======================================

Manipuler des images en Python, avec OpenCV

C.Turrier – 20 novembre 2020

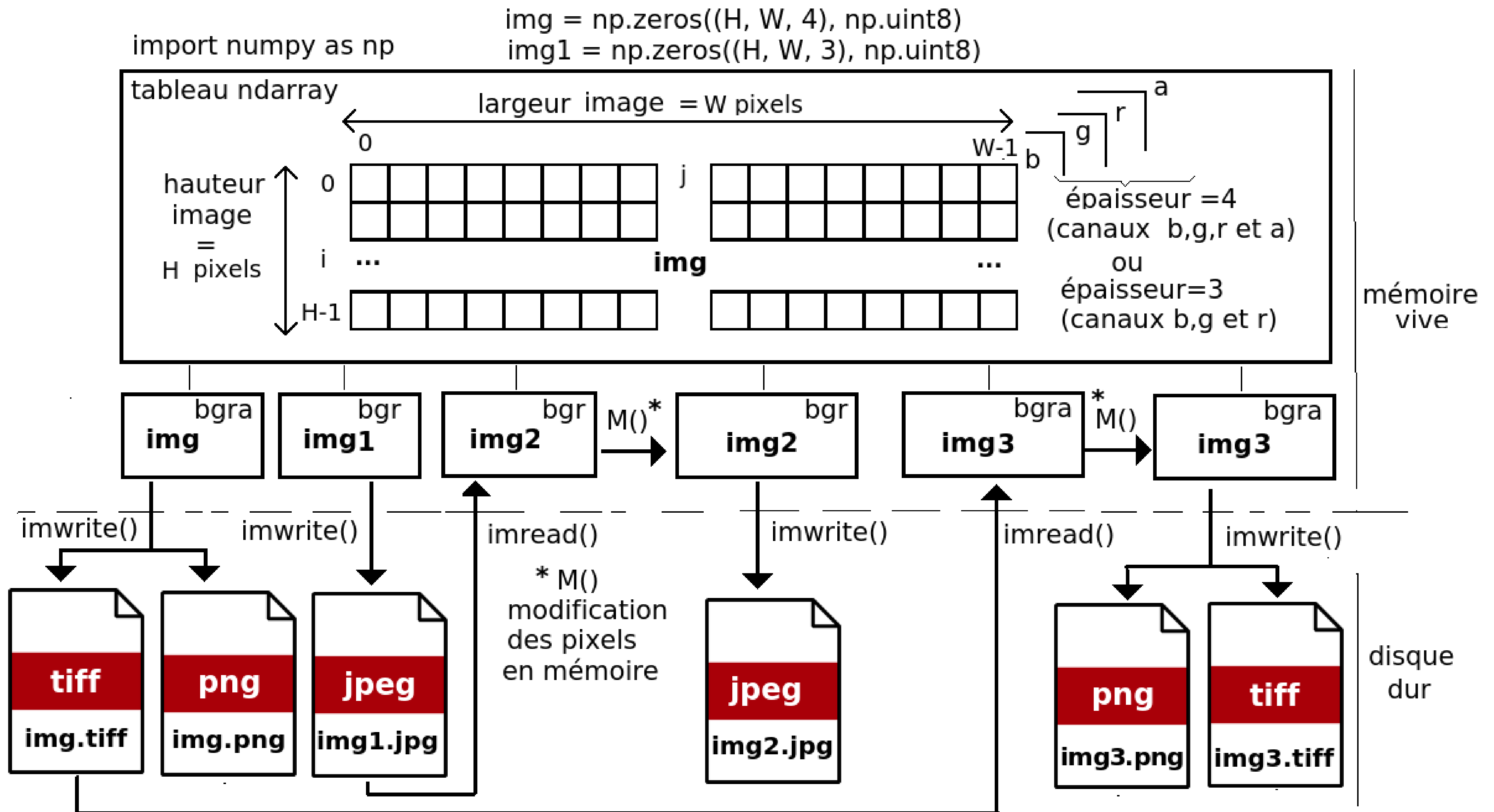
======================================

Le présent document et le programme exemple associé (pocv.py) montrent comment créer, charger, modifier, sauvegarder des images **png**, **tiff** et **jpg,** avec le langage **python** et la librairie **opencv** .

Pour manipuler des images, en langage **python,** avec **opencv**, on utilise :

- ✔ des tableaux de la classe **ndarray** du module **numpy** (et non pas la classe **Mat** comme en **C++**) pour créer, ranger et manipuler des images en mémoire ;
- ✔ la fonction **cv2.imread**() pour charger en mémoire, dans un tableau **ndarray**, des fichiers images (**jpg**, **png**, **tiff**...) ;
- ✔ la fonction **cv2.imshow**() pour afficher, dans une fenêtre, des images rangées en mémoire dans un tableau **ndarray** ;
- ✔ la fonction **cv2.imwrite**() pour enregistrer, dans des fichiers images (**jpg**, **png**, **tiff**,..), des images qui se trouvent en mémoire dans un tableau **ndarray.**

Le schéma suivant montre l'articulation des tableaux **ndarray** de **numpy**, des fonctions de **opencv** et des fichiers images **tiff**, **png** et **jpg**



Les exemples de code qui suivent fonctionnent comme cela est illustré dans ce schéma.

Rappel

Le «**slicing**», très utilisé en langage **python**, consiste à sélectionner une plage d'éléments dans un objet (tableau, liste,...).
La plage d'éléments sélectionnée (slice) peut ensuite être utilisée pour un traitement donné.

Exemples de slices souvent rencontrées en python :

t[start:stop] # définit les éléments du tableau t allant de t[start] à t[stop-1] inclus
t[start:] # définit les éléments du tableau t allant de t[start] jusqu'à la fin du tableau t
t[:stop] # définit les éléments du tableau t allant de t[0] jusqu'à t[stop-1] inclus
t[:] # définit tous les éléments du tableau t

1) Création d'une image mémoire img, bgra, H=5 et W=100 pixels

```
import cv2
import numpy as np
img = np.zeros((5, 100, 4), np.uint8)
bgra_color=(0, 0, 255, 128) #couleur rouge  semi-transparente
img[:] = bgra_color #slicing sur tout le tableau img
```

Pour créer l'image **img** en mémoire, on définit :

- ✗ un tableau **ndarray** img de 5 lignes (hauteur de l'image en nombre de pixels), 100 colonnes (largeur de l'image en nombre de pixels) et 4 couches en épaisseur (couches b, g, r, et a);
- ✗ une variable couleur bgra_color, de type **n-uplet**, initialisée avec une couleur rouge semi-transparente

On remplit tous les pixels du tableau **img** avec la couleur bgra_color

2) Modification des valeurs b, g, r et a des pixels (0,0) et (0,1) de img

```
img[0,0]=(255,255,255,255) #blanc opaque
img[0,1]=(255,0,0,64)      #bleu transparent
```

Les valeurs contenues dans les cases du tableau **img** peuvent être modifiées de deux façons différentes :

1) Modification par pixel
img [i,j] = (b,g,r,a)
avec 0⩽ i ⩽ H-1=4, 0⩽ i ⩽ W-1=99, 0⩽ b ⩽ 255, 0⩽ g ⩽ 255, 0⩽ r ⩽ 255 et 0⩽ a ⩽ 255,

2) Modification par pixel et par composante de couleur
img [i,j,0] = b, img [i,j,0] = g, img [i,j,0] = r, img [i,j,0] = a,

Dans le cas présent, on modifie les composantes de couleurs des pixels (0,0) et (0,1) à qui ont atribue respectivement une couleur blanc opaque et bleu transparent

3) Affichage de img dans la fenêtre fen

```
cv2.namedWindow("fen", cv2.WINDOW_AUTOSIZE);
cv2.imshow("fen", img);
```

La fonction **namedWindow**() crée une fenêtre et la fonction **imshow**() affiche l'image mémoire **img** dans cette fenêtre. Le paramètre cv2.WINDOW_AUTOSIZE conduit à ce que la taille de la fenêtre s'ajuste automatiquement à celle de l'image. Avec le paramètre cv2.WINDOW_NORMAL, on peut redimensionner la fenêtre.

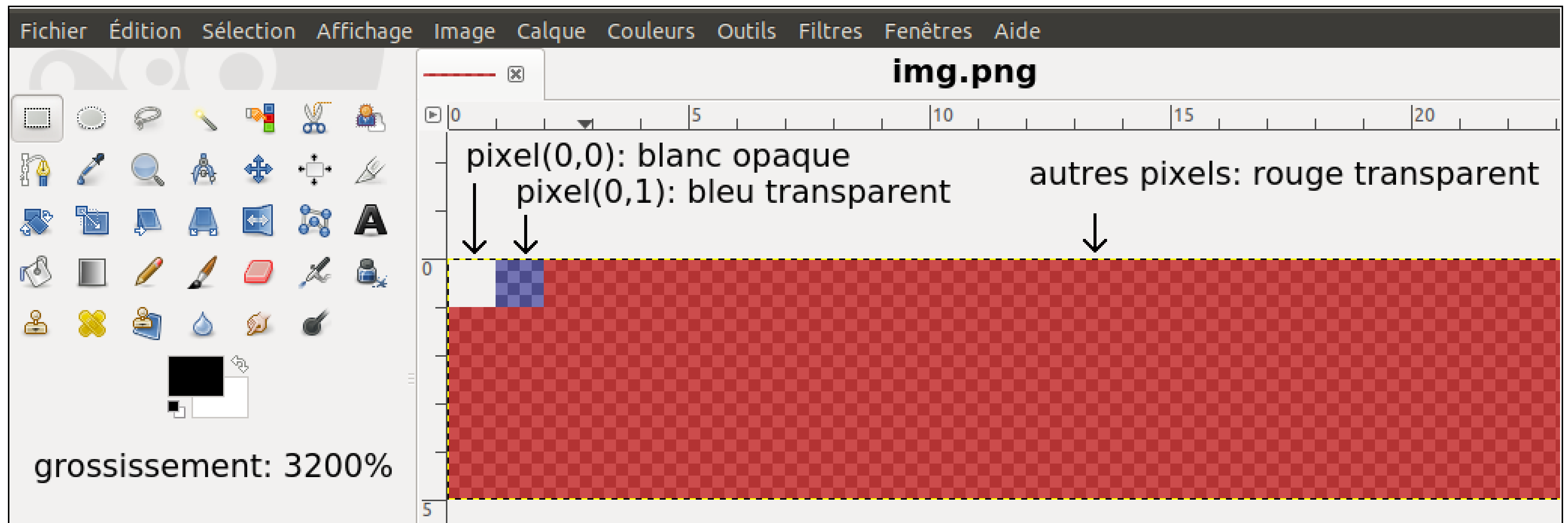
On peut remarquer au passage que la fonction imshow n'affiche pas la transparence à l'écran, comme peut le faire **Gimp** par exemple.

4) Enregistrement de img dans le fichier img.png

```
cv2.imwrite('img.png', img)
```

L'enregistrement de l'image mémoire **img** dans un fichier **png** se fait avec la fonction **imwrite**().
En ouvrant l'image enregistrée 'img.png' avec **Gimp**, on peut constater que tout est conforme.

On constate ainsi que bien que la fonction **imshow**() n'affiche pas la transparence à l'écran, la transparence est bien enregistrée dans le fichier img.png

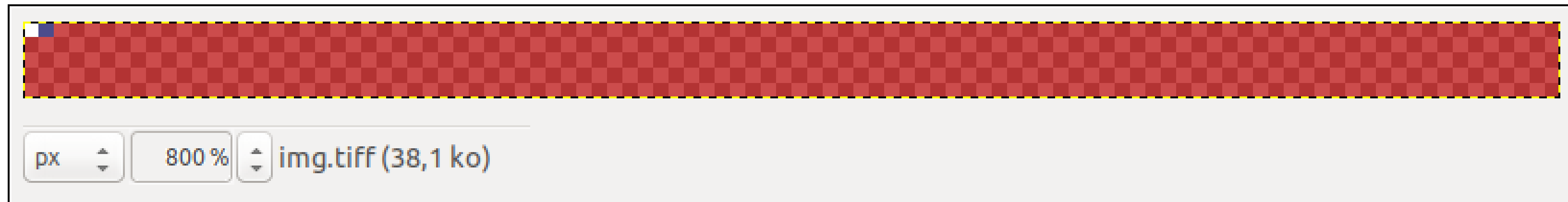


5) Enregistrement de img dans le fichier img.tiff, non compressé

```
tiff_params = ( 259, 1)   # tag 259=1 =>pas de compression
cv2.imwrite('img.tiff', img, tiff_params)
```

Le format **tiff** offre la possibilité de pouvoir enregistrer l'image **sans compression**, c'est à dire avec les pixels décrits directement avec leurs valeurs r,g,b et a comme en mémoire.
Pour cela il faut cependant définir la valeur 1 pour le **tag 259** qui est associé à la compression dans la norme tiff. En l'absence de cette précision, **opencv** enregistre l'image tiff par défaut avec compression.

On peut vérifier, à l'aide de **Gimp** et d'un **éditeur hexadécimal**, que tout est conforme.



blanc opaque — pixel(0,0) : r g b a

bleu semi transparent — pixel(0,1) : r g b a

header

```
00000000: 49 49 2a 00 d8 07 00 00 ff ff ff ff 00 00 ff 40
00000016: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
00000032: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
00000048: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
00000064: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
00000080: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
00000096: ff 00 00 80 ff 00 00 80 ff 00 00 80 ff 00 00 80
```

autres pixels

rouge semi transparent

1.2 Créer une image BGRA et

...

img.tiff

6) Création de l'image mémoire img1, bgr, H=10 et W=20 pixels

```
img1 = np.zeros((5, 100, 3), np.uint8)
bgr_color=(0, 0, 255) #couleur rouge
img1[:] = bgr_color
```

On définit un tableau ndarray **img1** de 5 lignes (hauteur de l’image en nombre de pixels), 100 colonnes (largeur de l’image en nombre de pixels) et 3 couches en épaisseur (couches b, g et r).

On remplit tous les pixels du tableau **img1** avec la couleur bgr_color, rouge (sans transparence)

7) Enregistrement de img1 dans img1.jpg

```
cv2.imwrite('img1.jpg', img1, [cv2.IMWRITE_JPEG_QUALITY, 100])
```

Le paramètre cv2.IMWRITE_JPEG_QUALITY permet de préciser le niveau de qualité souhaité (allant de 0 à 100) pour l’image jpg. En l’absence de ce paramètre, **opencv** adopte une valeur par défaut.

8) Chargement de img1.jpg en mémoire dans img2

```
img2 = cv2.imread('img1.jpg',cv2.IMREAD_UNCHANGED)
```

Pour charger en mémoire l’image contenue dans un fichier, on utilise la fonction **cv2.imread**().
Si l’image ne se trouve pas dans le répertoire du programme, il faut préciser son chemin (chemin '/home/img1.jpg' par exemple).

Le second paramètre de la fonction permet de préciser sous quelle forme on veut charger l’image :

- ✔ cv2.IMREAD_COLOR (ou 1) : l’image est chargée sans prendre en compte l’éventuel canal de transparence pouvant exister dans le fichier (option par défaut) ;
- ✔ cv2.IMREAD_GRAYSCALE (ou 0) : l’image est chargée en nuance de gris uniquement, même si elle est enregistrée en couleur dans le fichier ;

✔ cv2.IMREAD_UNCHANGED (ou -1) : l'image est chargée telle quelle. Si elle possède un canal de transparence enregistré dans le fichier, celui-ci est donc chargé également.

9) Modification des pixels de img2 en mémoire

```
for j in range (100) : #colonne j allant de 0 à 99
     for i in range (20) : #ligne i allant de 0 à 19
          img2[i,j]=(0,255,0) #couleur verte
```

On place une couleur verte dans tous les pixels de l'image **img2** en mémoire.

Remarque : Il faut veiller à faire varier les indices de lignes et de colonnes dans les bonnes plages de valeurs, sinon on obtient une erreur «out of range !»

10) Enregistrement de img2 dans img2.jpg

```
cv2.imwrite('img2.jpg', img2, [cv2.IMWRITE_JPEG_QUALITY, 100])
```

En ouvrant, avec **Gimp** ou avec une visionneuse d'images, les images enregistrées dans les fichiers **img1.jpg** et **img2.jpg**, on peut vérifier que le résultat obtenu correspond bien à ce qui est attendu.

11) Chargement de img.png en mémoire dans img3

```
img2 = cv2.imread('img.png',cv2.IMREAD_UNCHANGED)
```

12) Modification des pixels de img3 en mémoire

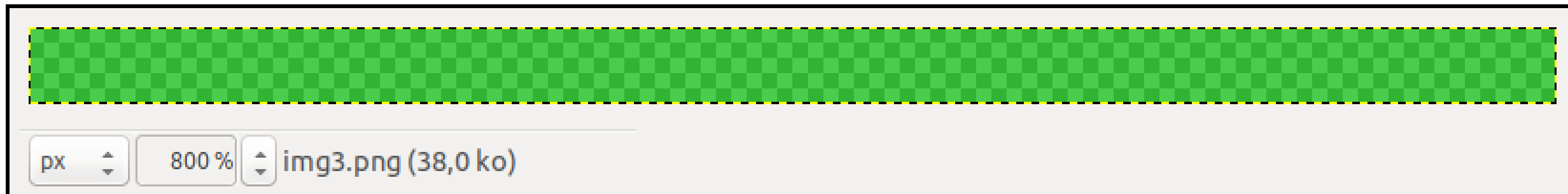
```
for j in range (100) : #colonne j allant de 0 à 99
     for i in range (5) : #ligne i allant de 0 à 4
          img3[i,j]=(0,255,0,128) # vert semi transparent
```

On remplace, dans **img3** en mémoire, la couleur initiale des pixels i,j (rouge semi transparent) par une couleur vert semi-transparent.

13) enregistrement de img3 dans img3.png

```
cv2.imwrite('img3.png', img3)
```

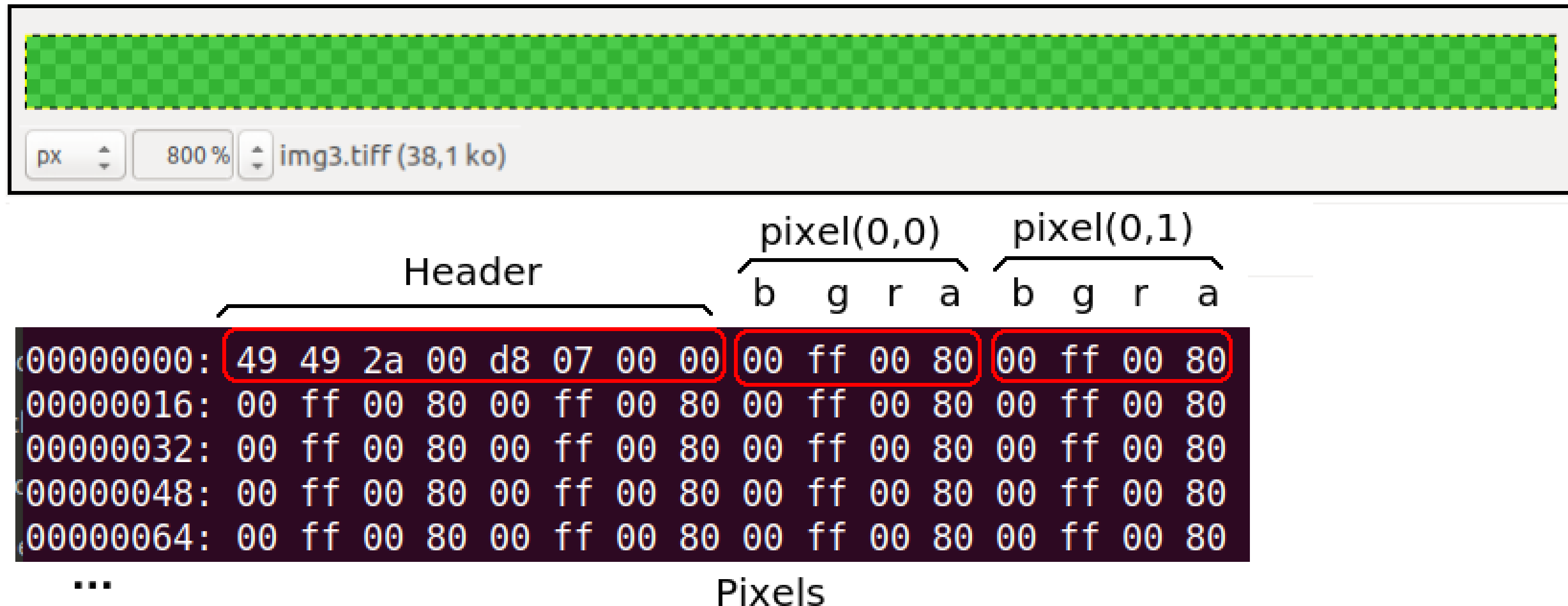
On enregistre l'image mémoire **img3** dans le fichier **img3.png**. En ouvrant ce fichier avec Gimp, on vérifie que tout est conforme.



14) enregistrement de img3 dans img3.tiff non compressé

```
tiff_params = ( 259, 1)   # tag 259=1 =>pas de compression
cv2.imwrite('img3.tiff', img3, tiff_params)
```

On enregistre l'image mémoire **img3** dans le fichier **img3.tiff** non compressé. En ouvrant ce fichier avec Gimp et avec un éditeur hexadécimal, on vérifie que tout est conforme.



15) fin du programme

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.waitKey () est une fonction qui est en attente de l'appui d'une touche du clavier.
cv2.destroyAllWindows () détruit toutes les fenêtres créées. Pour détruire une fenêtre spécifique, on utilise la fonction cv2.destroyWindow () en lui passant en argument le nom la fenêtre à détruire.