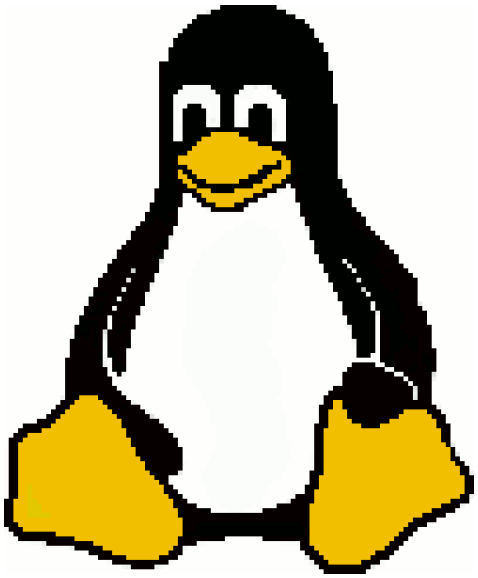
Lire et écrire des images numériques avec Python



Objectifs

A la fin de la séquence d'enseignement les élèves doivent être capable :

- d'ouvrir une image et d'accéder à chaque pixel
- de modifier une image
- d'afficher une image à l'écran et de l'enregistrer

Table des matières

1 Différents types d'images

1.1 Images vectorielles et images matricielles

On distingue deux grandes familles d'images (numériques) : les images vectorielles et les images matricielles.

Les images vectorielles sont (re-)construites à partir d'équations mathématiques et basées sur des formes géométriques. Zoomer ou dé-zoomer sur une image vectorielle relance le calcul pour affinée la représentation en fonction de la résolution de l'écran. Les formes paressent lissent. Le format vectoriel vient de la construction de l'image étape par étape à partir de formes élémentaires. Les polices sont aujourd'hui au format vectoriel. Le format `.svg` est un format vectoriel.

Les images matricielles, qui font l'objet de ce cours, sont des tableaux 2D de points qu'on appelle **pixels**. Zoomer sur une image matricielle met en évidence la discrétisation des contours. Ce format est particulièrement adapté à la photographie ou au dessin. La première police TEXétait au format matriciel. Les formats classiques d'images matriciels sont `.bmp`, `.jpg`, `.png`...

1.2 Tableau de pixels

Une image matricielle est un tableau de pixels rectangulaire. Sa taille est définie par le produit du nombre de pixels en largeur et le nombre de pixels en hauteur (par exemple 1920×1080 - format 16:9), ce qui lui donne sa taille en pixels (avec l'exemple 2 073 600 pixels soit environ 2 Mpx).

Chaque pixel est une liste de 3 entiers contenant les niveaux de rouge, vert et bleu pour une image au format RGB (ou RVB en français). On peut ajouter un entier supplémentaire pour coder le canal α ou niveau de transparence (format RGBA). L'image de l'exemple précédent a une taille d'environ 6 Mo au format RGB sans compression en utilisant un octet pour coder l'entier.

Avec un codage sur 24 bits, le triplet (ou 3-uplet) (255, 0, 0) code le rouge, le triplet (0, 255, 0) le vert et le bleu est codé par le triplet (0, 0, 255). On utilise la synthèse additive de couleurs. Ainsi le blanc est défini par le triplet (255, 255, 255) et le noir par le triplet (0, 0, 0). On utilise aussi une écriture hexadécimale pour coder les triplet. Le blanc devient `FFFFFF` et par exemple le bleu `0000FF`.

On appelle résolution le nombre de pixels par unité de longueur. Hélas, on utilise couramment le `dpi` (dot per inch) correspondant au nombre de pixels par pouce (**Rappel** 1 in = 2,54 cm). Pour une bonne impression papier, une résolution de 300 dpi est préconisée.

1.3 Images brutes et versions compressées

Le format `bmp` permet de stocker les images matricielles en 2 couleurs (1 bit), 16 couleurs (4 bits), 256 couleurs (8 bits), 65 536 couleurs (16 bits) et comme décrit dans la partie précédente en 16,8 millions de couleurs (3x8 bits). L'image n'est généralement pas compressée. Pour ce cours, c'est cette structure d'image qui sera utilisée.

Pour réduire la taille des fichiers images, différentes méthodes de compressions sont utilisées : avec pertes (type `jpg`) ou sans pertes (type `png`). Ces méthodes ne sont pas au programme mais peuvent faire l'objet d'un DS, d'un DM ou d'une partie de franche rigolade...

2 Obtenir l'image sous forme de tableaux exploitables

2.1 Différentes méthodes d'ouverture du fichier image

Une image du lycée est enregistrée dans différents formats. Le but est d'obtenir dans une variable `image` l'image sous forme d'une liste de listes de listes. Le premier indice correspond à l'indice de la ligne, le deuxième à l'indice de la colonne, le troisième à la valeur du canal (R, G, B voir A).

2.1.1 Avec matplotlib.pyplot

Le code suivant :

```
import matplotlib.pyplot as plt

for ext in ['.jpg', '.jpeg', '.png', '.bmp', '.pdf']:
    try :
        image = plt.imread('carnot' + ext)
        print(ext)
        print(len(image), len(image[0]), len(image[0][0]))
        print(image[0][0])
    except :
        print("Problème avec l'extension" + ext)
```

produit ceci :

```
.jpg
306 408 3
[162 216 252]
.jpeg
306 408 3
[162 216 252]
```

```
.png
306 408 4
[0.63529414 0.84705883 0.99607843 1.
]
.bmp
306 408 4
[162 216 254 255]
Problème avec l'extension.pdf
```

Remarque : le format `png` conduit à coder le niveau des couleurs entre 0 et 1 et non sur un octet quand l'image est lue avec matplotlib.pyplot.

2.1.2 Avec pillow

Le code suivant :

```
import numpy as np
from PIL import Image as im

for ext in ['.jpg', '.jpeg', '.png', '.bmp', '.pdf']:
    try :
        img = im.open('carnot' + ext)
        image = np.array(img)
        print(ext)
        print(len(image), len(image[0]), len(image[0][0]))
        print(image[0][0])
    except :
        print("Problème avec l'extension" + ext)
```

produit ceci :

```
.jpg
306 408 3
[162 216 252]
.jpeg
306 408 3
[162 216 252]
```

```
.png
306 408 4
[162 216 254 255]
.bmp
306 408 4
[162 216 254 255]
Problème avec l'extension.pdf
```

2.2 Obtenir des informations sur une image

```
>>> image = plt.imread('carnot.bmp')
>>> print(image.shape, image.dtype)
(306, 408, 4) uint8
```

```
>>> img = im.open('carnot.png')
>>> print(img.format, img.size, img.mode)
PNG (408, 306) RGBA
```

```
>>> image = plt.imread('carnot.png')
>>> print(image.shape, image.dtype)
(306, 408, 4) float32
```

Rappel matplotlib.pyplot a un comportement différent avec les images `png`. Les canaux sont codés sur un flottant de 32bits compris entre 0 et 1.

3 Écrire, visualiser et enregistrer une image

3.1 Visualiser une image à partir d'un tableau exploitable

Pour visualiser la variable `image` définie dans la partie 2.1, on peut utiliser les codes suivants :

```
plt.imshow(image)
plt.show()
```

```
image_pil = im.fromarray(image)
image_pil.show()
```

3.2 Créer/modifier une image

La partie 3.1 montre qu'à partir d'une image stockée sous forme de liste de listes de listes telle que définie en partie 2.1 il est possible de la visualiser directement avec matplotlib.pyplot. Pour la lire avec pillow reste à la convertir en tableau numpy (si ce n'est pas déjà le cas) puis en image pillow.

On utilise donc les outils numpy pour créer des tableaux de listes à 3 éléments. Le plus rapide pour obtenir une image noire et une image blanche de `nl` lignes et `nc` colonnes est :

```
image_noire = np. zeros ((nl , nc,3) ,dtype =np. int8 )
image_blanche = np. ones ((nl , nc,3) ,dtype =np. int8 )*255
```

3.3 Sauvegarder une image

A partir d'une image `image` stockée sous forme de liste de listes de listes telle que définie en partie 2.1 il est possible de l'enregistrer sous différents formats :

```
plt.imsave('nomdelimage.jpg',  image)
```

```
image_pil = im.fromarray(image)
image_pil.save('nomdelimage.jpg')
```

Attention ! avec `PIL`, il faut adapter l'extension en fonction de mode de l'image : par exemple `jpg` pour RGB ; `png` pour RGBA.