Traitement d'image

Première approche

Pour ce tp vous aurez à télécharger l'image `vautour.png` disponible sur le site `pcsi.lamartin.fr`, section informatique.

Notion d'image matricielle

Nous allons ici nous intéresser à la manipulation d'**images matricielles** (par opposition aux images vectorielles).

Une image matricielle est constituée d'une grille de pixels. Avec la **représentation RVB** que nous allons utiliser ici, chaque pixel est représenté par 3 composantes sous la forme d'un triplet (r, v, b).

Chacune de ces composantes représente respectivement la quantité de rouge, de vert ou de bleu du pixel. Plus la valeur est élevée, plus la couleur est présente.

Pour nous chaque composante sera un nombre flottant entre 0 et 1.

Par exemple $(0.5, 0.25, 1)$ représente 50% de rouge, 25% de vert et 100% de bleu.

Le triplet $(0, 0, 0)$ représente un pixel complètement noir. En revanche si les trois composantes sont maximales, le pixel sera blanc (ceci correspond à la synthèse *additive* des couleurs).

I. Les modules python utilisés

A. Utilisation de numpy

Pour représenter les grilles de pixels nous utiliserons des tableaux numpy.

Exercice 1.

Tester les commandes suivantes dans le shell (en affichant les variables au fur et à mesure...)

```
import numpy as np # ne pas oublier
hauteur = 2
largeur = 4
tab = np.zeros( (hauteur, largeur, 3) )
tab[0,2] = [3,5,8]
tab[1,2] = (4,5,6) # numpy accepte les tuples
```

```
(a,b,c) = tab[0,2]
```

```
dimensions = tab.shape
ha = dimensions[0] #hauteur
la = dimensions[1] #largeur
# pour nous dimensions[2] vaudra toujours 3
```

B. Gestion des images

Nous utiliserons ici les modules :

- `matplotlib.image` pour convertir image en tableau numpy (et vice-versa)
- `matplotlib.pyplot` pour gérer l'affichage des images

Votre script python commencera donc finalement par les importations suivantes

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

Voici les autres commandes dont vous aurez besoin

- `image = mpimg.imread('nomfichierimage.png')` convertit une image png existante en un tableau numpy `image` (le module `matplotlib.image` permet par défaut de ne manipuler *que des fichiers* PNG*)*.
- `pixel = image[x,y]` permet de récupérer le pixel de coordonnées (x, y) qui est donc un triplet. On peut aussi directement écrire `(r,v,b)= image[x,y]` (dépaquetage du triplet)
- Pour afficher avec python une image représentée par le tableau `image` on utilisera les commandes :

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

II. Le travail à faire

Toutes les fonctions demandées porteront sur des tableaux images. A vous de les tester avec l'image `vautour.png` et d'afficher le résultat obtenu.

Exercice 2. Découverte.

a. A l'aide des commandes décrites, transformez l'image `vautour.png` en un tableau numpy. Affichez le pour examiner sa structure.

(vous noterez au passage que les composantes sont des flottants de $[0, 1]$)

Utilisez ensuite les diverses commandes présentées sur ce tableau pour vous familiariser.

b. Créez une image de hauteur 100 et de largeur 300 complètement noire. Affichez-là.

Transformez la de sorte que le premier tiers soit uniformément rouge pur, le deuxième vert pur, le troisième bleu pur.

Vous pourrez utiliser le slicing sous la forme `image[:, a:b] = couleur` où `a` et `b` sont des entiers bien choisis et `couleur` est un triplet correspondant à la couleur voulue.

Exercice 3. Jeu avec les couleurs.

a. Écrire une fonction `negatif(image)` de paramètre un tableau image et qui renvoie un tableau où la valeur v de chaque couleur est remplacée par $1 - v$.

Remarque. Il est possible d'effectuer `1-pixel` sur un triplet pixel

Testez-là en affichant l'image obtenue.

b. Écrire une fonction `composante_bleue(image)` qui à partir d'un tableau image retourne un tableau où chaque pixel (r, v, b) est remplacé par $(0, 0, b)$.

Testez-là en affichant l'image obtenue.

Exercice 4. Conversion en niveaux de gris puis en noir et blanc. Un pixel est dit *gris* si les trois composantes ont même valeur : il est donc de la forme (g, g, g).

a. Écrire une fonction `niveaux_gris(image)` qui convertit le tableau image en niveaux de gris en utilisant la formule suivante $g = 0.21r + 0.72v + 0.07b$ qui tient compte de la luminosité de chaque couleur.

Testez-là en affichant l'image obtenue.

b. Vous pouvez ensuite convertir en noir et blanc. Il suffit de mettre en paramètre un seuil $s \in [0, 1]$: si $g > s$ on remplace le pixel par un pixel blanc, sinon par un pixel noir.

Exercice 5. Floutage d'une image.

Écrire une fonction `floutage(image)` qui retourne un nouveau tableau dont chaque pixel de coordonnées (x, y) est la moyenne des 9 pixels de `image` de coordonnées $(x - 1, y - 1); (x - 1, y); \ldots; (x + 1, y + 1)$.

Testez...

Exercice 6. Augmentation du contraste – correction gamma.

Pour améliorer le contraste, il suffit d'appliquer à chaque valeur de composante une fonction $f : [0, 1] \to [0, 1]$ bien choisie. A vous de tester avec $f(x) = \sqrt{x}$ puis $f(x) = \frac{1}{2}\left(1 + \sin(\pi\left(x - \frac{1}{2}\right)\right)$.

La correction gamma utilise $f(x) = x^\gamma$.

Exercice 7. Prise d'initiative.

Sans indication à vous de réfléchir comment effectuer les manipulations suivantes.

- Effet miroir (haut-bas ou gauche-droite à votre guise),
- diminution ou augmentation de la taille d'une image (pas si facile),
- juxtaposition d'images (pourquoi pas pour un effet pop art rudimentaire).