Computer Vision

13016370

Edited by:

Dr. Ukrit Watchareeruetai

International College

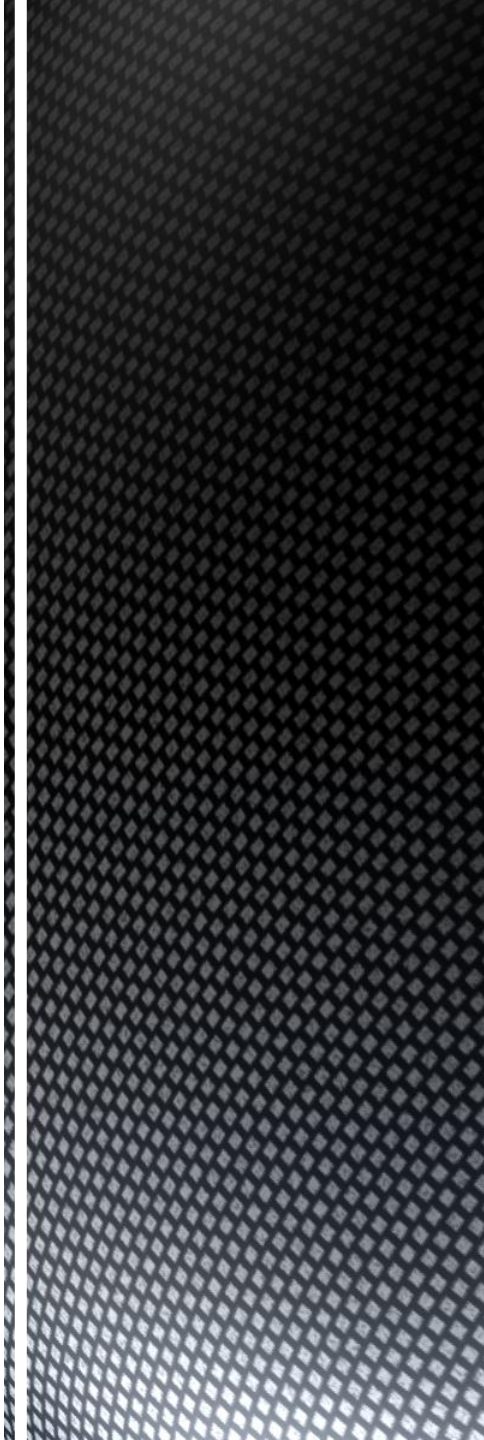
King Mongkut's Institute of Technology Ladkrabang

Lecture 2

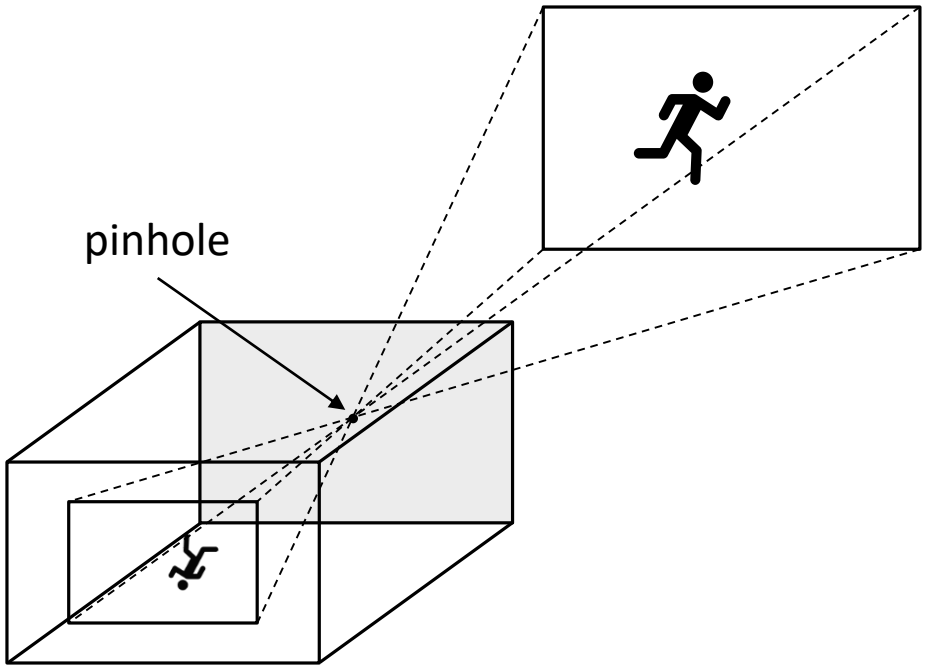
Representation of digital images

- Image acquisition
- Digital images
- Images in OpenCV

Image acquisition

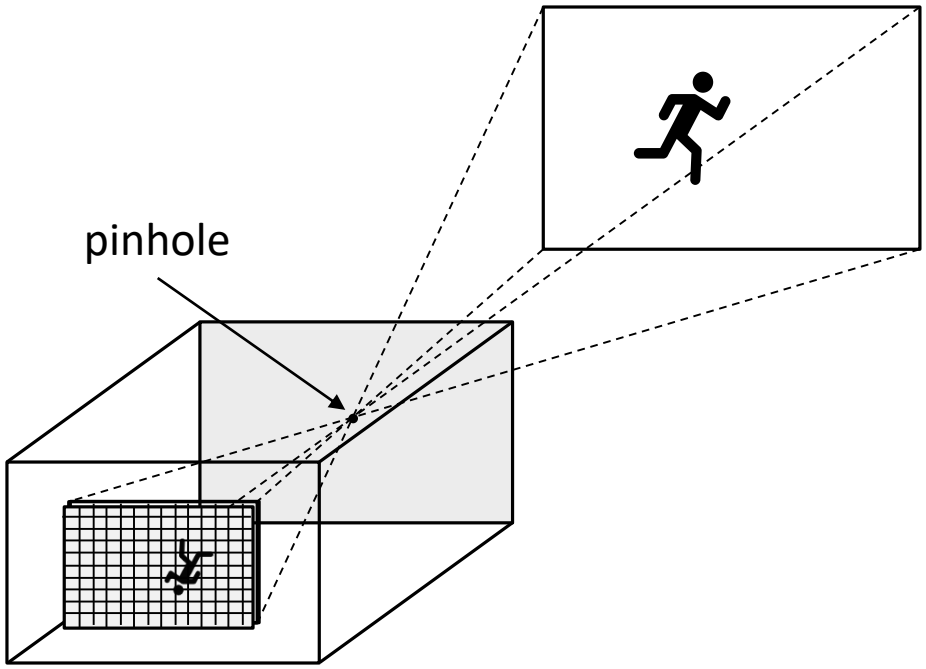


Pinhole camera model



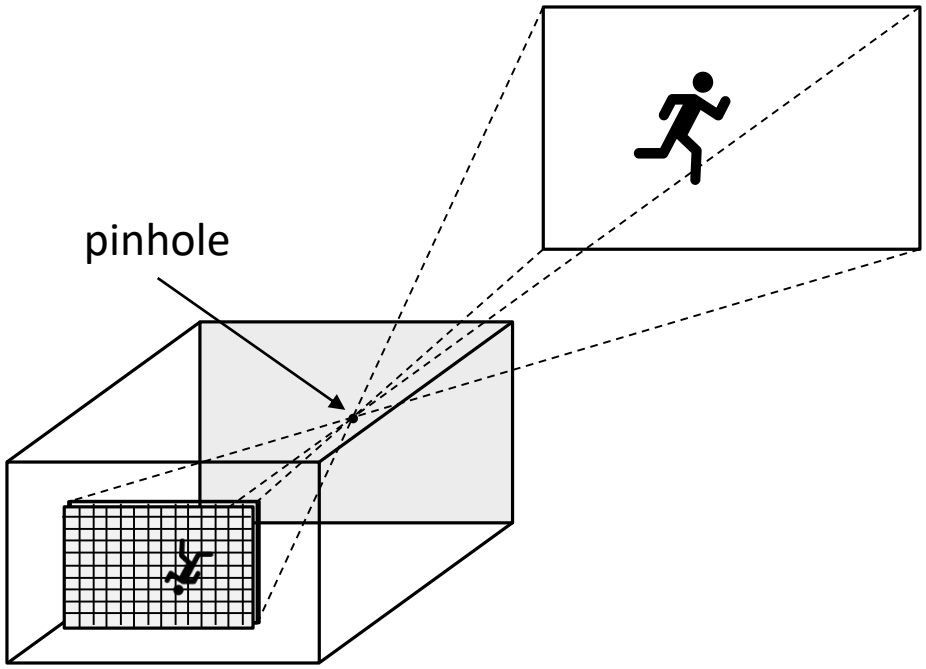
- A simple model of camera
  - No lens is required
- A lightproof box with a small hole
- Light from a object passes through the hole and projects on the other side of the box.
- An inverted image of the object is formed.
- This is considered a **continuous image signal**.

Image digitalization



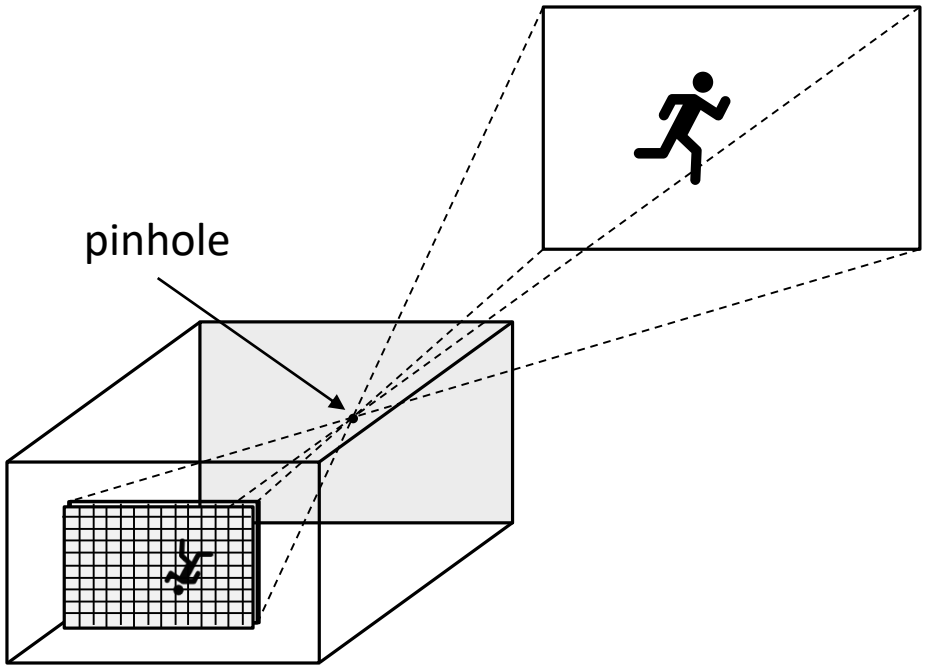
- Analog cameras:
  - Use photographic film to convert the projected light into an image.
- Digital cameras:
  - Use **image sensor** instead
    - Rectangular grid of photosensors
    - Place on the opposite side of the box
  - **Image digitalization** is performed to create a digital image
    - Consist of two main processes:
      - **Sampling**
      - **Quantization**

Sampling



- Each photosensor takes sample of the projected light at a certain location defined by the grid.
  - Each sample corresponds to one image element, i.e., **pixel**.
- All samples are converted into a matrix with H rows and W columns.
  - These two values control the quality of sampling.
  - Known as **spatial resolution** or **pixel resolution**
  - E.g., 800 × 600 pixels

Quantization



- Each continuous sample is then quantized to a discrete value.
  - Use a finite number of bits to store the value
  - If b bits is used, $K = 2^b$ different values can be represented.
    - The continuous range of data is divided into *K* intervals.
    - Control the quality of this process
    - Known as **bit resolution**
    - E.g., 8-bit grayscale image, 24-bit RGB image

256 × 256

128 × 128

64 × 64

32 × 32

J.C. Russ, The Image Processing Handbook, 6th Edition, CRC Press, 2011

32 gray levels

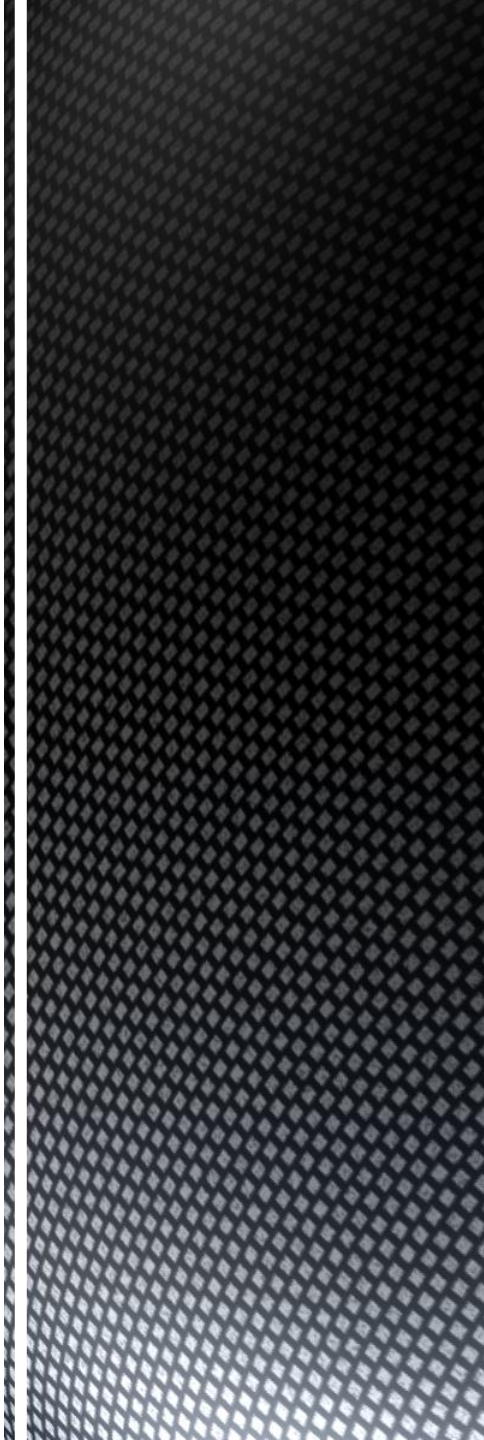
16 gray levels

8 gray levels

4 gray levels

J.C. Russ, The Image Processing Handbook, 6th Edition, CRC Press, 2011

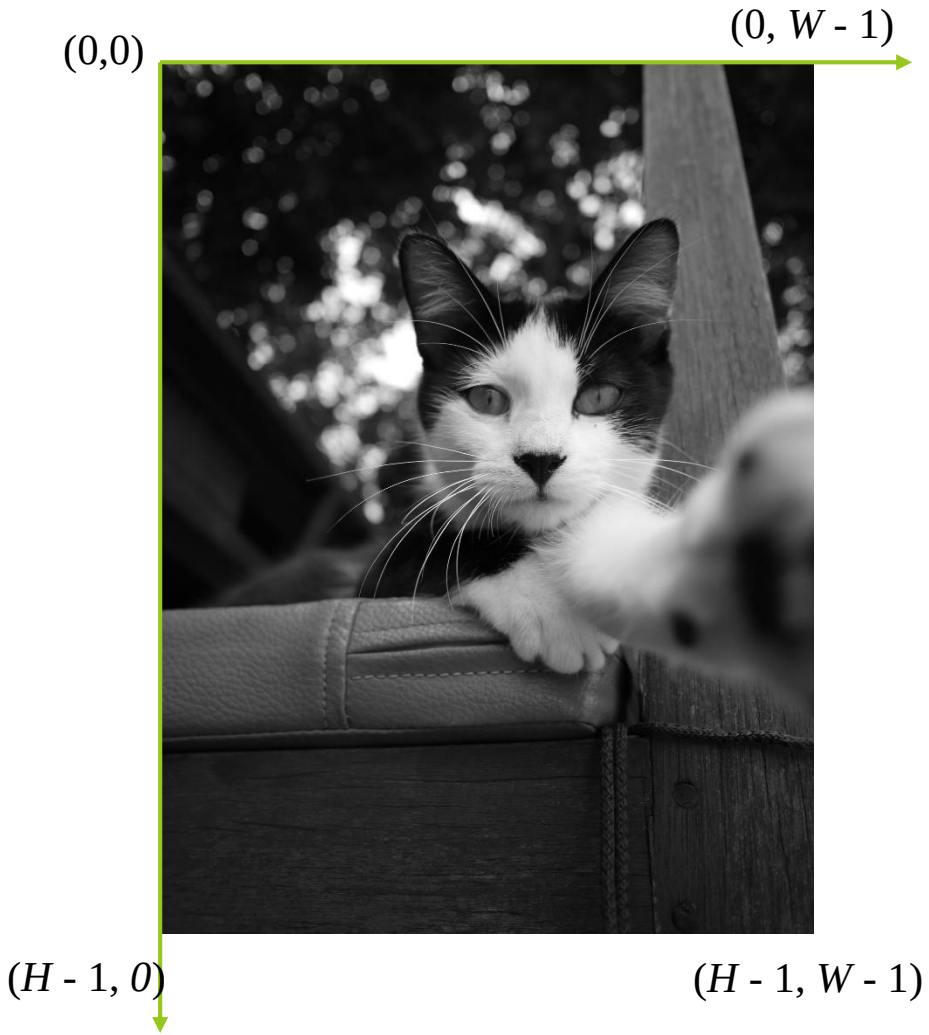
Digital images



Digital images

- A **digital image** is a discrete representation of data having both **spatial** and **intensity** information.
  - Spatial information – tell us the layout of image
    - Location of objects
  - Intensity information – tell us the brightness of each position in an image
    - Color of objects

Image representation



- Usually represented as a 2-D array of numbers $I(x,y)$
  - Let H and W denote the height and width (H rows and W columns)
  - Each element in the array is called a **pixel** (picture element)
    - Can be referred by two indices representing its location: (x,y)
    - The value of each pixel $I(x,y)$ represents the brightness or color
      - The response of image sensor at the corresponding position

25	27	26	22	31
25	23	32	29	230
26	25	31	227	225
37	232	236	226	229
237	243	235	236	236

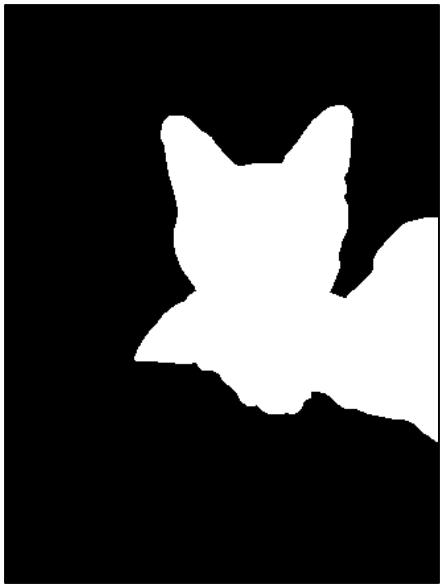
Grayscale images

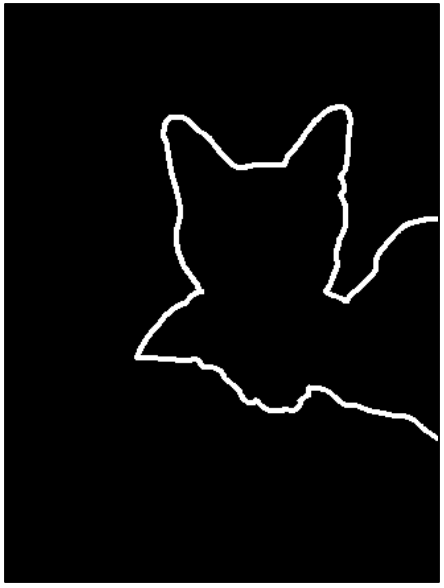
- A **grayscale image** is a 2-D array that assigns to each pixel in the array an integer value representing the intensity of that position in the image.
  - Also known as an **intensity image**
  - An 8-bit grayscale image assigns value in the range [0, 255] to each pixel.
    - **0 = black**
    - **255 = white**

0	0	0	0	0
0	0	0	0	255
0	0	0	255	255
0	255	255	255	255
255	255	255	255	255

Binary images

- A **binary image** is a special case of grayscale image in which the value of each pixel is chosen from the set {0, 1}.
  - Binary → two different intensity levels
- In practical, the set $\{0, 255\}$ is often used instead of $\{0, 1\}$.
  - If memory space is not concerned, as one pixel still requires 1 byte.

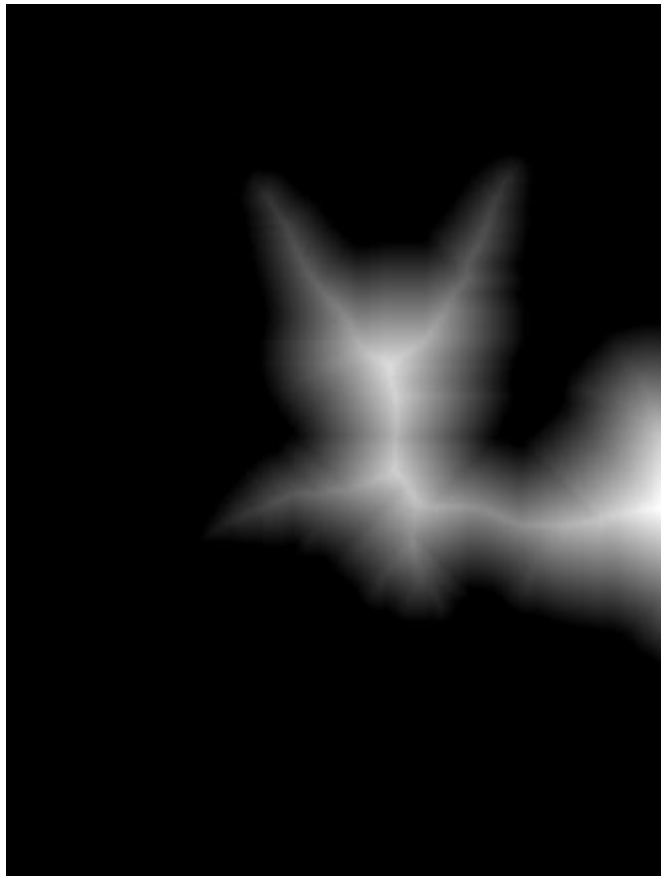




Binary images

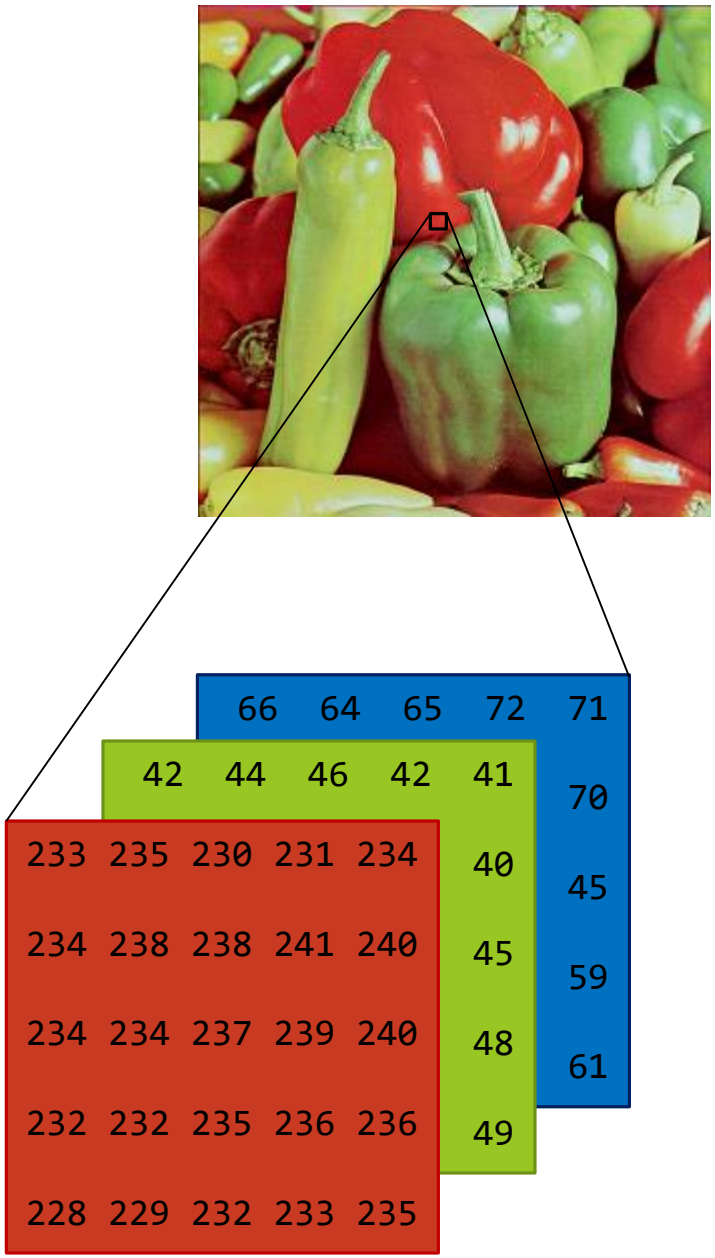
- A binary image can be used to represent a logical state:
  - 0/1, on/off, true/false
- Often used as an output of binary segmentation:
  - Represent foreground/background regions
- Or an output of edge detection:
  - Represent edges/non-edge pixels

Floating-point images



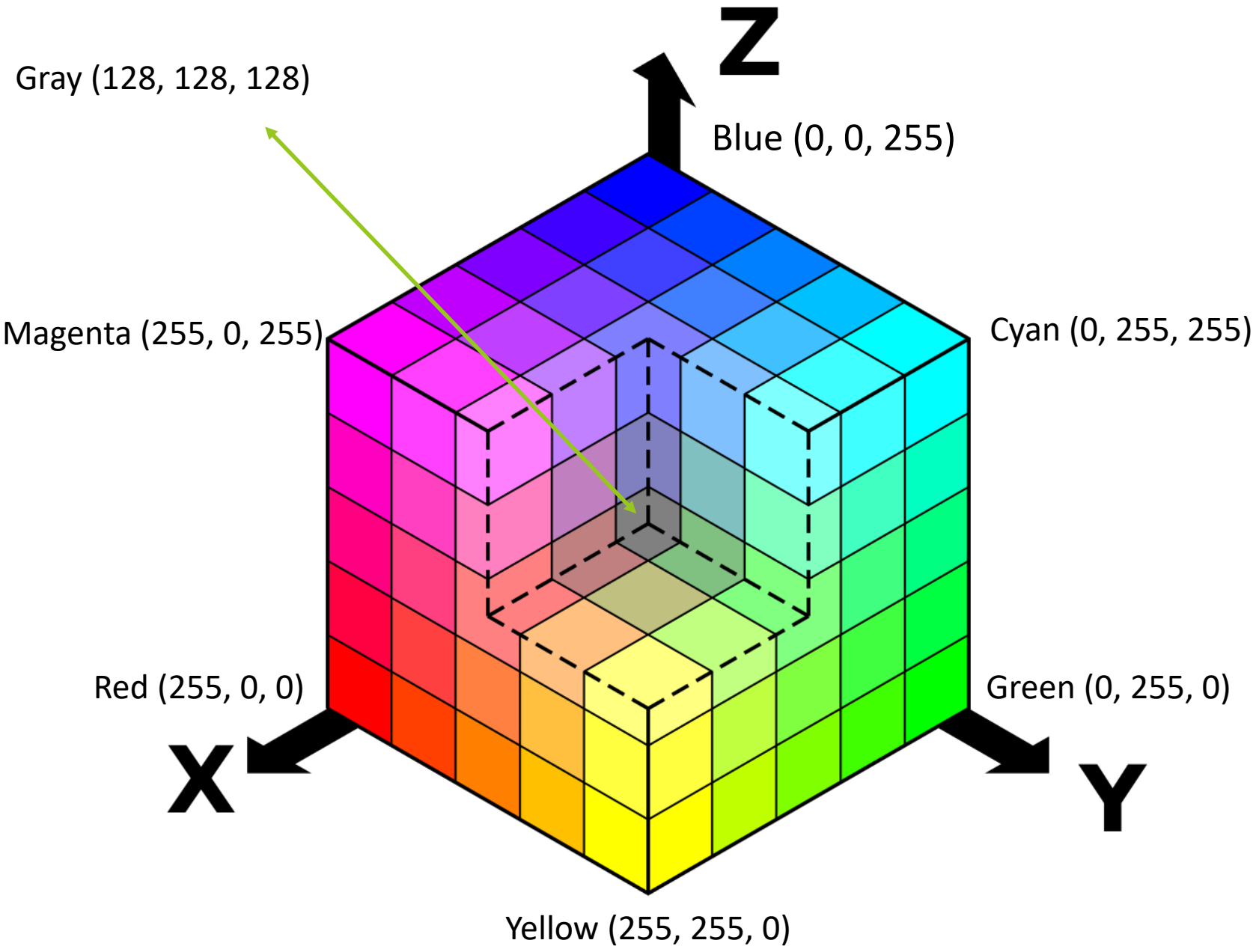
The intensity represents the distance from the boundary

- A **floating-point image** assigns a floating-point value to each pixel.
  - The assigned value can be positive, zero, or even negative, and has a fractional part.
  - It is usually used to store an intermediate result of a calculation.
    - Therefore, pixel value may representation something other than the intensity or color.

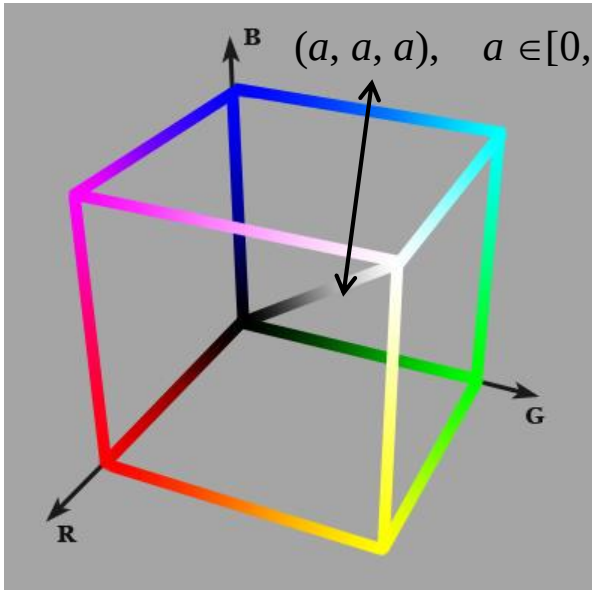


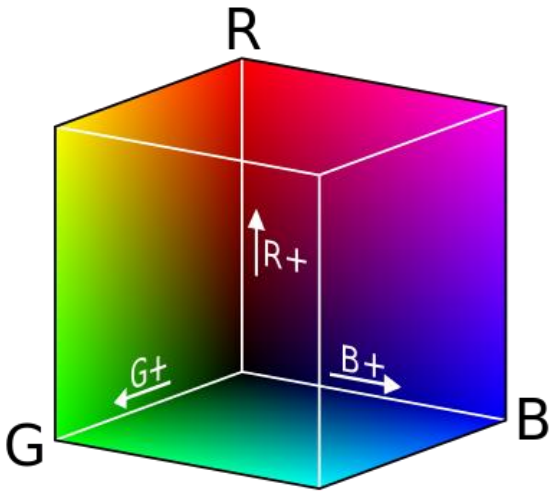
Color images

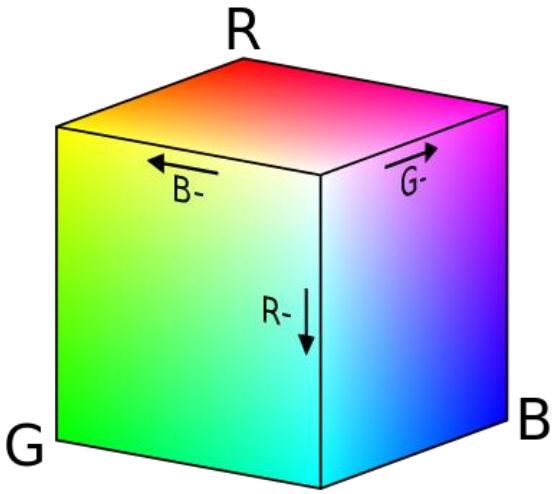
- A **color image** is usually represented as a 3-D array.
  - Each pixel (x,y) is assigned a vector of three elements defining its color.
    - In the **RGB color space**, the elements represent the **red**, **green**, and **blue** intensity of that location, respectively.
    - The value of each element is in the range [0, 255].
  - Can be conceptually considered as a set of three 2-D arrays (called **bands** or **channels**)

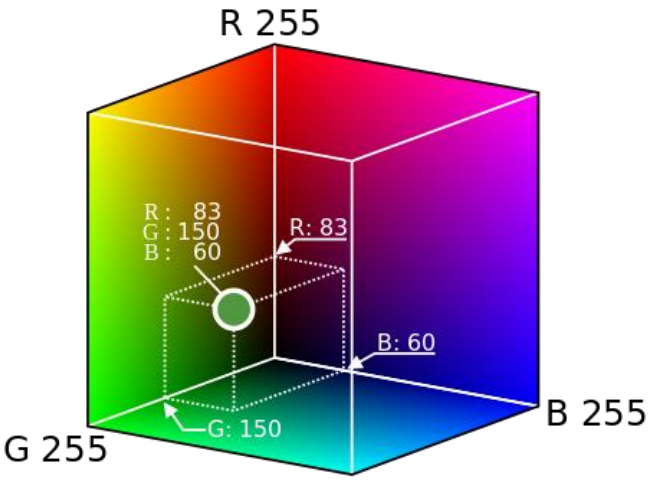


- Common misconception: a red object is the real world has only the red component!
- Fact: it is usually a mixed between three color components with red as the most dominant color.



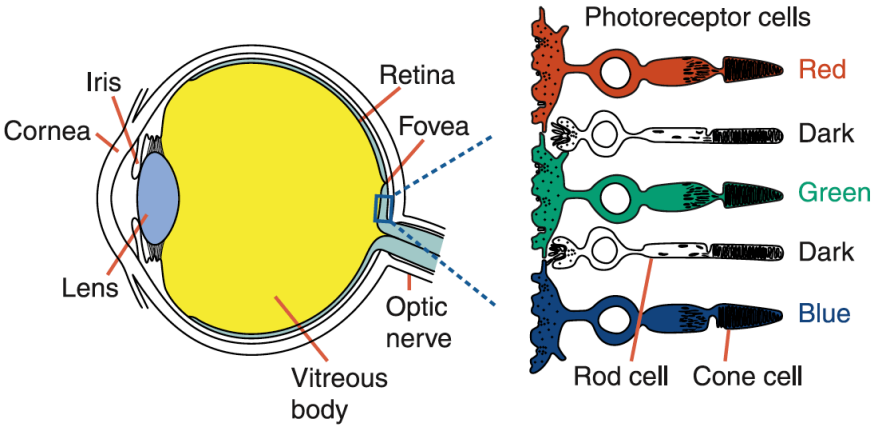






Human vision

- Photoreceptor cells
  - Rod cells:
    - Cannot sense color
    - Very sensitive to light
    - Allow us to see in less intense light
  - Cone cells:
    - Can sense colors
    - Three types: red, green, blue
    - Different sensitivity
    - Not active in less intense light



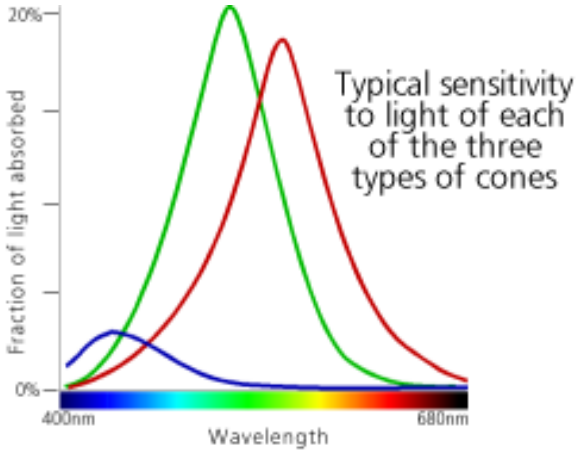


Image formats

- An **image format** defines how an array representing an image is stored into computer memory as an image file.

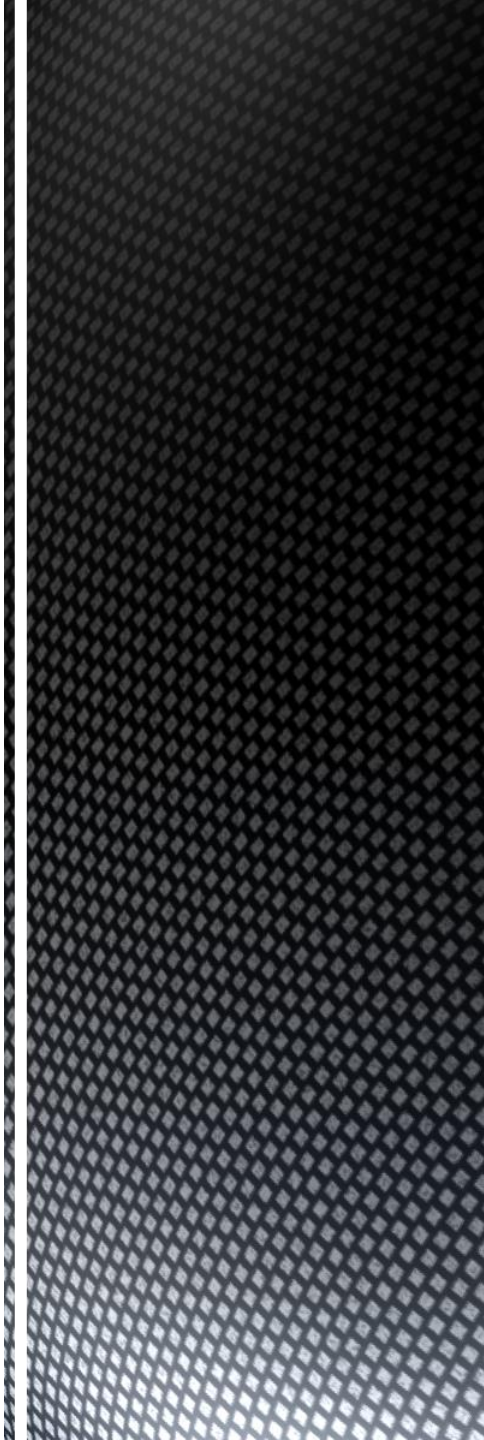
- An **image file** usually includes the following information:
  - Image properties, i.e., height, width, number of channels, etc.
  - Metadata, e.g., shutter speed, ISO number, imaging device
  - Sequence of image data (compressed or uncompressed)

- A **standard image format** is important when an image is shared to other computers.
  - It defines the way to correctly decode image data from an image file.

Standard image formats

Format	Extension	Compression	Description
Bit map picture	.bmp	No	Most basic file format
Portable network graphics	.png	Lossless	Newer image format; released 1996 Most widely used lossless format on the Internet Support 24-bit RGB and 32-bit RGBA
Tagged image file format	.tif, .tiff	No, lossy, lossless	Released 1986 Flexible, adaptable file format One of a popular format
Joint Photographic Expert Group	.jpg, .jpeg	Lossy	High compression rate Most widely used lossy format on the Internet
Graphics interchange format	.gif	Lossless	Support animation Limited to 256 colors (8 bits)

Images in OpenCV



Using OpenCV library

- C++:
  - Everything in OpenCV library is defined in a namespace called `cv`.
  - To access functions or classes you may use the `cv::` specifier
  - Or `using namespace cv;` directive
  - Include statements `"opencv2/module_name/module_name.hpp"`
    - `#include "opencv2/core/core.hpp"`
    - `#include "opencv2/highgui/highgui.hpp"`

- Python:
  - Installation: `pip install opencv-python`
  - Use import statement: `import cv2`
  - Begin with `cv2.` followed by the name of symbol/function

Class `Mat`

- The data structure used to store images in OpenCV is called `Mat`.
  - `Mat` can be used to store several types of images:
    - Grayscale
    - Floating-point
    - True-color (BGR)
  - Defined in the core module of OpenCV
- C++:
  - `#include "opencv2/core/core.hpp"`
  - `cv::Mat img;`
- Python:
  - Equivalent to class `numpy.ndarray`

imread()

- `imread()` is used to load image from a file to `Mat`.
  - C++: `Mat imread(const string& filename, int flags=1)`
  - Python: `cv2.imread(filename[,flags]) → retval`

  - Parameters:
    - **`filename`** – Name of an image file to be loaded
    - **`flags`** – Flags specifying the color type of a loaded image
      - `CV_LOAD_IMAGE_COLOR` - If set, always convert image to the color one (for python: `cv2.IMREAD_COLOR`)
      - `CV_LOAD_IMAGE_GRAYSCALE` - If set, always convert image to the grayscale one (for python: `cv2.IMREAD_GRAYSCALE`)
    - **`retval`** – Image object (`Mat`)
  - Defined in the highgui module

imwrite()

- `imwrite()` is used to save image stored in `Mat` to a file.
  - C++: `bool imwrite(const string& filename, Mat& img)`
  - Python: `cv2.imwrite(filename, img) → retval`
  - Parameters:
    - **`filename`** – Name of the image file
    - **`img`** – Image to be saved
  - Defined in the highgui module

Python example: `imread()` and `imwrite()`

- Load and save a color image

```
import cv2

img = cv2.imread("Sunset.jpg", cv2.IMREAD_COLOR)
cv2.imwrite("output.bmp", img)
```

- Load and save a gray-scale image

```
import cv2

img = cv2.imread("Sunset.jpg", cv2.IMREAD_GRAYSCALE)
cv2.imwrite("output.bmp", img)
```

C++ example: imread() and imwrite()

- Load and save a color image

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"

int main() {
   cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_COLOR);
   cv::imwrite("output.bmp", img);
}
```

- Load and save a gray-scale image

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"

int main() {
   cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_GRAYSCALE);
   cv::imwrite("output.bmp", img);
}
```

imshow()

- `imshow()` is used to display an image in a specific window.
  - C++: `void imshow(const string& winname, const Mat& img)`
  - Python: `cv2.imshow(winname, img)` → `None`

  - Parameters:
    - **`winname`** – Name of the window used to display the image
    - **`img`** – Image to be displayed

  - Defined in the highgui module

waitKey()

- `waitKey()` makes the program wait for a pressed key.
  - C++: `int waitKey(int delay=0)`
  - Python: `waitKey([delay]) → retval`

  - Parameters:
    - **`delay`** – Delay time in milliseconds. The values of 0 or negative mean "forever" (terminate when a key is pressed).
    - **`retval`** – the code of key pressed

  - Defined in the highgui module

Python example: `imshow()` and `waitKey()`

- Load and display image for five seconds

```
import cv2

img = cv2.imread("Sunset.jpg", cv2.IMREAD_COLOR)
cv2.imshow("Input image", img)
cv2.waitKey(5000)
cv2.destroyAllWindows()
```

- Load and display image until the user pressed a key

```
import cv2

img = cv2.imread("Sunset.jpg", cv2.IMREAD_COLOR)
cv2.imshow("Input image", img)
cv2.waitKey()
cv2.destroyAllWindows()
```

C++ example: `imshow()` and `waitKey()`

- Load and display image for five seconds

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp“
int main() {
   cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_COLOR);
   cv::imshow("Input image", img);
   cv::waitKey(5000);
}
```

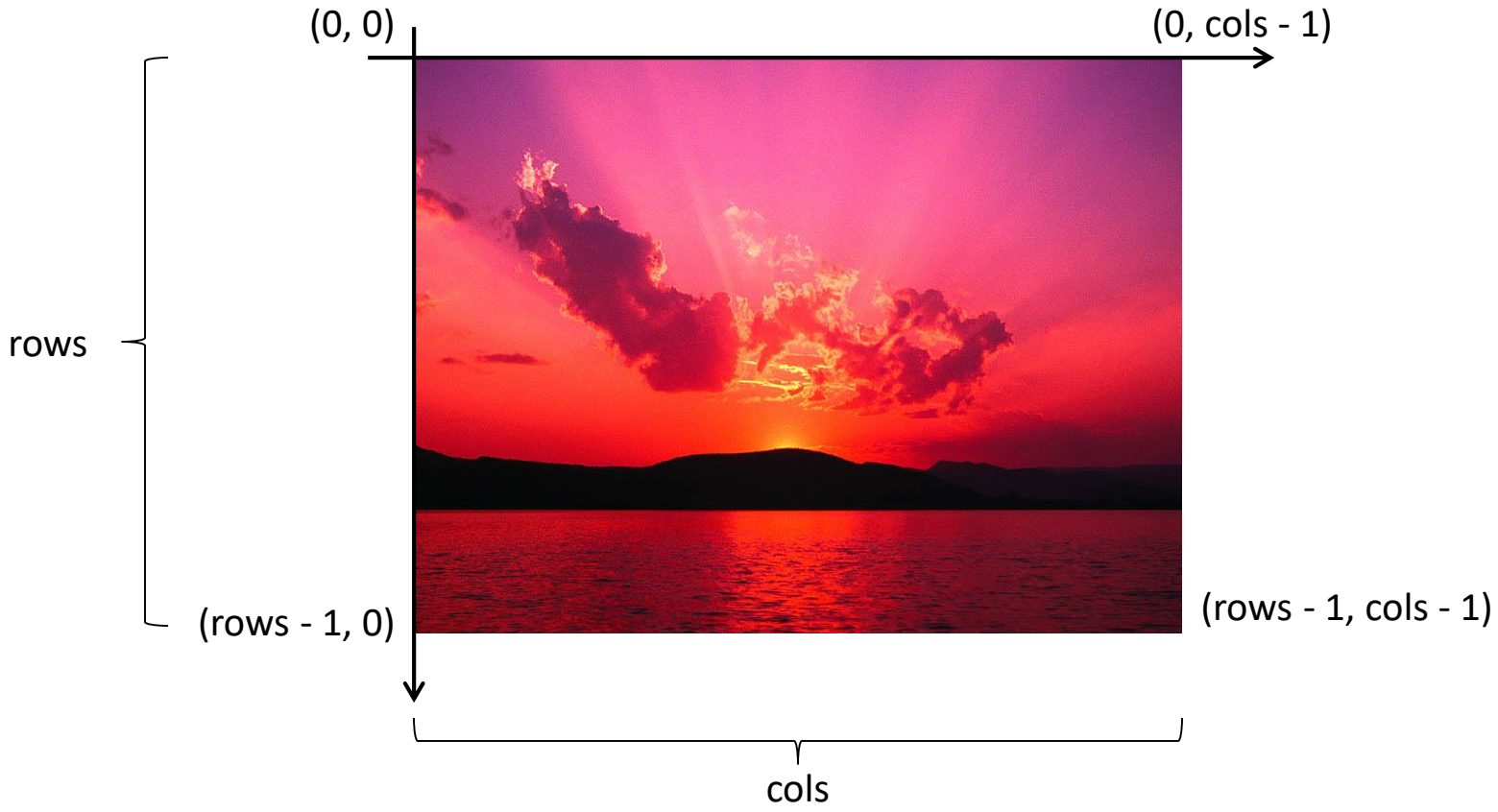
- Load and display image until the user pressed a key

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
int main() {
   cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_COLOR);
   cv::imshow("Input image", img);
   cv::waitKey();
}
```

Class Mat
rows and cols

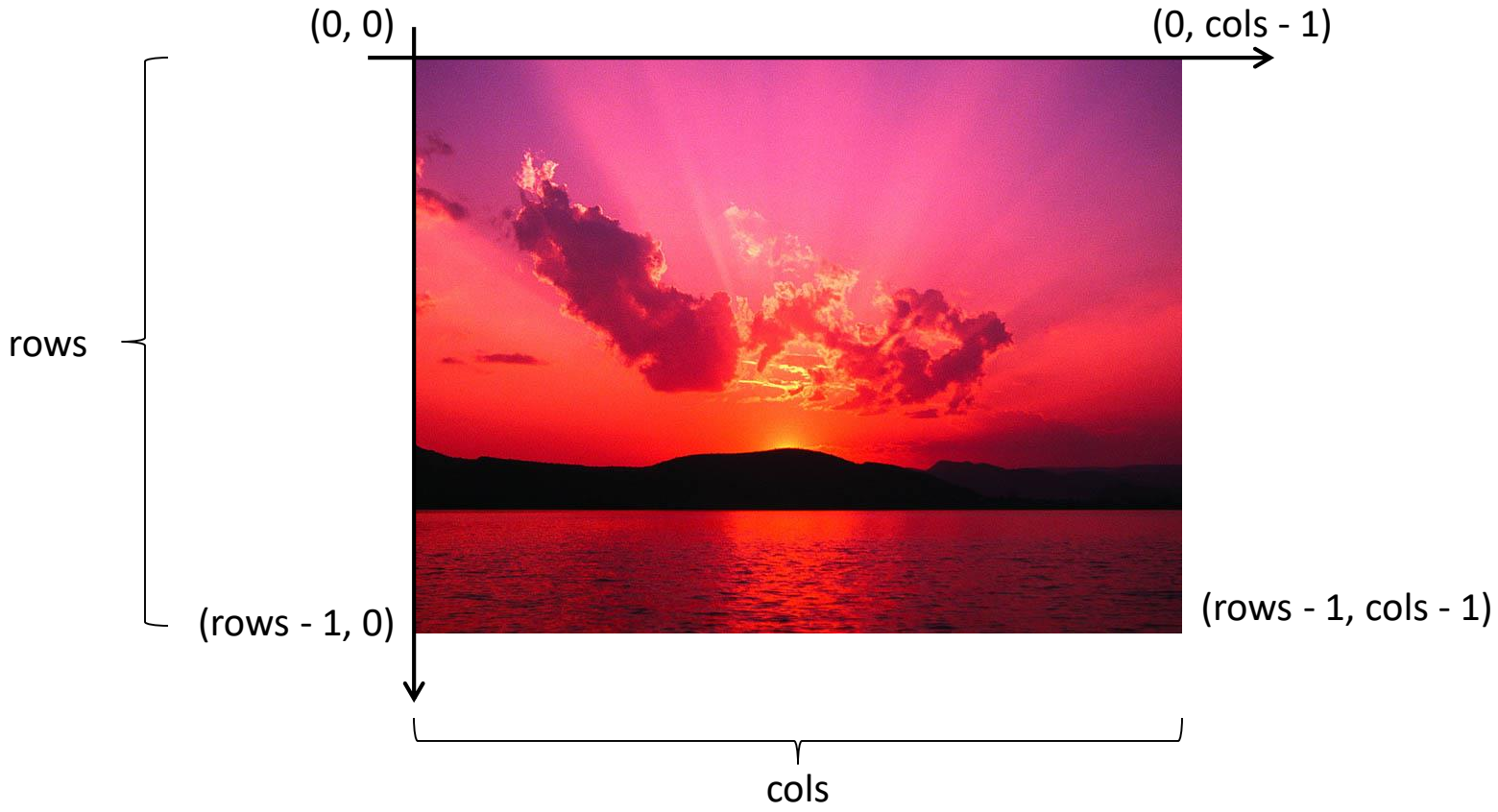
- In python, the attribute `shape` returns a tuple specifying the size of image: `(rows,cols)` or `(rows,cols,channels)`.

```
rows, cols, *chs = img.shape
```



Class `Mat` `rows` and `cols`

- In C++, `Mat` class has two attributes `rows` and `cols` that define the size (spatial resolution) of image.



Python example: Accessing pixels in grayscale image

- Reduce the intensity of each pixel

```
img = cv2.imread("Sunset.jpg", cv2.IMREAD_GRAYSCALE)
img[:, :] //= 2                    #integer division by 2
cv2.imshow("Processed image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Reduce the intensity of some pixels

```
img = cv2.imread("Sunset.jpg", cv2.IMREAD_GRAYSCALE)
img[0:300, 0:600] //= 2            #integer division by 2
cv2.imshow("Processed image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

C++ example: Accessing pixels in grayscale image

- Reduce the intensity of each pixel

```
cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_GRAYSCALE);
for(int i = 0; i < img.rows; ++i)
  for(int j = 0; j < img.cols; ++j)
    img.at<uchar>(i,j) /= 2;
cv::imshow("Processed image", img);
cv::waitKey();
```

- Reduce the intensity of some pixels

```
cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_GRAYSCALE);
for(int i = 0; i < 300; ++i)
  for(int j = 0; j < 600; ++j)
    img.at<uchar>(i,j) /= 2;
cv::imshow("Processed image", img);
cv::waitKey();
```

Python example: Accessing pixels in color image

- In python, the type of color pixel is `numpy.ndarray` containing three elements: B, G, and R.

```
import cv2

img = cv2.imread("Sunset.jpg", cv2.IMREAD_COLOR)
rows, cols, *chs = img.shape
img[0:rows//2, 0:cols//2] = [0, 0, 255]
cv2.imshow("Processed image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

C++ example: Accessing pixels in color image

- OpenCV uses BGR format for color images
  - Channel 0 – blue
  - Channel 1 – green
  - Channel 2 – red
- The type of color pixel is `cv::Vec3b`
  - A `vector` containing three elements

```
cv::Mat img = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_COLOR);
for(int i = 0; i < img.rows / 2; ++i)
  for(int j = 0; j < img.cols / 2; ++j) {
    img.at<cv::Vec3b>(i,j)[0] = 0;
    img.at<cv::Vec3b>(i,j)[1] = 0;
    img.at<cv::Vec3b>(i,j)[2] = 255;
    //img.at<cv::Vec3b>(i,j) = cv::Vec3b(0, 0, 255);
  }
cv::imshow("Processed image", img);
cv::waitKey();
```

Assignment 1

- Write a program to convert the upper image to the lower image
- The image is divided into four parts:
  - Top-left: the original color
  - Top-right: no red and green components
  - Bottom-left: no red and blue components
  - Bottom-right: no green and blue components

Assignment 1: Solution in Python

Assignment 1: Solution in C++

cvtColor()

- OpenCV provides a function `cvtColor()` used to convert an image of a color space into another color space.
  - C++: `void cvtColor(const Mat& src, Mat& dst, int code, int dstCn=0)`
  - Python: `cv2.cvtColor(src, code[, dst[, dstCn]]) → dst`

  - Parameters:
    - **src** – source image (input)
    - **dst** – destination image (output)
    - **code** – color space conversion code (e.g., `CV_BGR2GRAY`).
    - **dstCn** – number of channels in the destination image
      - if the parameter is 0 or is ignored, the number of the channels is derived automatically from `src` and `code` .

  - Defined in the imgproc module

Conversion code for `cvtColor()`

- The conversion code is a symbolic name written in the form:
  - C++: `CV_XXX2YYY`
  - Python: `cv2.COLOR_XXX2YYY`
  - where `XXX` is the original color space, and `YYY` is the desired color space
  - The color space may be one of the followings:
    - `BGR`, `RGB`, `GRAY`, `HSV`, `HLS`, `XYZ`, `Lab`, `Luv`, `YCrCb`
- Examples of conversion code:
  - `CV_BGR2RGB` – convert a BGR image into a RGB image
  - `CV_GRAY2BGR` – convert a gray-scale image into a BGR image
  - `CV_BGR2HSV` – convert a BGR image into a HSV image

Python example: cvtColor()

- Convert a BGR image to a gray-scale image

```
import cv2

colorImg = cv2.imread("Sunset.jpg", cv2.IMREAD_COLOR)

grayImg = cv2.cvtColor(colorImg, cv2.COLOR_BGR2GRAY)

cv2.imshow("Processed image", grayImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

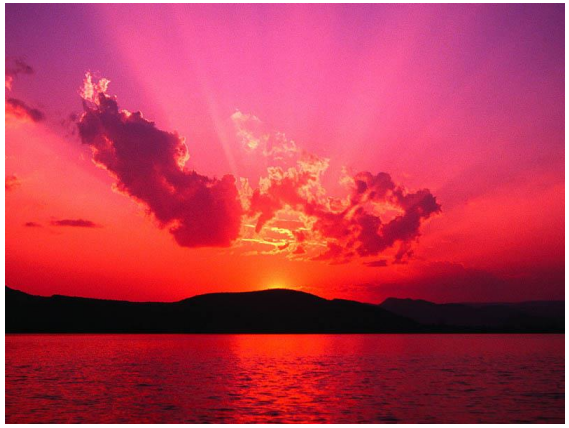
C++ example: cvtColor()

- Convert a BGR image to a gray-scale image

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

int main() {
   cv::Mat colorImg = cv::imread("Sunset.jpg", CV_LOAD_IMAGE_COLOR);
   cv::Mat grayImg;
   cv::cvtColor(colorImg, grayImg, CV_BGR2GRAY);
   cv::imshow("Converted image", grayImg);
   cv::waitKey();
}
```

RGB to grayscale conversion



- A color image can be transformed into a grayscale using the formula:

$$I(x, y) = 0.299 \times R(x, y) + 0.587 \times G(x, y) + 0.114 \times B(x, y)$$

  - Can be written in a matrix form:

$$I = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  - A grayscale image still possesses important features, e.g., edges, shapes, textures, etc.
    - Easier and faster to process

Assignment 2

- Write your own function that converts an BGR image into a gray-scale image

- Using the following formula:

$$I = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Assignment 2: Solution in Python

Assignment 2: Solution in C++

Assignment 3
Quantization

32 gray levels

16 gray levels

8 gray levels

4 gray levels

J.C. Russ, The Image Processing Handbook, 6th Edition, CRC Press, 2011

Assignment 3: Solution in Python

Assignment 3: Solution in C++

Assignment 3: Solution in C++

References

References

- M. Sonka, V. Hlavac, R. Boyle, Image Processing, Analysis, and Machine Vision (4th Edition), Cengage, 2014.
- J.C. Russ, The Image Processing Handbook (6th ed.), CRC Press, 2011.
- C. Solomon and T. Breckon, Fundamentals of Digital Image Processing: A Practical Approach With Examples in MATLAB, Wiley-Blackwell, 2011.
- A. McAndrew, An Introduction to Digital Image Processing with MATLAB, Course Technology, 2004.
- R. Laganière, OpenCV 2: Computer Vision Application Programming Cookbook, PACKT Publishing, 2011.
- www.opencv.org