TD : CAMOUFLER UNE IMAGE

1 OBJECTIF : CACHER UNE IMAGE SUR LES BITS DE POIDS FAIBLE D'UNE AUTRE IMAGE

On démarre avec deux images avec le même nombre de pixels : une première destinée à servir de masque « masque.jpg », et une seconde destinée seulement à un interlocuteur « secret.jpg ». Chaque image est stockée sous forme d'une liste de dimension 3 composée d'entiers entre 0 et 255. Les deux premières dimensions sont les coordonnées x et y des pixels, et la troisième dimension contient les niveaux des trois couleurs RGB (red, green, blue), codées sur un octet (soit un entier entre 0 et 255 pour chaque couleur, ce qui permet d'obtenir plus de 16 millions de couleurs différentes).

L'idée de départ est que pour chacun de ces entiers un codage entre 0 et 255 n'est peut être pas nécessaire. En effet, pour chaque entier codé sur 8 bits, les 4 bits de poids fort donnent quasiment toute l'information, les autres servant à apporter des nuances. On va alors tronquer l'information de chacune de ces valeurs et ne garder que l'information principale de chaque image. L'information principale de l'image à cacher sera alors dissimulée sur les bits de poids faible de l'image masque.

2 ENLEVER LES BITS DE POIDS FAIBLE D'UNE IMAGE

Q1. Ouvrez une console python (ou ipython) et placez- vous dans le répertoire du TD.

```
cd chemin_vers\Camouflage
```

Q2. Vérifiez que la commande « ls » renvoie bien les noms des images.

Q3. Importez les bibliothèques pylab pour pouvoir afficher des images.

```
from pylab import *
```

Q4. Importez la bibliothèque scipy pour pouvoir charger et enregistrer les images.

```
import scipy.misc as scm
```

Q5. Importez l'image masque.bmp affichez-la et enregistrez-la sous un autre nom.

```
im_masque=scm.imread("masque.bmp")
imshow(im_masque)
show()
scm.imsave("masque8bits.bmp",im_masque)
```

Damien Iceta - Paul Bernardoni –
Marc Dérumaux - Emmanuel Broutier - UPSTI

Q6. Vérifiez que l'image a bien été enregistrée dans le répertoire sous son nouveau nom et affichez-là.

Q7. Appliquez la commande "print" à l'image pour voir comment celle-ci est stockée.

Ainsi, pour afficher les 3 couleurs RGB du pixel de coordonnées (20,40), tapez :

```python
im_masque[20,40]
```

La taille (en pixels) de l'image peut être obtenue par les commandes :

```python
im_masque.shape[0]
im_masque.shape[1]
```

2.1 CONSERVATION DES BITS DE POIDS FORT D'UN PIXEL

Q8. Pour afficher en binaire la valeur de la couleur d'un pixel (i de 0 à 2 correspond à l'indice de la couleur : 0=Red, 1=Green, 2=Blue). Pour le Rouge tapez : $bin(im\_masque[20,40][0])$.

Pour travailler sur un code binaire on peut utiliser les opérateurs suivants :

Opération	Type retour	Description
$x\&y$	int	Opération et sur les bits de x et y
$x\|y$	int	Opération ou sur les bits de x et y
x^y	int	Opération ou exclusif sur les bits de x et y
$x << y$	int	Décalage à gauche de y bits sur x
$x >> y$	int	Décalage à droite de y bits sur x
x	int	Inversion des bits de x

En utilisant $im\_masque[20,40][0]\&0b00001000$ on peut sélectionner un bit et afficher le résultat en binaire avec : $bin(im\_masque[20,40][0]\&0b00001000)$.

Q9. Proposez une commande sur la console afin de sélectionner et d'afficher en binaire les 4 premiers bits correspondant à la couleur Rouge du pixel [20,40]. De même proposez une commande pour les 4 derniers bits. Vérifiez à chaque fois les résultats obtenus.

2.2 CONSERVATION DES BITS DE POIDS FORT DES PIXELS D'UNE IMAGE

Q10. En utilisant deux boucles imbriquées (pour chaque indice de ligne balayage de tous les pixels suivant un indice de colonne) complétez le programme ci-dessous permettant d'enlever tous les bits de poids faible de chacun des pixels. Enregistrez l'image obtenue sous le nom « masque4bits.bmp ».

– Pour éditer le programme vous pouvez taper la commande "edit" dans la console ou ouvrir un éditeur quelconque (spyder par exemple). Puis ouvrir le fichier $ex1\_bit\_poids\_fort\_a\_completer.py$ situé dans le même répertoire que précédemment (..\$chemin_vers$\Camouflage$).

– Enregistrez-le au même endroit sous le nom "ex1.py". Puis n'oubliez pas de le sauvegarder régulièrement. Ensuite pour l'exécuter, dans la console vous pouvez taper "python ex1.py".

```python
# importer les biblioth èques
from pylab import *
import scipy.misc as scm

# charger l'image masque en créant l'objet im_masque
im_masque=scm.imread("masque.bmp")
# déterminer le nombre de lignes et de colonnes de pixels
nb_ligne=...# A COMPLETER
nb_colonne=...# A COMPLETER

# 2 boucles imbriquées pour conserver les bits de poids forts pour chaque couleur
# A COMPLETER

# sauvegarder l'image obtenue
scm.imsave(" .... bmp", img1) #... A COMPLETER

# Afficher l'image obtenue
imshow(im_masque)
axis('off')
show()
```

Q11. Ne conservez maintenant que les deux bits de poids fort. Enregistrez l'image obtenue sous le nom masque2bits.bmp.

Q12. Comparez les 3 images "masque8bits.bmp", "masque4bits.bmp" et "masque2bits.bmp" et conclure.

3 ECRITURE DES BITS DE POIDS FORT DE L'IMAGE À CACHER SUR LES BITS DE POIDS FAIBLE DU MASQUE

Q13. Afin de décaler des bits on peut utiliser x»y (décalage à droite de y bits sur x). Testez $bin(im\_masque[20,40][0] >> 4)$.

Q14. Complétez alors le programme suivant afin d'écrire les bits de poids fort de l'image à cacher sur les bits de poids faible du masque : Ouvrir le fichier "$ex2\_cacher\_image\_a\_completer.py$" situé dans le même répertoire que précédemment et enregistrez-le au même endroit sous le nom "ex2.py".

```python
# charger les images masque4bits et secret en créant les objets im_masque et im_secret :
im_masque4bits=scm.imread("masque4bits.bmp")
im_secret=scm.imread("secret.bmp")

# déterminer le nombre de lignes et de colonnes de pixels
nb_ligne=im_masque4bits.shape[0]
nb_colonne=im_masque4bits.shape[1]

# Pour chaque couleur de chaque pixel: on décale les bits de poids fort vers la droite
for ligne in range(nb_ligne):
        for col in range(nb_colonne):
                ... #A COMPLETER

# Somme globale entre img_masque4bits (4 bits de poids forts de masque + 0000) et
# img2 (0000 + 4 bits de poids fort de secret. bmp) . Cela est possible car img1et img2 ont même taille
img_sortie=im_masque4bits+im_secret

# Sauvegarder l'image obtenue
scm.imsave("image_cachee.bmp", img_sortie)

# Afficher l'image obtenue
imshow(img_sortie)
axis('off')
show
```

L'image résultat de la modification est nommée "*image_cachee.bmp*".

Q15. Comparer l'image "*image_cachee.bmp*" avec "*masque4bits.bmp*" et "*masque8bits.bmp*" et conclure.

4 RETROUVER UNE IMAGE CACHÉE

Q16. On cherche maintenant à retrouver l'image présente sur les bits de poids faible. Complétez le programme suivant pour trouver cette image.

– Ouvrir le fichier "*ex3_retrouver_image_a_completer.py*" situé dans le même répertoire que précédemment et enregistrez-le au même endroit sous le nom "ex3.py".

```python
# charger les images image_cachee en créant l'objet img1:
img1=scm.imread('image_cachee.bmp')

# déterminer le nombre de lignes et de colonnes de pixels :
nb_ligne=img1.shape[0]
```

```python
nb_colonne=img1.shape[1]

# Pour chaque couleur, décaler les bits de poids faibles vers le bits de poids fort
for ligne in range(nb_ligne):
        for col in range(nb_colonne):
            .... #A COMPLETER

# Sauvegarder l'image obtenue
scm.imsave("secret4bits.bmp", img1)

# Afficher l'image obtenue
imshow(img1)
axis('off')
show()
```

Q17. Comparez les images secret4bits.bmp et secret et conclure.

Q18. Modifier légèrement ce programme pour retrouver l'image cachée dans "$image\_trouver.bmp$". Enregistrez ce nouveau programme sous le nom "ex4.py".

Q19. Comment cacher une image secrète dans une image masque ayant un nombre de colonnes de pixels et un nombre de lignes de pixels supérieurs à ceux de l'image secrète ?

Q20. Modifiez le programme "$ex2.py$" afin de cacher sur les 4 bits de poids faible de l'image masque "$masque.bmp$", l'image secrète "$petit\_secret.bmp$" qui est de taille plus petite (200 x 200). . Enregistrez ce nouveau programme sous le nom "$ex5.py$".

Q21. Vérifiez votre travail en modifiant légèrement le programme "ex4.py".