

Using OpenCV

Multimedia System
Spring 2020

OpenCV

► opencv.org



[ABOUT](#) [NEWS](#) [EVENTS](#) [RELEASES](#) [PLATFORMS](#) [BOOKS](#) [LINKS](#) [LICENSE](#)

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

Quick Links

- [Online documentation](#)
- [Tutorials](#)
- [User Q&A forum](#)
- [Report a bug](#)
- [Build farm](#)
- [Developer site](#)
- [Wiki](#)

► [Donate](#)

OpenCV

► Installation

- Download from <http://opencv.org> and compile from source
- **Windows:** Run executable downloaded from OpenCV website (for recent Visual Studio)
- Mac OS X: Install through MacPorts
- Linux: Install through the package manager (e.g. yum, apt) but make sure the version is sufficiently up-to-date for your needs
- OpenCV uses the **cv namespace** (omitted during this presentation to save space)

Installation

[ABOUT](#)[NEWS](#)[EVENTS](#)[RELEASES](#)[PLATFORMS](#)[BOOKS](#)[LINKS](#)[LICENSE](#)

Releases


 4.0.1

2018-12-22


 4.0.1

 [Documentation](#)


 [Sources](#)

 [Win pack](#)

 [iOS pack](#)

 [Android pack](#)


Visual Studio 버전에 따라
Source를 build필요
(최신 VS 버전은 build되어 있음)

 3.4.5


2018-12-22

 3.4.5

 [Documentation](#)

 [Sources](#)

 [Win pack](#)

 [iOS pack](#)

 [Android pack](#)

Image read/write

- ▶ Images in OpenCV are stored in a **Mat** object (Mat class)
- ▶ Consists of a matrix header and a pointer to the matrix containing the pixel values
- ▶ Header contains information such as the size of the matrix, the number of color channels in the image, etc.
- ▶ Access this information through functions:
 - `Mat.rows()`: Returns the number of rows
 - `Mat.columns()`: Returns the number of columns
 - `Mat.channels()`: Returns the number of channels

Image read/write

► Image read/write functions

Mat cv::imread (const **String** &filename, int flags=**IMREAD_COLOR**)

filename

Image type

IMREAD_GRAYSCALE
IMREAD_COLOR
IMREAD_ANYDEPTH
IMREAD_ANYCOLOR
.....

bool cv::imwrite (const **String** &filename, **InputArray** img, const
std::vector< int > ¶ms=std::vector< int >())

Image buffer

Image read/write

`using namespace cv;`



사용한다고 가정

- ▶ Read the color image named "foo.png"
 - `Mat foo=imread("foo.png", IMREAD_COLOR);`
- ▶ Write the data contained in foo to "bar.png"
 - `imwrite("bar.png", foo);`
- ▶ Create a window
 - `namedWindow("display", WINDOW_AUTOSIZE);`
- ▶ Display the data stored in foo
 - `imshow("display", foo);`
 - `waitKey();`

Create new image

► **cv::Mat::Mat (int rows, int cols, int type)**

- `Mat img(Size(640,480),CV_8UC3);`

Image size

Image type

(CV_8UC1, ..., CV_64FC4)

- `Mat img(500, 1000, CV_8UC3, Scalar(0,0, 100));`

color

Red color is 100

- `Mat img(500, 1000, CV_8UC1, Scalar(100));`

greylevel

Greylevel is 100

Image pixel types

- ▶ OpenCV defaults to **BGR**
- ▶ Can also handle other types of channels/orderings such as RGB, HSV or GRAY
- ▶ Can access the value for a particular channel at a specific pixel once we have know how to interpret the underlying data
 - **Mat.at(row,col)[channel]**: Returns a pointer to the image data at the specified location
- ▶ Convert between color spaces using
 - **cvtColor(foo, bar, CV_BGR2GRAY)**

Access pixel values

▶ Safer but slower way

```
Mat H(100, 100, CV_64F);  
for(int i = 0; i < H.rows; i++)  
for(int j = 0; j < H.cols; j++)  
H.at<double>(i,j)=1./(i+j+1);
```

▶ Efficient way (but lose some readability)

```
for(int i = 0; i < H.rows; i++) {  
    double* p = H.ptr(i);  
    for (int j = 0; j < H.cols; j++)  
        p[j] = 1./(i+j+1);  
}
```

Copy images

- ▶ The copy constructor and assignment operator (=) **only copy the headers and the pointer** to the underlying data
- ▶ Use **Mat::clone()** and **Mat::copyTo()** to perform a deep copy

Image Normalization and Thresholding

- ▶ Normalization remaps a range of pixel values to another range of pixel values
 - `void normalize(InputArray src, OutputArray dst,...)`
- ▶ OpenCV provides a general purpose method for thresholding an image
 - `double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)`
 - Specify thresholding scheme specified by the type variable

Image Smoothing

- ▶ Reduces the sharpness of edges and smooths out details in an image
- ▶ OpenCV implements several of the most commonly used methods
 - `void GaussianBlur(InputArray src, OutputArray dst,...)`
 - `void medianBlur(InputArray src, OutputArray dst,...)`

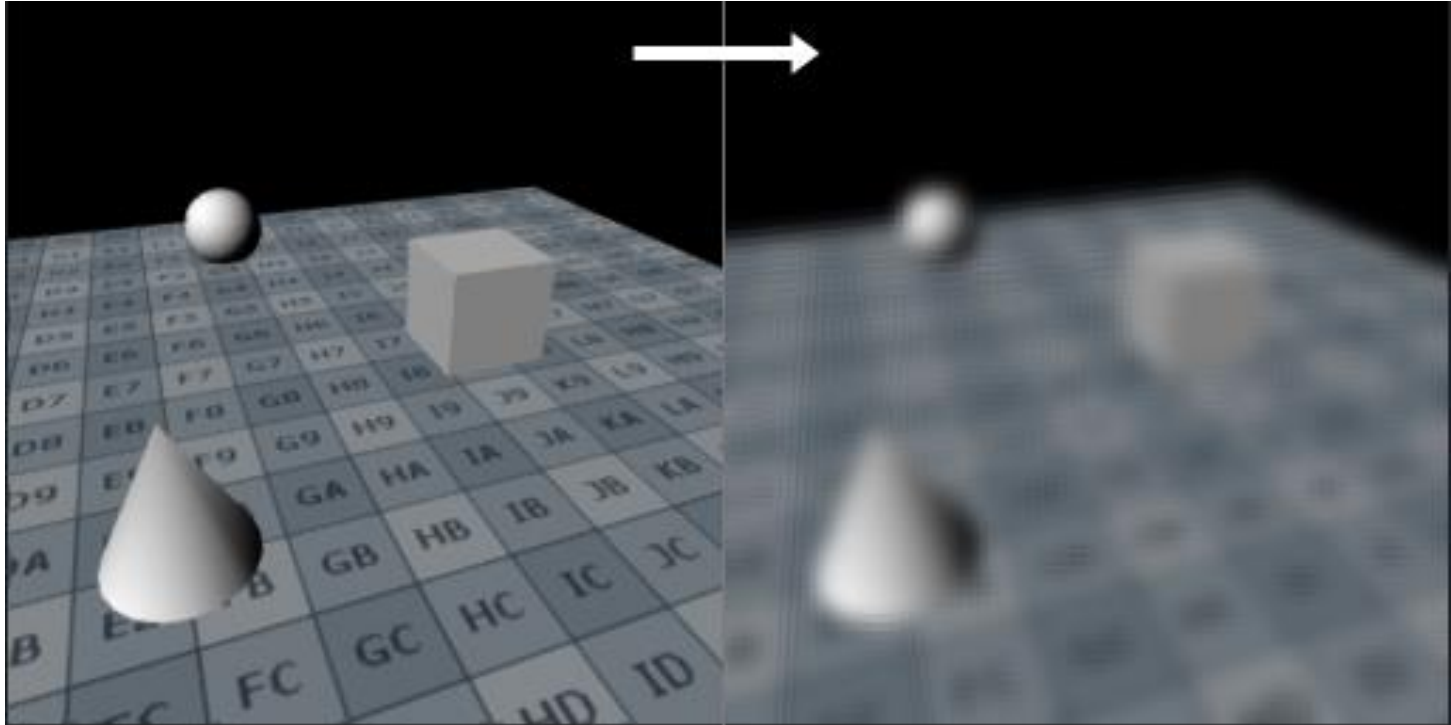
Image Smoothing

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
using namespace cv;

int main(int argc, char** argv) {
    //Read in colored image
    cv::Mat image=cv::imread(argv[1]);
    cv::imwrite("photo.jpg", image);
    //Apply Gaussian blur
    cv::Mat image_gaussian_blur;
    image.convertTo(image_gaussian_blur, CV_8UC3);
    cv::GaussianBlur(image_gaussian_blur, image_gaussian_blur, cv::Size(0,0), 9);
    cv::imwrite("photo_gaussian_blur.jpg", image_gaussian_blur);

    //Apply median blur
    cv::Mat image_median_blur;
    image.convertTo(image_median_blur, CV_8UC3);
    cv::medianBlur(image_median_blur, image_median_blur, 17);
    cv::imwrite("photo_median_blur.jpg", image_median_blur);
}
```

Image Smoothing



Edge Detection

- ▶ OpenCV implements a number of operators to help detect edges in an image
 - Sobel Operator
 - Scharr Operator
 - Laplacian Operator
- ▶ OpenCV also implements image detection algorithms such as Canny edge detection
- ▶ Sometimes smoothing the image before running edge detection gives better results

Edge Detection

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
```

} 또는 #include <opencv2/opencv.hpp>

```
int main(int argc, char** argv){
    //Read image as grayscale, delete zero to read in color
    cv::Mat image=cv::imread(argv[1],0);
    cv::imwrite("photo_gray.jpg",image);
    //Calculate x-gradient using Sobel operator
    cv::Mat image_gradient_x; image.convertTo(image_gradient_x,CV_32FC1);
    cv::Sobel(image_gradient_x,image_gradient_x,CV_32FC1,0,1);
    //Absolute value and normalize
    cv::convertScaleAbs(image_gradient_x,image_gradient_x);
    //Calculate y-gradient using Sobel operator
    cv::Mat image_gradient_y; image.convertTo(image_gradient_y,CV_32FC1);
    cv::Sobel(image_gradient_y,image_gradient_y,CV_32FC1,1,0);
    //Absolute value and normalize
    cv::convertScaleAbs(image_gradient_y,image_gradient_y);
    //Average the x and y gradients into one image
    cv::Mat image_gradient;
    cv::addWeighted(image_gradient_x,0.5,image_gradient_y,0.5,0,image_gradient);
    cv::imwrite("photo_gradient.jpg",image_gradient);
}
```