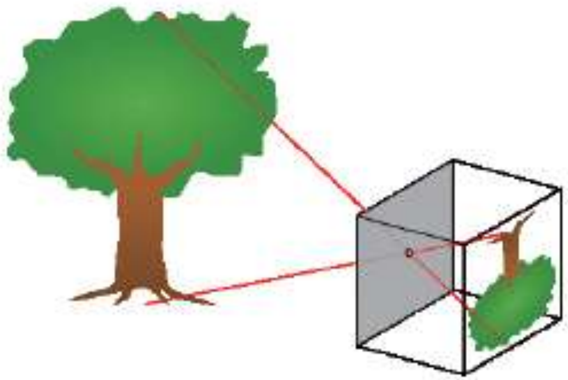
Practice:
Image Processing

COMPUTER VISION (COURSE-HY24011)

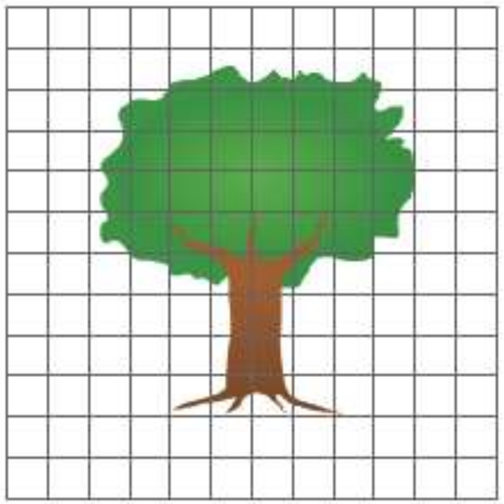
Q YOUN HONG

Digital Image

- Sampling and quantization
  - 2D image space is sampled by M x N (M x N: resolution)
  - Brightness (light intensity) is quantized as L levels ($L \in [0, L-1]$)



Pinhole camera model

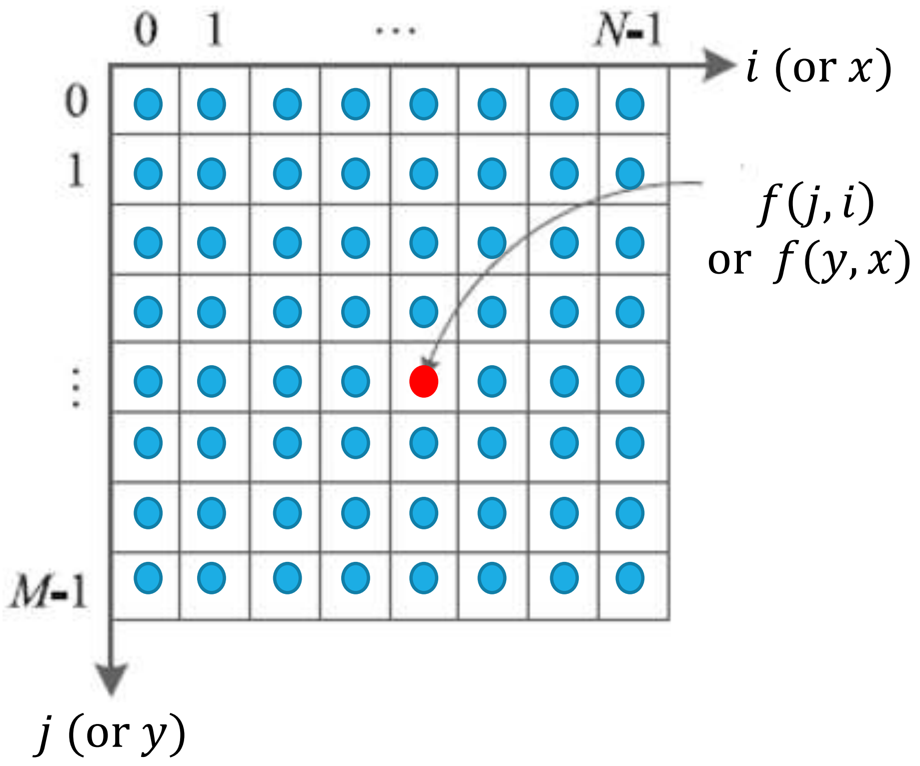


Sampling (M=N=12)

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	3	4	2	3	4	3	0	0	0
0	0	3	7	8	8	8	7	6	3	0	0
0	0	4	8	9	9	9	8	7	5	1	0
0	0	4	7	8	9	9	8	7	5	0	0
0	0	3	6	7	8	8	7	7	3	0	0
0	0	0	2	4	7	8	4	3	0	0	0
0	0	0	0	0	4	7	0	0	0	0	0
0	0	0	0	0	5	6	0	0	0	0	0
0	0	0	0	2	3	4	2	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Quantization (L=10)

Image Coordinate System



- Image is a 2D array of pixels
- Each pixel location: $f(j,i)$ or $f(y,x)$
  - y, x are integer row, column indices
- Image: $f(\boldsymbol{x})$ or

$$f(j,i), 0 \le j \le M-1, 0 \le i \le N-1$$

- Color image: each pixel has three values

$$f_r(\boldsymbol{x}), f_g(\boldsymbol{x}), f_b(\boldsymbol{x})$$

Image in OpenCV

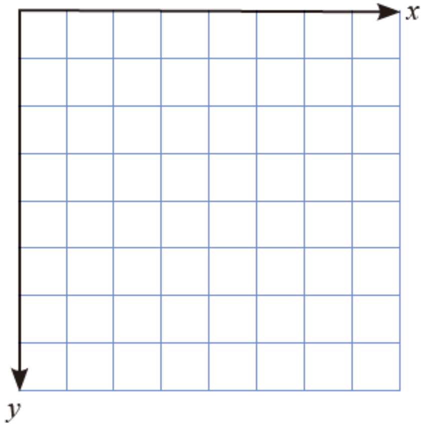
- An image in OpenCV: a numpy.ndarray object
  - Numpy: a package including mathematical functions, random number generators, linear algebra routines, etc. (https://numpy.org)
  - Numpy.ndarray: N-dimensional array object
  - Can use member functions of numpy.ndarray

    - 'dir(img)' to list all member functions

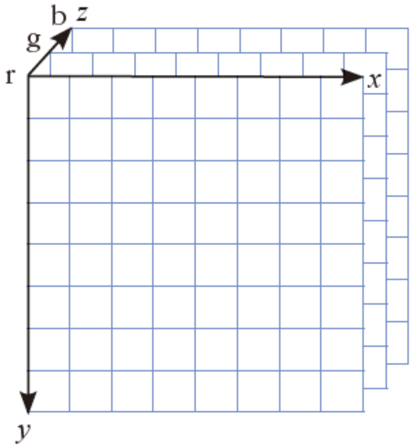
```
In [2]: type(img)
Out[2]: numpy.ndarray

In [3]: img.shape
Out[3]: (600, 800, 3)
```

```
In [7]: help(img.trace)
Help on built-in function trace:

trace(...) method of numpy.ndarray instance
    a.trace(offset=0, axis1=0, axis2=1, dtype=None, out=None)

    Return the sum along diagonals of the array.

    Refer to `numpy.trace` for full documentation.

    See Also
    --------
    numpy.trace : equivalent function
```

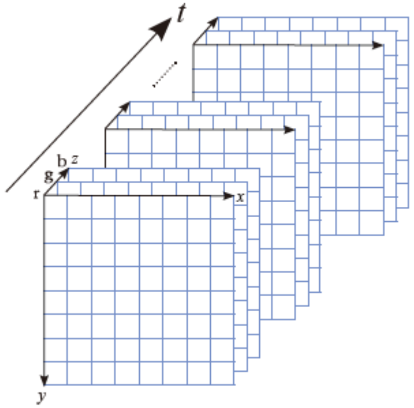
Different Types of Digital Images



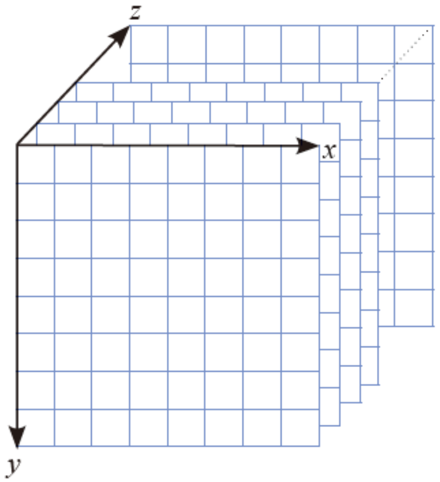
Binary image



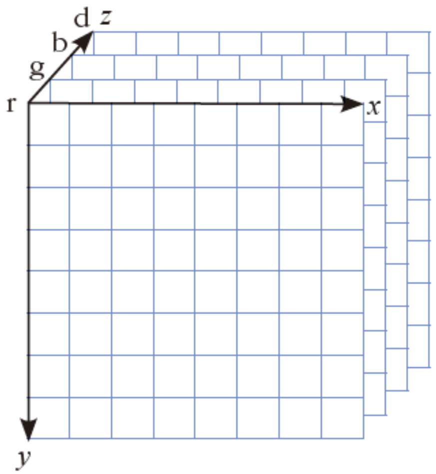
Color image (RGB)



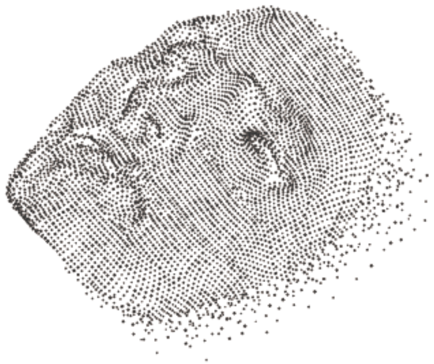
Color image sequence (video)



CT/MRI slice images



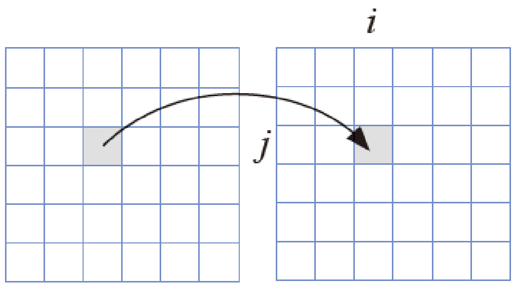
RGBD (RGBA) image



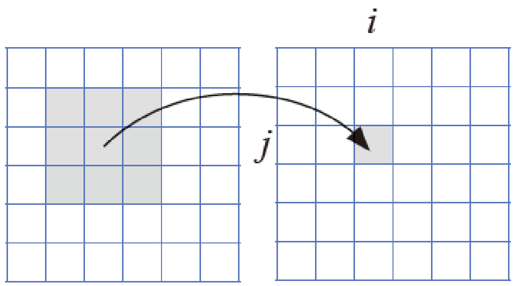
Point cloud image

Image Processing

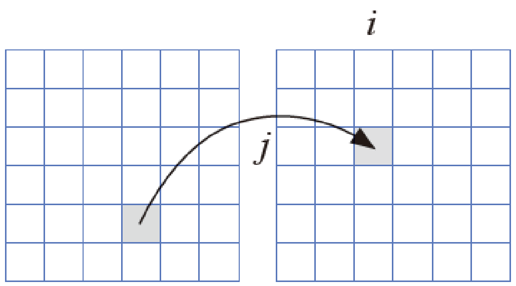
- Image processing operators: map pixel values from one image to another
  1. Point operators: manipulate each pixel independently
  2. Neighborhood (area-based) operators: each pixel depends on a small neighboring input values
  3. Geometric transformation: global operation such as rotations, shears, and perspective deformations



Point operators



Neighborhood (area-based) operators

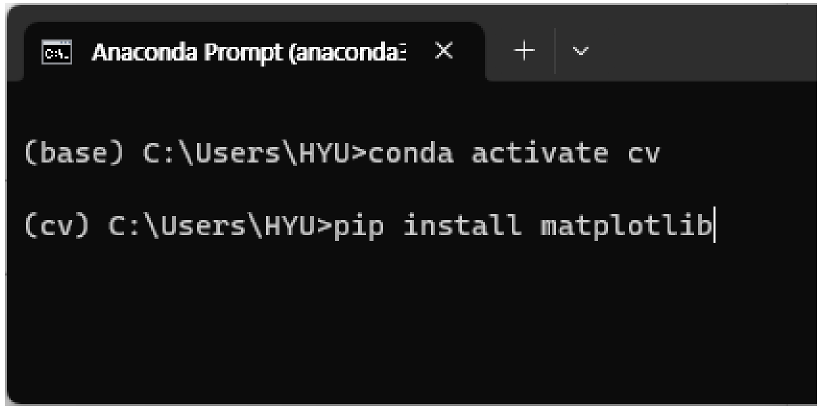


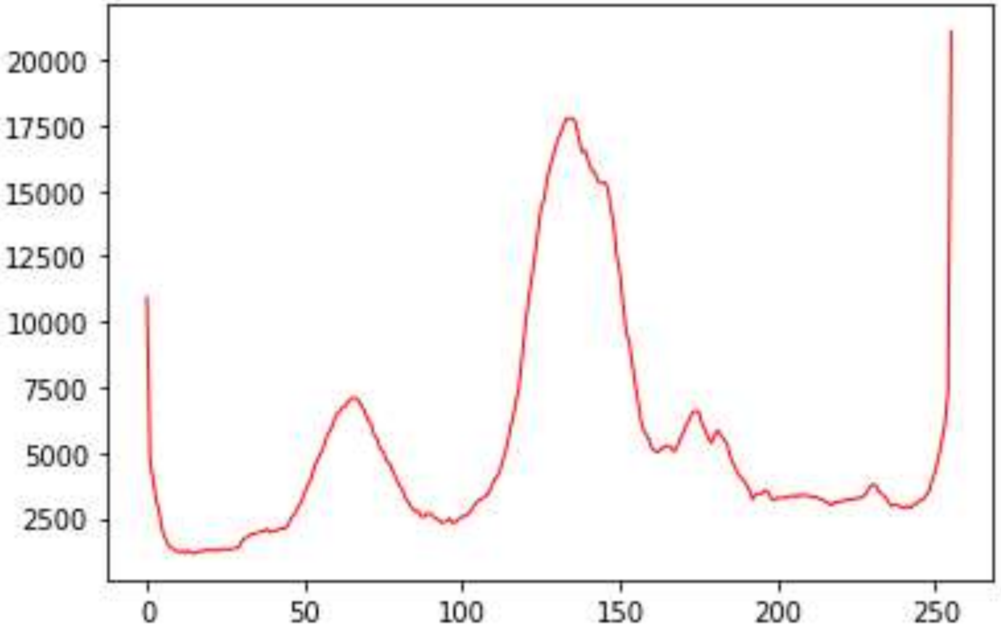
Geometric operators

(Prerequisite) Matplotlib Installation

- matplotlib: a library for drawing graphs

In Anaconda Prompt,





An example graph drawn using matplotlib

(Prerequisite) Numpy Programming

- Numpy tutorials: https://numpy.org/doc/stable/user/quickstart.html
- Numpy.ndarray
  - Numpy's array class to store n-dimensional array
  - We will use matrix, vector operations of Numpy.ndarray in image processing
- ndarray attributes
  - ndarray.ndim: the number of axes (dimensions)
  - ndarray.shape: the dimensions of the array
  - ndarray.size: the total number of elements
  - ndarray.dtype: the type of the elements
  - ndarray.itemsize: the size in bytes of each element
  - Ndarray.data: the buffer containing each element

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
```

(Prerequisite) Numpy Programming

- Array creation
  - Create from python list or tuple – the type of the array is either deduced from the elements or specified on creation

```
>>> import numpy as np
>>> a = np.array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

```
>>> c = np.array([[1, 2], [3, 4]], dtype=complex)
>>> c
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```

  - zeros() creates an array of 0s, and empty() creates an array of random numbers in memory

```
>>> np.zeros((3, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> np.ones((2, 3, 4), dtype=np.int16)
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
>>> np.empty((2, 3))
array([[3.73603959e-262, 6.02658058e-154, 6.55490914e-260],  # may vary
       [5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])
```

(Prerequisite) Numpy Programming

- Array creation
  - arange() creates an array of numbers in a range given a stepsize (≈ python range())

```
>>> np.arange(10, 30, 5)
array([10, 15, 20, 25])
>>> np.arange(0, 2, 0.3)  # it accepts float arguments
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

  - linspace() creates an array of numbers in a range given the number of elements

```
>>> from numpy import pi
>>> np.linspace(0, 2, 9)                 # 9 numbers from 0 to 2
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
>>> x = np.linspace(0, 2 * pi, 100)      # useful to evaluate function at lots of
>>> f = np.sin(x)
```

(Prerequisite) Numpy Programming

- Array operations

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.arange(4)
>>> b
array([0, 1, 2, 3])
>>> c = a - b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10 * np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a < 35
array([ True,  True, False, False])
```

```
>>> A = np.array([[1, 1],
...               [0, 1]])
>>> B = np.array([[2, 0],
...               [3, 4]])
>>> A * B     # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B     # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)  # another matrix product
array([[5, 4],
       [3, 4]])
```

```
>>> b = np.arange(12).reshape(3, 4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)     # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)     # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)  # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

- Arithmetic operators apply elementwise
- *: elementwise product, @: matrix product
- sum(), min(): can specify the axis (by default, apply to all elements)

(Prerequisite) Numpy Programming

- Indexing, slicing, iterating
  - One-dimensional array indexed, sliced, and iterated like python lists
  - Multi-dimensional array indexed, sliced, and iterated per axis

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> # equivalent to a[0:6:2] = 1000;
>>> # from start to position 6, exclusive, set every 2nd element to 1000
>>> a[:6:2] = 1000
>>> a
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
>>> a[::-1]  # reversed a
array([ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
```

```
>>> def f(x, y):
...     return 10 * x + y
...
>>> b = np.fromfunction(f, (5, 4), dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2, 3]
23
>>> b[0:5, 1]  # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[:, 1]    # equivalent to the previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, :]  # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

(Prerequisite) Numpy Programming

- Indexing, slicing, iterating
  - One-dimensional array indexed, sliced, and iterated like python lists
  - Multi-dimensional array indexed, sliced, and iterated per axis

```
>>> a = np.arange(10)**3
>>> a
array([   0,    1,    8,   27,   64,  125,  216,  343,  512,  729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> # equivalent to a[0:6:2] = 1000;
>>> # from start to position 6, exclusive, set every 2nd element to 1000
>>> a[:6:2] = 1000
>>> a
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
>>> a[::-1]  # reversed a
array([ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
```

```
>>> def f(x, y):
...     return 10 * x + y
...
>>> b = np.fromfunction(f, (5, 4), dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2, 3]
23
>>> b[0:5, 1]  # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[:, 1]    # equivalent to the previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, :]  # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

(Prerequisite) Numpy Programming

- Slicing in Numpy ndarray
  - Colon(:) is considered complete slices
  - If fewer indices are used, missing slices are considered complete slices
  - Dots (…) can be used to represent as many axes as needed

```
>>> b[-1]   # the last row. Equivalent to b[-1, :]
array([40, 41, 42, 43])
```

```
>>> c = np.array([[[  0,  1,  2],  # a 3D array (two stacked 2D arrays)
...                [ 10, 12, 13]],
...               [[100, 101, 102],
...                [110, 112, 113]]])
>>> c.shape
(2, 2, 3)
>>> c[1, ...]  # same as c[1, :, :] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[..., 2]  # same as c[:, :, 2]
array([[  2,  13],
       [102, 113]])
```

(Prerequisite) Numpy Programming

- Shape manipulation
  - Change the shape of an array: reshape(), resize()
  - Stacking together different arrays: hstack(), vstack(), column_stack(),
  - Splitting one array into smaller ones: hsplit(), vsplit()

```
>>> a = np.floor(10 * rg.random((3, 4)))
>>> a
array([[3., 7., 3., 4.],
       [1., 4., 2., 2.],
       [7., 2., 4., 9.]])
>>> a.shape
(3, 4)
```

```
>>> a.ravel()  # returns the array, flattened
array([3., 7., 3., 4., 1., 4., 2., 2., 7., 2., 4., 9.])
>>> a.reshape(6, 2)  # returns the array with a modified shape
array([[3., 7.],
       [3., 4.],
       [1., 4.],
       [2., 2.],
       [7., 2.],
       [4., 9.]])
>>> a.T  # returns the array, transposed
array([[3., 1., 7.],
       [7., 4., 2.],
       [3., 2., 4.],
       [4., 2., 9.]])
>>> a.T.shape
(4, 3)
>>> a.shape
(3, 4)
```

```
>>> a
array([[3., 7., 3., 4.],
       [1., 4., 2., 2.],
       [7., 2., 4., 9.]])
>>> a.resize((2, 6))
>>> a
array([[3., 7., 3., 4., 1., 4.],
       [2., 2., 7., 2., 4., 9.]])
```

(Prerequisite) Numpy Programming

- Shape manipulation
  - Change the shape of an array: reshape(), resize()
  - Stacking together different arrays: hstack(), vstack(), column_stack(),
  - Splitting one array into smaller ones: hsplit(), vsplit()

```
>>> a = np.floor(10 * rg.random((2, 2)))
>>> a
array([[9., 7.],
       [5., 2.]])
>>> b = np.floor(10 * rg.random((2, 2)))
>>> b
array([[1., 9.],
       [5., 1.]])
>>> np.vstack((a, b))
array([[9., 7.],
       [5., 2.],
       [1., 9.],
       [5., 1.]])
>>> np.hstack((a, b))
array([[9., 7., 1., 9.],
       [5., 2., 5., 1.]])
```

```
>>> from numpy import newaxis
>>> np.column_stack((a, b))  # with 2D arrays
array([[9., 7., 1., 9.],
       [5., 2., 5., 1.]])
>>> a = np.array([4., 2.])
>>> b = np.array([3., 8.])
>>> np.column_stack((a, b))  # returns a 2D array
array([[4., 3.],
       [2., 8.]])
>>> np.hstack((a, b))        # the result is different
array([4., 2., 3., 8.])
>>> a[:, newaxis]  # view `a` as a 2D column vector
array([[4.],
       [2.]])
>>> np.column_stack((a[:, newaxis], b[:, newaxis]))
array([[4., 3.],
       [2., 8.]])
>>> np.hstack((a[:, newaxis], b[:, newaxis]))  # the result is the same
array([[4., 3.],
       [2., 8.]])
```

(Prerequisite) Numpy Programming

- Shape manipulation
  - Change the shape of an array: reshape(), resize()
  - Stacking together different arrays: hstack(), vstack(), column_stack(),
  - Splitting one array into smaller ones: hsplit(), vsplit()

```
>>> a = np.floor(10 * rg.random((2, 12)))
>>> a
array([[6., 7., 6., 9., 0., 5., 4., 0., 6., 8., 5., 2.],
       [8., 5., 5., 7., 1., 8., 6., 7., 1., 8., 1., 0.]])
>>> # Split `a` into 3
>>> np.hsplit(a, 3)
[array([[6., 7., 6., 9.],
       [8., 5., 5., 7.]]), array([[0., 5., 4., 0.],
       [1., 8., 6., 7.]]), array([[6., 8., 5., 2.],
       [1., 8., 1., 0.]])]
>>> # Split `a` after the third and the fourth column
>>> np.hsplit(a, (3, 4))
[array([[6., 7., 6.],
       [8., 5., 5.]]), array([[9.],
       [7.]]), array([[0., 5., 4., 0., 6., 8., 5., 2.],
       [1., 8., 6., 7., 1., 8., 1., 0.]])]
```

Example1. Manipulating RGB Image Shape

```python
import cv2 as cv
import sys

img=cv.imread('Erica.jpg')

if img is None:
    sys.exit('File not found')

cv.imshow('original_RGB',img)
cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,
                            img.shape[1]//4:3*img.shape[1]//4,:])

cv.imshow('R channel',img[:,:,2])
cv.imshow('G channel',img[:,:,1])
cv.imshow('B channel',img[:,:,0])

cv.waitKey()
cv.destroyAllWindows()
```

Execution Results

Example1. Manipulating RGB Image Shape

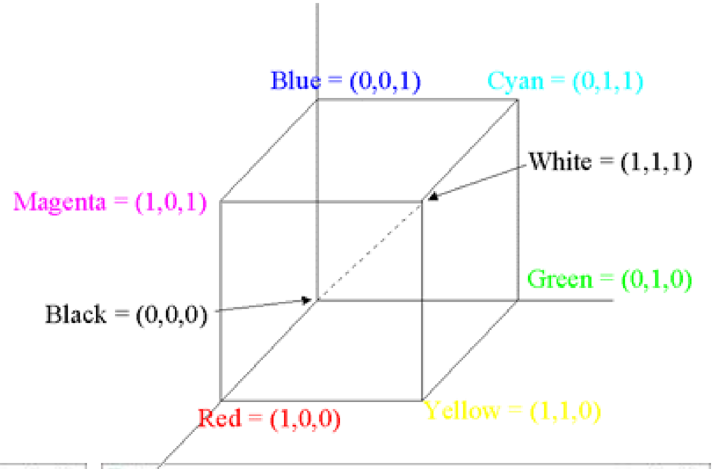
```
1  import cv2 as cv
2  import sys
3
4  img=cv.imread('Erica.jpg')
5
6  if img is None:
7      sys.exit('File not found')
8
9  cv.imshow('original_RGB',img)
10 cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
11 cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,
12                             img.shape[1]//4:3*img.shape[1]//4,:])
13
14 cv.imshow('R channel',img[:,:,2])
15 cv.imshow('G channel',img[:,:,1])
16 cv.imshow('B channel',img[:,:,0])
17
18 cv.waitKey()
19 cv.destroyAllWindows()
```

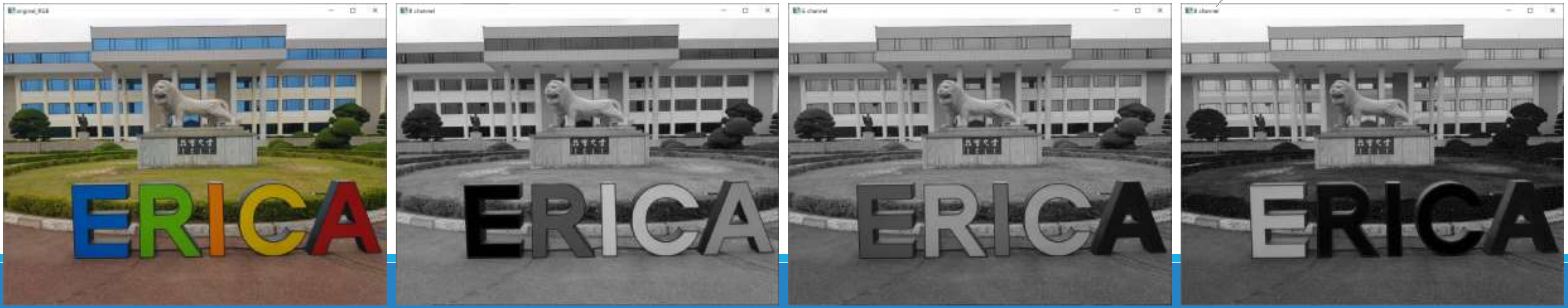
- (L10) Crop the upper left half of the original image
- (L11) Crop the center half of the original image
- Question) Can you draw the lower half of the original image?

Example1. Manipulating RGB Image Shape

```python
1  import cv2 as cv
2  import sys
3
4  img=cv.imread('Erica.jpg')
5
6  if img is None:
7      sys.exit('File not found')
8
9  cv.imshow('original_RGB',img)
10 cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
11 cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,
12                             img.shape[1]//4:3*img.shape[1]//4,:])
13
14 cv.imshow('R channel',img[:,:,2])
15 cv.imshow('G channel',img[:,:,1])
16 cv.imshow('B channel',img[:,:,0])
17
18 cv.waitKey()
19 cv.destroyAllWindows()
```

- (L14~16) Draw red, green, blue colors of the image
- Compare intensities of “ERICA” in three images
- Q1) What is the average of each channel?





Example2. Gamma Correction

```python
# Gamma correction
import cv2 as cv
import numpy as np

img=cv.imread('erica.jpg')
img=cv.resize(img,dsize=(0,0),fx=0.25,fy=0.25)

def gamma(f,gamma=1.0):
        ?????

gc=np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),
              gamma(img,2.0),gamma(img,3.0)))
cv.imshow('gamma',gc)

cv.waitKey()
cv.destroyAllWindows()
```

Execution result

Example2. Gamma Correction

```
1   # Gamma correction
2   import cv2 as cv
3   import numpy as np
4
5   img=cv.imread('erica.jpg')
6   img=cv.resize(img,dsize=(0,0),fx=0.25,fy=0.25)
7
8   def gamma(f,gamma=1.0):
9
10
11
12  gc=np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),
13                gamma(img,2.0),gamma(img,3.0)))
14  cv.imshow('gamma',gc)
15
16  cv.waitKey()
17  cv.destroyAllWindows()
```

- (L6) Scale down the input image by 1/4
- (L8-10) Define gamma function to perform gamma correction

$$f_{out}(j,i) = (L-1) \times \left(\hat{f}(j,i)\right)^{\gamma},$$

$$\text{where } \hat{f}(j,i) = f(j,i)/(L-1)$$

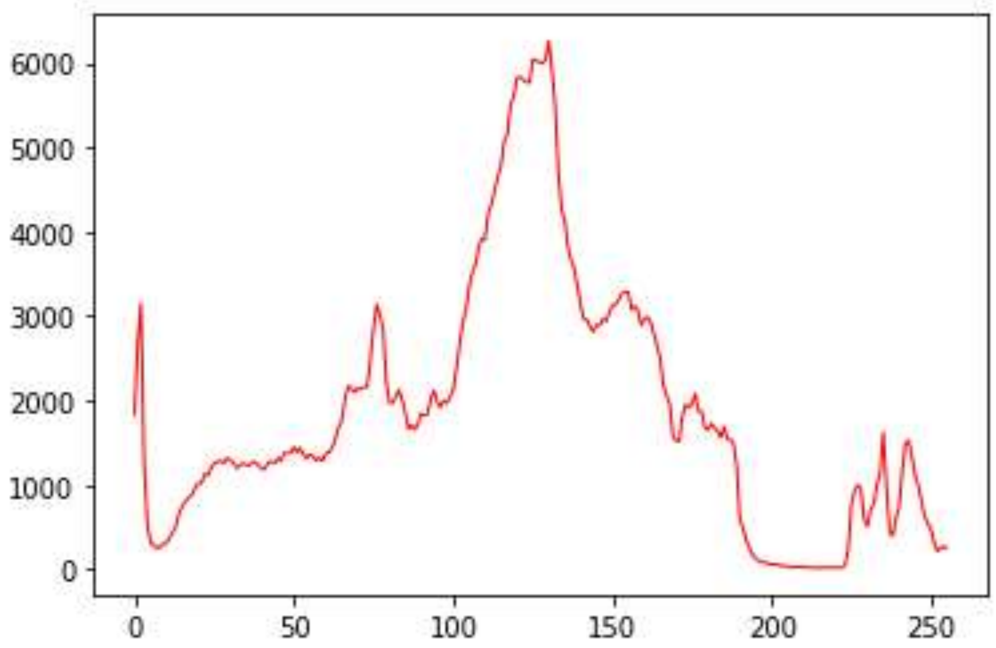
\- Remember that numpy.ndarray operations apply elementwise

- (L12) Stacking multiple images horizontally

Execution result

Example3. Histogram Calculation

```
import cv2 as cv
import matplotlib.pyplot as plt

img=cv.imread('Erica.jpg')
h=cv.calcHist([img],[2],None,[256],[0,256])
plt.plot(h,color='r',linewidth=1)
```



Execution result

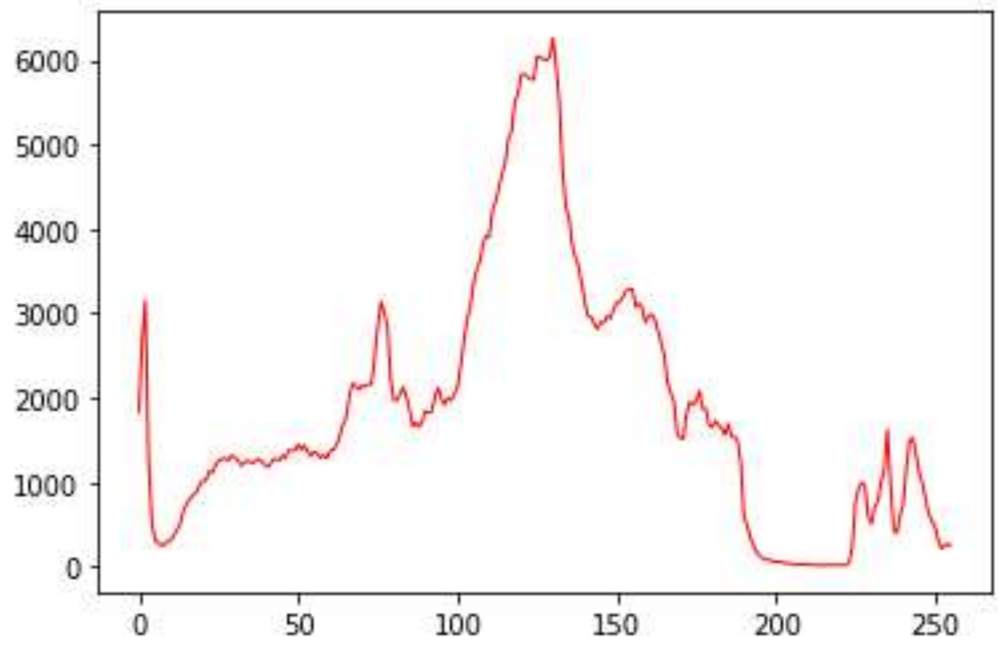
Input image

Example3. Histogram Calculation

```python
1  import cv2 as cv
2  import matplotlib.pyplot as plt
3
4  img=cv.imread('Erica.jpg')
5  h=cv.calcHist([img],[2],None,[256],[0,256])
6  plt.plot(h,color='r',linewidth=1)
```

```
calcHist(...)
    calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) -> hist
    .   @overload
    .
    .   this variant supports only uniform histograms.
    .
    .   ranges argument is either empty vector or a flattened vector of histSize.size()*2 elements
    .   (histSize.size() element pairs). The first and second elements of each pair specify the lower and
    .   upper boundaries.
```

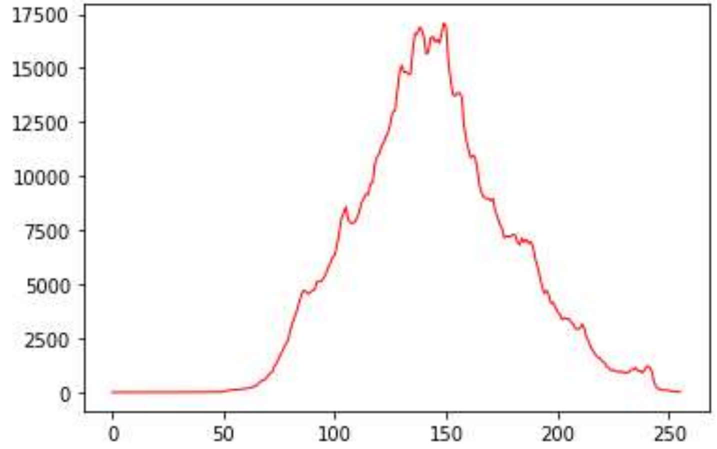
- (L5) Calculate the histogram of img
  - Every argument is given as a list form
  - Calculate a histogram in R channels
  - Find the histogram of full image
  - The number of histogram bins is 256
  - The range of intensity values is [0, 256]
- (L6) Plot the computed histogram

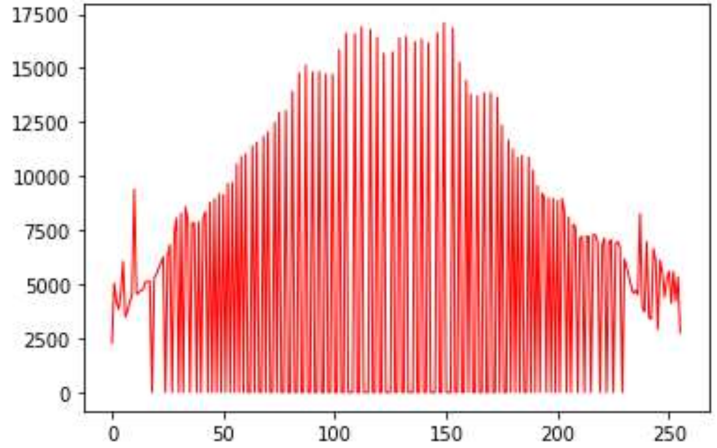


Question) Calculate the histogram of the lower half image?

Example4. Histogram Equalization

```python
# histogram equalization
import cv2 as cv
import matplotlib.pyplot as plt

gray=cv.imread('mistyroad.jpg', cv.IMREAD_GRAYSCALE)
# or
# img=cv.imread('mistyroad.jpg')
# gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)

plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]),

h=cv.calcHist([gray],[0],None,[256],[0,256])
plt.plot(h,color='r',linewidth=1), plt.show()

equal=cv.equalizeHist(gray)
plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()

h=cv.calcHist([equal],[0],None,[256],[0,256])
plt.plot(h,color='r',linewidth=1), plt.show()
```

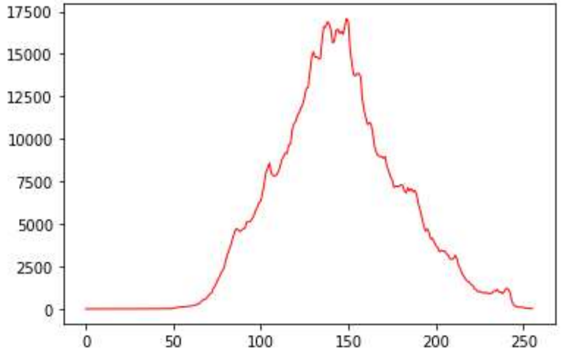


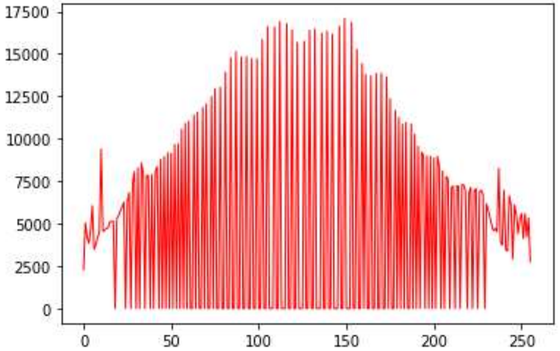


Execution Results

Example4. Histogram Equalization

```
1   # histogram equalization
2   import cv2 as cv
3   import matplotlib.pyplot as plt
4
5   gray=cv.imread('mistyroad.jpg', cv.IMREAD_GRAYSCALE)
6   # or
7   # img=cv.imread('mistyroad.jpg')
8   # gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
9
10  plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
11
12  h=cv.calcHist([gray],[0],None,[256],[0,256])
13  plt.plot(h,color='r',linewidth=1), plt.show()
14
15  equal=cv.equalizeHist(gray)
16  plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
17
18  h=cv.calcHist([equal],[0],None,[256],[0,256])
19  plt.plot(h,color='r',linewidth=1), plt.show()
```





Execution Results

- Histogram equalization: apply a mapping function f(l) to make the histogram as flat as possible

$$l_{out} = T(l_{in}) = round\big(c(l_{in}) \times (L-1)\big), \text{where } c(l_{in}) = \sum_{l=0}^{l_{in}} \hat{h}(l)$$

- (L15) cv.equalizeHist(): a OpenCV built-in function to perform histogram equalization
- (*) For adaptive histogram equalization, search cv.createCLAHE()

Review Task 2024-03-14

1. (a) Scale down the size of images to 50%

   (b) With HSV color model,

- Change the hue by 180 degrees
- Decrease the saturation by 50%
- Increase the value by 50%

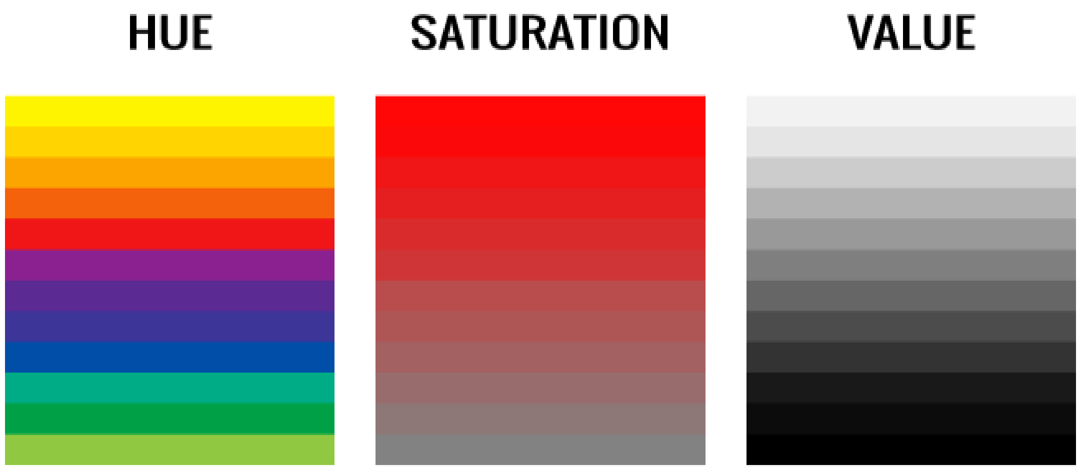
   (c) Merge 2x2 images into a

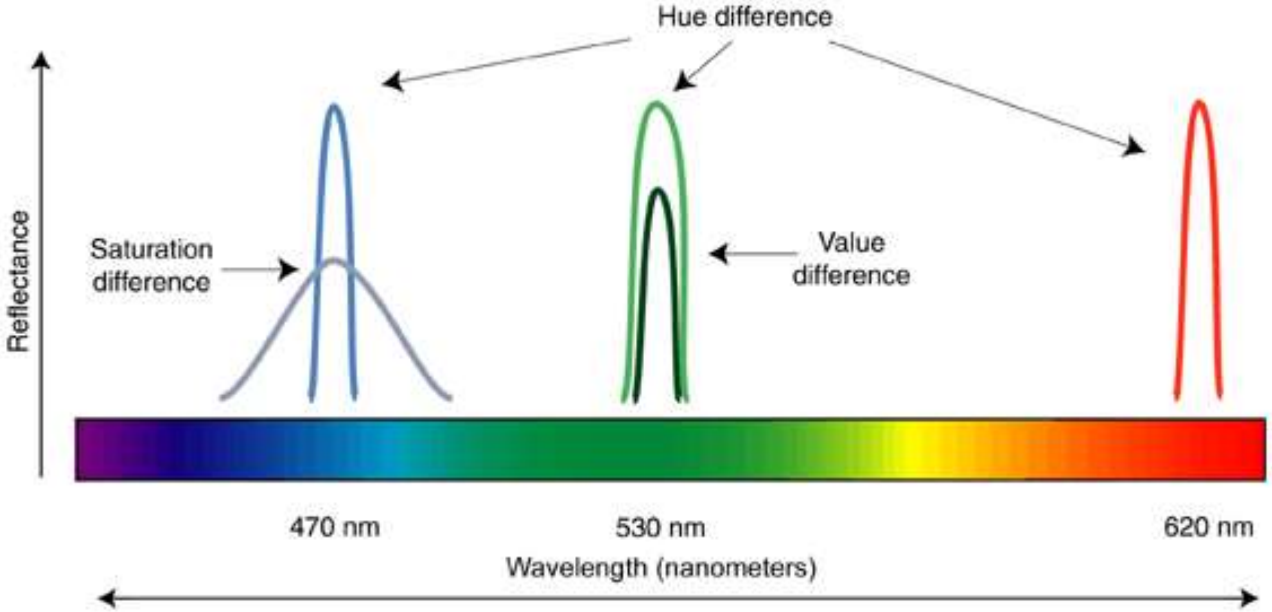
   a single image and save it to

   erica_new1.jpg

Properties of Light

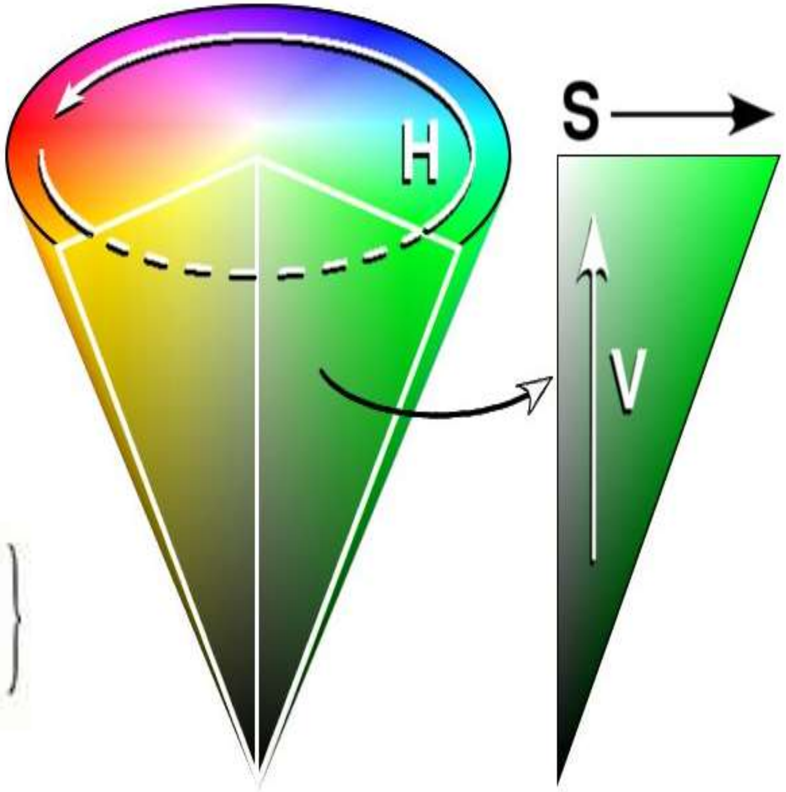
Perception (Qualitative)	Colorimetry (Quantitative)
Hue	Dominant wavelength
Saturation	Purity (Bandwidth)
Value(Brightness)	Luminance (Amount of energy)





HSV Color Model

- HSV (HSB) Model
  - Based on human perception
  - Hue, Saturation, Value
  - Cylindrical coordinate



$$H=\begin{cases}\theta & \text{if } B\leq G\\ 360-\theta & \text{if } B\geq G\end{cases}$$

$$\theta=\cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

$$S=1-\frac{3}{(R+G+B)}[\min(R,G,B)]$$

$$I=\frac{1}{3}(R+G+B)$$