Mat, Pixel type

Sung Soo Hwang

Basic Data Structure

- Mat
  - Declaration
    - Mat (int rows, int cols, int type)
    - Mat (Size size, int type)
    - Mat (const Mat & m)
    - Mat (Size size, int type, const Scalar& s)

Basic Data Structure

- Pixel type
  - CV_8U: 8-bit unsigned integer: uchar (0~255)
  - CV_8S: 8-bit signed integer: schar (-128~127)
  - CV_16U: 16-bit unsigned integer: ushort (0~65535)
  - CV_16S: 16-bit signed integer: short (-32768~32767)

Basic Data Structure

- Pixel type
  - CV_32S: 32-bit signed integer:
    int (-2147483648~2147483647)
  - CV_32F: 32-bit floating-point number:
    float (-FLT_MAX~FLT_MAX, INF, NAN)
  - CV_64F: 64-bit floating-point number:
    double (-DBL_MAX~ DBL_MAX, INF, NAN)
  - Multi-channel array:
    CV_8UC3, CV_8U(3), CV_64FC4, CV_64FC(4)

Basic Data Structure

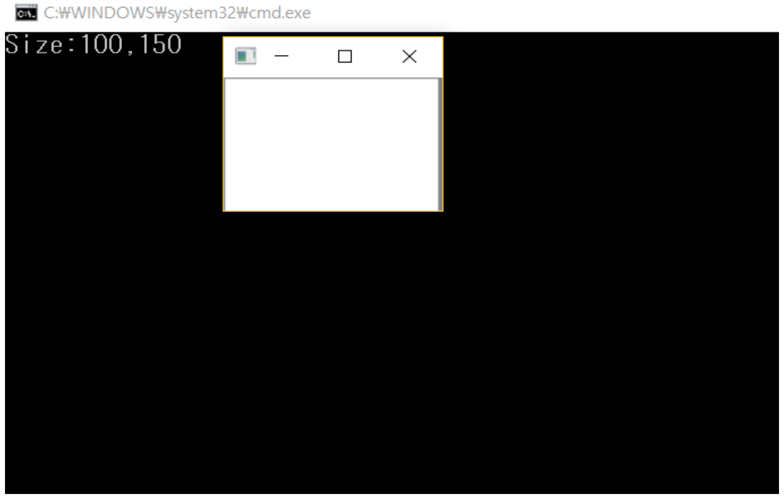
- Matrix declaration

```
Mat mtx(3, 3, CV_32F);
// make a 3x3 floating-point matrix

Mat mtx(10, 1, CV_64FC2);
 // make a 10x1 2-channel floating-point matrix
(10-element complex vector)

Mat img(1080, 1920, CV_8UC3);
 // make a 3-channel (color) image of 1920 columns and 1080 rows.

Mat img(Size(1920, 1080), CV_8UC3);
 // make a 3-channel (color) image of 1920 columns and 1080 rows.
```

Basic Data Structure

- Matrix declaration

```
int main()
{
        int w = 150, h = 100;
        Mat image(h, w, CV_8UC1, Scalar(255));
        cout << "Size: " << image.size().height << "," << image.size().width << endl;
        imshow("image", image);

        waitKey(0);
        return 0;
}
```



C:₩WINDOWS₩system32₩cmd.exe

```
Size:100,150
```

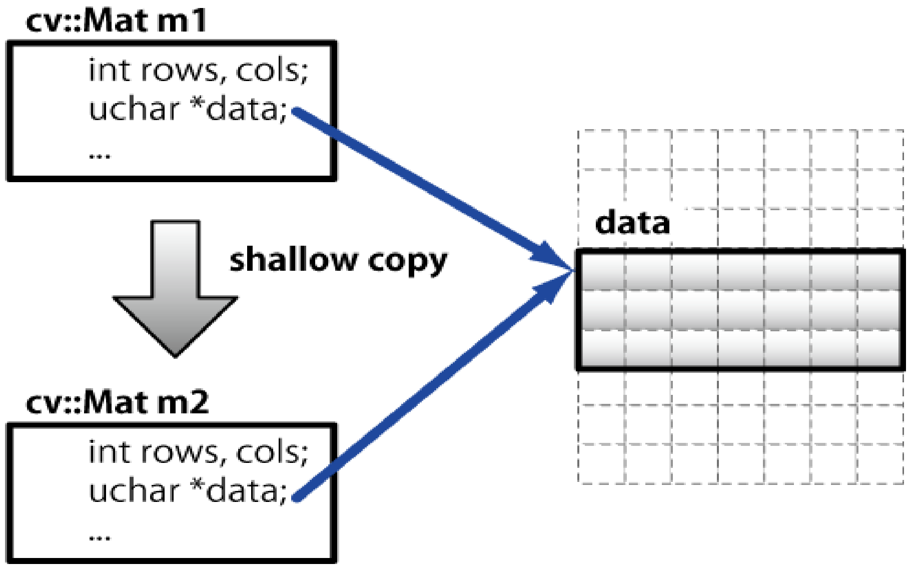
- For a multi-channel image use Scalar function as
  - Scalar(255, 0, 0)
  - Color order in OpenCV is Blue, Green, and Red ➔ BGR

Mat copy & conversion

Sung Soo Hwang

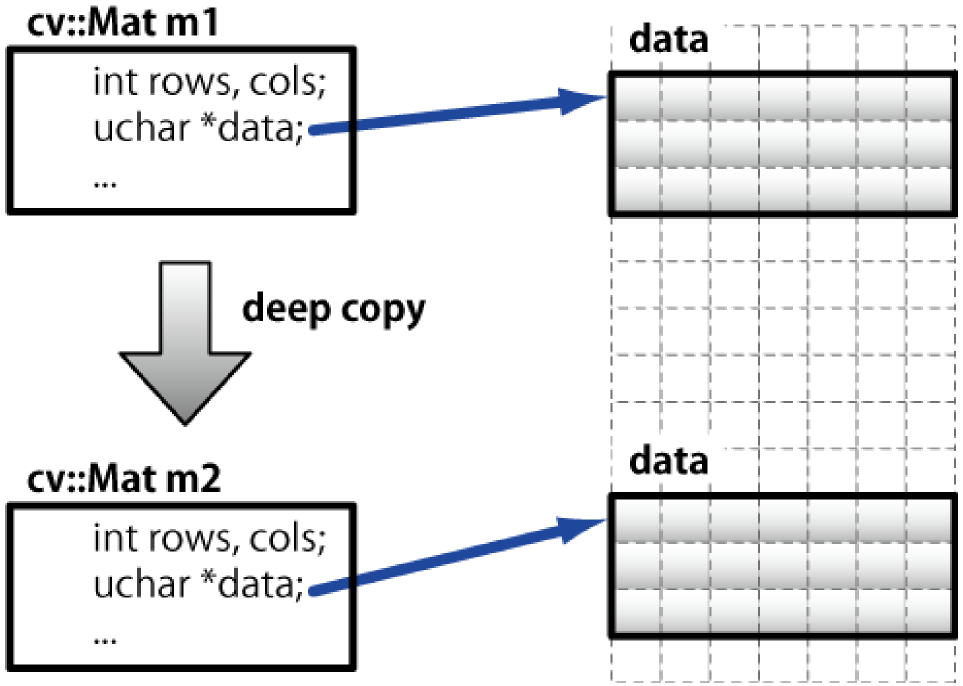
Mat copy

- Shallow copy
  - Mat data structure consists of header and data
  - In case of shallow copy, the address for data is copied
  - Use = for shallow copy



Mat copy

- Deep copy
  - Use clone() for deep copy
  - Mat creation and copyTo() are performed inside clone()



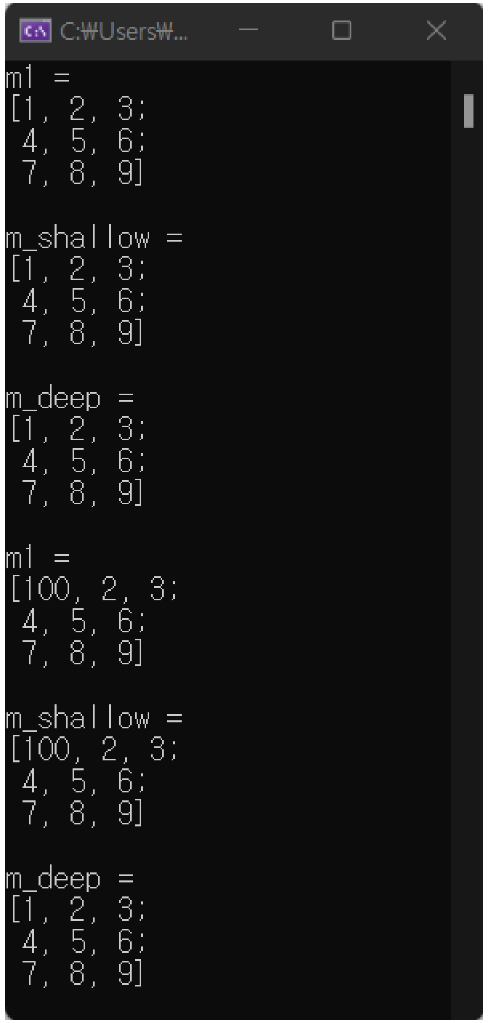
Mat copy

- copyTo
  - void copyTo(OutputArray m, InputArray mask)
    - m: Destination matrix. If it does not have a proper size or type before the operation, it is reallocated.
    - mask: Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels.

Mat copy

- Example

```
int main() {
        Mat m1 = (Mat_ < double >(3, 3)
                << 1, 2, 3, 4, 5, 6, 7, 8, 9);

        Mat m_shallow = m1;
        Mat m_deep = m1.clone();

        cout << "m1 =₩n" << m1 << endl << endl;
        cout << "m_shallow =₩n" << m_shallow << endl << endl;
        cout << "m_deep =₩n" << m_deep << endl << endl;

        // Update m1
        m1.at < double >(0, 0) = 100;
        cout << "m1 =₩n" << m1 << endl << endl;
        cout << "m_shallow =₩n" << m_shallow << endl << endl;
        cout << "m_deep =₩n" << m_deep << endl << endl;

        waitKey(0);
}
```



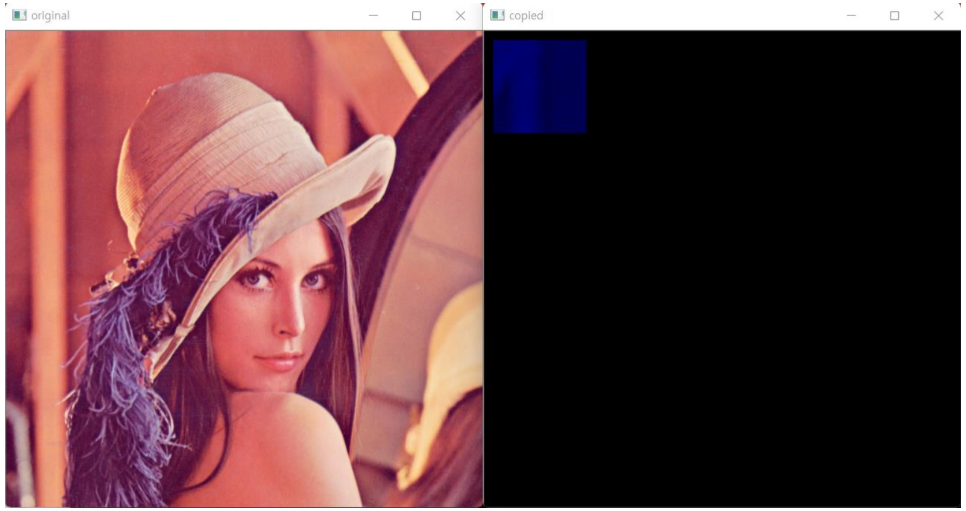
```
m1 =
[1, 2, 3;
 4, 5, 6;
 7, 8, 9]

m_shallow =
[1, 2, 3;
 4, 5, 6;
 7, 8, 9]

m_deep =
[1, 2, 3;
 4, 5, 6;
 7, 8, 9]

m1 =
[100, 2, 3;
 4, 5, 6;
 7, 8, 9]

m_shallow =
[100, 2, 3;
 4, 5, 6;
 7, 8, 9]

m_deep =
[1, 2, 3;
 4, 5, 6;
 7, 8, 9]
```

Mat copy

- Example

```
int main() {
        Mat image = imread("lena.png", 0);
        Mat copied_img = image.clone();
}
```

```
int main() {
   Mat image = imread("lena.png");
   Mat mask = Mat::zeros(image.size(), image.type());
   Mat copied;
   Rect rect = Rect(10, 10, 100, 100); // LT position, width, height
   rectangle(mask, rect, Scalar(255, 0, 0), -1, 8, 0);
   image.copyTo(copied, mask);
   imshow("original", image);
   imshow("copied", copied);
   waitKey(0);
   return 0;
}
```



Mat Conversion

- Matrix conversion
  - Mat convertTo(OutputArray m, int rtype, double alpha=1, double beta=0)
    - m : output matrix; if the size or type is not proper, it is reallocated
    - rtype: desired output matrix
    - m(x, y) = saturate_cast<rType> (alpha * (*this)(x, y) + beta)

Mat Conversion

- Matrix conversion
  - Mat setTo(InputArray value, InputArray mask=noArray()
    - Sets all or some of the array elements to the specified value
    - Mask : Operation mask of the same size as *this
    - Same as Operator = (const Scalar &s)

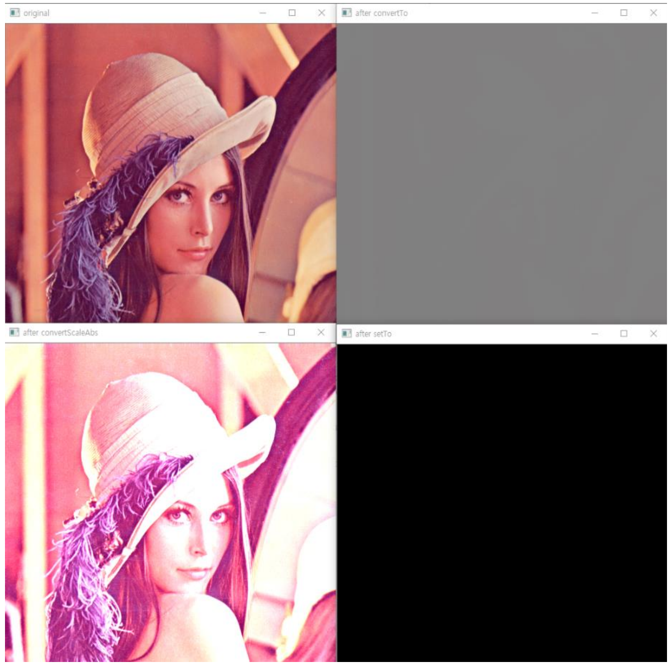
Mat Conversion

- Matrix conversion
  - void convertScaleAbs(InputArray src, OutputArray dst, double alpha=1, double beta=0)
    - Src – input array
    - Dst – output array
    - Alpha – optional scale factor
    - Beta – optional delta added to the scaled values
    - Dst(I) = saturate_cast<uchar>(|src(I)| * alpha + beta|)

Mat Conversion

- Example code

```
int main() {
    Mat image = imread("lena.png");
    Mat after_convertTo, after_convertScaleAbs;

    imshow("original", image);

    image.convertTo(after_convertTo, CV_16SC1);
    imshow("after convertTo", after_convertTo);

    convertScaleAbs(image, after_convertScaleAbs, 2, 3);
    imshow("after convertScaleAbs", after_convertScaleAbs);

    image.setTo(Scalar(0));
    imshow("after setTo", image);

    waitKey(0);
}
```



Read, display and resize Image/Video

Sung Soo Hwang

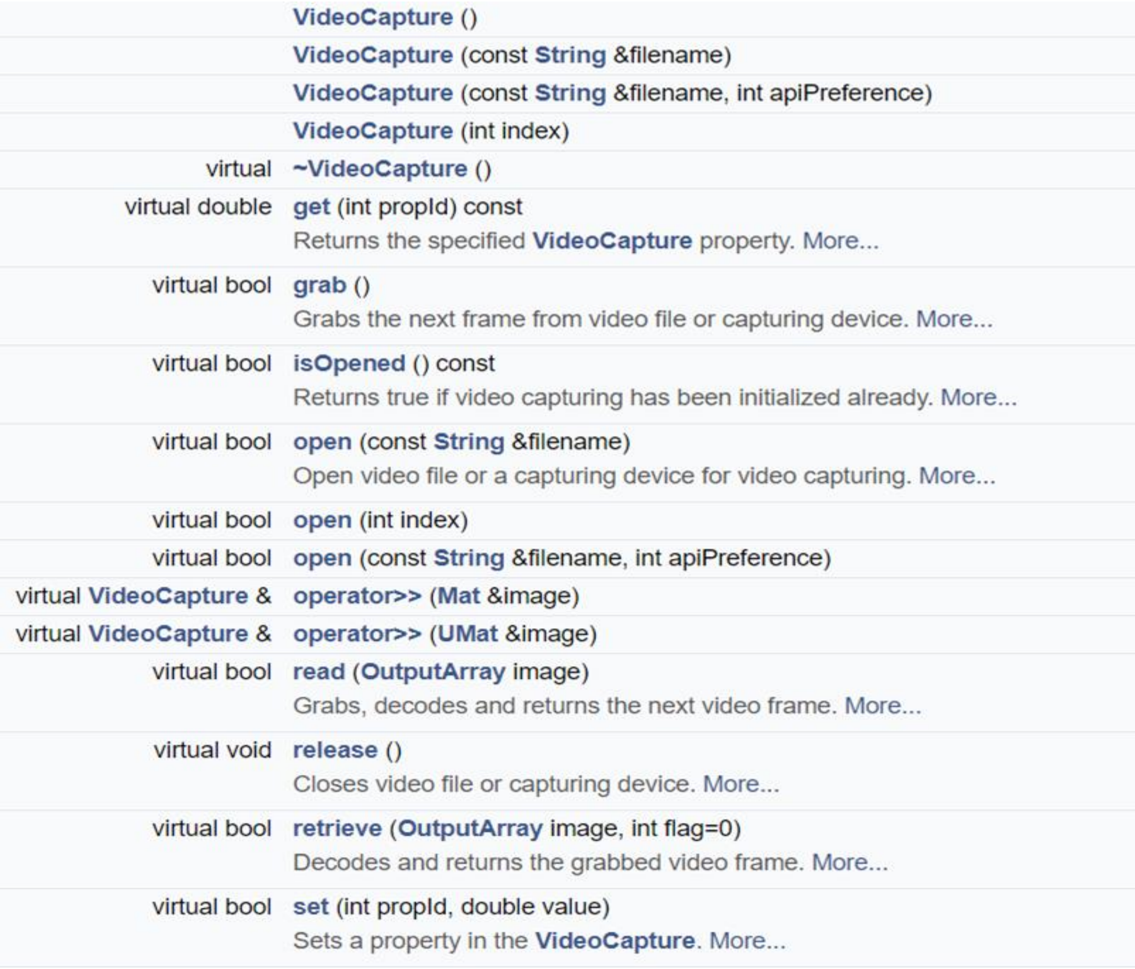
Read Image/Video

- Read an image
  - Mat imread(const string& filename, int flags=1)
    - Flag value as 1:read image as color image
    - Flag value as 0:read image as gray scale image

```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat gray_image, color_image;

   // 0 on the 2nd parameter means read img in grayscale
   gray_image = imread("lena.png", 0);

   // blank 2nd parameter means 1, which means read img in colors
   color_image = imread("lena.png");

   return 0;
}
```

Read Image/Video

- Read a video
  - Use VideoCapture Class
  - Methods in VideoCapture Class



	VideoCapture ()
	VideoCapture (const String &filename)
	VideoCapture (const String &filename, int apiPreference)
	VideoCapture (int index)
virtual	~VideoCapture ()
virtual double	get (int propId) const Returns the specified VideoCapture property. More...
virtual bool	grab () Grabs the next frame from video file or capturing device. More...
virtual bool	isOpened () const Returns true if video capturing has been initialized already. More...
virtual bool	open (const String &filename) Open video file or a capturing device for video capturing. More...
virtual bool	open (int index)
virtual bool	open (const String &filename, int apiPreference)
virtual VideoCapture &	operator>> (Mat &image)
virtual VideoCapture &	operator>> (UMat &image)
virtual bool	read (OutputArray image) Grabs, decodes and returns the next video frame. More...
virtual void	release () Closes video file or capturing device. More...
virtual bool	retrieve (OutputArray image, int flag=0) Decodes and returns the grabbed video frame. More...
virtual bool	set (int propId, double value) Sets a property in the VideoCapture. More...

Read Image/Video

- VideoCapture::get(int propId) – propId
  - **CAP_PROP_POS_MSEC** Current position of the video file in milliseconds or video capture timestamp.
  - **CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
  - **CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
  - **CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
  - **CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.

Read Image/Video

- VideoCapture::get(int propId) – propId
  - **CAP_PROP_FPS** Frame rate.
  - **CAP_PROP_FOURCC** 4-character code of codec.
  - **CAP_PROP_FRAME_COUNT** Number of frames in the video file.
  - **CAP_PROP_FORMAT** Format of the Mat objects returned by retrieve() .
  - **CAP_PROP_MODE** Backend-specific value indicating the current capture mode.

Read Image/Video

- VideoCapture::get(int propId) – propId
  - **CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
  - **CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
  - **CAP_PROP_SATURATION** Saturation of the image (only for cameras).
  - **CAP_PROP_HUE** Hue of the image (only for cameras).
  - **CAP_PROP_GAIN** Gain of the image (only for cameras).

Read Image/Video

- VideoCapture::get(int propId) – propId
  - **CAP_PROP_EXPOSURE** Exposure (only for cameras).
  - **CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
  - **CAP_PROP_WHITE_BALANCE** Currently not supported
  - **CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras
    (note: only supported by DC1394 v 2.x backend currently)

Read Image/Video

- Read a video from a file

```
#include “opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat frame;
   VideoCapture cap;

   // check if file exists. if none program ends
   if (cap.open("background.mp4") == 0) {
      cout << "no such file!" << endl;
      waitKey(0);
   }

   double fps = cap.get(CAP_PROP_FPS);
   double time_in_msec = cap.get(CAP_PROP_POS_MSEC);
   int total_frames = cap.get(CAP_PROP_FRAME_COUNT);
}
```

Image/Video Display

- Read a video from a webcam
  - Example

```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat frame;
   // capture from webcam
   // whose device number=0
   VideoCapture cap(0);
}
```

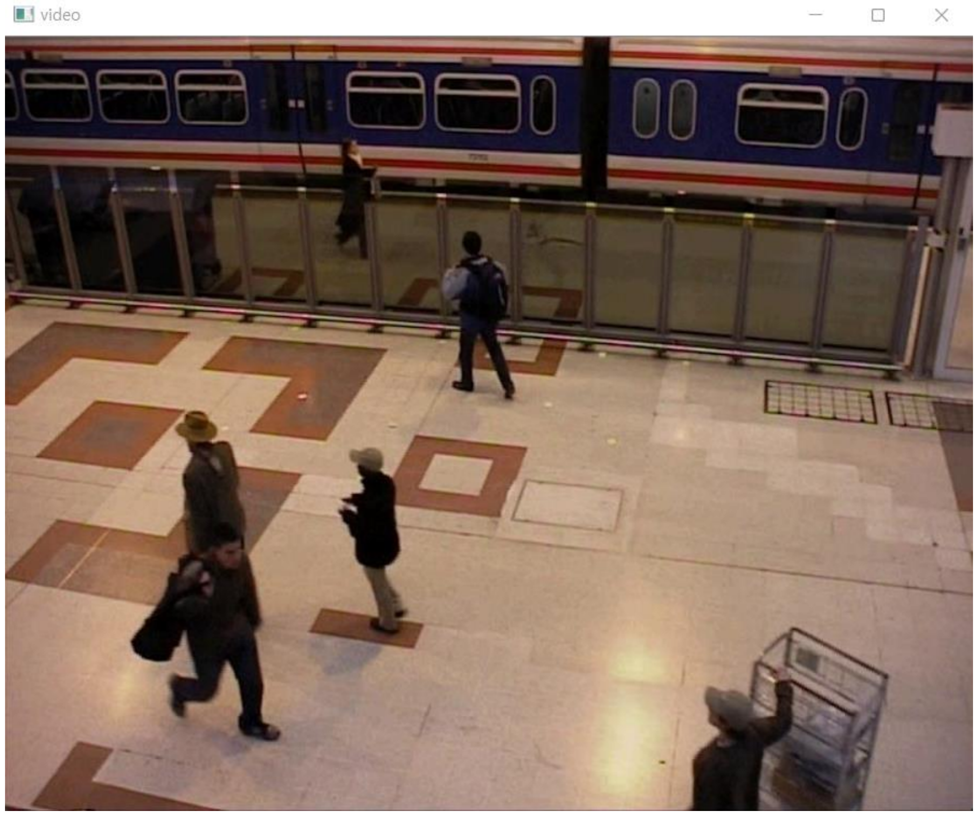
Image/Video Display

- Display an image
  - void imshow(const string &winname, InputArray mat)
    - winname: Name of the window
    - mat: image to be shown

```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat gray_image, color_image;

   // 0 on the 2nd parameter means read img in grayscale
   gray_image = imread("lena.png", 0);

   // blank 2nd parameter means 1, which means read img in colors
   color_image = imread("lena.png");

   imshow("gray image", gray_image);
   imshow("color image", color_image);

   waitKey(0);
   return 0;
}
```

Image/Video Display

- Display a video



```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat frame;
   int fps;
   int delay;
   VideoCapture cap;

   // check if file exists. if none program ends
   if (cap.open("background.mp4") == 0) {
      cout << "no such file!" << endl;
      waitKey(0);
   }

   fps = cap.get(CAP_PROP_FPS);
   delay = 1000 / fps;
   while (1) {
      cap >> frame;
      if (frame.empty()) {
         cout << "end of video" << endl;
         break;
      }
      imshow("video", frame);
      waitKey(delay);
   }
}
```

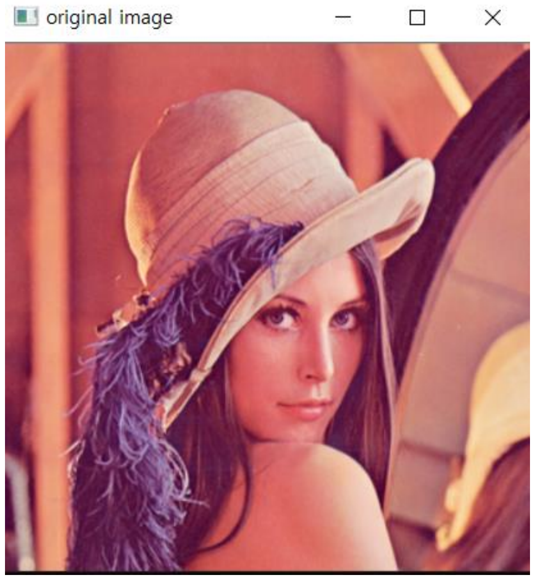
Image/Video Display

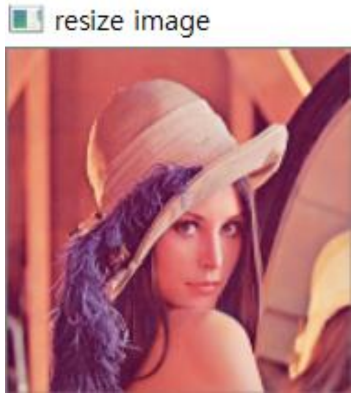
- int waitKey(int delay=0)
  - Delay in milliseconds.
  - 0 is the special value that means “forever”

Resize Image/Video

- Resize an image
  - resize(Mat src, Mat dst, Size(cols, rows)
    - src: input image
    - dst: output image
    - Size(cols, rows) : Size of image to convert

```
#include "opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main() {
   Mat img = imread(“lena.png”);
   Mat resize_img;
   resize(img, resize_img, Size(200, 200));
   imshow(“original image”, img);
   imshow(“resize image”, resize_img);
   waitKey(0);
   return 0;
}
```


original image


resize image