COMPUTER VISION AND IMAGE PROCESSING

LAB SESSION 1
INTRODUCTION TO OPENCV

ALESSIO TONIONI – ALESSIO.TONIONI@UNIBO.IT

In the meanwhile..

- Create a laboratory account if you don't already have one.
  - Select "**Crea Account**" at pc startup and follow the instruction.
- Download the lab material from:
  - didattica.arces.unibo.it
  - Prof. Luigi Di Stefano
  - «Computer Vision and Image Processing M» course
  - Material (from left-hand menu)
  - Scroll down to «Laboratory: Slides, Software and Images»
  - Download «Software ElabImage, OpenCV and Documentation» (zip file)
  - Unzip the archive and open the VisualStudio solution (ElabImage_2010.sln) located in the sub-folder: “Elabimage”

The OpenCV library

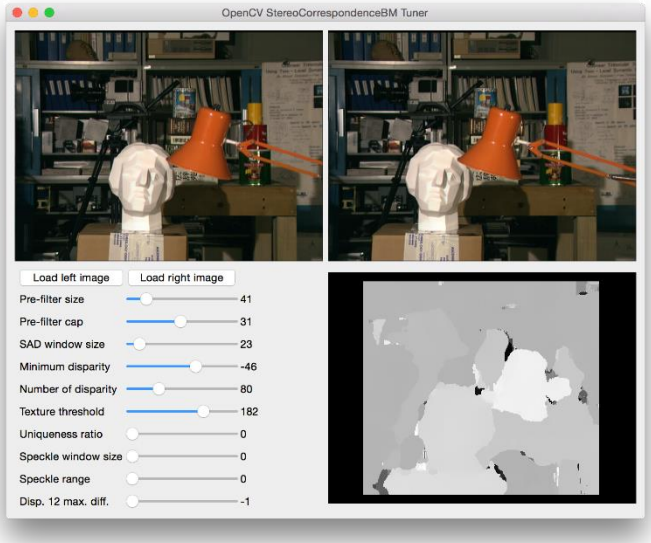
- Open Computer Vision Library: a collection of open source algorithms for computer vision and image processing
- Originally developed by Intel, then funded and supported by Willow Garage. Currently a non-profit foundation (www.opencv.org)
- Released under the BSD license – free for both academic and commercial use
- Main language: C/C++, with optimized routines (OMP/TBB, SIMD, CUDA, ...)
  - C was the main language before version 2.2, C++ is the current default interface
- Current version: 3.3/**2.4.13.3**
- Freely downloadable from:
  - http://opencv.org/releases.html
- Available for Windows, Linux, iOS and Android (*partial*).
- In our lab sessions:
  - O.S.: Windows
  - C/C++ Compiler: Visual Studio 2010

OpenCV structure

- **Several modules, allows linking only to required features**
  - **core:** defines basic data types such as points, vectors, single/multi channel matrices (images), includes also functions for linear algebra, DFT, XML, YAML-based I/O
  - **imgproc:** algorithms for image filtering, morphology, resizing, color mapping, image histograms, etc..
  - **highgui:** window handler for displaying images, video stream handler,…
  - **calib3d:** camera calibration, stereo matching,…
  - **features2d** 2D feature detectors and descriptors (SIFT, SURF, FAST, etc., including the new feature detectors-descriptor-matcher framework)
  - **flann:** wrapper of the Fast Library for Approximate Nearest Neighbors (FLANN) for Nearest Neighbor Search over high dimensional spaces
  - **ml:** machine learning algorithms (SVM, Decision Trees, Boosting, Random Forests, etc.)
  - **objdetect:** object detection on images (Haar & LBP face detectors, HOG people detector etc.)
  - **video:** algorithms for computer vision on video streams (tracking, optical flow, background subtraction,…)
  - **gpu:** acceleration of some OpenCV functionalities using CUDA (stereo, HOG, linear algebra)
  - **contrib:** contributed code that is not mature enough (SpinImages, Chamfer distance, …)
  - **legacy:** obsolete code, preserved for backward compatibility

Graphical User Interface

- OpenCV natively offers limited capabilities for what concerns user interaction and GUIs.
- Richer interface using Qt backend for OpenCV
  - Complete support for windows creation and interaction.
  - Text rendering using TT fonts.
  - Separated "control panel" with sliders, push-buttons, checkboxes and radio buttons.
  - Interactive zooming, panning of the images displayed in highgui windows, “save as“, etc…



How to get Help!

- Included in the zip file of the course material, you'll find:
  - */doc/opencv.pdf* : Tutorial about all the basic and advanced functionalities of openCV 2.4.9
  - */doc/opencv_cheatsheet.pdf*: OpenCV Cheatsheet (only for C++)

- Mail your tutor: alessio.tonioni@unibo.it

- Additional material:
  - Online OpenCV documentation: ***http://docs.opencv.org***
  - Online OpenCV tutorial: ***http://docs.opencv.org/2.4/doc/tutorials/tutorials.html***
  - Q&A website similar to Stack Overflow: ***http://answers.opencv.org/questions/***

OpenCV and VS projects (in lab)

- **Append the OpenCV «bin» subfolder to your Windows «PATH» environment variable**
  - Append the OpenCV subfolder containing the dlls to the Windows «PATH» environment variable. In our solution, the subfolder is the «cvdll» within «Elabimage»
- **Every time you create a new project:**
  - Create a new project via «File->New->New Project», specifying «**Visual C++ Empty Project**»
  - Specify the folder containing the include (.h) files: in "Project -> Properties", choose "Configuration Properties -> C/C++ -> General" and add in the field "Additional include directories" the path to the .h files of OpenCV. In ElabImage, the path can be given as
    - "../include"
  - Specify the required .lib files: in "Project -> Properties", choose "Configuration Properties -> Linker -> Input" and add in the field "Additional dependencies" a string such as:

\path\to\opencv_core249.lib;\path\to\opencv_imgproc249.lib;\path\to\opencv_highgui249.lib

  - Add the appropriate "#include" commands for the OpenCV headers at the beginning of your code. E.g.:

C Interface

```
#include "opencv/cv.h"
#include "opencv/highgui.h"
```

C++ Int.

```
#include "opencv2/opencv.hpp"
```

OpenCV and VS projects (at home)

- **Append the OpenCV «bin» subfolder to your Windows «PATH» environment variable**
  - this should be already done by the installer - just say yes when prompted; otherwise you can do it manually.
- **For each new project:**
  - Create a '.cpp' file where you will put the project main.
  - Add the OpenCV «include» folder as an «additional include directory»: in "Project –> Properties", select the "Configuration Properties –> C/C++ –> General" tab and add in the field "Additional Include Directories" your own include path:
    - *"ROOT_OPENCV\include"*
  - Add the OpenCV «libs» folder as an «additional library directory»: in "Project –> Properties", select the "Configuration Properties –> Linker –> General" tab and add in the "Additional Library Directories" your own lib path:
    - *"ROOT_OPENCV\lib"*
  - Specify the required OpenCV libs for the current project. In "Project –> Properties", select the "Configuration Properties –> Linker –> Input" tab and add in the field "Additional dependencies" the required lib files, e.g.
    - *"opencv_core249.lib opencv_imgproc249.lib opencv_highgui249.lib"*
  - Add the "#include" command and the required .h files appropriately at the beginning of the header files of your project, e.g.
    - *#include "opencv2/opencv.hpp*

OpenCV - CMake

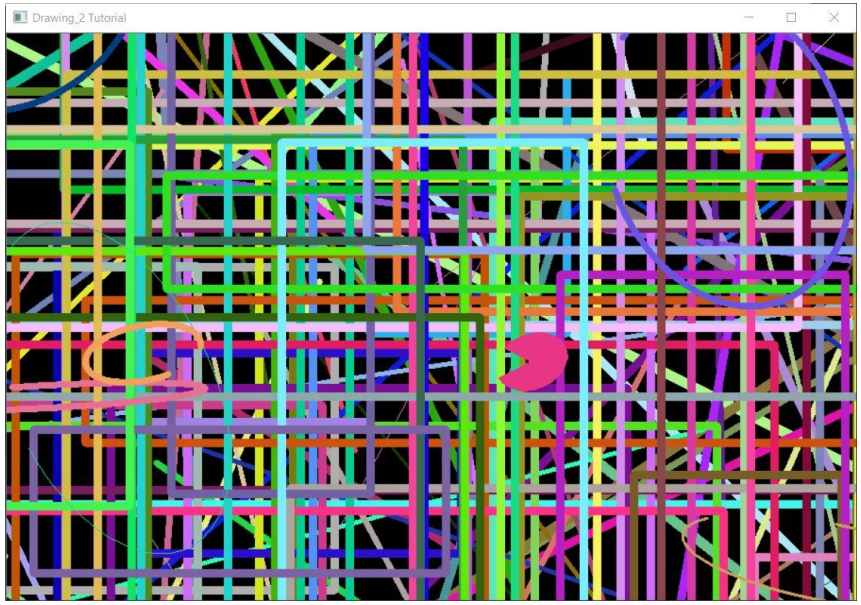
- CMake is a program that allows the creation of C/C++ projects portable across different operating system (e.g. Windows, linux…).
- Each project has a '*CMakeLists.txt*' file that acts like a *recipe* describing all the dependencies and which file should be compiled to create the executable.
- Based on this *recipe* CMake can create all the files needed for compilation across different platforms (i.e. visual studio projects, eclipse projects and Makefiles…).
- Take a look at *'CMakeLists.txt'* inside the *ElabImage* folder for an example of CMakeLists file that links OpenCV libraries.

- Some usefull links:
  - CMake tutorial
  - Using OpenCV with CMake

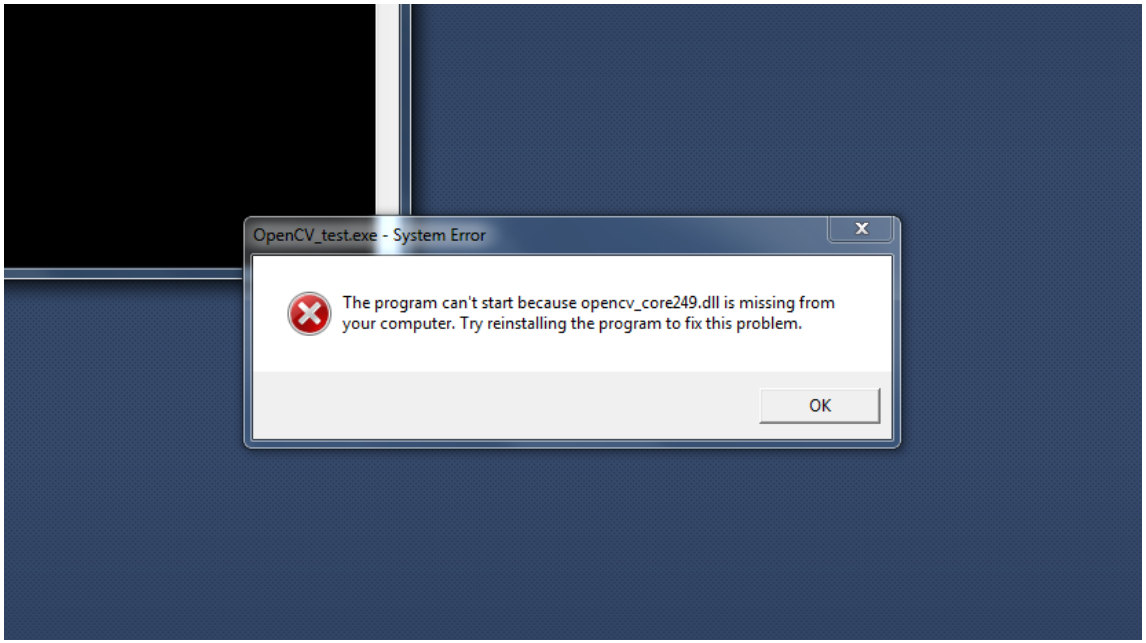
Exercise 0 – OpenCV warm up

Test code thanks to Ana Huamán - Link

- Append the «cvdll» subfolder to your Windows «PATH» environment variable.
- Open 'ElabImage_2010.sln'.
- Select "Visual C++ Development Settings"
- Compile and run the '**OpenCV_test**' project



Exercise 0 – OpenCV warm up

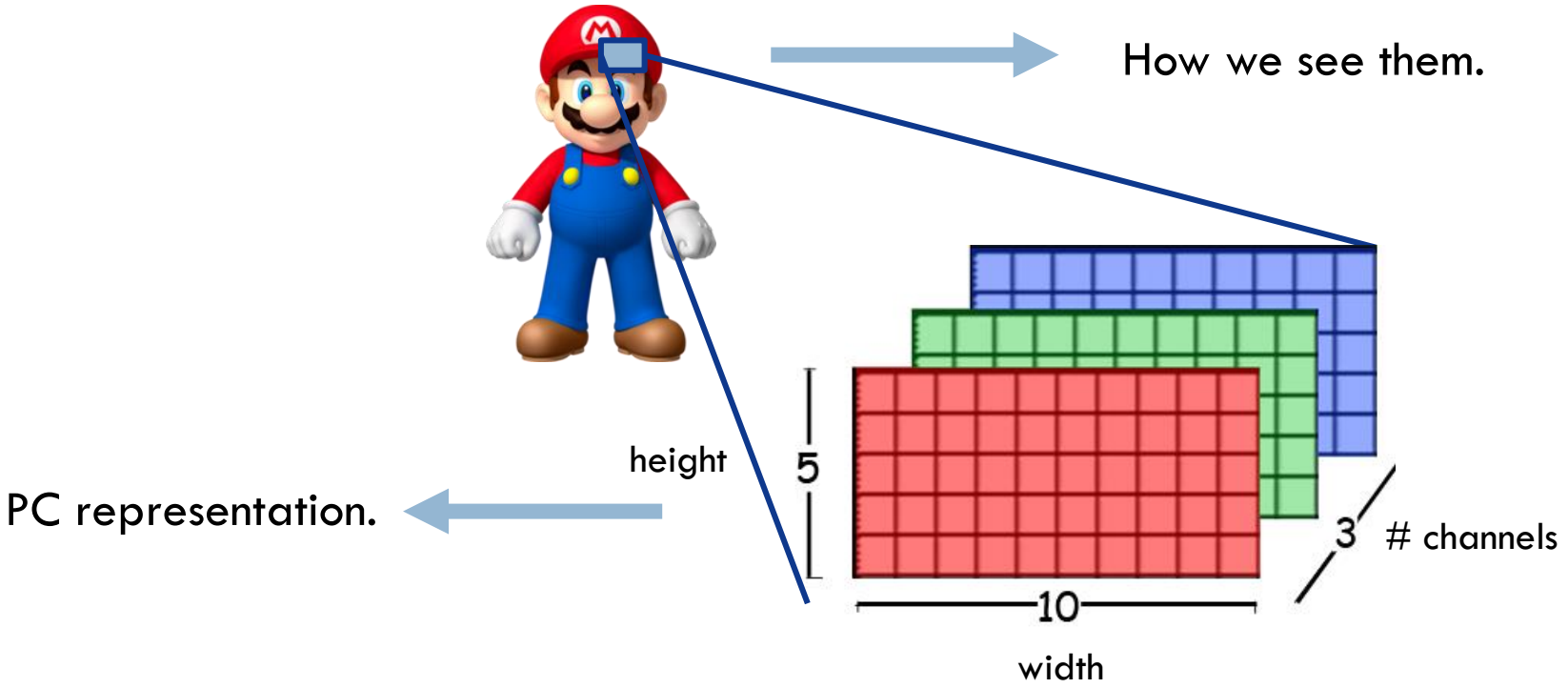


Do not Panic!!

and

Append the «cvdll» subfolder to your Windows «PATH» environment variable

Images



- Matrix like data structures to handle images.
- Each pixel of the image is a value (usually an int between 0-255) in one cell of the matrix, if the image has multiple channel, multiple value for each cell.

C++ vs C

- During the lab you can use both the C or C++ interface of OpenCV.
- At the beginning of each cpp:

C++

```
#define OPENCV_CPP_INTERFACE
```

```
#define OPENCV_CPP_INTERFACE

#ifndef OPENCV_CPP_INTERFACE
    #include "opencv/cv.h"
    #include "opencv/highgui.h"
#else
    #include "opencv2/opencv.hpp"
#endif
```

C

```
//#define OPENCV_CPP_INTERFACE
```

```
//#define OPENCV_CPP_INTERFACE

#ifndef OPENCV_CPP_INTERFACE
    #include "opencv/cv.h"
    #include "opencv/highgui.h"
#else
    #include "opencv2/opencv.hpp"
#endif
```

Basic Image Type

`cv::Mat`

- Basic C++ data structure to represent images and their properties (width, height, channels…).
- Defined in opencv_core.
- Can be used to encode any kind of n-dimensional and multi-channel matrix, image as a special case.

`IplImage`

- Basic C data structure to represent images and their properties (width, height, channels…).
- Defined in opencv_core.
- Can be used only to represent image.

Image creation/destruction

C++ → `cv::Mat`

- Creation:

```
cv::Mat m(cv::Size size, int type);
```

  - `Size`: `struct` with `width` (cols) and `height` (rows) field.
  - `Type`: type of internal data and channels saved in constant with this format:

    `'CV_'+depth+'C'+#channels`

    (i.e. CV_8UC1 or CV_8UC4)

- Automatic management of the memory, no need to manually release image!

C → `IplImage`

- Creation:

```
IplImage* cvCreateImage (CvSize size, int depth, int channels);
```

  - `Size`: `struct` with `width` and `height` field
  - `Depth`: type of internal data
    - (`IPL_DEPTH_8U`, `IPL_DEPTH_8S`, `IPL_DEPTH_16S`, `IPL_DEPTH_32S`, `IPL_DEPTH_32F`, `IPL_DEPTH_64F`)
  - `Channels`: number of channels

- `void cvReleaseImage(IplImage** image)`

Load and display an image

C++ → `cv::Mat`

- **Load:**

```
cv::Mat m = cv::imread(std::string filepath, int flags);
```

  - `Filepath:` Path to the image file.
  - `Flags:` How to laod the image:
    - CV_LOAD_IMAGE_COLOR
    - CV_LOAD_IMAGE_GRAYSCALE

- **Display:**

```
cv::imshow(std::string windowName, cv::Mat image);
```

  - `windowName:` name of the window created.
  - `Image:` image to be displayed.

C → `IplImage`

- **Load:**

```
IplImage* m = cvLoadImage(char* filepath, int flags);
```

  - `Filepath:` Path to the image file.
  - `Flags:` How to laod the image:
    - CV_LOAD_IMAGE_COLOR
    - CV_LOAD_IMAGE_GRAYSCALE

- **Display:**

```
cvShowImage(char* windowName, IplImage* image);
```

  - `windowName:` name of the window created.
  - `Image:` image to be displayed.

Exercise 1 – load and display

- **Task:** load an image from the data folder and display it in a window.
- Open 'ElabImage_2010.sln', for today we will work on the 'Lab session 1' project.
- Modify 'lab_1.cpp' take a look at the code and fill the missing parts according to comments.
- Fill only the space marked as 'Exercise 1'.

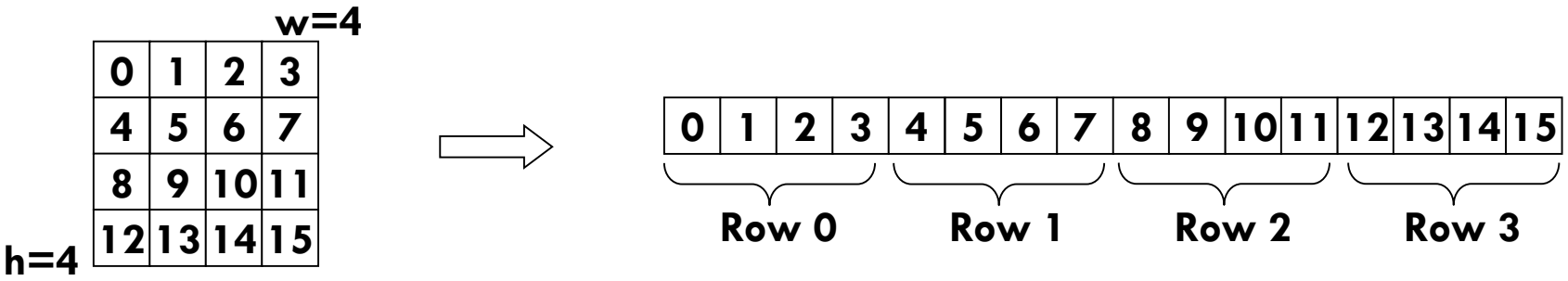
```
//=======================TO DO:EXERCISE 1=======================
//Load in source_image the image located at image_filepath

//==============================================================
```

Access to pixels

- Images: bidimensional entities, but stored in memory as **monodimensional vectors**, data access for reading/writing is done by means of only one index:

```
index = row * w + column
```

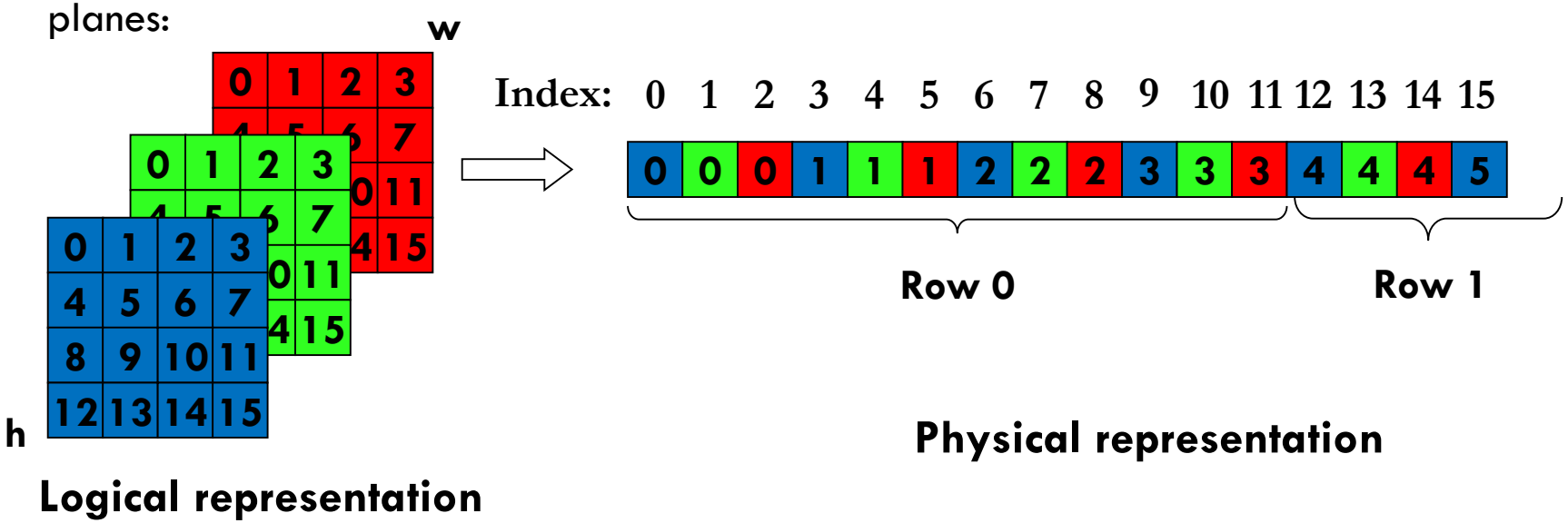
E.g. : the element in row 2, column 1 (n° 9) : 2 * w + 1 = 2 * 4 + 1 = 9



Logical representation **Physical representation**

Access to pixels: color images

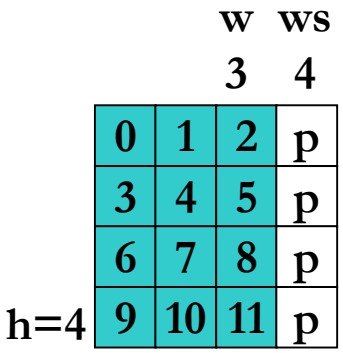
- Color images have 3 channels: Blue, Green, Red (**BGR**) stored as **interleaved** planes:



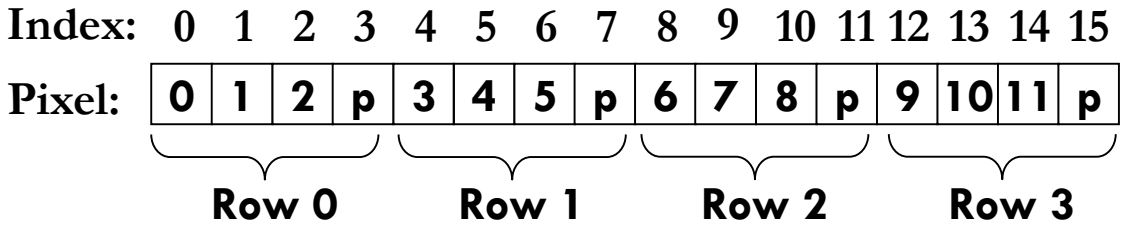
- To access a specific channel (color) of a pixel, the column index must be multiplied by 3 and summed to the appropriate offset:
  - B: `index = row*w*3 + column*3`
  - G: `index = row*w*3 + column*3 + 1`
  - R: `index = row*w*3 + column*3 + 2`

Access to pixels (in practice)

- OpenCV can store «padding» column with meaningless content to guarantee memory alignment of the image rows.
- Both `IplImage` and `cv::Mat` save the effective width in memory (Width Step) in internal field.



E.g. a 3x4 image = 12 elements; h= 4, w = 3, ws = 4.



- To access the 7th image element (coordinates (2,1), physical index 9):
  - by means of `width`: 2*w + 1 = 7 the red element is accessed (meaningless)
  - by means of `widthstep`: 2*ws + 1 = 9 the green element is accessed (correct)

Access to pixels (in practice)

C++ → `cv::Mat`

```
cv::Mat image;
int row, col;

datatype pixelValue =
((datatype*)image.data)
[row*image.step+col];
```

- `Datatype`: **type of data stored in the matrix**
  - `CV_8UC1` → `unsigned char` (grayscale)
  - `CV_8UC3` → `cv::Vec3b` (color image, array with three value B,G,R)

C → `IplImage`

```
IplImage * image;
Int row, col;

datatype pixelValue =
((datatype*)image->imageData)
[row*image->widthStep+col];
```

- `Datatype`: **type of data stored in the matrix**
  - `IPL_DEPTH_8U` → `unsigned char` (grayscale).
  - Each channel of a color image accessed with a different index.

cv::Mat Extra

- Templetized pixel access:

```
Datatype pixelValue = imagename->at<datatype>(row, col);
```

- Linear algebra operations carried out by re-definition of the operators, i.e. it is possible to use MATLAB-like syntax in our C++ programs:

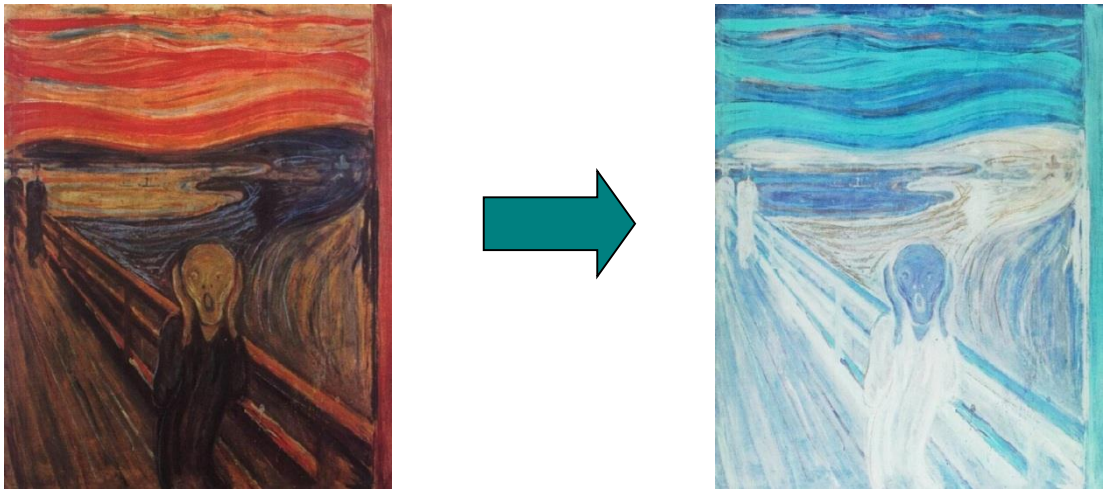
```
cv::Mat A(3,3,CV_8UC1);
cv::Mat B(3,3,CV_8UC1);
//element-wise subtraction
cv::Mat C = A - B;
//sets each element of C to 255 minus the corresponding element of A
C = 255 - A;
//Solve a over-complete linear system A x = b with the pseudo-inverse matrix algorithm
cv::Mat x = (A.t()*A).inv()*(A.t()*b);
```

Exercise 2 – “invert grey”

- Compute the “negative” of a grayscale image
- Given a grayscale image (range of each pixel between [0 255]), substitute each pixel having intensity I with the value: 255-I

Exercise 3 – “invert RGB”

- Same as before, but in this case we want to compute the negative of a color image.
- The image has 3 channels, representing the 3 RGB values
- The intensity of each channel ranges between [0 255]
- For each image pixel, we need to substitute the (B,G,R) triplet with its «inverse» (255-B, 255-G, 255-R)



Exercise 4 – Image difference

- Build a new VS project (see slide 7) which performs the following:
  - loads 2 images (Image 1, I1 and Image 2, I2)
  - computes the pixel-wise difference between the two images:
  - computes an output image where each pixel of coordinates (x,y) contains the absolute difference of the corresponding pixels on I1 and I2:

$$Out(x,y) = abs(I1(x,y) - I2(x,y))$$

  - Displays on a window the output image