==========

Manipuler des images en C++, avec OpenCV

C.Turrier – 20 novembre 2020

==========

1) La classe Mat (matrice)

La classe **Mat** permet de placer des objets image en mémoire. Elle comprend deux parties: un en-tête et un tableau de pixels.
L' en-tête contient des informations telles que :

- la largeur et la hauteur du tableau de pixels (cf m.cols et .rows)
- le nombre de canaux de l'image (3 pour rgb et 4 pour rgba)
- le type de couleurs de l'image (CV_8UC4, CV_8UC3,...)

On peut créer un objet *m* de la classe **Mat**, à l'aide d'une instruction ayant la forme suivante :
Mat m(H, W, CV_8UC4, Scalar(b,g,r,a));

int W,H; // largeur et hauteur de l'image
unsigned char b, g, r, a; // valeurs (de 0 à 255) des composantes de couleur r,g,b,a de l'image
CV_8UC4 signifie **8** bits par composante de couleur, de type **u**nsigned **c**har, **4** canaux (rgba)

a=255 donne une image entièrement opaque et a=0 donne une image entièrement transparente.
CV_8UC3 signifierait qu'on utilise un type unsigned char de 8 bits par composante de couleur, avec 3 composantes de couleur b, g et r.

Le tableau suivant donne la correspondance entre les types de couleurs des images et les valeurs entières de ces types.

	C1	C2	C3	C4
CV_8U	0	8	16	24
CV_8S	1	9	17	25
CV_16U	2	10	18	26
CV_16S	3	11	19	27
CV_32S	4	12	20	28
CV_32F	5	13	21	29
CV_64F	6	14	22	30

Exemple

```
Mat m(300, 200, CV_8UC4, Scalar(0,0,255,255)); //crée matrice de pixels rgba rouges opaques
printf("type= %d\n", m.type());
> 24
Mat m(300, 200, CV_8UC3, Scalar(0,0,255)); //crée matrice de pixels rgb rouges
printf("type= %d\n", m.type());
> 16
```

2) Les fonctions namedWindow() et imshow()

Il est utile de pouvoir afficher, directement dans une fenêtre, l'image définie en mémoire dans un objet **m** de la classe **Mat**.

La fonction **namedWindow**("m", WINDOW_AUTOSIZE); permet de créer une fenêtre, destinée à afficher l'image chargée dans un objet **m**.

Le paramètre WINDOW_AUTOSIZE conduit à l'ajustement automatique de la fenêtre aux dimensions de l'image qu'elle contient.

La fonction **imshow**("m", m); affiche l'image contenue dans un objet **m**, dans la fenêtre qui lui est associée.

- ✔ Si l'image est de type 8-bits unsigned, elle est affichée en l'état;
- ✔ Si l'image est de type 16-bits unsigned ou 32-bits integer, les composantes de couleur sont divisées par 256. La plage de valeurs [0, 256*256] se trouve alors automatiquement mise en correspondance (mappée) avec la plage [0,255];
- ✔ Si l'image est de type 32-bits ou 64-bits floating-point, les composantes de couleur sont multipliées par 255. La plage de valeurs [0,1] se trouve alors automatiquement mise en correspondance (mappée) avec la plage [0,255].

On peut constater que la fonction **imshow**() n'affiche pas le canal transparence même si celui-ci existe dans l'image **m** en mémoire. En revanche un outil comme **Gimp** par exemple affiche bien le canal de transparence présent dans la sauvegarde de l'image **m**, dans un fichier **png** ou **tiff**.

La fonction **waitKey**(0); crée une mise en attente du programme, jusqu'à ce que une touche clavier soit pressée.

Exemple

```
namedWindow("mafenetre2", WINDOW_AUTOSIZE);
```

```
imshow("mafenetre2",m2);
```

3) La fonction imwrite()

La fonction **imwrite**(filename, m, params); permet d'enregistrer l'image mémoire **m** dans le fichier **filename** (fichier png, tiff, jpg,...).

Les paramètres de l'image (c'est à dire «compression» ou non pour un fichier **png** ou **tiff**, et niveau de «qualité» pour un fichier **jpg**) doivent être placés dans une structure de type **vector<int>** params, avant l'appel de la fonction **imwrite()**.

Exemple

```
vector<int> png_params;
png_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
png_params.push_back(0);//0 (sans compression) -> 9 (compression max)
imwrite("f.png", m, png_params);
```

4) La fonction imread()

La fonction **Mat** m= **imread**(filename, param) permet de charger en mémoire, dans l'objet **m** de la classe **Mat**, l'image contenue dans le fichier **filename** (fichier png, tiff, jpg,...).

Le paramètre **param** dépend du type d'image. Il doit être égal à **-1** pour la prise en compte du canal de transparence d'une image **png**.

Exemple

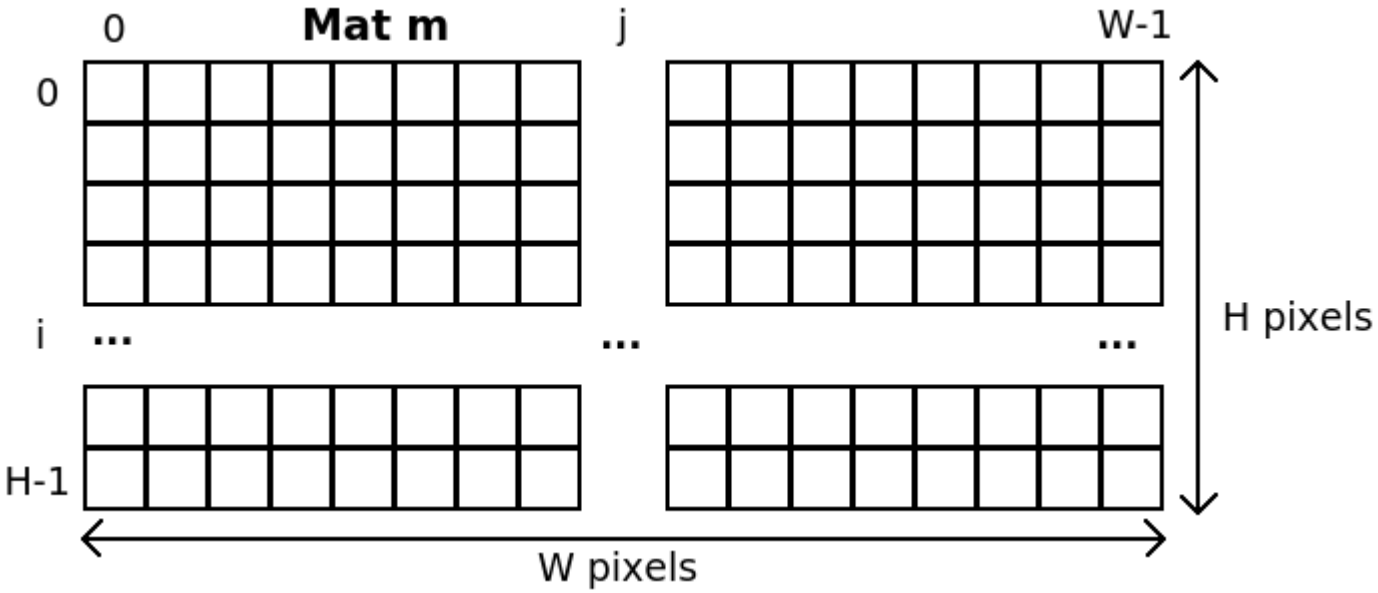
```
Mat m3 = imread("f.png", -1); //-1 car canal alpha
```

5) La méthode Mat m.at<Vec..>()

La méthode **Mat m.at<Vec..>**() permet de modifier les valeurs des pixels en mémoire d'un objet image **m** de la classe **Mat**.

Exemple

```
Mat m3 = imread("f.png", -1); //-1 car canal alpha
W =m3.cols; H=m3.rows;
for(int i =0; i < H; i++){
for(int j = 0; j < W; j ++){
Vec4b &v = m3.at<Vec4b>(i,j);
v[0] = 0;   //b   (0 à 255)
v[1] = 0;   //g   (0 à 255)
v[2] = 255; //r   (0 à 255)
v[3] = 128; //a   (0 à 255)
}}
```



Une fois que les valeurs de couleurs, des pixels d'une image mémoire **m**, ont été modifiées (à l'aide de la méthode **m.at**), on enregistre le résultat dans un fichier (un fichier **png** ou **tiff** si l'image possède un canal de transparence ou un fichier **jpg** sinon).

Le schéma suivant illustre les opérations qui sont effectuées dans le programme **tocv.cpp** ci-joint.