UNIVERSITÀ DI PARMA

OpenCV
a brief introduction (for C++)

Summary

- OpenCV
- Installation
- Modules
- C pointers
- cv::Mat class + companions
- Few examples and simple.cpp skeleton

OpenCV

- OpenCV (Open Source Computer Vision Library) is an Open Source library for computer vision and machine learning
- BSD License (also commercial use!)
- Thousands of algorithms
- Tenth of thousands of users
- Millions of downloads
- **C++**, Python, JAVA, MATLAB support

OpenCV

- Main functionalities
  - Read/write images, sequences of images, or videos
  - Process images
    - Many off the shelf libraries
  - Graphic output

Installation

- Linux/gcc
- Two possibilities
  - Package manager
  - Download and compile sources
- Remember to install both core and contribs

Download & build

UNIVERSITÀ DI PARMA

- Prerequisites:
  - Development environment (C++, cmake, git)
  - Spefic packages (sudo apt install vtk7 libvtk7-dev)
- Use git for download
  - git clone https://github.com/opencv/opencv.git opencv
  - git clone https://github.com/opencv/opencv_contrib.git opencv-contribs

Download & build

- Build instructions:
  - mkdir opencv/build
  - cd opencv/build
  - cmake -DOPENCV_EXTRA_MODULES_PATH=../../opencv-contribs ..
    - Check errors and whether specific packages are installed (i.e. viz)
  - make -j8   #if memory issues, reduce the 8
  - **sudo make install**

Modules

- OpenCV main modules are:
  - Core, basic data structures:
    - Mat, Scalar, Point, Range…
  - Image processing, we will use some just to match our results
  - Video, motion estimation, tracking, background subraction…
  - Calib3d, camera calibration
  - Features2d, features extraction and matching
  - …

C pointers

UNIVERSITÀ DI PARMA

- It is an OpenCV slide presentation, isn't it?
- Yes but we need some recap about how to access memory…
- What is a C pointer?
  - Kind of data to store memory addresses
  - 32 bits/64 bits

C pointers

UNIVERSITÀ DI PARMA

- Address is simply a number
- Anyway C pointers feature a data type:
  - char *c $\longrightarrow$ pointer to a char data
  - float *f $\longrightarrow$ pointer to a float data
  - …
  - void *v $\longrightarrow$ pointer to something to be better specified

C pointers

- Why we need a data type for pointers?
- Basically for pointer arithmetics
- f=f+1 $\longrightarrow$ what is the result?
  - It depends on which kind of data is expected to be found at address f
  - If f is a char*, f=f+1 $\longrightarrow$ address f is increased by 1 byte
  - If f is a uint32_t, f=f+1 $\longrightarrow$ address f is increased by 4 bytes

C pointers

- How to access to the pointed data?
- When f is a pointer we can use *f
  - Both read/write
- Anyway usually we deal with large chunks of data ⟶ arrays
- To access the nth element we can use:
  - *(f+n) ⟶ old fashion, please avoid...
  - f[n]

C pointers

- f[n]
  - It makes sense only when f contains the address of a set of consecutives values
    - Monodimensional arrays → only one index
  - It works when *f type exactly matches the type of data stored at the f address

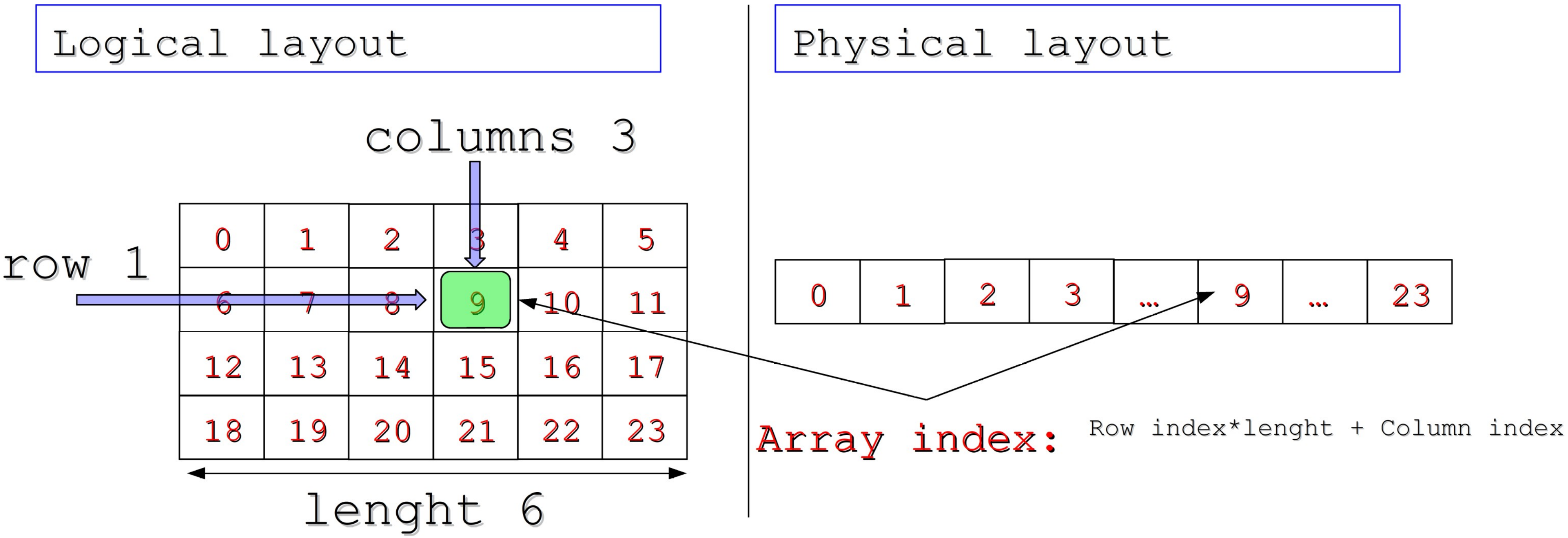
C pointers

- We already know that images are (at least) 2D structures
  - Two coordinates: column & row
- We can use pointers for that?
- Yes, we can use pointers to other pointers
  - char **c;
- If we consider other dimensions things get even creepier…
- Hint: do not do that!

C pointers

- Use simple array to deal with multidimensional matrices
- If we need to store n × m values:
  - data_type data[n*m];
- Access element at coordinates (x,y)
  - Considering that
    - rows are one after the other
    - Each row contains *m* elements
  - data[y*m + x]
- Logical representation vs Physical one

C pointers


Logical layout
Physical layout
columns 3
row 1
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
lenght 6
0 1 2 3 … 9 … 23
Array index:
Row index*lenght + Column index

cv::Mat

- Basic Image Container
- Two main elements:
  - Handler
    - Description of data
  - “Shared” pointer for data
    - Actual data pointer
    - Be careful! clone() and copyTo() methods
    - a=b (!)

cv::Mat constructors

UNIVERSITÀ DI PARMA

- cv::Mat()
- cv::Mat(int rows, int cols, int type)
- cv::Mat(int rows, int cols, int type, cv::Scalar s)
- cv::Mat(cv::Size size, int type)
- cv::Mat(cv::Size size, int type, cv::Scalar s)
- cv::Mat(const cv::Mat &m)
- cv::Mat(const cv::Mat &m, cv::Range rowRange)
- cv::Mat(const cv::Mat &m, cv::Range rowRange, cv::Range colRange)
- cv::Mat(const cv::Mat &m, cv::Rect roi)
- …

OpenCV types

CV –	C1	C2	C3	C4
8U	0	8	16	24
8S	1	9	17	25
16U	2	10	18	26
16S	3	11	19	27
32S	4	12	20	28
32F	5	13	21	29
64F	6	14	22	30

- Often used
  - CV_8UC1
    - greylevel images
  - CV_8UC3
    - RGB images
  - CV_32SCx or CV32FCx
    - result of different processings

Some othe cv:: classes

- cv::Scalar
  - Basically a short vector (up to 4) template
- cv::Rect
  - Template class for 2D rectangles
- cv::Range
  - Template class for a continuous subsequence

cv::Mat contruction examples

- `cv::Mat A, B;                                  // empty images`
- `cv::Mat C(A);                                  // copy (!)`
- `cv::Mat D(1024, 900, CV_8UC3)                  // set size/type`
- `cv::Mat E(A, Rect(10, 10, 100, 100));          // only part of A`
- `cv::Mat M(2,2, CV_8UC3, Scalar(0,0,255)); // also set pixel initial value`
- `cv::Mat F = A.clone();`
- `cv::Mat G;`
- `A.copyTo(G);`

cv::Mat M members/methods

M.rows	rows
M.cols	columns
M.channels()	channels
M.type()	image type (OpenCV type!)
M.elemSize()	pixel size (bytes)
M.elemSize1()	single channel size (bytes, <= M.elemSize())

- i.e. RGB8
  - M.channels() == 3
  - M.elemSize() == 3
  - M.elemSize1() == 1
  - M.type() == CV_8UC3 3 channels, 1 byte/channel

cv::Mat M

- Where is my image?
- **uchar *cv::Mat::data** can be used
  - Sort of shared pointer
- M.data $\longrightarrow$ address of image buffer
- M.data $\longrightarrow$ points to first image byte
- It does not depend on pixel type
  - Cast can be needed

Example #1

UNIVERSITÀ DI PARMA

- Bare image access

```
cv::Mat M;
...
for(size_t i =0;i<M.rows*M.cols*M.elemSize();++i)
    M.data[i] = i;
```

Example #2

- Single channel access

```
cv::Mat M;
...
for(size_t i =0;i<M.rows*M.cols;i+=M.elemSize())
    {
     M.data[i] = i;                                  //B
     M.data[i+M.elemSize1()] = i + 1;                //G
     M.data[i+M.elemSize1()+M.elemSize1()] = i + 2; //R
}
```

Example #3

- Row/Column access

```
cv::Mat M;
...
for(size_t v = 0; v<M.rows; ++v)
   {
    for(size_t u = 0; u<M.cols; ++u)
       {
        M.data[(u + v*M.cols)*M.elemSize()] = u;                   //B
        M.data[(u + v*M.cols)*M.elemSize() + M.elemSize1()] = u+1; //G
        M.data[(u + v*M.cols)*M.elemSize() + M.elemSize1()] = u+2; //R
       }
   }
```

Example #3

- Row/Column/Channel (1 byte) access

```
cv::Mat M;
...
for(size_t v = 0; v < M.rows; ++v)
     {
      for(size_t u = 0; u < M.cols; ++u)
         {
           for(size_t k = 0; k < M.channels(); ++k)
            {
              M.data[(u + v*M.cols)*M.channels() + k] = u + k;
            }
         }
     }
```

cv::Mat other access methods

- To access specific row:
  - uchar * cv::Mat::ptr(int i)
  - Allows to access buffer at row i
- Actually a template
  - T * cv::Mat::ptr<T>(int i)
- Also single pixel can be referenced:
  - T cv::Mat::at<T>(row=0,col=0)[channel]
  - Allows to access to value/address
  - Do not use it before first homework

simple.cpp

UNIVERSITÀ DI PARMA

- Skeleton for… everything?
- Prerequisites:
  - OpenCV
  - g++
  - cmake + make
- Build:

```
mkdir build; cd build
cmake ..
make
```

- Enjoy!