Sommaire

Préambule

Ce TP utilise la bibliothèque Pillow : voir les instructions.

Avant de commencer, je vous suggère d'aller piocher une image de test pas trop grosse (disons au maximum 1024 pixels dans chaque dimension). À défaut en voilà deux, obtenues à partir d'une photo mise à disposition par Hans Stieglitz sur les *Wikimedia commons*, et soumise à la licence CC-BY-SA 3.0 :

- tigre.jpg
- tigrenb.png

Filtres d'image

Cette partie du TP concerne l'algorithmique de l'image. Plus précisément, on manipulera des images matricielles, c'est-à-dire représentées par des tableaux de pixels.

On utilise Pillow pour s'affranchir de la question des formats de fichiers.

Ouverture et enregistrement de fichiers d'image avec Pillow

Le bout de code suivant convertit le fichier `tigre.jpg` (au format JPEG) en `tigre.png` (au format PNG) :

Afficher/masquer les numéros de lignes

```
1 import PIL.Image as Image
2 im = Image.open(r'tigre.jpg')
3 im.save(r'tigre.png')
```

Essayez chez vous.

Informations sur une image

Dans un interpréteur Python, essayez :

Afficher/masquer les numéros de lignes

```
1 import PIL.Image as Image
2 im = Image.open(r'tigre.jpg')
```

puis essayez d'utiliser la documentation interne de python (par exemple via la fonction `dir` de Python ou la commande `help` de l'interpréteur) pour explorer les informations fournies par l'objet image.

Essayez par exemple d'obtenir sa taille.

Représentation d'une image en mémoire

Si `im` est une image chargée avec `PIL.Image.open`, on accède à ses pixels via la fonction `im.load()` qui renvoie un tableau *indexé par des couples d'entiers* (et non pas une matrice au sens python du terme).

Par exemple,

Afficher/masquer les numéros de lignes

```
1 pixels = im.load()
2 print(pixels[0,0])
```

renvoie la valeur du pixel en haut à gauche de l'image.

Affichez des pixels de l'image en couleurs, puis de l'image en noir et blanc. Que remarquez-vous ?

Modifier une image

Pour modifier un pixel, on change sa valeur dans le tableau des pixels. Essayez :

Afficher/masquer les numéros de lignes

```
1 pixels[0,0] = 0
2 im.save(r'tigre_mod.png')
```

Est-ce que ça fonctionne avec l'image en noir et blanc ? Avec celle en couleurs ? Quel est l'effet produit.

Premiers filtres

Vous êtes parés pour écrire votre premier filtre : écrivez une fonction `rev(im)` qui remplace tous les pixels de l'image `im` par leur valeur en négatif. Traitez d'abord le cas en noir et blanc, puis celui en couleurs. Essayez de la modifier pour que ça fonctionne dans tous les cas.

Couleurs

Autre exercice basique : écrivez une fonction `canaux(im)` qui prend en argument une image `im` en couleurs (mode `"RGB"`) et renvoie le triplet d'images en nivaux de gris

(mode `"L"`) donnant les valeurs de chaque canal (rouge, vert, bleu). *Indice: il faut créer de nouvelles images en utilisant* `Image.new` *(voir* `help(Image.new)` *dans l'interpréteur).*

Écrivez une fonction `couleur_vers_gris(im)` qui transforme une image en couleurs vers une image en niveaux de gris. Est-ce convaincant ? Comparez avec l'image en niveaux de gris proposée plus haut : comment expliquez-vous la différence ?

Dans la suite, on pourra se limiter au cas des images en noir et blanc.

Lissage et bruit

Développez une série de filtres :

- filtre en moyenne (chaque pixel est remplacé par la moyenne de son voisinage de Moore, c'est-à-dire le pixel et ses 8 voisins)
- filtre médian (la même chose mais avec la médiane)
- filtre qui génère un bruit aléatoire (on réfléchira ensemble à ce que ça signifie)

Testez ensuite l'effet du filtre en moyenne et du filtre médian sur une image bruitée : qu'en pensez-vous ?

Filtrage linéaire

Programmez le filtre de Sobel comme une fonction `sobel(im)` : chaque pixel de l'image filtrée est une approximation de la norme du gradient de l'intensité.

Pour réaliser cette étape, il sera utile de traiter d'abord le cas général du filtrage linéaire, qui consiste à calculer pour chaque pixel le produit de convolution de son voisinage de Moore avec une matrice 3x3 fixée (le noyau) : écrivez une fonction `filtre(im,noyau)` qui fait ce travail.

Vous pourrez alors tester cette méthode avec les deux matrices utilisées dans le filtre de Sobel (voir la page Wikipédia), qui génèrent une approximation des composantes horizontale et verticale du gradient.

⚠ À cause des limites de valeurs pour un pixel (de 0 à 255), on perd les valeurs négatives ou trop fortes des composantes du gradient : on ne distingue que le passage d'une partie foncée à une partie claire. Pour mieux comprendre, essayez avec les matrices opposées.

Le filtre de Sobel est alors obtenu en appliquant pour chaque pixel les deux matrices, en en prenant la norme du couple de valeurs obtenues (essayez avec les normes N_1, N_2 et N_inf).

⚠ Vue la remarque précédente, la méthode qui consisterait à générer deux filtrages linéaires avec les matrices pour le gradient horizontal et vertical, puis en calculant la norme point à point à partir des deux images obtenues fonctionne mal : on a tronqué les valeurs trop tôt.

WikISN: Filtres d'image (dernière édition le 2016-01-25 14:11:53 par LionelVaux)