# Raspberry Pi OpenCV La classe Mat

## 1 Introduction

La classe **Mat** permet de représenter une image en mémoire sous la forme d'un objet matrice à deux dimensions. Les points forts de cette classe sont nombreux :

- L'allocation de mémoire pour les objets de type Mat est automatiquement réalisée selon la taille de l'image chargée copiée ou clonée.
- La libération de la mémoire est aussi automatique lorsque la matrice n'est plus référencée.

```
Mat ImgSource;    // création d'une matrice vide seulement les entêtes
ImgSource = imread("Penguins.jpg", CV_LOAD_IMAGE_COLOR);
```

- L'opérateur d'assignation (=) et le constructeur de copie ont pour effet de copier uniquement l'entête de la matrice, les données ne sont pas recopiées.

```
Mat  dest1(ImgSource);       // utilise le constructeur de copie pour Dest1
Mat  dest2 = dest1;          // Opérateur d'assignation.
```

Mat est une classe qui possède les informations concernant l'entête (les dimensions (le nombre de colonne et de rangée, la méthode de rangement en mémoire et un pointeur sur les éléments qui contiennent les pixels.

Vous vous demandez peut-être à quoi bon plusieurs objets qui pointe tous au même endroit sur la matrice de pixel. L'avantage devient clair lorsqu'on fait référence à un sous-ensemble de la matrice de pixel. Par exemple, pour créer une région d'intérêt (**R**egion **O**f **I**nterest en anglais), vous n'aurez qu'à créer
un nouvel entête avec par exemple :

```
Mat ImgDest (ImgSource, Rect(10,10,100,100) );
```

La libération de la mémoire est automatique, c'est le dernier objet à utiliser la mémoire qui sera en charge de le faire. À cet effet, il existe un attribut « refcount » qui garde un compte du nombre d'objet qui font référence à la matrice de pixel.



Lorsqu'on fait une copie d'un objet Mat, ce compte est incrémenté et, à l'inverse, décrémenté lorsqu'on n'utilise plus cet objet.

## 2 Copier les données de la matrice

il est possible de copier la matrice de pixel dans un autre objet de type Mat. Pour ce faire, il existe 2 méthodes qui se nomme « **clone** » et « **copyTo** ». Voici un exemple :

```
Mat ImgDest6 = ImgSource.clone();

Mat ImgDest7 ;
ImgSource.copyTo( ImgDest7 ) ;
```

## 3 L'opérateur <<

On peut afficher les valeurs de la matrice en utilisant l'opérateur « **<<** » (le même que celui que vous utilisez dans un « cout » ). Cependant, veuillez prendre note que cet opérateur ne fonctionne que pour les matrices à 2 dimensions.

Exemple :

```
Mat ImgDest4 = ImgSource(Rect(100, 300, 5 , 5));
cvtColor( ImgDest4, ImgGray, CV_RGB2GRAY);
cout << ImgGray << endl;
```

la dernière ligne donne l'affichage suivant :

```
[254, 228, 121, 114, 174;
 253, 206, 114, 128, 152;
 245, 169, 108, 132, 165;
 230, 140, 112, 139, 154;
 202, 145, 121, 103, 148]
```

## *4 Création d'objet Mat par son constructeur*

Nous pouvons créer des matrices

- **avec le constructeur de Mat.**

  Exemple: pour une matrice M carré de 2 par 2, dont chaque valeur vaut 255

```
Mat M(2,2, CV_8U, Scalar(255));
```

Les type OpenCV :

**CV_8UC1** -> valeur avec 1 octet (8) non signé (U) et sur un seul canal (C1). Convient parfaitement aux images en ton de gris.
**CV_8UC3** -> valeur avec 1 octet(8) non signé (U) et sur 3 canaux (C3). Convient parfaitement pour une images couleur RVB.
**CV_16SC1** -> valeur avec 2 octets(16) signé (S) et sur 1 canal (C1). Convient surtout aux images temporaires provenant de calcul comme les convolutions de Sobel ou Laplace.
**CV_32FC** -> valeur avec 4 octets (32) de type « float » (F) et possédant un canal (C).
**CV_64FC** -> valeur sur 8 octets (64) de type double et possédant un canal.

- **avec la méthode create**

```
Mat M;
M.create(4,4, CV_8UC(2));
cout << M << endl;
```

## *5 Création de Matrices particulières*

Les initialisateurs zeros(), ones() et eyes() provoque l'assignation des matrices particulières.

Zeros() :

```
Mat O = Mat::zeros(2, 2, CV_32F);
cout << "O = " << endl << " " << O << endl << endl;
```

matrice de taille 2,2 remplie de 0



ones() :

```
Mat O = Mat::ones(2, 2, CV_32F);
cout << "O = " << endl << " " << O << endl << endl;
```

matrice de taille 2,2 remplie de 1

eye():

```
Mat E = Mat::eye(4, 4, CV_64F);
cout << "E = " << endl << " " << E << endl << endl;
```

matrice de taille 4,4 avec des 1 sur la diagonale et des zéros ailleurs

## *5 Accès aux pixels / éléments d'une matrice*

Accès individuel à un pixel :

Pour obtenir la valeur d'un pixel, vous devez connaître le type d'image et le nombre de canaux. Voici un exemple pour une image RGB (trois canaux) et pour les coordonnées de pixels x et y:

```
scalar intensity = Img.at<uchar>(2, 2);
cout << intensity << endl;
```

On obtient à l'affichage : [254, 0, 0, 0]

Un autre exemple avec une variable de type vecteur 3 octets Vec3b :

```
Vec3b intensity = ImgSource.at<Vec3b>(20, 20);
int blue = intensity.val[0];
int green = intensity.val[1];
int red = intensity.val[2];
cout << "bleu : " << dec << blue << " vert : " << green << "rouge : " << red << endl;
```