TP6 : Transformations d'images

1 Introduction

Cette séance de travaux pratiques est basée sur les travaux de M. Delahaye Jean-Paul et M. Mathieu Philippe publiés dans la revue *Pour la science* de Décembre 1997. [1]

Pendant cette séance, nous allons étudier certaines transformations bijectives d'une image qui déplacent les points d'un endroit à un autre sans en ajouter ni en enlever. Ces transformations sont comme des permutations de l'ensemble des points de l'image. Une propriété intéressante de ces transformations bijectives est qu'elles reviennent toujours à leur point de départ après un certain nombre d'applications. Cela est dû au fait que le groupe des permutations d'un ensemble fini a un ordre fini, donc tous ses éléments aussi. Par exemple, la transformation par symétrie de l'image par rapport à un axe vertical passant par son milieu a un ordre de 2, tandis que la transformation par rotation d'un quart de tour a un ordre de 4.

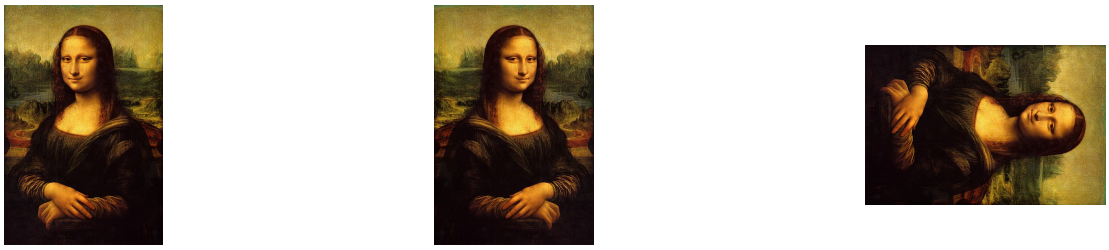


FIGURE 1 – Une image, sa transformée par symétrie verticale et sa transformée par rotation d'un quart de tour.

Nous allons nous intéresser plus particulièrement à deux transformations bijectives pour lesquelles le nombre d'étapes avant de voir réapparaître l'image initiale dépend des dimensions de celle-ci et peut être dans certains cas très grand.

2 Traitement d'image en Python

Pour ce TP nous aurons besoin des deux bibliothèques suivantes : `numpy` et `matplotlib.pyplot` :

```
import numpy as np
import matplotlib.pyplot as plt
import os
```

La seconde partie contient deux fonctions utiles :

- La fonction `plt.imread` prend en argument une chaîne de caractères qui décrit l'emplacement d'un fichier image au format PNG et renvoie un tableau `numpy`. Ce tableau a la même dimension que l'image, et chaque case contient un triplet de valeurs RGB du pixel correspondant de la forme `[r,g,b]` où r, g et b sont des valeurs entre 0 et 255 donnant la quantité de rouge, vert, bleu donnant la teinte du pixel.
- La fonction `plt.imshow` prend en argument un tableau `numpy` et affiche l'image associée à cette matrice (utilisez éventuellement `plt.show()` si le mode interactif n'est pas activé).

Récupérer sur le site le fichier *mario.png* et sauvegardez-le dans le répertoire courant. Exécutez ensuite le script suivant :

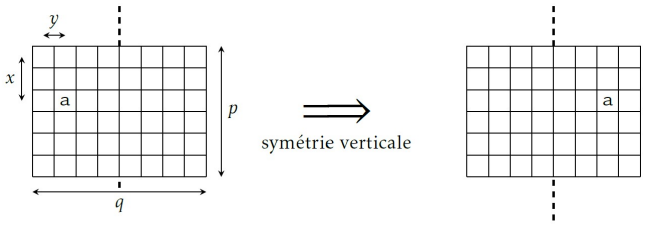
```
os.chdir(r'VotreCheminRepertoire')
mario = plt.imread('mario.png')
plt.imshow(mario)
```

La première ligne permet de désigner le chemin où chercher les documents. La seconde convertit l'image *mario.png* en une matrice `numpy` appelée "*mario*", et la seconde ligne affiche cette image à l'écran. Cette image de taille à définir va être utilisée pour expérimenter certaines transformations bijectives. Pour commencer, nous allons calculer les transformées de cette image, en commençant par une symétrie verticale puis une rotation d'un quart de tour. Cependant, plutôt que de transformer l'image elle-même, nous allons créer une nouvelle image contenant l'image transformée (ce qui est plus simple). Pour cela, nous allons copier chaque pixel de l'image initiale dans la nouvelle image vide à sa nouvelle position.

1. Images brouillées, Images retrouvées (*Pour la science* num 242, dec 1997)

2.1 Symétrie axiale

1. Créer une fonction `matrice_blanche(p,q)` qui renvoie la matrice associée à une image entièrement blanche (elle contient uniquement le triplet `[255,255,255]` répété pq fois). Comment obtient-on une image noire ? Une image bleue ?
2. Étant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat d'une symétrie d'axe vertical passant par le centre de l'image.
3. En déduire une fonction `symetrie` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la symétrie d'axe vertical. Les dimensions $p \times q$ de l'image initiale peuvent être calculées par l'instruction `np.shape`.

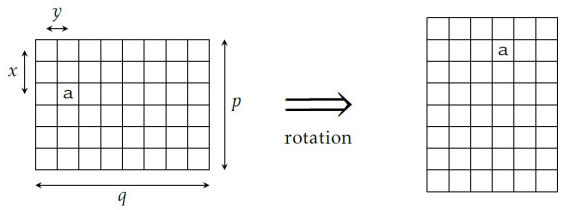


On testera cette fonction en faisant apparaître à l'écran le résultat de `symetrie(mario)`.

2.2 Rotation d'un quart de tour

Pour une rotation d'un quart de tour, les dimensions de l'image résultat diffèrent de celles de l'image initiale lorsque cette dernière est rectangulaire. Pour créer la matrice-image vierge, on utilisera l'instruction `np.zeros((q, p))` pour créer une matrice vierge à q lignes et p colonnes.

4. Étant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat d'une rotation d'un quart de tour vers la droite.



5. En déduire une fonction `rotation(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la rotation d'un quart de tour.

On testera cette fonction en faisant apparaître à l'écran le résultat de `rotation(mario)` puis de ses itérées.

3 Transformation du photomaton

Cette transformation « réduit » la taille de l'image de moitié pour obtenir quatre morceaux analogues que l'on place en carré pour obtenir une image de même taille que l'image d'origine.

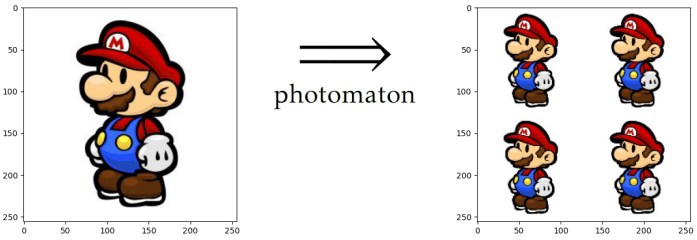
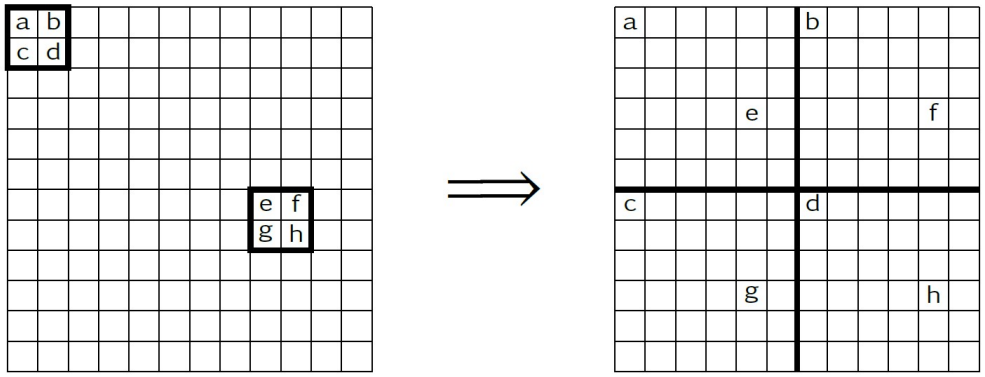


FIGURE 2 – Le plus célèbre des pompiers après une étape de photomaton.

Pour que cette transformation soit bijective, on a découpé l'image initiale en paquets carrés de quatre pixels (2×2) puis pour chaque paquet carré de quatre pixels, on utilise celui en haut à gauche pour l'image réduite en haut à gauche, celui en haut à droite pour l'image réduite en haut à droite, etc.



On notera que pour que cette transformation soit définie il est nécessaire que les dimensions horizontale et verticale de l'image soient paires, bref que l'image soit carrée.

6. À partir des coordonnées (x, y) d'un pixel de l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat. Il sera nécessaire de distinguer quatre cas suivant la parité de x et de y.
7. En déduire une fonction `photomaton(img)` qui prend en argument une matrice-image et renvoie une nouvelle matrice-image résultant de la transformée du photomaton de l'image initiale.
8. Rédiger un script Python pour visualiser les transformations successives de l'image *mario.png* jusqu'à revenir à celle-ci. Quelle est la période de la transformation du photomaton pour cette image ? On considère que les pixels sont revenus à leur place lorsque les pixels de même teinte sont aux mêmes endroits.

Plus généralement, si on considère une image de taille $p \times q$ il est possible de prouver que la période de la transformation du photomaton est le plus petit entier n pour lequel $p-1$ et $q-1$ divisent $2^n - 1$.

9. Rédiger une fonction `periode_photomaton(img)` qui prend en argument une matrice-image et retourne la période de la transformation du photomaton pour celle-ci.
10. Récupérer l'image *mario.png*. Quelle est la période de sa transformation du photomaton ?

On souhaite visualiser la 180^{e} itération de l'image *mario.png*, mais il n'est pas question de calculer les 179 images intermédiaires car le calcul serait trop long.

11. Rédiger une fonction `photomaton2(img,n)` qui prend en arguments une image et un entier n et qui retourne la n^{e} itération de l'image par la transformation du photomaton, en calculant celle-ci directement.
12. Utiliser cette fonction pour visualiser la 180^{e} itération de l'image *mario.png*. Pouvez-vous expliquer le résultat obtenu ?

4 Transformation du boulanger

Cette transformation s'appelle ainsi car elle s'apparente au travail du boulanger qui réalise une pâte feuilletée. L'image est tout d'abord aplatie. Puis elle est coupée en deux. Enfin la partie droite est placée sous la partie gauche en la faisant tourner de 180°. Pour que cette transformation soit bien définie, il est nécessaire là encore que les dimensions de l'images soient paires. L'image initiale est découpée en paquets carrés de quatre pixels puis ceux-ci sont entrelacés pour obtenir l'image intermédiaire aplatie. :

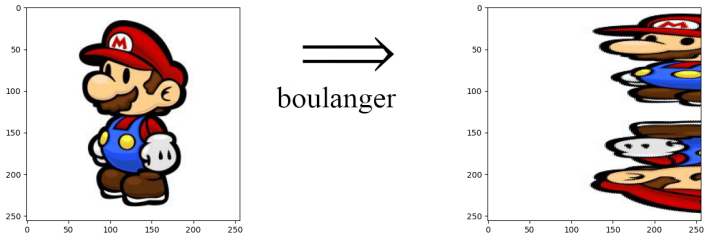
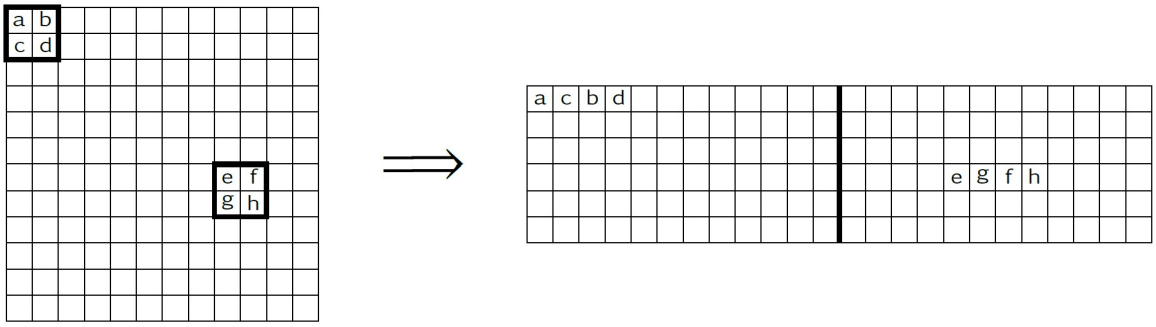


FIGURE 3 – Le plus célèbre des pompiers après une étape de boulanger.



13. Rédiger une fonction `boulanger(img)` qui prend en argument une image et retourne la transformée du boulanger de celle-ci.
14. Rédiger un script *Python* pour visualiser les transformations successives de l'image `mario.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du boulanger pour cette image ?

5 Et pour les plus rapides

Le calcul de la période de la transformation du boulanger est plus délicat à réaliser, et peut conduire à des valeurs très importantes notamment parce qu'on doit comparer toutes les teintes (c'est à dire trois valeurs) de tous les pixels à chaque étape. On l'obtient en déterminant tout d'abord la période de retour $r(x, y)$ de chaque pixel de coordonnées (x, y) puis en prenant le plus petit multiple commun de toutes ces valeurs $r(x, y)$. On ne s'attache donc ici plus uniquement à la teinte du pixel mais directement à l'évolution de ses coordonnées.

15. Écrire une fonction `periode_pixel(x,y,p,q)` qui prend en arguments les coordonnées (x, y) d'un pixel ainsi que la taille (p, q) de l'image auquel il appartient et qui retourne la période de retour de ce pixel à sa place initiale.
16. En déduire une fonction `tableau_des_periodes(img)` qui calcule le tableau des périodes de tous les pixels d'une image. On optimisera le temps de calcul en observant que tous les pixels appartenant à une même orbite [2] ont une période identique.
17. Rédiger enfin une fonction `periode_boulanger(img)` qui calcule la période de la transformation du boulanger d'une image. On pourra utiliser la fonction `gcd` du module `math` pour calculer le *PGCD* de deux entiers naturels. Quelle est la période de l'image *mario.png* ?
18. (a) Rédiger une fonction `boulanger2(img,n)` qui calcule la n^{e} transformation du boulanger d'une image.
    (b) Pour pouvoir utiliser de grandes valeurs de n sans que le temps d'attente soit rédhibitoire, il faudra calculer la position finale de chaque pixel (x, y) à l'aide du reste de la division euclidienne de n par $r(x, y)$, et traiter chaque pixel d'une même orbite dans le même temps. Afficher l'itération de rang 67911 de l'image *mario.png*. Quel est le pourcentage de pixels situés à leur place initiale ? Même question avec $n = 1496775000382576200$.

2. On appelle orbite d'un pixel l'ensemble des positions qu'il occupe durant les applications successives de la transformation du boulanger.