

# Programmation séquentielle - TP noté

## Traitement d'images

02.05.2023 - rendu le 19.06.2023

## 1 Cahier des charges

### 1.1 Énoncé

Créer un programme de manipulation d'images en *niveaux de gris*. Ce programme devra permettre d'effectuer les opérations suivantes:

1. Lecture d'une image au format PGM depuis un format texte.
2. Sauvegarde d'une image dans le format PGM texte.
3. Effectuer des opérations sur les images:
  1. Symétrie *horizontale* et *verticale* d'une image.
  2. *Filtre* d'une image.
  3. *Rogner* une image.
  4. Calculer le *négatif* d'une image.
  5. Effectuer une rotation d'un quart de tour d'une image.
4. Compresser avec et sans perte une image à l'aide d'un arbre quaternaire.
5. Enregistrer et lire sur le disque des images compressées à l'aide d'un arbre quaternaire dans un format approprié.
6. Visualiser une image PGM obtenue à l'aide de la librairie SDL.

Ce programme doit disposer d'une interface interactive avec l'utilisateur (en ligne de commande ou graphique) permettant d'appliquer les opérations sur les images. Il doit être possible d'ouvrir des images les unes après les autres et d'y appliquer des transformation sans quitter le programme.

Il doit être possible d'ouvrir une image au format PGM et de l'enregistrer au format "arbre quaternaire" et vice versa.

Les filtres appliqués doivent pouvoir être saisis par l'utilisateur directement dans la CLI (command line interface), ou spécifiés dans des fichiers dont vous déciderez le format.

Lors d'une compression d'image, l'utilisateur doit pouvoir choisir si la compression est avec ou sans perte. Dans le cas d'une compression avec perte, il doit pouvoir choisir le seuil de compression.

Pour ce travail, en plus de la réalisation du programme de traitement d'images, vous devez utiliser le logiciel de gestion de version `git` et le projet à l'aide d'un `Makefile`.

## 2 Travail à rendre

- Ce travail est à réaliser individuellement.
- Vous devez rendre votre code sur votre répo `git` au plus tard le 19 juin 2023.
- Communiquez l'adresse de votre dépôt `git` lorsque vous commencez à travailler sur ce projet.
- Suite à ce rendu, vous devrez effectuer une présentation orale de votre travail d'une durée d'une dizaine de minutes avec des diapositives qui sera suivie par une séance de questions d'une dizaine de minutes également.

### 2.0.1 Représentation informatique des images

Dans ce travail pratique, une image n'est rien d'autre qu'une matrice de nombres entiers, au sens du type `matrix` que vous avez implémenté dans un précédent travail. Pour simplifier les images seront uniquement représentées en *niveaux de gris*. Ainsi chaque élément de la matrice représente un pixel. Les valeurs des pixels devront être limitées entre 0 et une valeur maximale `max`.

Pour simplifier encore, les images seront supposées être données au format PGM. Le format PGM (portable graymap file format) est un format de fichier très simple permettant de stocker des images en niveau de gris. Nous utiliserons le format PGM `textuel`. Vous trouverez une définition du format sur la page [https://fr.wikipedia.org/wiki/Portable\\_pixmap](https://fr.wikipedia.org/wiki/Portable_pixmap).

Le format PGM implique la lecture dans un premier temps d'une entête contenant le texte `P2` sur la première ligne, puis les dimensions de l'image sur la deuxième ligne. Sur la troisième ligne se trouve le niveau de gris maximal. Finalement, les pixels de l'image en format texte sont stockés dans les lignes restantes.

Afin de tester vos fonctions vous pouvez utiliser l'image du mandrill.

## 3 Les transformations d'images

### 3.1 Le négatif

Le négatif d'une image consiste à *inverser* la valeur des pixels de l'image par rapport à la valeur maximale permise. Ainsi si on représente `max` niveaux de gris, le négatif d'un pixel de niveau de gris, `p`, est donné par `max-p`.

### 3.2 Les symétries

Une symétrie d'axe vertical ou d'axe horizontale consiste à inverser l'ordre des pixels autour de l'axe vertical ou horizontal respectivement.

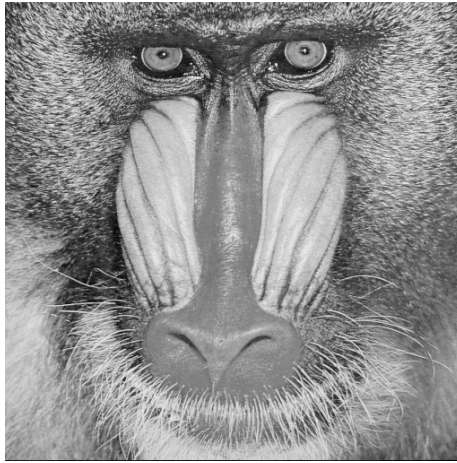


FIGURE 1 – Image d'un mandrill.

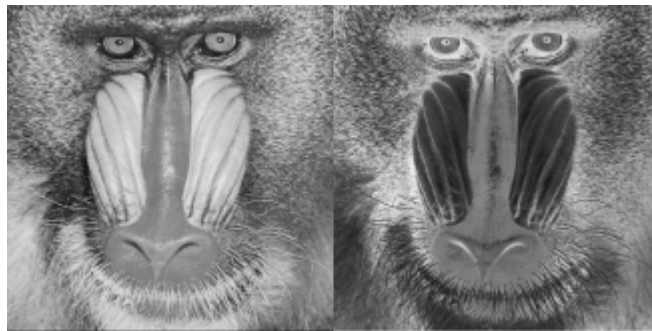


FIGURE 2 – L'original (image de gauche) et le négatif (image de droite) de la photo du mandrill.

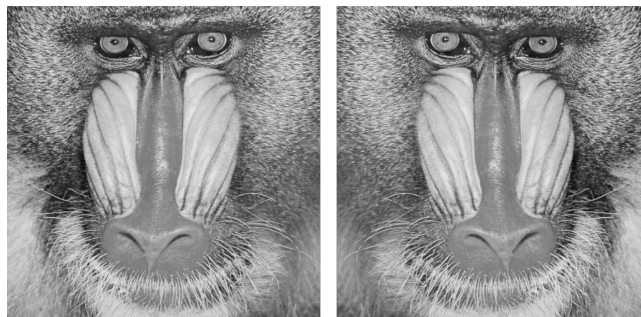


FIGURE 3 – Exemple de symétrie d'axe horizontal.

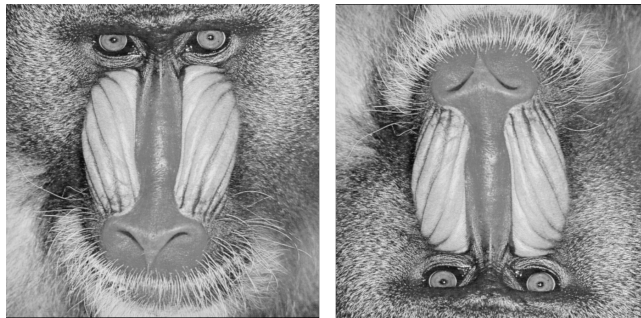


FIGURE 4 – Exemple de symétrie d’axe vertical.

### 3.3 La rotation

La rotation consiste à faire “tourner” les pixels d’une image autour de son point central. Ici, on se limitera à des rotations d’un quart de tour dans le sens horaire ou anti-horaire, tel qu’illustré dans le cours sur les arbres quaternaires.

### 3.4 Rogner

Le rognage d’une image est une opération assez simple. Elle consiste à extraire une sous partie rectangulaire des pixels de l’image d’origine.

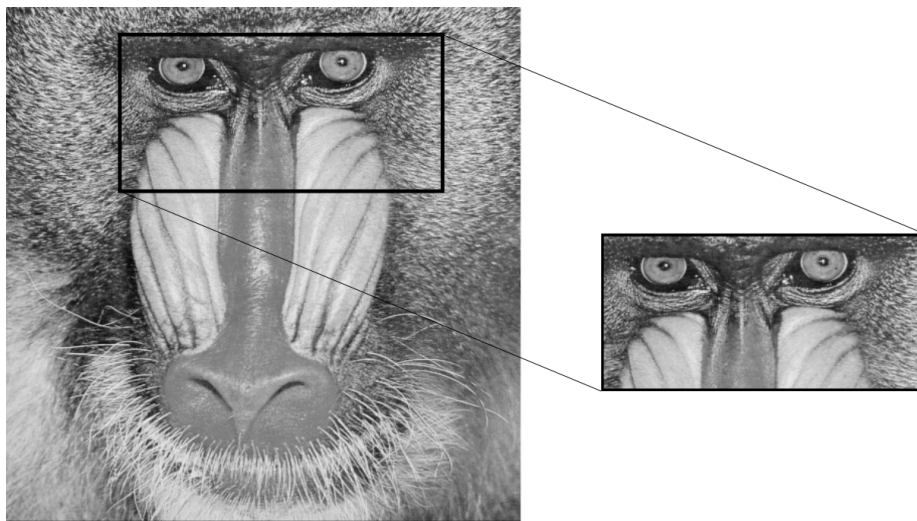


FIGURE 5 – Rognage de la photo de mandrill.

### 3.5 Convolution et filtres<sup>1</sup>

Explication dans la documentation de gimp: <https://docs.gimp.org/2.8/fr/plugin-convmatrix.html>

1. Repris du cours de math de N. Eggenberg et O. Malaspinas

Les matrices de convolutions sont particulièrement utiles dans le traitement d'images. On les appelle également *noyaux* ou *masques*. L'image traitée est obtenue en faisant la *convolution* entre la matrice et l'image. Ce genre d'opération est effectuée tout le temps dans vos téléphones portables pour appliquer des filtres sur vos photos (floutage, vieillissement, ...).

Il existe une grande quantité de noyaux (vous pouvez en trouver des exemple sur la page [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))), mais l'opération pour effectuer le traitement de l'image reste toujours la même. Une image peut se représenter sous la forme d'une matrice  $m \times n$

$$\underline{\underline{A}} = \begin{pmatrix} a_{11} & \dots & a_{1,j-1} & a_{1,j} & a_{1,j+1} & \dots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i-1,1} & \dots & a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} & \dots & a_{i-1,n} \\ a_{i,1} & \dots & a_{i,j-1} & a_{i,j} & a_{i,j+1} & \dots & a_{i,n} \\ a_{i+1,1} & \dots & a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} & \dots & a_{i+1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,j-1} & a_{m,j} & \dots & \dots & a_{mn} \end{pmatrix}. \quad (1)$$

Si nous choisissons une matrice de convolution,  $\underline{\underline{C}}$ ,  $3 \times 3$  (cela se généralise pour toutes les tailles), de la forme

$$\underline{\underline{C}} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}, \quad (2)$$

nous pouvons écrire la transformation de tous les éléments  $a_{i,j}$  de la matrice  $\underline{\underline{A}}$ , que nous noterons  $b_{i,j}$ , comme

$$\begin{aligned} b_{i,j} &= \left[ \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} * \begin{pmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{pmatrix} \right] \\ &= c_{11}a_{i-1,j-1} + c_{12}a_{i-1,j} + c_{13}a_{i-1,j+1} \\ &\quad + c_{21}a_{i,j-1} + c_{22}a_{i,j} + c_{23}a_{i,j+1} \\ &\quad + c_{31}a_{i+1,j-1} + c_{32}a_{i+1,j} + c_{33}a_{i+1,j+1}. \end{aligned} \quad (3)$$

On voit donc que la convolution est une combinaison linéaire de tous les éléments d'une sous matrice de  $\underline{\underline{A}}$  dont les poids sont donnés par la matrice de convolution. La somme des éléments de la matrice de convolution est en général de 1 (on dit qu'elle est normalisée à 1) pour éviter de modifier la luminosité des pixels.

### 3.5.1 Illustration (Floutage)

Si la matrice de convolution est donnée par

$$\underline{\underline{C}} = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad (4)$$

on voit que son effet est de moyenner la valeur de chaque pixel avec tous ses voisins

$$b_{ij} = \frac{1}{9}(a_{i-1,j-1} + a_{i-1,j} + a_{i-1,j+1} + a_{i,j-1} + a_{i,j} + a_{i,j+1} + a_{i+1,j-1} + a_{i+1,j} + a_{i+1,j+1}). \quad (5)$$

On peut aisément constater que ces opérations sont mal définies pour les pixels se trouvant sur les bords de l'image (aux endroits où les pixels n'ont pas suffisamment de voisins pour effectuer l'opération de convolution). Il existe différentes solutions possibles:

1. Diminuer la taille de l'image traitée. L'image sera plus petite que l'image de départ de  $n$  pixels, où  $n \times n$  est la taille de la matrice de convolution.
2. La taille de l'image est étendue en copiant le dernier pixel sur tous les pixels manquants dans une direction donnée.

On utilisera ici cette deuxième solution.

### 3.5.2 Remarques

Il se peut que suite à l'application d'un filtre, vous dépassiez la valeur maximale autorisée pour un pixel, ou obteniez une valeur plus petite que zéro. Dans ces cas il faudra ramener les valeurs dans l'intervalle  $[0, \text{max}]$ :

1. Si la valeur du pixel est inférieure à 0, alors le pixel vaudra 0.
2. Si la valeur du pixel est supérieure **max**, alors le pixel vaudra **max**.

Par ailleurs, il se peut que la valeur des pixels ne soit plus entière. Il faudra donc tronquer le nombre obtenu pour en faire un entier.

## 3.6 Afficher l'image avec la librairie SDL

Vous devez afficher les images chargées et les transformations obtenues à l'aide de la librairie SDL.

Vous trouverez toutes les fonctions nécessaires dans l'exemple se trouvant sur [ce lien](#). Cet exemple affiche du bruit (des valeurs de gris aléatoires sur un grand nombre de pixels). Vous **devez** réutiliser les fonctions se trouvant dans cet exemple qui sont là pour vous faciliter la vie (et non tenter de réinventer la roue).

Le code dont vous devez vous inspirer est dans le fichier `gfx_example.c` (qui utilise `gfx.h/c`). Outre les fonctions de création/destruction du contexte, la fonction importante est `render()` qui affiche un pixel à la position `x, y` à un niveau de gris `color`, à l'aide de la fonction `put_pixel()`.

## 3.7 Compression à l'aide d'arbres quaternaire

Les arbres quaternaires permettent de compresser une image en ne stockant qu'une valeur pour un cadran de celle-ci. Cette compression est sans perte si les pixels dans le cadran choisi sont tous de la même couleur, ou avec perte si tous

les pixels ne sont pas de la même couleur. L'utilisateur doit pouvoir choisir le seuil de compression.

Vous devez mettre au point un format de fichier textuel permettant de représenter une image sous forme d'arbre quaternaire.

Votre programme doit être capable de :

- Transformer une image sous forme de matrice en arbre quaternaire pour le compresser avec ou sans perte.
- Transformer une image sous forme d'arbre quaternaire en matrice.
- Enregistrer sur le disque une image sous forme d'arbre quaternaire.
- Lire du disque une image sous forme d'arbre quaternaire.

Effectuez également une comparaison de la taille des fichiers sur le disque avec quelques images et quelques seuils de compression différents, afin de déterminer si la compression est avantageuse en terme d'espace disque.