Multimedia
(Lab 02)

Spring, 2018

Department of Software

Yong Ju Jung (정용주)

가천대학교 소프트웨어학과
Gachon University

Summary

- In previous lab, you have installed Visual C++ and OpneCV library in your computer.

- In this lab, continue to the basic practice.
  - Load an image and display it on your screen.
  - Also, you will learn about simple pixel transforms in images.

[Lab01-2] cont'd

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

http://docs.opencv.org/3.2.0/db/deb/tutorial_display_image.html

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/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.c_str(), 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;
}
```


* Source from OpenCV tutorial

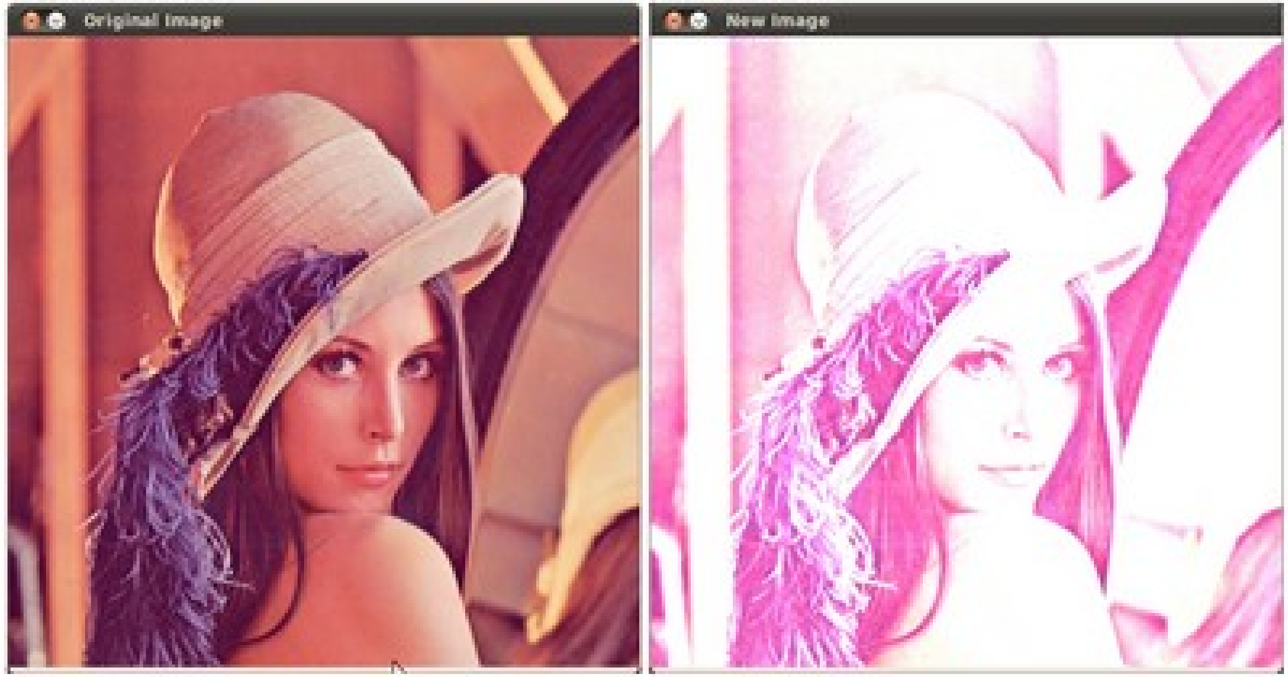
[Lab02-1]

- A general image processing operator is a function that takes one or more input images and produces an output image.

- Simple pixel transform: We will learn how to change our image appearance.
  - Load an image (using **cv::imread**)
  - Change its contrast and brightness of an image
    - $g(x) = \alpha * f(x) + \beta$
    - The parameters $\alpha>0$ and β are often called the *gain* and *bias* parameters; sometimes these parameters are said to control *contrast* and *brightness* respectively.
  - Display the result image in an OpenCV window (using **cv::imshow**)

Example results

- using α=2.2 and β=50



* Source from OpenCV tutorial

Some useful function

- **Saturation Arithmetic**
  - OpenCV deals a lot with image pixels that are often encoded in a compact, **8- or 16-bit per channel**, form and thus have a limited value range.
  - Furthermore, certain operations on images, like color space conversions, brightness/contrast adjustments, sharpening, complex interpolation can **produce values out of the available range**.
  - If you just store the lowest 8 (16) bits of the result, this results in visual artifacts and may affect a further image analysis.
  - To solve this problem, the so-called *saturation* arithmetics is used.
  - For example, to store *r*, the result of an operation, to an 8-bit image, you find the nearest value within the 0..255 range:
    - $\mathbf{I}(x,y)=\min(\max(\mathrm{round}(r),0),255)$
-

```
I.at<uchar>(y, x) = saturate_cast<uchar>(r);
```

- To access each element of an image matrix, use
  - .at<data type>(y, x)

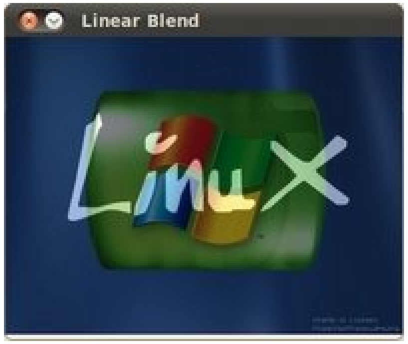
  - For example, in case of gray image
    - Mat gray_img(Size(1920, 1080), CV_8UC1);
    - int value = gray_img.at<uchar>(100, 200);

  - In case of color image
    - Mat color_img(Size(1920, 1080), CV8UC3);
    - int blue = color_img.at<Vec3b>(y, x)[0];
    - int green = color_img.at<Vec3b>(y, x)[1];
    - int red = color_img.at<Vec3b>(y, x)[2];

Useful Tip for Debug

- **Image Watch: viewing in-memory images in the Visual Studio debugger**
  - *Author:* Wolf Kienzle
  - You will learn how to visualize OpenCV matrices and images within Visual Studio 2012.

[Lab02-2]



- Linear blending of two images
  - Load two images (using **cv::imread**)
  - Do linear blending of the two images
    - $dst(x,y) = \alpha * src_1(x,y) + \beta * src_2(x,y)$
    - where $\beta = 1 - \alpha$.
    - cv::addWeighted(src1, α, src2, β, γ, dst)
  - Display the result image in an OpenCV window (using **cv::imshow**)

    - By varying α from $0 \rightarrow 1$ this operator can be used to perform a temporal cross-dissolve between two images