Informatique PCSI
Prérequis TP 7 : manipulation d'une image

1 Fichiers images

Si on regarde les propriétés d'un fichier avec l'extension jpg représentant une image, on obtient des informations sur l'emplacement, la taille, la date de création. Il y a aussi la possibilité d'avoir plus de détails pour un fichier d'une photo provenant d'un appareil comme un smartphone. Nous pouvons obtenir de nombreuses informations sur le fichier mais aussi sur l'image, la photo, l'appareil utilisé, le logiciel, la date de prise de vue, la position GPS, etc. Ces informations sont enregistrées dans le fichier avec l'image proprement dite.

1.1 Une image

On distingue parmi les images numériques, les images vectorielles et les images matricielles. Les images vectorielles sont créées par des équations mathématiques. Les images matricielles sont constituées d'un ensemble de points ou pixels. La suite concerne les images matricielles.

Une image est définie par son nombre de pixels. Par exemple une image contenant 1600 pixels en largeur et 1200 pixels en hauteur a une définition de 1600×1200 pixels soit 1920000 pixels (près de 2 mégapixels). La résolution est exprimée en points ou pixels par pouce (ppp) ou dot per inch en anglais (dpi). Un pouce vaut environ 2,54 cm. Ainsi, une résolution de 72 ppp en largeur et en hauteur signifie qu'un carré de côté 2,54 cm contient $72 \times 72 = 5184$ pixels.

Une image de longueur 1600 pixels et de résolution 300 ppp, qui est une résolution correcte pour une impression, a une longueur de 5,33 pouce soit 13,55 cm.

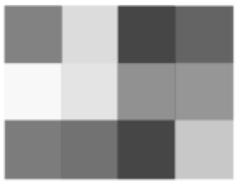
Dans le codage de couleur RGB (ou RVB en français pour rouge, vert, bleu), la couleur de chaque pixel est codée par trois octets. Chaque octet représente l'intensité d'une composante qui varie donc entre 0 et 255. Le triplet (255, 0, 0) code le rouge, le triplet (0, 0, 0) code le blanc, le triplet (255, 255, 255) code le noir, le triplet (120, 120, 120) code un niveau de gris, le triplet (255, 255, 0) code le jaune.

Le programme suivant aide à comprendre comment est composée une image. La fonction `pixel` permet de dessiner un pixel, en fait un petit carré. La fonction `dessine` dessine les pixels donnés dans une matrice.

```
from turtle import *

def pixel(coords, taille, couleur): # définition d'un pixel
    ht()
    up()
    goto(coords)
    down()
    color(couleur)
    begin_fill()
    for i in range(4): # un pixel est représenté par un carré
        forward(taille)
        right(90)
    end_fill()

def dessine(image, resolution, origine=(0,0)):
    speed('fastest')
    n = len(image)
    p = len(image[0])
```

```
    taille = 1 // resolution
    x_origine, y_origine = origine
    x0 = x_origine -p * taille // 2
    y0 = y_origine + n * taille // 2
    up(); goto((x0, y0)); down()
    for i in range(n):
        for j in range(p):
            coords = (x0 + j * taille, y0 -i * taille)
            pixel(coords, taille, image[i][j])

colormode(255)
image1 = [[(255, 0, 0), (0, 255, 0), (0, 0, 255), (100, 100, 100)],
          [(255, 255, 0), (0, 255, 255), (255, 0, 255), (150, 150, 150)],
          [(128, 128, 0), (0, 128, 128), (128, 0, 128), (200, 200, 200)]]

dessine(image1, 1/30, (0, 100)) # image1 est une matrice de pixels
```

Les « pixels » sont représentés ci-dessous en niveaux de gris.



La liste des pixels est longue à écrire. On peut créer une image (une matrice) aléatoire.

```
from random import randint
def alea(n, p): # largeur, hauteur
    img = [[(0, 0, 0) for j in range(n)] for i in range(p)]
    for i in range(p):
        for j in range(n):
            r, v, b = randint(0, 255), randint(0, 255), randint(0, 255)
            img[i][j] = (r, v, b)
    return img

image2 = alea(16, 12)
dessine(image2, 1/16, (-200, 0))
```

Le dessin prend du temps et le nombre de points est limité si on souhaite le voir s'afficher à l'écran. Ce n'est évidemment pas une bonne méthode pour construire des images.

1.2 Transformation d'une image

La modification d'une image consiste à modifier la matrice de pixels. En général cela s'effectue avec deux boucles imbriquées qui permettent d'avoir accès à chaque pixel. La complexité en temps est alors de l'ordre de $n \times p$ si n est le nombre de lignes et p le nombres de colonnes. On peut essayer autant que possible à limiter la complexité en espace.

2 Fichiers PBM, PGM et PPM

Ce sont les formats de fichiers : portable bitmap (pbm), portable graymap (pgm) et portable pixmap (ppm).

On peut obtenir un fichier pbm, pgm ou ppm à partir d'une image dans un format quelconque. On l'ouvre avec un logiciel comme GIMP et on exporte l'image en choisissant l'extension. Attention, pour chaque extension il existe un format texte ASCII et un format binaire. On trouve les informations nécessaires par exemple à l'adresse : `https://fr.wikipedia.org/wiki/Portable_pixmap`.

Choisir un fichier image au format jpeg ou png, le copier et renommer le "image1" (.jpg ou .png). L'ouvrir avec GIMP.

Au besoin : dans le menu Image, Echelle et taille de l'image, choisir la largeur et la hauteur de l'ordre de 800 par 600 et cliquer sur mise à l'échelle. (Obtenir une taille maxi de 500 Ko).

Ensuite dans le menu Fichier, Exporter sous ..., modifier le nom en "image1.pbm", cliquer sur Exporter. Dans Formatage des données, choisir ASCII et cliquer sur Exporter.

Comparer la taille avec celle du fichier original. Ouvrir le fichier obtenu avec un éditeur de texte. On obtient : un code P1, puis éventuellement un commentaire # ..., puis deux nombres. Que signifient ces deux nombres ? La suite est composée de 0 et de 1 (l'image est en noir est blanc).

Recommencer la procédure d'exportation et modifier le nom en "image1.pgm" (choisir toujours ASCII). Cette fois avec un éditeur de texte on lit le code P2, puis éventuellement un commentaire # ..., puis les deux même nombres que précédemment. La suite est composée d'entiers entre 0 et 255.

Recommencer pour obtenir un fichier "image1.ppm".

2.1 Écriture

L'objectif est d'écrire un programme avec trois fonctions qui lisent chacune un de ces types de fichier et placent les données dans une matrice de pixels (une liste de listes), une sous-liste représentant une ligne de pixels : en pbm et pgm une valeur par pixel, 0 ou 1 pour noir et blanc, de 0 à 255 pour niveaux de gris, en ppm 3 nombres par pixels chacun de 0 à 255.

Écriture d'un fichier pbm

```
def ecrire_fichier_pbm(nom, mat):
    assert nom[len(nom)-3: len(nom)] == "pbm"
    hauteur = len(mat)
    largeur = len(mat[0])
    f = open(nom, 'w')
    f.write('P1\n' + str(largeur) + ' ' + str(hauteur) + '\n')
    for i in range(hauteur):
        for j in range(largeur):
            bit = str(mat[i][j])
            f.write(bit)
            if (i * largeur + j + 1) % 70 == 0:
                f.write('\n')
    f.close()
```

Écriture d'un fichier pgm

```
def ecrire_fichier_pgm(nom, mat, maxi):
    assert nom[len(nom)-3: len(nom)] == "pgm"
    hauteur = len(mat)
```

```
    largeur = len(mat[0])
    f = open(nom, 'w')
    f.write('P2\n' + str(largeur) + ' ' + str(hauteur) + '\n')
    f.write(maxi + '\n')
    for i in range(hauteur):
        for j in range(largeur):
            octet = str(mat[i][j])
            f.write(octet)
            f.write('\n')
    f.close()
```

Écriture d'un fichier ppm

```
def ecrire_fichier_ppm(nom, mat, maxi):
    assert nom[len(nom)-3: len(nom)] == "ppm"
    hauteur = len(mat)
    largeur = len(mat[0])
    f = open(nom, 'w')
    f.write('P3\n' + str(largeur) + ' ' + str(hauteur) + '\n')
    f.write(maxi + '\n')
    for i in range(hauteur):
        for j in range(largeur):
            pixel = mat[i][j]
            r, v, b = str(pixel[0]), str(pixel[1]), str(pixel[2])
            f.write(r + '\n' + v + '\n' + b + '\n')
    f.close()
```

On peut regrouper les trois fonctions en une seule :

```
def ecrire(nom, mat, maxi=None):
    if nom[len(nom)-3: len(nom)] == "pbm":
        ecrire_fichier_pbm(nom, mat)
    elif nom[len(nom)-3: len(nom)] == "pgm":
        ecrire_fichier_pgm(nom, mat, maxi)
    elif nom[len(nom)-3: len(nom)] == "ppm":
        ecrire_fichier_ppm(nom, mat, maxi)
```

2.2 Lecture

L'objectif est d'écrire un programme qui à partir d'une matrice de pixels écrit le fichier image. On peut vérifier le résultat obtenu avec GIMP.

Lecture d'un fichier pbm

Il s'agit de lire un fichier texte et de récupérer les informations : le code P1 caractéristique d'un fichier pbm, la largeur, la hauteur et les valeurs des pixels 0 ou 1.

```
def lire_fichier_pbm(nom):
    f = open(nom, 'r')
```

```
    magic = f.readline().rstrip()
    dim = f.readline()
    while dim[0] == "#":
        dim = f.readline()
    dim = dim.rstrip().split(" ")
    dim = [int(dim[0]), int(dim[1])]
    mat = [[0 for j in range(dim[0])] for i in range(dim[1])]
    tab = f.readlines()
    i, j = 0, 0
    for ligne in tab:
        if ligne[0] == "#":
            continue
        ligne = ligne.rstrip()
        for n in ligne:
            if n != " ":
                mat[i][j] = int(n)
                j = (j+1) % dim[0]
                if j == 0:
                    i = (i+1) % dim[1]
    f.close()
    return mat
```

Lecture d'un fichier pgm

```
def lire_fichier_pgm(nom):
    f = open(nom, 'r')
    magic = f.readline().rstrip()
    dim = f.readline()
    while dim[0] == "#":
        dim = f.readline()
    dim = dim.rstrip().split(" ")
    dim = [int(dim[0]), int(dim[1])]
    maxi = f.readline().rstrip()
    mat = [[0 for j in range(dim[0])] for i in range(dim[1])]
    tab = f.readlines()
    i, j = 0, 0
    for ligne in tab:
        if ligne[0] == "#":
            continue
        ligne = ligne.rstrip().split(" ")
        for n in ligne:
            if n != " ":
                mat[i][j] = int(n)
                j = (j+1) % dim[0]
                if j == 0:
                    i = (i+1) % dim[1]
    f.close()
    return mat, maxi
```

Lecture d'un fichier ppm

```
def lire_fichier_ppm(nom):
    f = open(nom, 'r')
    magic = f.readline().rstrip()
    dim = f.readline()
    while dim[0] == "#":
        dim = f.readline()
    dim = dim.rstrip().split(" ")
    dim = [int(dim[0]), int(dim[1])]
    maxi = f.readline().rstrip()
    mat = [[0 for j in range(dim[0])] for i in range(dim[1])]
    tab = f.readlines()
    i, j = 0, 0
    pixel = []
    for ligne in tab:
        if ligne[0] == "#":
            continue
        ligne = ligne.rstrip().split(" ")
        for n in ligne:
            if n != " ":
                if len(pixel) == 2:
                    pixel.append(int(n))
                    mat[i][j] = pixel
                    j = (j+1) % dim[0]
                    if j == 0:
                        i = (i+1) % dim[1]
                elif len(pixel) == 3:
                    pixel = [int(n)]
                else:
                    pixel.append(int(n))
    f.close()
    return mat, maxi
```

On peut regrouper les trois fonctions en une seule :

```
def lire(nom):
    if nom[len(nom)-3: len(nom)] == "pbm":
        return lire_fichier_pbm(nom)
    elif nom[len(nom)-3: len(nom)] == "pgm":
        return lire_fichier_pgm(nom)
    elif nom[len(nom)-3: len(nom)] == "ppm":
        return lire_fichier_ppm(nom)
```