TD ITC n°3
Traitement d'images (ne niveaux de gris)

I. Contexte : représentation d'une image

1.1. Formats de fichiers image

Problème de la représentation des données

Un ordinateur traite de l'information sous forme numérique binaire. Comment coder les images à l'aide de codes binaires ?

Il existe deux modes de codage d'une image numérique :
- le mode matriciel (« bitmap ») : il repose sur le principe d'une grille de pixels ;
- le mode vectoriel (« vector ») : on décrit les propriétés mathématiques des formes de l'image.

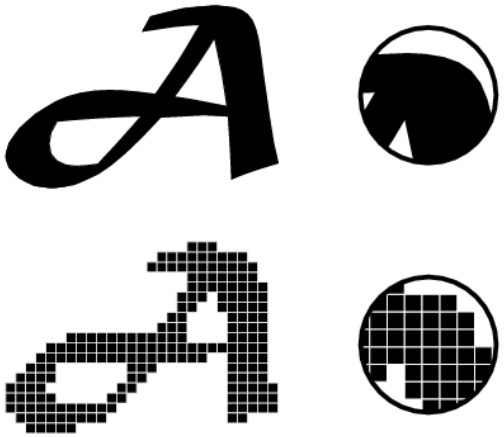


Figure 1 : En haut, une image vectorielle ; en bas, une image bitmap

Une image vectorielle peut être zoomée sans que cela n'altère la qualité du rendu. Ce mode est adapté aux formes et images qui ne sont pas trop complexes : logos, typographie, plans, cartes, etc. Les formats de fichiers vectoriels sont PostScript, PDF, SVG, etc.

Pour une image matricielle, le nombre de pixels utilisés par unité de longueur donne un rendu plus ou moins précis des formes de l'image. Un zoom trop important fait apparaître la pixellisation de l'image. Les formats de fichiers matriciels sont BMP, GIF, PNG, JPEG, etc.

On s'intéresse par la suite au codage des images par le mode matriciel.

1.2. Représentation dans ce TD

Une image matricielle est représentée en mémoire par un tableau de pixels. Chaque pixel a une position donnée (i, j), et pour une image en niveaux de gris, possède une valeur de gris. La structure est donc celle d'un tableau de dimension 2, de taille (n_ℓ, n_c) où n_ℓ est le nombre de lignes, n_c le nombre de colonnes, et la case (i, j) contient une valeur **entière** entre 0 et VAL_{MAX}. Ainsi `image[i][j]` contient le niveau de gris (entre 0 et VAL_{MAX}) dans le pixel de la ligne i, colonne j.

Dans toute la suite, « une image » est donc synonyme de « un tableau 2D d'entiers compris entre » 0 et VAL_{MAX}.

II. Contexte de travail dans ce TD

On fournit un module, et on demande d'exécuter le code suivant au début du fichier :

```
import td5_traitement_images_module_v39_12 as td5  # à adapter selon votre version
VAL_MAX = 255
image_1, image_2 = td5.récupérer_images_tests()
```

Celui-ci définit :
- une constante globale `VAL_MAX` pour la valeur maximale de niveau de gris (ici 255) ;
- deux tableaux 2D (nl, nc) qui représentent deux images : `image_1` et `image_2` ; ils pourront servir pour les tests ;
- une procédure `td5.afficheur` qui permet d'afficher côte à côte autant d'images qu'on le souhaite : elle prend en paramètre **une liste d'images** et les affiche côte à côte. Par exemple, on l'appelera avec

```
td5.afficheur([image_1, image_2])
```

Nous travaillerons avec des valeurs entières comprises dans l'intervalle $[0; \text{VAL}_{\text{MAX}}]$. Il faudra donc être particulièrement attentifs à utiliser des entiers et non des réels, notamment en cas de division ou de multiplication par un réel (on rappelle que la fonction `int` permet de faire une conversion vers un entier, et que la division d'entiers est l'opérateur `//`). Par exemple, pour diviser par deux la quantité de gris dans le pixel i, j, on pourra écrire

```
image[i][j] = int(image[i][j] / 2)
# ou bien
image[i][j] = image[i][j] // 2
```

mais

```
image[i][j] = image[i][j] / 2
```

n'aura pas le comportement attendu.

III. Création d'image à la main

1 – Écrire une fonction `créer_image_vide(nl, nc)` qui prend en paramètres le nombre de lignes n_ℓ et le nombre de colonnes n_c, et renvoie une image de taille $n_\ell \times n_c$, contenant des zéros.

2 – Écrire une fonction `créer_drapeau_tricolore(nl, nc)` qui prend en paramètres le nombre de lignes n_ℓ et le nombre de colonnes n_c, et renvoie une image de taille $n_\ell \times n_c$ en trois parties verticales de couleurs différentes, comme les drapeaux français, allemand, belge ou italien.

3 – Écrire une fonction `créer_dégradé_droit(nl, nc)` qui prend en paramètres le nombre de lignes n_ℓ et le nombre de colonnes n_c, et renvoie une image de taille $n_\ell \times n_c$ avec un dégradé : les pixels de la colonne de gauche valent 0, ceux de la colonne de droite valent $\mathrm{VAL}_{\mathrm{MAX}}$, et les valeurs augmentent progressivement de l'un à l'autre.

4 – Écrire une fonction `créer_dégradé_diagonale(nl, nc)` qui prend en paramètres le nombre de lignes n_ℓ et le nombre de colonnes n_c, et renvoie une image de taille $n_\ell \times n_c$ avec un dégradé : le pixel en haut à gauche vaut 0, celui en bas à droite vaut $\mathrm{VAL}_{\mathrm{MAX}}$, et les valeurs augmentent progressivement de l'un à l'autre selon la formule

$$f_{i,j} = \frac{(i+j) \times \mathrm{VAL}_{\mathrm{MAX}}}{n_\ell + n_c}$$

5 – On affichera les quatre images ainsi générées (de taille 200×300) côte à côte.

IV. Détection de contours

On peut détecter les contours d'une image en calculant le gradient des niveaux de gris (pour repérer les variations rapides) :

$$\overrightarrow{\mathrm{grad}}\, f = \frac{\partial f}{\partial x}\vec{\mathrm{u}}_x + \frac{\partial f}{\partial y}\vec{\mathrm{u}}_y$$

Pour estimer numériquement les dérivées, on utilise la formule suivante, pour tous les pixels (i, j) entre $i, 1 = 1$ et $i, j = n_x - 1, n_y - 1$:

$$\mathrm{grad}_x f_{ij} = \frac{f_{i+1,j} - f_{i-1,j}}{2} \qquad \mathrm{grad}_y f_{ij} = \frac{f_{i,j+1} - f_{i,j-1}}{2}$$

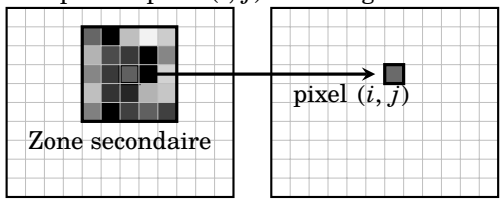
> **i** En utilisant cette formule, on obtient un gradient défini sur une taille $(n_\ell - 2) \times (n_c - 2))$ et non $n_\ell \times n_c$, puisque la formule ne gère pas les bords. On laissera donc les bords en noir.

6 – Écrire une fonction `calculer_gradient(image)` qui prend en paramètres une image de taille (n_ℓ, n_c) en niveaux de gris et qui renvoie une image de taille $n_\ell \times n_c$, représentant **la norme** du gradient de l'image entre les pixels $(1, 1)$ et $(n_\ell - 1, n_c - 1)$, et du noir sur les bords.

7 – Afficher côte à côte une image en niveaux de gris et le résultat du calcul de gradient. Conclure sur la détection de contours.

V. Floutage

On peut flouter l'image en moyennant localement chaque pixel avec les pixels voisins : on extrait autour du pixel (i, j) une zone secondaire de taille $a \times a$, puis on calcule la moyenne des valeurs des pixels dans cette zone secondaire centrée sur (i, j). On utilise alors cette nouvelle valeur pour le pixel (i, j) de l'image floutée.



8 – Écrire une fonction `extraire_zone_secondaire` qui prend en paramètres

- une image en niveaux de gris ;
- un entier `taille_zone` représentant la taille de la zone à extraire en pixels ;
- deux entiers i, j correspondant aux coordonnées du centre de la zone à extraire ;

et qui renvoie un tableau carré de côté `taille_zone`, ayant le contenu de la zone du tableau `image` centrée sur (i, j). On sera attentif à la gestion des bords de l'image (dans ce cas, la zone secondaire sera tronquée sur les bords).

9 – Écrire une fonction `calculer_moyenne` qui prend en paramètre un tableau à deux dimensions et renvoie la valeur moyenne des éléments du tableau.

10 – Écrire une fonction `floutage_moyenne_locale` qui calcule une image floutée à partir de l'image, en effectuant les moyennes sur des zones secondaires de taille `taille_flou` (valeur par défaut 5 pixels).

11 – Afficher côte à côte l'image en niveaux de gris, les contours et la version floutée.