

# OpenCV 2.4 Cheat Sheet (C++)

The OpenCV C++ reference manual is here:

<http://docs.opencv.org>. Use **Quick Search** to find descriptions of the particular functions and classes

## Key OpenCV Classes

|                           |  |
|---------------------------|--|
| <a href="#">Point_</a>    | Template 2D point class  |
| <a href="#">Point3_</a>   | Template 3D point class  |
| <a href="#">Size_</a>     | Template size (width, height) class  |
| <a href="#">Vec</a>       | Template short vector class  |
| <a href="#">Matx</a>      | Template small matrix class  |
| <a href="#">Scalar</a>    | 4-element vector   |
| <a href="#">Rect</a>      | Rectangle  |
| <a href="#">Range</a>     | Integer value range  |
| <a href="#">Mat</a>       | 2D or multi-dimensional dense array (can be used to store matrices, images, histograms, feature descriptors, voxel volumes etc.) |
| <a href="#">SparseMat</a> | Multi-dimensional sparse array   |
| <a href="#">Ptr</a>       | Template smart pointer class   |

## Matrix Basics

### Create a matrix

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

### [Re]allocate a pre-declared matrix

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

### Create a matrix initialized with a constant

```
Mat A33(3, 3, CV_32F, Scalar(5));
```

```
Mat B33(3, 3, CV_32F); B33 = Scalar(5);
```

```
Mat C33 = Mat::ones(3, 3, CV_32F)*5.;
```

```
Mat D33 = Mat::zeros(3, 3, CV_32F) + 5.;
```

### Create a matrix initialized with specified values

```
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_32F, B22data).clone();
```

### Initialize a random matrix

```
randu(image, Scalar(0), Scalar(256)); // uniform dist
```

```
randn(image, Scalar(128), Scalar(10)); // Gaussian dist
```

### Convert matrix to/from other structures

(without copying the data)

```
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
```

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

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

```
IplImage oldC1 = newC; CvMat oldC2 = newC;
```

### ... (with copying the data)

```
Mat newC2 = cvarrToMat(oldC0).clone();
```

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

### Access matrix elements

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

```
Mat dyImage(image.size(), image.type());  
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]);  
}  
Mat_<Vec3b>::iterator it = image.begin<Vec3b>(),  
    itEnd = image.end<Vec3b>();  
for(; it != itEnd; ++it)  
    (*it)[1] ^= 255;
```

## Matrix Manipulations: Copying, Shuffling, Part Access

|   |   |
|---|---|
| <a href="#">src.copyTo(dst)</a>                     | Copy matrix to another one  |
| <a href="#">src.convertTo(dst,type,scale,shift)</a> | Scale and convert to another datatype                                   |
| <a href="#">m.clone()</a>                           | Make deep copy of a matrix  |
| <a href="#">m.reshape(nch,nrows)</a>                | Change matrix dimensions and/or number of channels without copying data |
| <a href="#">m.row(i), m.col(i)</a>                  | Take a matrix row/column  |
| <a href="#">m.rowRange(Range(i1,i2))</a>            | Take a matrix row/column span   |
| <a href="#">m.colRange(Range(j1,j2))</a>            |   |
| <a href="#">m.diag(i)</a>                           | Take a matrix diagonal  |
| <a href="#">m(Range(i1,i2),Range(j1,j2))</a>        | Take a submatrix  |
| <a href="#">m(roi)</a>                              |   |
| <a href="#">m.repeat(ny,nx)</a>                     | Make a bigger matrix from a smaller one                                 |
| <a href="#">flip(src,dst,dir)</a>                   | Reverse the order of matrix rows and/or columns                         |
| <a href="#">split(...)</a>                          | Split multi-channel matrix into separate channels                       |
| <a href="#">merge(...)</a>                          | Make a multi-channel matrix out of the separate channels                |
| <a href="#">mixChannels(...)</a>                    | Generalized form of split() and merge()                                 |
| <a href="#">randShuffle(...)</a>                    | Randomly shuffle matrix elements  |

### Example 1. Smooth image ROI in-place

```
Mat imgroi = image(Rect(10, 20, 100, 100));  
GaussianBlur(imgroi, imgroi, Size(5, 5), 1.2, 1.2);
```

### Example 2. Somewhere in a linear algebra algorithm

```
m.row(i) += m.row(j)*alpha;
```

### Example 3. Copy image ROI to another image with conversion

```
Rect r(1, 1, 10, 20);  
Mat dstroi = dst(Rect(0,10,r.width,r.height));  
src(r).convertTo(dstroi, dstroi.type(), 1, 0);
```

## Simple Matrix Operations

OpenCV implements most common arithmetical, logical and other matrix operations, such as

- [add\(\)](#), [subtract\(\)](#), [multiply\(\)](#), [divide\(\)](#), [absdiff\(\)](#), [bitwise\\_and\(\)](#), [bitwise\\_or\(\)](#), [bitwise\\_xor\(\)](#), [max\(\)](#), [min\(\)](#), [compare\(\)](#)  
– correspondingly, addition, subtraction, element-wise multiplication ... comparison of two matrices or a matrix and a scalar.

Example. [Alpha compositing](#) function:

```
void alphaCompose(const Mat& rgba1,  
    const Mat& rgba2, Mat& rgba_dest)  
{  
    Mat a1(rgba1.size(), rgba1.type(), r1);  
    Mat a2(rgba2.size(), rgba2.type());  
    int mixch[]={3, 0, 3, 1, 3, 2, 3, 3};  
    mixChannels(&rgba1, 1, &a1, 1, mixch, 4);  
    mixChannels(&rgba2, 1, &a2, 1, mixch, 4);  
    subtract(Scalar::all(255), a1, r1);  
    bitwise_or(a1, Scalar(0,0,0,255), a1);  
    bitwise_or(a2, Scalar(0,0,0,255), a2);  
    multiply(a2, r1, a2, 1./255);  
    multiply(a1, rgba1, a1, 1./255);  
    multiply(a2, rgba2, a2, 1./255);  
    add(a1, a2, rgba_dest);  
}
```

- [sum\(\)](#), [mean\(\)](#), [meanStdDev\(\)](#), [norm\(\)](#), [countNonZero\(\)](#), [minMaxLoc\(\)](#),  
– various statistics of matrix elements.
- [exp\(\)](#), [log\(\)](#), [pow\(\)](#), [sqrt\(\)](#), [cartToPolar\(\)](#), [polarToCart\(\)](#)  
– the classical math functions.
- [scaleAdd\(\)](#), [transpose\(\)](#), [gemm\(\)](#), [invert\(\)](#), [solve\(\)](#), [determinant\(\)](#), [trace\(\)](#), [eigen\(\)](#), [SVD](#),  
– the algebraic functions + SVD class.
- [dft\(\)](#), [idft\(\)](#), [dct\(\)](#), [idct\(\)](#),  
– discrete Fourier and cosine transformations

For some operations a more convenient [algebraic notation](#) can be used, for example:

```
Mat delta = (J.t()*J + lambda*  
    Mat::eye(J.cols, J.cols, J.type()))  
    .inv(CV_SVD)*(J.t()*err);
```

implements the core of Levenberg-Marquardt optimization algorithm.

## Image Processing

### Filtering

|  |   |
|--|---|
| <a href="#">filter2D()</a>   | Non-separable linear filter   |
| <a href="#">sepFilter2D()</a>  | Separable linear filter   |
| <a href="#">boxFilter()</a> ,<br><a href="#">GaussianBlur()</a> ,<br><a href="#">medianBlur()</a> ,<br><a href="#">bilateralFilter()</a> | Smooth the image with one of the linear or non-linear filters   |
| <a href="#">Sobel()</a> , <a href="#">Scharr()</a>   | Compute the spatial image derivatives   |
| <a href="#">Laplacian()</a>  | compute Laplacian: $\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$ |
| <a href="#">erode()</a> , <a href="#">dilate()</a>   | Morphological operations  |

Example. Filter image in-place with a 3x3 high-pass kernel (preserve negative responses by shifting the result by 128):

```
filter2D(image, image, image.depth(), (Mat_<float>(3,3)«
-1, -1, -1, -1, 9, -1, -1, -1, -1), Point(1,1), 128);
```

## Geometrical Transformations

|                                |  |
|--------------------------------|--|
| <code>resize()</code>          | Resize image                                 |
| <code>getRectSubPix()</code>   | Extract an image patch                       |
| <code>warpAffine()</code>      | Warp image affinely                          |
| <code>warpPerspective()</code> | Warp image perspectively                     |
| <code>remap()</code>           | Generic image warping                        |
| <code>convertMaps()</code>     | Optimize maps for a faster remap() execution |

Example. Decimate image by factor of  $\sqrt{2}$ :

```
Mat dst; resize(src, dst, Size(), 1./sqrt(2), 1./sqrt(2));
```

## Various Image Transformations

|                                  |   |
|----------------------------------|---|
| <code>cvtColor()</code>          | Convert image from one color space to another   |
| <code>threshold()</code>         | Convert grayscale image to binary image using a fixed or a variable threshold   |
| <code>adaptiveThreshold()</code> | Find a connected component using region growing algorithm   |
| <code>floodFill()</code>         | Compute integral image  |
| <code>integral()</code>          | build distance map or discrete Voronoi diagram for a binary image.  |
| <code>distanceTransform()</code> | marker-based image segmentation algorithms. See the samples <a href="#">watershed.cpp</a> and <a href="#">grabcut.cpp</a> . |
| <code>watershed()</code>         |   |
| <code>grabCut()</code>           |   |

## Histograms

|                                |   |
|--------------------------------|---|
| <code>calcHist()</code>        | Compute image(s) histogram              |
| <code>calcBackProject()</code> | Back-project the histogram              |
| <code>equalizeHist()</code>    | Normalize image brightness and contrast |
| <code>compareHist()</code>     | Compare two histograms                  |

Example. Compute Hue-Saturation histogram of an image:

```
Mat hsv, H;
cvtColor(image, hsv, CV_BGR2HSV);
int planes[]={0, 1}, hsize[] = {32, 32};
calcHist(&hsv, 1, planes, Mat(), H, 2, hsize, 0);
```

## Contours

See [contours2.cpp](#) and [squares.cpp](#) samples on what are the contours and how to use them.

## Data I/O

[XML/YAML storages](#) are collections (possibly nested) of scalar values, structures and heterogeneous lists.

### Writing data to YAML (or XML)

```
// Type of the file is determined from the extension
```

```
FileStorage fs("test.yml", FileStorage::WRITE);
fs < "i" < 5 < "r" < 3.1 < "str" < "ABCDEFGH";
fs < "mtx" < Mat::eye(3,3,CV_32F);
fs < "mylist" < "[" < CV_PI < "1+1" <
  "{" < "month" < 12 < "day" < 31 < "year"
  < 1969 < "]" < "]";
fs < "mystruct" < "{" < "x" < 1 < "y" < 2 <
  "width" < 100 < "height" < 200 < "lbp" < "[:";
const uchar arr[] = {0, 1, 1, 0, 1, 1, 0, 1};
fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0]));
fs < "]" < "]";
```

*Scalars (integers, floating-point numbers, text strings), matrices, STL vectors of scalars and some other types can be written to the file storages using < operator*

### Reading the data back

```
// Type of the file is determined from the content
FileStorage fs("test.yml", FileStorage::READ);
int i1 = (int)fs["i"]; double r1 = (double)fs["r"];
string str1 = (string)fs["str"];
Mat M; fs["mtx"] > M;
FileNode t1 = fs["mylist"];
CV_Assert(t1.type() == FileNode::SEQ && t1.size() == 3);
double t10 = (double)t1[0]; string t11 = (string)t1[1];
int m = (int)t1[2]["month"], d = (int)t1[2]["day"];
int year = (int)t1[2]["year"];
FileNode tm = fs["mystruct"];
Rect r; r.x = (int)tm["x"], r.y = (int)tm["y"];
r.width = (int)tm["width"], r.height = (int)tm["height"];
int lbp_val = 0;
FileNodeIterator it = tm["lbp"].begin();
for(int k = 0; k < 8; k++, ++it)
  lbp_val |= ((int)*it) < k;
```

*Scalars are read using the corresponding FileNode's cast operators. Matrices and some other types are read using > operator. Lists can be read using FileNodeIterator's.*

### Writing and reading raster images

```
imwrite("myimage.jpg", image);
Mat image_color_copy = imread("myimage.jpg", 1);
Mat image_grayscale_copy = imread("myimage.jpg", 0);
```

*The functions can read/write images in the following formats: BMP (.bmp), JPEG (.jpg, .jpeg), TIFF (.tif, .tiff), PNG (.png), PBM/PGM/PPM (.p?m), Sun Raster (.sr), JPEG 2000 (.jp2). Every format supports 8-bit, 1- or 3-channel images. Some formats (PNG, JPEG 2000) support 16 bits per channel.*

### Reading video from a file or from a camera

```
VideoCapture cap;
if(argc > 1) cap.open(string(argv[1])); else cap.open(0);
Mat frame; namedWindow("video", 1);
for(;;) {
```

```
cap > frame; if(!frame.data) break;
imshow("video", frame); if(waitKey(30) >= 0) break;
}
```

## Simple GUI (highgui module)

|  |   |
|--|---|
| <code>namedWindow(winname, flags)</code> | Create named highgui window   |
| <code>destroyWindow(winname)</code>      | Destroy the specified window  |
| <code>imshow(winname, mtx)</code>        | Show image in the window  |
| <code>waitKey(delay)</code>              | Wait for a key press during the specified time interval (or forever). Process events while waiting. <i>Do not forget to call this function several times a second in your code.</i> |
| <code>createTrackbar(...)</code>         | Add trackbar (slider) to the specified window   |
| <code>setMouseCallback(...)</code>       | Set the callback on mouse clicks and movements in the specified window  |

See [camshiftdemo.cpp](#) and other [OpenCV samples](#) on how to use the GUI functions.

## Camera Calibration, Pose Estimation and Depth Estimation

|  |  |
|--|--|
| <code>calibrateCamera()</code>         | Calibrate camera from several views of a calibration pattern.                      |
| <code>findChessboardCorners()</code>   | Find feature points on the checkerboard calibration pattern.                       |
| <code>solvePnP()</code>                | Find the object pose from the known projections of its feature points.             |
| <code>stereoCalibrate()</code>         | Calibrate stereo camera.   |
| <code>stereoRectify()</code>           | Compute the rectification transforms for a calibrated stereo camera.               |
| <code>initUndistortRectifyMap()</code> | Compute rectification map (for <code>remap()</code> ) for each stereo camera head. |
| <code>StereoBM, StereoSGBM</code>      | The stereo correspondence engines to be run on rectified stereo pairs.             |
| <code>reprojectImageTo3D()</code>      | Convert disparity map to 3D point cloud.   |
| <code>findHomography()</code>          | Find best-fit perspective transformation between two 2D point sets.                |

To calibrate a camera, you can use [calibration.cpp](#) or [stereo\\_calib.cpp](#) samples. To get the disparity maps and the point clouds, use [stereo\\_match.cpp](#) sample.

## Object Detection

|                                |   |
|--------------------------------|---|
| <code>matchTemplate</code>     | Compute proximity map for given template.   |
| <code>CascadeClassifier</code> | Viola's Cascade of Boosted classifiers using Haar or LBP features. Suits for detecting faces, facial features and some other objects without diverse textures. See <a href="#">facedetect.cpp</a>       |
| <code>HOGDescriptor</code>     | N. Dalal's object detector using Histogram-of-Oriented-Gradients (HOG) features. Suits for detecting people, cars and other objects with well-defined silhouettes. See <a href="#">peopledetect.cpp</a> |