CS412 – A Practical Guide to OpenCV

Pham Minh Hoang, Tran Anh Duy

October 23, 2020

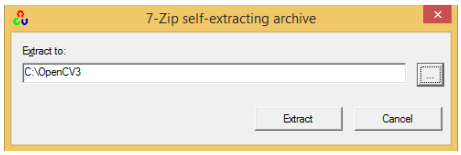
1 Installation and Configuration

1.1 Download OpenCV

Download the latest version from here.

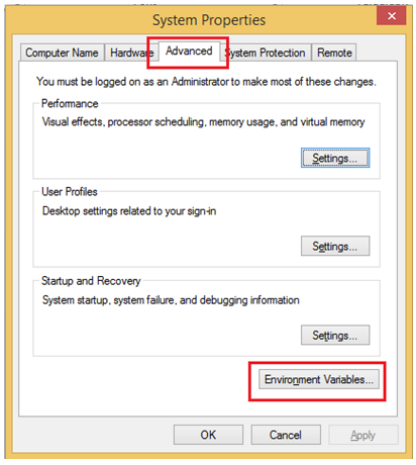
1.2 Unzip

Click the installation file to decompress



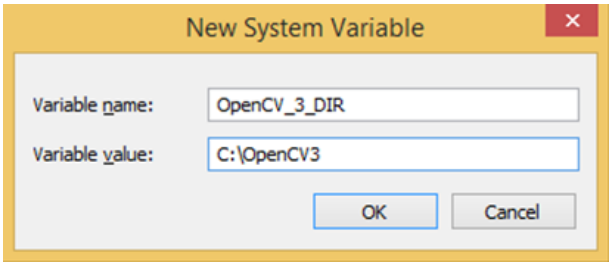
1.3 Set the environment variable

- Right click on the *"Computer"* icon, select *"Properties"*



- Select *"Advanced System Setting"*

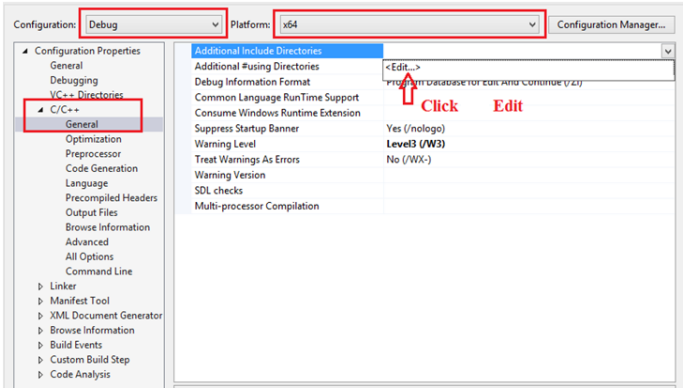
- In the *"System Properties"* dialog, select the *"Advance"* tab, click on the *"Environment Variables"* button
- In the *"System Variables"* dialog, select *"New"* and fill the *"Variable name"* box and the *"Variable value"* box as below figure

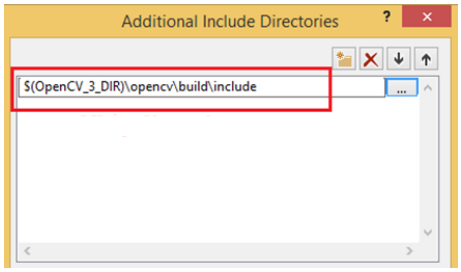


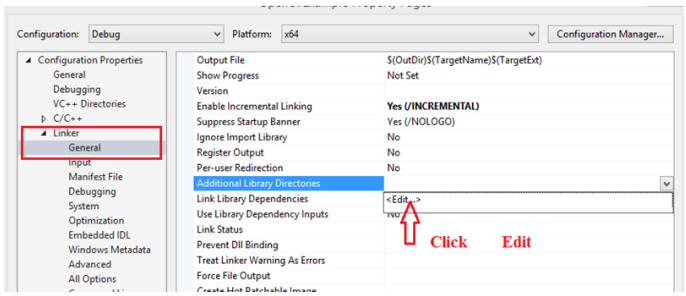
- Restart your computer to update the environment variable

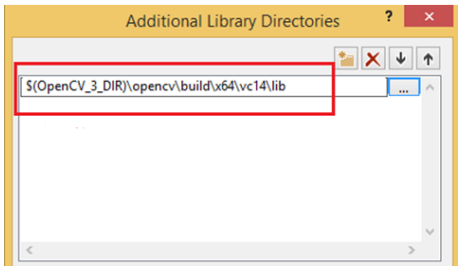
1.4 Project configuration

- Start *Visual Studio*, select project type *"Visual C++ Console Application"*
- Right click on project, choose *"Properties"*
- Note the configuartion (Debug or Release) and the platform (x64 or Win32)
- Add *"Include Directories"*

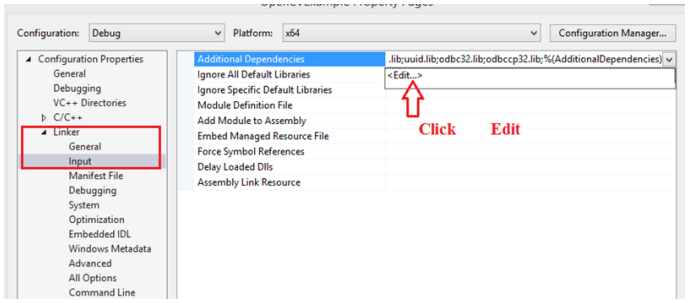


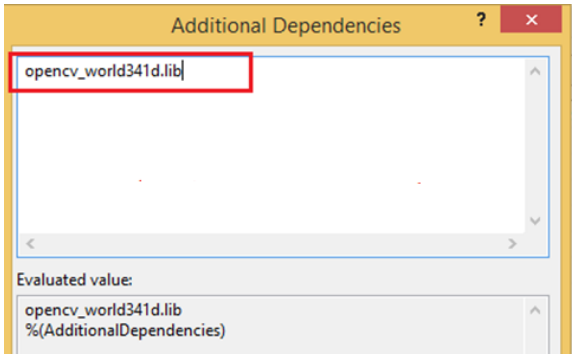






- Add *"Library Directories"* (above figure)
- Add lib file





- Build project to create exe file

1.5 Copy dll files to the folder containing the program file

Copy *"opencv_world341d.dll"* (Debug mode) or *"opencv_world341.dll"* (Release mode) from `C:\OpenCV3\opencv\build\x64\vc14\bin` to the folder containing the exe file.

1.6 Try to run the program

Copy this source code into the main cpp file and try to run it

Source Code 1:

```
#include "stdafx.h"
#include "opencv2/opencv.hpp"
#include "opencv2/highgui/highgui.hpp"
using namespace cv;

int main()
{
        //read the input image from file
        Mat image = imread("D:\\lena.png", CV_LOAD_IMAGE_COLOR);

        //create and set the window name
        namedWindow("Show Image");

        //show the image on window
        imshow("Show Image", image);

        //close the window
        waitKey(0);
        return 0;
}
```

The installation and project configuartion can be finised if the program shows the image sucessfully

2 Pratical guide OpenCV

In OpenCV, an image (or matrix) is stored by a *Mat* object

2.1 Mat structure

Define the *Mat* structure

```
class CV_EXPORTS Mat
{
public:
    // ... a lot of methods ...
    ...

    /*! includes several bit-fields:
         - the magic signature
         - continuity flag
         - depth
         - number of channels
     */
    int flags;
    //! the array dimensionality, >= 2
    int dims;
    //! the number of rows and columns or (-1, -1) when the array has more than 2 dimensions
    int rows, cols;
    //! pointer to the data
    uchar* data;

    //! pointer to the reference counter;
    // when array points to user-allocated data, the pointer is NULL
    int* refcount;

    // other members
    ...
};
```

Source Code 2: Matrix declaration

```
//declare an empty matrix
Mat::Mat();

//declare an matrix with the number of rows and columns
Mat::Mat(int rows, int cols, int type);

//declare an matrix with its size
Mat::Mat(Size size, int type);

//declare an matrix with the default value
Mat::Mat(int rows, int cols, int type, const Scalar& s);
```

Explaination of parameters

- *rows*: the number of rows is equivalent to the image height
- *cols*: the number of columns is equivalent to the image width
- *type*: data type of matrix are represented in the belowed form

  CV_<#bits><data type><#channels>

  - *#bits*: the number of bits in a channel
  - *data type*: data type of each element in the matrix (UC: unsigned char, SC: signed, FC: float)
  - *#channels*: the number of channels
- For example
  - *CV_8UC1*: 1 channel, 8 bit per channel, data type: unsigned char, grayscale image
  - *CV_8UC3*: 3 channel, 8 bit per channel, data type: unsigned char, color image
  - *CV_16SC1*: 1 channel, 16 bit per channel, data type: signed short
  - *CV_32FC1*: 1 channel, 32 bit per channel, data type: float
  - *CV_64FC1*: 1 channel, 64 bit per channel, data type: double

Source Code 3: Example of matrix declaration

```
//Matrix a with size of 100 x 100, 1 channel
//data type of each element: unsigned char, grayscale image
Mat a = Mat(100, 100, CV_8UC1);

//Matrix a with size of 200 x 200, 3 channels,
//data type of each element: unsigned char, RGB image
Mat a = Mat(200, 200, CV_8UC3, Scalar(0));

//Matrix a with size of 120 x 240, 1 channel,
//data type of each element: 32 bit float
Mat a = Mat(Size(120, 240), CV_32FC1);

//Matrix a with size of 240 x 120, 1 channel,
//data type of each element: 64 bit float (double)
Mat a = Mat(Size(120, 240), CV_64FC1);
```

2.2 Some basis functions

Load image from file

Source Code 4:

```
//load a color image
Mat image = imread("filename.jpg", CV_LOAD_IMAGE_COLOR);

//load a grayscale image
Mat image = imread("filename.jpg", CV_LOAD_IMAGE_GRAYSCALE);
```

Save image

Source Code 5:

```
imwrite("filename.jpg", image);
```

Display image on window

Source Code 6:

```
//create a window and set a name "Show image"
namedWindow("Show Image");

//display image on the window "Show Image"
imshow("Show Image", image);

//wait for user to close the window
waitKey(0);
```

2.3 Pixel processing

OpenCV stores a grayscale image matrix [1]

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

OpenCV stores a color image according to BGR order (the first color is Blue, the next color is Green, and the last one is Red) [1]

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

To get a pixel, we can use three methods

Source Code 7: Using at() function

```
void method_1(cv::Mat &image, int n) {
    //get image width
    int width = image.cols;
    //get image height
    int height = image.rows;
    for (int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            //for grayscale image
            if (image.channels() == 1) {
                image.at<uchar>(x, y) = 255;
            }
            //for color image
            else if (image.channels() == 3) {
                cv::Vec3b& pixel = image.at<cv::Vec3b>(x, y);
                //get the first color in this pixel (blue)
                pixel[0] = 255;
                //get the second color in this pixel (green)
                pixel[1] = 255;
                //get the last color in this pixel (red)
                pixel[2] = 255;
            }
        }
    }
}
```

Source Code 8: Using the pointer to the first element of each row

```
void method_2(cv::Mat &image, int n) {
    int width = image.cols, height = image.rows;
    int nChannels = image.channels();
    for (int y = 0; y < height; y++) {
        //get the pointer to the first element of the y-th row
        uchar* pRow = image.ptr<uchar>(y);
        for(int x = 0; x < width; x++, pRow += nChannels) {
            //get the first color in this pixel (blue)
            pRow[0] = 255;
            //get the second color in this pixel (green)
            pRow[1] = 255;
            //get the last color in this pixel (red)
            pRow[2] = 255;
        }
    }
}
```

According to [2], the class *Mat* represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a *SparseMat*). The data layout of the array M is defined by the array $M.step[]$, so that the address of element $(i_0, ..., i_{M.dims-1})$, where $0 \leq i_k < M.size[k]$, is computed as:

$$addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]i_0 + M.step[1]i_1 + \cdots + M.step[M.dims-1]i_{M.dims-1}$$

In case of a 2-dimensional array, the above formula is reduced to:

$$addr(M_{i,j}) = M.data + M.step[0]i + M.step[1]j$$

where $M.step[0]$ is also *widthStep*, which is the byte distance between two consecutive pixels on the same column, and $M.step[1]$ is the number of channels

Note that $M.step[i] \geq M.step[i+1]$ (in fact, $M.step[i] \geq M.step[i+1]M.size[i+1]$). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. $M.step[M.dims-1]$ is minimal and always equal to the element size $M.elemSize()$.

Source Code 9: Using the pointer to manage the image data

```
void method_3(cv::Mat &image, int n) {
    int width = image.cols, height = image.rows;
    int widthStep = image.steps[0];
    int nChannels = image.steps[1];
    uchar* pData = (uchar*)image.data;
    for (int y = 0; y < height; y++, pData += widthStep) {
        uchar* pRow = pData;
        for(int x = 0; x < width; x++, pRow += nChannels) {
            //get the first color in this pixel (blue)
            pRow[0] = 255;
            //get the second color in this pixel (green)
            pRow[1] = 255;
            //get the last color in this pixel (red)
            pRow[2] = 255;
        }
    }
}
```

2.4 GUI programming

To add a trackbar, we can use the function *createTrackbar* [3], [4]

int createTrackbar(const string& trackbarname, const string& winname, int* value, int count, TrackbarCallback onChange = 0, void* userdata = 0)

Explaination of parameters

- trackbarname – Name of the created trackbar.
- winname – Name of the window that will be used as a parent of the created trackbar.
- value – Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.
- count – Maximal position of the slider. The minimal position is always 0.
- onChange – Pointer to the function to be called every time the slider changes position. This function should be prototyped as void Foo(int,void*); , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only value is updated.

- userdata – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

In the following OpenCV example, we have added a trackbar to change the value. And a callback function is implemented for this trackbar.

Source Code 10: Using a trackbar

```
void CallbackFunction(int pos, void *userData)
{
     //cast userData to value
     int valueFromUser = *( static_cast<int*>(userData) );
    ...
}

int main(int argc, char** argv)
{
     // Read original image
    src = imread("MyPic.JPG");

    // Create a window
    namedWindow("My Window", 1);

    int pos = 50;
    int val = 50;

    //Create track bar
    createTrackbar("Trackbar Name", "My Window", &pos, 100,
        CallbackFunction, &val);

    imshow("My Window", src);

    // Wait until user press some key
    waitKey(0);


    return 0;
}
```

References

[1] The OpenCV Tutorials

[2] The OpenCV Document page
https://docs.opencv.org/3.4/index.html

[3] OpenCV Tutorial C++
https://www.opencv-srf.com/2011/11/track-bars.html

[4] Learn OpenCV by Examples
http://opencvexamples.blogspot.com/2013/10/adding-trackbar.html