# Introduction au traitement d'images

Enseignement intégré
TSIG3 | Télécommunications 2025-2026

**Rémi Giraud**
*remi.giraud@enseirb-matmeca.fr*
https://remi-giraud.enseirb-matmeca.fr/

# Plan

- **Image numérique**
  - Format/affichage
  - Visualisation
  - Synthèse
- **Espaces couleurs**
  - Espace YcbCr
    Applications : compression, esquisse, incrustation (TP)
- **Traitements**
  - Filtrage linéaire & non linéaire
    Applications : anonymisation, débruitage
  - Opérateurs de morphologie mathématique
  - Espace fréquentiel, transformée de Fourier
    Applications : recouvrement fréquentiel
  - Détection de contours
    Applications : réhaussement de contraste

# Objectifs

## Traitement d'images

- Culture du traitement d'images et des applications
- Comprendre le système de vision humain et ce qu'est une image couleur
- Savoir afficher de manière pertinente n'importe quel contenu 2D
- Savoir identifier des dégradations et des solutions adéquates
- Outils : changement d'espace, filtrage, Fourier, compression, ...

## Programmation Python

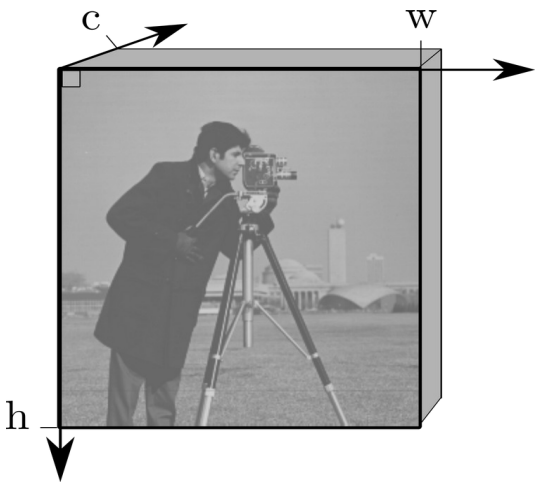
- Prise en main de Spyder
- Rappels de syntaxe Python (import, indentation, boucles, fonctions, ...)
- Librairies *numpy, matplotlib, scikit-image*
- Manipulation de tableaux (*np.array, 0:L-1, axis=X, opérations vectorielles)*

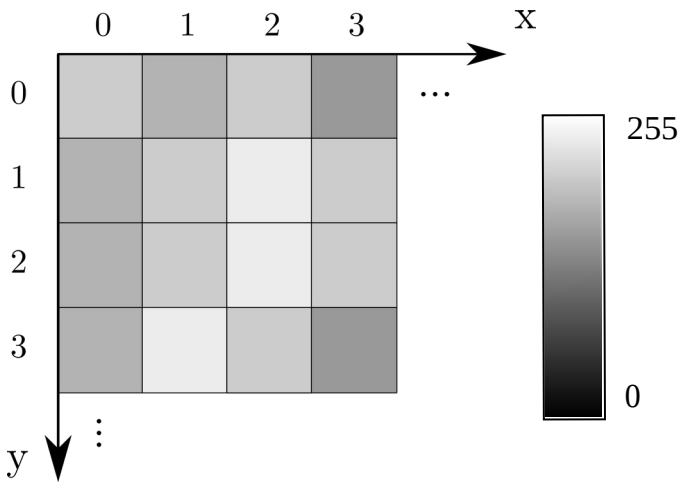
## Qu'est-ce qu'une image numérique ? (1/2)

- Image synthétisée ou obtenue par un dispositif de numérisation
- Processus de discrétisation du monde réel
- Résolution d'une image : nombre de pixels sur une longueur donnée

## Qu'est-ce qu'une image numérique ? (2/2)

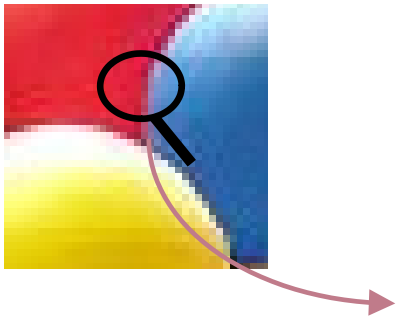
- Une image 2D I est un tableau à deux dimensions associée a :
  - Une taille : h x w
  - Un nombre de canaux : c (ex. pour une image couleur c=3)
- Chaque élément de ce tableau est un pixel, associé à :
  - Une position (i,j) avec i∈[0,h-1] et j∈[0,w-1]
  - Une couleur ou intensité I(i,j) (format usuel [0, 255])



## Pour une image couleur

- 3 composantes/canaux
  - Rouge (R), Vert (V), Bleu (B)



| | | | |
|---|---|---|---|
| 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

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

R

⊕

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

V

⊕

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

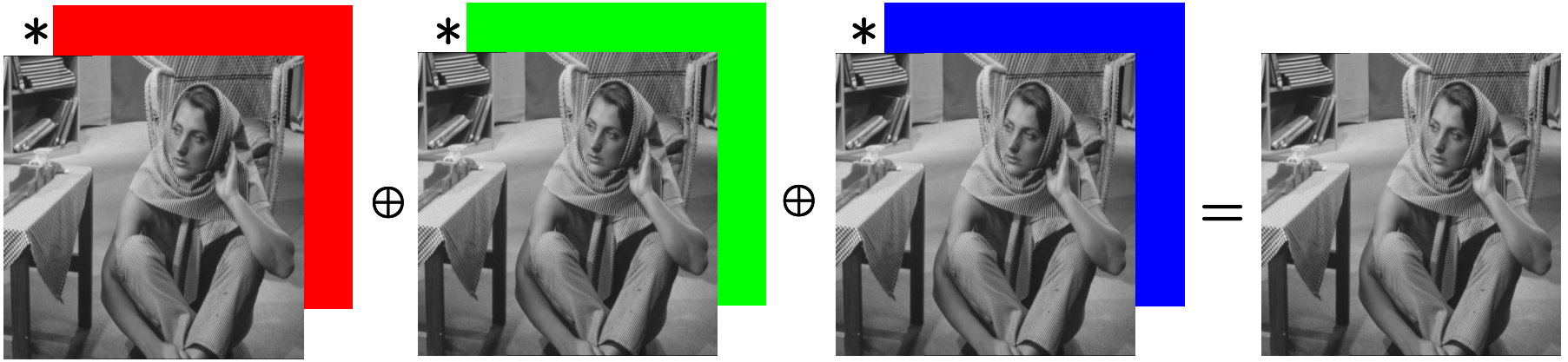
B

## Comment afficher une image couleur ?

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

## Comment afficher une image couleur ?

- En niveaux de gris : Triplication du seul canal L

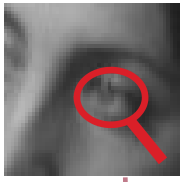



L L L Image en « niveaux de gris »

- Note : pour passer facilement d'une image couleur à une image à un seul canal qu'on peut afficher en « niveaux de gris », on peut calculer la luminance **L : (R+V+B)/3**

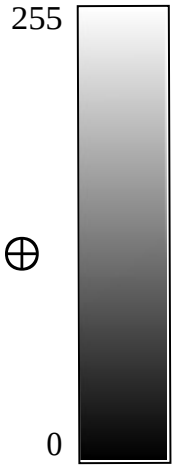
## Comment afficher une image avec un seul canal ?

- En fausses couleurs ou couleurs indéxées :
  - Une palette (table de correspondance couleur) associe une couleur à chaque intensité scalaire



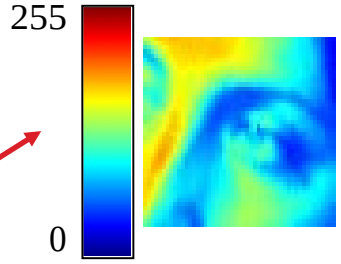
| | | | | | |
|---|---|---|---|---|---|
| 60 | 64 | 62 | 62 | 55 | 51 |
| 59 | 73 | 98 | 90 | 66 | 54 |
| 72 | 105 | 167 | 170 | 120 | 74 |
| 88 | 91 | 188 | 202 | 184 | 150 |
| 103 | 77 | 191 | 205 | 203 | 190 |
| 75 | 131 | 208 | 209 | 207 | 202 |

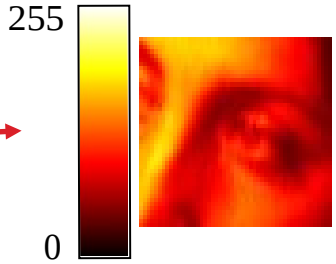
Intensité scalaire

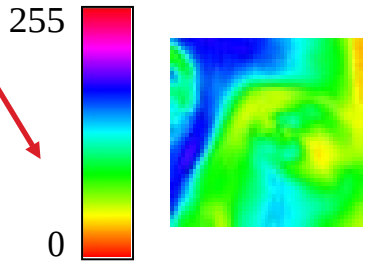



Palette
(niveaux de gris)

Autres palettes

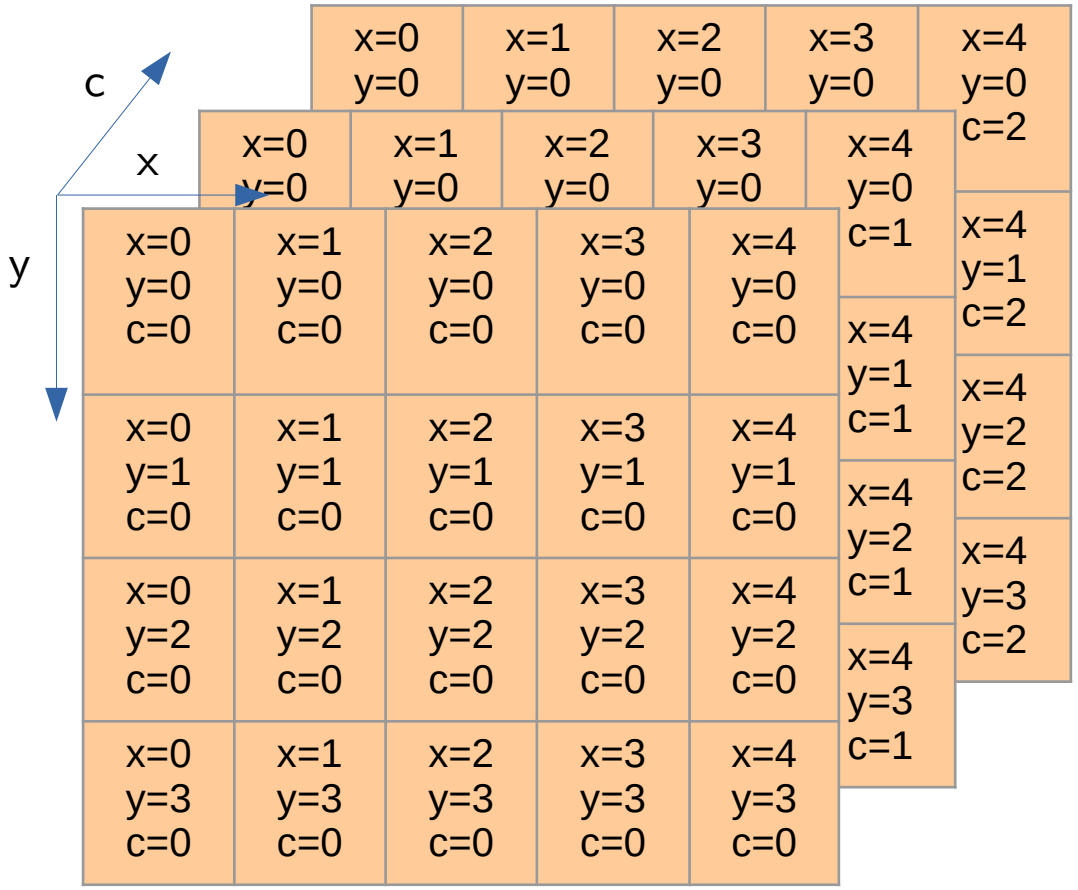


## Conventions

- Accès à une image RGB = Accès à un tableau tridimensionnel




~~img[x,y,canal]~~

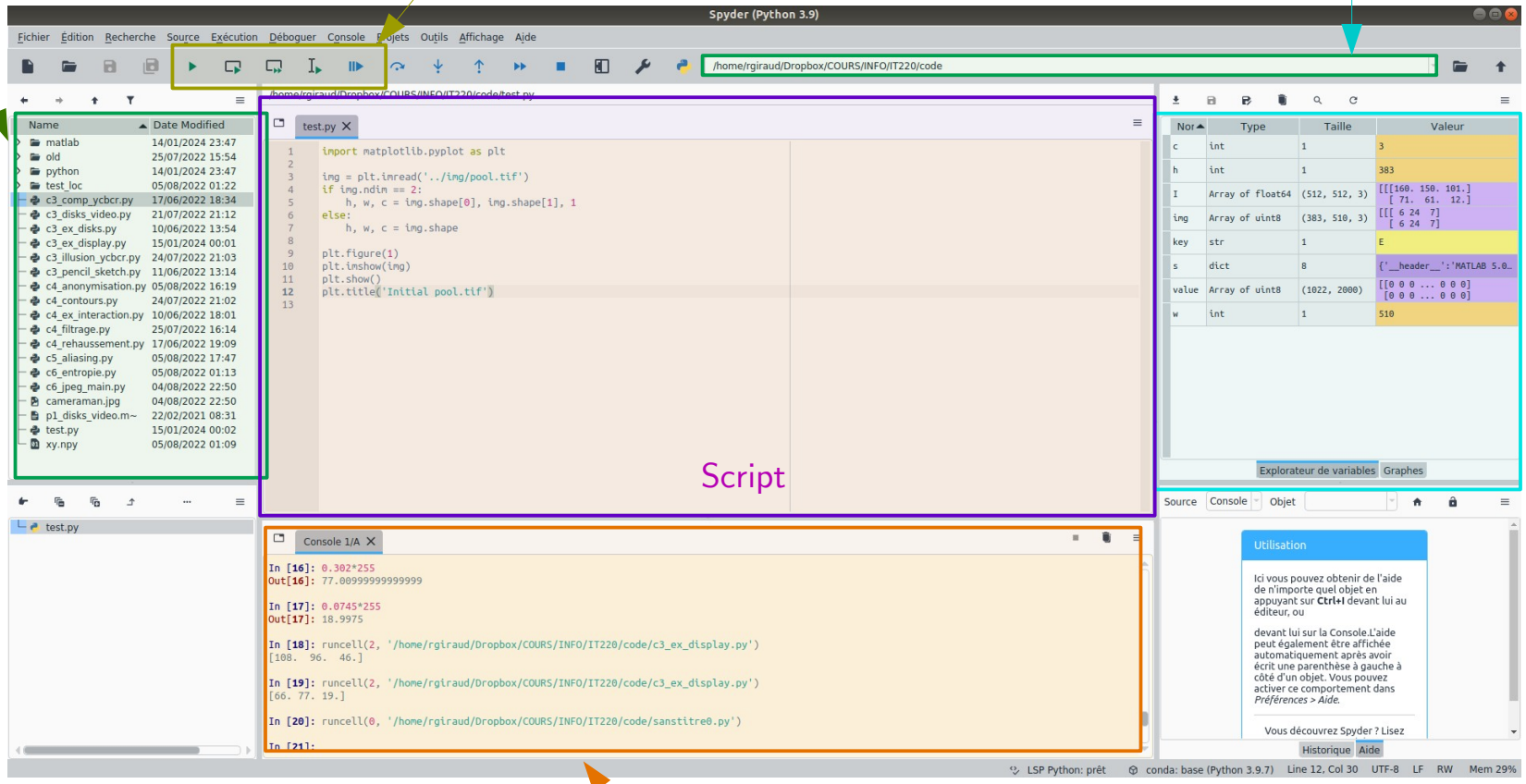
img[ligne,colonne,canal]
=
img[y,x,canal]

# Interface de Spyder

Variables déclarées
(prendre l'habitude de vérifier
leur taille)

Dossier courant (créer et se placer
dans un dossier spécifique au cours)

Exécuter le script




Console : Sortie d'affichage standard
Permet également de taper des commandes

# Lecture et affichage

**Dataset d'images :** https://remi-giraud.enseirb-matmeca.fr/teaching/

**Bibliothèques :** *matplotlib* : chargement, affichage
*numpy* : calcul
*scikit-image* : espace couleur, redimensionnement
*scipy* : convolution

## Image en « vraies couleurs »

```
>> import matplotlib.pyplot as plt
>> img = plt.imread('../img/bdx.jpg')
>> whos (dans la console)

Variable   Type       Data/Info
------------------------------
img        ndarray    383x510x3: 585990 elems, type `uint8`, 585990 bytes

>> plt.figure(1)
>> plt.imshow(img)
>> plt.show()
```

hauteur

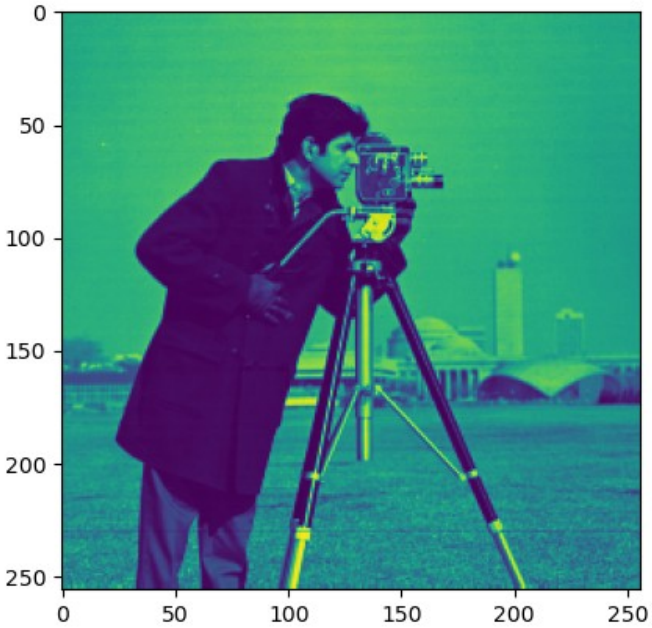
largeur

« vraies » couleurs

## Image en « fausses couleurs »

```
>> img = plt.imread('./img/cameraman.tif')
>> plt.figure(1)
>> plt.imshow(img)
>> plt.show()
```

Et dans l'explorateur de fichiers, l'image ressemble à cela ?

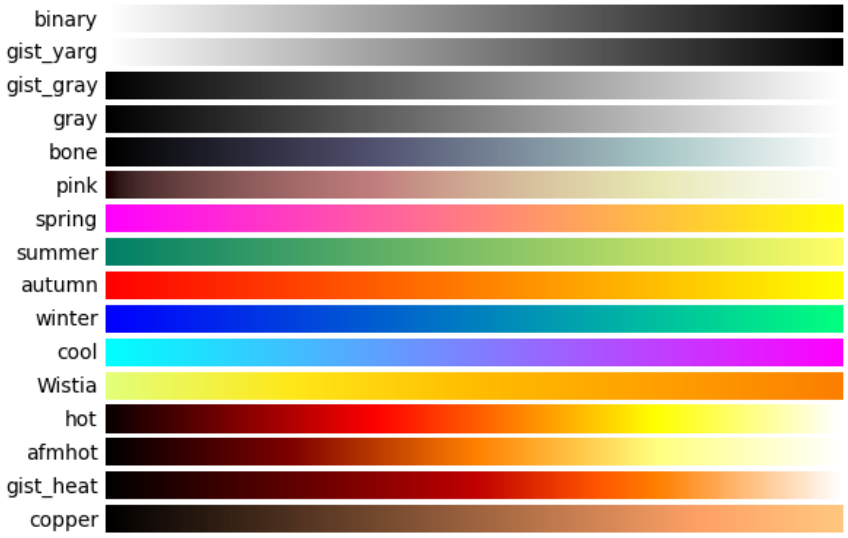


## Palette couleur

R, G, B

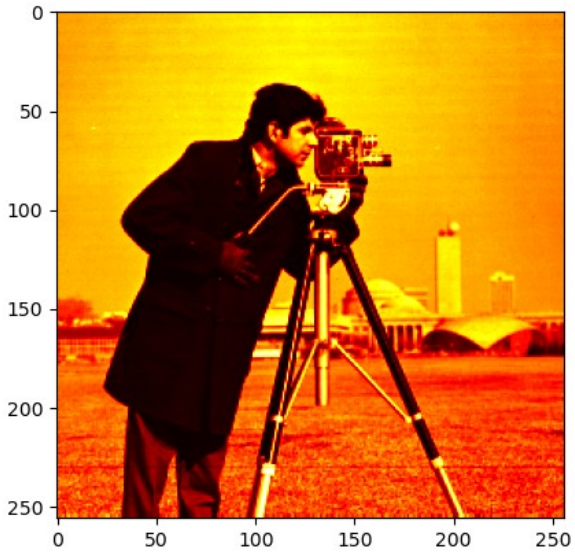
```
>> from matplotlib import cm
>> map = cm.gray(range(256))

[[0.         0.         0.         1.        ]
 [0.00392157 0.00392157 0.00392157 1.        ]
 [0.00784314 0.00784314 0.00784314 1.        ]
 ...
 [0.99215686 0.99215686 0.99215686 1.        ]
 [0.99607843 0.99607843 0.99607843 1.        ]
 [1.         1.         1.         1.        ]]
```

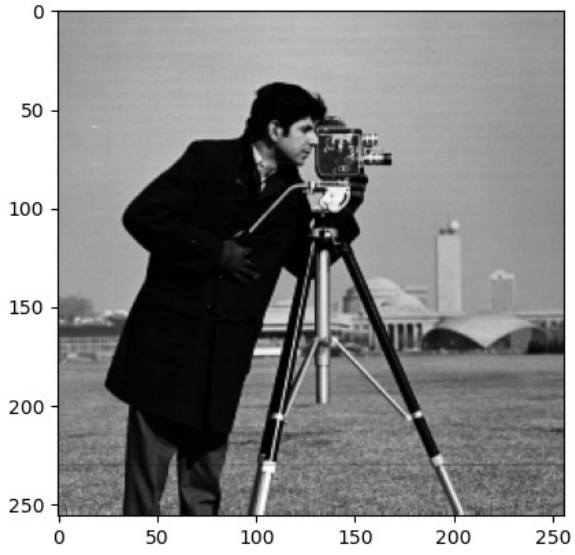

## Autres palettes



## Image en « fausses couleurs »

```
>> from matplotlib import cm
>> from matplotlib.colors import ListedColormap

>> img = plt.imread('../img/cameraman.tif')
>> map = cm.hot(range(256))
>> plt.figure(1)
>> plt.imshow(img, cmap=ListedColormap(map))
>> map = cm.gray(range(256))
>> plt.figure(2)
>> plt.imshow(img, cmap=ListedColormap(map))
```



## Comment extraire les informations d'une image ?

- Bilan des caractéristiques de l'image
  - Dimensions spatiales
  - Codage : « vraies couleurs » couleurs indexées
  - Format numérique
  - Intervalles d'intensité
  - Répartition des intensités

```
h, w, c = img.shape
```

```
whos (dans la console)
```

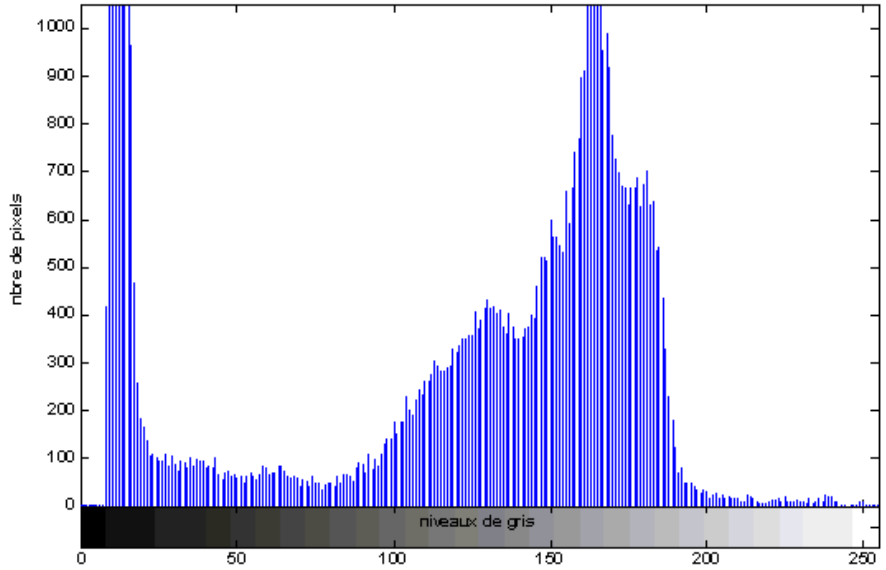
```
[np.min(I(:)) np.max(I(:))]
```

```
np.histogram(I)
```

## Définition

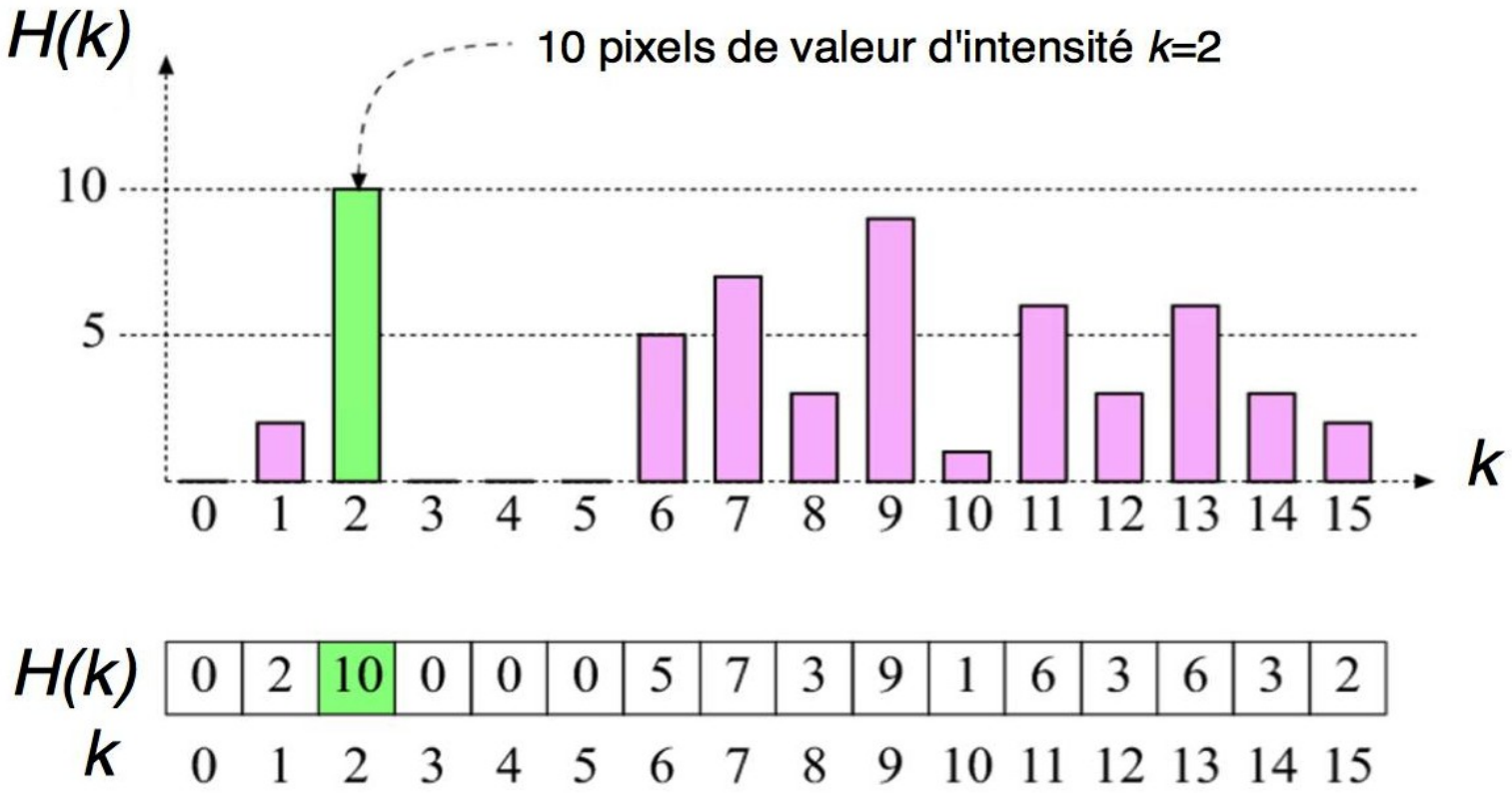
Graphique permettant de représenter la distribution des intensités des pixels d'une image (nombre de pixels pour chaque intensité)

**Informations** : Distribution statistique, bornes de la répartition, ...



## Définition

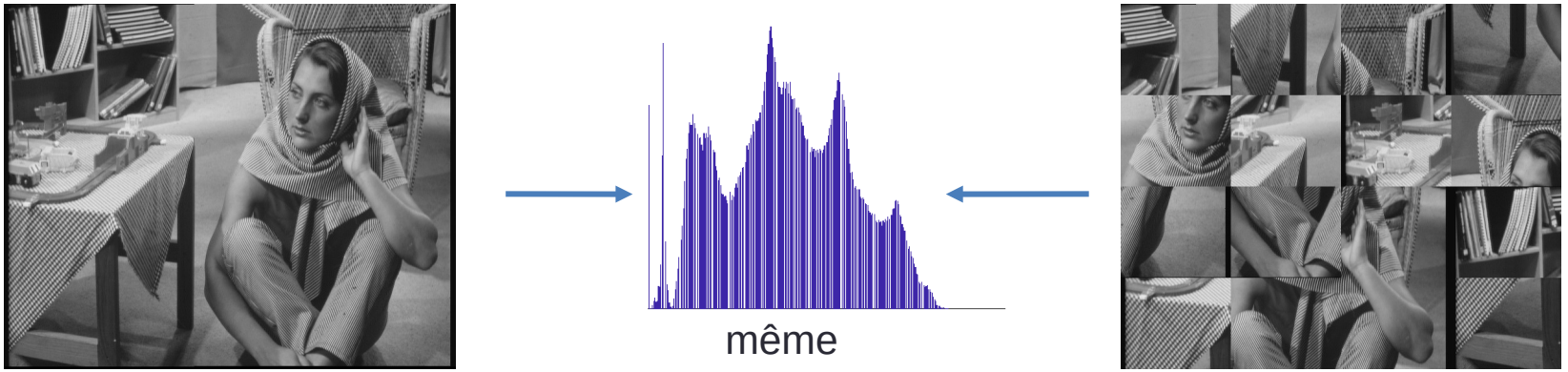
Histogramme $H(k) = \#\{(i,j) \in M \times N : I(i,j) = k\}$



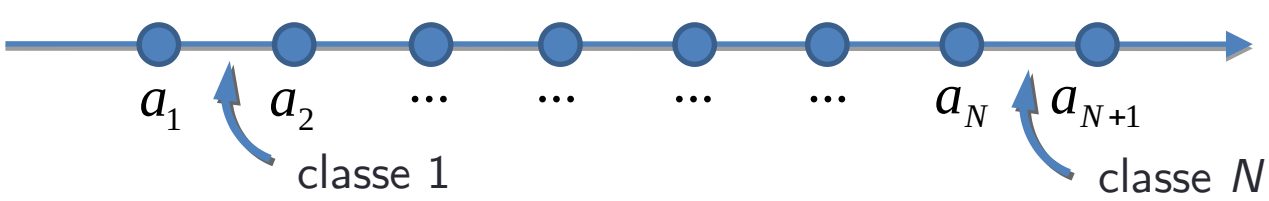
## Précautions

- Choisir un nombre pertinent d'intervalles
- H ne reflète pas l'information spatiale.
- 2 images différentes peuvent avoir le même histogramme

$$H(I) = H(I') \not\rightarrow I = I'$$



## Définitions des intervalles

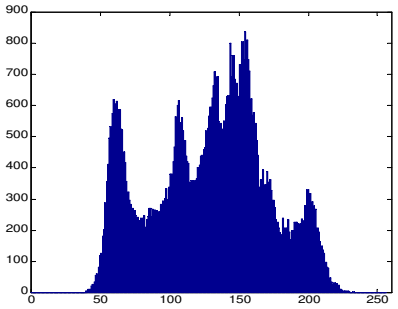



$$a \in \left[a_1, \ldots, a_{N+1}\right[$$

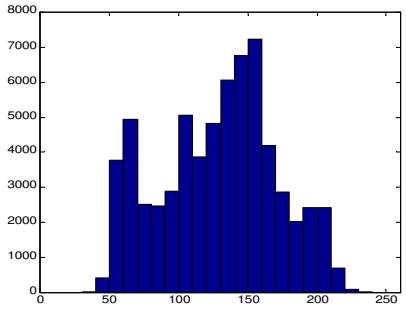
$$H(k) = \begin{cases} \#\{(i,j) \in M \times N : I(i,j) < a_{i+1}\} & i = 1 \\ \#\{(i,j) \in M \times N : a_i \leq I(i,j) < a_{i+1}\} & 1 < i < N \\ \#\{(i,j) \in M \times N : a_i \leq I(i,j)\} & i = N \end{cases}$$

amplitude de plus en plus forte

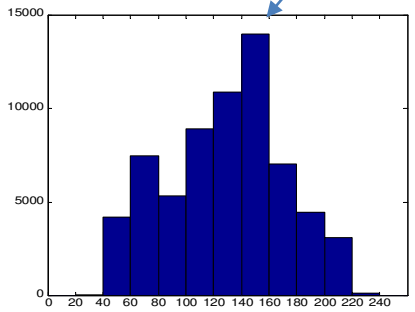
représentativité de plus en plus faible




Histogramme de « 1 en 1 »




Histogramme de « 10 en 10 »




Histogramme de « 20 en 20 »

## Définitions des intervalles

- L'histogramme d'une image nous informe sur son contraste :
  - Image sombre : valeurs proches de 0
  - Image claire : valeurs proches de 255
  - Contraste faible : valeurs tassées
  - Contraste élevé : valeur réparties

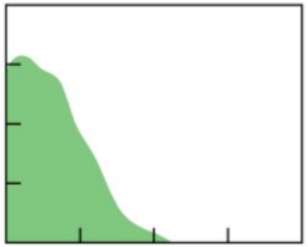


image sombre

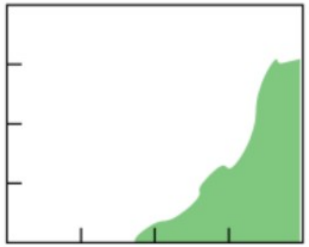


image claire

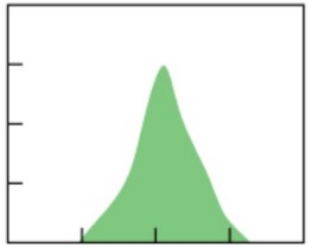


image peu contrastée

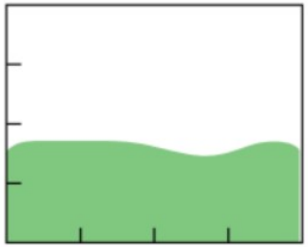
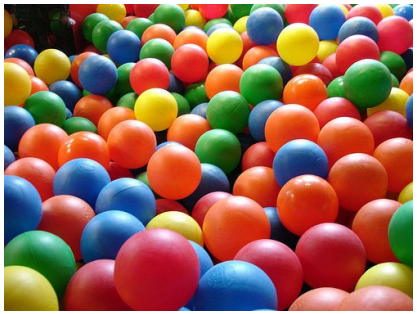
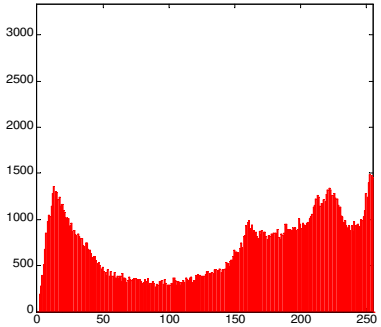


image très contrastée

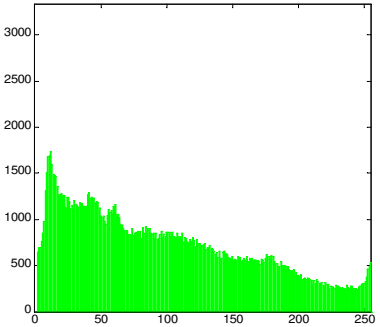
# Histogramme d'une image couleur

- Histogrammes séparés pour chaque composante

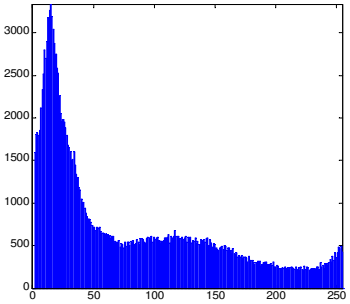





Histogramme de la composante rouge

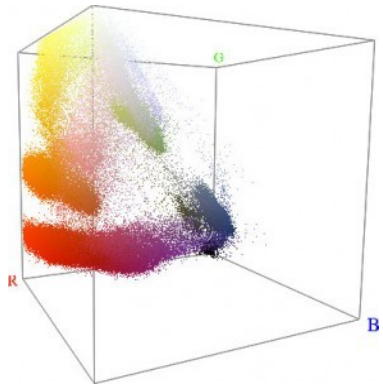



Histogramme de la composante verte



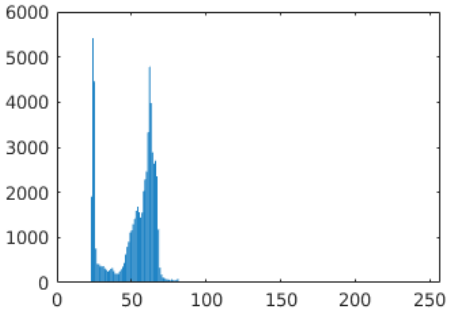

Histogramme de la composante bleue

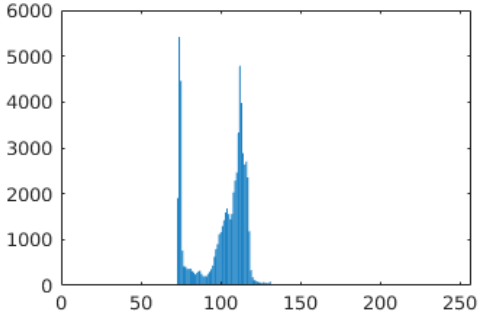
- Visualisation 3D



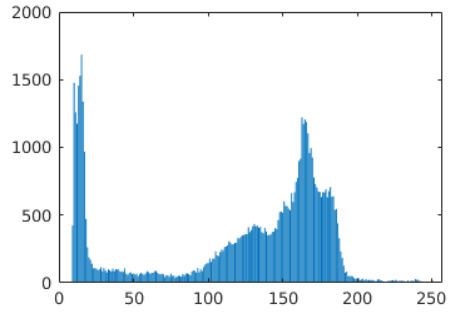
## Transformations linéaires

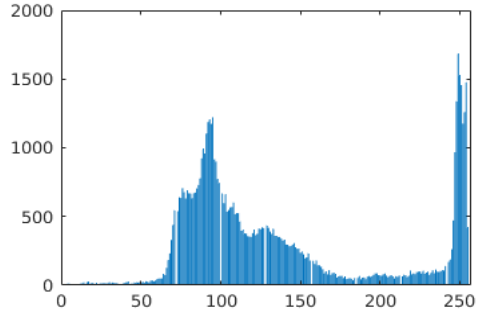
- Transposition / Gain linéaire $\{I_s(m,n)\} = \{I_e(m,n)\} + 50$


- Négatif $T(I_e) = I_{e\max} + I_{e\min} - I_e$

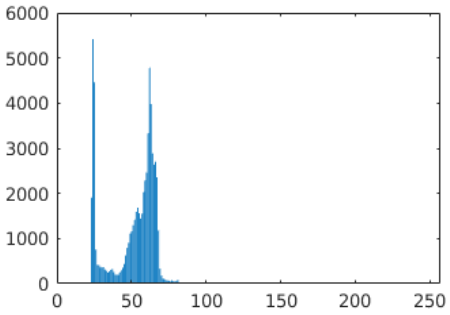



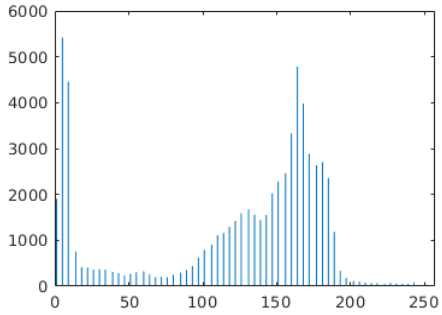



## Normalisation / Étirement d'histogramme

- Modifier l'intervalle de variation $[\min, \max] \rightarrow [\min', \max']$

$$[\min, \max] \rightarrow [0, 255] \qquad I'(i,j) = \frac{255}{\max - \min}(I(i,j) - \min)$$




*(effet produit par `imshow`)*

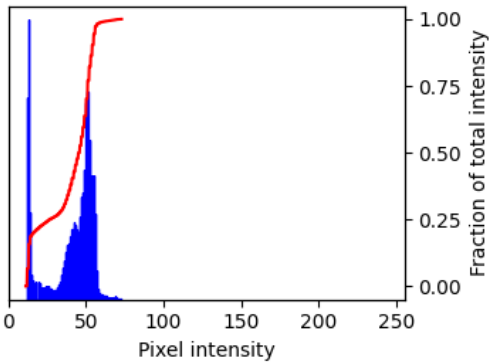
- Quelles sont les limites de cette méthode ?

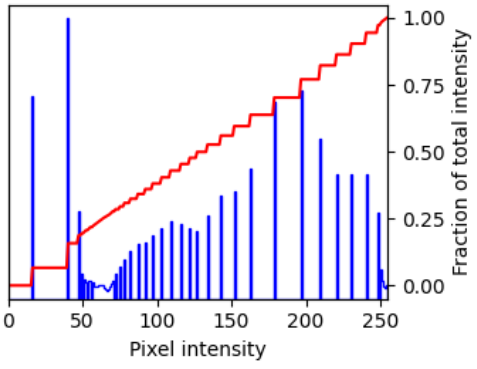
# Équilibrage d'histogramme

- Étalement uniforme des intensités sur l'intervalle de valeurs disponibles (`skimage.exposure.equalize_hist`)
  - Histogramme normalisé : $H_n(k) = \frac{H(k)}{MN}$
  - Histogramme cumulé : $H_n(k) = \sum_{l=0}^{k} H(l)$
  - Histogramme cumulé normalisé : $H_{cn}(k) = \sum_{l=0}^{k} H_n(l)$

$\rightarrow$ Nouvelle intensité $k'$ pour un pixel d'intensité $k$

$$k' = I_{\max} H_{cn}(k) = I_{\max} \sum_{l=0}^{k} H_n(l) = I_{\max} \sum_{l=0}^{k} \frac{H(l)}{MN}$$



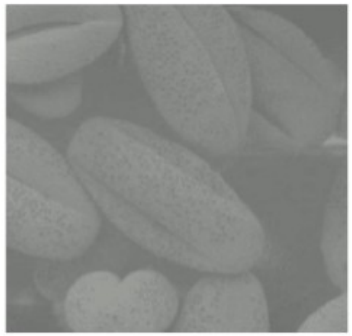
# Équilibrage d'histogramme

Normalisation

Equilibrage

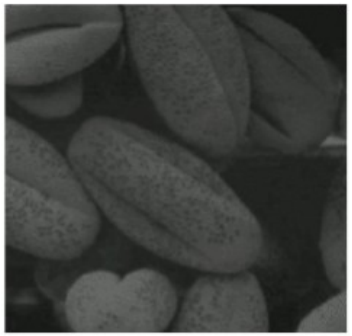
## **Quiz** : Retrouver à qui correspondent ces histogrammes

(1)

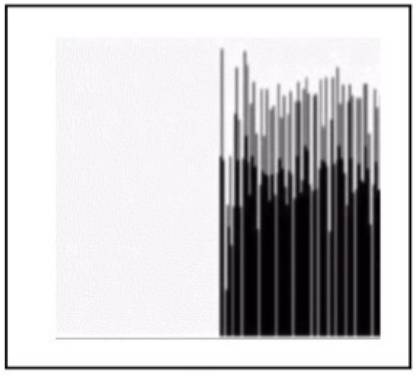


(2)

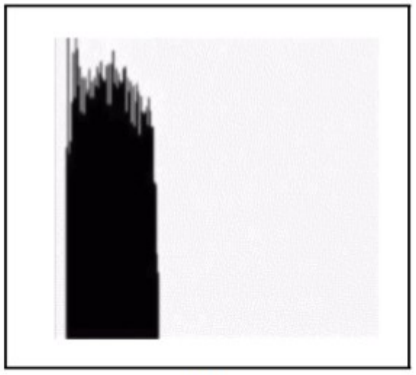
(3)



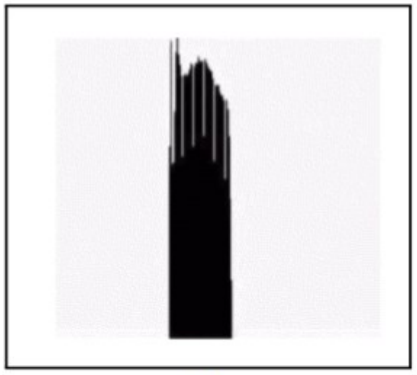
(4)



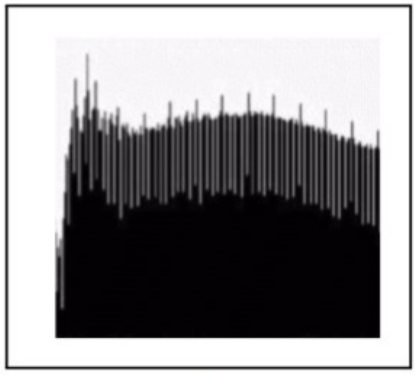
(a)



(b)



(c)



(d)

## Comment extraire les informations d'une image ?

- Bilan des caractéristiques de l'image
  - Dimensions spatiales
  - Codage : « vraies couleurs » couleurs indexées
  - Format numérique
  - Intervalles d'intensité
  - Répartition des intensités

```
h, w, c = img.shape
```

```
Whos (dans la console)
```

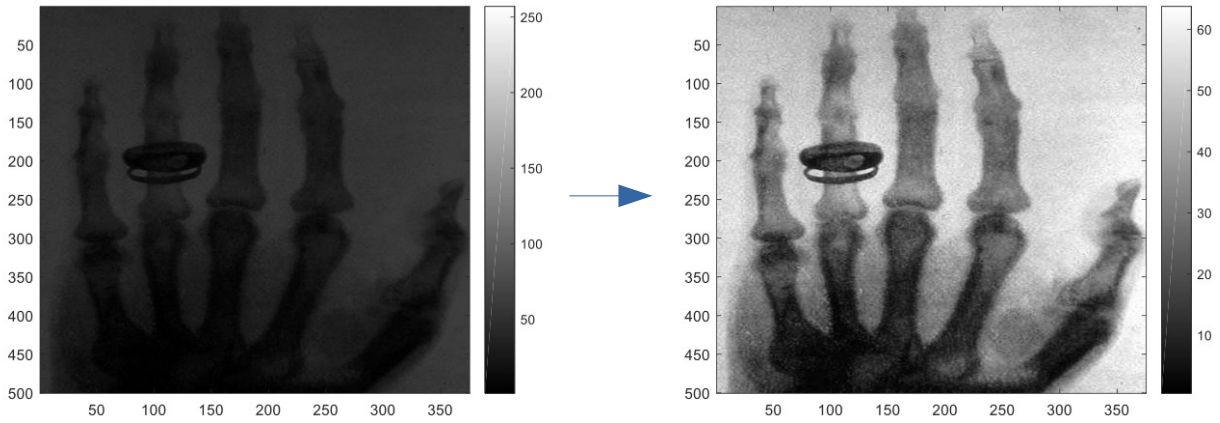
```
[np.min(I(:)) np.max(I(:))]
```

```
np.histogram(I)
```

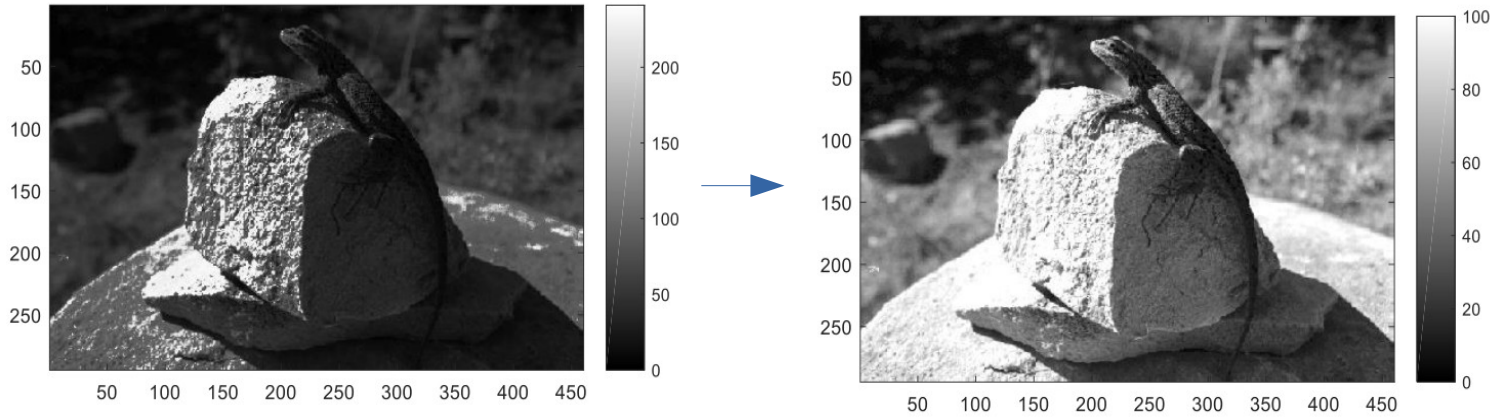
- Identification de la fonction d'affichage
  - Déterminer le contexte
    - Informationnel
    - De fidélité
    - De comparaison
  - Choix : palette, ratio L/H, intervalles d'intensité, etc.

## **Contexte « informationnel »** (couleurs indéxées)

### Étalement automatique des intensités (`imshow(I)`)




### Étalement contrôlé (`imshow(I, vmin=X, vmax=X)`)

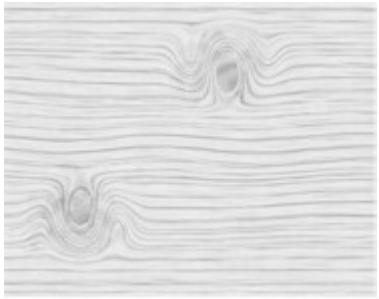


## Contexte de « fidélité »

Respect des intensités initiales
(`imshow(I, vmin=0, vmax=255)`)

*insaturée*

*sombre*

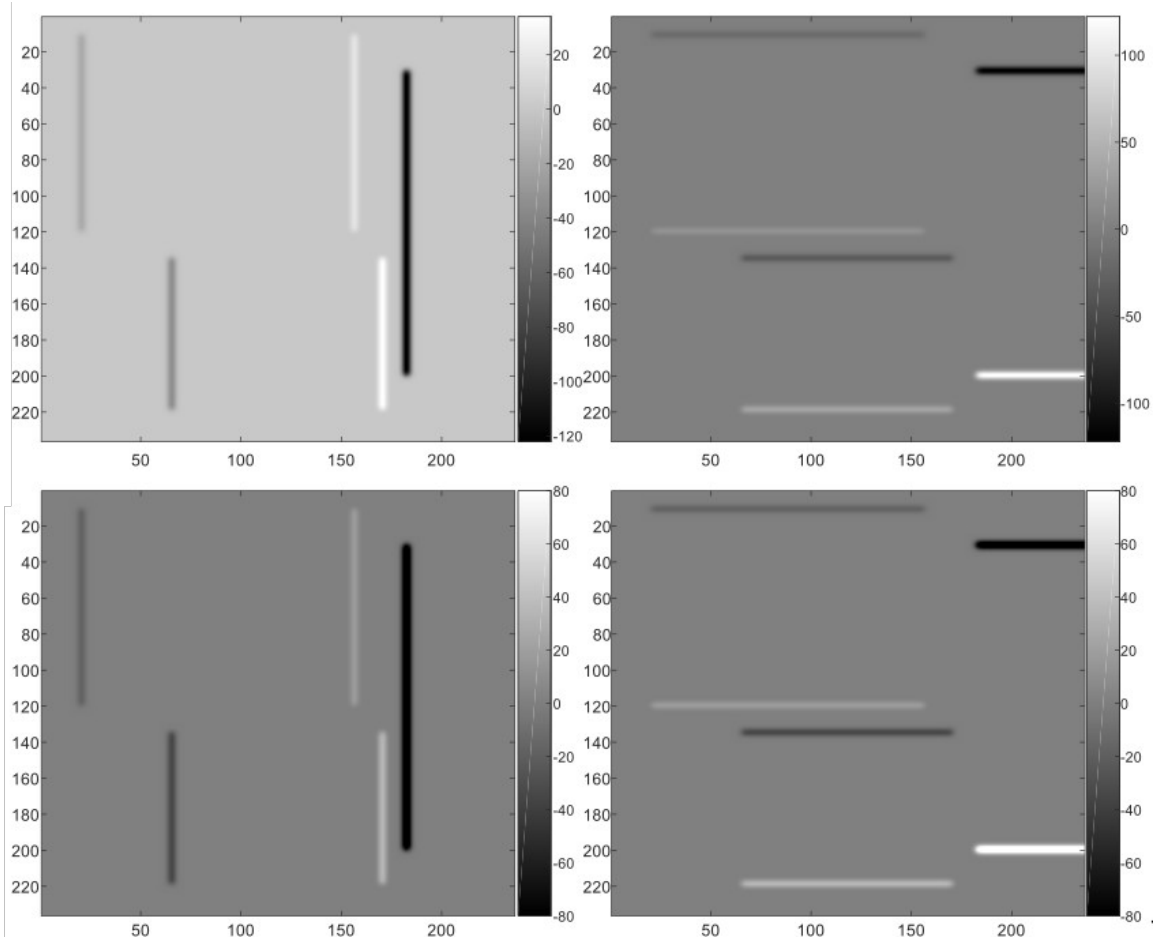


*claire*

*saturée*

## **Contexte de « comparaison »** (couleurs indéxées)

Même étalement contrôlé (`imshow(I, vmin=X, vmax=X)`)

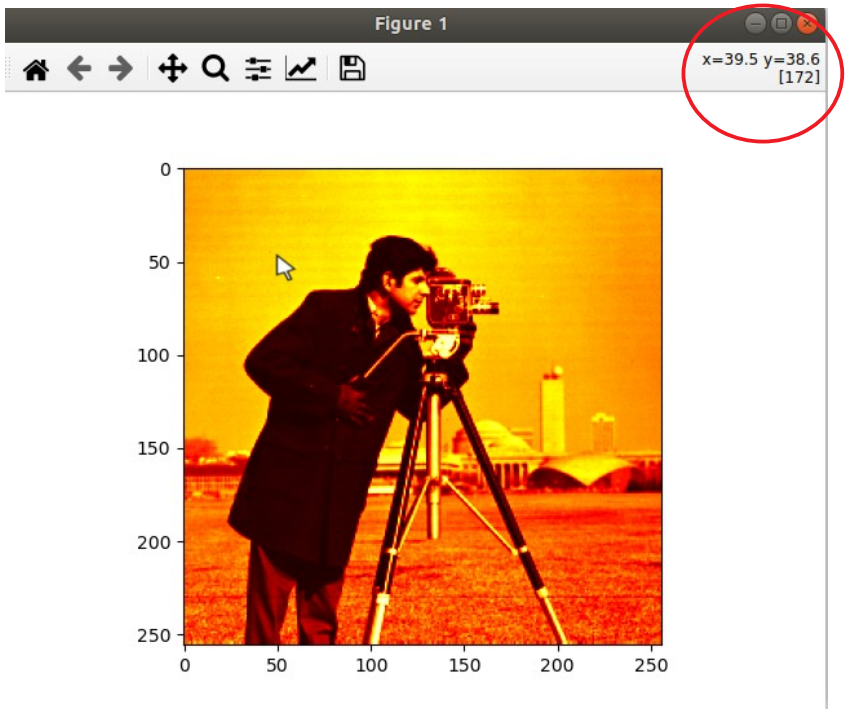


**Auto**

**Contrôlé**

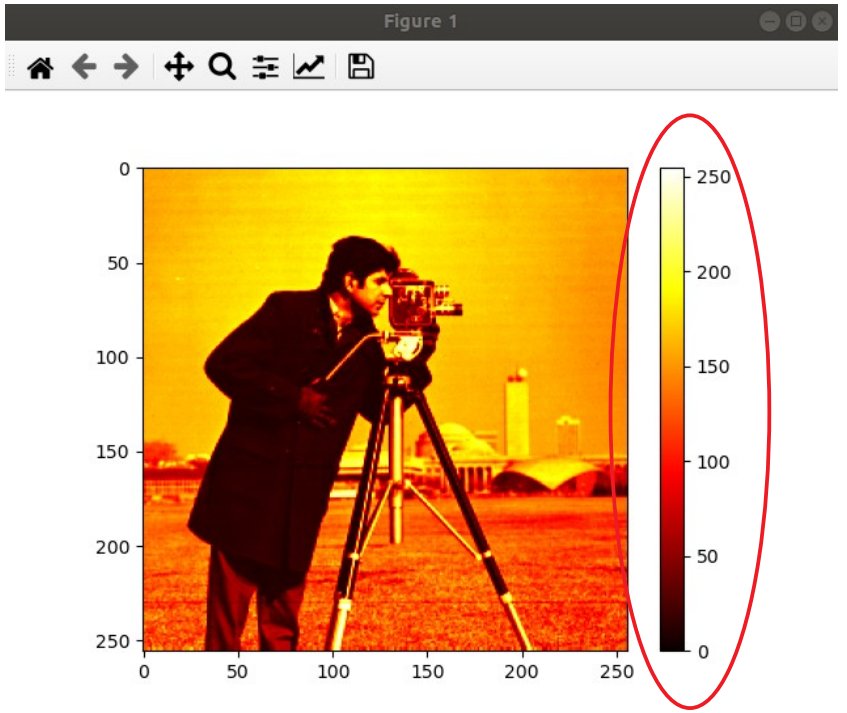
## Data cursor

Pour visualiser rapidement les intensités dans une région




## Colorbar

Pour visualiser la gamme des intensités d'une image à 1 canal (`plt.colorbar()`)



# Mémo Python (1/2)

- Bonnes pratiques :
  - Utiliser un vrai éditeur de code Python (spyder, jupyter-notebook, etc.)
  - Se placer dans un dossier dédié
  - Toujours écrire dans un script (*fichier.py*)
  - Surveiller le workspace pour voir quelles sont et surtout la taille des variables
  - Consulter l'aide des fonctions (`help(function)`)
- Modules :
  - Accéder aux fonctions d'autres modules :

```
import numpy as np #toutes les fonctions,
     possibilité de renommer le module
from signal import convolve
  #import d'une seule fonction
import subfolder.my_module
  #import de ./subfolder/my_module.py
```

- Fonctions :
  - Peuvent être écrites dans le script (avant le code appelant, comme en C)

```
def min_max(T):
    min_ = min(T)
    max_ = max(T)
    return min_, max_
```

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

```
h, w, c = img.shape

#%% Affichage
plt.figure(), plt.imshow(img)

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

  - Par sélection : **F9**

```
h, w, c = img.shape

#Affichage
plt.figure(), plt.imshow(img)
```

- Principales différences avec Matlab :
  - Indices [ ], à partir de 0 : tableau de taille `l`

```
tab[0] #premier élément
tab[l-1] = tab[-1] #dernier élément
```

  - Vecteur d'échantillonnage :

```
range(0,100) #0, 1, ... , 99
```

  - Opération terme à terme par défaut :

```
a = np.array([1,2,3,4])
a*a #array([1,4,9,16])
```

# Mémo Python (2/2)

- Commandes de bases :

```python
#Modules utiles
import numpy as np #tableau, opérateurs maths
import matplotlib.pyplot as plt #image, affichage
import skimage  #par ex. espace couleur ycbcr
from scipy import signal #par ex. convolution

#Manipulation d'image
img = plt.imread('path/img.png') #chargement
h, w, c = img.shape  #image couleur
print(h)

img_vect = img.ravel()  #Vectorisation
G = img[:,:,1] #accès dimension 2e canal = vert

#Mise à zéro
img = np.zeros((h,w,c)) #ones() existe aussi
img = np.copy(img*0)

#Sous-échantillonnage
img_se = img[1:h:4, 1:w:4] #ou img[::4,::4]


#Création d'un vecteur/d'une matrice
mat = [[1,1],[2,2],[3,3]]  #liste de taille 3x2
mat = np.array(mat)  #np.array

#Produits vecteurs/matriciels
vect = np.array(np.matrix(range(1,11,2))).T
            #range(début,fin,pas)  #T = transposée
vect_5_1 = vect*vect #terme à terme
vect_5_5 = np.dot(vect, vect.T) #prod. matriciel
vect_1_1 = np.dot(vect.T, vect) #prod. matriciel

#Somme sur matrice nD
sum_G = np.sum(G**2) #somme de tous les termes^2
sum_G = np.mean(img, axis=2) #conversion niv. gris
```

```python
#Seuillage sur une matrice
mask = G > 100 #mask = carte binaire (hxw)
#Équivalent à faire :
mask = np.zeros((h,w))
for i in range(0,h):
   for j in range(0,w):
      if (G[i,j]>100):
          mask[i,j] = 1

G[mask==0] = 0 #Mise à zéro des pixels de G où mask=0
G = G*mask #Équivalent à multiplication terme à terme

#changement de type
G = G.astype('uint8') #ou G = np.uint8(G)

#changement d'espace couleur
img_ycbcr = skimage.color.rgb2ycbcr(img)

#convolution image couleur (même filtrage sur R,G,B)
filter_ = np.ones([7,7,1])/49
img_f = signal.convolve(img, filter_, mode='same')

#Affichage
img_L = np.mean(img, axis=2)
plt.figure()
plt.subplot(121) #Affichage multiples 1x2
plt.plt(img_L[0,:])
plt.title('Profil de la première ligne de L')
plt.subplot(122)
plt.plot(img[0,:,0],'ro') #superposition par défaut
plt.plot(img[0,:,1],'g+')
plt.plot(img[0,:,2], color=[0,0,1])
plt.title('Profil RGB de la première ligne')
plt.xlabel('x'), plt.ylabel('Intensité')
plt.show()
```

**Autres fonctions utiles :** `np.squeeze, np.tile, plt.ginput, …`

**Liens vers les docs :** Tuto général Python  Tuto scikit-image
Tuto numpy, matplotlib, scipy

- Trouver une façon satisfaisante d'afficher les images :
  *challenge#A-C.npy*

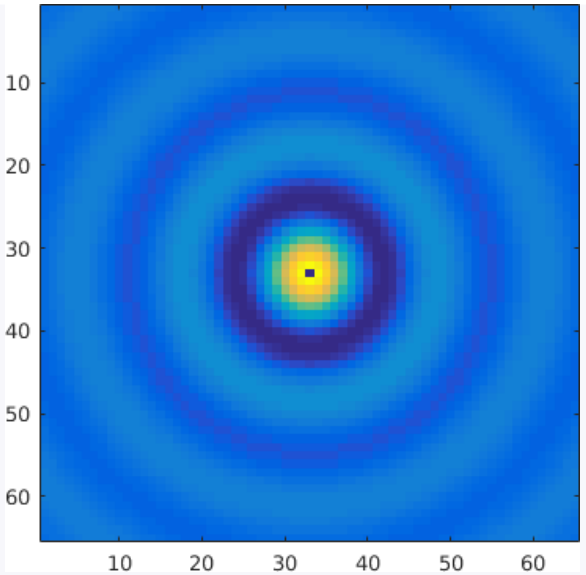
| Nom | Donnée | Problématique |
|---|---|---|
| Mandrill | challengeA.npy | Format numérique |
| Radio | challengeB.npy | Intervalle d'intensité |
| World map | challengeC.npy | Palette discrète |

```
A = np.load('challengeA.npy')
```

*Attention, ici on charge les données avec np.load depuis un fichier archive .npy, pour récupérer des données matricielles (avec probablement des problèmes à traiter...).*
*Le reste du temps, on utilisera plt.imread() pour lire des images aux formats .jpg, .png, etc.*

## Synthèse analytique en couleurs indéxées

```
#solution longue (7 lignes)
x = range(-34,35)
y = range(-32,33)
img1 = np.zeros((len(y), len(x)))
for i in range(0, len(y)):
   for j in range(0, len(x)):
       r = np.sqrt(x[j]**2+y[i]**2)
       img1[i,j] = 1000*np.sin(r/2)/r
plt.figure(1)
plt.imshow(img1)
```

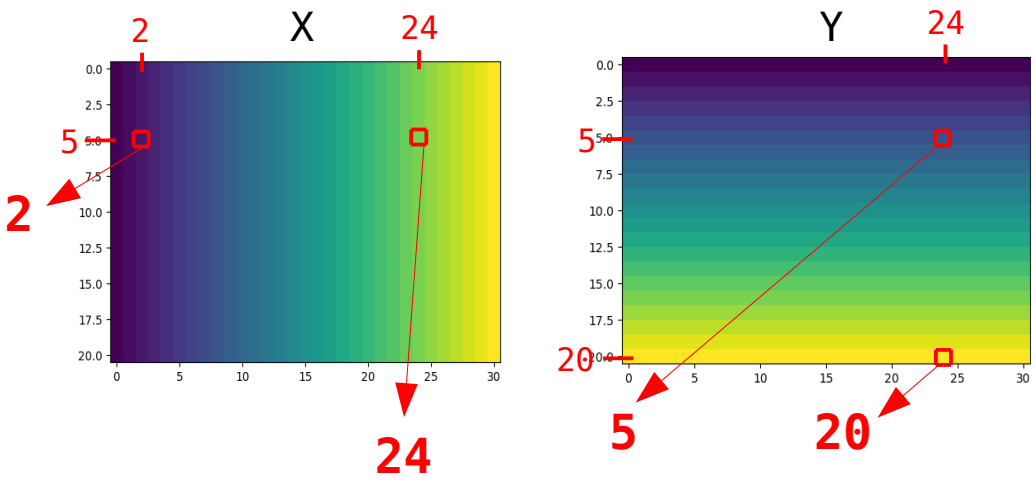



```
%solution courte (3 lignes)
X, Y = np.meshgrid(range(-34, 35), range(-32,33))
R = (X**2 + Y**2)**0.5;
img2 = 1000*np.sin(R/2)/R;
plt.figure(2)
plt.imshow(img2)
```

# Fonction np.meshgrid

**Objectif** : Se passer des boucles de parcours Hauteur/Largeur

**Exemple** : `X, Y = np.meshgrid(range(0,31), range(0,21))`




Crée deux matrices de taille 21x31 qui stockent en valeur pour chaque pixel :

- X : le numéro de colonne
- Y : le numéro de ligne

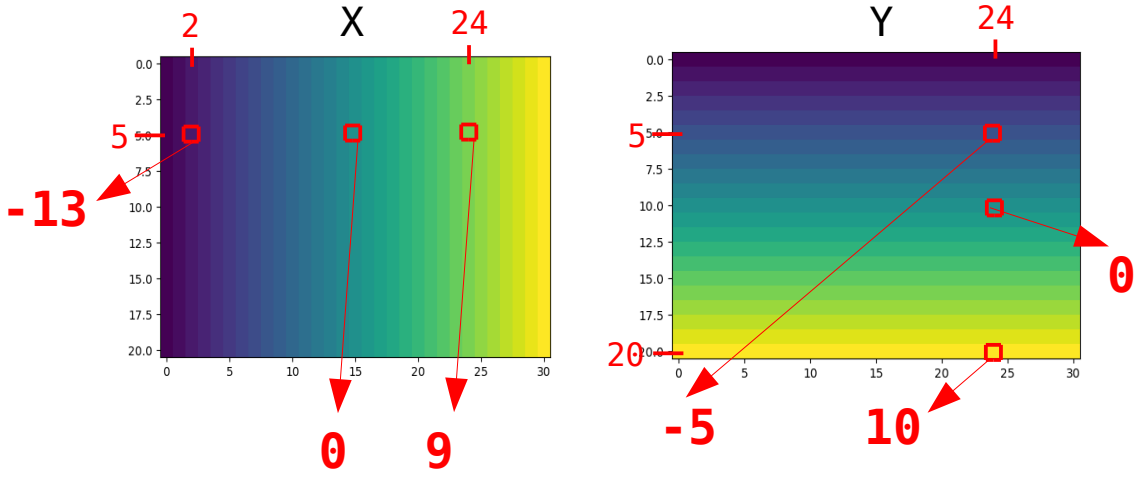
```
x_v = range(0, 31)
y_v = range(0, 21)
X, Y = np.meshgrid(x_v, y_v);
I = X**2 + Y**2
```

```
x_v = range(0, 31)
y_v = range(0, 21)
I = np.zeros((len(y_v), len(x_v))
for i in range(0, len(y_v)):
   for j in range(0, len(x_v)):
      I[i,j] = y_v[i]**2 + x_v[j]**2
```

# Fonction np.meshgrid

**Objectif** : Se passer des boucles de parcours Hauteur/Largeur

**Exemple** : `X, Y = np.meshgrid(range(-15,16), range(-10,11))`



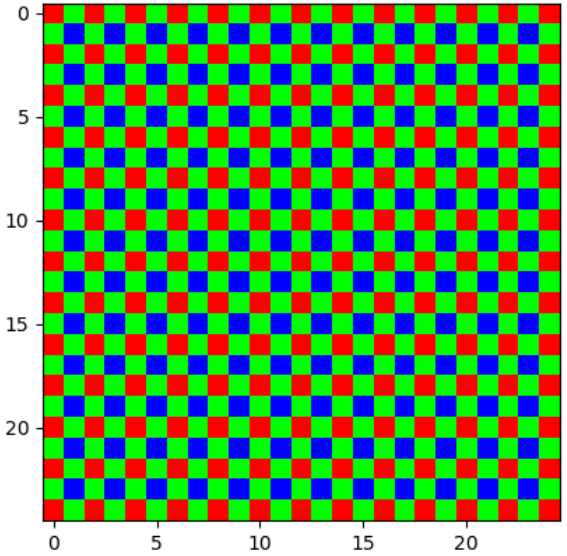

Crée deux matrices de taille 21x31 qui stockent en valeur pour chaque pixel le numéro de colonne (X) et le numéro de ligne (Y)

```
x_v = range(-15, 16)
y_v = range(-10, 11)
X, Y = np.meshgrid(x_v, y_v);
I = X**2 + Y**2
```

```
x_v = range(-15, 16)
y_v = range(-10, 11)
I = np.zeros((len(y_v), len(x_v))
for i in range(0, len(y_v)):
   for j in range(0, len(x_v)):
      I[i,j] = y_v[i]**2 + x_v[j]**2
```

## Synthèse en « vraies couleurs »

```
Nx = 25
Ny = 25
x = range(0, Nx)
y = range(0, Ny)

X, Y = np.meshgrid(x, y)

R = (1-np.mod(X,2))*(1-np.mod(Y,2))
G = np.mod(X+Y,2)
B = (1-np.mod(X+1,2))*(1-np.mod(Y+1,2))

I  = np.stack((R,G,B), axis=2).astype('double')

plt.figure(1)
plt.imshow(I)
```

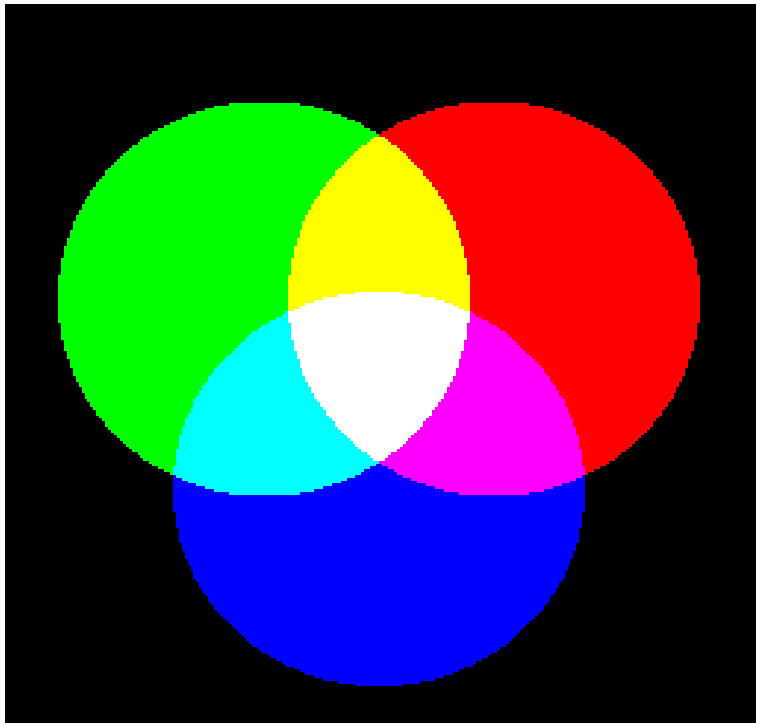


- Écrire une fonction *disk* qui génère cette image :

  En « vraies couleurs »

  En couleurs indexées

  La taille de l'image, la largeur des cercles et leur espacement seront des paramètres.



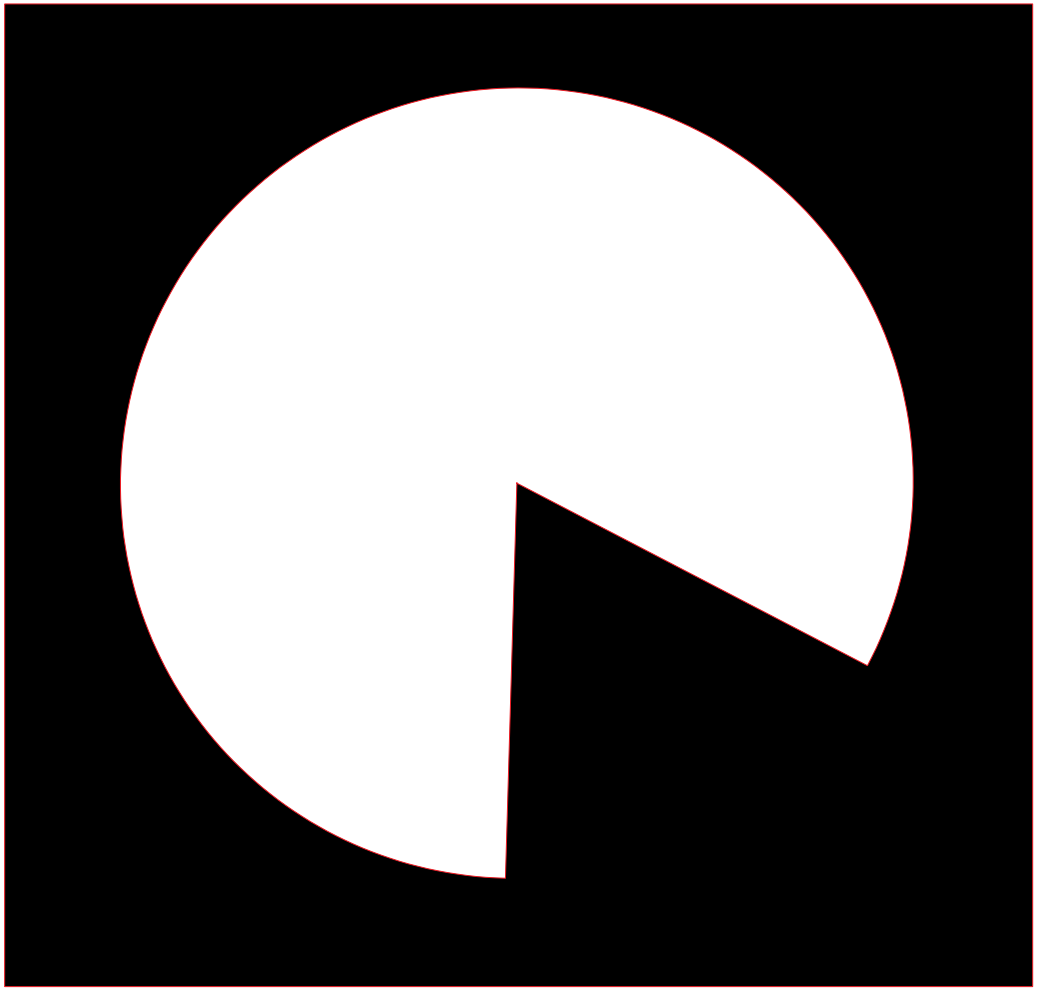
# Synthèse additive dynamique

- Création d'une vidéo avec openCV :

```
size = 255
radius = 70
dist = 45

R, G, B = p1_disks(size, radius, dist)
h, w = R.shape

#Créer l'objet vidéo avec VideoWriter
video = cv2.VideoWriter('video.avi', cv2.VideoWriter_fourcc('M','J','P','G'), 5, (w, h))

for n in range(0,100):

    q = np.mod(n,3)

    if (q == 0):   #C1=Rouge-C2=Vert-C3=Bleu
        img = np.stack((R,G,B), axis = 2).astype('uint8')
    elif (q == 1):
        img = np.stack((B,R,G), axis = 2).astype('uint8')
    else:
        img = np.stack((B,G,R), axis = 2).astype('uint8')

    #Ajout du frame dans la vidéo
    video.write(img)

video.release()
```

# Bonus : Timer

- Créer un timer animé sous forme de camembert progressif

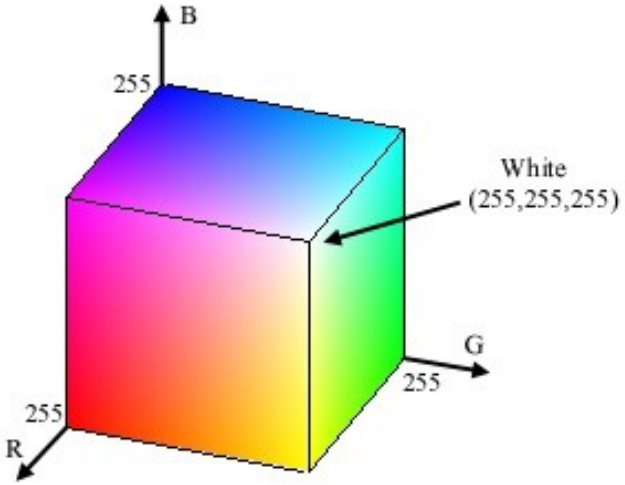


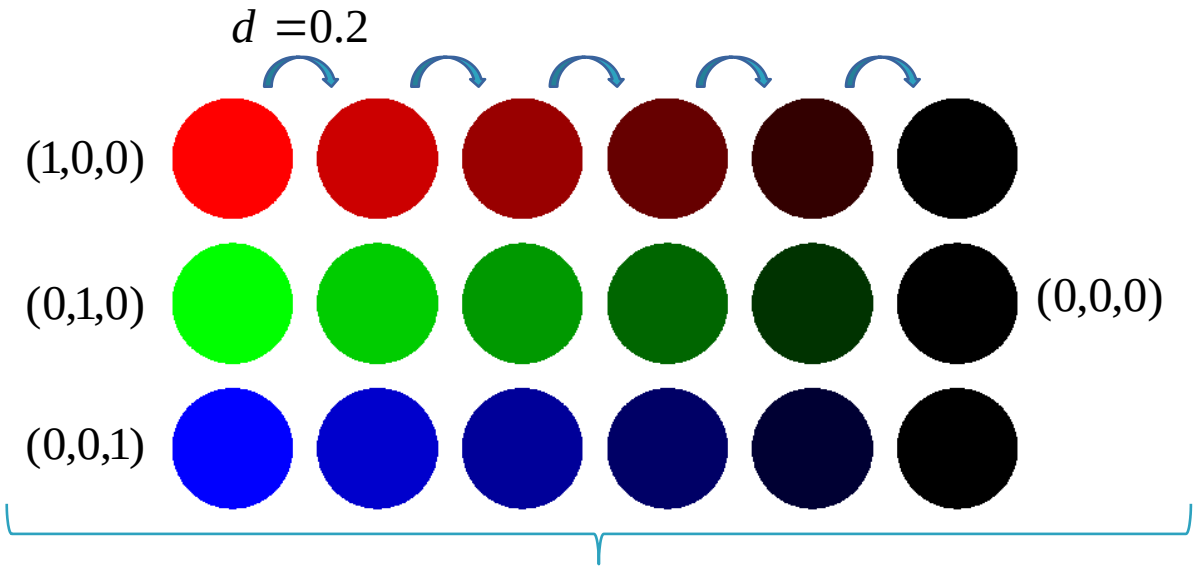
# Plan

- **Image numérique**
  - Format/affichage
  - Visualisation
  - Synthèse
- **Espaces couleurs**
  - Espace YcbCr
    Applications : compression, esquisse, incrustation (TP)
- **Traitements**
  - Filtrage linéaire & non linéaire
    Applications : anonymisation, débruitage
  - Opérateurs de morphologie mathématique
  - Espace fréquentiel, transformée de Fourier
    Applications : recouvrement fréquentiel
  - Détection de contours
    Applications : réhaussement de contraste

## Espace RGB

- Information de couleur et d'intensité mélangée dans les trois canaux
- Luminance globale donnée par **L = (R+G+B)/3**
- Perception différente selon les trois canaux



## Espace RGB

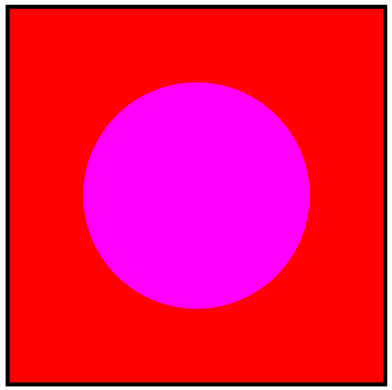
- Sensibilité aux contrastes différente sur les trois canaux


Niveaux de gris

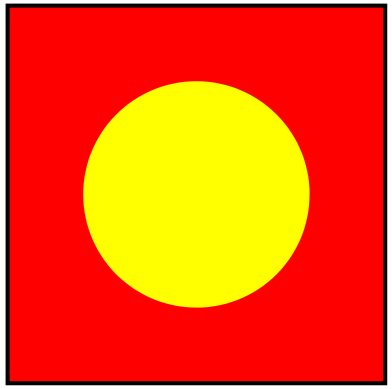
Palettes de primaires pures
(intensités identiques et autres composantes éteintes)

## Canal Y de luminance « perceptuelle »

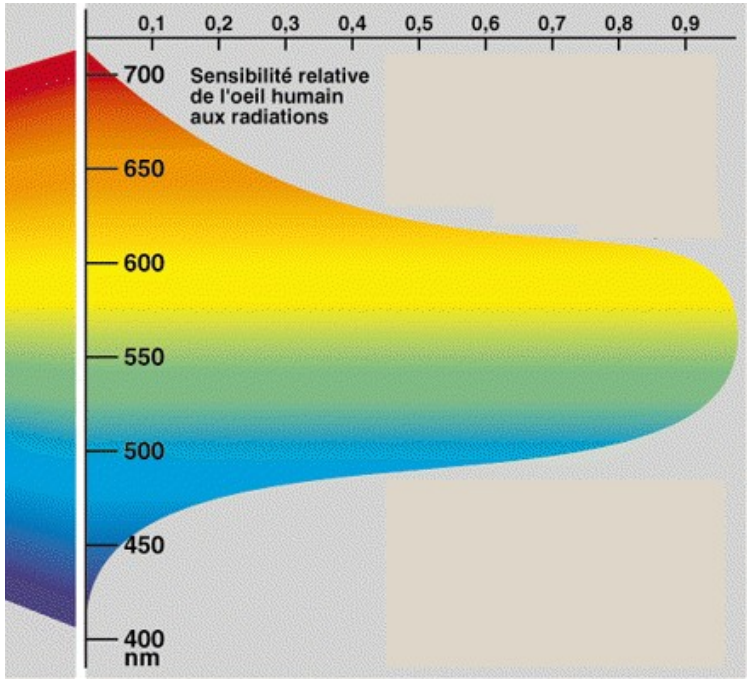
- Tient compte de la sensibilité aux couleurs de l'oeil humain



Rouge/Bleu



Rouge/Vert

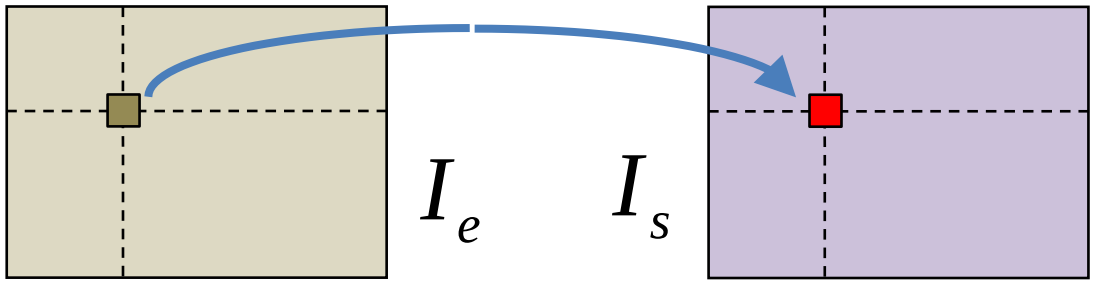



$$Y = 0.299\,R + 0.587\,V + 0.114\,B$$

pondération plus faible

## De nombreux espaces couleurs

- RGB : acquisition/restitution écran
- HSV/HSL : espace intuitif, vision humaine
- YUV/YCbCr : transmission et codage
- La*b*/Lu*v* : espace uniforme, distance entre les couleurs
- CMY : impression
- XYZ : modélisation des couleurs
- espaces non linéaires
- etc



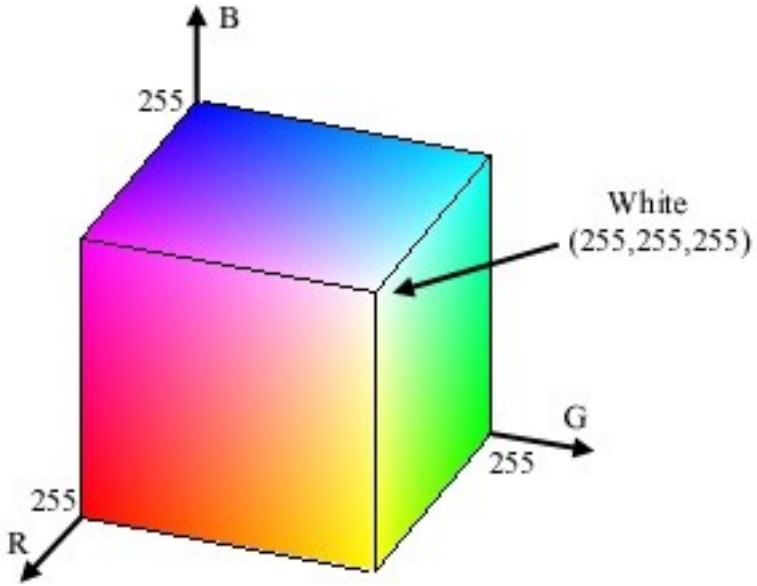
## Espace RGB

- Par défaut
  - Canaux très corrélés
  - Non perceptuel

R

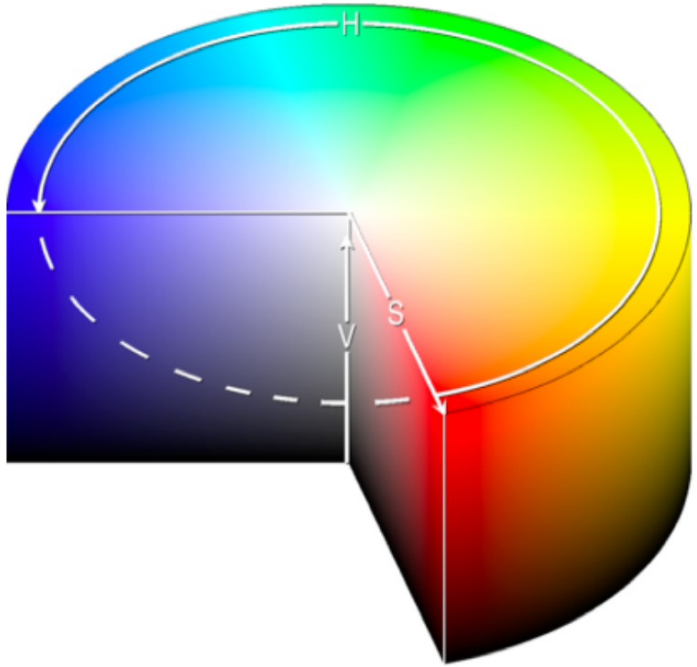
G

B

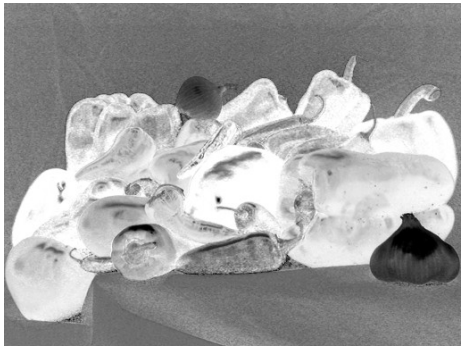


## Espace HSV

- Teinte – Saturation – Valeur
  - (Hue – Saturation – Value)
  - Utile dans les applications graphiques




H

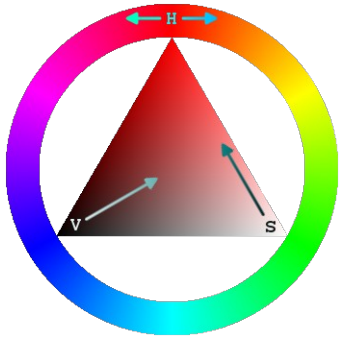


S

V

## Espace HSV

- Teinte – Saturation – Valeur
  - (Hue – Saturation – Value)
  - Utile dans les applications graphiques

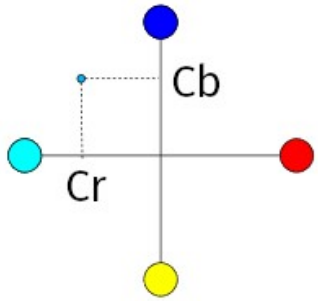



## Espace YCbCr

- Un canal de luminance **Y** et deux canaux de chrominance **Cb Cr**
- Utilisé par ex. pour la compression et la transmission hertzienne
- Les composantes sont obtenues par les formules :

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = 0.564(B - Y) + 128 \\ Cr = 0.713(R - Y) + 128 \end{cases}$$

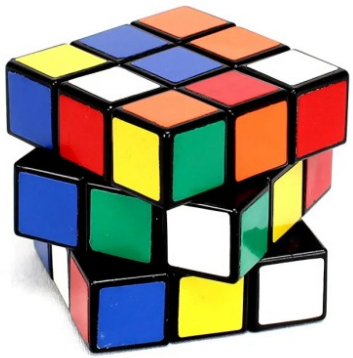
- Les canaux **Cb** et **Cr** correspondent respectivement aux contrastes Bleu/Jaune et Rouge/Cyan.

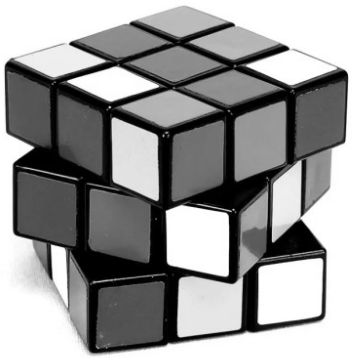


# Espace YCbCr

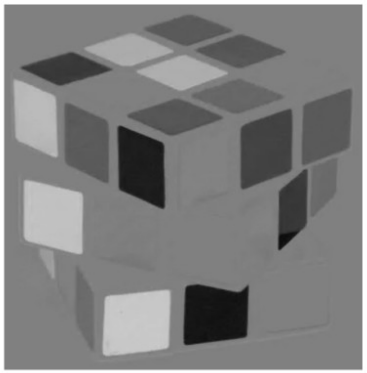
```
I = plt.imread('pool.tif')
I = I.astype('double')
R = I[:,:,0]
G = I[:,:,1]
B = I[:,:,2]

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

YCbCr = skimage.color.rgb2ycbcr(I)
Y = YcbCr[:,:,0]
Cb = YcbCr[:,:,1]
Cr = YcbCr[:,:,2]
```

```
L  = (R+G+B)/3;

plt.figure(1)
plt.imshow(I.astype('uint8'))
plt.title('I')

plt.figure(2)
plt.imshow(L.astype('uint8'))
plt.title('L')

plt.figure(3)
plt.imshow(Y.astype('uint8'))
plt.title('Y')
```

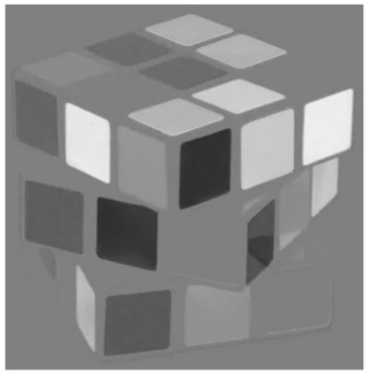




Y

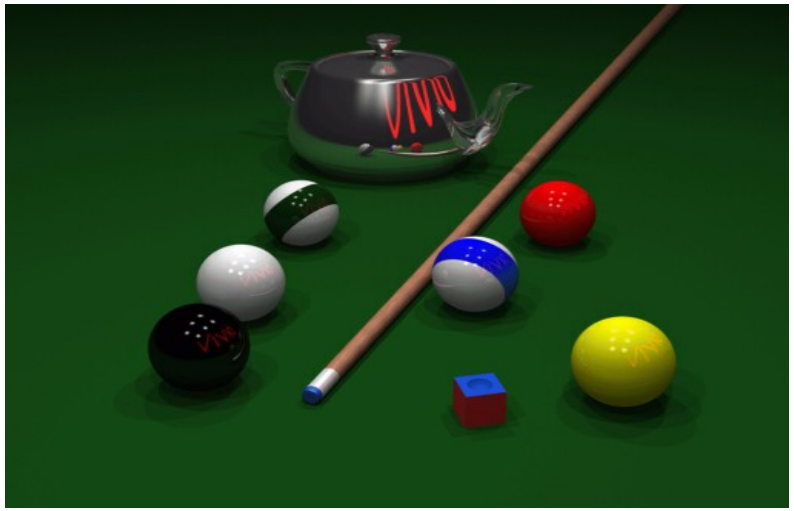


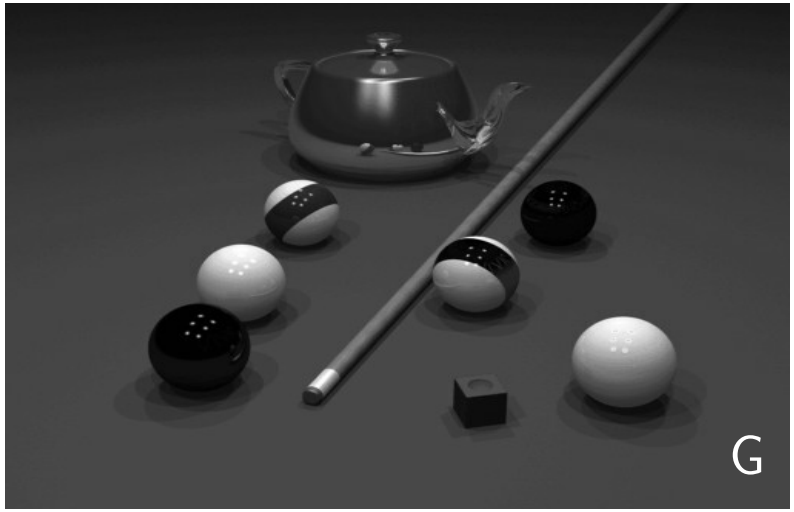
Cb



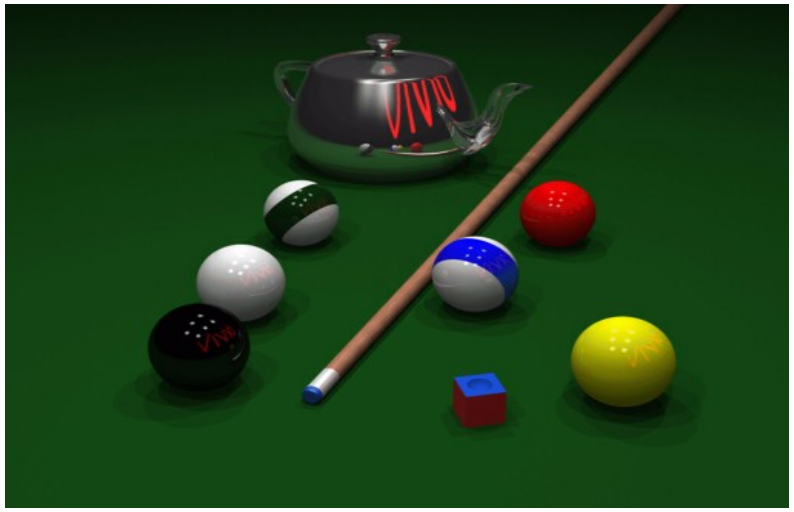
Cr

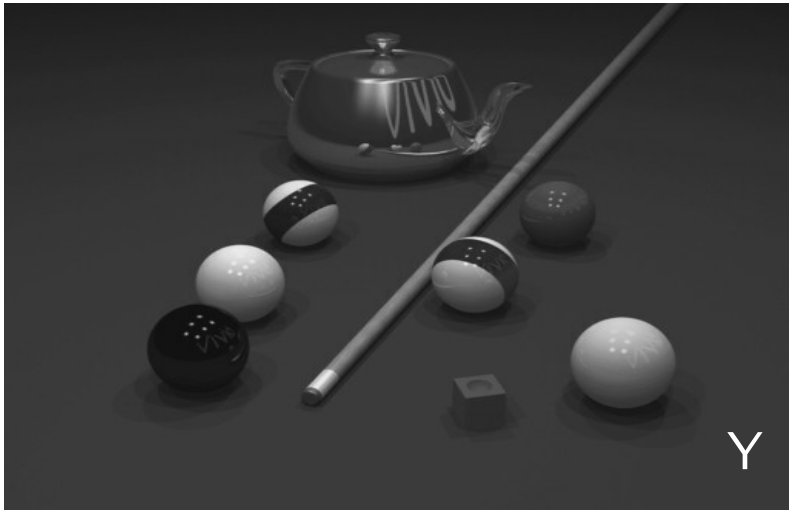
## Espace RGB vs YCbCr

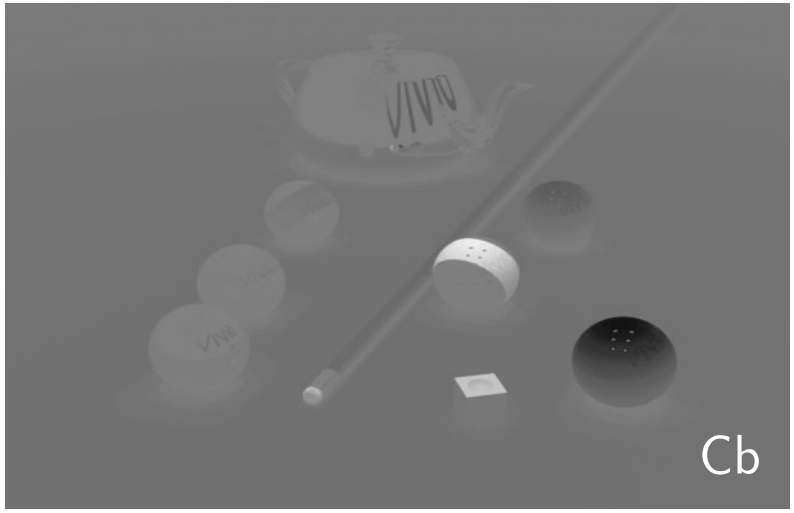


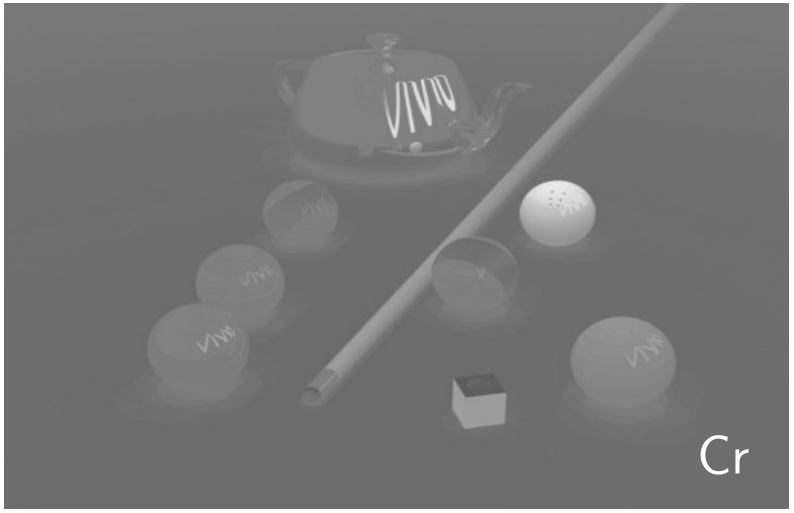




# Espace RGB vs YCbCr



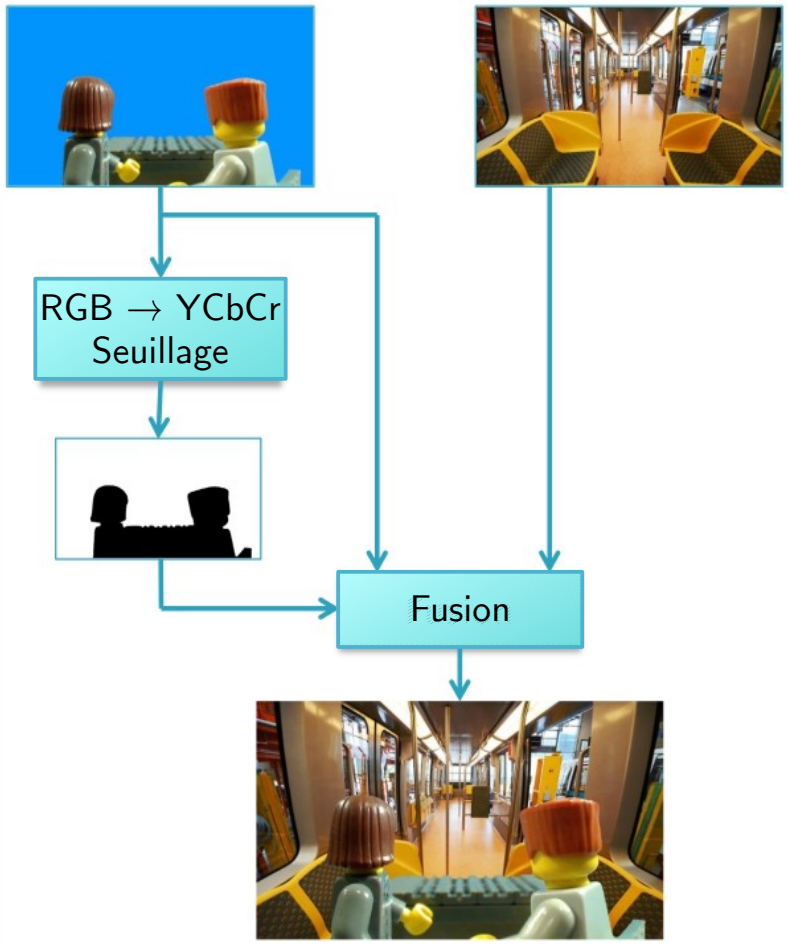


## Exemple d'application : incrustation (*chroma-keying*)

- Meilleure séparation des objets grâce aux canaux de chrominance




→ Implémentation en TP

# Compression dans l'espace YCbCr

- Convertir l'image *pool.tif* (hxwx3) en YCbCr (`skimage.color.rgb2ycbcr`)
- Compresser en taille uniquement les canaux de chrominance CbCr (`skimage.transform.resize`) par un facteur $r<1$ qu'on fera varier
- Reconstruire l'image en ramenant Cb et Cr à leur taille initiale
- Repasser en RGB (`skimage.color.ycbcr2rgb`) et comparer avec l'image initiale
- Quantifier le gain en taille mémoire

Image initiale

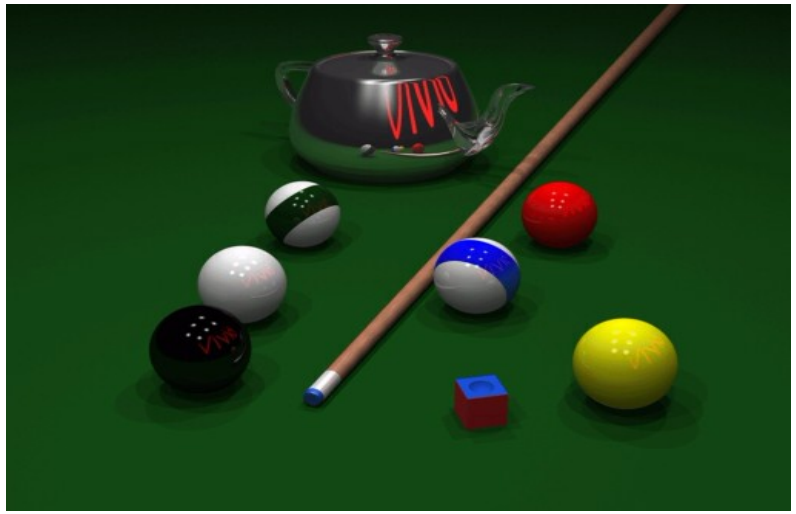


Image compressée $r = 0.5$

## Effet Pencil Sketch

- Calculer une carte de contours C de l'image *home.jpg* (`skimage.feature.canny`)
- 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 :

## Bonus : Illusion d'adaptation chromatique

- Convertir l'image *campagne.jpg* dans l'espace YCbCr et en « niveaux de gris »
- Inverser les canaux de chrominance (Cb=255-Cb, Cr=255-Cr)
- Revenir dans l'espace RGB
- Dessiner un petit cercle noir au centre de l'image RGB modifiée et de l'image originale en « niveaux de gris » (`np.meshgrid`)
- Afficher l'image RGB transformée, faire une pause, puis afficher sur la même figure, l'image « niveaux de gris » (`plt.figure/plt.imshow/plt.pause(12)`)
- Lorsque l'image RGB transformée est affichée, fixer le cercle noir. Que voyez-vous lorsque l'image en « niveaux de gris » s'affiche ?

Image RGB

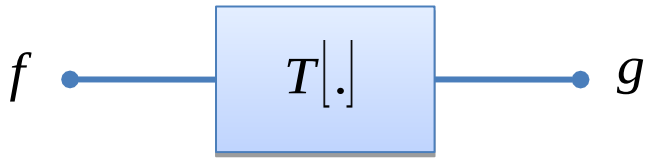
Image RGB
avec chrominance inversée

Image en « niveaux de gris »

# Plan

- **Image numérique**
  - Format/affichage
  - Visualisation
  - Synthèse
- **Espaces couleurs**
  - Espace YcbCr
    Applications : compression, esquisse, incrustation (TP)
- **Traitements**
  - Filtrage linéaire & non linéaire
    Applications : anonymisation, débruitage
  - Opérateurs de morphologie mathématique
  - Espace fréquentiel, transformée de Fourier
    Applications : recouvrement fréquentiel
  - Détection de contours
    Applications : réhaussement de contraste

## Système linéaire





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

$$g(t) = (f * h)(t) = \int_{-\infty}^{+\infty} f(t-\tau)\,h(\tau)\,d\tau$$

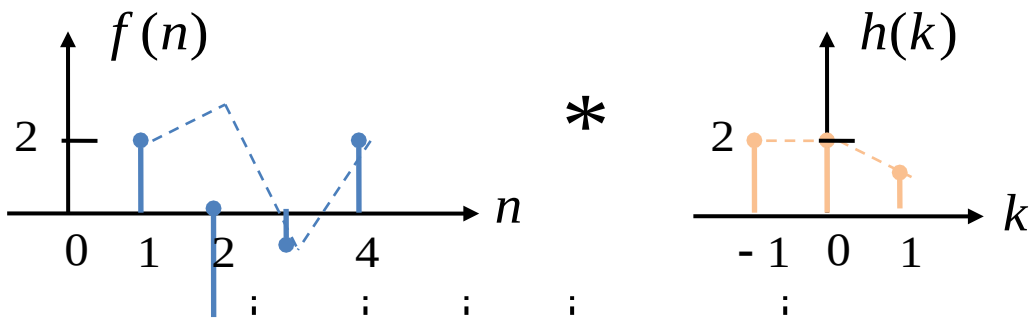
## Filtre linéaire RIF

Application d'un filtre linéaire discret à réponse impulsionnelle finie

défini sur $2K+1$ échantillons

$$g(n) = \sum_{k=-K}^{K} f(n-k)\,h(k) \quad \longleftrightarrow \quad I_s = I_e * H$$

## Convolution discrète 1D



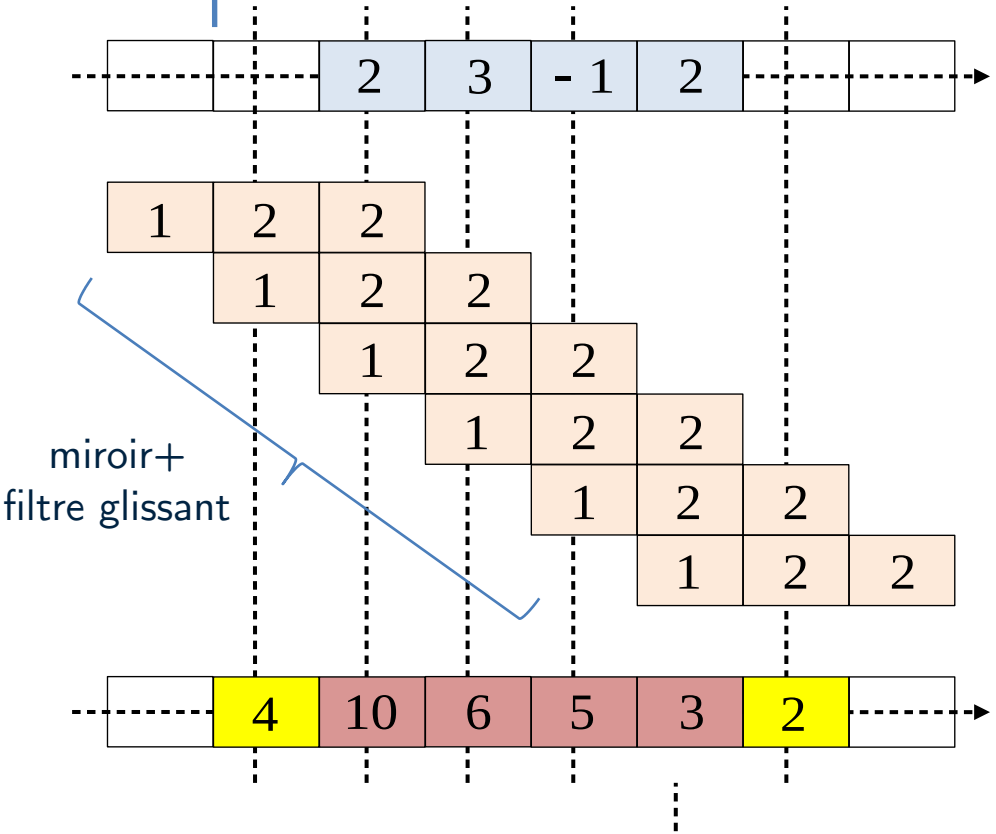

*

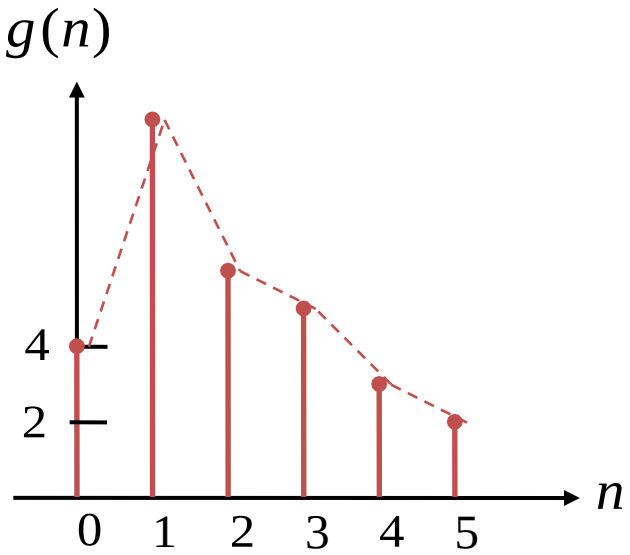



$$g(n) = \sum_{k=-K}^{K} f(n-k)h(k)$$

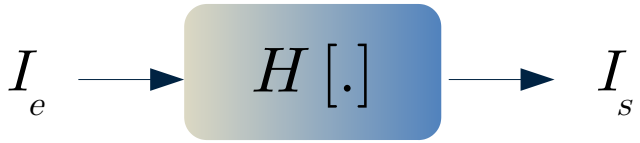
$$= \sum_{k=-K}^{K} f(n+k)h(-k)$$

miroir



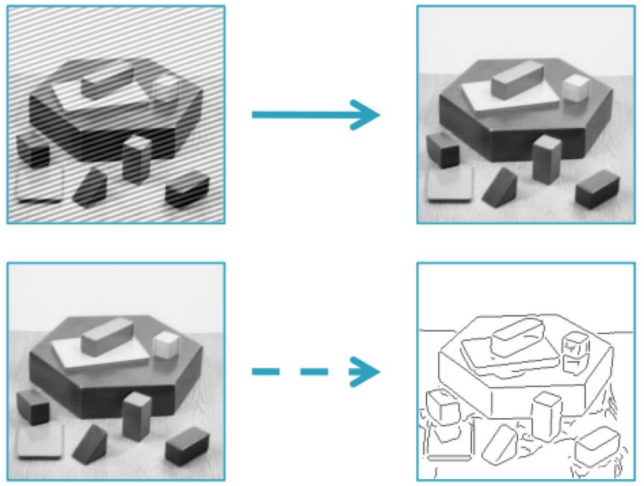



## Système linéaire




Entrée et sortie reliées par un produit de convolution

$$I_s = I_e * H$$




## Filtre linéaire RIF

défini sur (2K+1)x(2L+1) échantillons

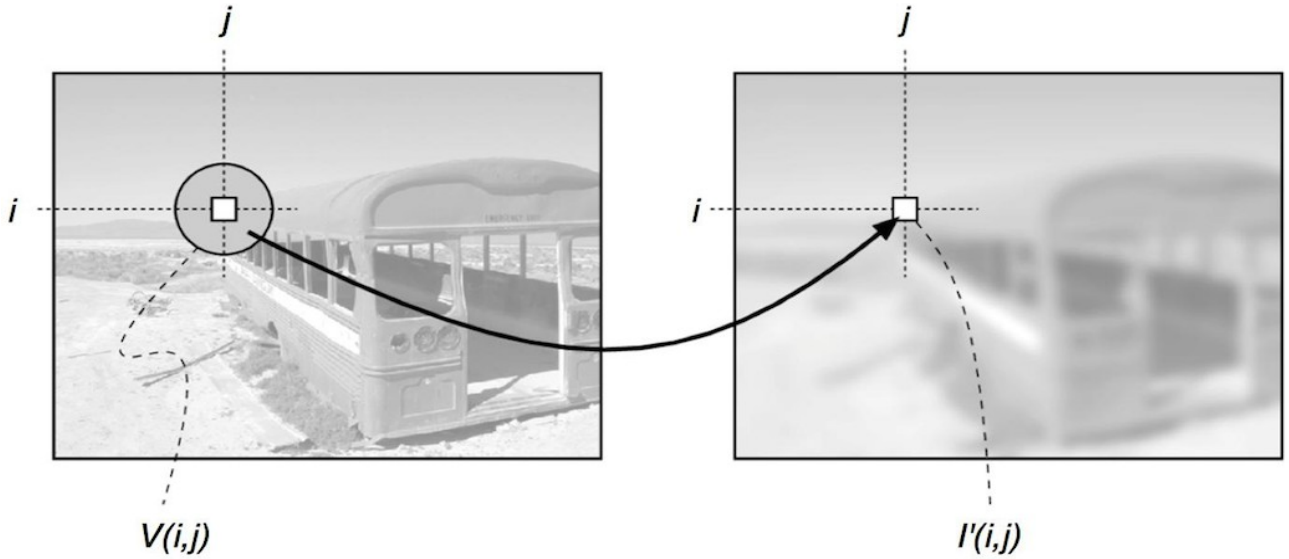
$$g(m,n) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} f(m-k, n-l) h(k,l)$$

## Qu'est-ce qu'un filtre 2D ?

Opérateur local ou « de voisinage » qui permet de :

- Débruiter,
- Détecter les contours,
- Transformer géométriquement l'image, ...

**Principe :** Combiner la valeur du pixel I(i, j) avec son voisinage V(i, j) défini autour de (i, j). Par ex. : moyenne sur V(i, j)

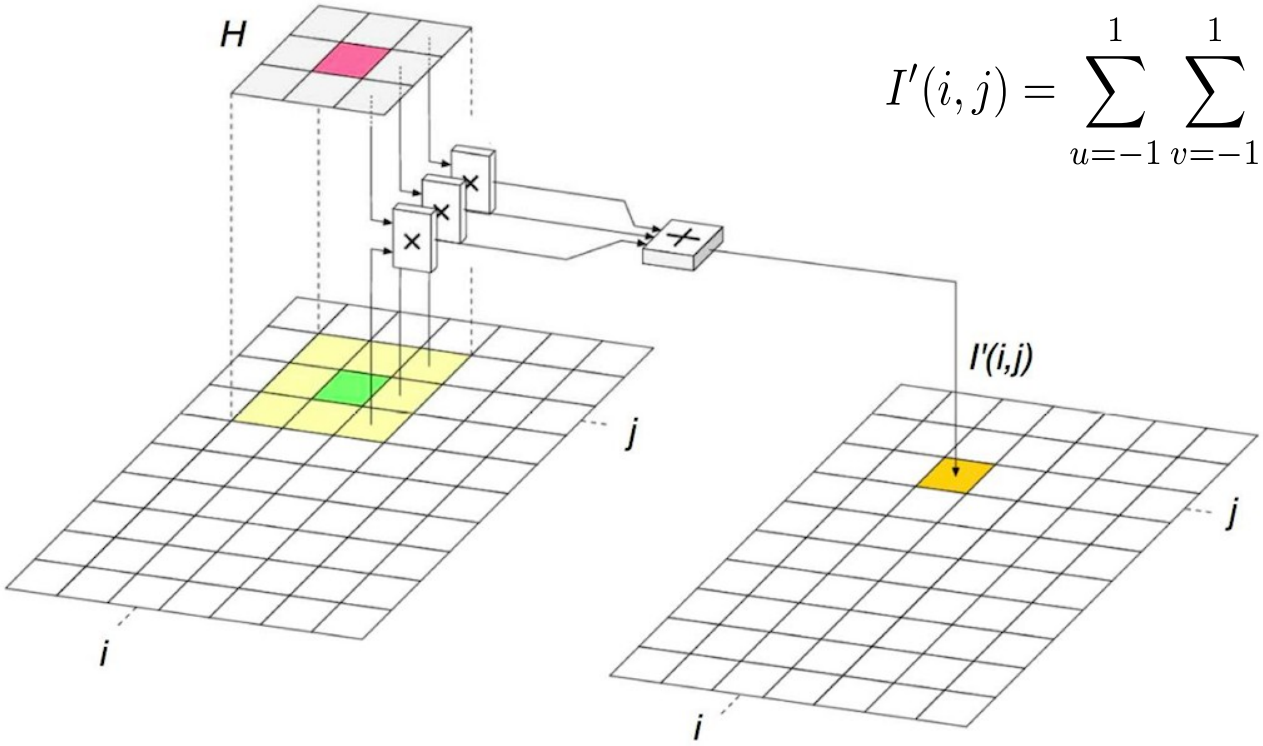


## Produit de convolution 2D

$$I'(i,j) = \sum_{(u,v)\in H} I(i-u, j-v).H(u,v)$$ avec H centré sur (i,j)

Par exemple pour un filtre 3x3 :

$$I'(i,j) = \sum_{u=-1}^{1} \sum_{v=-1}^{1} I(i-u, j-v).H(u,v)$$



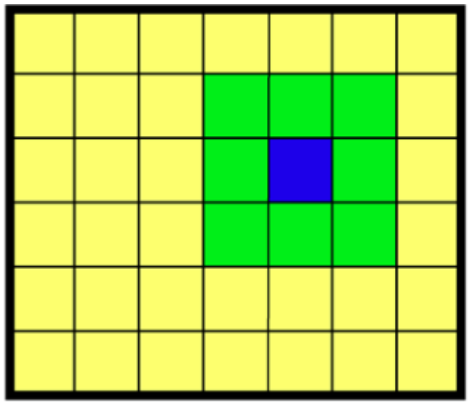
## Exemple : filtre moyenneur

$$I'(i,j) = \frac{1}{|V(i,j)|} \sum_{(u,v)\in V(i,j)} I(i-u, j-v)$$

Différents paramètres :

- Taille : 3x3, 5x5, ...
- Pondération : coefficient spatiaux, souvent isotrope

Exemple : $V(i,j) \rightarrow 3 \times 3 \quad I'(i,j) = \frac{1}{9} \sum_{u=-1}^{1} \sum_{v=-1}^{1} I(i-u, j-v)$

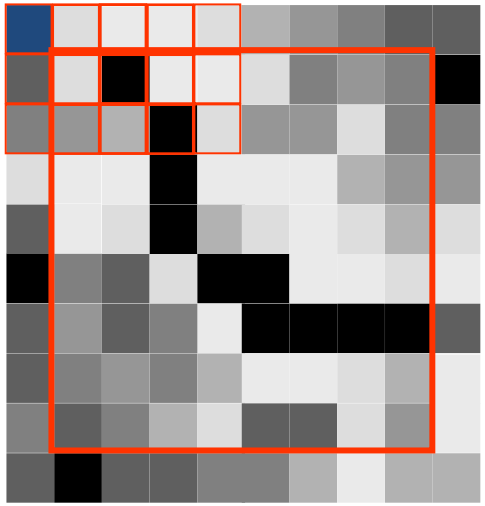


$$H(u,v) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{\mathbf{1/9}} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

# Exemple : filtre moyenneur

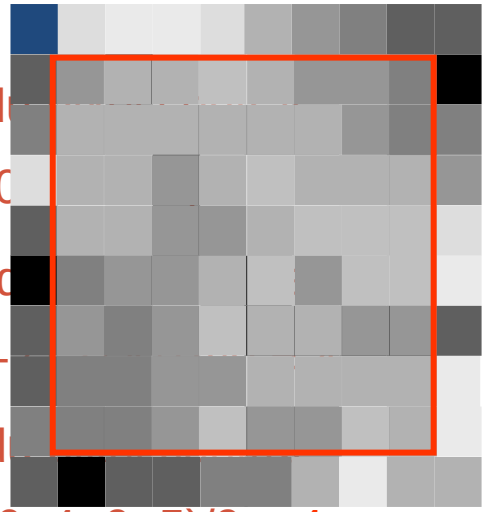
$h$

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |



$A$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |



Novelle valeur d

= (0+5+7+1+5+0

Novelle valeur d

= (5+7+6+5+0+

Novelle valeur d

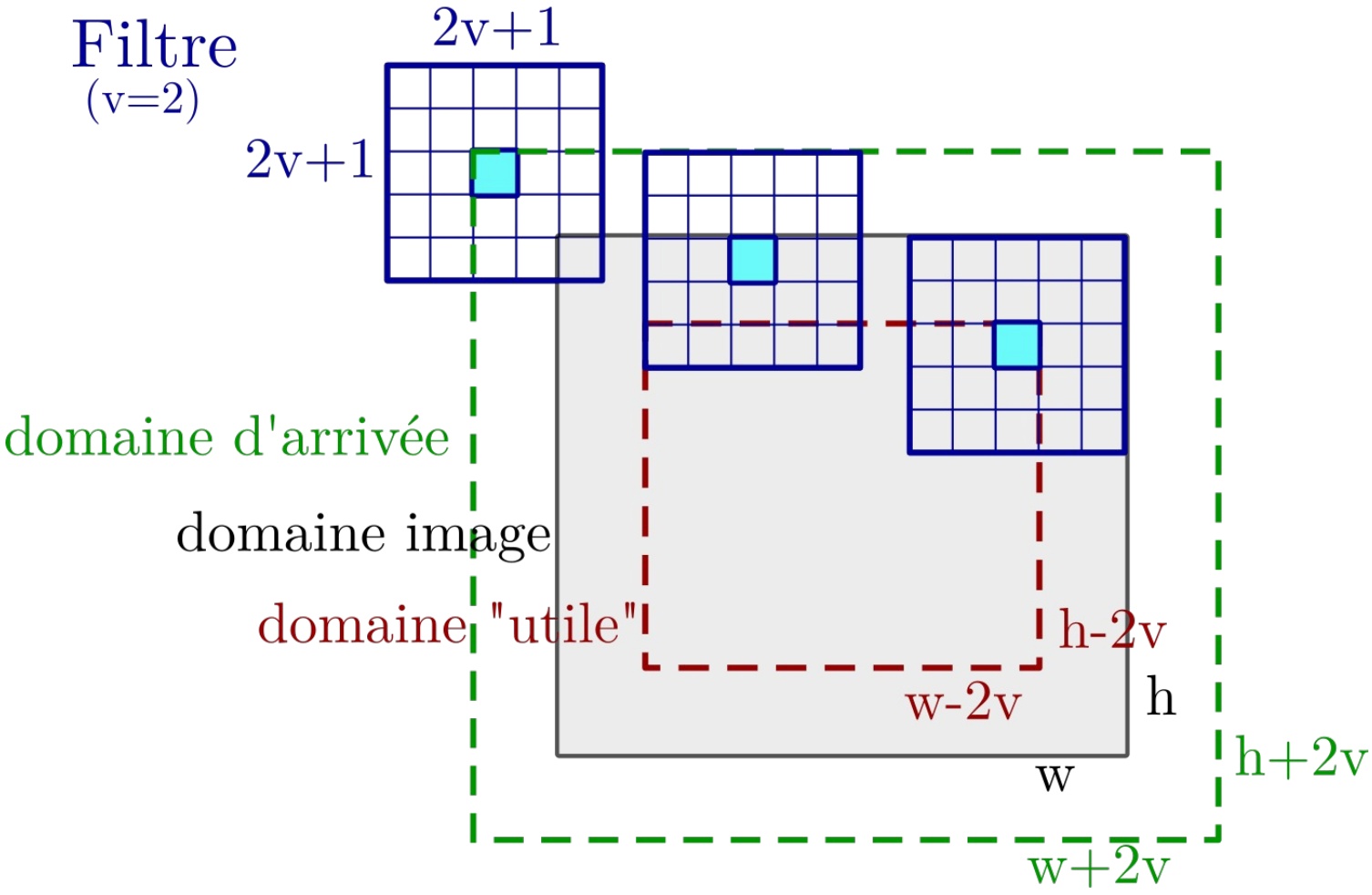
= (7+6+5+0+7+6+4+0+5)/9 = 4

$C$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

## Gestion des effets de bords

- Différents domaines/tailles possibles en sortie de convolution



## Gestion des effets de bords

- Différentes stratégies pour étendre l'image aux bords

zero-padding

extension

miroir

périodique

## Filtres moyenneurs

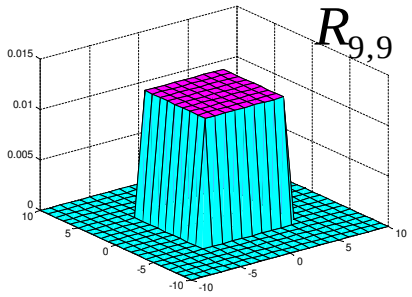
- Filtre rectangle uniforme

normalisation

$$\begin{cases} R_{2K+1} = \dfrac{1}{(2K+1)}\underbrace{\begin{bmatrix}1 & \dots & \dots & \dots & 1\end{bmatrix}}_{2K+1} \\ R_{2K+1,2L+1} = R_{2K+1} * R_{2L+1}{}^{T} \end{cases}$$

$$R_3 = R_{3,1} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

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

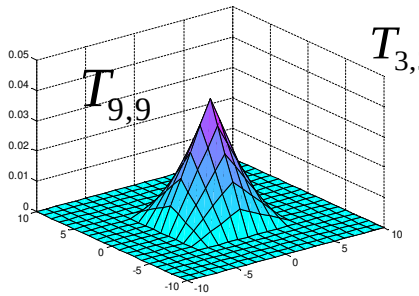



$$R_{3,3} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

plus petit filtre rectangle 2D

- Filtre moyenneur triangle

$$\begin{cases} T_{2K+1} = \dfrac{1}{(K+1)^2}\begin{bmatrix}1 & 2 & \dots & K+1 & \dots & 2 & 1\end{bmatrix} \\ T_{2K+1,2L+1} = T_{2K+1} * T_{2L+1}{}^{T} \end{cases}$$




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

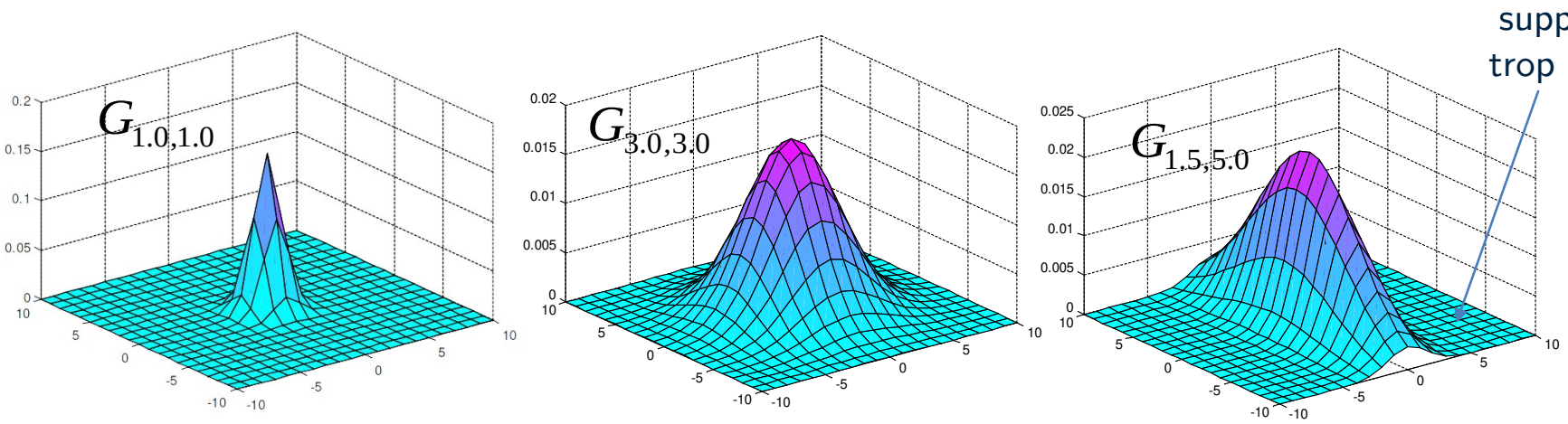
plus petit filtre triangle 2D

## Filtres moyenneurs

- **Gaussien** : Pondère spatialement la contribution des pixels voisins

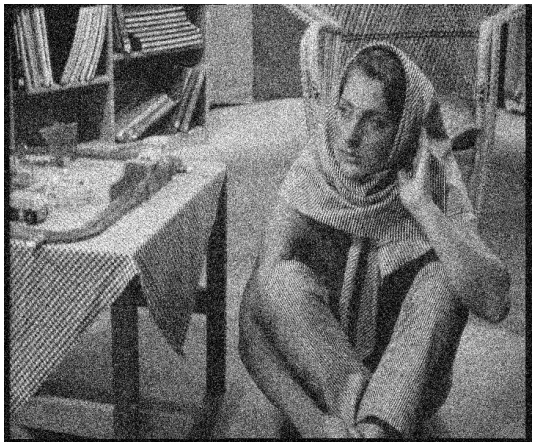
$$H(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)\right)$$

- (x,y) exprime la distance par rapport au centre de la Gaussienne :
- Le paramètre $\sigma$ détermine la largeur de la Gaussienne (écart-type) :
  $\rightarrow$ contrôle un filtrage +/- fort
  $\rightarrow$ permet de connaître la taille du support nécessaire



## Filtres moyenneurs

$R_{11,1}$

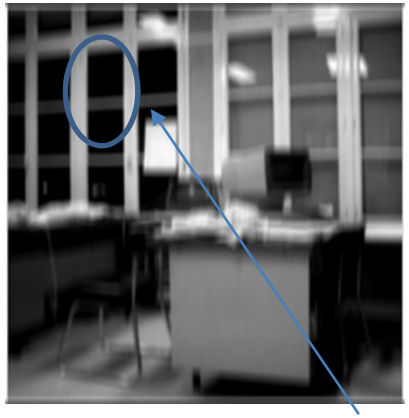


$G_{1.5,1.5}$

étalement

$R_{1,11}$

$G_{3.0,3.0}$



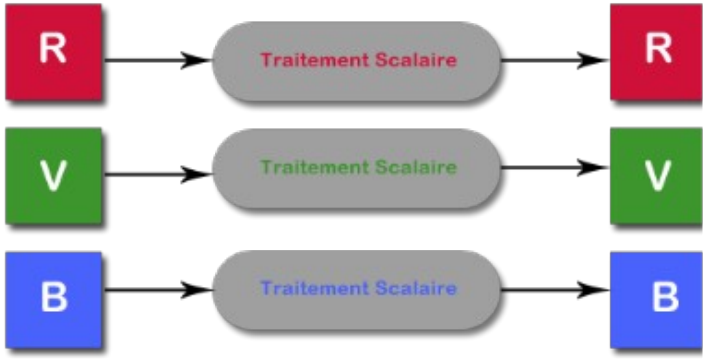
flou vertical

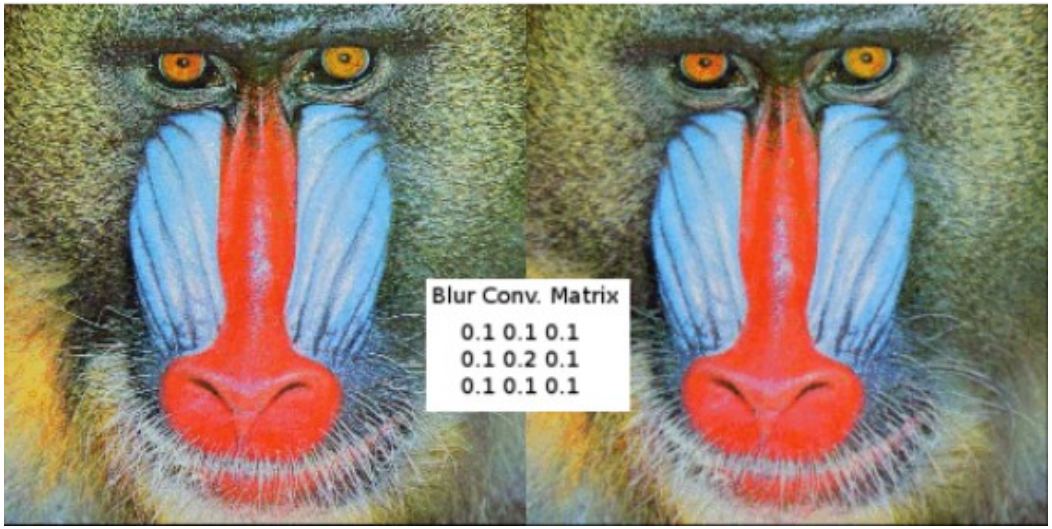
flou isotrope

Atténuation du bruit par moyennage spatial

## Cas d'une image couleur

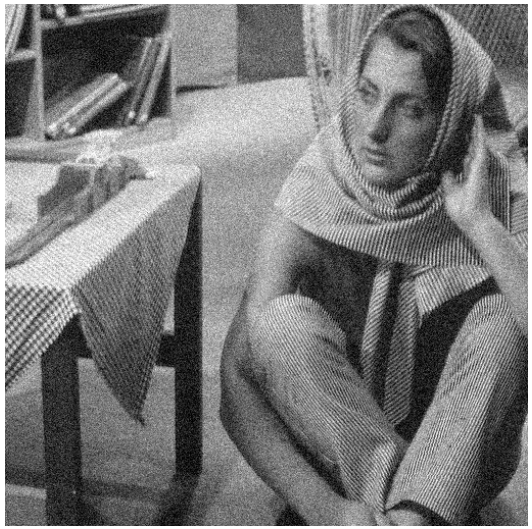
- Combinaison de chaque canal filtré indépendamment



## Filtrage moyenneur (linéaire)

- Implémenter les filtrages linéaires suivants :
  - **Filtrage moyenneur** carré (`signal.convolve2d`)
  - **Filtrage Gaussien** (`np.meshgrid, scipy.signal.convolve2d`)
- Tester sur les images : *barbara_awgn_noise.png*, et *cameraman_sp_noise.png*

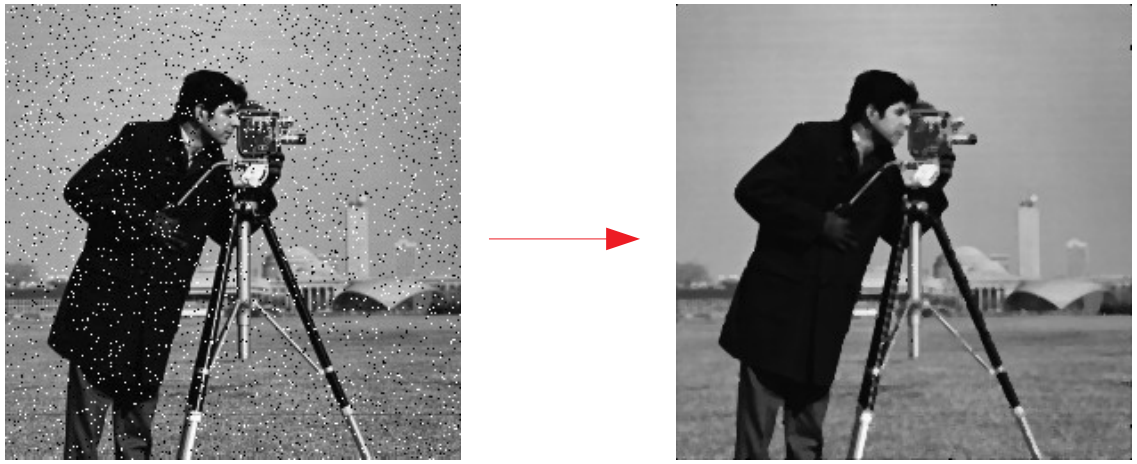


- Pour quelles images les filtrages s'avèrent-ils satisfaisants ? Pourquoi ?

# Filtrage médian (non linéaire)

- Implémenter le filtrage médian (valeur médiane dans un voisinage)

Algo :
```
#Parcourir tous les pixels de l'image
for i in range(v, h-v):
   for j in range(v, w-v):
      #Déterminer un voisinage 2D (de taille (2v+1)*(2v+1))
      #...
      #Calculer la médiane sur l'ensemble des pixels (np.median)
      #...
```

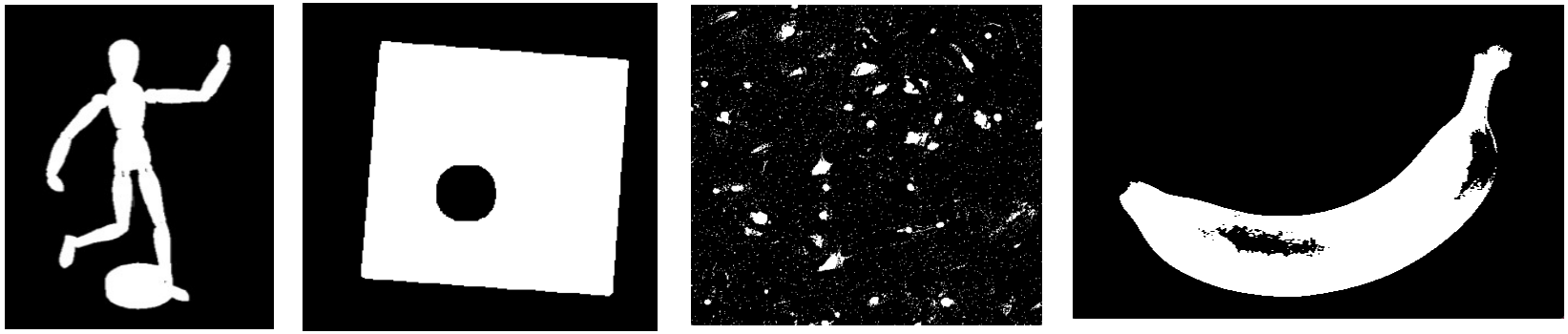


- Comparer votre résultat avec celui de `scipy.ndimage.median_filter`

# Opérateurs de morphologie mathématique

**Liés à la notion d'objets sur des images binaires :**

- Pixels objets identifiés par l'intensité 1 (blanc)
- Pixels de fond identifiés par l'intensité 0 (noir)



Exemples d'images binaires

**Principe :** image binaire (entrée) $\rightarrow$ image binaire (sortie)

- Image de sortie obtenue par convolution, puis seuillage binaire
- Ces opérateurs peuvent permettre de :
  - Supprimer du bruit
  - Grossir/Diminuer la taille d'un objet
  - Décoller/Recoller les composantes (connexes) d'un objet

## Érosion

- Un pixel d'une image $I$ est un pixel objet si la région centrée sur ce pixel ne contient que des pixels objet $\rightarrow$ "rétrécit" l'objet
- Équivalent à convoluer, par ex. par un filtre de type 4 voisins : $h = 1/5 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

$$I_c = I * h$$

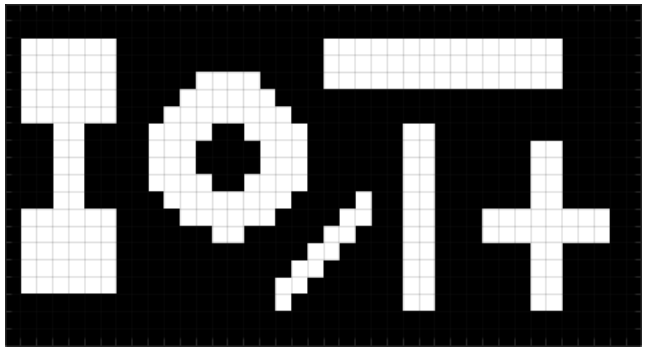


Image initiale $I$

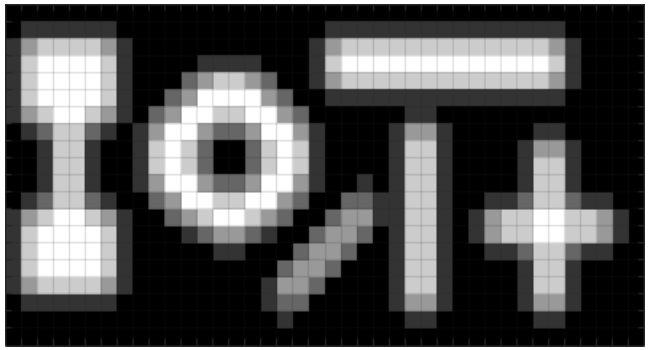


Image convoluée $I_c$

- Puis seuiller de manière binaire :

$$I_e(i,j) = \begin{cases} 1 \text{ si } I_c(i,j) = 1 \\ 0 \text{ sinon} \end{cases}$$

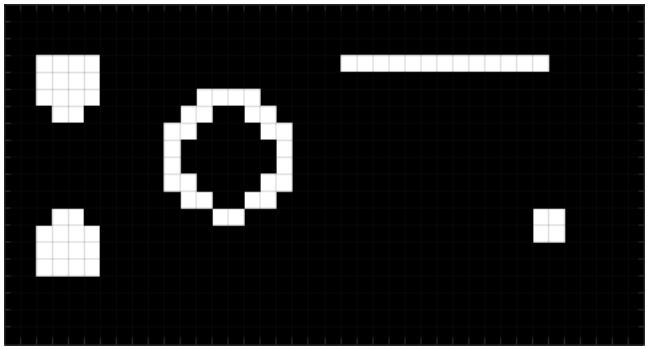


Image érodée $I_e$

## Dilatation

- Un pixel est un pixel objet si la région centrée sur ce pixel contient au moins un pixel objet → "épaissit" l'objet
- Équivalent à convoluer, par ex. par un filtre de type 4 voisins : $h = 1/5 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

$$I_c = I * h$$

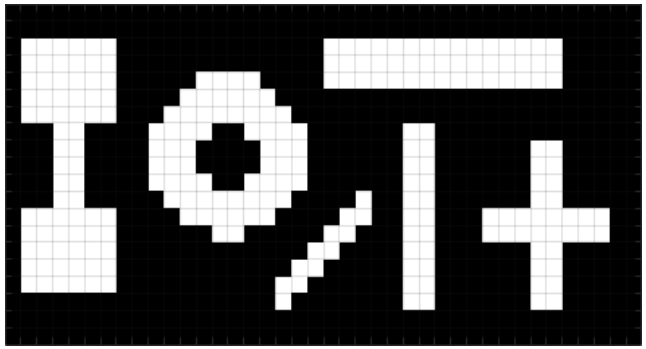


Image initiale $I$

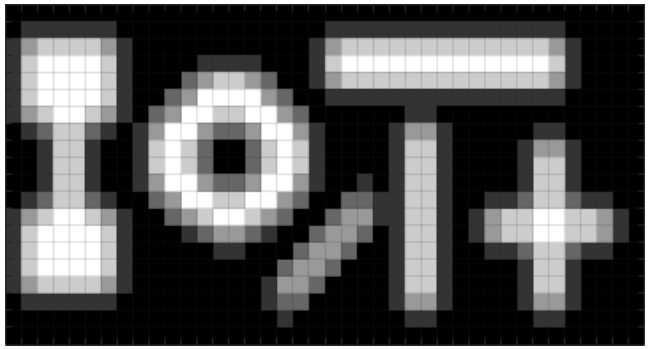


Image convuée $I_c$

- Puis seuiller de manière binaire :

$$I_d(i,j) = \begin{cases} 1 \text{ si } I_c(i,j) > 0 \\ 0 \text{ sinon} \end{cases}$$

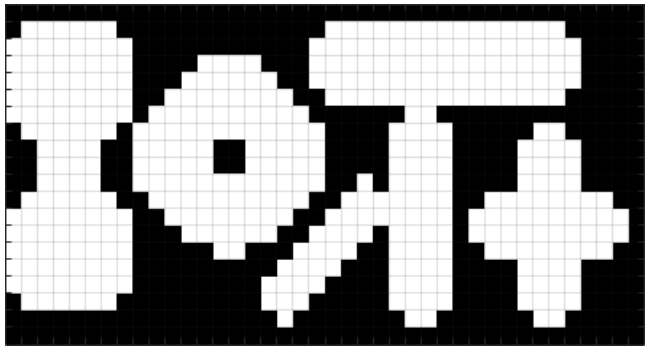


Image dilatée $I_d$

## Fermeture

- Fusionne les structures proches, les trous sont comblés. Permet de recoller les composantes d'un objet.
- Effectuer une **Dilatation** puis une **Érosion**.

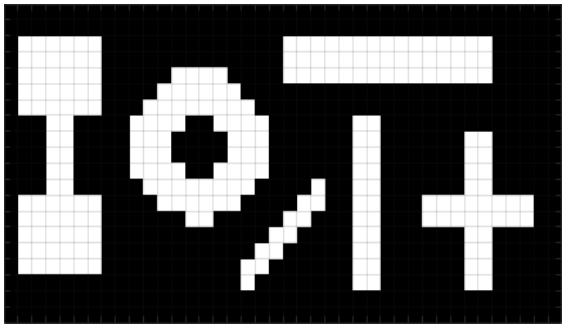


Image initiale

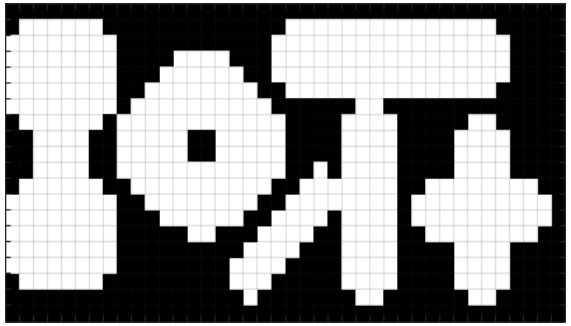


Image dilatée

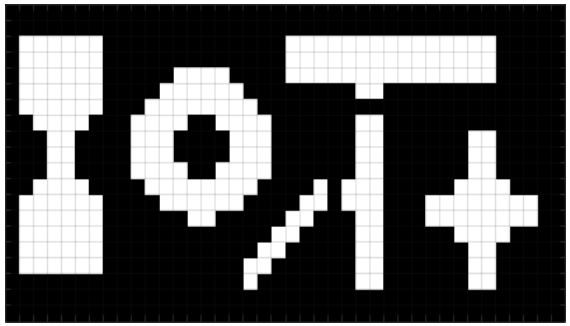


Image dilatée puis érodée

- Possibilité **d'itérer** chaque étape

Image initiale

Image dilatée (x20)

Image dilatée (x20) puis érodée (x20)

## Ouverture

- Supprime les petites structures. Permet de déconnecter des objets indépendants.
- Effectuer une **Érosion** puis une **Dilatation**.

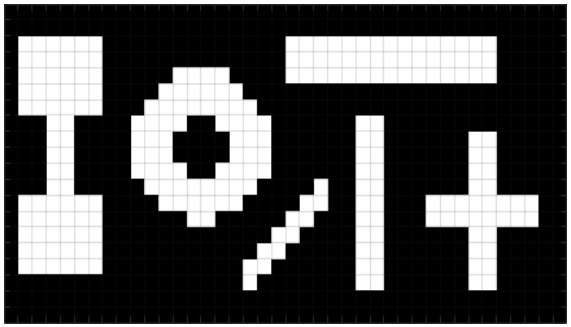


Image initiale

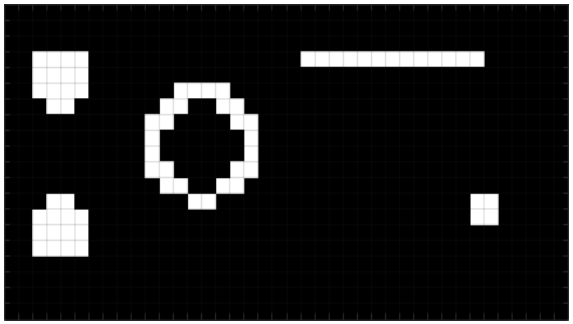


Image érodée

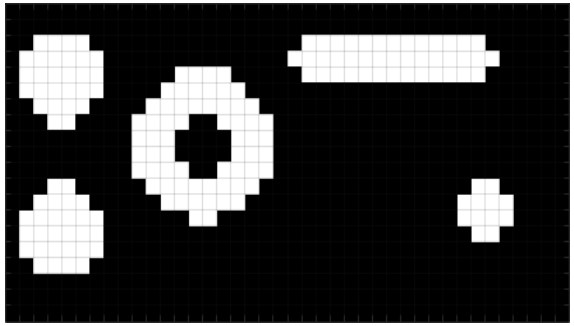


Image érodée puis dilatée

- Possibilité **d'itérer** chaque étape

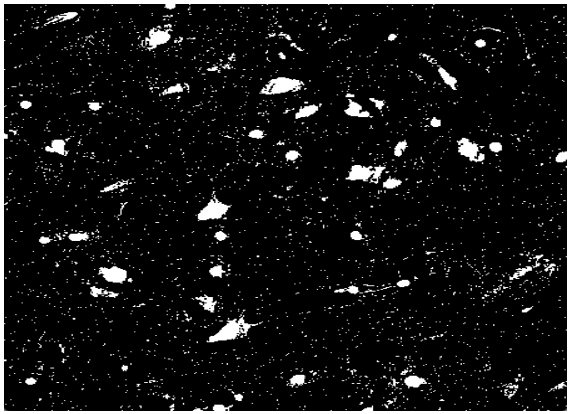


Image initiale

Image érodée (x2)

Image érodée (x2) puis dilatée (x2)

## Composantes connexes

- Ensemble de pixels connectés
  - Au sens d'un critère (par ex. d'intensité 1) et d'un voisinage (4, 8, etc).
  - Possibilité d'atteindre tous les pixels de la composante sans en sortir.
- Étiquetage en composantes connexes : parcours d'une image binaire en assignant un nombre entier (étiquette/label) à chaque composante connexe de l'image. (`skimage.measure.label`)

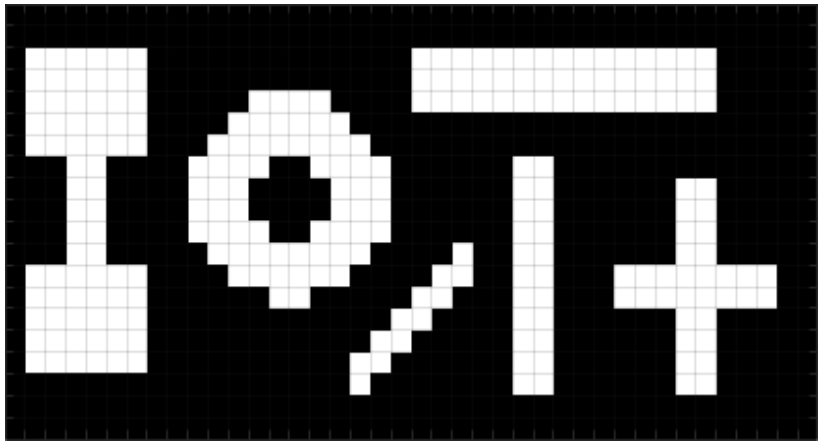
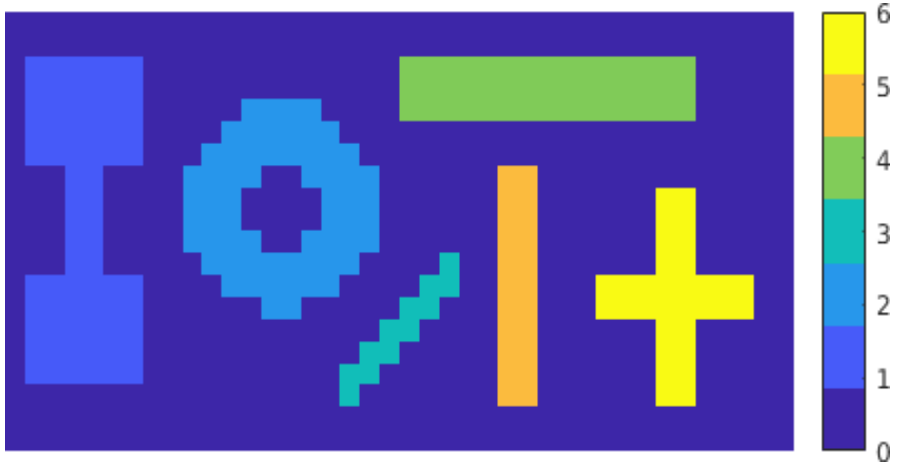


Image initiale

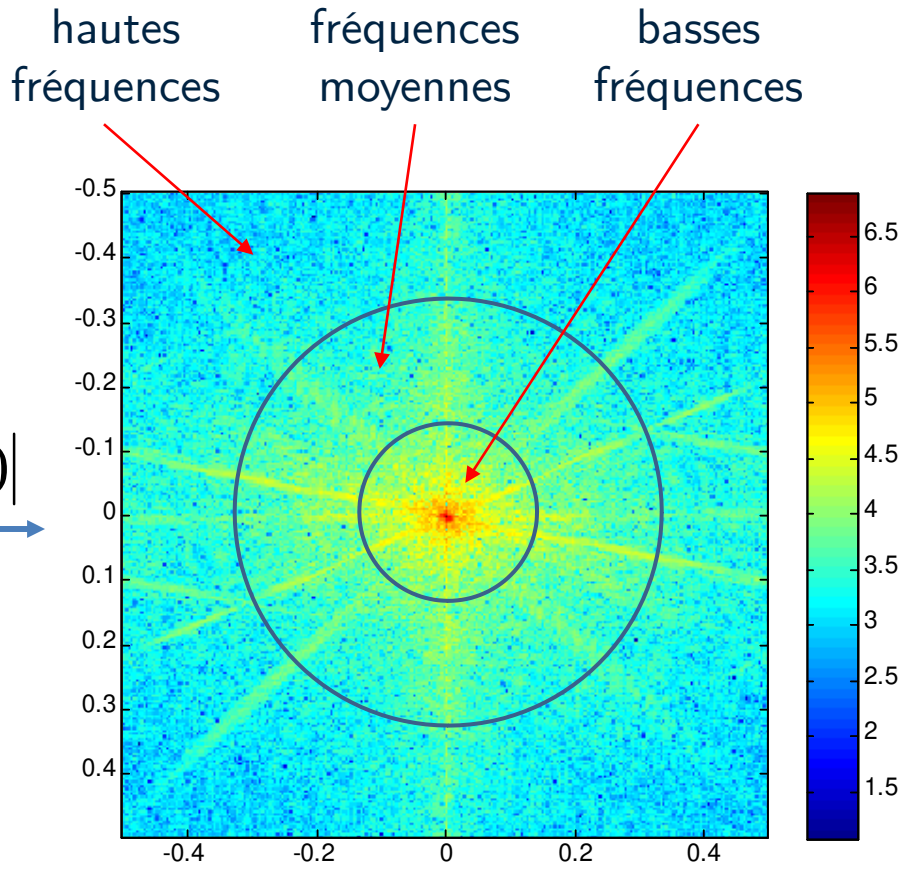



Étiquetage en composantes connexes

# Interprétation fréquentielle

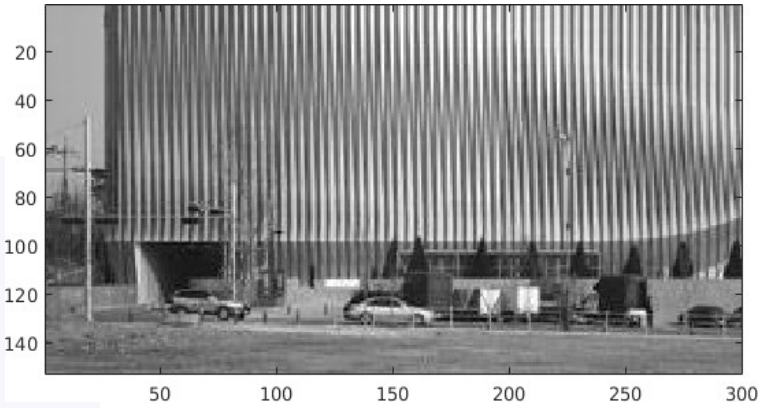
Image naturelle

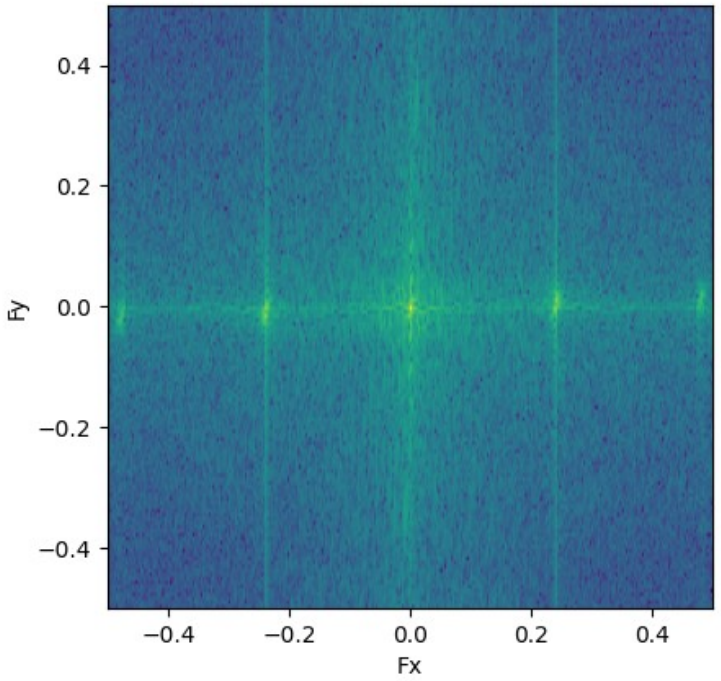
$|TF(.)|$



## Affichage

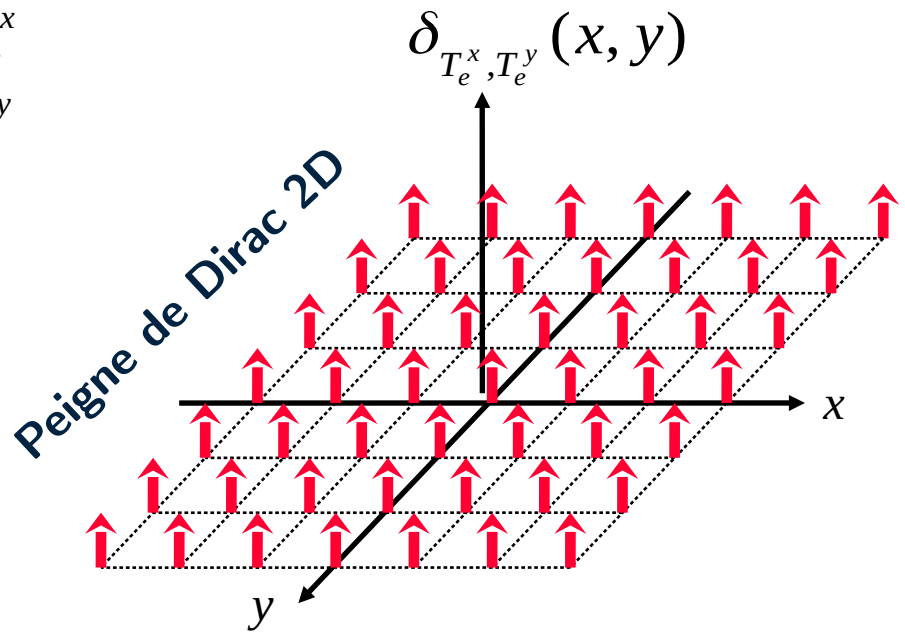
```
import matplotlib.pyplot as plt
import numpy as np

I = plt.imread('./img/building.jpg')
I = I.astype('double')

plt.figure(1)
plt.imshow(I, cmap='gray')
[h,w] = I.shape

#Transformee de Fourier centree en 0
I_fft2 = np.fft.fftshift(np.fft.fft2(I))
#Affichage du log module
I_fft2_lm = np.log10(np.abs(I_fft2))

#Attention à fixer les axes pour interpreter les
#frequences
plt.figure(2)
plt.imshow(I_fft2_lm, extent=[-0.5, 0.5, -0.5, 0.5])
plt.ylabel('Fy')
plt.xlabel('Fx')
plt.show()
```

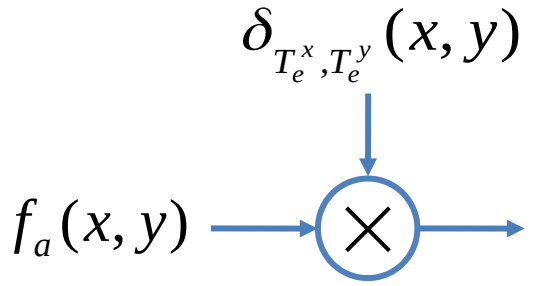


**Domaine discret** : Échantillonnage par un peigne de Dirac

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



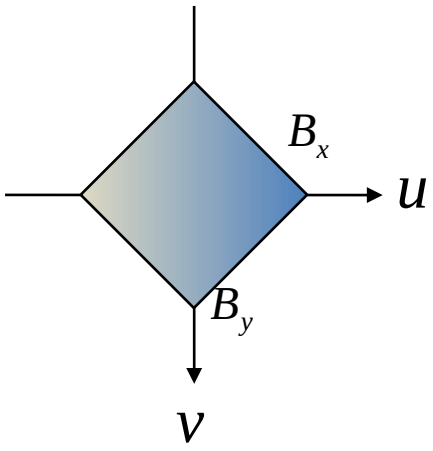

$$f_n(x,y) = f_a(x,y)\,\delta_{T_e^x,T_e^y}(x,y)$$

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

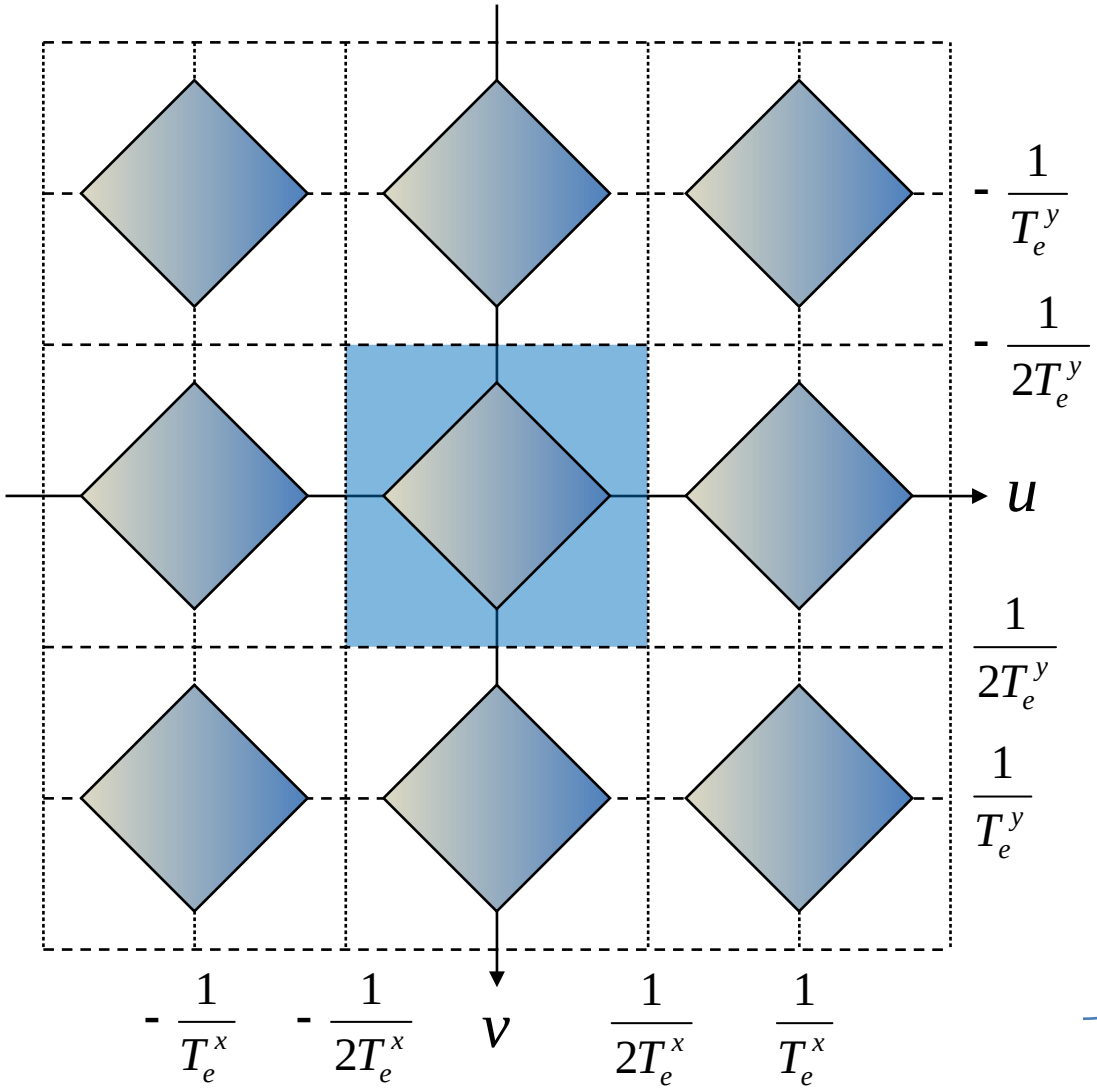
→ Séquence discrète $f_n(x,y)$ notée $f(mT_e^x, nT_e^y)$ ou $f(m,n)$

# Périodisation du spectre dans l'espace fréquentiel

(fe$^x$, fe$^y$)-périodique

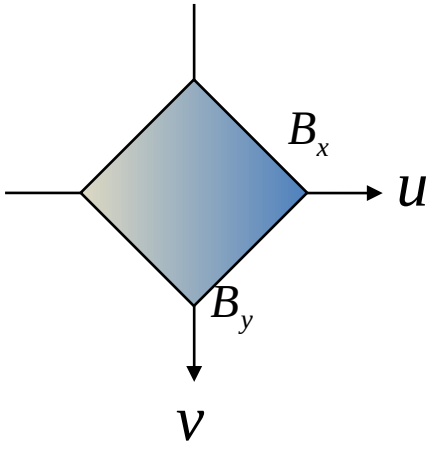



Signal 2D à bande passante limitée

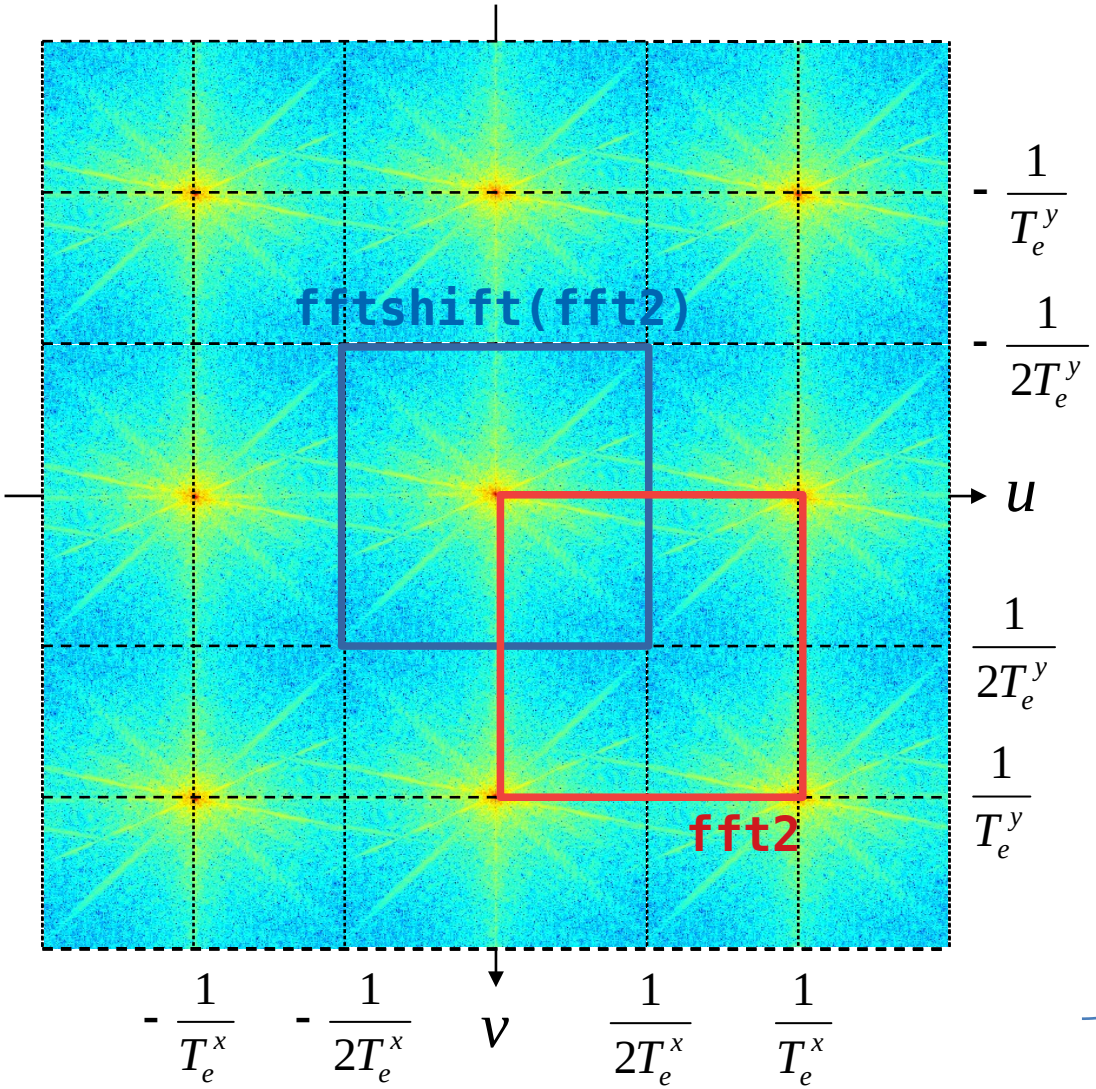


## Périodisation du spectre dans l'espace fréquentiel

(fe$^x$, fe$^y$)-périodique

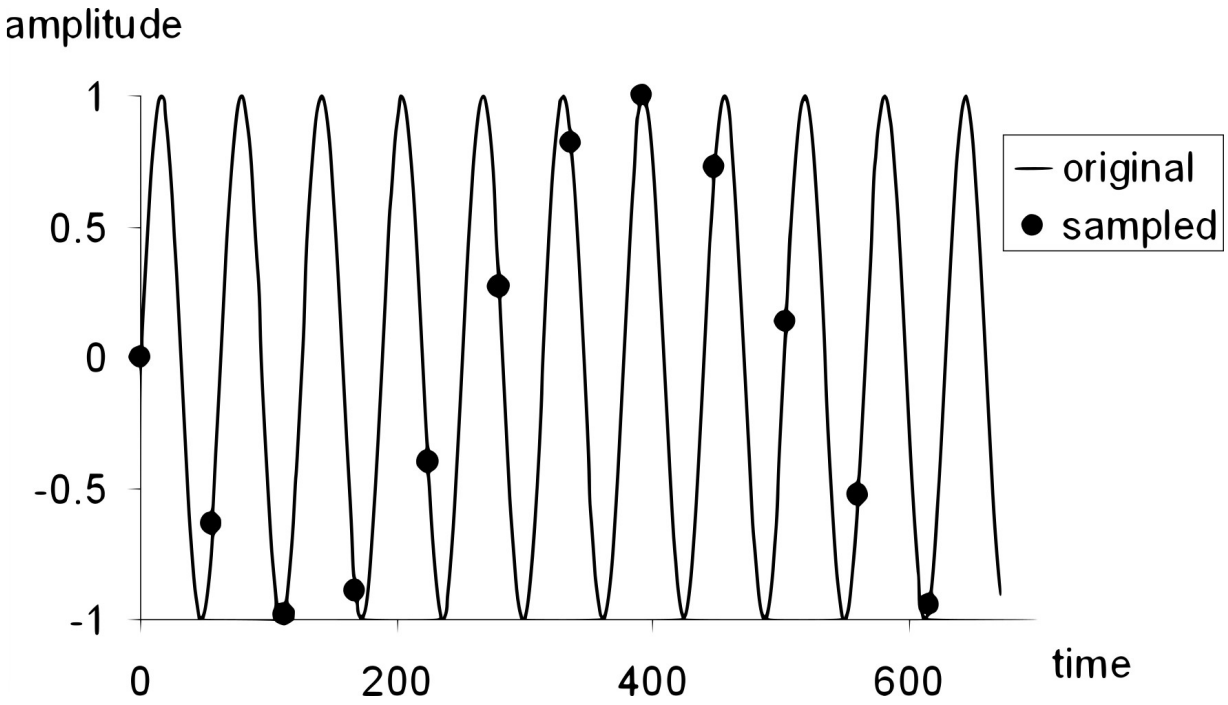



Signal 2D à bande passante limitée



## Phénomène d'Aliasing (ou repliement spectral)

- Choisir une période / fréquence d'échantillonnage adaptée pour capturer les variations du signal
- Sinon risque de dénaturer le signal échantillonné

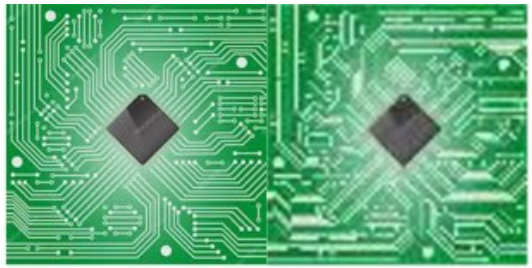


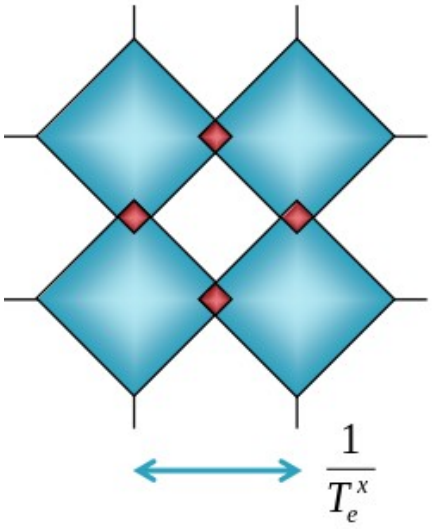


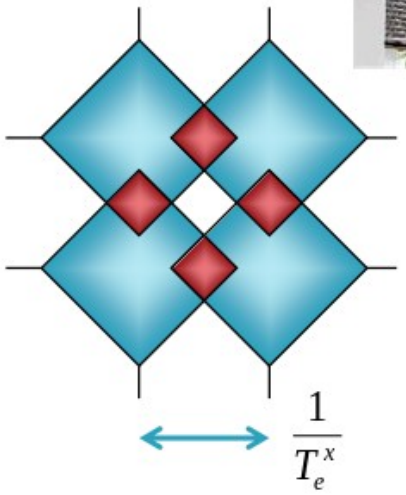

(Source : A. Manzanera)

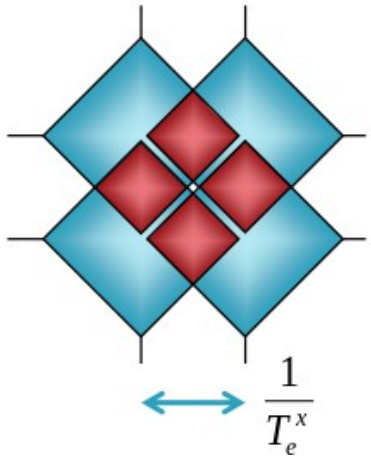
## Phénomène d'Aliasing (ou repliement spectral)

- Sur des images naturelles



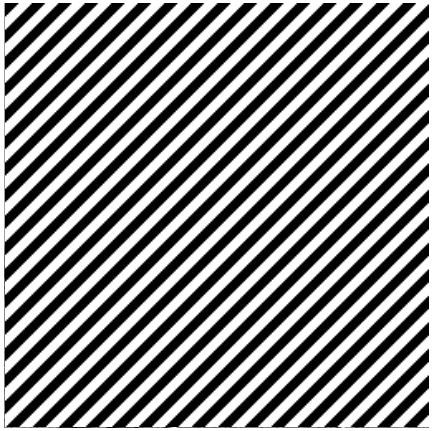


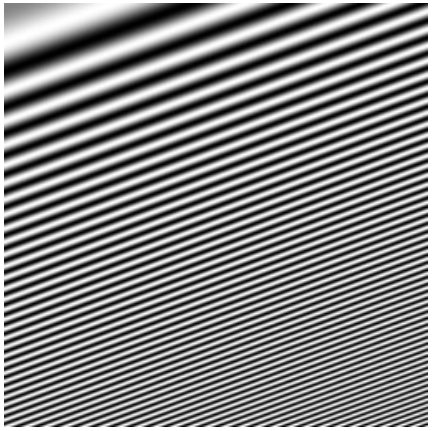



**Quiz :** Retrouver à quelles images correspondent ces transformées

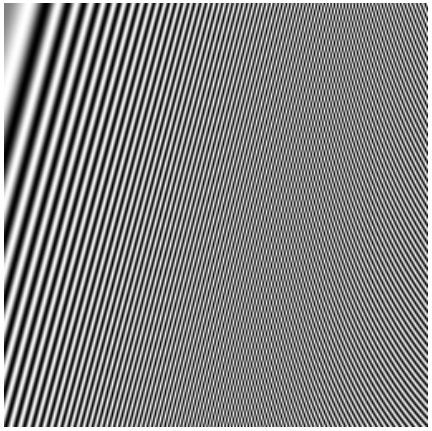
(1)



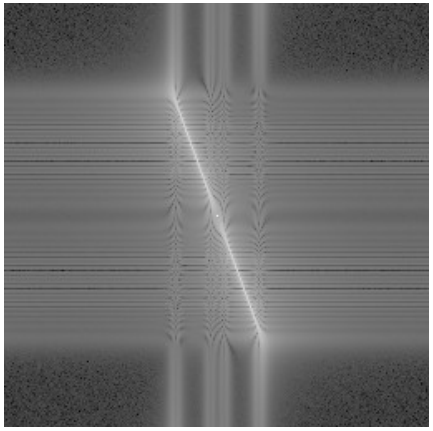
(2)



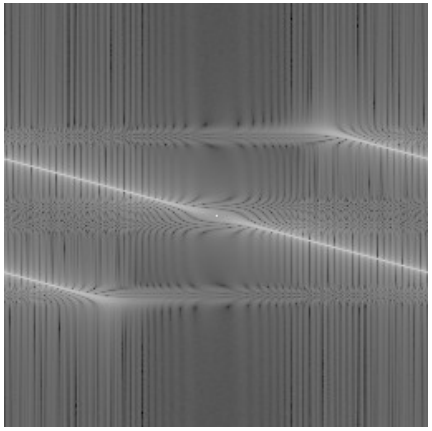
(3)



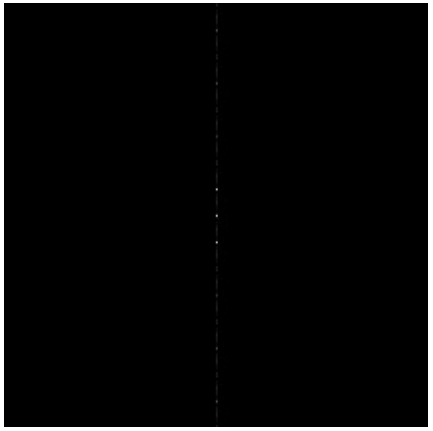
(4)



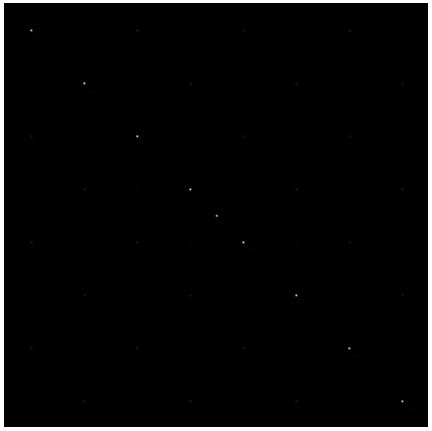
(a)



(b)



(c)



(d)



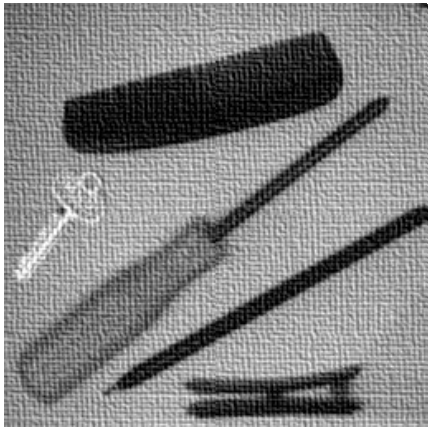
*(Source : A. Manzanera)*

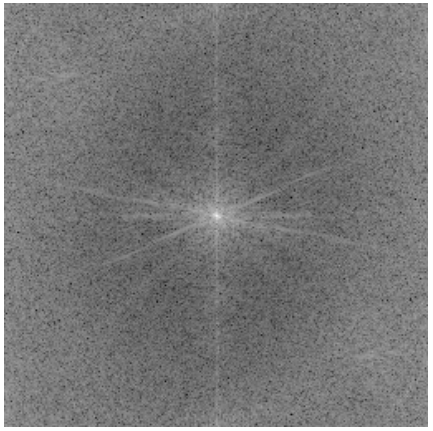
**Quiz :** Retrouver à quelles images correspondent ces transformées

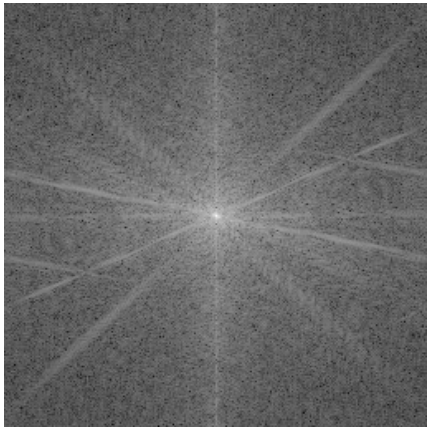

(1)

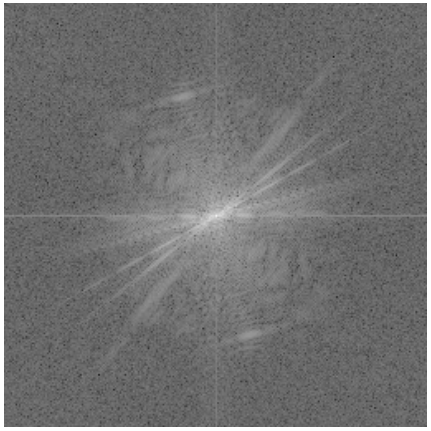

(2)

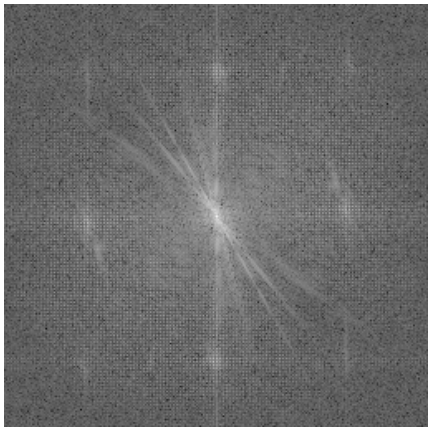

(3)


(4)


(a)


(b)


(c)


(d)



*(Source : A. Manzanera)*

## 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 `skimage.transform.resize`.

Image initiale (h x w)

Image sous-échantillonnée (h/4 x w/4)

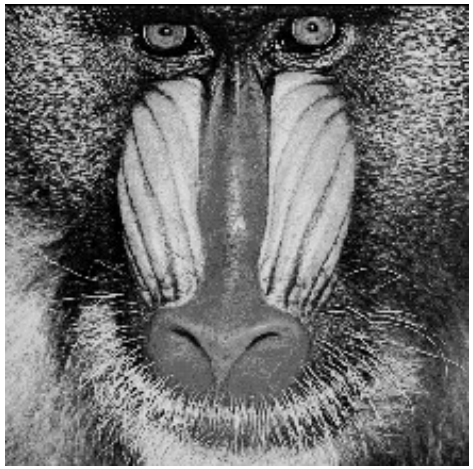
Image filtrée et sous-échantillonnée (h/4 x w/4)

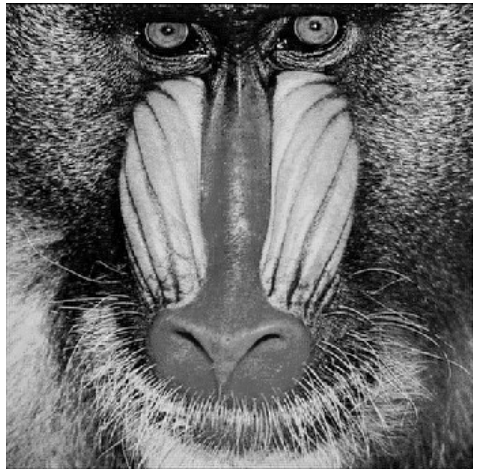
## Exemples d'application – Analyse de textures

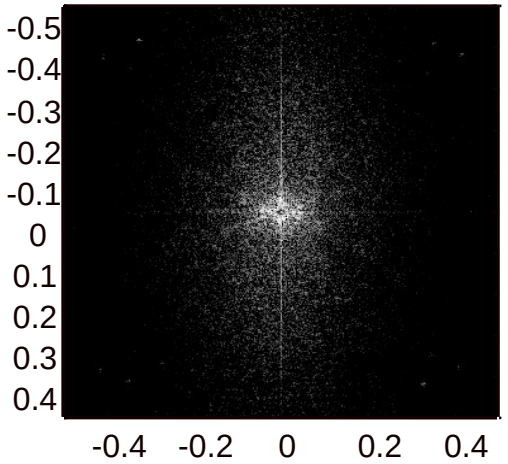
- TF efficace pour caractériser les textures

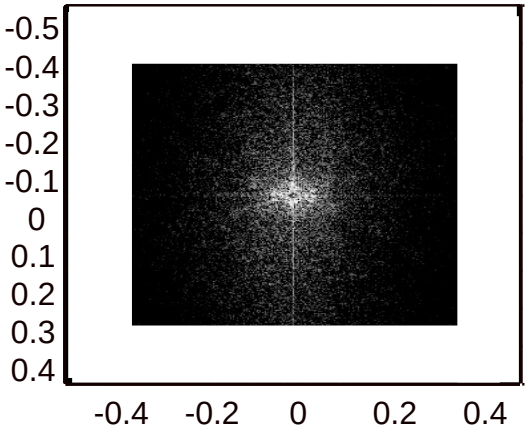
## Exemples d'application – Compression d'images (JPEG)

- Suppression des hautes fréquences → implémentation au chap. 6



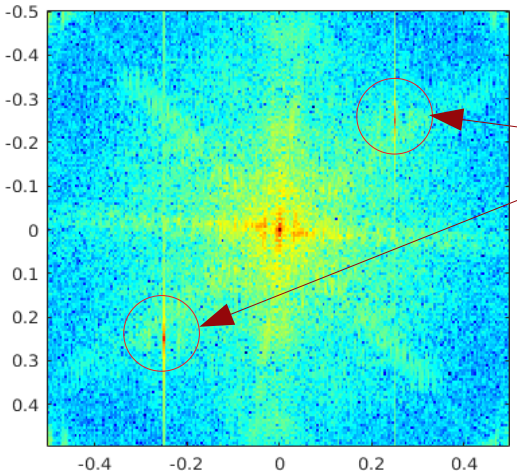




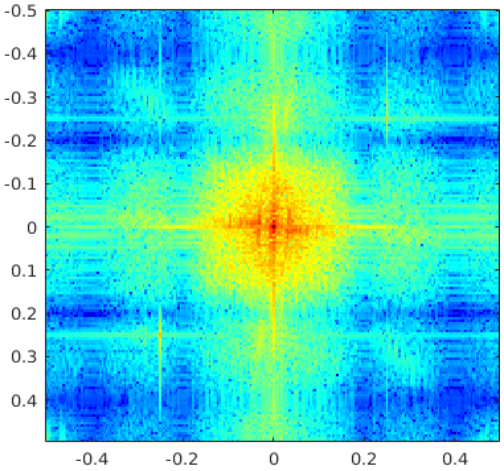

Mise à zéro des coefficients haute fréquence

- **Exemples d'application – Filtrage fréquentiel**
  - Bruit fréquentiel. Inefficacité des filtres moyenneurs classiques

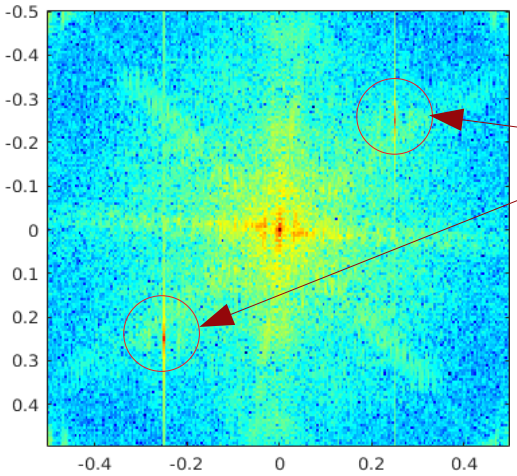



présence du bruit

Filtre moyenneur carré 5x5

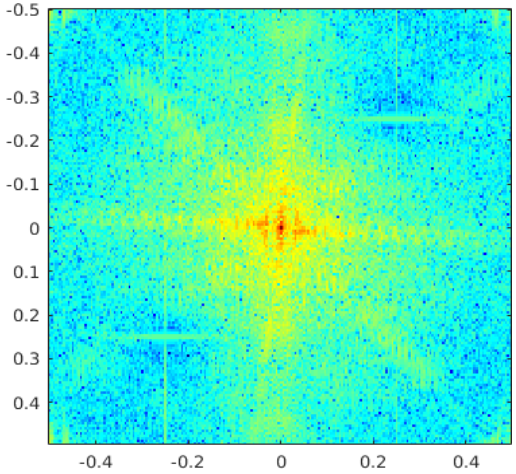


- **Exemples d'application – Filtrage fréquentiel**
  - Filtrage par un filtre coupe bande → implémentation en TP

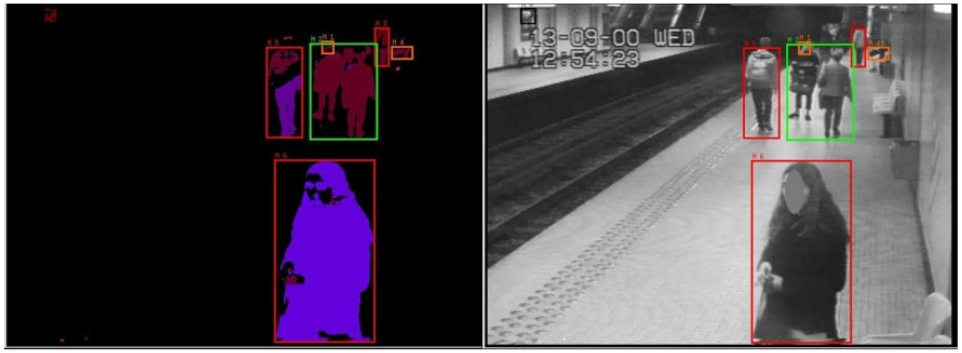


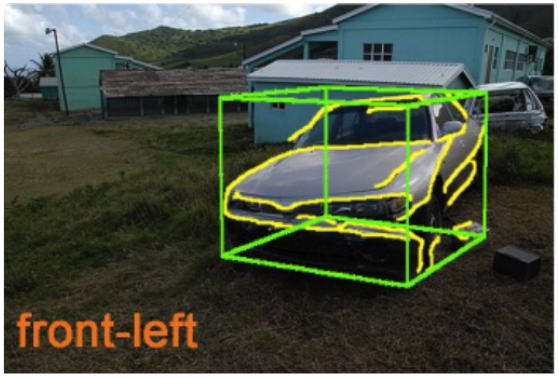

présence du bruit

Filtre coupe bande spécifique



## Intérêt



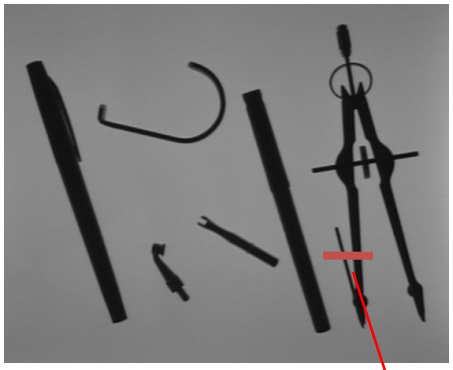


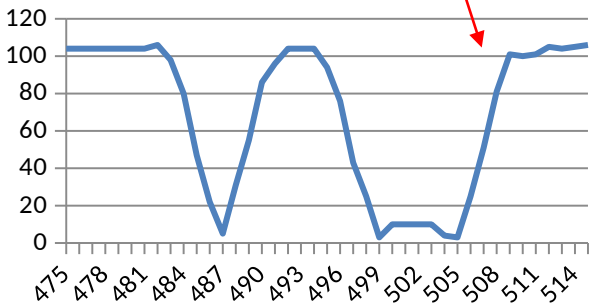


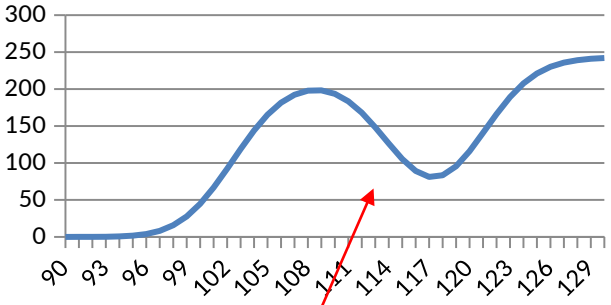
# Définition

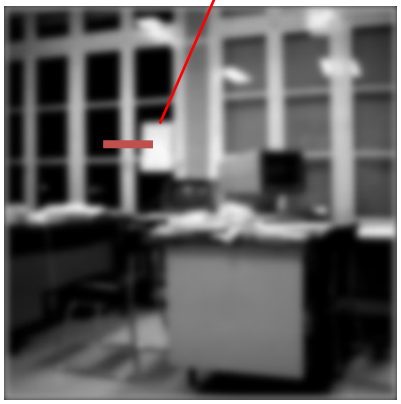
- Bord ou limite d'une région (objets, intensités)
- Séparation ou frontière entre régions
- Variation plus ou moins rapide d'un front d'intensité

Contours nets



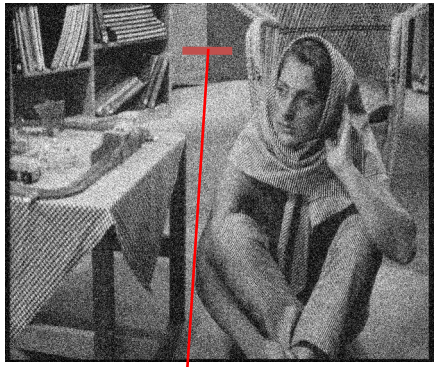


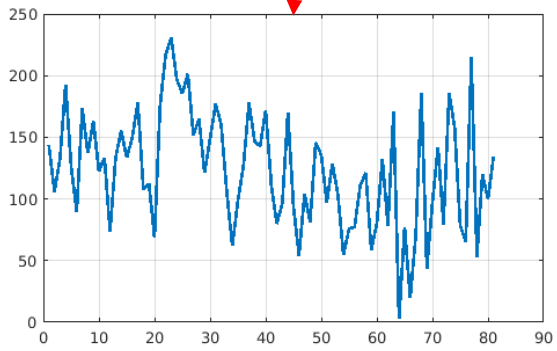



Contours flous

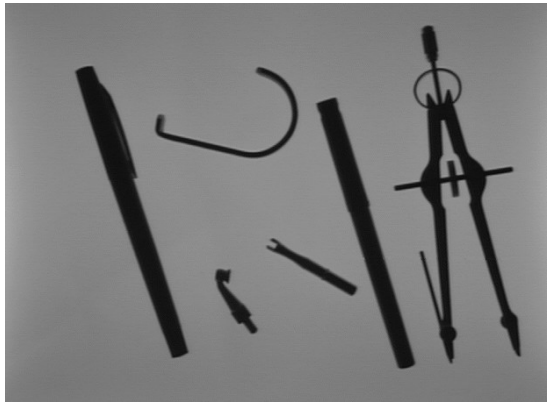
Contours bruités

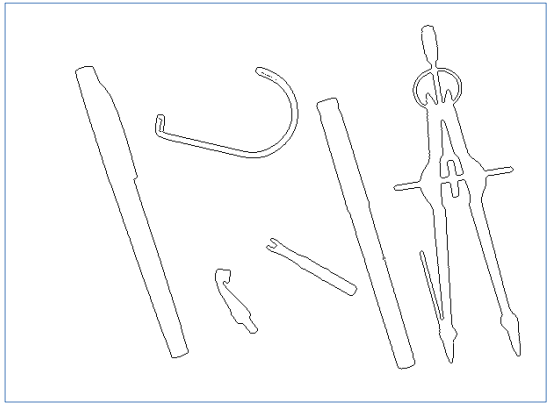




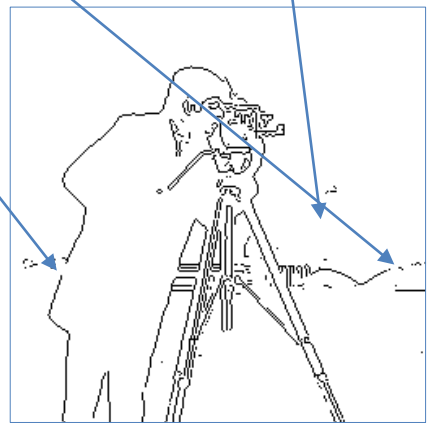
## Définition

- Cas simple : contours nets et quasi fermés, rares contours superflus





- Cas complexe : contours ouverts, discontinus, non définis



## Filtres de Sobel

- Deux filtres dérivateurs Sx et Sy dans les deux dimensions

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

Filtres de Sobel

$$S_x = \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$$

dérivateur horizontal

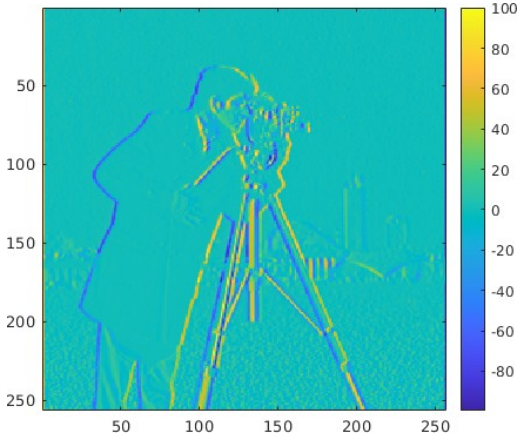
$$S_y = S_x^{\,T}$$

dérivateur vertical

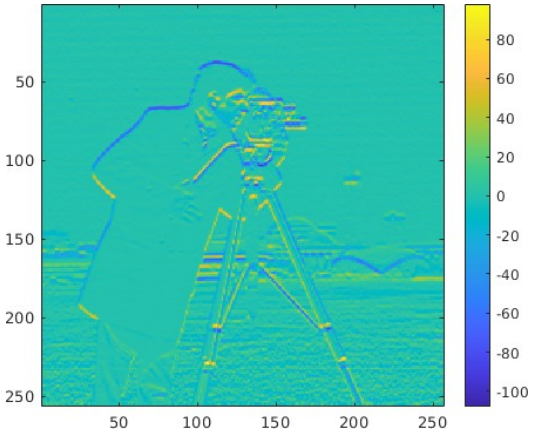
# Filtres de Sobel

- Combinaison des deux détections puis seuillage pour obtenir une carte binaire de détection de contours

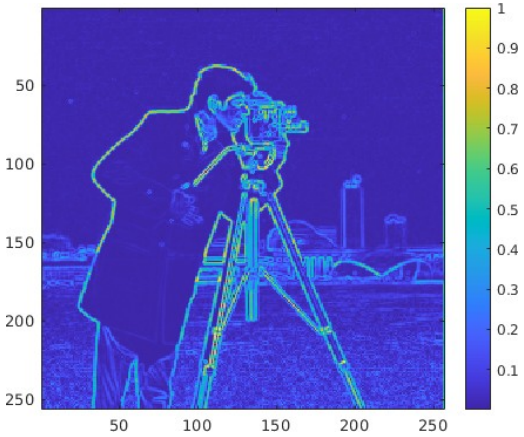
Image initiale



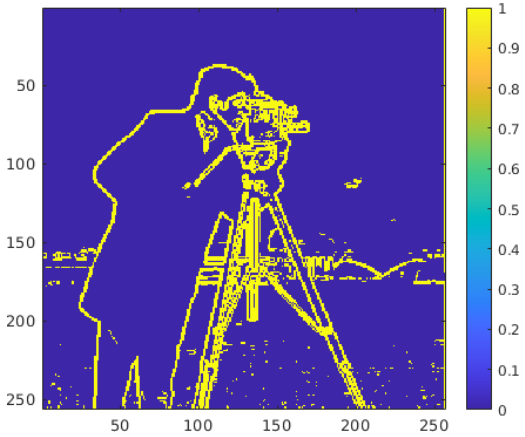

Carte de détection en X



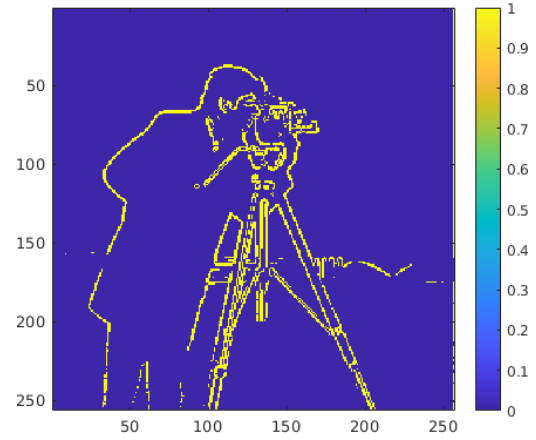

Carte de détection en Y




Carte globale normalisée




Carte seuillée (seuil=0.25)




Carte seuillée (seul=0.4)

## Filtres de Sobel

- Créer les deux filtres de Sobel :

Filtres de Sobel

$$S_x = \frac{1}{2}\begin{bmatrix}1 & 0 & -1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$$ dérivateur horizontal

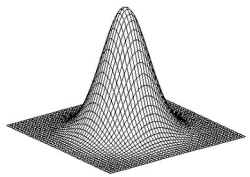
$$S_y = S_x^{\,T}$$ dérivateur vertical

- Convoluer l'image *cameraman.tif* avec ces deux filtres et observer les deux réponses.
- Calculer la norme 2 de ces deux cartes pour obtenir une unique carte d'intensité de contours.
- Normaliser cette carte entre 0 et 1.
- Seuiller cette carte pour obtenir une détection binaire des contours.

## Approche de Canny

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

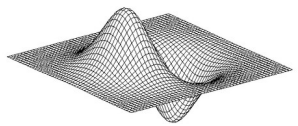
**Ordre 1** $\nabla I \approx I * \nabla G$ avec $G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)}$



lissage et dérivation

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

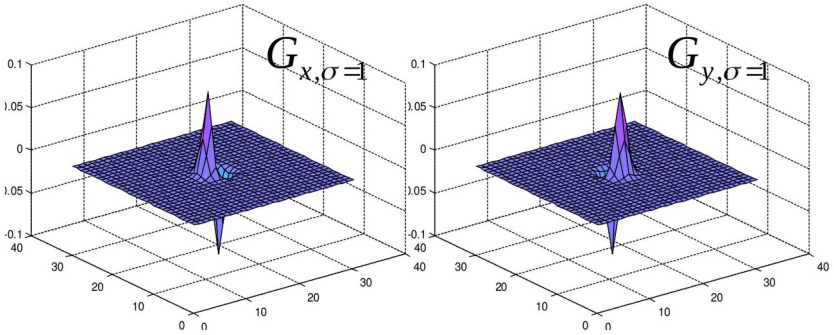
$$\nabla G = \begin{bmatrix} G_x(x,y) = -\frac{x}{2\pi\sigma_x^3\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)} \\ G_y(x,y) = -\frac{y}{2\pi\sigma_x\sigma_y^3} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)} \end{bmatrix}$$

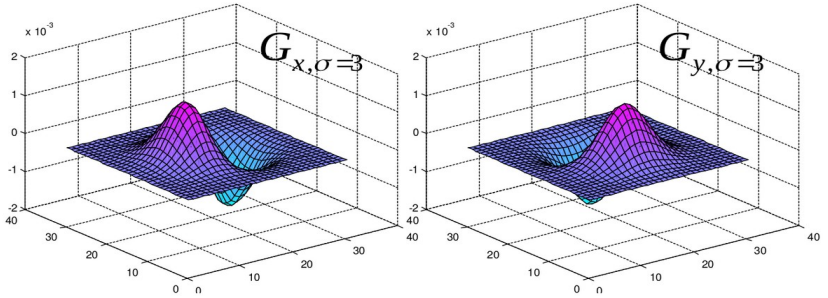


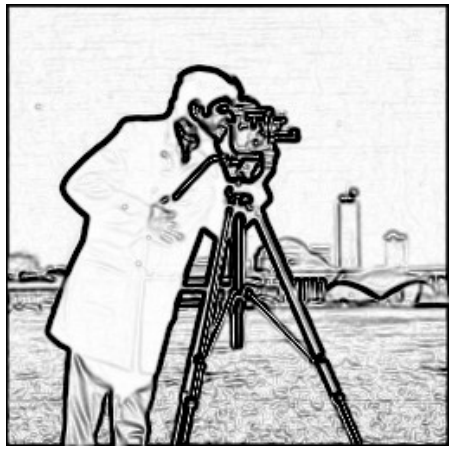
Filtres 2D RIF par échantillonnage et troncature

# Approche de Canny

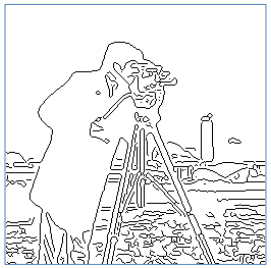
- Compromis entre sur et sous-détection

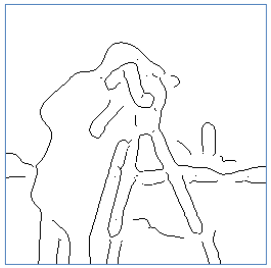


Norme du gradient
à petite échelle





Norme du gradient
à grande échelle

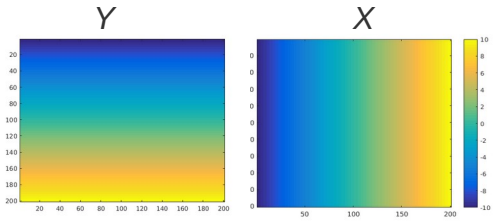
# Approche de Canny

$$g(x,y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Créer un filtre issu d'une dérivée de Gaussienne 2D :

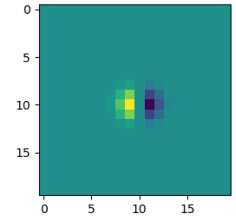
  Utiliser meshgrid pour créer un pavage d'indices :

```
P = range(-10,10)
X, Y = np.meshgrid(P,P)
```

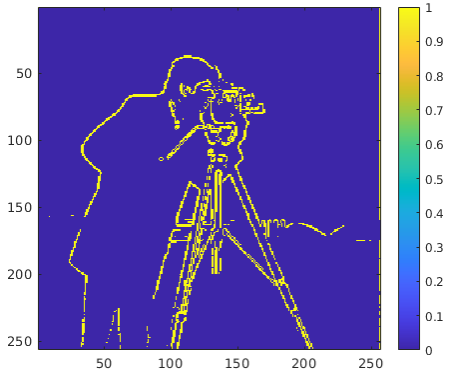



  On retrouve l'expression des dérivées en x et y de g pour leur fournir les cartes du meshgrid et obtenir deux filtres détecteurs de contours Gx Gy :

```
sig = 1
Gx = -X/(2*np.pi*sig**4)*np.exp(-(X**2+Y**2)/(2*sig**2))
```



- Convoluer les filtres avec l'image pour obtenir les deux réponses et calculer leur norme pour obtenir une carte de détection de contours.
- Normaliser la carte entre 0 et 1.
- Utiliser cette méthode dans l'application Pencil Sketch pour calculer la carte de contours C sur home.jpg. Comparer avec le résultat de `skimage.feature.canny`.



## Détection de contraste

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

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

Filtre 1D RIF horizontal

Filtre 1D RIF vertical

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

somme

Filtre 2D RIF

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

Forme en croix (4 voisins)

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

Forme diagonale (4 voisins)

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

Forme cumulée « isotrope » (8 voisins)

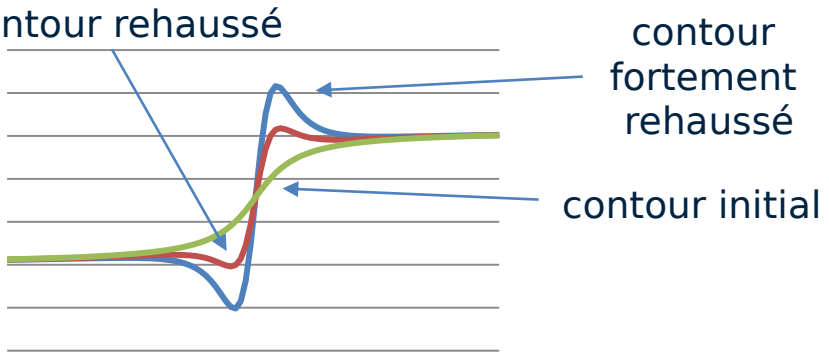
## Rehaussement de contraste

- Similaire au mode « Netteté » sur les smartphones.

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

paramètre de contrôle

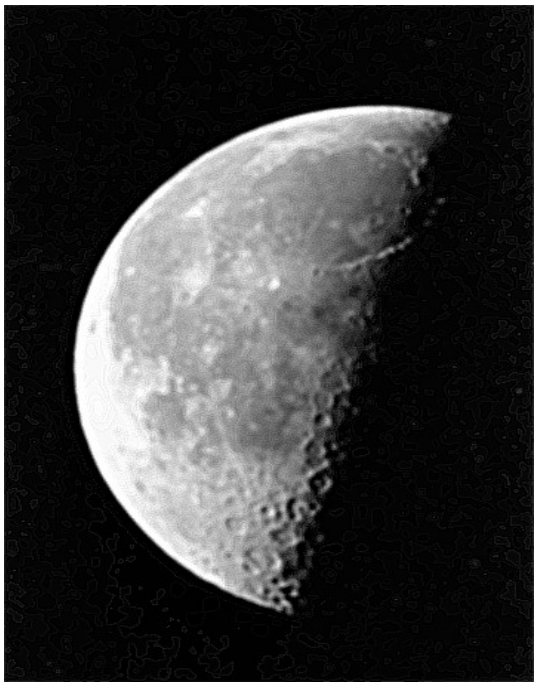
convolution de I avec la forme cumulée

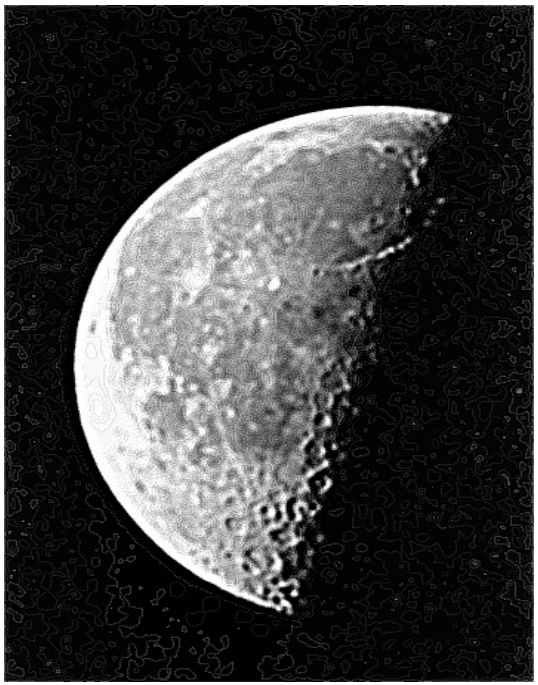


## Rehaussement de contraste

- Implémenter le rehaussement en utilisant la version isotrope du filtre.
- Tester sur les images *moon.png*, *cat.jpg*.
- Quel est le risque avec la transformation effectuée ? Regardez les valeurs.

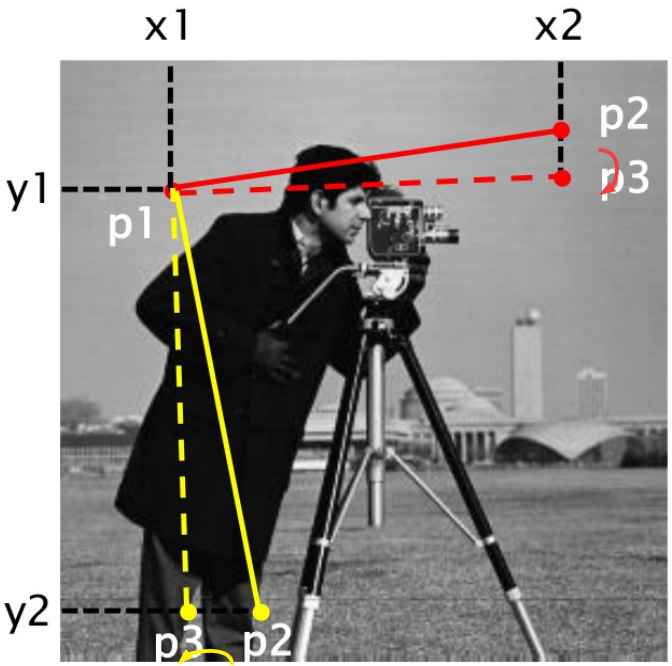

Image initiale

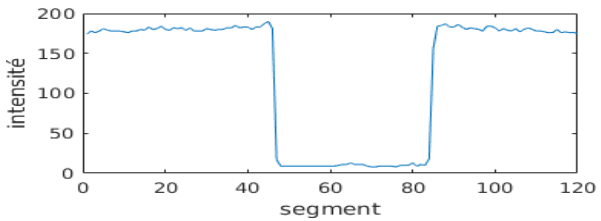

Rehaussement moyen

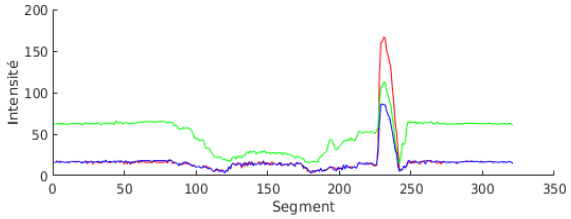

Rehaussement fort

## Extraction de lignes

- Affichage d'une image
  (par exemple *cameraman.tif*)
- Définir un segment horizontal
  ou vertical par 2 clics (`plt.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
- Superposer les profils R, G, B
  d'une image couleur



## Filtrage moyenneur (linéaire)

- Flouter l'image *street.png* avec un filtre moyenneur carré
- Afficher l'image et identifier la position du centre d'un visage
- Créer un masque binaire avec des 1 autour de ce centre
- Combiner les deux images grâce au masque pour obtenir une image où seuls les visages sont floutés.
- Adapter le pipeline pour l'image en couleur et plusieurs positions.

# Objectifs

## Traitement d'images

- Culture du traitement d'images et des applications
- Comprendre le système de vision humain et ce qu'est une image couleur
- Savoir afficher de manière pertinente n'importe quel contenu 2D
- Savoir identifier des dégradations et des solutions adéquates
- Outils : changement d'espace, filtrage, Fourier, compression, ...

## Programmation Python

- Prise en main de Spyder
- Rappels de syntaxe Python (import, indentation, boucles, fonctions, ...)
- Librairies *numpy, matplotlib, scikit-image*
- Manipulation de tableaux (*np.array, 0:L-1, axis=X, opérations vectorielles)*

# ANNEXES

# Principe :

Moyenne pondérée des voisins selon leur proximité spatiale et couleur :

- **Spatial** : la distance des voisins influence la moyenne (= filtre Gaussien).
  Plus un pixel est proche spatialement, plus il contribue.
- **Couleur** : la différence en intensité des voisins influence la moyenne.
  Plus un pixel est de même intensité/couleur, plus il contribue.
  → Préservation des contours

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

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

Pondération couleur

Pondération spatiale

## Résultats

Image initiale

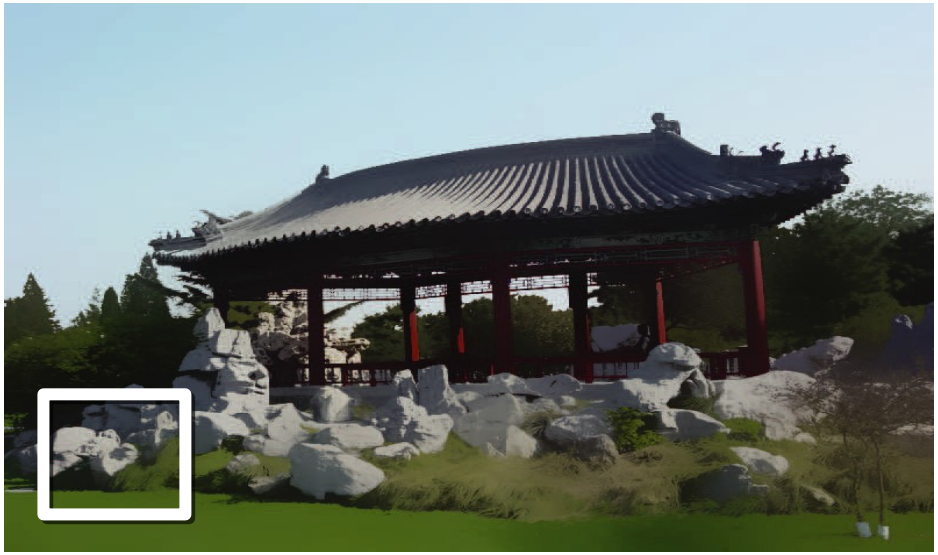


Image filtrée

## Résultats

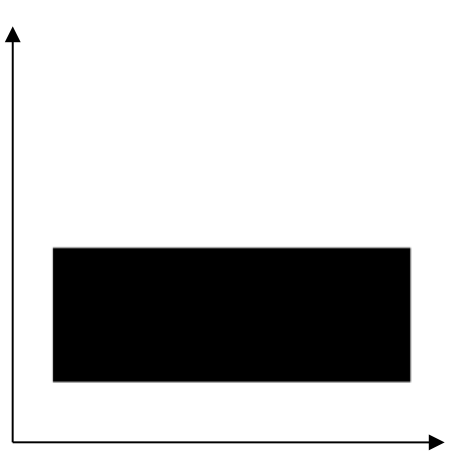
Image initiale

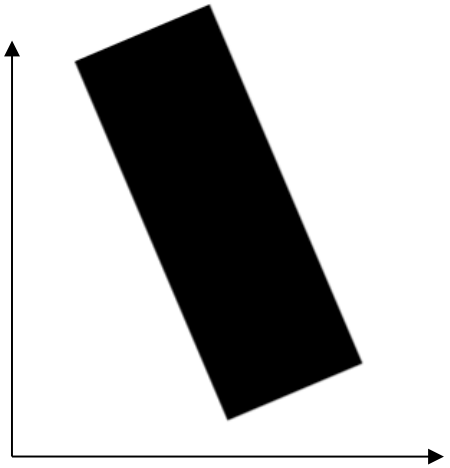
Image filtrée

## Rotation, translation, similarité

$$\vec{p}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \xrightarrow{\text{f(.)}} \vec{p}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

$$\vec{p}_2 = \vec{t} + SR\vec{p}_1 \qquad R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \vec{s} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}$$





## D'un point vue numérique

- Pas de bijection entre l'espace de départ et d'arrivée

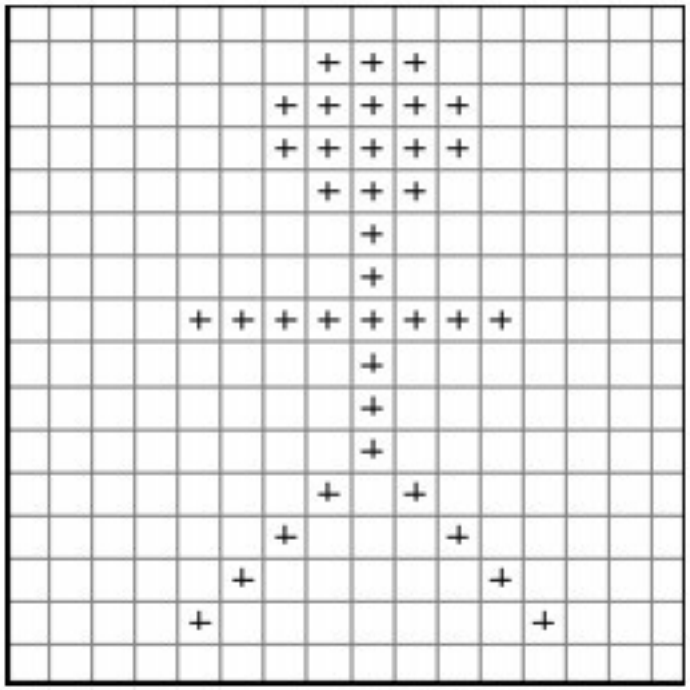


Image de départ

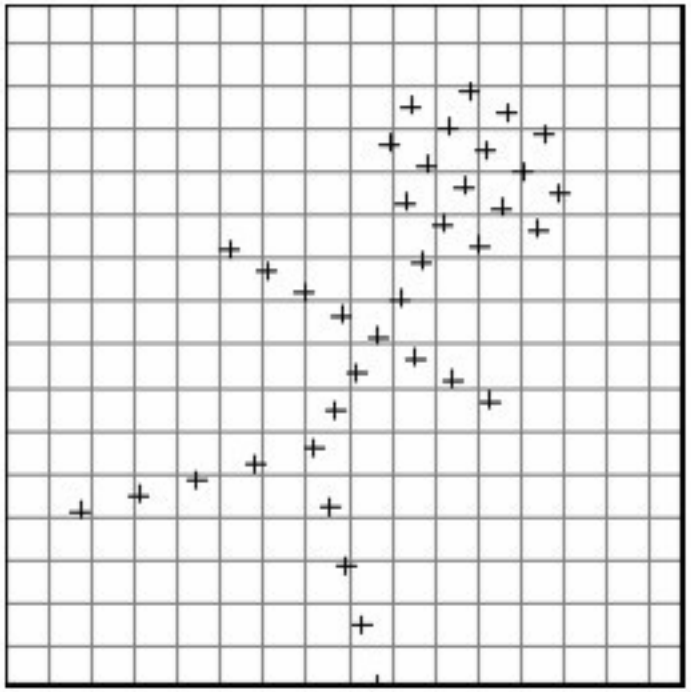


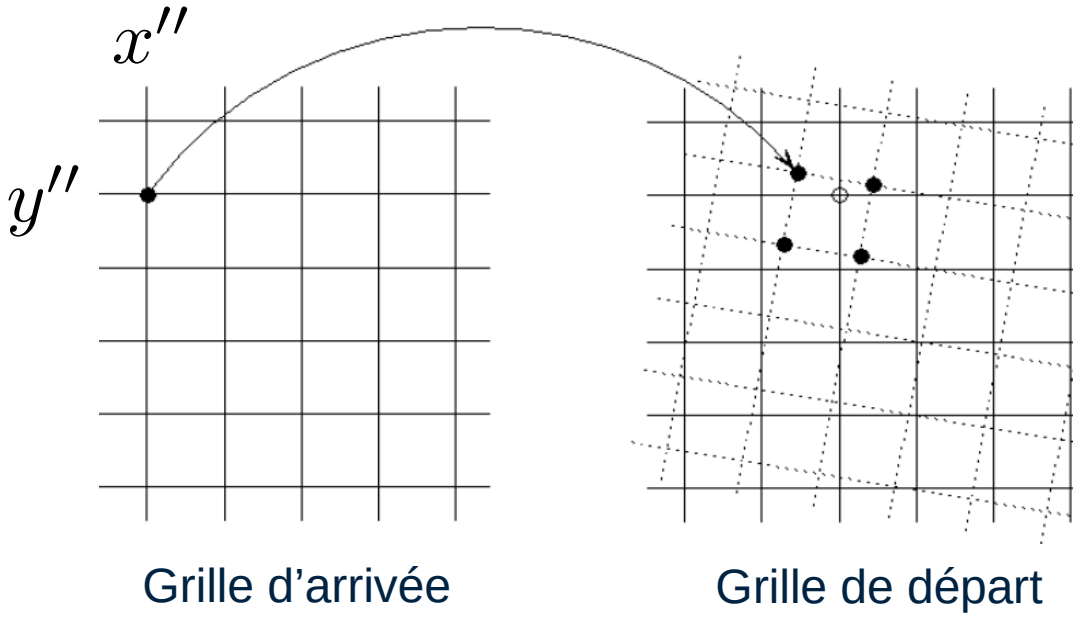
Image d'arrivée

## D'un point vue numérique

- Pas de bijection entre l'espace de départ et d'arrivée

  $\rightarrow$ Application de la transformation inverse

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathrm{T}^{-1}\left(\begin{bmatrix} x'' \\ y'' \end{bmatrix}\right)$$

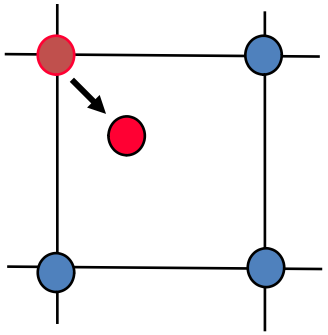


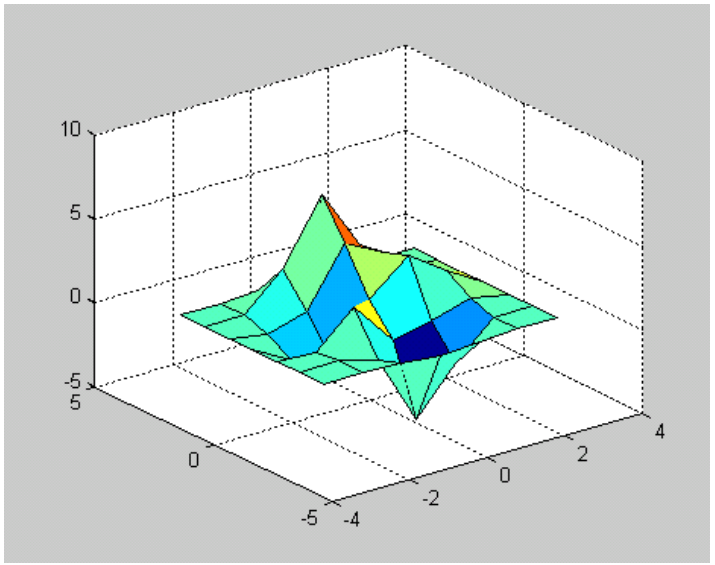
## Méthode d'interpolation

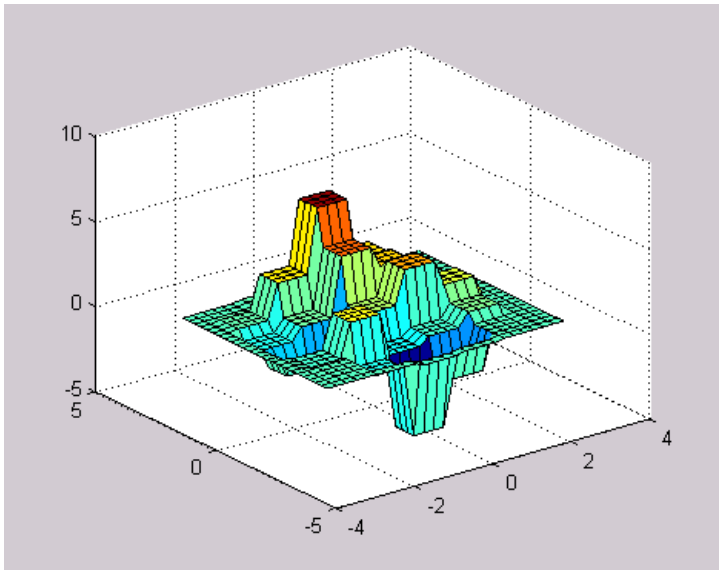
- Plus proche voisin : sélection du pixel le plus proche

$$\hat{I}(\bullet) = I(\bullet)$$

$$\hat{I}(x', y') = I(round(x'), round(y'))$$





## Méthode d'interpolation

- Bilinéaire : moyenne pondérée des 4 pixels les plus proches

$$I(x', y') = b(1-a)I(y+1, x) + baI(y+1, x+1) \\ +(1-b)(1-a)I(y, x) + a(1-b)I(y, x+1)$$

$$\begin{cases} a = x' - x \\ b = y' - y \end{cases}$$

$$\begin{cases} x = floor(x') \\ y = floor(y') \end{cases}$$