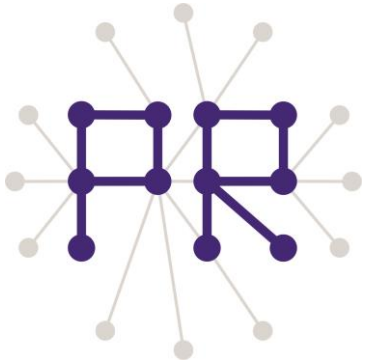
Multimedia Retrieval Exercise Course

2 Basic of Image Processing by OpenCV

Kimiaki Shirahama, D.E.

Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany

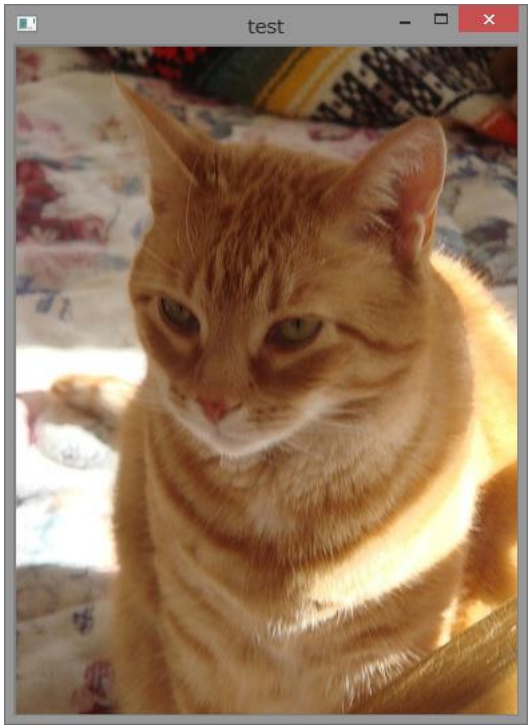


Last lesson's Program

- Read an image and show it in a dialog

```
int main(int argc, char* argv[]){
    IplImage* img = cvLoadImage("C:\\opencv\\samples\\c\\cat.jpg");
    cvNamedWindow("test", CV_WINDOW_AUTOSIZE);
    cvShowImage("test", img );
    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
```

The image filename assumes Windows. Please change it depending on your PC's OS.



Overview of Today's Lesson

1. IplImage
   - Structure of IplImage
   - cvLoadImage
   - cvCreateImage
   - cvReleaseImage
   - Access to image pixels
2. Useful functions in HighGUI
   - cvNamedWindow
   - cvShowImage
   - cvWaitKey
3. Examples of simple image processing
   - Image resize
   - Color space conversion
   - Edge detection

OpenCV Information

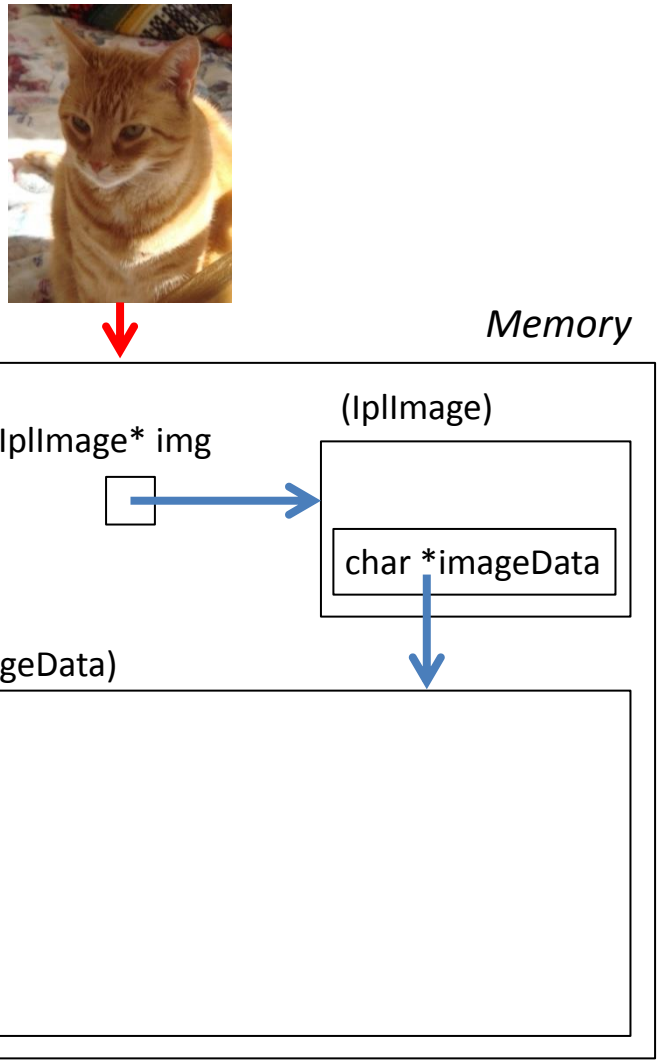
Useful web page: http://opencv.jp/opencv-1.1.0_org/docs/index.htm

- CXCORE: Functions for basic data structures in OpenCV
- CV: Functions for image processing and analysis
- ML (Machine Learning): Functions for statistical classification,
- regression and clustering
- HighGUI: High-level GUI and Media I/O

IplImage

Structure of the header for representing an image on the memory

```
typedef struct _IplImage {

  // Variables regarding properties of an image
  int width;
  int height;
  int imageSize;
  int nChannels;

  // Variables regarding how an image is stored
  char *imageData;
  int origin;
  int depth;
  int widthStep;

  // Variable regarding a specified region in an image
  struct _IplROI *roi

  …(many other variables)

} IplImage;
```

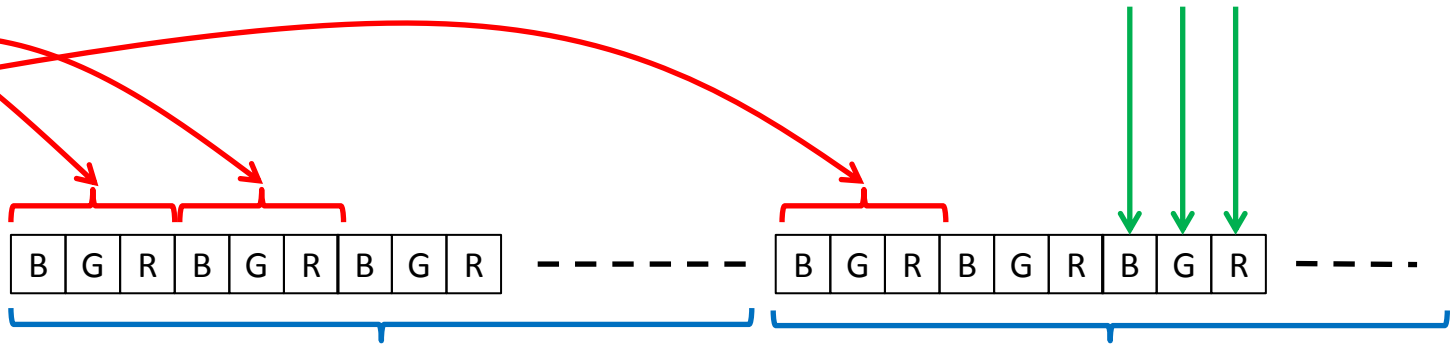


IplImage->imageData

```
typedef struct _IplImage {
  …
  // Variables regarding how an image is stored
  char *imageData; // ONE-dimensional array for storing pixel values
  int origin; // Which pixel is represented by the first element in imageData
  int depth; // How many bits are used to represent the value of one channel in each pixel
  int widthStep; // How many bits are used to represent pixels in one row
  …
} IplImage;
```

By default, origin = 0 (top-left origin)

The size of each element (B G R) is specified by depth
(IPL_DEPTH_8U (default): Each element is 8bit unsigned char)



Useful and Necessary Functions for IplImage

1. IplImage* cvLoadImage(const char* filename, int flags=CV_LOAD_IMAGE_COLOR)

CV_LOAD_IMAGE_COLOR is usually OK. If you want to load an image in the grayscale mode, use CV_LOAD_IMAGE_GRAYSCALE.

2. IplImage* cvCreateImage(CvSize size, int depth, int channels)

This function is often used to allocate an IplImage region, to which the already loaded image (pointed by IplImage*) is copied or converted. Thus, by imaging the copied or converted region, input variables should be appropriately set.
-> This function will be used later.

3. void cvReleaseImage(IplImage image)**

This function is very important for avoiding “out-of-memory”. Allocated IplImage regions should be appropriately released by yourself. It should be noted that the input variable is the pointer of the pointer of IplImage!

Checking variables in IplImage

Please try to add some sentences to the last lesson's code, in order to check the width, height, and nChannels of IplImage.

```
cout << "Width:" << img->width << ", Height:" << img->height << ", # of channels:" << img->nChannels << endl;
```

Some tips

1. While the most basic function for outputing texts to a terminal is "printf" of C, "cout" of C++ is a more convenient because it can automatically recognise the variable types. The summary of using "cout" is as follows:

```
#include <iostream>
using namespace std; // If this is missing, "cout" has to be written as "std::cout"
```

NOTE: Variable type recognition does not work for unsigned char!

2. For windows users, due to the specification of Visual C++, a terminal will quickly disappear after reaching the end of a program. To keep the terminal appearing, one of the simplest approach is as follows:

```
int a = 0;
cin >> a;  // Wait until some value is input into "a"
```

Accessing Pixels in Images though IplImage

Please try to print the value of each pixel, using “for” loop based on the structure of IplImage->imageData

```
unsigned char p[3];
for(int y = 0; y < img->height; y++){
   for(int x = 0; x < img->width; x++){
      p[0] = img->imageData[img->widthStep * y + x * img->nChannels];
      p[1] = img->imageData[img->widthStep * y + x * img->nChannels + 1];
      p[2] = img->imageData[img->widthStep * y + x * img->nChannels + 2];
      printf("(x:%d, y:%d) -> B:%u, G:%u, R:%u\n", x, y, p[0], p[1], p[2]);
      // If you remove the comment symbol (//) for the following lines, you can change pixel values
      //img->imageData[img->widthStep * y + x * img->nChannels] = 255; // B
      //img->imageData[img->widthStep * y + x * img->nChannels + 1] = 0; // G
      //img->imageData[img->widthStep * y + x * img->nChannels + 2] = 0; // R
   }
}
```

Some tips

1. Use “printf” (“cout” does not work)
2. Store pixel values into an array of “unsigned char”

Useful Functions in HighGUI

1. int cvNamedWindow(const char* name, int flags=CV_WINDOW_AUTOSIZE)

The first variable specifies the name of the window where an image is shown.

2. void cvShowImage(const char* name, const CvArr* image)

Using the first variable, you can specify which window is used to show an image

3. void cvDestroyWindow(const char* name)

Release the memory used for the window

4. int cvWaitKey(int delay=0)

The function is used to wait for some key input or “delay” milliseconds. The return value represent which key is pressed. Returns the code of the pressed key

Simple Image Processing (1)
- Image Resize -

void cvResize(const CvArr* src, CvArr* dst, int interpolation=CV_INTER_LINEAR)
The first and second variables represent the source and resized image (which can be Specified by IplImage*). The last variable indicates what kind of approximation is conducted to create the resized image.

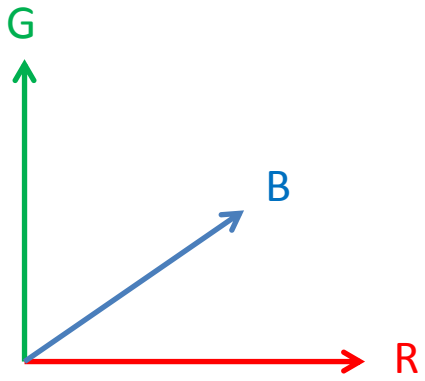
```
// Allocate the memory region where the resized image will be stored
// In this case, I am creating the image with the half size of the original image
IplImage* img_s = cvCreateImage(cvSize(img->width/2,img->height/2), img->depth, img->nChannels);

cvResize(img, img_s, CV_INTER_CUBIC);
```

Simple Image Processing (2)
- Color Space Conversion -

RGB color space

- Red
- Green
- Blue

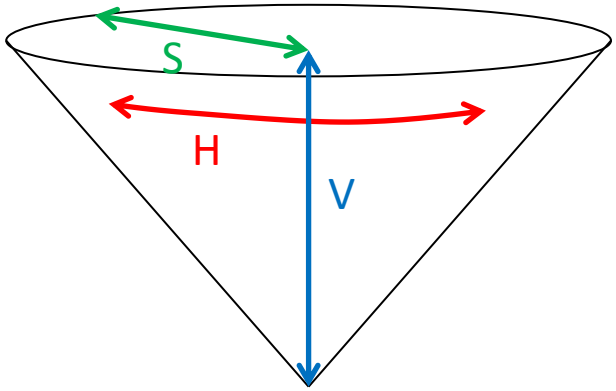
Not intuitive for humans



HSV color space

- Hue: Primary color
- Saturation: Effect of white
- Value: Darkness (effect of black)

More intuitive for humans



void cvCvtColor(const CvArr* src, CvArr* dst, int code)

The first and second variables represent the source and result image (which can be specified by IplImage*). The last variable indicates what kind of color conversion is conducted.

```
// Allocate the memory region where the HSV image will be stored. It has three channels (H, S and V)
IplImage* img_hsv = cvCreateImage(cvSize(img->width,img->height), img->depth, img->nChannels);
cvCvtColor(img, img_hsv, CV_BGR2HSV);
```

Simple Image Processing (3)
- Edge Detection -

void cvCanny(const CvArr* image, CvArr* edges, double threshold1,
double threshold2, int aperture_size=3)

The first and second variables represent the source and result image (which can be specified by IplImage*). The third variable is used to examine the connection of edges. The forth variable is used to detect strong edges in the initial process. The last variable specifies the size of Sobel filter. Please test various values.

```
// Allocate the memory region which will store the black-and-white image (one channel) showing detected edges
IplImage* img_edge = cvCreateImage(cvSize(img->width,img->height), img->depth, 1);
cvCanny(img, img_edge, 50.0, 200.0);
```