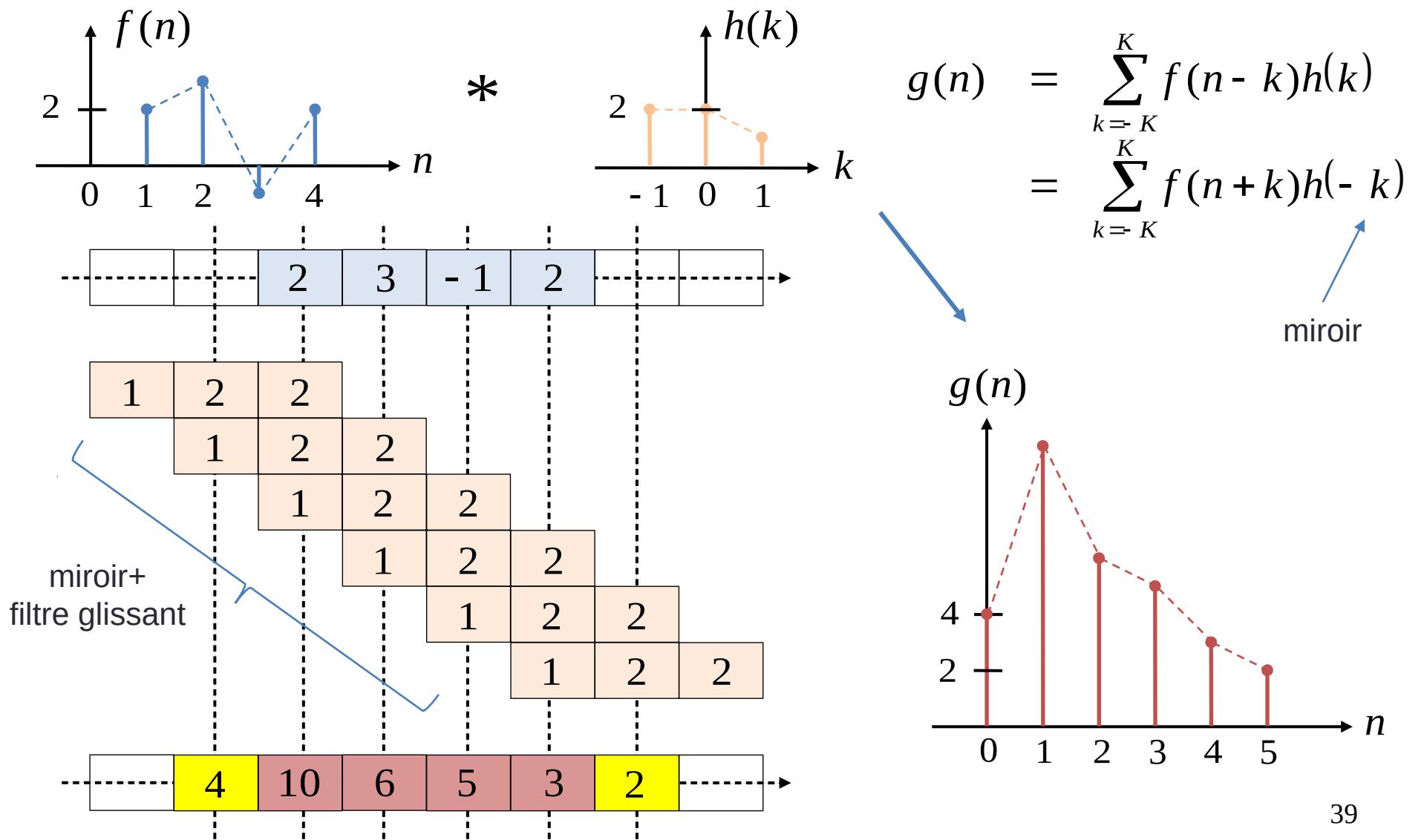
Application d'un filtre linéaire  
discret à réponse  
impulsionnelle finie

$$g(n) = \sum_{k=-K}^K f(n-k)h(k)$$

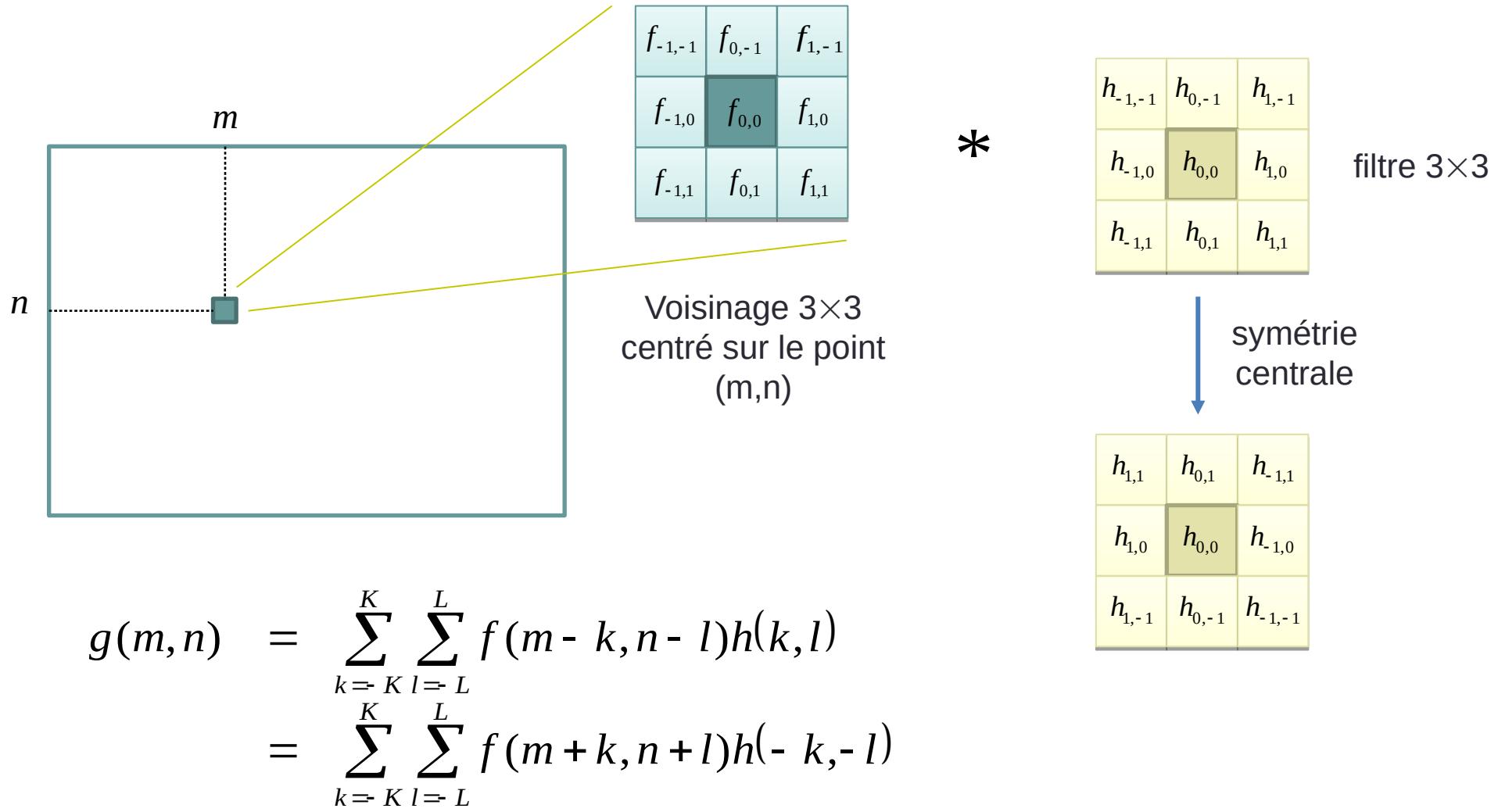
défini sur  $2K+1$  échantillons

$$I_s = I_e * H$$

# Convolution discrète 1D



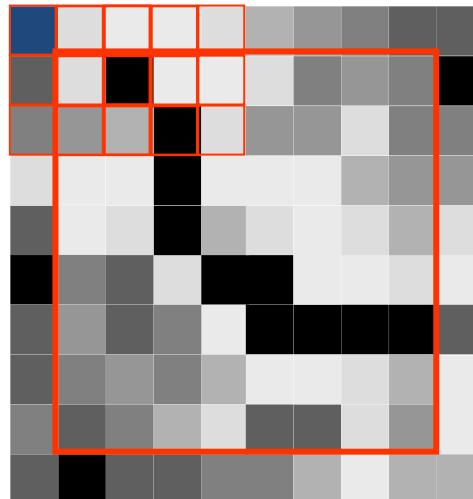
# Convolution discrète 2D



# Exemple de convolution discrète 3x3

Convolution (2D) :

$$h = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$



Novelle valeur de  $C$

$$= (0+5+7+1+5+0+0+0+0)/9 = 3.33$$

Novelle valeur de  $C$

$$= (5+7+6+5+0+0+0+0+0)/9 = 4.44$$

Novelle valeur de  $C$

$$= (7+6+5+0+7+6+4+0+5)/9 = 4$$

$A$

0	5	7	6	5	4	3	2	1	1
1	5	0	7	6	5	2	3	2	0
2	3	4	0	5	3	3	5	2	2
5	7	6	0	6	6	6	4	3	3
1	7	5	0	4	5	6	5	4	5
0	2	1	5	0	0	7	6	5	6
1	3	1	2	7	0	0	0	0	1
1	2	3	2	4	6	7	5	4	7
2	1	2	3	5	1	1	5	2	6
1	0	1	1	2	2	3	6	4	4

$C$

0	5	7	6	5	4	3	2	1	1
1	3	4	4	5	4	3	3	2	0
2	4	4	4	4	4	4	3	2	2
5	4	4	3	4	5	4	4	4	3
1	4	4	3	3	4	5	5	5	5
0	2	3	3	4	5	3	5	5	6
1	3	2	3	5	4	4	3	3	1
1	2	2	3	3	4	4	4	4	7
2	2	2	3	5	3	3	5	4	6
1	0	1	1	2	2	3	6	4	4

# Filtres moyenneur (1/3)

- Filtre rectangle (uniforme)

normalisation

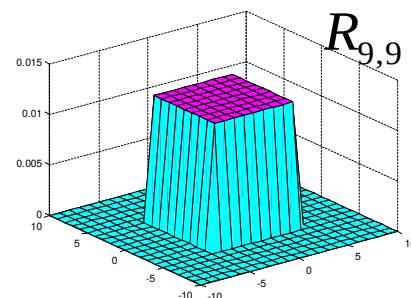
$$\begin{cases} R_{2K+1} = \frac{1}{(2K+1)} [1 \quad \dots \quad \dots \quad \dots \quad 1] \\ R_{2K+1, 2L+1} = R_{2K+1} * R_{2L+1}^T \end{cases}$$

$2K+1$

$$R_3 = R_{3,1} = \frac{1}{3} [1 \quad 1 \quad 1]$$

$$R_{1,3} = R_{3,1}^T = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

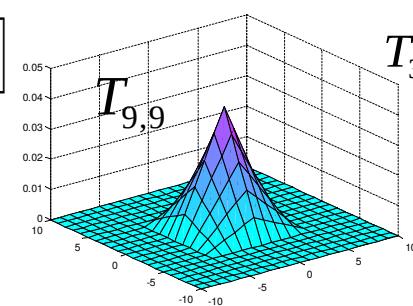
$$R_{3,3} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



plus petit filtre rectangle 2D

- Filtre triangle

$$\begin{cases} T_{2K+1} = \frac{1}{(K+1)^2} [1 \quad 2 \quad \dots \quad K+1 \quad \dots \quad 2 \quad 1] \\ T_{2K+1, 2L+1} = T_{2K+1} * T_{2L+1}^T \end{cases}$$



$$T_{3,3} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

plus petit filtre triangle 2D

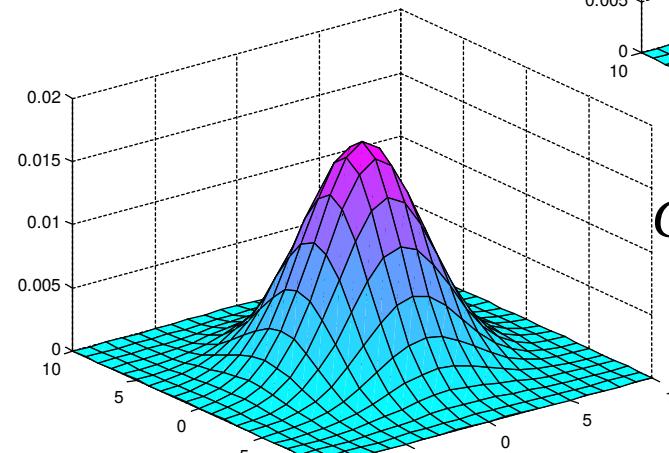
# Filtres moyenneur (2/3)

- Filtre Gaussien

$$h(k) = \frac{e^{-\frac{k^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$$

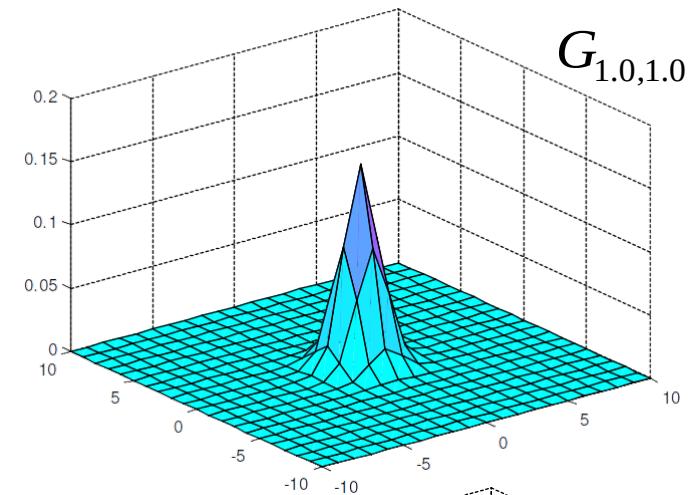
$$h(k, l) = \frac{e^{-\left(\frac{k^2}{2\sigma_k^2} + \frac{l^2}{2\sigma_l^2}\right)}}{2\pi\sigma^2} \text{ version 2D}$$

$G_{\sigma_k, \sigma_l}$



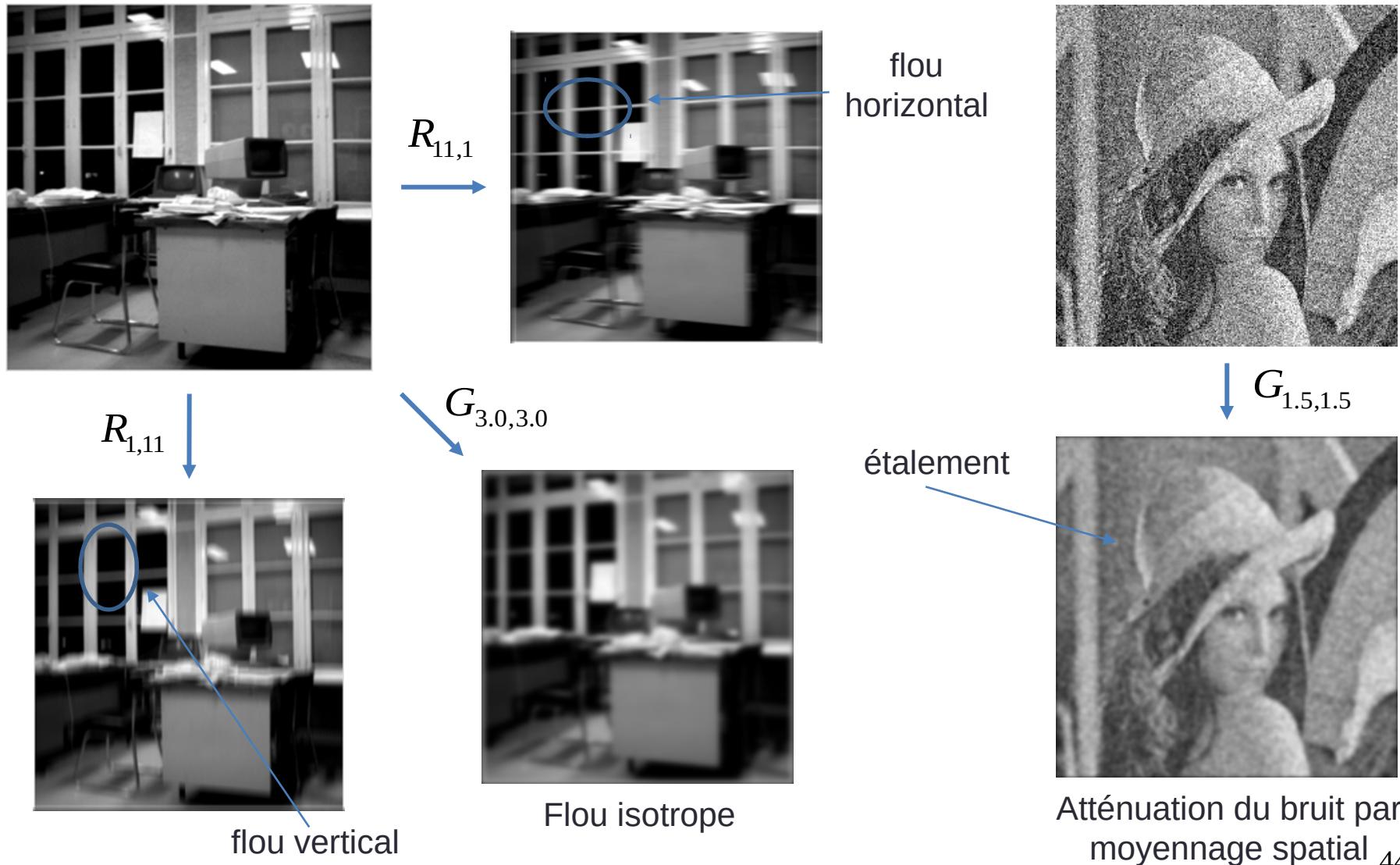
$G_{3.0,3.0}$

troncature trop forte



$G_{1.0,1.0}$

# Filtres moyenneur (3/3)



# Exercice : Anonymisation

---

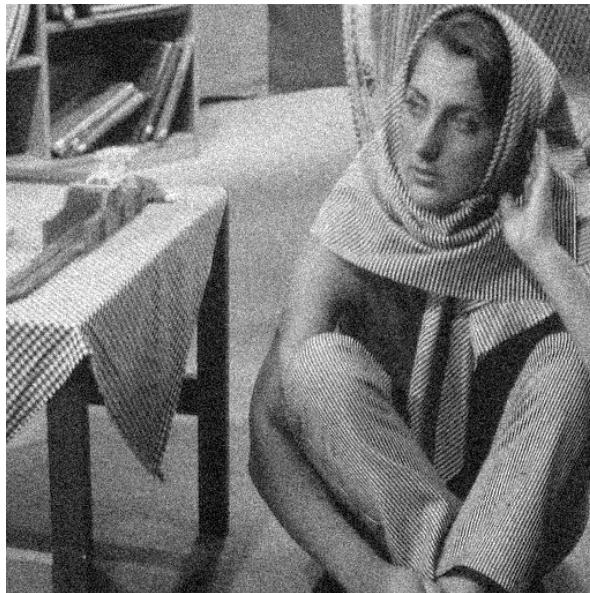
- Identifier les centres des visages à anonymiser sur l'image *street.png* (*ginput*).
- Créer un masque binaire avec des 1 autour de ces centres (*meshgrid*, équation du cercle)
- Flouter l'image avec un filtre moyenneur (rectangle uniforme)
- Combiner les deux images grâce au masque pour obtenir une image où seuls les visages sont floutés.



# Exercice : Débruitage (linéaire)

---

- Implémenter les filtrages linéaires suivants :
  - Filtrage moyenneur : valeur moyenne dans un voisinage (`conv2`)
  - Filtrage Gaussien : convolution par une Gaussienne 2D (`fspecial`, `conv2`) (visualisation des filtres avec `surf`)
- Tester sur les images : `barbara_awgn_noise.png`, et `cameraman_sp_noise.png`



- Pour quelles images les filtrages s'avèrent-ils satisfaisants ? Pourquoi ?

# Exercice : Débruitage (non linéaire)

---

- Pour *cameraman\_sp\_noise.png*, les filtrages linéaires ont-ils été satisfaisants ?
- Implémenter le filtre suivant :
  - Filtrage médian : valeur médiane dans un voisinage

Algo : Parcourir tous les pixels de l'image

```
for i=1+v:h-v, for j=1+v:w-v
```

Déterminer un voisinage 2D (de taille  $(2v+1) \times (2v+1)$ )

```
window = img(i-v:i+v, j-v:j+v);
```

Calculer la valeur médiane sur l'ensemble des pixels

```
I_filtre(i,j) = median(window(:));
```



Filtre médian



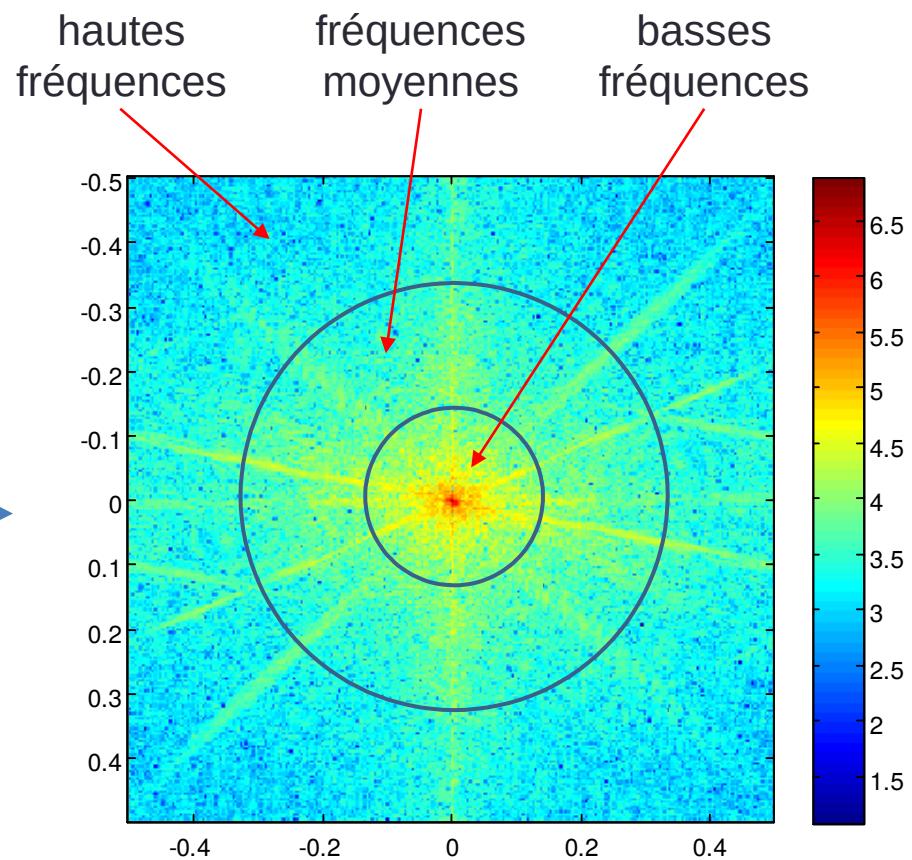
# Rappels Transformée de Fourier

- Interprétation fréquentielle :



Image naturelle

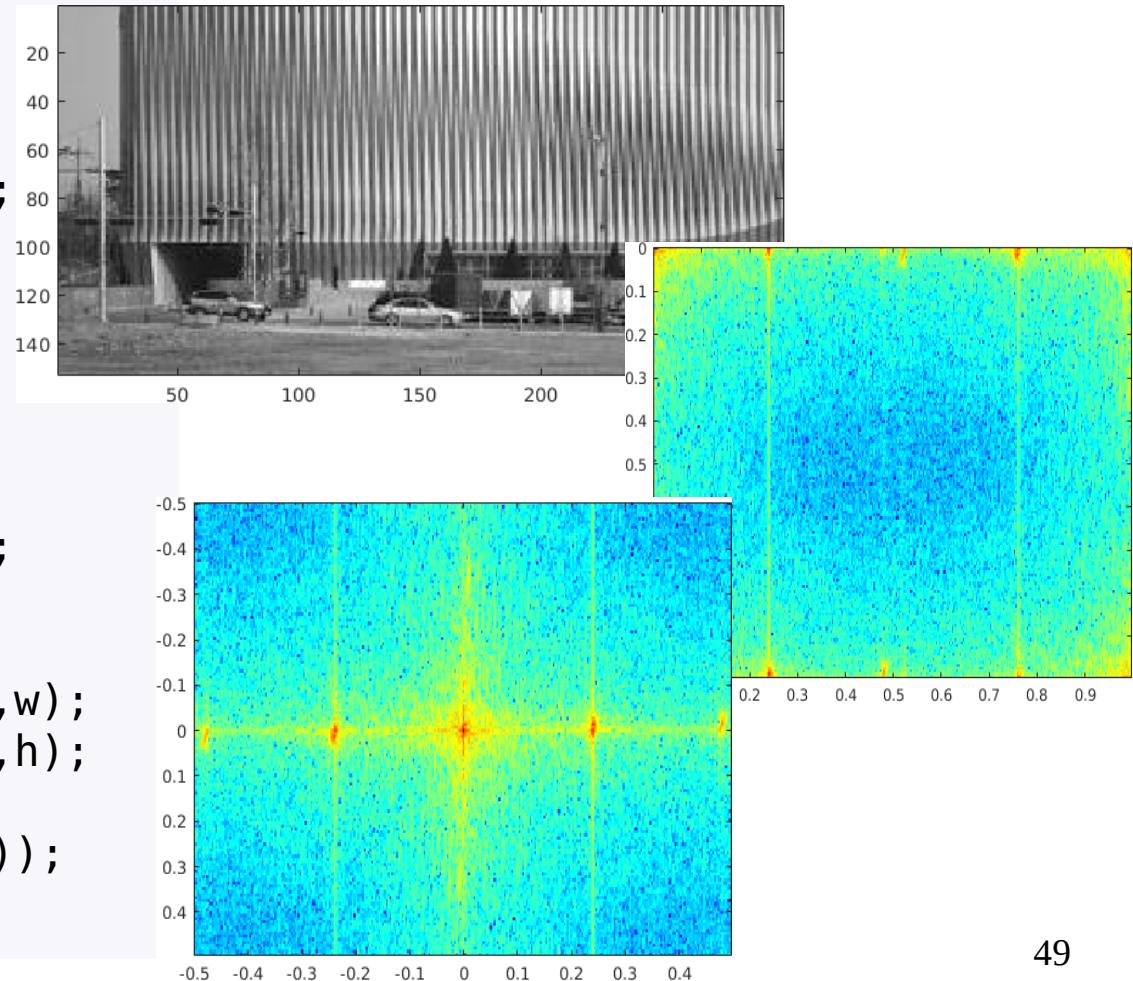
$$|TF(.)| \rightarrow$$



# Rappels Transformée de Fourier

- Affichage :

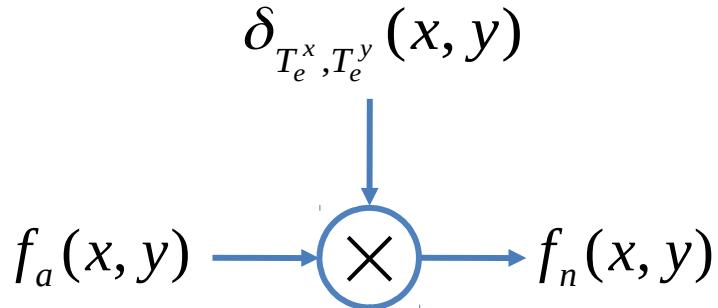
```
clear all  
close all  
  
A=imread('building.jpg');  
figure, imagesc(A)  
colormap(gray(256))  
[h,w]=size(A);  
B=log10(abs(fft2(A)));  
fx=linspace(0,1-1/w,w);  
fy=linspace(0,1-1/h,h);  
figure, imagesc(fx,fy,B);  
colormap(jet(256))  
  
fx=linspace(-0.5,0.5-1/w,w);  
fy=linspace(-0.5,0.5-1/h,h);  
figure  
imagesc(fx,fy,fftshift(B));  
colormap(jet(256))
```



# Transformée de Fourier discrète 2D

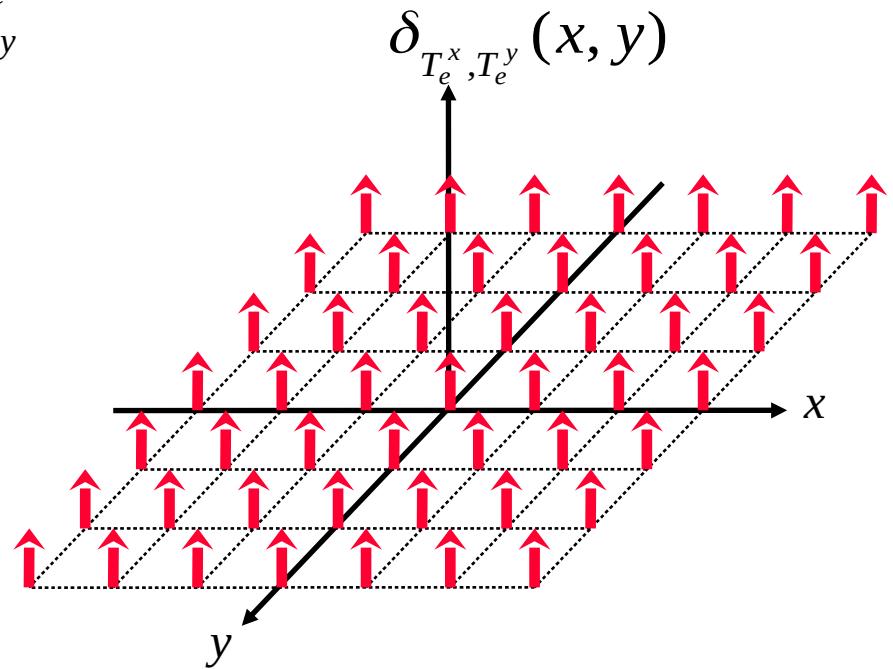
---

$$f_n(x, y) = \begin{cases} f_a(x, y) & \text{pour } \begin{cases} x = mT_e^x \\ y = nT_e^y \end{cases} \\ 0 & \text{sinon} \end{cases}$$



$$f_n(x, y) = f_a(x, y) \delta_{T_e^x, T_e^y}(x, y)$$

séquence discrète  
notée  $f(mT_e^x, nT_e^y)$  où  $f(m, n)$

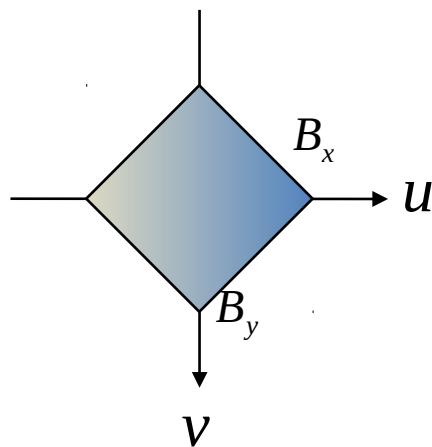


Peigne de Dirac 2D

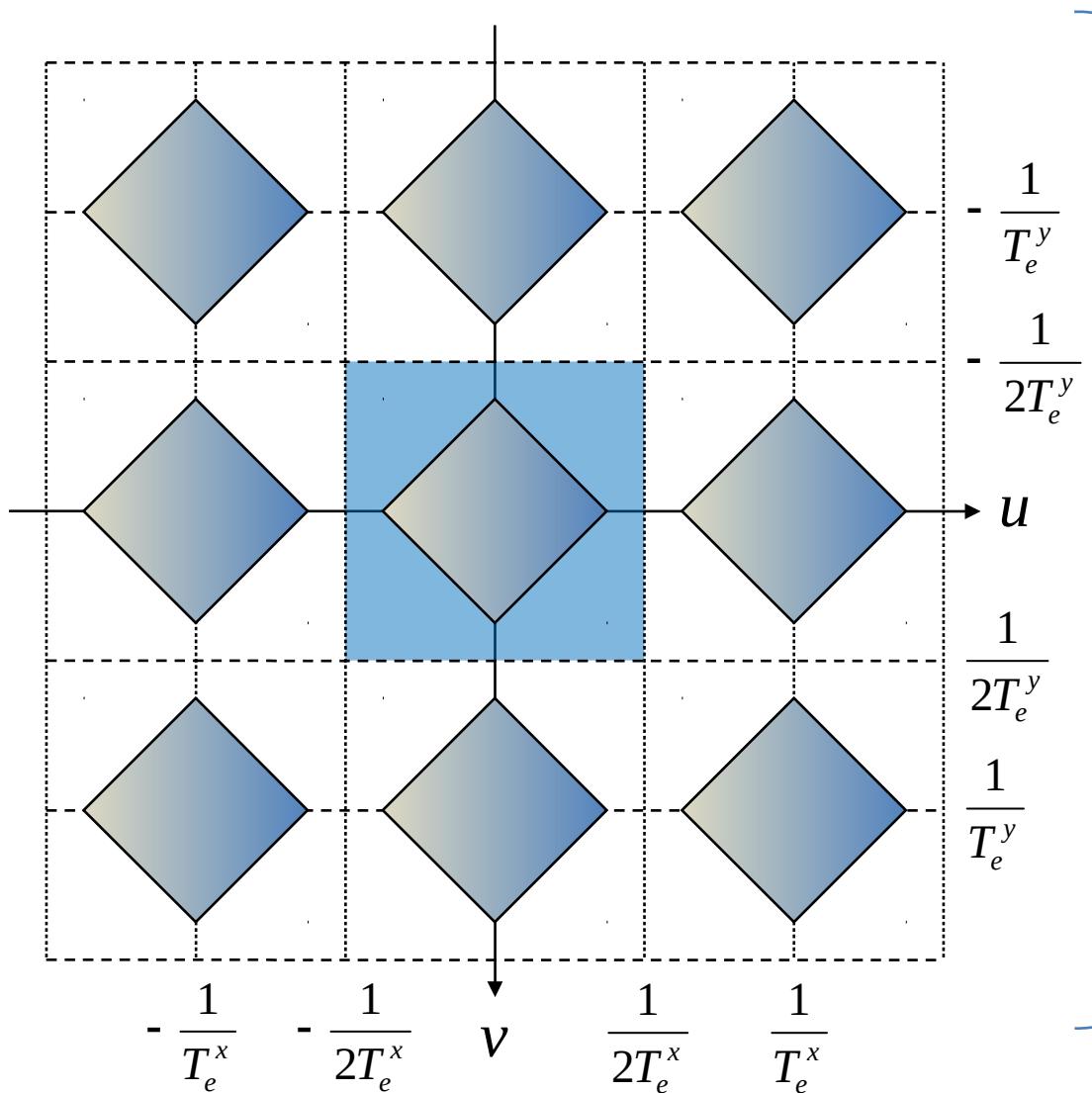
$$\delta_{T_e^x, T_e^y}(x, y) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \delta(x - mT_e^x, y - nT_e^y)$$

# Transformée de Fourier discrète 2D

( $f_e^x, f_e^y$ )-périodique

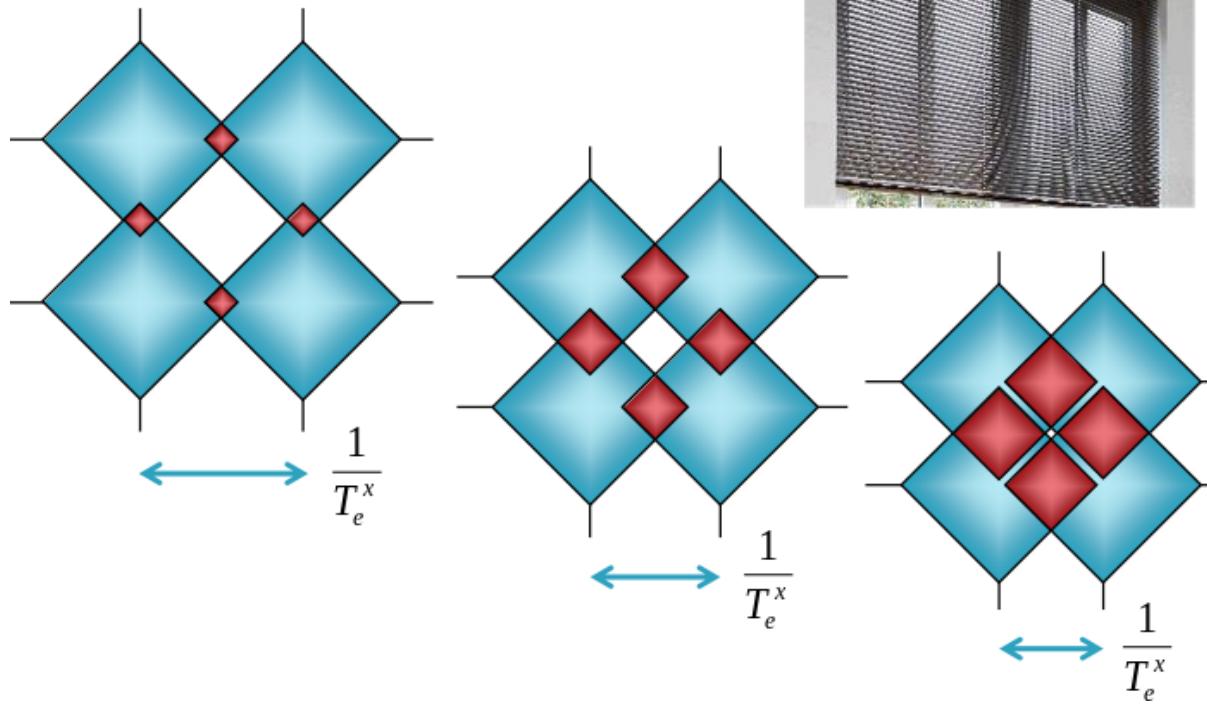
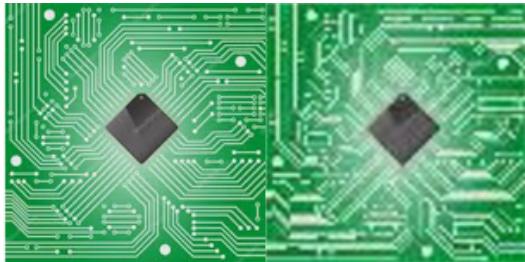


Signal 2D à bande  
passante limitée



# Recouvrement fréquentiel (aliasing)

- Des hautes fréquences aux basses fréquences



# Exercice : Filtrage anti-aliasing

---

- Observer ce problème de repliement de spectre liés à l'échantillonnage spatial en sous-échantillonnant *barbara.png* ou *bricks.png* d'un facteur 4.
- Observer également la transformée de Fourier des images.
- Lisser l'image initiale par un filtre passe-bas d'anti-repliement, type Gaussienne, avant le sous-échantillonnage. On comparera ce résultat également à celui obtenu par `imresize`.



Image initiale ( $h \times w$ )



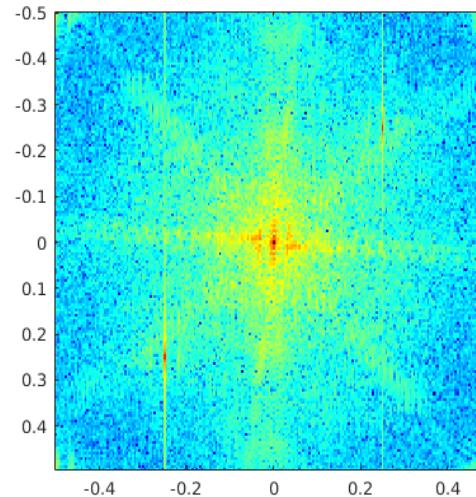
Image sous-échantillonnée ( $h/4 \times w/4$ )



Image filtrée et sous-échantillonnée  
( $h/4 \times w/4$ )

# Filtrage bruit fréquentiel

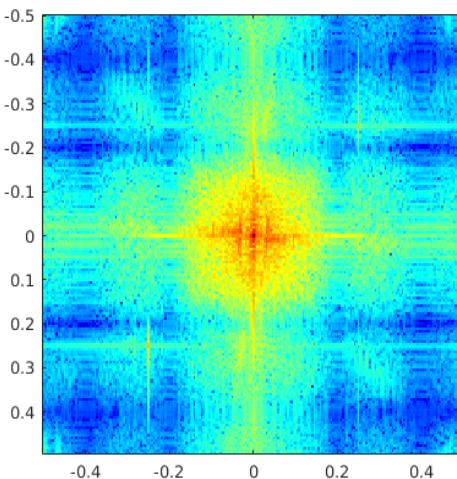
- Bruit fréquentiel :



- Inefficacité des filtrages spatiaux classiques :



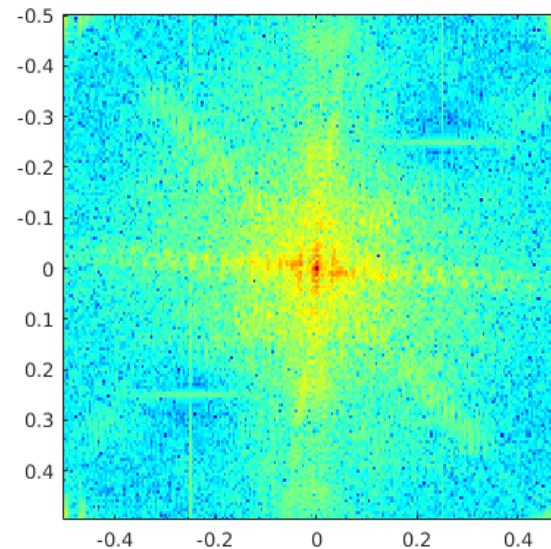
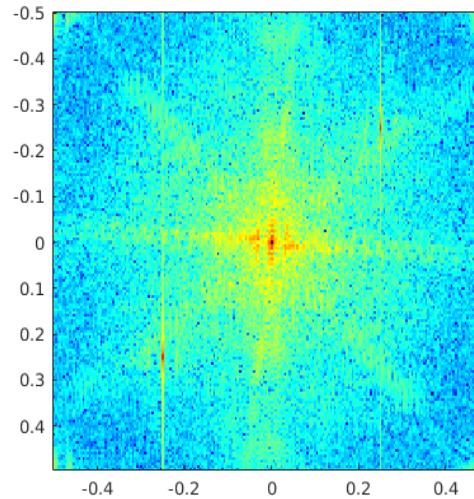
Filtre moyenneur  
carré 5x5



# Filtrage bruit fréquentiel

---

- Filtrage de *pise\_ext.bmp* par un filtre coupe bande en TP



# Détection de contours

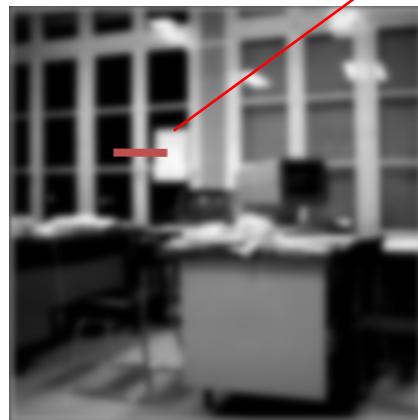
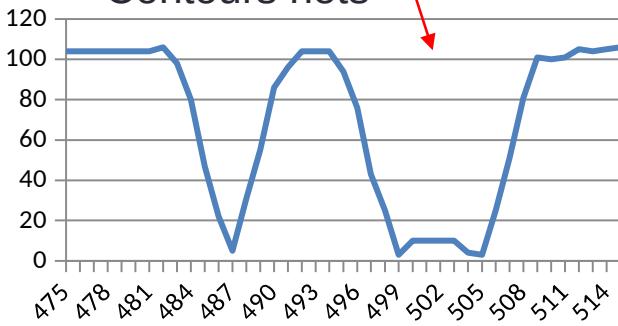
- Rappels :

- Bord ou limite d'une région
- Séparation ou frontière entre régions

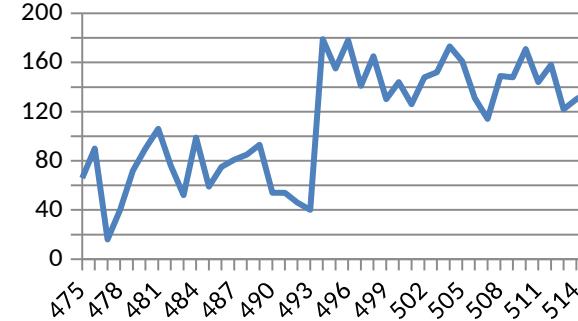
variation plus ou moins rapide d'un front d'intensité



Contours nets



Contours flous



Contours bruités

# Exercice : Détection de contours

- Créer les deux filtres de Sobel :

moyennage (lissage) dans la direction orthogonale à  
la direction de dérivation

Filtres de  
Sobel

$$\left\{ \begin{array}{l} S_x = \frac{1}{2} [1 \quad 0 \quad -1] * \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ déivateur horizontal} \\ S_y = S_x^T \text{ déivateur vertical} \end{array} \right.$$

- Convoluer l'image *cameraman.tif* avec ces deux filtres et observer les deux réponses

## Approche de Canny (1/2)

- Principe

Les dérivées (d'ordre n) peuvent être approchées par la convolution avec les dérivées (d'ordre n) d'une gaussienne

Ordre 1     $\nabla I \approx I * \nabla G$

lissage et dérivation

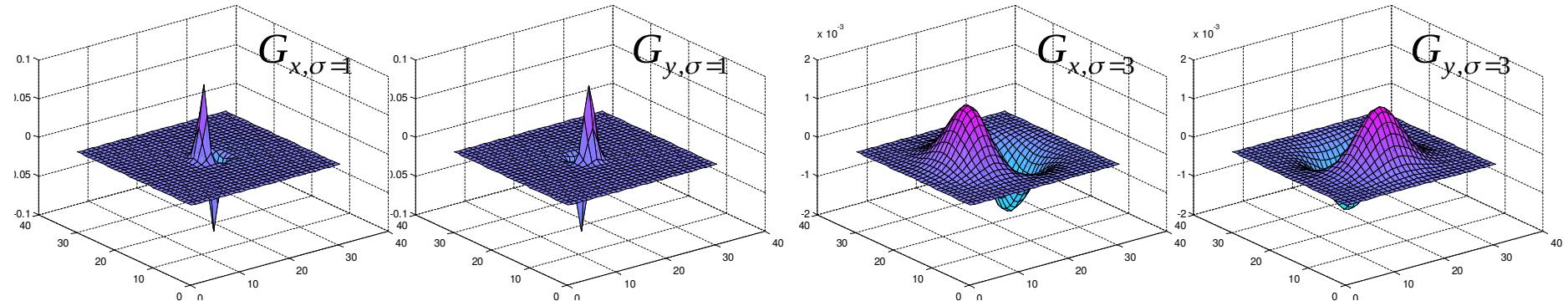
avec  $G(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)}$

paramètres d'échelle contrôlant la force du lissage et de la dérivation

$$\nabla G = \begin{bmatrix} G_x(x, y) = -\frac{x}{2\pi\sigma_x^3\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)} \\ G_y(x, y) = -\frac{y}{2\pi\sigma_x\sigma_y^3} e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)} \end{bmatrix} \longrightarrow$$


Filtres 2D RIF par échantillonnage et troncature

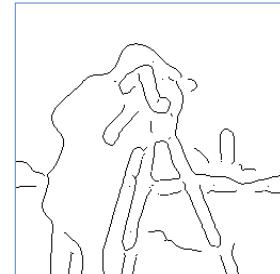
# Approche de Canny (2/2)



Norme du gradient à petite échelle



Norme du gradient à grande échelle

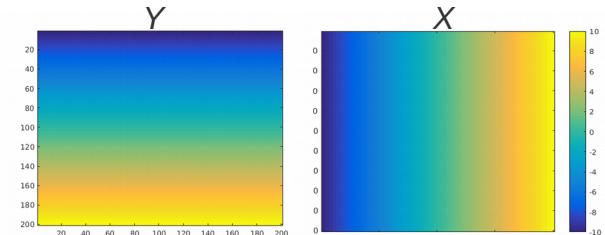


# Exercice : Détection de contours

- Créer un filtre issu de la dérivée d'une Gaussienne 2D :  $g(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$

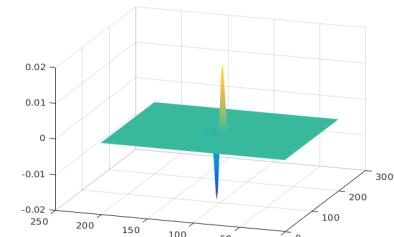
1) Utiliser `meshgrid` pour créer un pavage d'indices :

```
P = -10:10;  
[X, Y] = meshgrid(P, P);
```



2) On retrouve l'expression des dérivées en x et y de  $g(x,y)$  pour leur fournir les cartes du `meshgrid` et obtenir deux filtres détecteurs de contours  $G_x$   $G_y$  :

```
sig = 0.05;  
Gx = -X/(2*pi*sig^4).*exp(-(X.^2+Y.^2)/(2*sig^2));  
figure, surf(Gx), shading interp;
```



3) Convoluter les filtres avec l'image pour obtenir les deux réponses et calculer leur norme pour obtenir une carte de détection de contours

4) Utiliser cette carte dans l'application Pencil Sketch. Comparer avec le résultat de edge



# Réhaussement de contraste

$$\Delta I = I_{xx} + I_{yy}$$

$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$   
Filtre 1D RIF horizontal

$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

Filtre 1D RIF vertical

somme

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Filtre 2D RIF  
Forme en croix (4 voisins)

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Forme diagonale (4 voisins)

Invariance par rotation

$$\frac{1}{2} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Forme cumulée « isotrope » (8 voisins)

# Exercice : Réhaussement de contraste

- Implémenter le réhaussement de contraste en utilisant un filtre isotrope :

$$I_s = I - \beta \Delta I$$

paramètre de contrôle

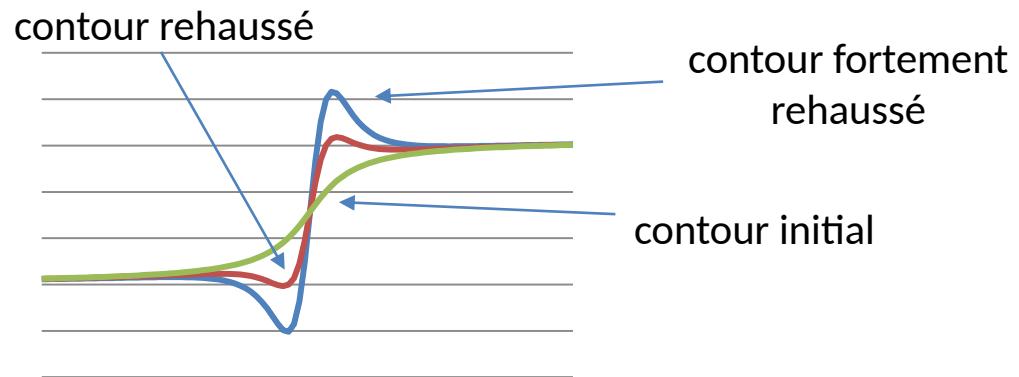


Image initiale



Rehaussement moyen



Rehaussement fort

# ANNEXES

---

# Exercice Bonus : Filtrage bilatéral

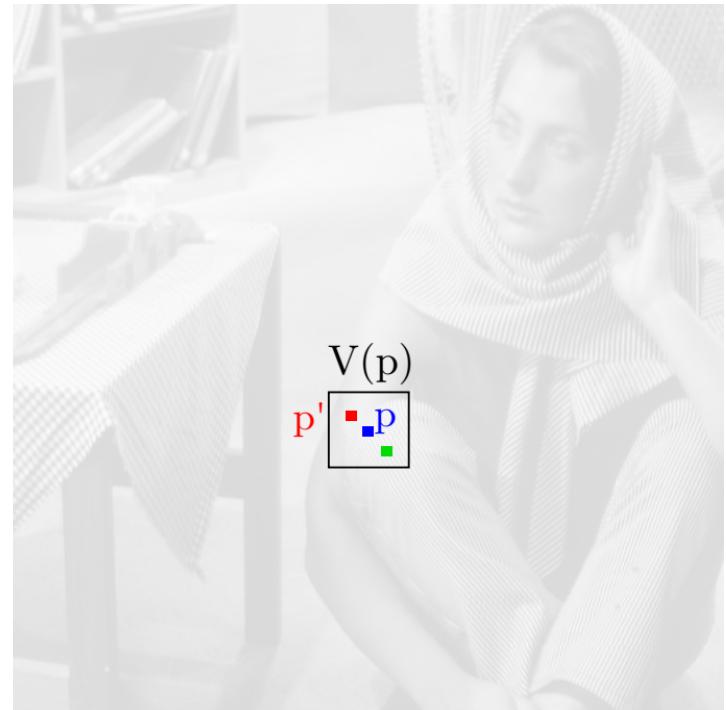
Moyenne pondérée des intensités des voisins selon leur proximité couleur et spatiale :

$$I_F(p) = \frac{\sum_{p' \in V(p)} \omega(p, p') I(p')}{\sum_{p' \in V(p)} \omega(p, p')}$$

$$\omega(p, p') = \exp \left( -\frac{\|I(p) - I(p')\|_2}{2\sigma_c^2} - \frac{(p - p')^2}{2\sigma_s^2} \right)$$

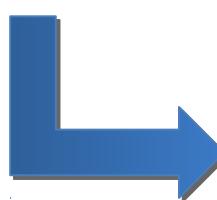
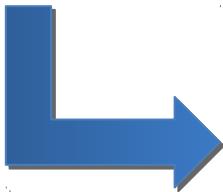
Pondération  
couleur

Pondération  
spatiale



# Exercice Bonus : Filtrage bilatéral

---



# Exercice Bonus : Filtrage bilatéral

---



Image initiale

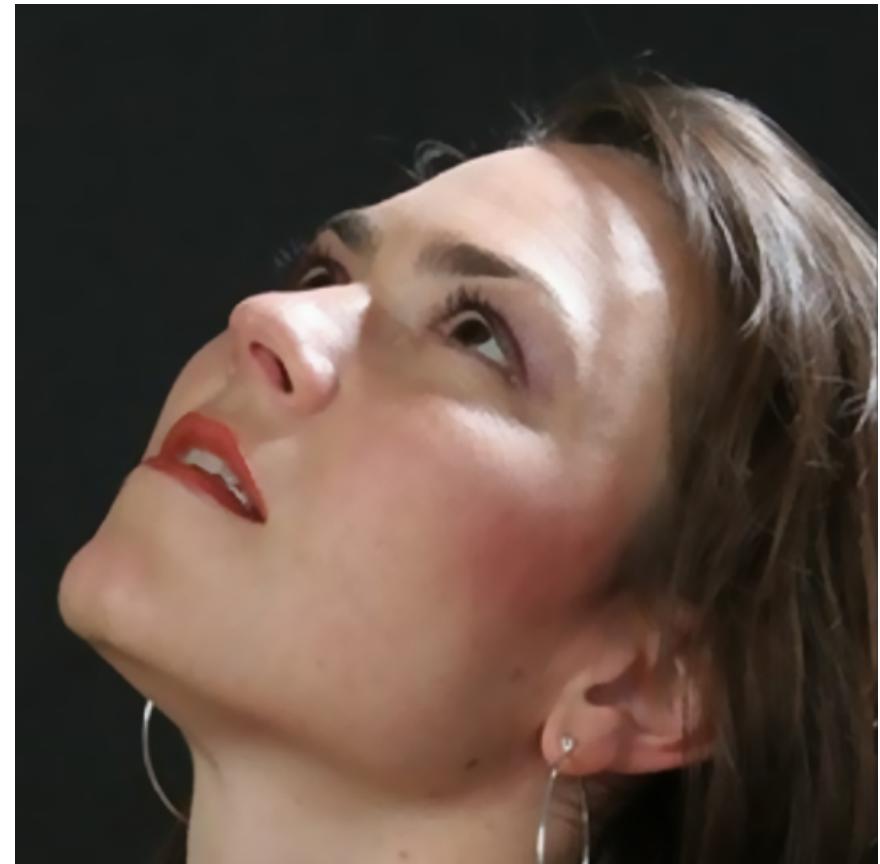


Image filtrée

# Exercice Bonus : Filtrage bilatéral

---

```
img = double(imread('cameraman.tif'));
[h,w,z] = size(img);

%Paramètres (à faire varier)
v_size = 5; sigma_s = 1; sigma_c = 4;

img_bf = double(img*0);
%Parcours de tous les pixels de l'image
for i=1:h
    for j=1:w
        %Sélection d'une fenêtre (2*v_size+1)x(2*v_size+1) ajustée au bord
        for ii=max(i-v_size,1):min(i+v_size,h)
            for jj=max(j-v_size,1):min(j+v_size,w)
                %A compléter
            end
        end
    end
end
end
```