

Module : Informatique

Classes: 2^{ème} Années MP – PC – T

PARTIE III: Simulation numérique

Thème 3: Traitement d'Images

Enseignant : Dr. Khemaies GHALI (ghali_khemaies@yahoo.fr)

Page Facebook: <https://www.facebook.com/groups/ipein.info/>

Année universitaire 2016/2017

IPEIN 2016/2017

2ème Année - Informatique - Thème 3



1/40

Plan

1. Introduction
2. Images matricielles
3. Operations sur les images
4. Exemple
5. Exercice
6. Transformations géométriques élémentaires
7. Compression
8. Détection de contour
9. Convolution
10. Dilatation

IPEIN 2016/2017

2ème Année - Informatique - Thème 3



2/40

Introduction (1/3)

- ❑ Qu'est-ce que le traitement d'images ?
 - Au sens large, c'est la manipulation et la transformation d'images numériques pour obtenir d'autres images,
 - Extraire des informations d'intérêt des images (réduire l'information): position d'un objet, nombre de personnes, etc.
- ❑ On désigne par "image numérique" toute image stockée sous forme d'un fichier sur un support informatique,
 - Exp. Ordinateur, clé USB, téléphone portable, Nintendo DS, iPod...
- ❑ On distingue deux types d'images numériques :
 - **Image vectorielle (vector format)** : les données sont des formes géométriques simples (ensemble de segments de droites):

IPEIN 2016/2017

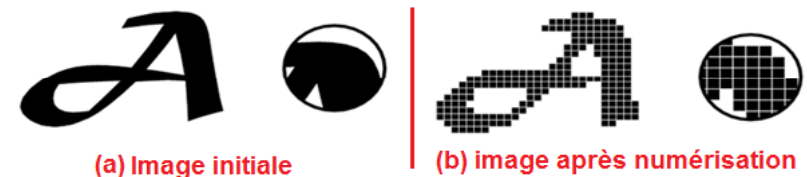
2ème Année - Informatique - Thème 3



3/40

Introduction (2/3)

- ❖ Exp.: Caractères, logos, cartes, topographie, plans, etc.
- ❖ Formats de fichiers vectoriels : PostScript, PDF, SVG, etc.
- **Image matricielle (bitmap=carte de bits)**: matrice de petits carrés élémentaires appelés pixels.
- ❖ Formats de fichiers matriciels : BMP, GIF, PNG, JPEG, etc.



- ❑ Dans ce cours, on s'intéresse aux images matricielles.

IPEIN 2016/2017

2ème Année - Informatique - Thème 3



4/40

Introduction (3/3)

- ❑ Une image matricielle est caractérisée par:
 - sa **définition** c'est-à-dire le nombre de pixels utilisés
 - sa **résolution** c'est-à-dire le nombre de pixels par unité de longueur et s'exprime alors en pixels par pouce (dpi : dot per inch ou ppp : point par pouce).
 - La quantification des couleurs ou la couleur donnée à un pixel exprimée en bits par pixel (bpp) :
 - ❖ 1bit (2 couleurs: noir et blanc),
 - ❖ 8bits (niveaux de gris: entre 0 et 255 soit 256 couleurs),
 - ❖ 24 bits (modèle RVB: rouge vert et bleu chacun codé sur 1 octet), etc.

Les applications T.I.

- ❑ Grâce aux capteurs CCD bon marché, il est de nos jours possible de produire de gros volumes d'images à faible coût.
- ❑ De nombreuses applications reposent donc sur l'extraction d'information à partir d'images, il s'agit d'un domaine en plein essor.
- ❑ Parmi les diverses applications, on peut citer :
 - Diagnostic médical (présence de tumeurs, forme des cellules, etc.)
 - Contrôle industriel (détection de défauts)
 - Reconnaissance automatique (visages sur Facebook, etc.)
 - Extraction de données scientifiques à partir d'images dans une expérience scientifique (position d'une bulle, d'une particule, ...)

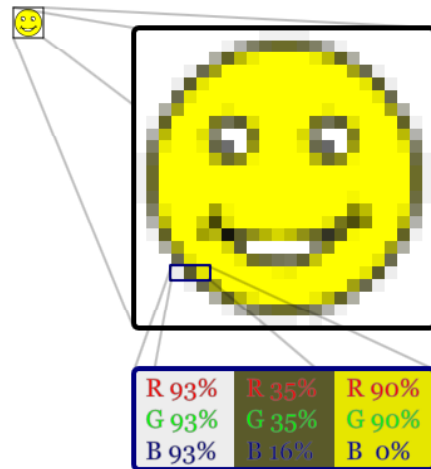
Images matricielles (1/2)

- ❑ Les images matricielles peuvent facilement être créées et stockées dans un tableau de pixels, la lecture et l'écriture d'un pixel sont aisées.
 - Image = matrice de pixels: photos, films.
 - = tableau de lignes
 - = tableau de tableaux
 - Pixel = petit carré = image élémentaire:
 - = 0 ou 1
 - ∈ [0; 1]: intensité de gris
 - ∈ [0; 255]: intensité de gris codée sur un octet
 - = triplet (r, v, b) ∈ [0; 255]³: composante rouge, vert, bleu (environ 16 millions de couleurs).
 - (FF 00 00)₁₆ = rouge, codage en hexadécimal

Images matricielles (2/2)

- ❑ Par contre, les fichiers peuvent être très gros,
 - D'où la nécessité de compression ou réduction de taille.
- ❑ Pour les traitements basiques sur les images nous avons besoin:
 - Des fonctions de la bibliothèque externe « PIL » (Python Imaging Library) ou « pillow » pour la version 3.
 - Aussi les fonctions de « numpy » pour les calculs matriciels
 - Et la bibliothèque « scipy » avec ses sous bibliothèques : « ndimage » et « misc » pour le traitement d'images.

Codage des couleurs



Les formats d'images matricielles

- ❑ Format sans perte « Bitmap » :
 - sans compression: .bmp
 - avec compression: .png .gif
- ❑ Format avec perte:
 - compressé: .jpg
 - la perte peut être nulle...



	IPEIN_30emeAns.jpg	Type : Fichier JPG Dimensions : 800 x 480	Prise de vue : 10/01/2017 11:56 Taille : 213 Ko
	IPEIN_30emeAns.png	Type : Image PNG Dimensions : 800 x 480	Taille : 853 Ko
	IPEIN_30emeAns_24bits.bmp	Type : Fichier BMP Dimensions : 800 x 480	Taille : 1,09 Mo
	IPEIN_30emeAns_NB.bmp	Type : Fichier BMP Dimensions : 800 x 480	Taille : 46,9 Ko

En python: Images matricielles (1/2)

```

1. >>> import os
2. >>> import numpy as np
3. >>> import matplotlib.pyplot as plt
4. >>> os.chdir("c:/")
5. >>> os.getcwd()
6. 'c:\\'
7. >>> os.chdir("images")
8. >>> os.getcwd()
9. 'c:\\images'
10. >>> os.listdir()
11. ['IPEIN_30emeAns.jpg',
    'IPEIN_30emeAns.png',
    'IPEIN_30emeAns_24bits.bmp',
    'IPEIN_30emeAns_NB.bmp']
12. >>> img = plt.imread('IPEIN_30emeAns_24bits.bmp')
13. >>> type(img)
14. <class 'numpy.ndarray'>

15. >>> img.ndim
16. 3
17. >>> img.size
18. 1152000
19. >>> 480*800*3
20. 1152000
21. >>>>> img.shape
22. (480, 800, 3)
23. >>> img
24. array([[ 11,  17,   3],
        [ 15,  18,   7],
        [ 14,  13,   9],
        ...,
        [ 41, 101, 164],
        [ 39,  99, 162],
        [ 38,  98, 161]],
        dtype=uint8)
    
```

En python: Images matricielles (2/2)

```

35. >>> im = plt.imshow(img)
36. >>> plt.show()
    
```

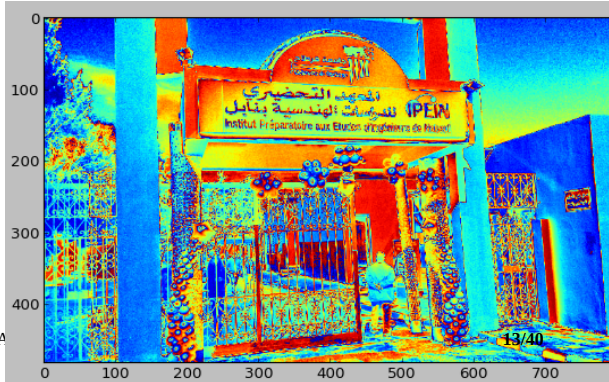


Conversion en niveaux de gris (1/2)

```
1. >>> img_ng = (img[:, :, 0] + img[:, :, 1] + img[:, :, 2]) / 3.
2. >>> img_ng.ndim; img_ng.shape; img_ng
3. 2
4. (480, 800)
5. array([[ 10.33333333,  13.33333333, ..., 13.66666667],
6.        [ 26.33333333,  18., ..., 14.66666667],
7.        ...,
8.        [ 21.33333333,  23.33333333, ..., 79.66666667]])
9. >>> im = plt.imshow(img_ng)
10. >>> plt.show()
```

NB:
Dans ce cas, il manque la table des niveaux des gris!

IPEIN 2016/2017 2ème A



Conversion en niveaux de gris (2/2)

```
1. >>> img_ng = (img[:, :, 0] + img[:, :, 1] + img[:, :, 2]) / 3.
2. >>> im = plt.imshow(img_ng)
3. >>> im.set_cmap('gray') #Ajout de la table des gris
4. >>> type(im) #'gray_r' inverse les couleurs
5. <class 'matplotlib.image.AxesImage'>
6. >>> im.write_png("IPEIN_30emeAns_ng.png", True)
7. >>> plt.show()
```

IPEIN 2016/2017 2ème Année



Operations sur les images (1/3)

□ Addition:

```
1. >>> I = img_ng + img_ng
2. >>> im = plt.imshow(np.concatenate([img_ng, I]))
3. >>> im.set_cmap('gray')
4. >>> plt.show()
```



IPEIN 2016/2017

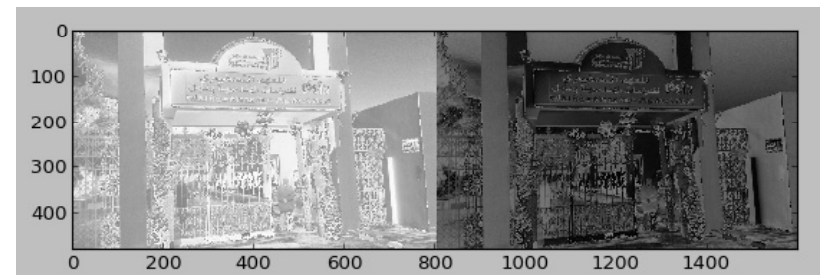
2ème Année - Informatique - Thème 3

15/40

Operations sur les images (2/3)

□ Soustraction:

```
1. >>> I = img_ng + img_ng
2. >>> im = plt.imshow(np.concatenate([img_ng, img_ng - I],
3. >>> im.set_cmap('gray')
4. >>> plt.show()
```



IPEIN 2016/2017

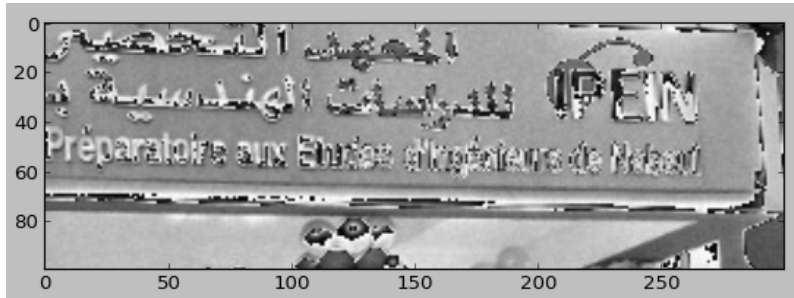
2ème Année - Informatique - Thème 3

16/40

Operations sur les images (3/3)

❑ Sélection:

```
1. >>> J = img_ng[100:200, 300:600]
2. >>> im = plt.imshow(J)
3. >>> im.set_cmap('gray')
4. >>> plt.show()
```



Le module « scikit-image » (1/2)

- ❑ Le module scikit-image « <http://scikit-image.org/> » est le module principal de Python scientifique pour le traitement d'images.
- ❑ Il est prévu pour fonctionner avec des tableaux **numpy**,
 - ce qui permet d'utiliser facilement le **scikit-image** en même temps que les autres modules de calcul scientifique:
- ❑ Pour installer ce module avec Python 3.4:
 - Télécharger le fichier «scikit_image-0.13.0-cp34-cp34m-win32.whl » à partir du site suivant:
<http://www.lfd.uci.edu/~gohlke/pythonlibs/#scikit-image>
 - Lancer la commande en mode commande MS-DOS:
 - > `pip install scikit_image-0.13.0-cp34-cp34m-win32.whl`

Le module « scikit-image » (2/2)

```
C:\Users\ghali\Downloads>pip install scikit_image-0.13.0-cp34-cp34m-win32.whl
Processing c:\users\ghali\downloads\scikit_image-0.13.0-cp34-cp34m-win32.whl
Collecting networkx>=1.8 (from scikit-image==0.13.0)
  Downloading networkx-1.11-py2.py3-none-any.whl (1.3MB)
  100% |#####| 1.3MB 189kB/s
Collecting PyWavelets>=0.4.0 (from scikit-image==0.13.0)
  Downloading PyWavelets-0.5.2-cp34-none-win32.whl (4.1MB)
  100% |#####| 4.1MB 116kB/s
Requirement already satisfied: six>=1.7.3 in c:\python34\lib\site-packages (from
scikit-image==0.13.0)
Collecting pillow>=2.1.0 (from scikit-image==0.13.0)
  Downloading Pillow-4.1.0-cp34-cp34m-win32.whl (1.2MB)
  100% |#####| 1.2MB 263kB/s
Collecting decorator>=3.4.0 (from networkx>=1.8->scikit-image==0.13.0)
  Downloading decorator-4.0.11-py2.py3-none-any.whl
Requirement already satisfied: numpy>=1.9.1 in c:\python34\lib\site-packages (fr
om PyWavelets>=0.4.0->scikit-image==0.13.0)
Collecting olefile (from pillow>=2.1.0->scikit-image==0.13.0)
  Downloading olefile-0.44.zip (74kB)
  100% |#####| 81kB 288kB/s
Installing collected packages: decorator, networkx, PyWavelets, olefile, pillow,
scikit-image
Running setup.py install for olefile ... done
Successfully installed PyWavelets-0.5.2 decorator-4.0.11 networkx-1.11 olefile-0
.44 pillow-4.1.0 scikit-image-0.13.0
C:\Users\ghali\Downloads>
```

Exemple (1/3)

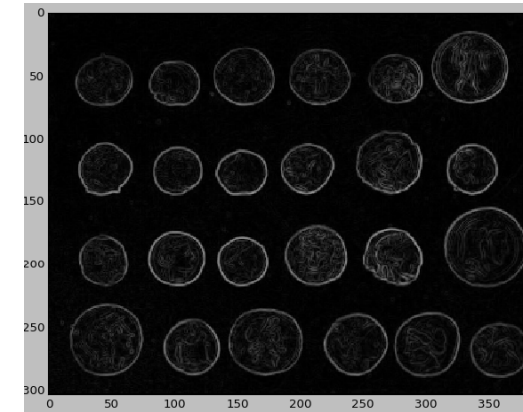
```
1. #On commence par importer les modules dont on
2. #aura besoin pour le traitement d'images:
3. >>> import numpy as np
4. >>> from skimage import data, io, filters
5. >>> from scipy import ndimage
6. >>> import matplotlib.pyplot as plt
7. #On charge une image comme un tableau numpy
8. #C:\Python34\Lib\site-packages\skimage\data\coins.png
9. >>> image_array = data.coins() #ou un array du NumPy
10. >>> image_array.dtype
11. dtype('uint8')
12. >>> image_array.shape
13. (303, 384)
14. # Affichage
15. >>> io.imshow(image_array)
16. >>> io.show()
17. # ou plt.imshow(image_array, cmap='gray')
18. # plt.show()
```

Exemple (2/3)



```
1. >>> img_sob = filters.sobel(image_array)
2. >>> io.imshow(img_sob)
3. >>> io.show()
```

Exemple (3/3)



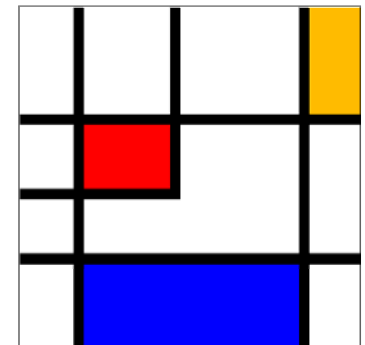
- ❑ Il existe d'autres modules de Python spécialisés pour le traitement d'image, par exemple: OpenCV, Mahotas, Pink

Changement du format

```
1. #chargement de l'image en couleur
2. >>> img_couleur = io.imread('IPEIN_30emeAns_24bits.bmp')
3. >>> img_couleur.shape; img_couleur.dtype
4. (480, 800, 3)
5. dtype('uint8')
6. #chargement de l'image en Noir et Blanc
7. >>> img_ng =
  io.imread('IPEIN_30emeAns_24bits.bmp', as_grey=True)
8. >>> img_ng.shape; img_ng.dtype
9. (480, 800)
10. dtype('float64')
11. #conversion en niveau de gris sur 8 bits
12. >>> img_ng_8bits = skimage.img_as_ubyte(img_ng)
13. >>> img_ng_8bits.shape; img_ng_8bits.dtype
14. (480, 800)
15. dtype('uint8')
16. >>> io.imshow(img_ng_8bits)
17. >>> io.show()
18. #enregistrement dans un fichier
19. >>> io.imsave('IPEIN_30emeAns_ng8bits.png', img_ng_8bits)
```

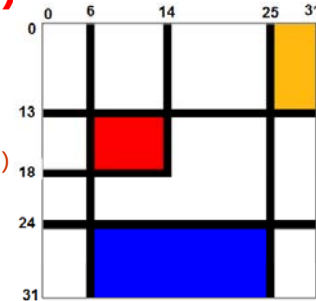
Exercice (1/3)

- ❑ Cet exercice a pour but de s'entraîner :
 - 1) au slicing des tableaux numpy et
 - 2) à la représentation d'images couleurs sous la forme de tableaux numpy à trois canaux.
- ❑ Réaliser une image ressemblant à l'image ci-dessus, sous la forme d'un tableau numpy.
 - ❖ On pourra par exemple prendre l'image de forme 32x32, et de type np.uint8 (entiers de 0 à 255).
- ❑ Les trois canaux de couleur sont dans l'ordre R, V, B (rouge vert bleu).
 - ❖ Le jaune correspond au triplet (R, V, B) = (255, 185, 15)
 - ❖ Le noir = (0,0,0)
 - ❖ Le blanc = (255,255,255)



Exercice (2/3)

```
1. #Solution n°1 avec module skimage
2. import numpy as np
3. from skimage import io
4. #img = np.array([265 for i in
   range(32*32*3)], dtype=np.dtype('i1'))
5. img = np.array([255 for i in
   range(32*32*3)], dtype='uint8')
6. img = img.reshape(32, 32, 3)
7. img[13, :, :] = 0; img[24, :, :] = 0
8. img[18, 0:15, :] = 0; img[:, 6, :] = 0
9. img[:, 25, :] = 0; img[0:18, 14, :] = 0
10. img[14:18, 7:14, 1:3] = 0 #la zone rouge
11. img[25:32, 7:25, 0:2] = 0 #la zone bleu
12. img[0:13, 26:32, :] = (255, 185, 15) #la zone jaune
13. io.imshow(img)
14. io.imsave('Exe_sol1.png', img)
15. io.show()
```



Exercice (3/3)

```
1. #Solution n°2 avec module matplotlib
2. import matplotlib.pyplot as plt
3. import numpy as np
4. img = 255 * np.ones((32, 32, 3), dtype=np.uint8)
5. img[10] = 0; img[23] = 0
6. img[:, 5] = 0; img[:, 26] = 0
7. img[11:17, 6:14, 1:] = 0
8. img[:, 18, 14] = 0; img[17, :14] = 0
9. img[27:, 6:26, 0] = 0; img[27:, 6:26, 1] = 0
10. img[24:, 6:26, 1] = 0; img[24:, 6:26, 0] = 0
11. img[:, 10, 27:, 1] = 185; img[:, 10, 27:, 2] = 15
12. img[:, 10, 27:, 0] = 255
13. plt.figure(figsize=(6, 6))
14. plt.imshow(img, interpolation='nearest')
15. plt.axis('off')
16. plt.imsave('Exe_sol2.png', img)
17. plt.show()
```

Accès à la grille des pixels (1/3)

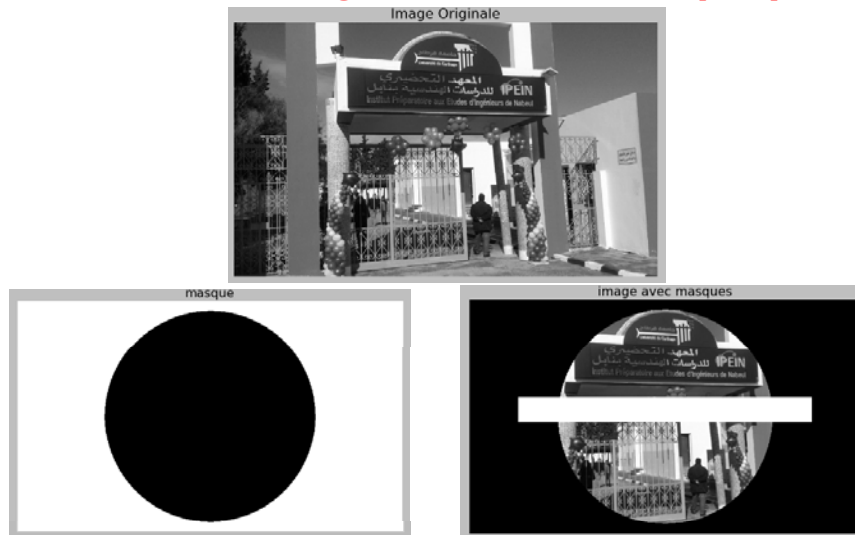
- ❑ Il est possible d'avoir facilement accès à la grille des pixels de l'image, sous forme de tableau, et de faire des traitements matriciels directs sur les images.
- ❑ Le code suivant décrit comment :
 - appliquer un masque binaire sur une image donnée, à partir de la définition de ce masque sous forme d'une formule calculée sur la grille des pixels.
 - accéder à une série de pixels pour afficher des valeurs prédéfinies

```
1. #Fichier Masque_Circulaire.py
2. from scipy import misc
3. import matplotlib.pyplot as plt
4. import numpy as np
```

Accès à la grille des pixels (2/3)

```
1. #ouverture d'une image couleur sur le disque
2. j = misc.imread('images/IPEIN_30emeAns_ng8bits.png')
3. L,C = j.shape #récupération des dimensions
4. #recuperation d'une grille ayant la dimension de
   l'image
5. X, Y = np.ogrid[0:L,0:C]
6. #création d'un masque circulaire
7. masque = (X-L/2)**2 + (Y-C/2)**2 > L*C/8
8. plt.imshow(masque, cmap=plt.cm.gray)
9. plt.title('masque'); plt.axis('off'); plt.show()
10. j[masque]=0 #Application du masque sur l'image
11. j[200:250,100:700]=255 #Accès à une zone de la
   grille
12. #La palette est mise à la palette en niveaux de gris
13. plt.imshow(j, cmap=plt.cm.gray)
14. plt.title('image avec masques')
15. plt.axis('off'); plt.show()
```

Accès à la grille des pixels (3/3)



IPEIN 2016/2017

2ème Année - Informatique - Thème 3

29/40

Eclaircir, assombrir une image (1/2)

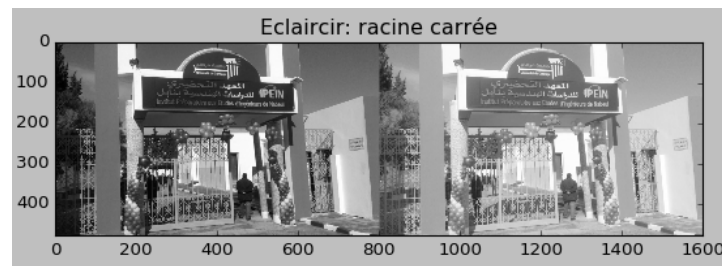
```
1. #Fichier Eclaircir_Assombrir.py
2. from scipy import misc;
3. import matplotlib.pyplot as plt;
4. import numpy as np
5. img = misc.imread('images/IPEIN_30emeAns_ng8bits.png')
6. img_reel = img/img.max() #conversion pixels de int8 en
   reel[0,1]
7. img_sqrt = np.sqrt(img_reel) #ou img_reel**(1/2)
8. plt.title('Eclaircir: racine carrée')
9. plt.imshow(np.concatenate([img_reel,img_sqrt],True),cmap=plt.cm.gray)
10. plt.show()
11. img_square = np.square(img_reel) #ou img_reel**2
12. plt.title('Assombrir: carré')
13. plt.imshow(np.concatenate([img_reel,img_square],True),cmap=plt.cm.gray)
14. plt.show()
```

IPEIN 2016/2017

2ème Année - Informatique - Thème 3

30/40

Eclaircir, assombrir une image (2/2)



IPEIN 2016/2017

2ème Année - Informatique - Thème 3

31/40

Transformations géométriques élémentaires

- ❑ Il est très aisé d'appliquer des transformations géométriques élémentaires (linéaires) sur des images.
- ❑ Les programmes suivants présentent quelques exemples :
 - région d'intérêt, rotations, symétrie, zoom
- ❑ les fonctions utilisées proviennent à la fois de numpy et de scipy:
 - ndimage, ensemble de fonctions pour le traitement d'images multidimensionnelles

```
1. #Fichier Transform_geom.py
2. from scipy import misc, ndimage
3. import matplotlib.pyplot as plt;
4. import numpy as np
5. img = misc.imread('images/IPEIN_30emeAns_ng8bits.png')
```

IPEIN 2016/2017

2ème Année - Informatique - Thème 3

32/40

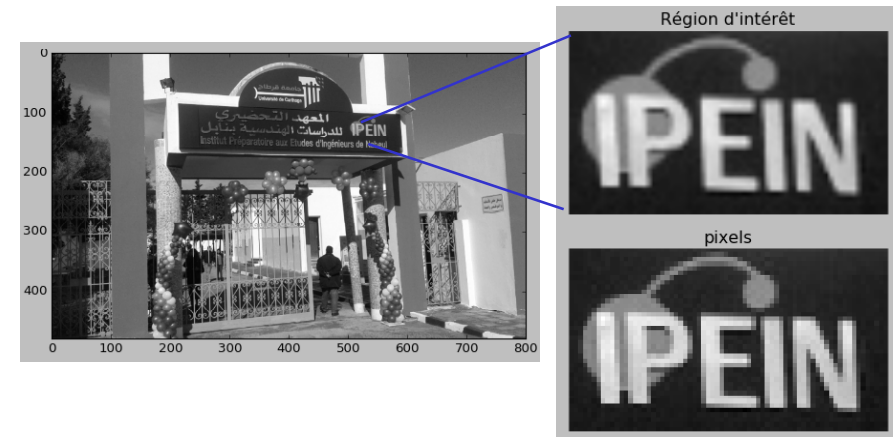
Zone d'intérêt (1/2)

```

1. #Définition d'une zone d'intérêt
2. f, tab = plt.subplots(2, sharey=True)
3. logo = img[105:145, 500:570]
4. tab[0].imshow(logo, cmap=plt.cm.gray)
5. tab[0].set_title("Région d'intérêt")
6. tab[0].axis('off')
7. #visualisation précise de la zone d'intérêt
8. tab[1].imshow(logo, cmap=plt.cm.gray, interpolat
   ion='nearest')
9. tab[1].set_title('pixels')
10. tab[1].axis('off')
11. plt.show()

```

Zone d'intérêt (2/2)



Symétrie & Rotations (1/3)

```

1. #symétrie haut-bas
2. f, ax = plt.subplots(2, 2)
3. ax[0,0].imshow(img, cmap=plt.cm.gray)
4. ax[0,0].axis('off')
5. ax[0,0].set_title('Originale')
6. img_flip = np.flipud(img) #Flip array in the
   up/down
7. #img_flip = np.fliplr(img) #Flip array in the
   left/right
8. ax[0,1].imshow(img_flip, cmap=plt.cm.gray)
9. ax[0,1].axis('off')
10. ax[0,1].set_title('Symetrie')

```

Symétrie & Rotations (2/3)

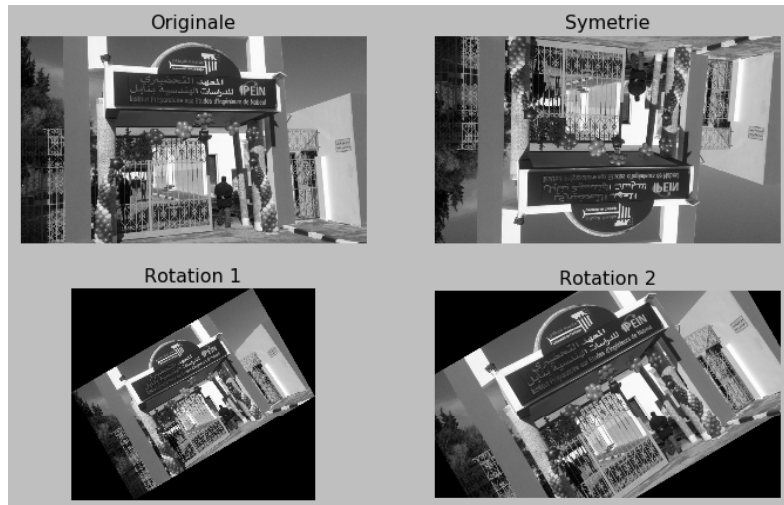
```

1. #rotations
2. img_tournee1 = ndimage.rotate(img, 30)
3. ax[1,0].imshow(img_tournee1, cmap=plt.cm.gray)
4. ax[1,0].axis('off')
5. ax[1,0].set_title('Rotation 1')

6. img_tournee2 =
   ndimage.rotate(img, 30, reshape=False)
7. ax[1,1].imshow(img_tournee2, cmap=plt.cm.gray)
8. ax[1,1].axis('off')
9. ax[1,1].set_title('Rotation 2')

```

Symétrie & Rotations (3/3)



IPEIN 2016/2017

2ème Année - Informatique - Thème 3

37/40

Zoom d'image

1. #zoom avec différents facteurs sur chaque axe
2. `img_zoom = ndimage.zoom(img, [3,5])`
3. `plt.imshow(img_zoom, cmap=plt.cm.gray)`
4. `plt.title('Homothetie')`
5. `plt.axis('off')`
6. `plt.show()`



IPEIN 2016/2017

2ème Année - Informatique - Thème 3

38/40

Annexe 1: numpy.dtype

- ❑ The first character specifies the kind of data and the remaining characters specify the number of bytes per item, except for Unicode, where it is interpreted as the number of characters.
- ❑ The item size must correspond to an existing type, or an error will be raised.
- ❑ The supported kinds are:

'b' --> boolean	'i'--> (signed) integer	'u'--> unsigned integer
'f'--> floating-point	'c'--> complex-floating point	
'm'--> timedelta	'M'--> datetime	'O'--> (Python) objects
'S', 'a'--> (byte)-string	'U' --> Unicode	'V'--> raw data (void)

❑ Example:

1. `>>> dt = np.dtype('i4') # 32-bit signed integer`
2. `>>> dt = np.dtype('f8') # 64-bit floating-point number`
3. `>>> dt = np.dtype('c16') # 128-bit complex floating-point number`
4. `>>> dt = np.dtype('a25') # 25-character string`

IPEIN 2016/2017

2ème Année - Informatique - Thème 3

39/40

Références bibliographiques

- [1] http://python-prepa.github.io/ateliers/image_tuto.html
- [2] skimage: module Python pour la traitement d'images
<http://scikit-image.org/>
- [3] OpenCV: Computer Vision with Python
http://docs.opencv.org/3.1.0/d6/d00/tutorial_py_root.html
- [4] Mahotas: set of functions for image processing and computer vision in Python <http://luispedro.org/software/mahotas/>
- [5] PINK Image Processing Library
<https://pinkhq.com/joomla/index.php>
- [6] Python Imaging Library (PIL)
<http://www.pythonware.com/products/pil/>

IPEIN 2016/2017

2ème Année - Informatique - Thème 3

40/40