Load and Display an Image

Goal

In this tutorial you will learn how to:

- Load an image (using **cv::imread**)
- Create a named OpenCV window (using **cv::namedWindow**)
- Display an image in an OpenCV window (using **cv::imshow**)

Source Code

Download the source code from here.

```cpp
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>
#include <string>

using namespace cv;

using namespace std;

int main( int argc, char** argv )
{
    String imageName( "../data/HappyFish.jpg" ); // by default
    if( argc > 1)
    {
        imageName = argv[1];
    }

    Mat image;

    image = imread( imageName, IMREAD_COLOR ); // Read the file

    if( image.empty() )                        // Check for invalid input
    {
        cout <<  "Could not open or find the image" << std::endl ;
        return -1;
    }

    namedWindow( "Display window", WINDOW_AUTOSIZE ); // Create a window for display.

    imshow( "Display window", image );               // Show our image inside it.
```

```
    waitKey(0); // Wait for a keystroke in the window
    return 0;
}
```

Explanation

In OpenCV 2 we have multiple modules. Each one takes care of a different area or approach towards image processing. You could already observe this in the structure of the user guide of these tutorials itself. Before you use any of them you first need to include the header files where the content of each individual module is declared.

You'll almost always end up using the:

- *core* section, as here are defined the basic building blocks of the library
- *highgui* module, as this contains the functions for input and output operations

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>
#include <string>
```

We also include the *iostream* to facilitate console line output and input. To avoid data structure and function name conflicts with other libraries, OpenCV has its own namespace: *cv*. To avoid the need appending prior each of these the *cv::* keyword you can import the namespace in the whole file by using the lines:

```
using namespace cv;
```

This is true for the STL library too (used for console I/O). Now, let's analyze the *main* function. We start up assuring that we acquire a valid image name argument from the command line. Otherwise take a picture by default: "HappyFish.jpg".

```
    String imageName( "../data/HappyFish.jpg" ); // by default
    if( argc > 1)
    {
        imageName = argv[1];
    }
```

Then create a *Mat* object that will store the data of the loaded image.

```
    Mat image;
```

Now we call the **cv::imread** function which loads the image name specified by the first argument (*argv[1]*). The second argument specifies the format in what we want the image. This may be:

- IMREAD_UNCHANGED (<0) loads the image as is (including the alpha channel if present)
- IMREAD_GRAYSCALE (0) loads the image as an intensity one

- IMREAD_COLOR (>0) loads the image in the RGB format

```
image = imread( imageName, IMREAD_COLOR ); // Read the file
```

Note

> OpenCV offers support for the image formats Windows bitmap (bmp), portable image formats (pbm, pgm, ppm) and Sun raster (sr, ras). With help of plugins (you need to specify to use them if you build yourself the library, nevertheless in the packages we ship present by default) you may also load image formats like JPEG (jpeg, jpg, jpe), JPEG 2000 (jp2 - codenamed in the CMake as Jasper), TIFF files (tiff, tif) and portable network graphics (png). Furthermore, OpenEXR is also a possibility.

After checking that the image data was loaded correctly, we want to display our image, so we create an OpenCV window using the **cv::namedWindow** function. These are automatically managed by OpenCV once you create them. For this you need to specify its name and how it should handle the change of the image it contains from a size point of view. It may be:

- *WINDOW_AUTOSIZE* is the only supported one if you do not use the Qt backend. In this case the window size will take up the size of the image it shows. No resize permitted!
- *WINDOW_NORMAL* on Qt you may use this to allow window resize. The image will resize itself according to the current window size. By using the | operator you also need to specify if you would like the image to keep its aspect ratio (*WINDOW_KEEPRATIO*) or not (*WINDOW_FREERATIO*).

```
namedWindow( "Display window", WINDOW_AUTOSIZE ); // Create a window for display.
```

Finally, to update the content of the OpenCV window with a new image use the **cv::imshow** function. Specify the OpenCV window name to update and the image to use during this operation:

```
imshow( "Display window", image );              // Show our image inside it.
```

Because we want our window to be displayed until the user presses a key (otherwise the program would end far too quickly), we use the **cv::waitKey** function whose only parameter is just how long should it wait for a user input (measured in milliseconds). Zero means to wait forever.

```
waitKey(0); // Wait for a keystroke in the window
```

Result

- Compile your code and then run the executable giving an image path as argument. If you're on Windows the executable will of course contain an *exe* extension too. Of course assure the image file is near your program file.
  ```
  ./DisplayImage HappyFish.jpg
  ```
- You should get a nice window as the one shown below: