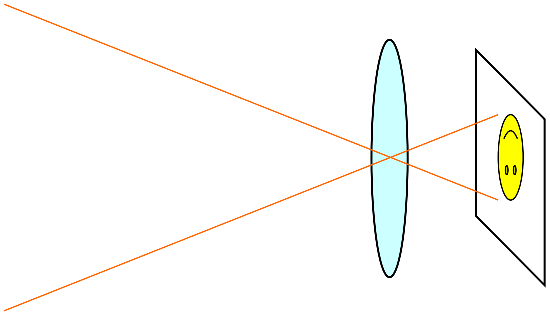
Intelligent Control Systems

Image Processing (1)
— Basic Concepts and Introduction of OpenCV —

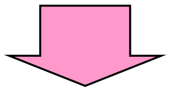
Shingo Kagami
Graduate School of Information Sciences,
Tohoku University
swk(at)ic.is.tohoku.ac.jp

http://www.ic.is.tohoku.ac.jp/ja/swk/

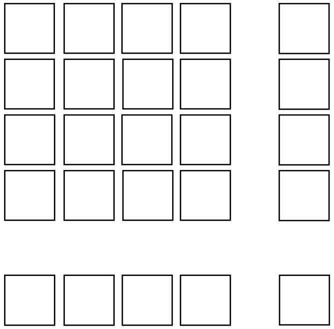
Digital Images



Analog distribution of light intensity



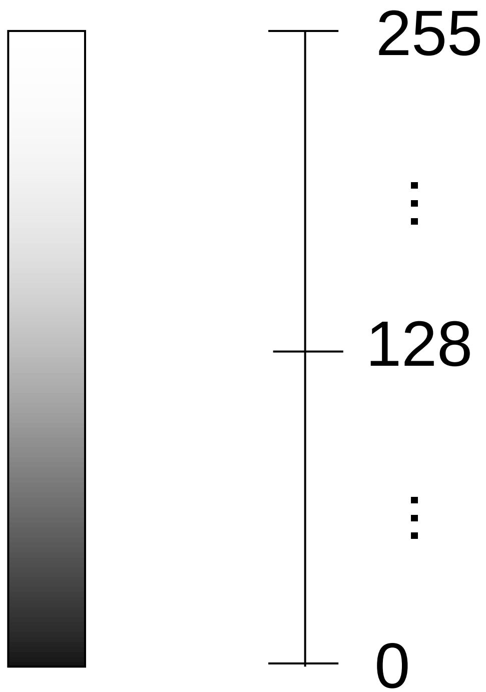
2-D discretization (into pixels)
quantization of intensity (ADC)



A digital image:
2-D array of pixel values

Pixel Value

(analog) light intensity; illuminance; voltage
(digital) pixel value; intensity value; gray level; grayscale value



quantized into [0, 255] integer:
8-bit graylevel image

cf.:

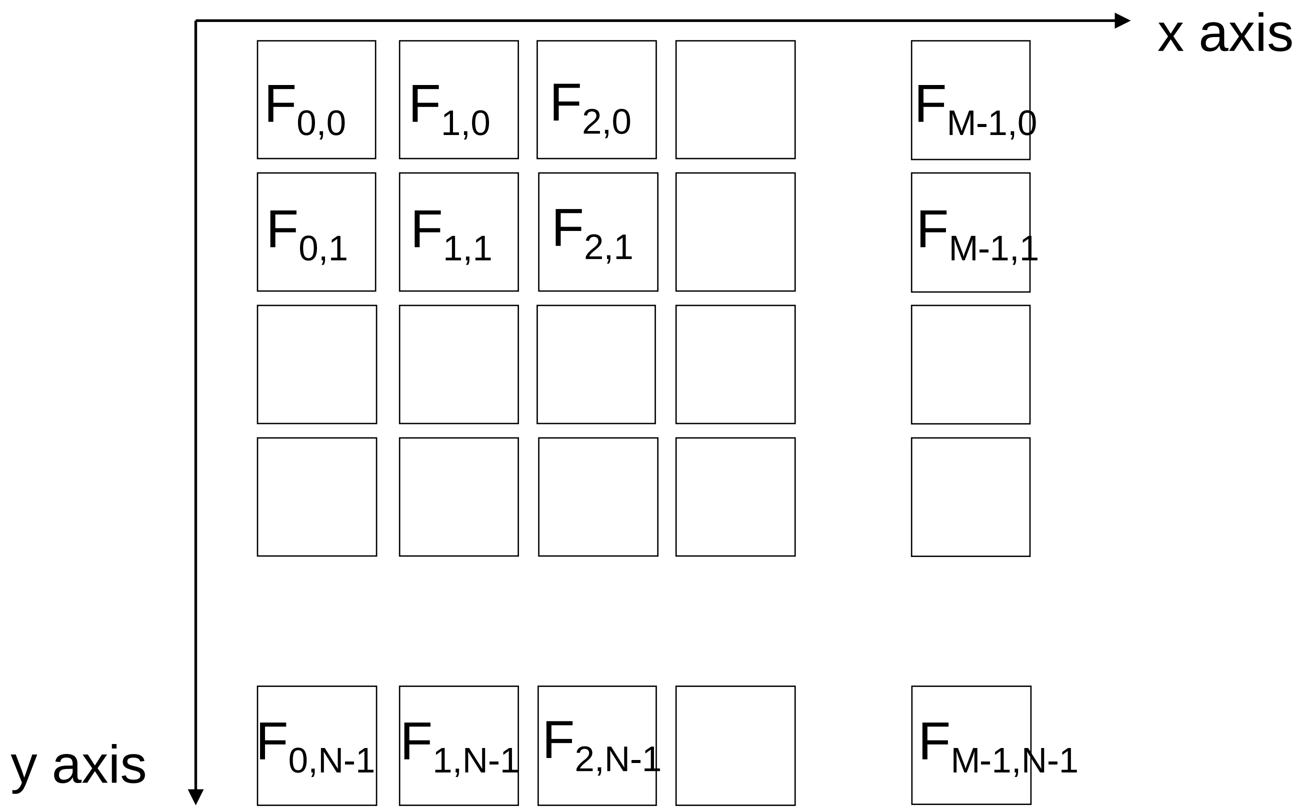
- binary image (= 1-bit graylevel)
- color image (= 3-channel image)

Expression of a Digital Image

M $\times$ N pixels digital image:

$\{ F_{x,y} \}$, $x = 0, 1, \cdots, M-1$, $y = 0, 1, \cdots, N-1$

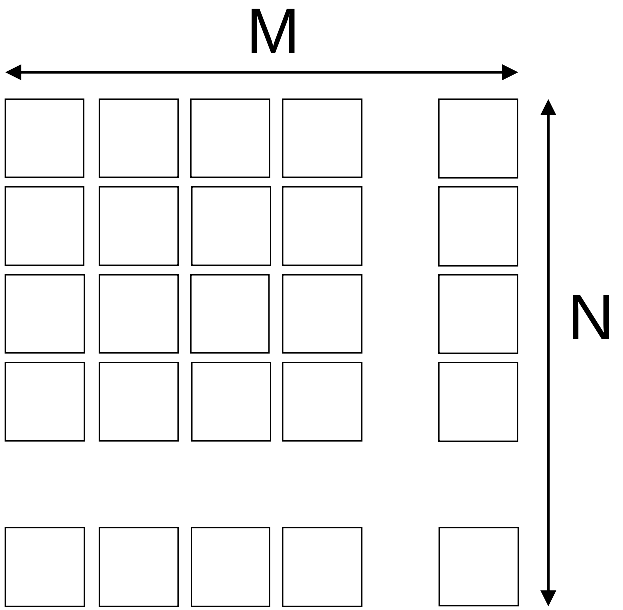
Pixel value at (x, y): $F_{x,y}$



Example in C

```
#define M 640
#define N 480
unsigned char image[M * N];

image[M * y + x] = 30;
// F(x, y) := 30
```

8-bit



- 2-D array is not convenient in C (e.g. not flexible in sizes)
- 1-D array is often preferred

A Simple Example Code in C

binarization (or thresholding)

```c
#define M 640
#define N 480
#define THRESHOLD 128

unsigned char image[M * N];
int i, j;

for (j = 0; j < N; j++) {
    for (i = 0; i < M; i++) {
        if (image[M * j + i] >= THRESHOLD) {
            image[M * j + i] = 255;
        } else {
            image[M * j + i] = 0;
        }
    }
}
```

Image Processing Libraries

- Writing image processing programs by your own in this way is possible, but cumbersome
  - How do we read image/video from a file or a camera?
  - How do we display images?
  - Do we have to implement every low-level image processing by ourselves?

There are many libraries or toolkits convenient for image processing, and using some of them is a good choice

In this lecture, we use OpenCV for Python

note: our goal is not to show you detailed features of OpenCV, but to lecture basic concepts of image processing by using OpenCV as a tool

Setup

- See the separate file: kagami_ic2022_install.pdf

- Open the WinPython Command Prompt (or anything else you set up for this course) and change the working directory to where the sample codes are.

  - Make sure you have ic_utils.py (uploaded on July 6) in your sample directory
  - When you want to edit a sample file, it is recommended to make a copy of the file with another name (e.g., ic01_thresh.py → ic01_thresh2.py) or use Git to create your own test branch

Example of Interactive Execution

Once you finished the setup, run ipython command (or use IPython window of spyder) and type the followings:

```
cd C:/ic2022/sample/
import cv2
img = cv2.imread('lena.jpg')
cv2.imshow('test_win', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

import matplotlib.pyplot as plt
plt.imshow(img)  # Oops!
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
quit()
```

Hit any key on the image window to proceed

OpenCV uses (Blue, Green, Red) order to encode pixel colors, while matplotlib uses (Red, Green, Blue)

Full Python Code for Thresholding (1/2)

ic01_thresh.py:

```
import cv2
import numpy as np


def threshold_impl(src, thresh, maxval):
    width = src.shape[1]
    height = src.shape[0]
    dest = np.zeros_like(src)
    for j in range(height):
        for i in range(width):
            if src[j, i] > thresh:
                dest[j, i] = maxval
            else:
                dest[j, i] = 0
    return dest
```

OpenCV module

NumPy module

definition of a function

zero-initialized image with the same size and data type as `src`

`range(n)` generates a list `[0, 1, 2, …, n-1]`

Indices for pixel access are given in [Y-axis, X-axis] order (= [row, column] order)

Note that indentation matters

Full Python Code for Thresholding (2/2)

```
def main():
    frame = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)

    th = 128
    thresh_my = threshold_impl(frame, th, 255)
    ret, thresh_cv = cv2.threshold(frame, th, 255, cv2.THRESH_BINARY)
    thresh_np = np.where(frame > th, np.uint8(255), np.uint8(0))

    cv2.imshow('result', thresh_my)
    cv2.imshow('result2', thresh_cv)
    cv2.imshow('result3', thresh_np)

    cv2.waitKey(0)

    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

imread function in cv2 module is called

(window name, image)

Wait infinitely until any key is pressed, while refreshing the graphics

An idiom to start the program from “main” function

A function can return multiple values

List, Tuple, and Dictionary in Python

list

```
x = [0, 1, 'apple', [2, 4]]
x[2]
 -> 'apple'
x.append(999)
x
 -> [0, 1, 'apple', [2, 4], 999]
```

tuple

```
y = (0, 1, 'apple')
y[1]
 -> 1
y.append(999)  # error
y[1] = 123     # error
```

tuple is similar to list, but mutation is not allowed

dict

```
d = {'value': 123, 'pos': (10, 20), 'name': 'Foo'}
d['value']
  -> 123
d['name'] = 'Bar'
d['name']
  -> 'Bar'
```

NumPy n-dimensional array (ndarray)

- NumPy is a de factor standard library for scientific computing with Python
- NumPy's ndarray is used to represent images in OpenCV in Python
  (Note: different from OpenCV in C++, which uses cv::Mat class)

List in Python is flexible but inefficient

```
x = [0, 1, 'apple', [2, 4]]
```

NumPy ndarray is efficient because it is a straightforward array with fixed data type

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9]], dtype=np.uint8)
x
  -> array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

Initialization and conversion of ndarray

2-dimensional array:

```
x = np.array([[1,2,3], [4,5,6]])
x.shape
  -> (2, 3)
x.dtype
  -> dtype('int32')
```

(height, width): different from image processing convention; it's more like matrix convention

data type is automatically inferred if not specified

1-dimensional array:

```
x = np.array([1.0, 2.0, 3.0])
x.shape
  -> (3,)
x.dtype
  -> dtype('float64')
```

a tuple of length 1

data type conversion

unsigned 8-bit integer

```
x = np.array([[1,2,3], [4,5,6]], dtype=np.uint8)
x = np.float32(x)
x = x.astype(np.int16)
```

Video Processing Example: Thresholding

```
def main():
    cap = cv2.VideoCapture('vtest.avi')
    #cap = cv2.VideoCapture(0)

    while True:
        grabbed, frame = cap.read()
        if not grabbed:
            break
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        thresh_img = threshold_impl(frame, 128, 255)
        cv2.imshow('result', thresh_img)
        key = cv2.waitKey(30)
        if key == ord('q'):
            break

    cv2.destroyAllWindows()
```

images are captured from video file or a camera device (specified by integer index)

break if no image is available

wait for 30 ms (or key input)

character code of ‘q’

Switching Capture Source by Command-Line Argument

```
import sys
import cv2

...

    cap_src = 'vtest.avi'
    if len(sys.argv) == 2:
        if sys.argv[1].isdecimal():
            cap_src = int(sys.argv[1])
        else:
            cap_src = sys.argv[1]
    cap = cv2.VideoCapture(cap_src)
```

sys.argv[0] is command name
sys.argv[1] is 1st argument

Now you can run the program by executing:

```
python ic01_thresh_video.py
python ic01_thresh_video.py books.mp4
python ic01_thresh_video.py 0
```

vtest.avi is read by default
books.mp4 is read
your 0-th web camera is used

Just-In-Time Compilation by numba

We want to iterate through the pixels using nested `for` loops, but it is extremely slow in Python.

- Good practice is to avoid iteration by using numpy methods thoughtfully, but it is not main focus of this course
- We use numba module as a workaround: the function is compiled when it is executed for the first time and therefore runs fast for the second time and on
- Note however that some of numpy functionalities are not supported by numba

```python
from numba import jit

@jit
def threshold_impl(src, thresh, maxval):
    ...
```

Trackbars in OpenCV

```python
def do_nothing():
    pass

def main():
    cap = cv2.VideoCapture('vtest.avi')

    cv2.namedWindow('result')
    cv2.createTrackbar('thresh', 'result', 128, 255, do_nothing)

    while True:
        grabbed, frame = cap.read()
        if not grabbed:
            break
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        th = cv2.getTrackbarPos('thresh', 'result')
        thresh_img = threshold_impl(frame, th, 255)

        cv2.imshow('result', thresh_img)
        key = cv2.waitKey(30)
        if key == ord('q'):
            break
```

ranging from 0 to 255
initial value is set to 128

Defining Your Own Module

ic_utils.py

```
import cv2

def select_capture_source(argv):
    cap_src = 'vtest.avi'
    if len(argv) == 2:
        if argv[1].isdecimal():
            cap_src = int(argv[1])
        else:
            cap_src = argv[1]
    return cv2.VideoCapture(cap_src)
```

ic01_thresh_video.py

```
import sys
import ic_utils as ic

def main():
    cap = ic.select_capture_source(sys.argv)
```

Mouse Interaction in OpenCV (1/2)

```
def on_mouse_rect(event, x, y, flag, mstate):
    if event == cv2.EVENT_LBUTTONDOWN:
        mstate['selection'] = 'ongoing'
        mstate['xybegin'] = (x, y)
    elif event == cv2.EVENT_LBUTTONUP:
        mstate['selection'] = 'valid'
    elif event == cv2.EVENT_RBUTTONDOWN:
        mstate['selection'] = 'invalid'

    if mstate['selection'] == 'ongoing':
        mstate['xyend'] = (x, y)
```

```
def main():
    mstate = {
        'selection': 'invalid',
        'xybegin': (-1, -1),
        'xyend': (-1, -1),
    }
    cv2.setMouseCallback('result', on_mouse_rect, mstate)
```

function `on_mouse_rect` is called every frame a mouse event occurs with final argument `mstate`

Mouse Interaction in OpenCV (2/2)

ic01_thresh_mouse_rect.py

```
...
        img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        th = cv2.getTrackbarPos('thresh', 'result')
        xbegin, ybegin = mstate['xybegin']
        xend, yend = mstate['xyend']

        if mstate['selection'] == 'valid':
            roi = img[ybegin:yend, xbegin:xend]
            thresh_roi = threshold_impl(roi, th, 255)
            img[ybegin:yend, xbegin:xend] = thresh_roi

        elif mstate['selection'] == 'ongoing':
            cv2.rectangle(img, (xbegin, ybegin), (xend, yend),
                          color=0, thickness=2)
...
```

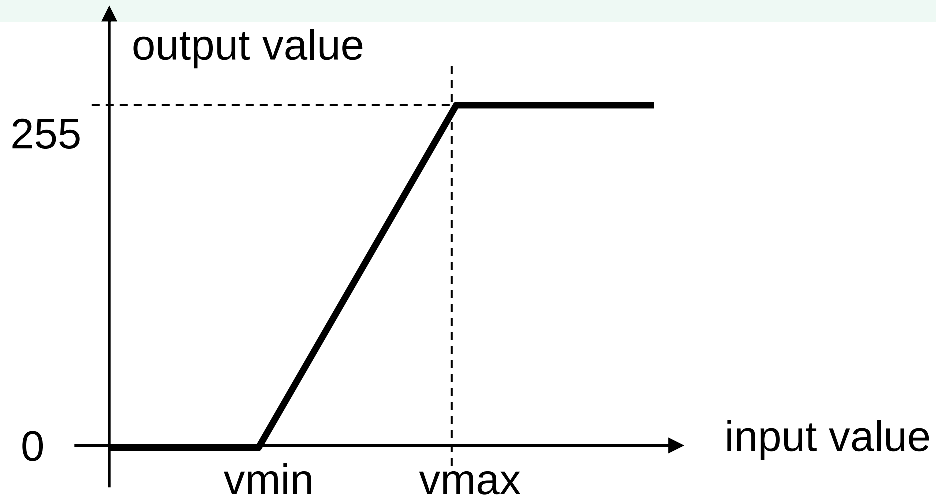
keyword arguments
(cf. positional arguments)

`img[ybegin:yend, xbegin:xend]`:
subregion of `img` whose top-left corner is at (xbegin, ybegin)
and bottom-right corner (exclusive) is at (xend, yend)

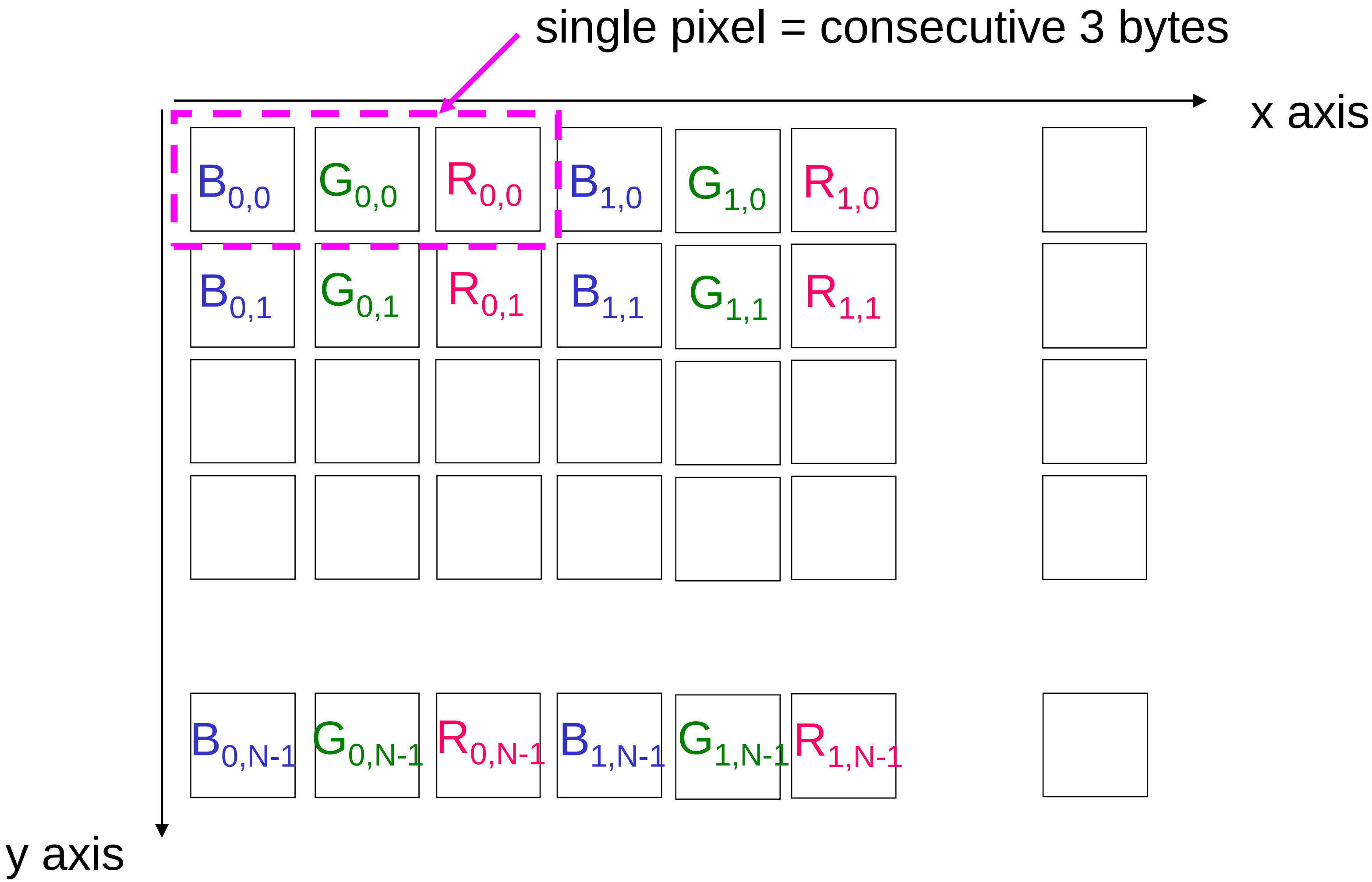
Converting Pixel Values

ic01_convert_graylevels.py:

```
def convert_graylevels(src, vmin, vmax):
    width = src.shape[1]
    height = src.shape[0]
    dest = np.zeros_like(src)
    for j in range(height):
        for i in range(width):
            val = (255 * (src[j, i] - vmin)) / (vmax - vmin)
            val = max(0, val)   ## clipping negative values
            val = min(val, 255) ## clipping values over 255
            dest[j, i] = val
    return dest
```



Color Image Representation



Representation in OpenCV for Python (NumPy)

(height, width)-shape array with 3 channels
(or equivalently, (height, width, 3)-shape tensor) is used

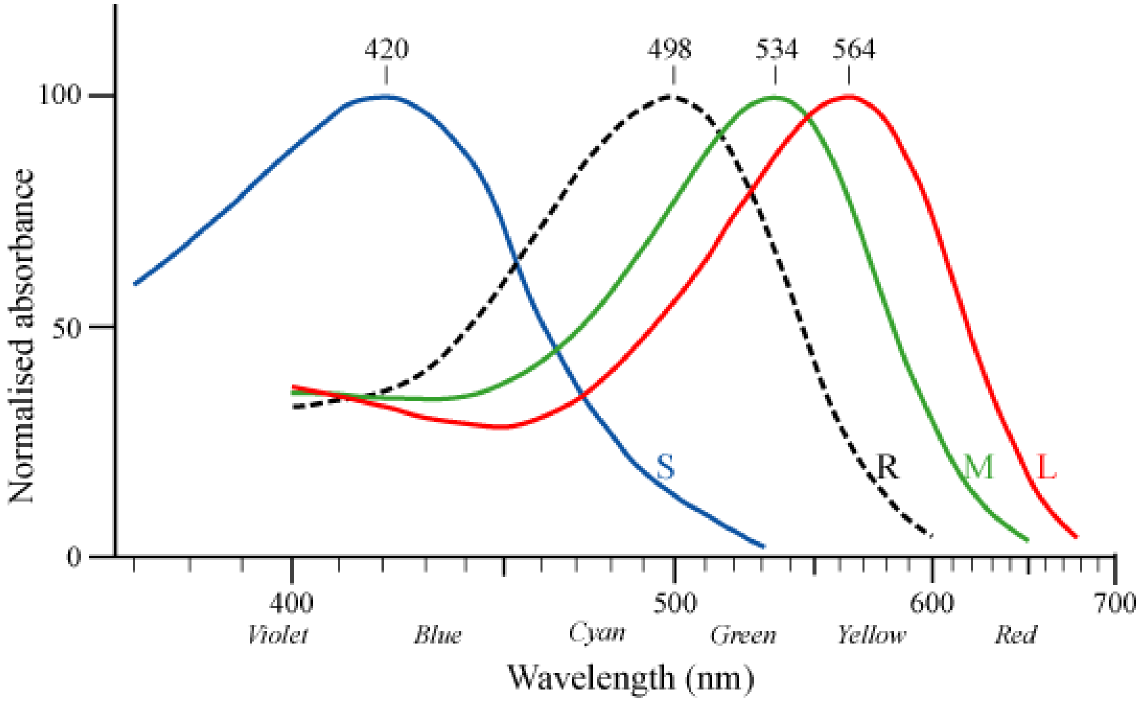
```
fruits = cv2.imread('fruits.jpg')
fruits.shape
  -> (480, 512, 3)

fruits[100, 100]
  -> array([ 52,  98, 116], dtype=uint8)
fruits[100, 100, 0]
  -> 52
```

```
np.array([[[B00, G00, R00], [B10, G10, R10], … ],
           [… ],
           …
           [… ]], dtype=np.uint8)
```

RGB Color Space

Why R, G, and B?

- Our eyes have three types of wavelength-sensitive cells (cone cells)
  - cf. rod cells
- So, the color space we *perceive* is three-dimensional



http://commons.wikimedia.org/wiki/File:Cone-response.png

Other Color Spaces

XYZ, L*a*b, L*u*v

defined by CIE (Commission Internationale de l'Eclairage)

YIQ, YUV, YCbCr

used in video standards (NTSC, PAL, …)

HSV (HSI, HSL)

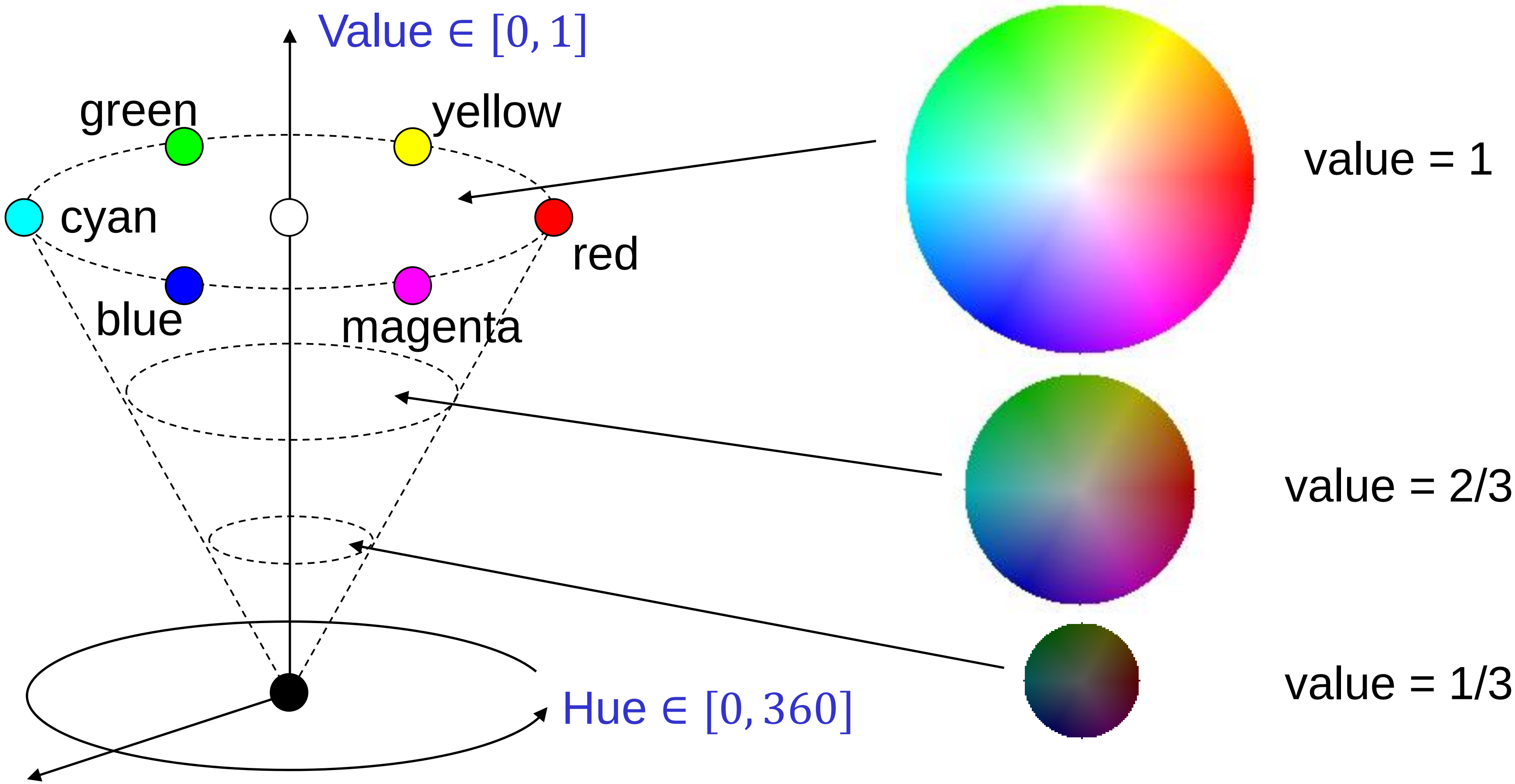
based on Munsell color system

cf. CMY, CMYK (for printing; subtractive color mixture)

```
img_out = cv2.cvtColor(img_in, cv2.COLOR_BGR2HSV)
```

HSV Color Space

ic01_hsv_illustrated.py

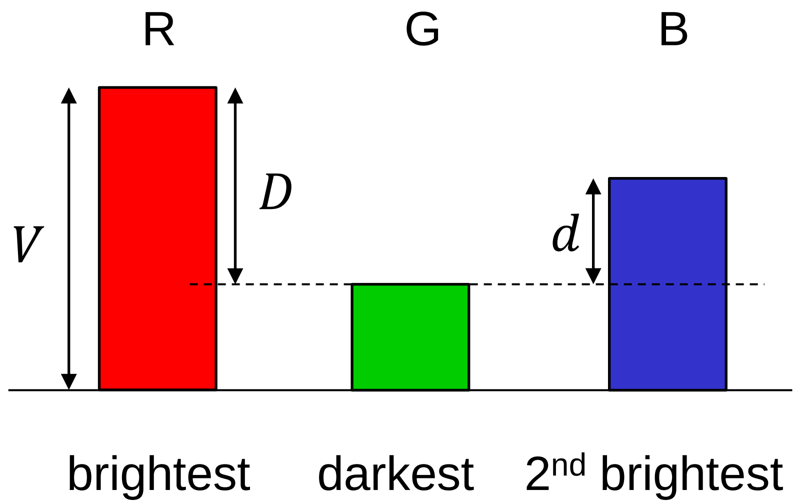


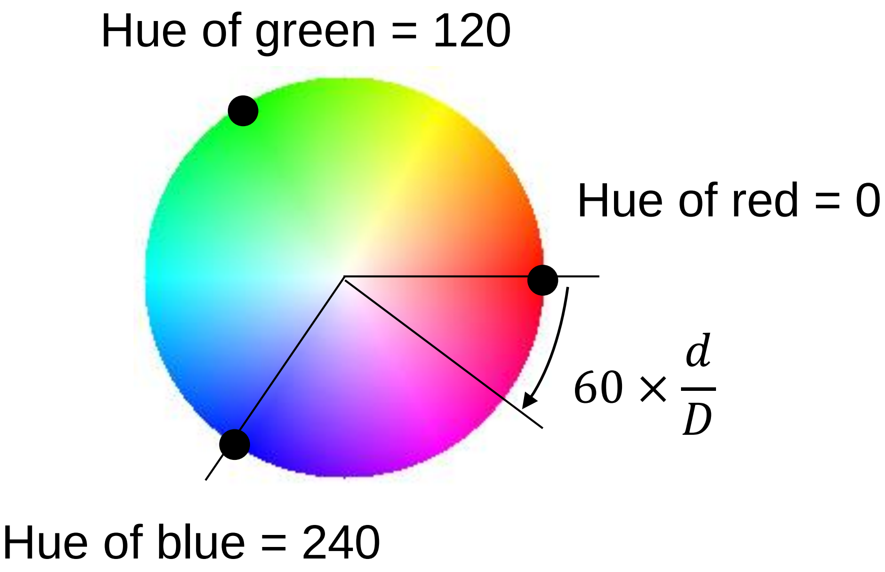
Saturation ∈ [0, 1] (If a color is made of only two of RGBs, it is called saturated)

Note: OpenCV (uint8) employs different ranges:

$0 \leq$ Hue ≤ 180, $0 \leq$ Saturation ≤ 255, $0 \leq$ Value ≤ 255

RGB to HSV

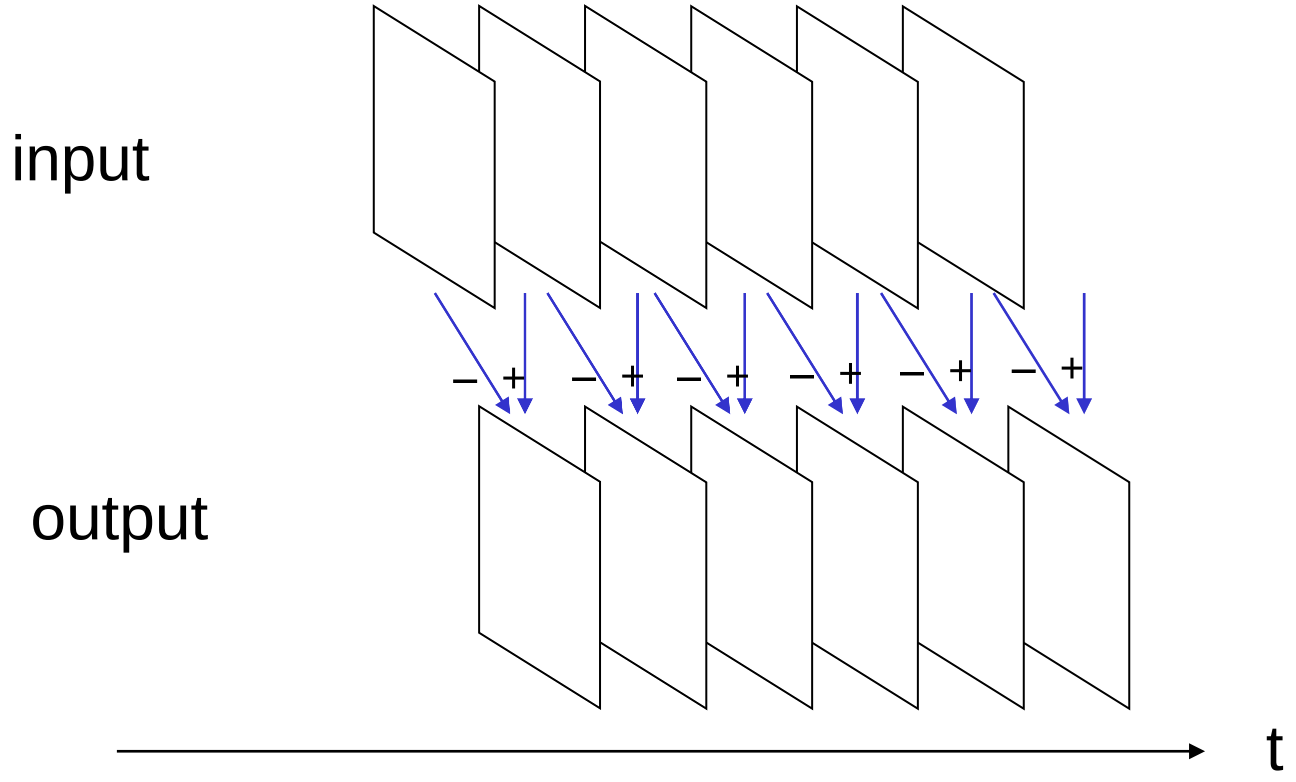




1. Determine V, D, and d
2. Set H to interpolate between the hues of the brightest and 2nd brightest
3. Set $S = \frac{D}{V}$

ic01_convert_color.py

Video Processing Example: Frame Difference



Frame Difference

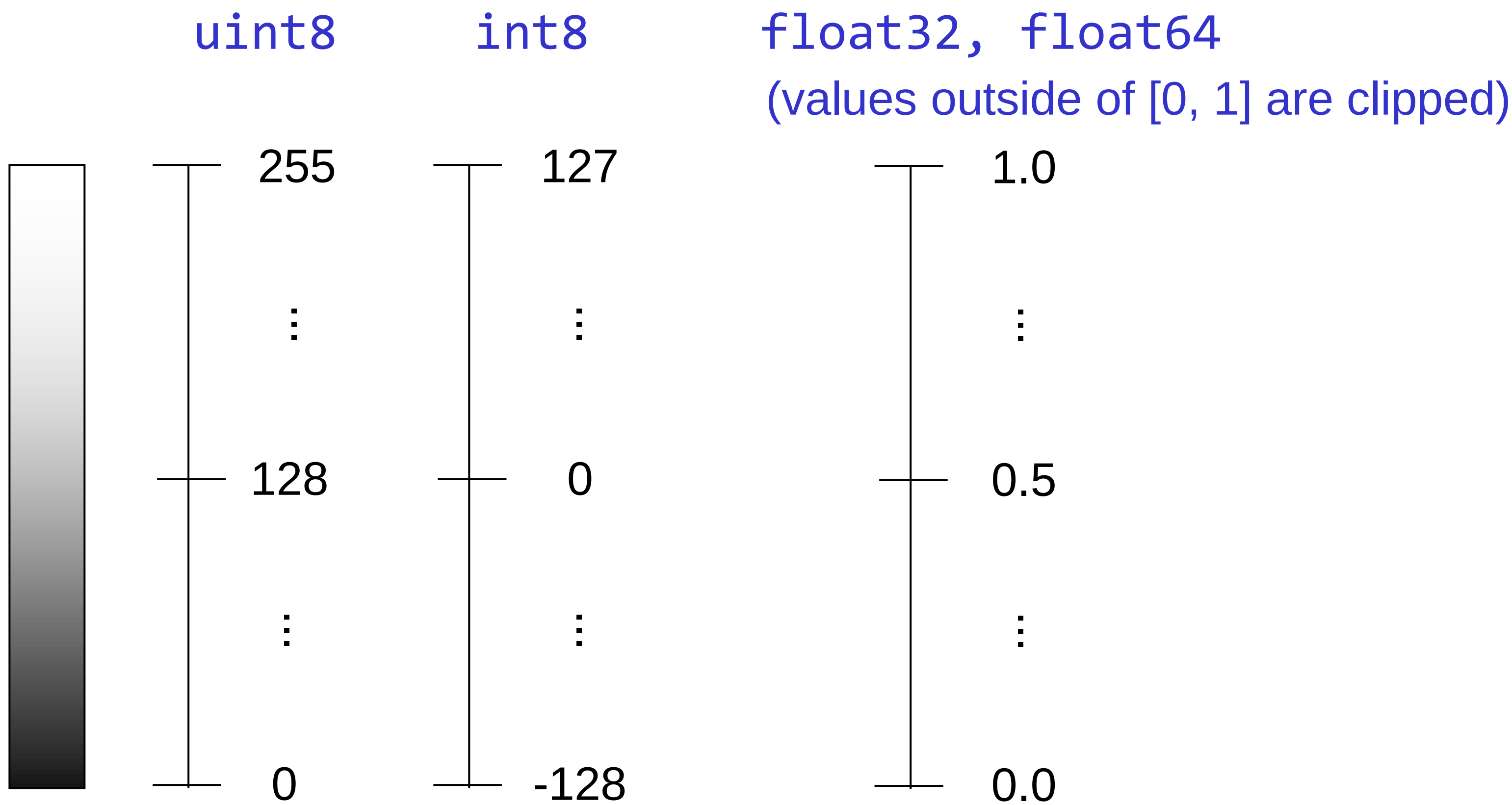
ic01_frame_diff.py:

```
def main():
    cap = ic.select_capture_source(sys.argv)
    grabbed, prev_frame = cap.read()
    prev_frame = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

    while True:
        grabbed, frame = cap.read()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        if not grabbed:
            break

        diff_img = np.int8((np.int16(frame) - np.int16(prev_frame)) / 2)
        prev_frame = frame

        ## cv2.imshow adds 128 to pixel values when it receives np.int8 image
        cv2.imshow('result', diff_img)

        key = cv2.waitKey(30)
        if key == ord('q'):
            break
```

converted to 16-bit signed integer image to deal with negative values, and then converted to 8-bit signed integer

cv2.imshow and data type

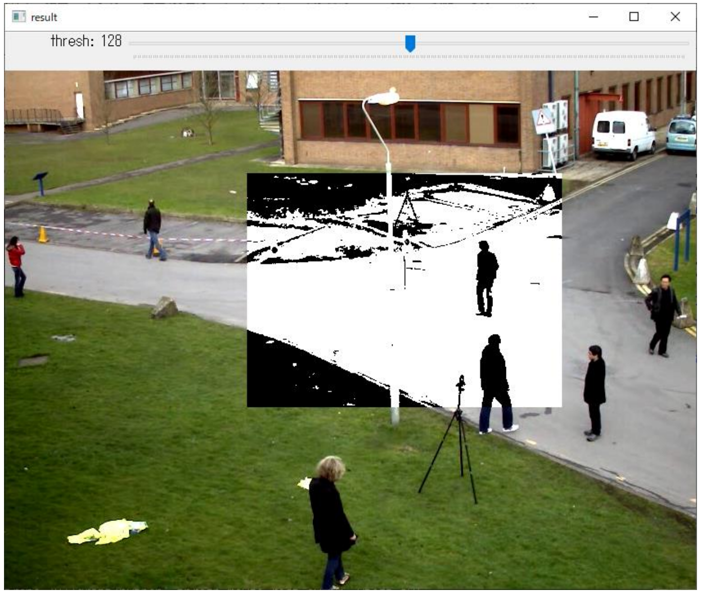
```
cv2.imshow('window name', img)
```



- Behaviors for other types (16 and 32-bit integers) are less intuitive
- These are rules for `cv2.imshow` only and do not generalize at all

Exercises (Not Assignments)

Copy ic01_thresh_mouse_rect.py to another name and modify it to draw the thresholding result on the color input image (instead of monochrome one).



Hints:

- When you copy an image to a subregion of another image, the number of channels of both images must agree
- Conversion from graylevel to RGB can be done by cv2.COLOR_GRAY2BGR

References

Sample codes of this course are available at

https://github.com/shingo-kagami/ic

Manuals of OpenCV and NumPy are found at:

- https://docs.opencv.org/master/
- https://docs.opencv.org/master/d6/d00/tutorial_py_root.html
- http://www.numpy.org/

Some Book resources:

- A. Kaehler, G. Bradski: Learning OpenCV 3, O'Reilly, 2017. (詳解 OpenCV 3, オライリー・ジャパン, 2018) : A book written by OpenCV contributors; Unfortunately, the codes are in C++
- J. Howse, J. Minichino: Learning OpenCV 4 Computer Vision with Python 3, Packt Publishing, 2020.
- A. F. Villan, Mastering OpenCV 4 with Python, Packt Publishing, 2019.