

TP2 Optionnel

Traitement d'images**1 – Présentation**

Ce TP consiste à implémenter 4 effets de base sur les images en Python, en s'appuyant sur le module Pillow.



Le résultat des effets est illustré sur Lena pour vous donner un résultat de référence, mais n'hésitez pas à tester vos effets sur vos propres images.

2 – Effets à implémenter

Toutes les fonctions réalisant un traitement d'image prennent une image en paramètre et renvoient une nouvelle image, correspondant au résultat de l'application d'un effet sur l'image fournie.

Par conséquent, les fonctions de traitement d'images que vous allez rendre doivent :

- Créer une nouvelle image, et en aucun cas modifier l'image fournie en argument
- Ne rien faire d'autre que créer l'image correspondant à l'effet demandé. En particulier, elle ne doit ni afficher, ni sauvegarder le résultat dans un fichier. Si vous voulez faire ceci pour vos tests, le faire **en dehors** des fonctions demandées.



En plus des fonctions imposées, il est possible (et même recommandé) d'implémenter toutes les fonctions utilitaires que vous trouverez pertinentes.

Négatif

Cet effet reproduit ce que l'on peut observer sur les pellicules photo argentiques avant qu'elles ne soient développées.

Calculer le *négatif* d'une image consiste à remplacer chaque composante x par $255 - x$, pour chaque pixel de l'image.

Par exemple :

- Le négatif d'un pixel noir (0, 0, 0) est un pixel blanc (255, 255, 255) et inversement.
- Le négatif d'un pixel jaune (255, 255, 0) est un pixel bleu (0, 0, 255) et inversement.

Implémentez une fonction :

```
def negatif(image) :
```

qui crée et renvoie le négatif de l'image donnée en argument.

Exemple d'utilisation :

Dans tous les exemples de ce sujet, on considère que la variable `lena` contient l'image illustrée ci-contre.



Appel à la fonction	Image contenue dans <code>lena_negatif</code>
<code>lena_negatif = negatif(lena)</code>	

Niveaux de gris

Transformer une image couleur en niveaux de gris consiste à remplacer chaque pixel (r, g, b) par (l, l, l) , où l est la luminosité du pixel. Un calcul basique de luminosité consiste à définir l comme la moyenne de r, g et b .


En appliquant ce calcul, un pixel rouge $(255, 0, 0)$ deviendrait $(85, 85, 85)$.

Implémentez une fonction :

```
def niveaux_gris(image) :
```

qui crée et renvoie une version en niveaux de gris de l'image donnée en argument.

Exemple d'utilisation (sur Lena) :

Appel à la fonction	Image contenue dans <code>lena_niveaux_gris</code>
<code>lena_niveaux_gris = niveaux_gris(lena)</code>	

Rotation

Implémentez une fonction :

```
def rotation90(image, sens_trigo) :
```

qui crée et renvoie une image qui est le résultat d'une rotation de 90° de l'image fournie en argument :

- Dans le sens trigonométrique si sens_trigo vaut **True**
- Dans le sens horaire sinon.

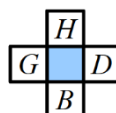
Exemples d'utilisation (sur Lena) :

Appel à la fonction	Image contenue dans lena_tournee
lena_tournee = rotation90(lena, True)	
lena_tournee = rotation90(lena, False)	

Détection de contours

Il existe de nombreux algorithmes de détection de contours. Leur principe est de détecter les fortes variations de luminosité dans les pixels.

Si on considère un pixel (illustré en bleu) avec le voisinage suivant :



On peut calculer la variation de luminosité sur les axes horizontaux et verticaux par :

$$v = |l_G - l_D| + |l_H - l_B|$$

où l_X désigne la luminosité du pixel X .

En se basant sur ce calcul, détecter les contours d'une image consiste à construire une nouvelle image selon l'algorithme suivant :

Pour chaque pixel p :

- si la variation de luminosité calculée en p est supérieure à un seuil s , alors p sera un pixel noir
- sinon, p sera un pixel blanc

Implémentez une fonction :

```
def contours(image, s) :
```

qui crée et renvoie une image contenant les contours de l'image fournie, basée l'application d'un seuil s sur la variation de luminosité.



Attention aux bords de l'image

Sur les bords de l'image, le voisinage d'un pixel peut être incomplet. Un pixel a en général 4 voisins (gauche, droite, haut, bas). Mais ce voisinage va être réduit sur les bords de l'image, par exemple :

Pixel au bord : 3 voisins au lieu de 4	Pixel en coin : 2 voisins au lieu de 4

Convention : s'il manque des voisins pour calculer la variation de luminosité dans une direction, on considérera que la **variation** est égale à 0.

Exemple d'utilisation (sur Lena) :

Appel à la fonction	Image contenue dans lena_contours
<pre>lena_contours = contours(lena, 50)</pre>	