TP 14 - Mini-projet : Traitements d'images

Objectifs

L'objectif de ce TP, outre celui de développer des capacités en manipulation de fichiers, est de se familiariser avec la structure d'un fichier de données binaires et en particulier avec ses traitements. Il s'agit de faire du traitement d'image.

Les images cibles seront deux œuvres de l'illustre Léonard de Vinci : La Joconde et un autoportrait. Ces deux images sont transmises dans un des formats images les plus simples mais aussi les plus lourds : le format BMP (ici en 24 bits)

Un document ressource (DR) est à disposition pour pouvoir répondre aux attentes de ce TP. Il a pour premier but de détailler la structure d'un fichier BMP. Quelques informations nécessaires à la manipulation de données binaires sous Python et aux manipulations de base sur les données relatives aux valeurs des composantes de couleur d'un pixel y figurent également.

Préliminaire

Supposons que l'on récupère dans une variable nommée `n_octets` de type byte (donc binaire) n octets successifs. Ces n octets représentent un nombre (c'est le cas par exemple dans un fichier image lorsque 4 octets successifs représentent la taille de l'image...).

Q1. Définir une fonction `val_dec(n_octets)` qui retourne la valeur décimale correspondant au nombre codé sur `n_octets`, dans l'hypothèse où le premier octet indexé est l'octet de rang le plus faible.

Par exemple, « b'AZ' » est le nombre binaire de deux octets correspondant aux codes ASCII de la chaîne AZ. Le code ASCII de A est : 65. Le code ASCII de Z est : 90. Ainsi, `val_dec(b'AZ')` doit évaluer 65 + 90 x 256 soit 23105.

Découverte de la structure d'un fichier BMP

Q2. Écrire une fonction `info_fichier(image)` permettant d'ouvrir en lecture un fichier image et d'en afficher les informations suivantes : signature du fichier, taille totale du fichier, offset de l'image, largeur de l'image, hauteur de l'image et profondeur de codage de la couleur.

Q3. Tester la fonction sur un des deux fichiers images fournis. En utilisant l'explorateur Windows pour afficher les propriétés puis les détails du fichier image, vérifier quelques données lues dans le fichier.

Traitement d'images

Q4. Proposer une fonction permettant d'obtenir une image en ne gardant que la composante rouge de l'image initiale. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 1.

Q5. Proposer une fonction permettant d'inverser les couleurs d'une image. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 2.

Q6. Proposer une fonction permettant d'obtenir une image en niveau de gris de l'image initiale. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 3.

Image 1

Image 2

Image 3

Q7. Proposer une fonction permettant d'obtenir une image monochrome à partir d'une image en niveau de gris. En plus de l'image sur laquelle s'appliquera le traitement, on prendra comme arguments `seuil`, `niveau1` et `niveau 2` où `seuil` est la valeur de la composante critique en dessous de laquelle la composante prend la valeur `niveau1` et au delà de laquelle la composante prend la valeur `niveau2`. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 4.

Q8. Proposer une fonction permettant de faire une rotation de 180° de l'image initiale. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 5.

Q9. Proposer une fonction permettant de faire une transformation miroir horizontal. Par exemple, cette fonction appliquée au fichier image vinci.bmp donne l'image 6.

Image 4

Image 5

Image 6

Mise en forme

Q10. Construire un script offrant à l'utilisateur le choix de rentrer le nom du fichier à traiter, le nom de fichier destination et enfin le choix du traitement à effectuer parmi la liste précédente. Par exemple :

```
>>>traitement()
TRAITEMENT D'IMAGE

Nom du fichier image source : vinci_gris.bmp
Nom du fichier image destination : vinci_modif.bmp
Traitement propose :
1.Inversion des couleurs
2.Transformation en niveaux de gris
3.Appliquer un effet de contraste ou monochrome
(s'applique sur une image en niveau de gris)
4.Appliquer une rotation à 180°
5.Appliquer un effet miroir
6.Appliquer un filtre rouge

Votre choix : 3
Valeur du seuil [0-255]: 128
Valeur niveau 1 [0-255]: 0
Valeur niveau 2 [0-255]: 255
```

Après ouverture des fichiers vinci_modif.bmp, on obtient ainsi :

Dossier ressources

I. Format d'un fichier BMP

Le format BMP est un des nombreux formats de codage d'une image. Comme tout fichier, un fichier .bmp possède un entête plus ou moins long permettant de spécifier ses caractéristiques utiles suivi des données. Le tableau ci-dessous présente l'organisation du fichier.

Structure d'un fichier BMP		
Champs principaux	Sous-champs	Description
Entête du fichier	Signature (2 octets)	Indique qu'il s'agit d'un fichier BMP à l'aide de 2 caractères : - BM : Bitmap Windows - BA : Bitmap OS/2 - CI : icone couleur OS/2 - CP : pointeur de couleur OS/2 - IC : icône - PT : pointeur OS/2
	Taille totale du fichier en octets (4 octets)	/
	Champs réservé (4 octets)	/
	Offset de l'image (4 octets)	Offset : décalage en français définit la position en octet du début des données images par rapport au début du fichier
Entête de l'image	Taille de l'entête de l'image en octets (4 octets)	/
	Largeur de l'image (4 octets)	Width : nombre de pixels horizontalement
	Hauteur de l'image (4 octets)	Height : nombre de pixels verticalement
	Nombre de plans (2 octets)	Vaut toujours 1 (!)
	Profondeur de codage de la couleur (2 octets)	Définit le nombre de bits utilisés pour coder la couleur. La profondeur peut valoir 1, 4, 8, 16, 24 ou 32.
	Méthode de compression (4 octets)	Valeurs possibles : - 0 : Pas de compression - 1 : codage RLE de 8 bits par pixel - 2 : codage RLE de 4 bits par pixel - 3 : codage Bitfields
	Taille totale de l'image en octets (4 octets)	/
	Résolution horizontale (4 octets)	Nombre de pixels/m horizontalement
	Résolution verticale (4 octets)	Nombre de pixels/m verticalement
	Nombre de couleurs de la palette (4 octets)	/
	Nombre de couleurs importantes de la palette (4 octets)	Peut être égale à 0 lorsque chaque couleur a son importance
Palette de l'image	4 octets	Optionnelle, si elle est définie, elle contient 4 octets qui correspondent successivement à : - la composante bleue - la composante verte - la composante rouge - un champ réservé
Codage de l'image	Images en 16 couleurs : 4bits/pixel (1 octet = 2 pixels)	Le codage de l'image se fait en écrivant successivement les bits correspondant à chaque pixel, ligne par ligne en commençant par le pixel en bas à gauche.
	Images en 256 couleurs : 8 bits/pixel (1 octet = 1 pixel)	
	Images en couleurs réelles : 24 bits/pixel (3 octets = 1 pixel) Chaque octet représente une composante, en respectant l'ordre bleu, vert, rouge.	

II. Manipulation de données binaires sous Python

II.1. Types binaires

Sous Python, il existe deux types de données binaires :

- byte : non modifiable
- bytearray : modifiable

Ces deux types de données peuvent chacun contenir un à plusieurs octets.

Soit *data* une variable de type byte. On ne peut pas la modifier. Ainsi, si on veut supprimer un octet sur 3 de *data*, le programme suivant renverra une erreur :

```
i=0
longueur=len(data)
while i<longueur:
      if i%3==0:
            data[i]=0
      i+=1
```

Il faut écrire :

```
i=0
longueur=len(data)
data=bytearray(data) #conversion de type pour travailler avec un type de données modifiable
while i<longueur:
      if i%3==0:
            data[i]=0
      i+=1
```

II.2. Fichiers binaires

Pour ouvrir un fichier en mode 'rb' (read binary), soit en lecture binaire, il suffit de faire (exemple avec le fichier « vinci.bmp ») :

```
source=open('vinci.bmp','rb')
```

Pour lire et récupérer les données, il suffit d'utiliser la méthode *.read()* . Attention, on récupère alors des données de type byte.

```
data=source.read()
```

Pour ouvrir un fichier en mode 'wb' (write binary) soit en mode écriture, il suffit de faire :

```
destination=open('copie_vinci.bmp','wb')
```

Par exemple, le script ci-dessous recopie le fichier image « vinci.bmp » dans un nouveau fichier « copie_vinci.bmp » (à tester!) :

```
source=open('vinci.bmp','rb')
data=source.read()
source.close()

destination=open('copie_vinci.bmp','wb')
destination.write(data)
destination.close()
```

Remarque : Si on ne veut lire que les *n* premiers octets, il suffit de faire `source.read(n)`. Puis, pour lire les *m* suivants, il suffit d'écrire à la suite : `source.read(m)` etc...

Remarque 2 : la méthode *.seek(i)* permet de se déplacer à l'index i... Ainsi, *source.seek(0)* ramène l'index à la position de début du fichier, et *source.seek(45)* amène l'index à la position 45.

II.3. Notions sur le code ASCII binaire

Pour représenter un caractère dans la machine, un entier naturel est attribué à chaque caractère par exemple à l'aide du code ASCII binaire (American Standard Code for Information Interchange). Ce code utilise un octet par caractère, avec le premier bit toujours à 0, et permet donc de représenter $2^7 = 128$ caractères. Ce sont les caractères qui se trouvent sur les touches d'un clavier d'ordinateur. Par exemple :

- les nombres de 65 à 90 correspondent aux 26 lettres capitales de l'alphabet A, B, C..., Z ;
- les nombres de 97 à 122 aux 26 lettres minuscules a,b,c...,z ;
- les nombres de 48 à 57 aux dix chiffres 0,1,2,...,9
- l'espace est codé par le nombre 32, le point par le nombre 46, etc...

Sur Python, l'instruction `chr(65)` donne le caractère 'A' et l'instruction `ord('A')` donne l'entier 65.

III.Traitement d'image

III.1. Principe

L'objectif va être de modifier une image.

Pour cela, on va :

- ouvrir en lecture binaire le fichier source et en écriture binaire le fichier destination
- copier les données qui ne sont pas relatives au codage de l'image (entête de fichier, entête de l'image.. sur toute la longueur de l'offset) dans le fichier destination qui gardera les mêmes caractéristiques que le fichier source.
- copier le reste des données (relatives au codage de l'image) dans une autre variable, modifier ces données selon la problématique posée puis les copier dans le fichier destination.
- refermer le fichier source et le fichier destination.

III.2. Faire disparaître une composante de couleur de pixel

Pour faire disparaître une composante, il suffit de mettre à zéro la valeur de cette composante.

III.3. Inversion des couleurs d'une image (négatif)

« Inverser les couleurs » d'une image est très simple. Si un pixel est codé sur 24 bits, cela signifie en fait que chaque couleur est codée sur 1 octet. Chaque composante a donc une valeur évoluant de 0 à 255.

Pour obtenir un effet inversé, il suffit de modifier la valeur de chaque composante de chaque pixel de l'image comme suit : valeur_composante = 255 – valeur_composante.

III.4. Obtenir une image en niveau de gris

Une des possibilités pour obtenir une image en niveau de gris à partir d'une image codée sur les 3 composantes de couleur (bleu, vert, rouge) est de remplacer la valeur de chaque composante de chaque pixel par la moyenne des valeurs des 3 composantes du pixel considéré.

III.5. Obtenir une image monochrome

Chaque composante peut prendre 256 valeurs, de 0 à 255. Pour rendre l'image monochrome (2 couleurs), il faut donc réduire le nombre de valeurs de chaque composante et donc la couleur de chaque pixel à 2.

Pour ce faire, il suffit de fixer un seuil (par exemple 128). Pour chaque pixel, pour chaque composante, il faut alors effectuer une comparaison. Si la valeur de la composante est inférieure au seuil, alors la composante prend par exemple la valeur 0. Sinon, elle prend la valeur 255.

NB : Les valeurs données dans l'explication ci-dessous (128, 0, 255) sont arbitraires. Libre à vous de choisir ses valeurs dans le cadre de ce projet.