IF6 – Traitement d'images

Conçue et améliorée au cours du 19ème siècles, la photo argentique a connu ses heures de gloire jusqu'aux années 1980 pour laisser place petit à petit à la photo numérique.

En 1969, Williard Boyle et George Smith conçoivent le premier capteur électronique capable de transformer un rayonnement électromagnétique en signal électrique, ce qui va permettre la réalisation de photos numériques.

Compte tenu du nombre impressionnant de pixels que comporte les images aujourd'hui, un algorithme de codage comme le jpeg a dû être réalisé.

Objectifs :
- Connaître quelques formats d'images ;
- Comprendre le principe d'algorithmes de transformations simples ;
- Programmer des modifications et transformations diverses.

I – Fichiers images

I-1) Une image

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.

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 × 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.

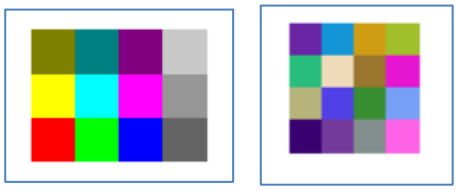
Dans le codage de couleur RGB (ou RVB en français pour rouge, vert, bleu), la couleur de chaque pixel est codée par trois octets. Chaque octet représente l'intensité d'une composante qui varie donc entre 0 et 255. Le triplet (255, 0, 0) code le rouge, le triplet (0, 0, 0) code le noir, le triplet (255, 255, 255) code le blanc, 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
    up() #Etat du stylo - déplacement
    goto(coords) #Déplacement du stylo
    down() #Etat du stylo - écriture
    color(couleur) #Choix de la couleur
    begin_fill() #À appeler juste avant de dessiner une forme à remplir.
    for i in range(4): # un pixel est représenté par un carré
        forward(taille) #On trace les côtés de notre pixel
        right(90)
    end_fill() #Remplit la forme dessinée après le dernier appel à begin_fill()

def dessine(image,taille,origine):
    speed=10 #Vitesse de tracé entre 1 (le plus lent) à 10 (le plus rapide) - 0 = pas d'animation
    n=len(image) #Nombre de pixels sur les y
    p=len(image[0]) #Nombre de pixels sur les x
    x0,y0=origine
    up()
    goto((x0,y0)) #On démarre le tracé en haut à gauche
    down()
    for i in range(n):
        for j in range(p):
            coords=(x0+j*taille,y0-i*taille) #On déplace notre curseur
            pixel(coords,taille,image[i][j]) #On dessine notre pixel

clearscreen() #On efface la figure de turtle
colormode(255) #Renvoie le mode de couleur utilisé
image1=[[(255,255,255),(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(image1,30,(0,0)) # image1 est une matrice de pixels4
```

```
from random import randint
def alea(n,p): # largeur, hauteur
    img=[[(0, 0, 0) for j in range(n)] for i in range(p)] #On crée une image de pixels noirs.
    for i in range(p):
        for j in range(n):
            r,v,b=randint(0,255),randint(0,255),randint(0,255) #On remplit de couleurs aléatoires
            img[i][j]=(r,v,b)
    return img

image2=alea(4,4)
dessine(image2,16,(-200,0))
```

Sous Jupyter cela crée une fenêtre graphique à part. Les deux programmes proposés sont très lents, il va donc falloir trouver une autre méthode pour construire des images.

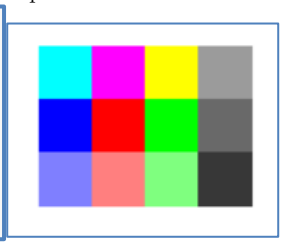


I-2) Transformation simple d'une image

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 × 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.

Par exemple, on peut écrire une fonction « inverse » qui inverse la couleur d'un pixel.

```
def inversion(pixel):
    r,v,b=pixel
    return (255-r,255-v,255-b)

def inverse(image):
    n=len(image) #nombre de lignes
    p=len(image[0]) #nombre de colonnes
    inv=[[inversion(image[i][j]) for j in range(p)] for i in range(n)]
    return(inv)

image3=inverse(image1)
dessine(image3,1/30,(0,0))
```



La fonction inverse sur l'image1 donne le résultat suivant. Cela sera plus simple d'analyser le résultat sur une image noir et blanc.

II - Quelques types de fichiers

II-1) 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.

Si on ouvre un fichier type « PBM » avec un éditeur de texte, on obtient : un code P1, puis éventuellement un commentaire # ..., puis la largeur et la hauteur de l'image. Ensuite on a une suite composée de 0 et de 1 (l'image est en noir est blanc).

Si on ouvre un fichier type « PGM » avec un éditeur de texte, on obtient un code P2, puis éventuellement un commentaire # ..., puis la largeur et la hauteur de l'image. Ensuite on a une suite composée de nombres entre 0 et 255…(l'image est en échelle de gris).

Si on ouvre un fichier type « PPM » avec un éditeur de texte, on obtient un code P3, puis éventuellement un commentaire # ..., puis la largeur et la hauteur de l'image. Ensuite on a une suite composée de 3 nombres entre 0 et 255…(l'image est en échelle de couleurs RVB).

On va choisir un fichier image au format jpeg ou png, le copier et le renommer en "image1".

- On ouvre avec GIMP l'image voulue.
    - Lien de téléchargement : https://www.gimp.org/downloads/
- On redimensionne l'image (menu image -> Echelle et Taille) afin d'avoir une image de moins de 500ko. Le format 800 par 600 s'y prête bien.
- Ensuite dans le menu Fichier, Exporter sous ..., modifier le nom en "image01.pbm", cliquer sur Exporter. Dans Formatage des données, **choisir ASCII** et cliquer sur Exporter.

On utilisera des images de ce type en TP pour effectuer des manipulations d'image.

II-2) Ecriture de fichiers

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.

Ce programme vous sera distribué en TP.

II-3) Lecture de fichiers

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.

Ce programme vous sera distribué en TP.

III – Exemple de manipulation : la rotation

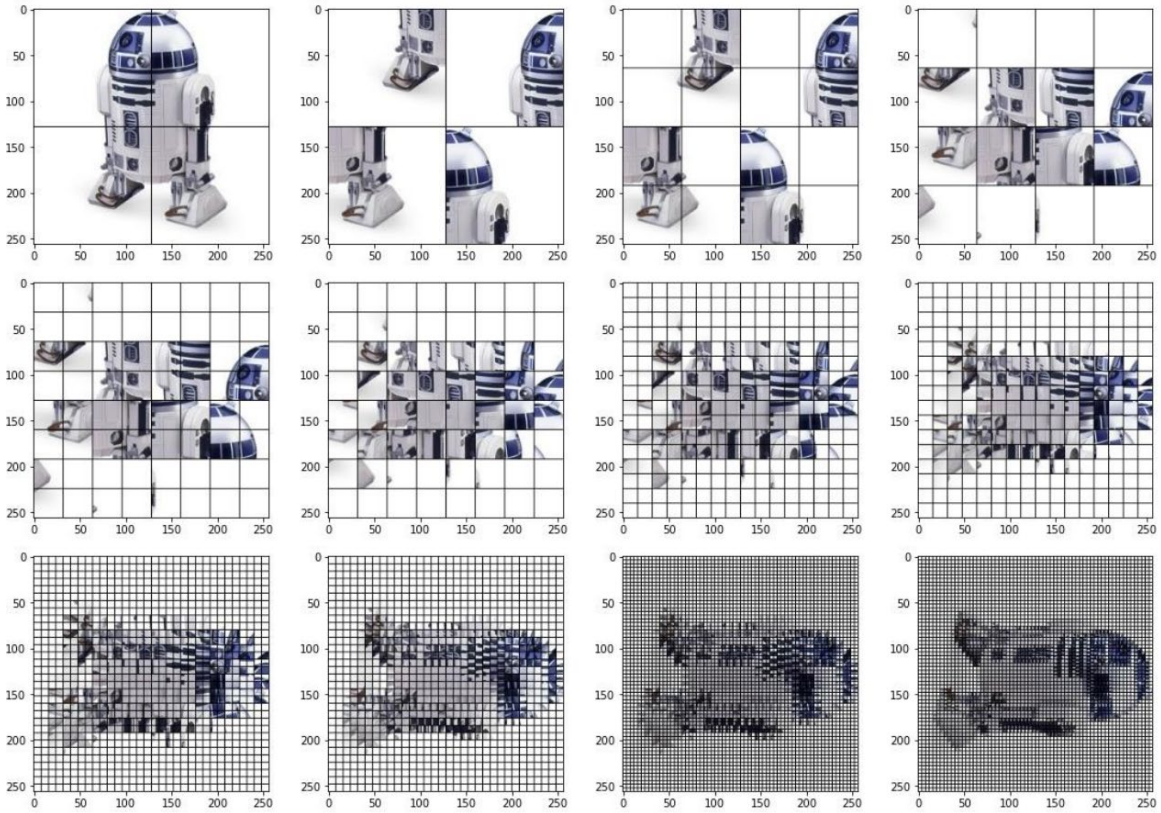
III-1) Principe

La méthode « diviser pour régner » consiste à découper un problème en sous-problèmes similaires (d'où l'algorithme récursif résultant) afin de réduire la difficulté du problème initial. On espère casser récursivement le problème en sous-problèmes de plus en plus petits jusqu'à obtenir des cas simples permettant une résolution directe. L'expression provient du latin « divide ut imperes » et désignait

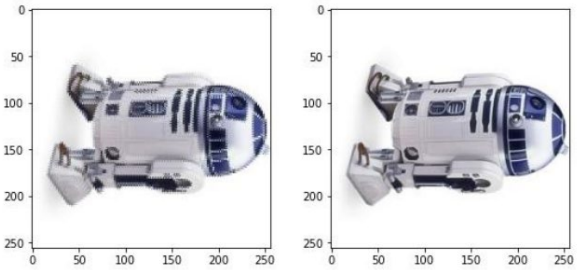
une stratégie militaire (ou politique).

Pour effectuer une rotation d'un quart de tour, on peut diviser l'image en 4, effectuer une permutation circulaire des quatre quadrants, puis appliquer récursivement le même processus à chaque quadrant, jusqu'à obtenir des quadrants contenant un seul pixel.

III-2) Exemple D2-R2

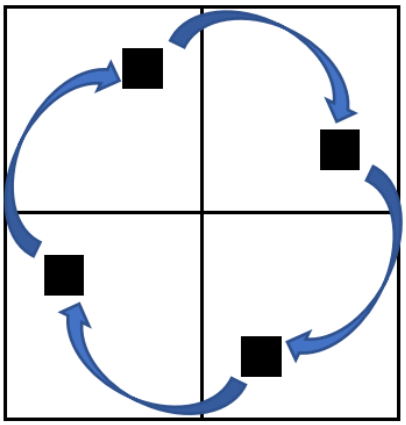


Sans le quadrillage, on obtient le résultat suivant :



C'est cet algorithme que l'on va implanter en TP mais celui-ci est assez long à mettre en place.

On peut utiliser une méthode plus directe :



Dont on pourra implémenter l'algorithme en TP, si vous avez le temps…