

ensiie



Introduction à Open CV en C++

<http://docs.opencv.org/>

OpenCV

- Open Computer Vision
 - Vadim Pisarevsky
 - Historique
 - 1999 : début du projet : Intel research initiative
 - 2006 : Version 1, C : p.ex.
 - 2008/9 : Version 2, C++
 - 2014/11 : Version 3.0 Beta
 - 2017/04 : Version 2.4.9 !!!
 - 2017/12 : Version 3.2
 - Langages
 - C, puis C++, puis Python, puis Java
 - Variantes :
 - lpp (Intel Performance Primitives)
 - tbb (Thread Building Blocks)
 - gpu (Graphics Processing Unit) / ocl (Open Computing Library)
 - Plateformes
 - Desktop : Windows, Linux, Mac OS
 - Handheld : Android, iOS



Structures / Types de base

Matrices

Vecteurs

Scalaires

Points

...



Matrices

■ Mat : `class Mat`

- Matrices n dimensionnelles [nbLines × nbCols × nbChannels] utilisées pour les matrices et les images

■ Type des éléments

- CV_8UC(n) : unsigned char (uchar) × n Channels

- (p.ex. CV_8UC3 pour les images RGB ou BGR)

- CV_16SC(n) : signed int × n Channels

- CV_32FC(n) : float × n Channels

- CV_64FC(n) : double × n Channels

■ Mat_<_Tp> : `template<typename _Tp> class Mat_ : public Mat`

- Forme template des matrices n-dimensionnelles

■ Matx<_Tp,nl,nc> : `template<typename _Tp, int m, int n> class Matx`

- « Petites » matrices bidimensionnelles



Vecteurs

- `Vec` : `template<typename _Tp, int cn> class Vec : public Matx<_Tp, cn, 1>`
- Vecteurs
 - `Vec3b` : `typedef Vec<uchar, 3> Vec3b`; vecteurs de 3 bytes
 - `Vec2d` : `typedef Vec<double, 2> Vec2d`; vecteurs de 2 doubles
- `Scalar_` : `template<typename _Tp> class Scalar_ : public Vec<_Tp, 4>`
 - `typedef Scalar_<double> Scalar`;
 - Valeurs scalaires adaptées (en partie) aux matrices n-dimensionnelles



Points

■ Complexes : `template<typename _Tp> class Complex`

■ `_Tp re, im;`

■ `typedef Complex<float> Complexf;`

■ `typedef Complex<double> Complexd;`

■ Points 2D : `template<typename _Tp> class Point_`

■ `_Tp x, y;`

■ `typedef Point_<int> Point;`

■ Points 3D : `template<typename _Tp> class Point3_`

■ `_Tp x, y, z;`

■ `typedef Point3_<int> Point3i;`

■ `typedef Point3_<float> Point3f;`

■ `typedef Point3_<double> Point3d;`



Sélecteurs / Matrices

- Tailles : `template<typename _Tp> class Size_`
 - `_Tp width, height;`
 - `typedef Size_<int> Size;`
- Séquences : `class Range`
 - `int start, end;`
 - Pour extraire des sous-matrices
- ROI : Regions of Interest
 - Rect : `template<typename _Tp> class Rect_`
 - `_Tp x, y, width, height;`
 - `typedef Rect_<int> Rect;`
 - RotatedRect : `class RotatedRect`
 - `Point2f center;`
 - `Size2f size;`
 - `float angle;`



Matrices : créations

// Création de matrice

```
Mat image(240, 320, CV_8UC3);
```

// Réallocation de matrice

```
image.create(480, 640, CV_8UC3);
```

// Création et remplissage par une valeur constante

```
Mat A33(3, 3, CV_32FC1, Scalar(5));
```

```
Mat B33(3, 3, CV_32FC1);
```

```
B33 = Scalar(5);
```

```
Mat C33 = Mat::ones(3, 3, CV_32F) * 5.0;
```

```
Mat D33 = Mat::zeros(3, 3, CV_32F) + 5.0;
```

// Création et remplissage par des valeurs spécifiques

```
double a = CV_PI/3;
```

```
Mat A22 = (Mat_<float>(2, 2) << cos(a), -sin(a), sin(a), cos(a));
```

```
float B22data[] = {cos(a), -sin(a), sin(a), cos(a)};
```

```
Mat B22 = Mat(2, 2, CV_32FC1, B22data).clone();
```



Matrices : conversions

// Conversion vers/depuis d'autres structures sans copier les données

```
Mat image_alias = image;
```

```
float* Idata=new float[480*640*3];
```

```
Mat I(480, 640, CV_32FC3, Idata);
```

```
vector<Point> iptvec(10);
```

```
Mat iP(iptvec); // iP - 10x1 CV_32SC2 matrix
```

// Conversion vers/depuis les anciennes structures OpenCV 1.x

```
IplImage* oldC0 = cvCreateImage(cvSize(320, 240), 16, 1);
```

```
Mat newC = cvarrToMat(oldC0);
```

```
IplImage oldC1 = newC;
```

```
CvMat oldC2 = newC;
```

// Conversion vers/depuis d'autres structures en copiant les données

```
Mat image_copy = image.clone();
```

```
Mat P(10, 1, CV_32FC2, Scalar(1, 1));
```

```
vector<Point2f> ptvec = Mat_<Point2f>(P);
```



Matrices : accès aux éléments (1/4)

// Accès aux éléments d'une matrice

```
A33.at<float>(i,j) = A33.at<float>(j,i)+1; // A33.type() == CV_32FC1
```

```
Mat dyImage(image.size(), image.type()); // image.type() == CV_8UC3
```

```
for (int y = 1; y < image.rows-1; y++)
```

```
{
```

```
    Vec3b* prevRow = image.ptr<Vec3b>(y-1);
```

```
    Vec3b* nextRow = image.ptr<Vec3b>(y+1);
```

```
    for (int x = 0; x < image.cols; x++)
```

```
    {
```

```
        for (int c = 0; c < 3; c++)
```

```
        {
```

```
            dyImage.at<Vec3b>(y, x)[c] = saturate_cast<uchar>(nextRow[x][c] - prevRow[x][c]);
```

```
        }
```

```
    }
```

```
}
```

Utilisation de l'opérateur [] de Vec

```
Mat_<Vec3b>::iterator it = image.begin<Vec3b>();
```

```
Mat_<Vec3b>::iterator itEnd = image.end<Vec3b>();
```

```
for(; it != itEnd; ++it)
```

```
{
```

```
    (*it)[1] ^= 255;
```

```
}
```



Matrices : accès aux éléments (2/4)

```
/*
 * Pixel iterators :
 * 90 ms/img
 */
```

src1, src2 & dst sont des images de type **CV_8UC3** de même taille

```
Mat_<Vec3b>::const_iterator src1It = src1.begin<Vec3b>();
Mat_<Vec3b>::const_iterator src1End = src1.end<Vec3b>();
Mat_<Vec3b>::const_iterator src2It = src2.begin<Vec3b>();
Mat_<Vec3b>::iterator dstIt = dst.begin<Vec3b>();
for (; src1It != src1End; ++src1It, ++src2It, ++dstIt)
{
    Vec3b p1 = *src1It;
    Vec3b p2 = *src2It;
    // compute dst from computation on p1 & p2
    *dstIt = p1 ... p2;
}
```



Matrices : accès aux éléments (3/4)

```
/*  
 * Random access operator:  
 * img.at<Vec3b>(i,j) : 83 ms/img  
 */  
for(int i = 0; i < src1.rows; ++i)  
{  
    for (int j = 0; j < src1.cols; ++j)  
    {  
        Vec3b p1 = src1.at<Vec3b>(i,j);  
        Vec3b p2 = src2.at<Vec3b>(i,j);  
        // compute dst from computation on p1 & p2  
        dst.at<Vec3b>(i,j) = p1 ... p2;  
    }  
}
```



Matrices : accès aux éléments (4/4)

```
/*  
 * Pointer ops: img.ptr<Vec3b>(i,j) :  
 * ~50 ms/img  
 */  
for(int i = 0; i < src1.rows; ++i)  
{  
    for (int j = 0; j < src1.cols; ++j)  
    {  
        const Vec3b * p1 = src1.ptr<Vec3b>(i,j);  
        const Vec3b * p2 = src2.ptr<Vec3b>(i,j);  
        // compute dst from computation on p1 & p2  
        *(dst.ptr<Vec3b>(i,j)) = *p1 ... *p2;  
    }  
}
```



Copies et extractions de sous-matrices

// Copies et extraction de sous-matrices

```
Mat src(Size(320, 240), CV_16SC1);
```

```
Mat dst;
```

*// Copie d'une matrice vers une autre (evt avec
réallocation de dst)*

```
src.copyTo(dst);
```

// Changement d'échelle et conversion vers un autre type

```
int type = CV_32FC1;
```

```
double scale = 0.5;
```

```
double shift = 137.8;
```

```
src.convertTo(dst, type, scale, shift);
```

// Copie d'une matrice

```
Mat m = dst;
```

// Copie profonde d'une matrice

```
Mat n = m.clone();
```

// Changement de dimensions et/ou de nombre de canaux

// sans copier les données

```
int nch = 2;
```

```
int nrows = 120;
```

```
Mat mReshaped = m.reshape(nch, nrows);
```

// Extraction d'une ligne ou d'une colonne

```
Mat mr = m.row(i);
```

```
Mat mc = m.col(i);
```

// Extraction d'un ensemble de lignes ou de colonnes

```
int i1 = 10;
```

```
int i2 = 20;
```

```
int j1 = 10;
```

```
int j2 = 20;
```

```
Mat subRowM = m.rowRange(Range(i1,i2));
```

```
Mat subColM = m.colRange(Range(j1,j2));
```

*// Extraction d'une diagonale (>0 : au dessus, <0 en
dessous)*

```
int diagLevel = 1;
```

```
Mat mDiag = m.diag(diagLevel);
```

// extraction d'une sous matrice

```
Mat subM = m(Range(i1,i2),Range(j1,j2));
```

```
Rect roi(10, 10, 20, 20);
```

```
Mat mROI = m(roi);
```



Gestion mémoire avec « Mat »

// Création d'une matrice de doubles

Mat A(4, 4, CV_64F);

// Création d'une autre matrice pointant vers les mêmes données

// B.data == A.data : Pas de copie de données

Mat B = A;

// Création d'une matrice ligne pointant vers la 2nde ligne de A

// Pas de copies de données

Mat C = B.row(1);

// Création d'une matrice contenant une copie séparée des données

Mat D = B.clone();

// Copie de la 4ème ligne de B dans C

// <==> copie de la 4ème ligne de A dans sa 2ème ligne

B.row(3).copyTo(C);

// partage des données de D vers A : A.data pointe vers D.data

// L'ancienne matrice A est toujours référencée par B et C

A = D;

// Vidage de la matrice B (B.data pointe vers NULL & B.empty() == true)

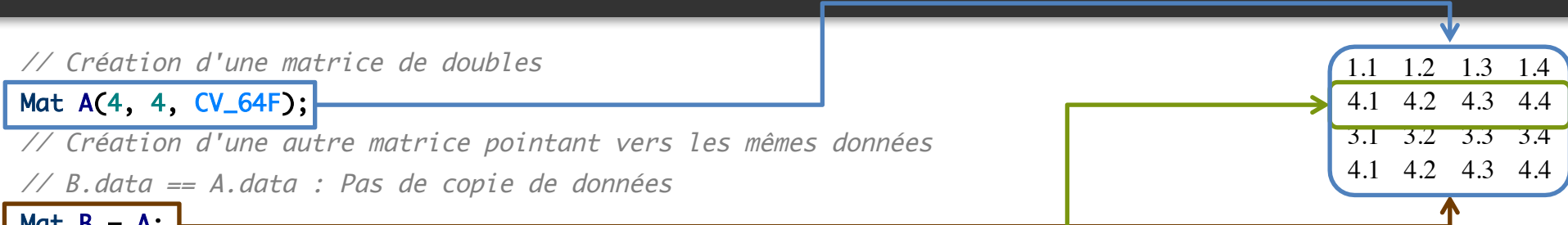
// L'ancienne version de A reste référencée par C

B.release();

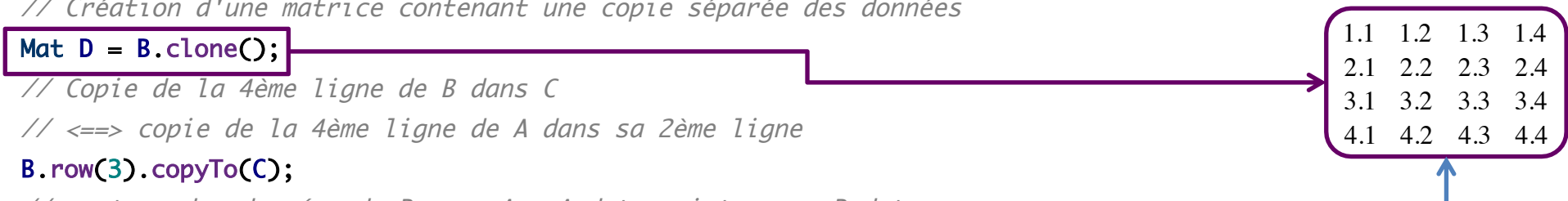
// Finalement création d'une copie distincte de C dans lui même

// L'ancienne version de A n'est plus référencée, elle est donc désallouée

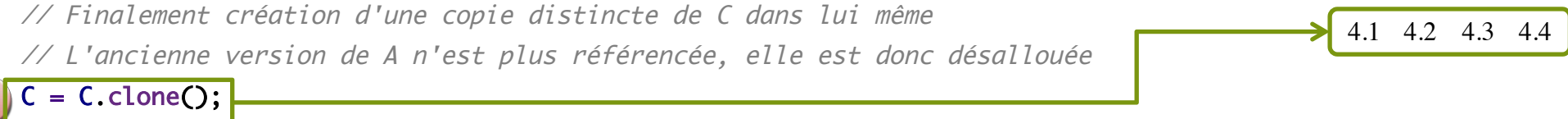
C = C.clone();



1.1	1.2	1.3	1.4
4.1	4.2	4.3	4.4
3.1	3.2	3.3	3.4
4.1	4.2	4.3	4.4



1.1	1.2	1.3	1.4
2.1	2.2	2.3	2.4
3.1	3.2	3.3	3.4
4.1	4.2	4.3	4.4



4.1	4.2	4.3	4.4
-----	-----	-----	-----



HighGui

Interface graphique simplifiée et lecture/écriture de médias

- GUI
- Capture



Fenêtres

■ Création de fenêtre :

- `void namedWindow(const string & winname, int flags);`
- `namedWindow("input", CV_WINDOW_AUTOSIZE |
CV_GUI_NORMAL);`

■ Affichage d'une image dans une fenêtre :

- `void imshow(const string& winname,
InputArray mat);`
 - `imshow("input", image);`
- Proxy datatype for passing Mat's and vector<>'s as input parameters

■ Destruction(s) de fenêtre(s)

- `void destroyWindow(const string & winname);`
- `void destroyAllWindows();`



Clavier / Souris

▣ Touches du clavier

▣ `int waitKey(int delay=0);`

```
char keycode = (char) waitKey(10);  
switch (keycode)  
{  
    case 'i' :  
        // ...  
        break;
```

▣ Click souris

▣ `typedef void (*MouseCallback)(int event, int x, int y,
int flags, void* param);`

▣ `void setMouseCallback(const string & windowName,
MouseCallback onMouse, void* param=0);`



Acquisition

□ Lecture d'image

- `Mat imread(const string & filename, int flags=1);`

- `Mat firstFrame = imread(inputFileName, CV_LOAD_IMAGE_COLOR);`

□ Ouverture d'un flux vidéo

- `class VideoCapture`

- `virtual bool open(const string & filename);`

- `virtual bool open(int device);`

- `virtual VideoCapture & operator >> (Mat & image);`



Example

```
VideoCapture cap;  
  
if (argc > 1)  
{  
    // opens file or stream  
    cap.open(string(argv[1]));  
}  
else  
{  
    // opens default camera  
    cap.open(0);  
}  
  
if (!cap.isOpened())  
{  
    return EXIT_FAILURE;  
}  
  
Mat frame;  
namedWindow("video", CV_WINDOW_AUTOSIZE);
```

```
for (;;)   
{  
    cap >> frame;  
  
    if (frame.data == NULL)  
    {  
        break;  
    }  
  
    imshow("video", frame);  
  
    if (waitKey(30) >= 0)  
    {  
        break;  
    }  
}  
  
cap.release();  
frame.release();  
  
return EXIT_SUCCESS;
```



Sauvegarde

■ Ecriture d'image

```
■ bool imwrite(const string & filename, InputArray img,  
               const vector<int> & params=vector<int>());  
  
■ vector<int> fileOptions;  
  fileOptions.push_back(CV_IMWRITE_JPEG_QUALITY);  
  fileOptions.push_back(80);  
  imwrite("Capture.jpg", inFrame, fileOptions);
```

■ Ecriture d'un flux vidéo

```
■ class VideoWriter  
  
  ■ virtual bool open(const string & filename, int fourcc,  
                     double fps, Size frameSize, bool isColor=true);  
  
  ■ virtual VideoWriter & operator << (const Mat & image);
```



QT + OpenCV

- QT
 - UI
 - Signaux / Slots
- OpenCV
 - Capture
 - Traitements



Image Processor

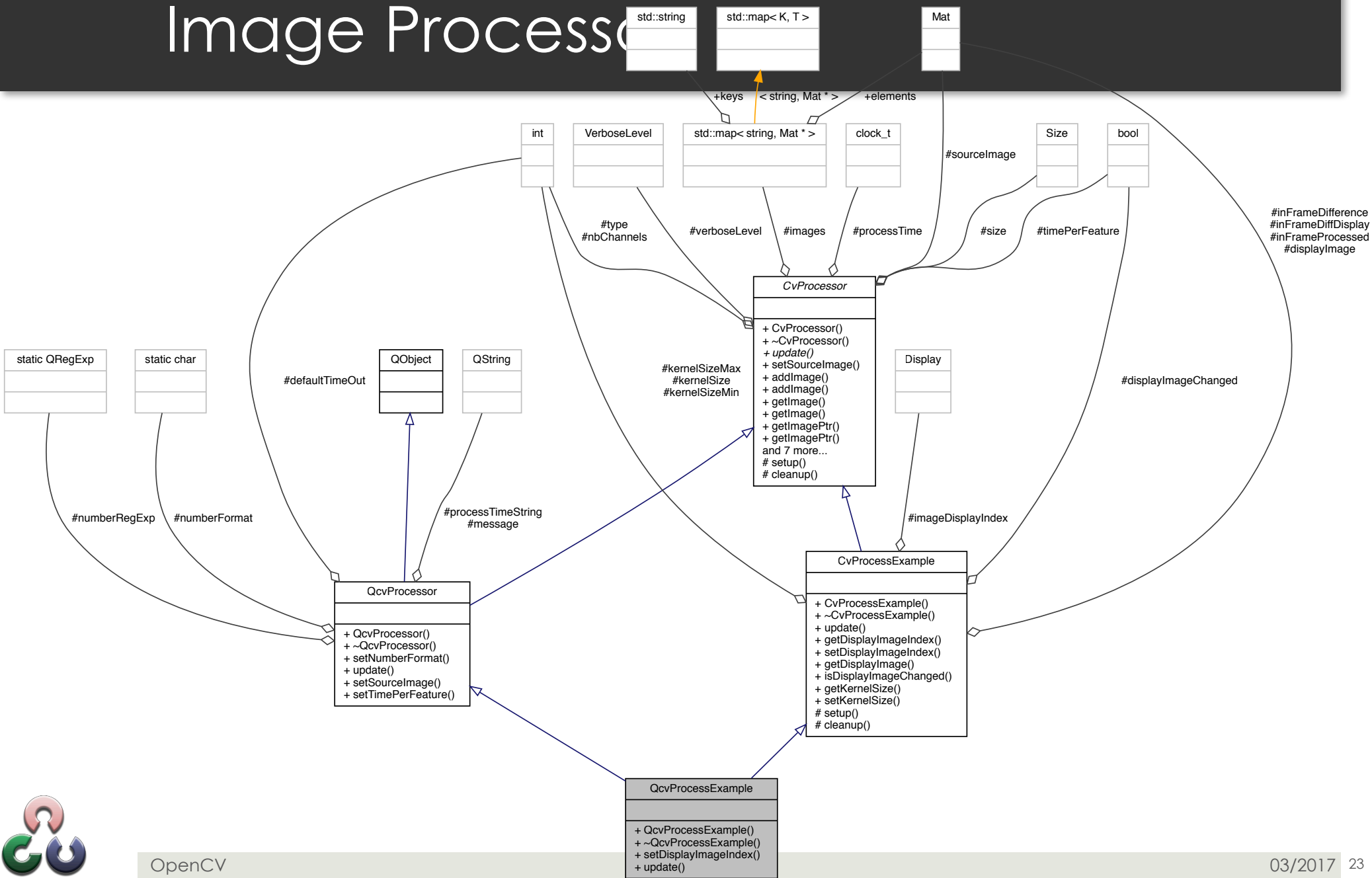


Image Capture

