Industrial Computer Vision

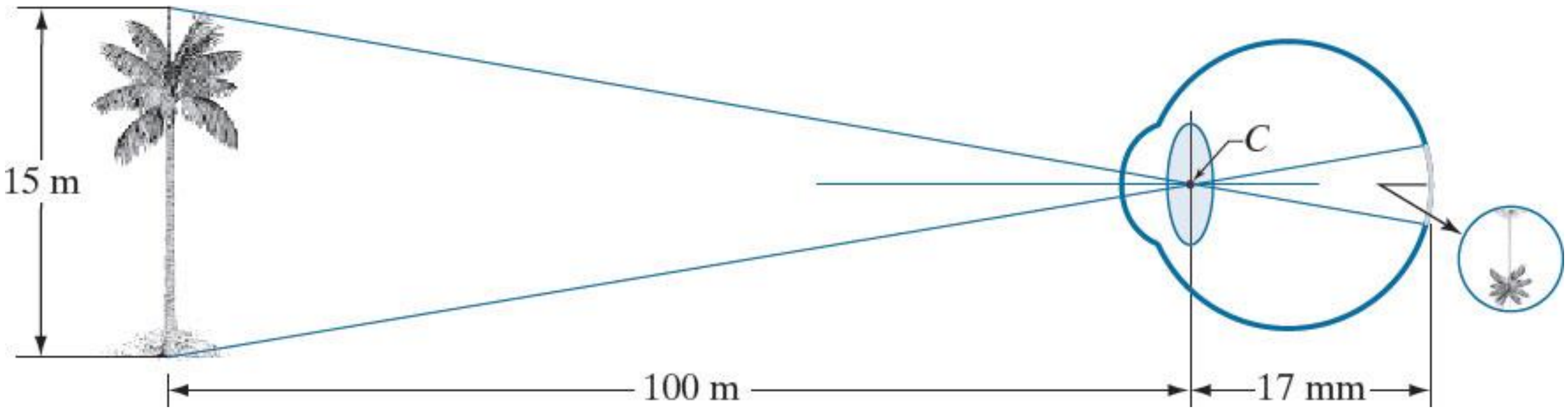
- Image Input/Output & GUI

2nd lecture, 2021.09.08

Lecturer: Youngbae Hwang

Image formation

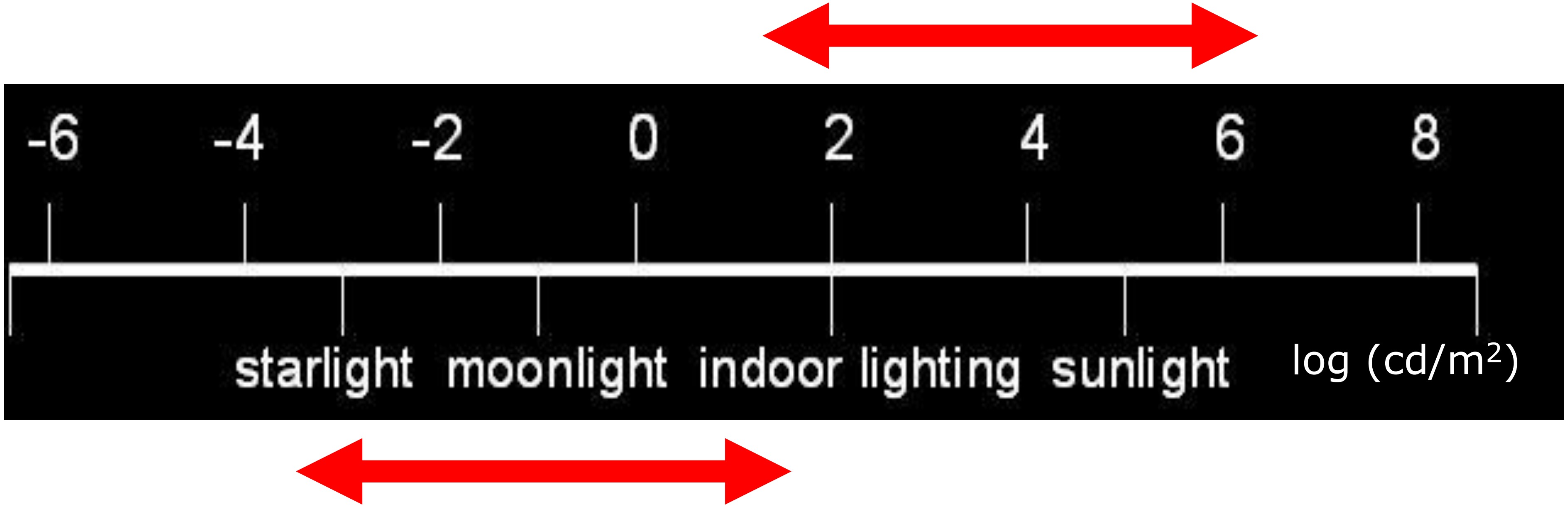
- Image formation in the eye
  - Distance between center of lens and retina (focal length) vary between 14-17 mm.
  - Image length h = 17(mm) x (15/100)



Range of human visual system

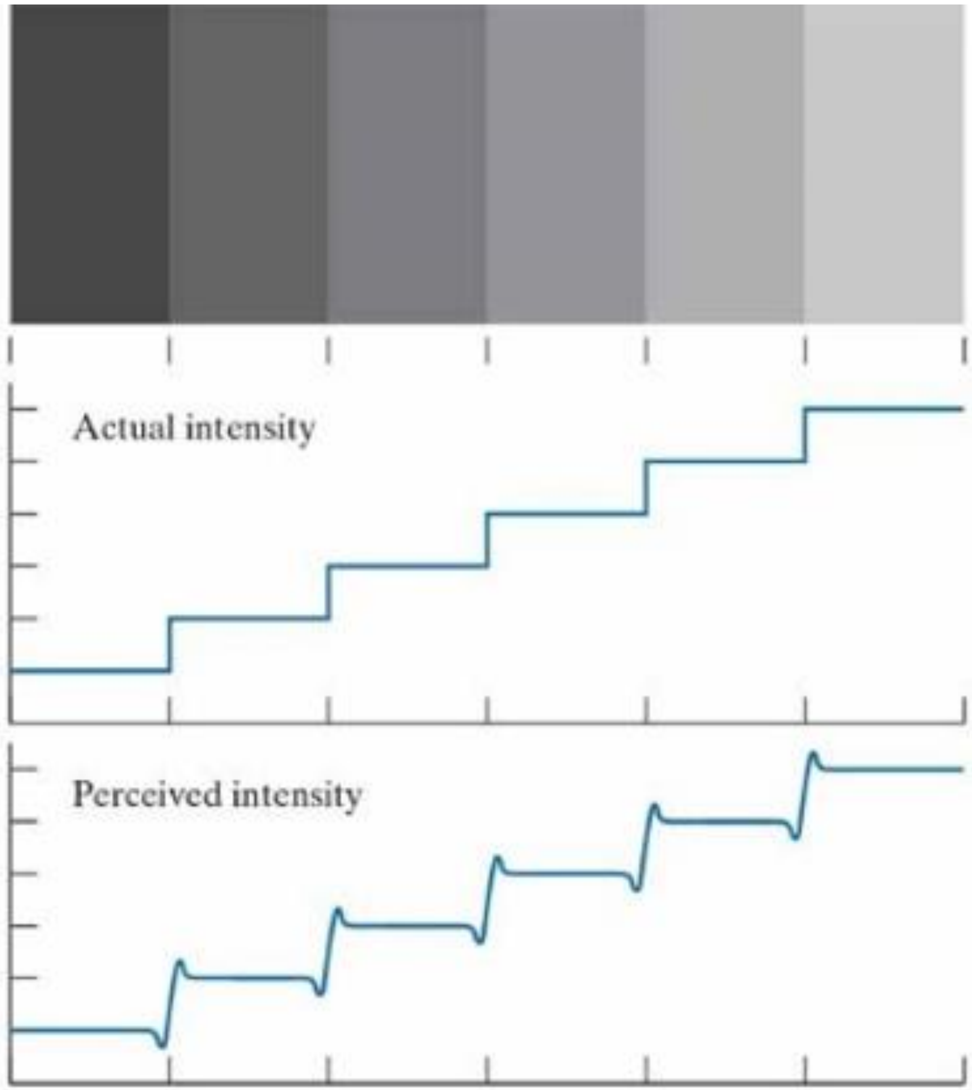


Human simultaneous luminance vision range
(5 orders of magnitude)



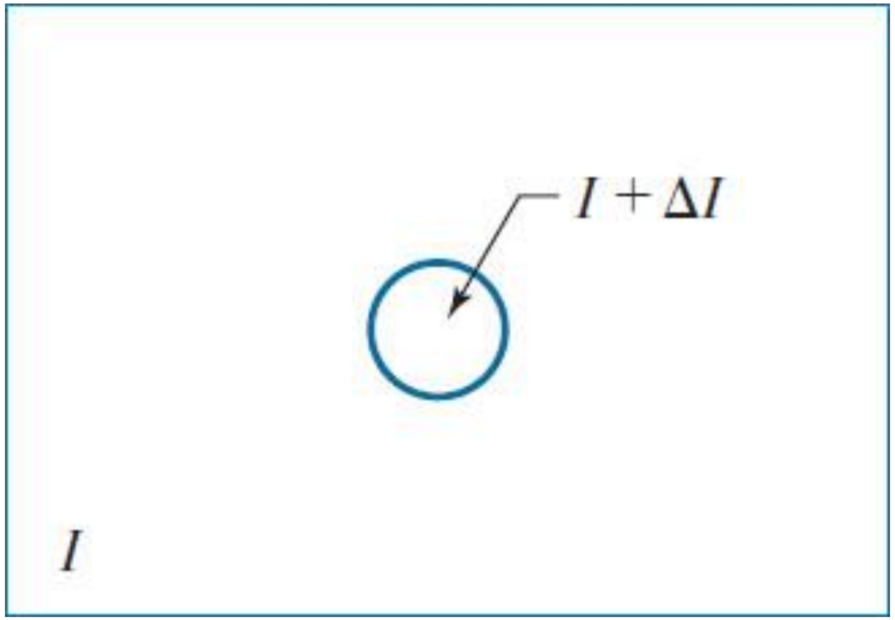
Perceived intensity

- Illustration of the Mach band effect. Perceived intensity is not a simple function of actual intensity

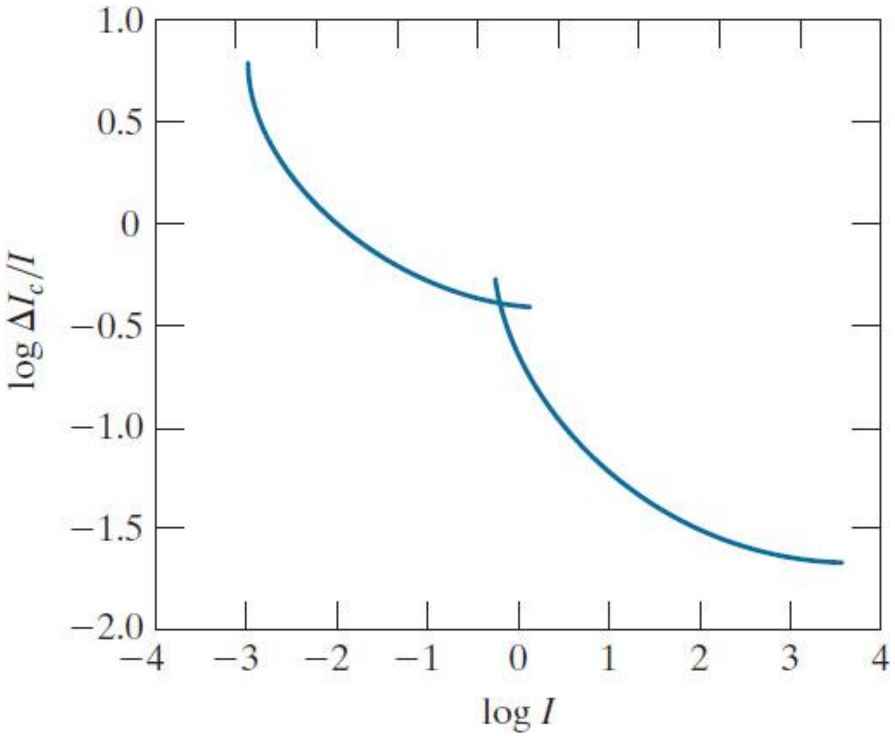


Bright discrimination

- Perceivable changes at a given adaptation level



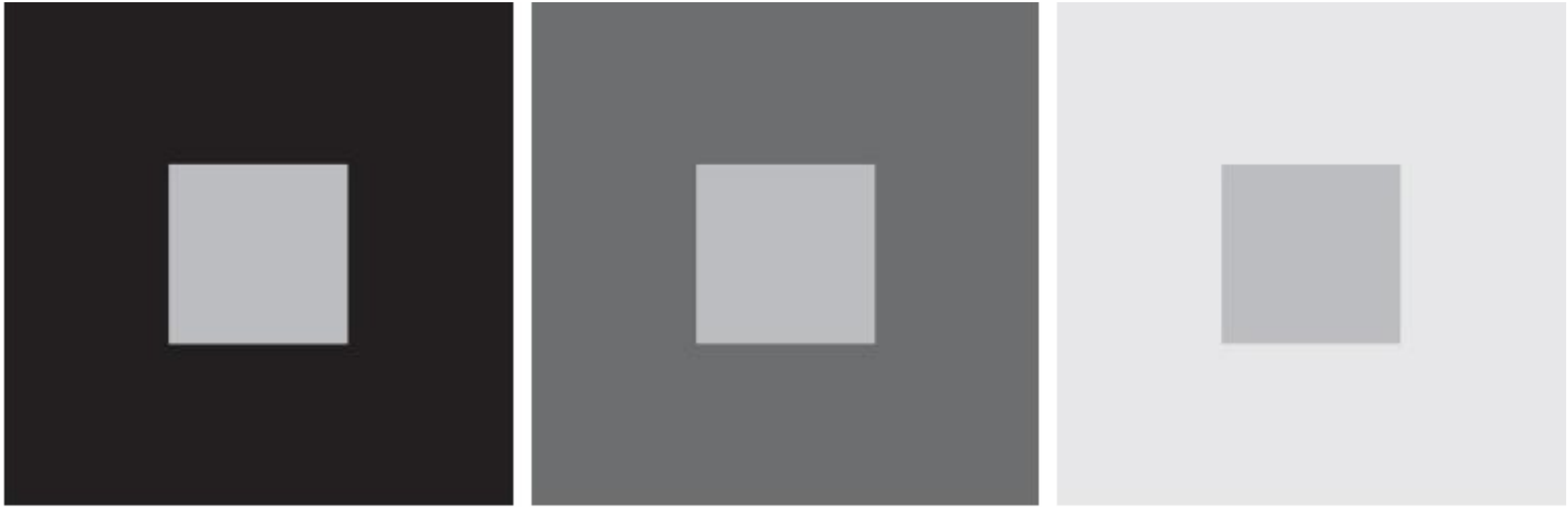
Experimental setup used
to characterize brightness discrimination



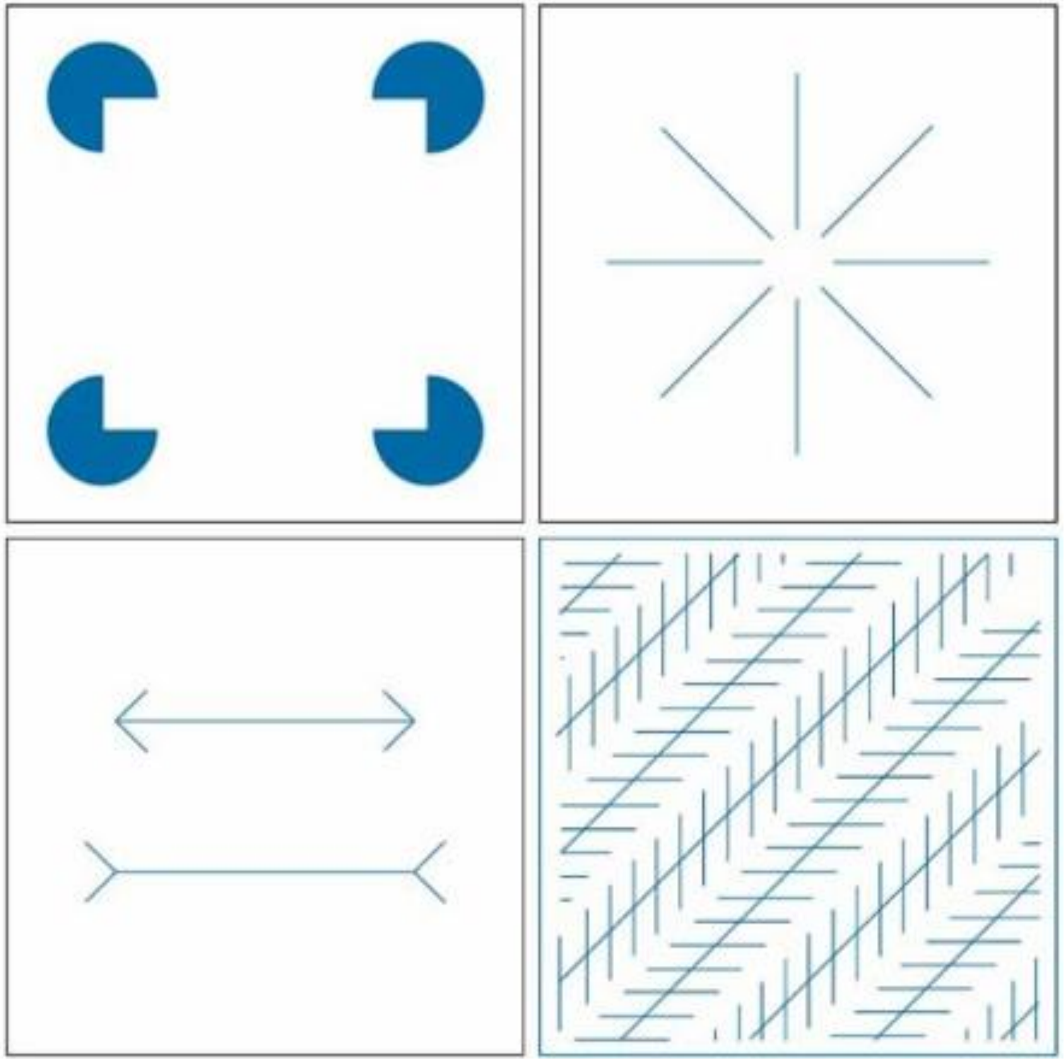
Weber ratio as a function of intensity

Simultaneous constrast

- All the inner squares have the same intensity, but they appear progressively darker as the background becomes lighter.

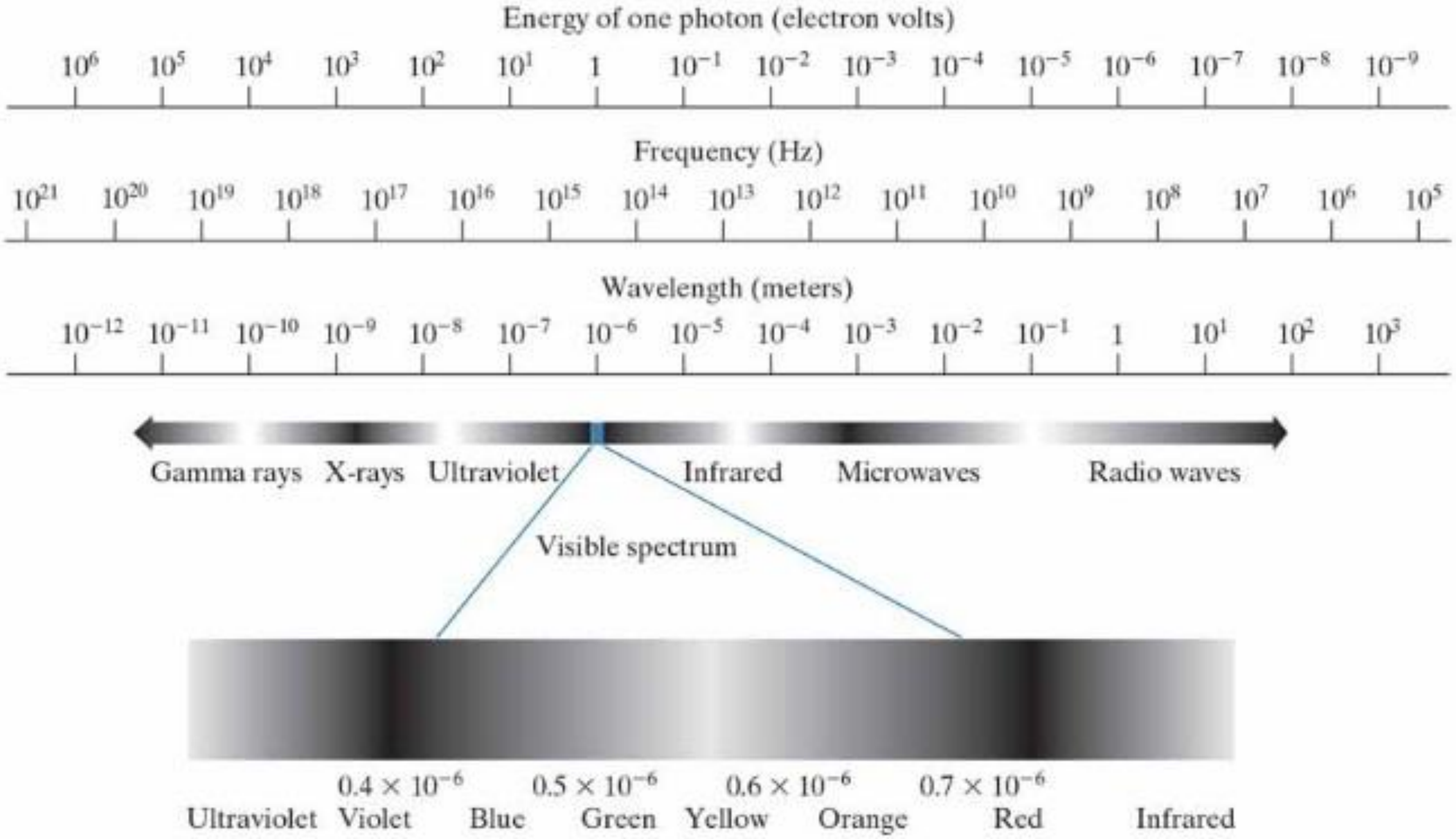


Some well-known optical illusions



Electromagnetic spectrum

- $c = \dfrac{\lambda}{T} = \lambda\nu \rightarrow \nu = \dfrac{c}{\lambda}$ $\qquad \lambda_{vg} \approx 0.55\,\mu m$ $\qquad \nu = \dfrac{3\cdot 10^8\, m/s}{0.55\cdot 10^{-6} m} = \dfrac{3}{0.55} 10^{14}\dfrac{1}{s} \approx 5.45{\cdot}10^{14} Hz$
- $$\begin{aligned} E &= h\nu = 4.13\cdot 10^{-15}\, eVs \cdot 5.45\cdot 10^{14} Hz \\ &\approx 22\cdot 10^{-1} eV = 2.2eV \end{aligned}$$



Light

- Light is a particular type of electromagnetic wave
- The colors that humans perceive are determined by the light *reflected* by the object:

  - all the light reflected: white object

  - some components (of the visible spectrum) absorbed, some reflected: color (wavelength reflected).

  - Light reflected/absorbed at the same rate for all wavelengths: *monochromatic* light.

- Thus we speak of *intensity* or *gray level*

Light

- Properties of light sources/reflected light:
  - Chromatic light (colors): from 0.43 µm to 0.79 µm wavelength

- *Radiance*: Total amount of energy out of the light source (Watts)

- *Luminance*: Amount of light perceived from a light source (lumen) ex. Stars

- *Brightness*: earlier a synonymous of luminance, is now a subjective measurement of light perceived from a light source.

Image acquisition

(a) Single sensing element.

(b) Line sensor.

(c) Array sensor.

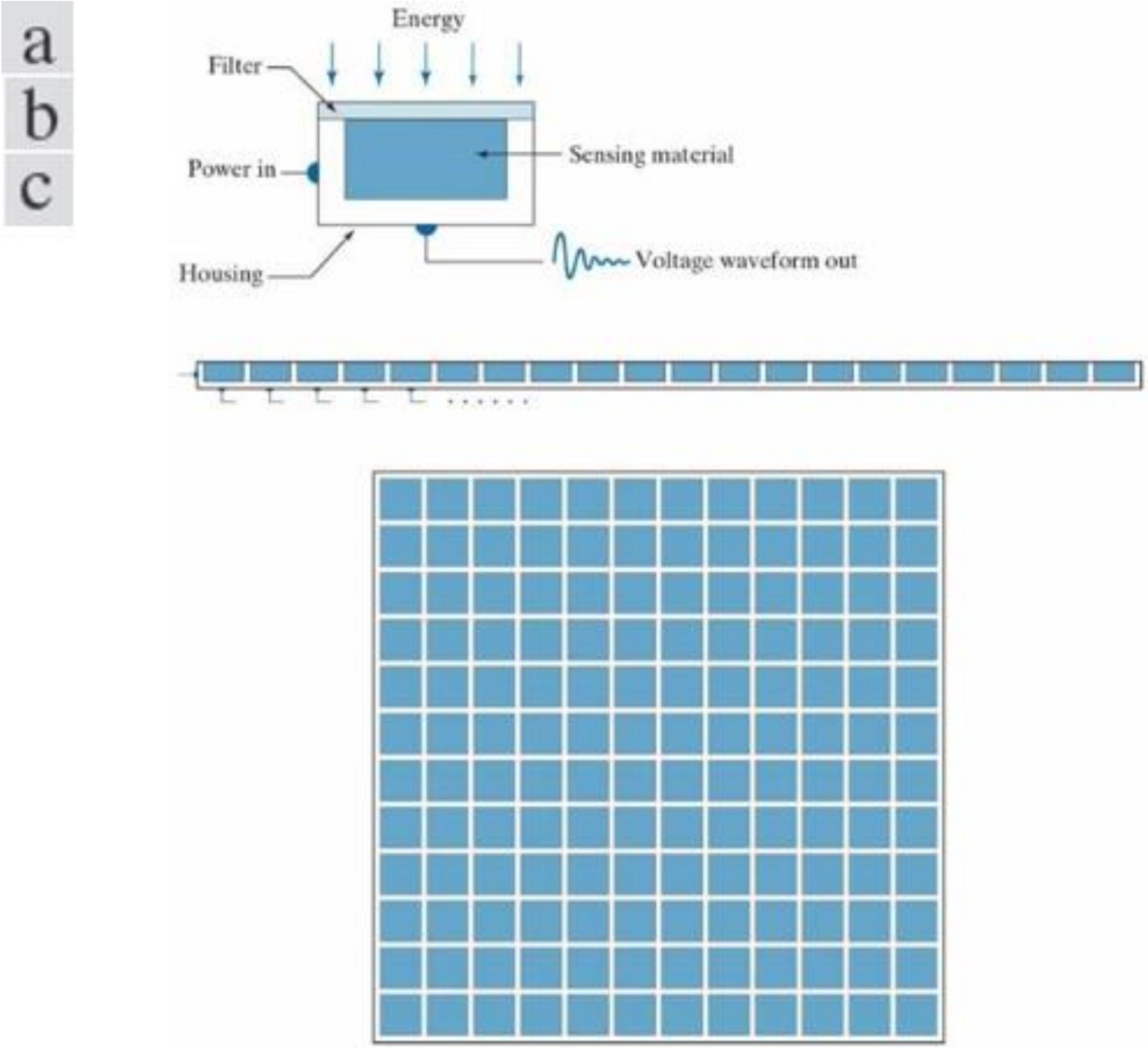
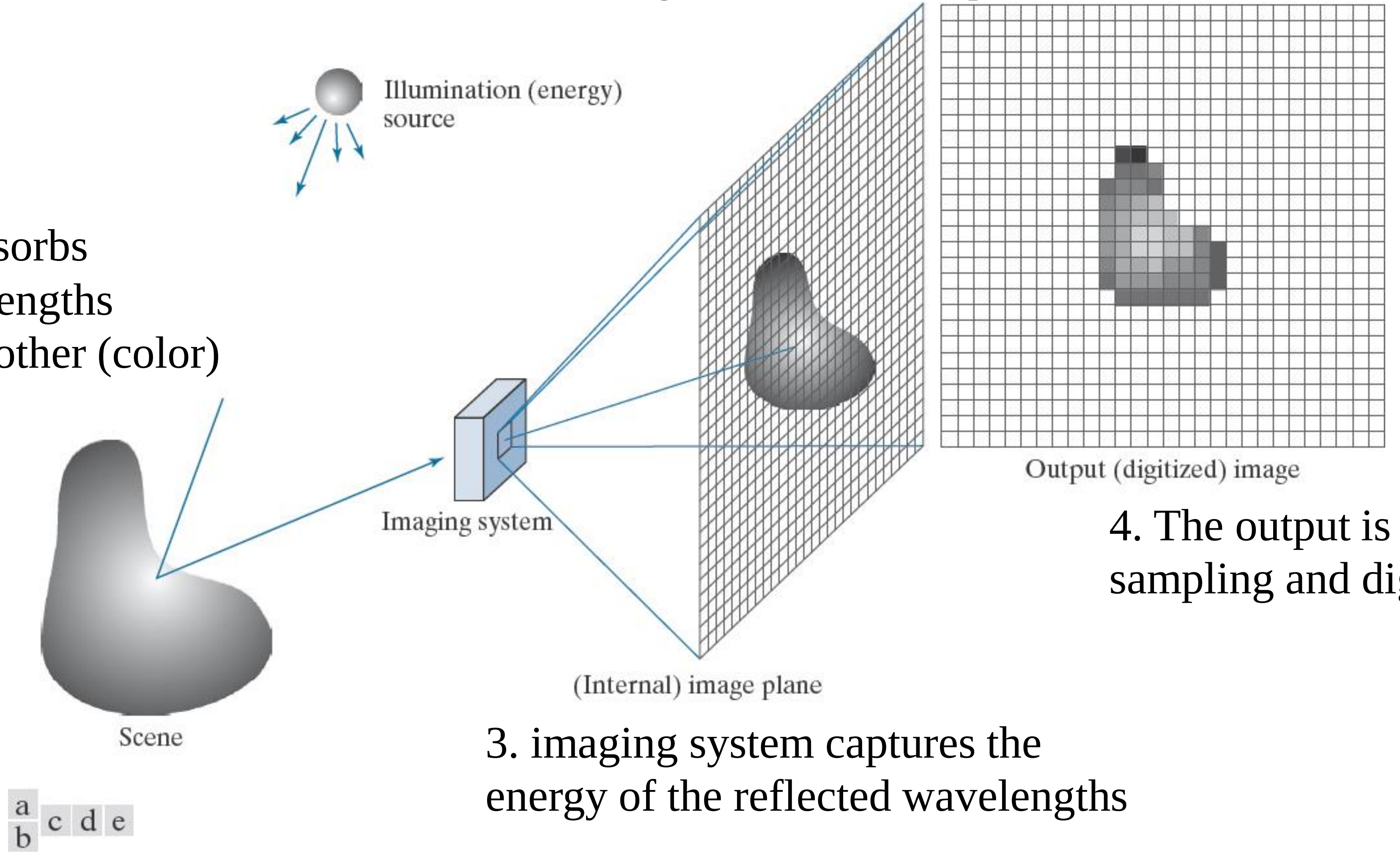


Image acquisition process

1. Illumination source emits light
(all wavelengths in the visible spectrum)

2. Object absorbs
Some wavelengths
and reflects other (color)



4. The output is obtained through
sampling and digitalization

3. imaging system captures the
energy of the reflected wavelengths

What is Digital Image Processing

- Digital Image
  - A two-dimensional function x and y are spatial coordinates $f(x, y)$
  - The amplitude of f is called intensity or gray level at the point (x, y)
- Digital Image Processing
  - Process digital images by means of computer, it covers low-, mid-, and high-level processes
  - low-level: inputs and outputs are images
  - mid-level: outputs are attributes extracted from input images
  - high-level: an ensemble of recognition of individual objects
- Pixel
  - the elements of a digital image

A simple image formation model

$$f(x, y) = i(x, y) \cdot r(x, y)$$

$f(x, y)$: intensity at the point (x, y)

$i(x, y)$: illumination at the point (x, y)

(the amount of source illumination incident on the scene)

$r(x, y)$: reflectance/transmissivity at the point (x, y)

(the amount of illumination reflected/transmitted by the object)

where $0 < i(x, y) < \infty$ and $0 < r(x, y) < 1$

A simple image formation model

In practice, for any point (x_0, y_0) of the image, we require

$$i_{min} r_{min} = L_{min} \leq \ell = f(x_0, y_0) \leq L_{max} = i_{max} r_{max}$$

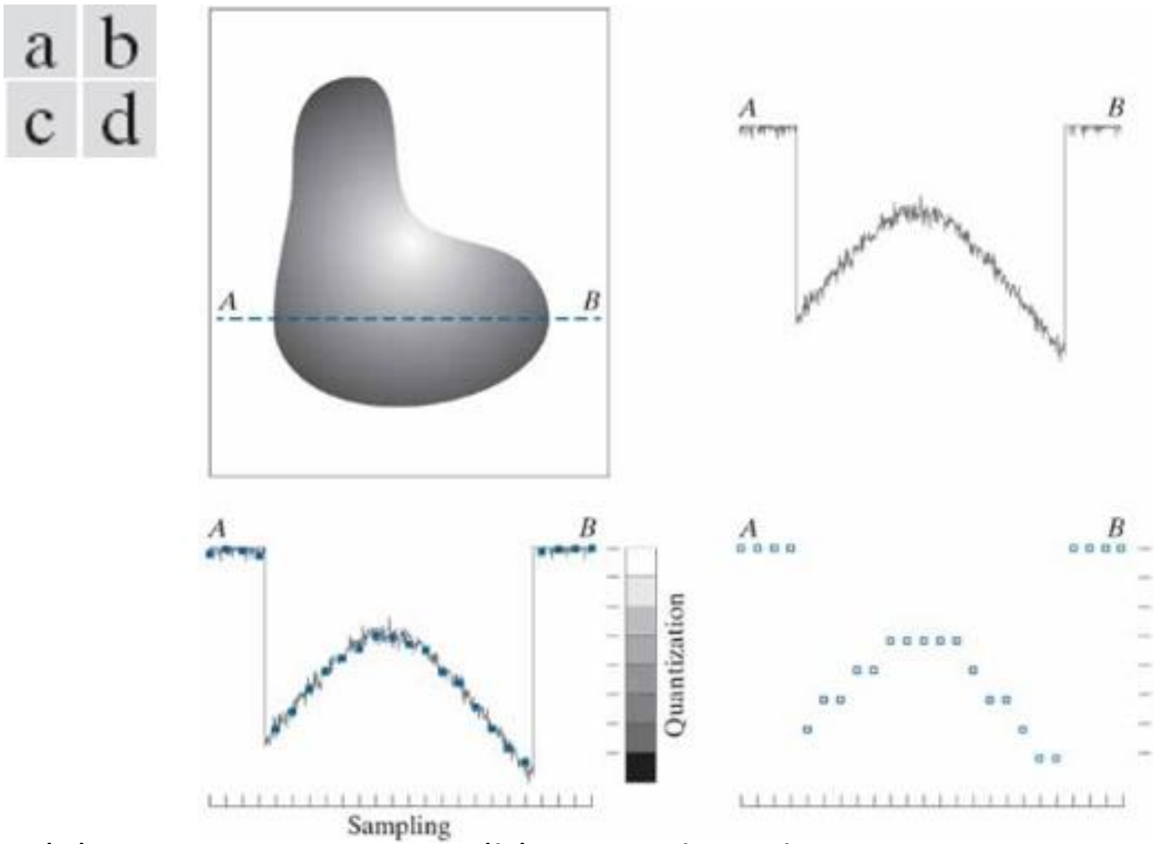
where L_{max} is required to be finite

$[L_{min}, L_{max}]$ is the *intensity* scale (also *gray scale*) of the image.

Common practice: $[L_{min}, L_{max}]$ is shifted to $[0, L-1]$, where $\ell = 0$ is black and $\ell = L - 1$ is white.

Basic concepts in sampling and quantization

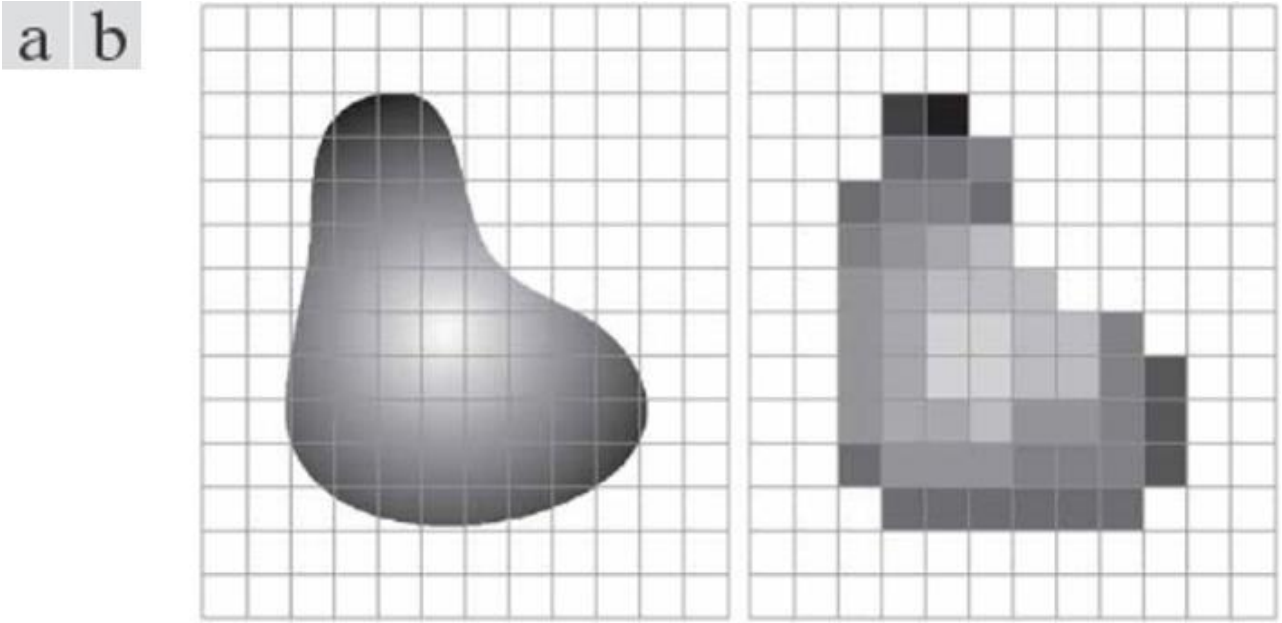
- Image sampling and quantization:
  - From continuous (+noise)
  - to discrete (digitalized)



(a) Continuous image. (b) A scan line showing intensity variations along line **AB** in the continuous image. (c) Sampling and quantization. (d) Digital scan line.

Basic concepts in sampling and quantization

- Because of sampling, the image will be described by a finite set of points (pixels)



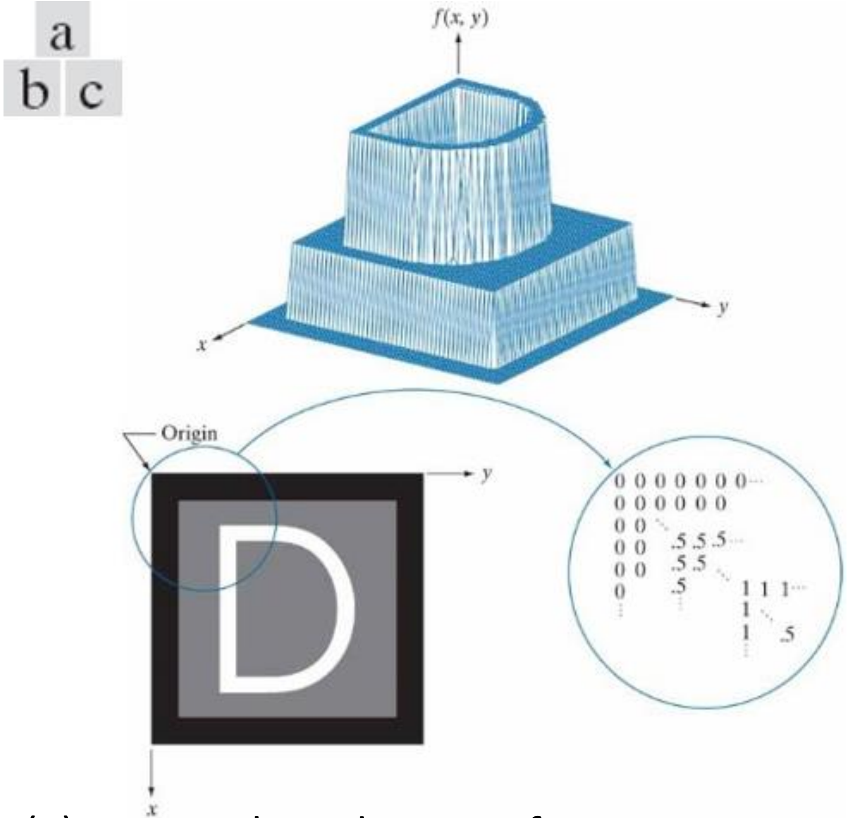
(a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.

Representing digital images

- The representation of an MxN numerical array as

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

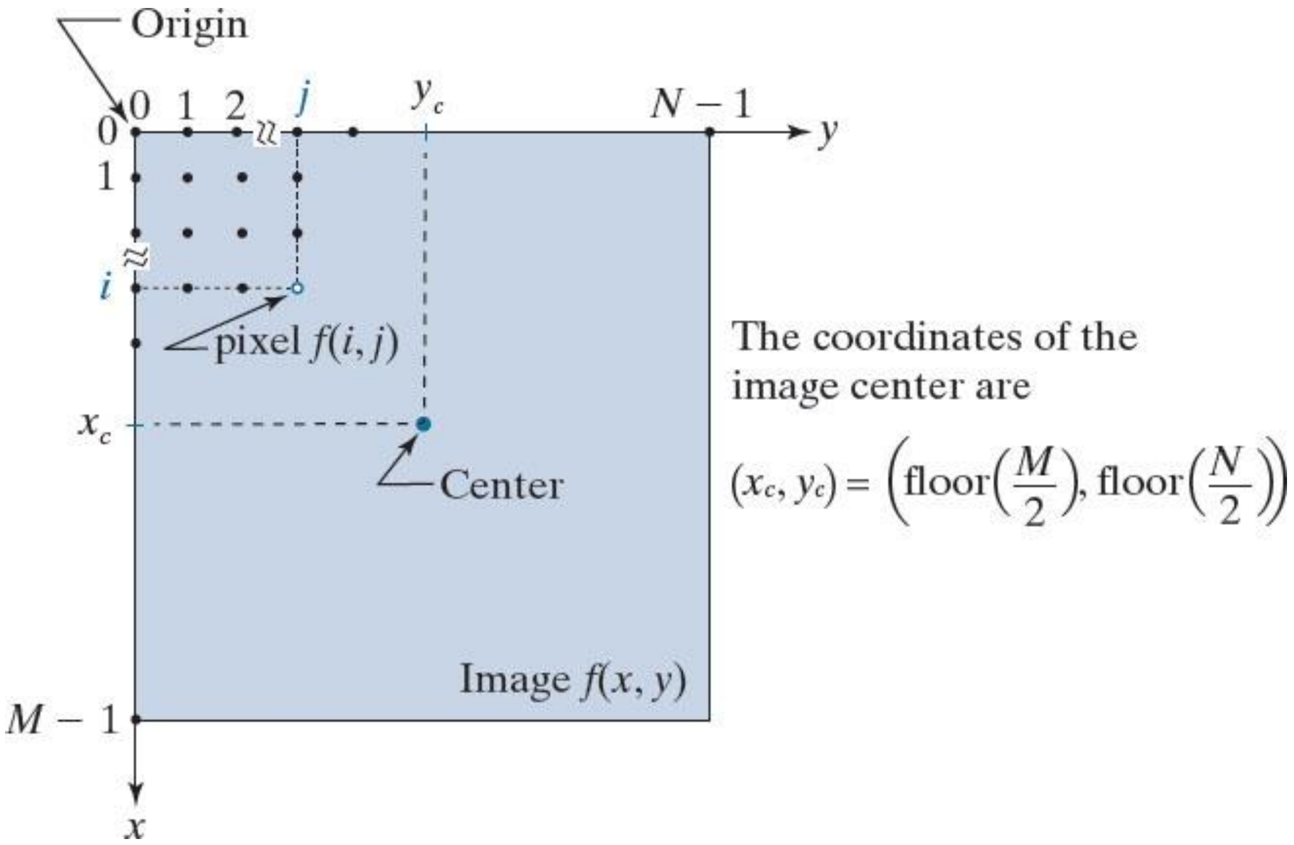
$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,N-1} \\ \dots & \dots & \dots & \dots \\ a_{M-1,0} & a_{M-1,1} & \dots & a_{M-1,N-1} \end{bmatrix}$$



(a) Image plotted as a surface.
(b) Image displayed as a visual intensity array.
(c) Image shown as a 2-D numerical array.

Coordinate convention

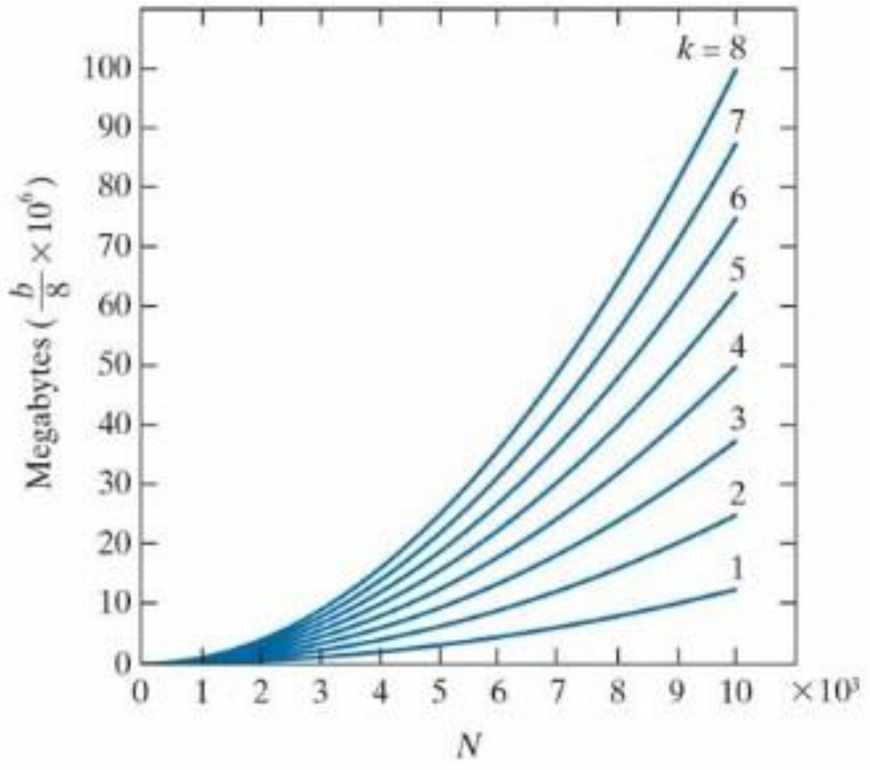
- Coordinate convention used to represent digital images.
  - Because coordinate values are integers, there is a one-to-one correspondence between **x** and **y** and the rows (**r**) and columns (**c**) of a matrix.



Storing digital images

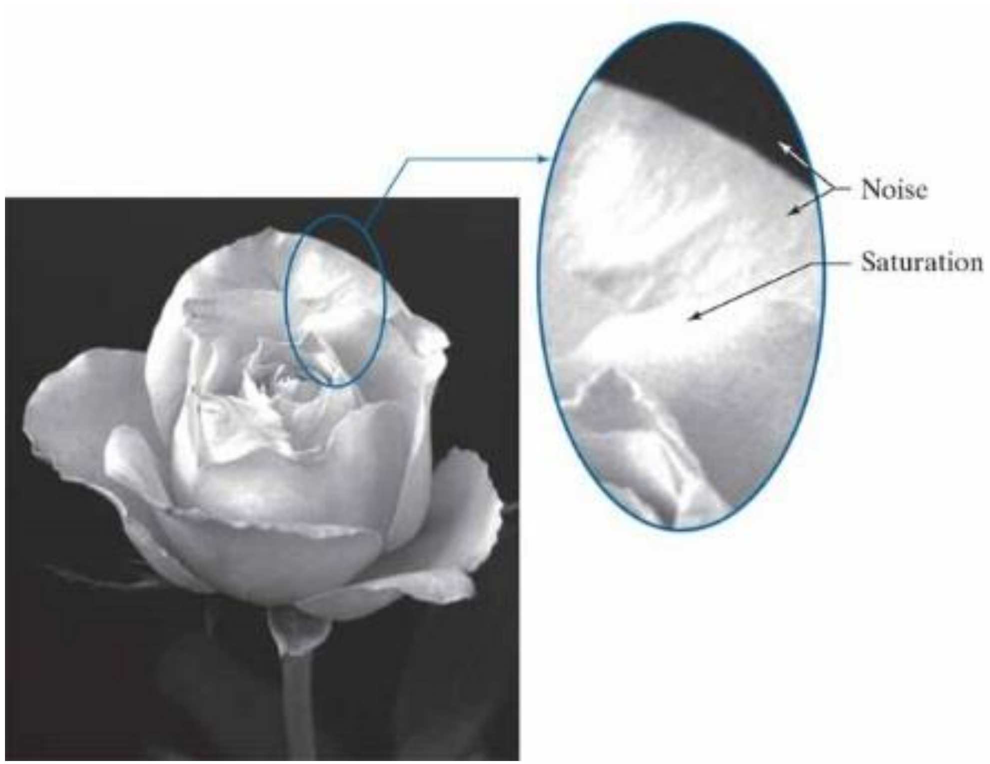
- Discrete intensity interval [0, L-1], $L=2^k$
  - The number b of bits required to store a $M \times N$ digitized image

$$b = M \times N \times k = N^2 k \text{ (when } M=N)$$



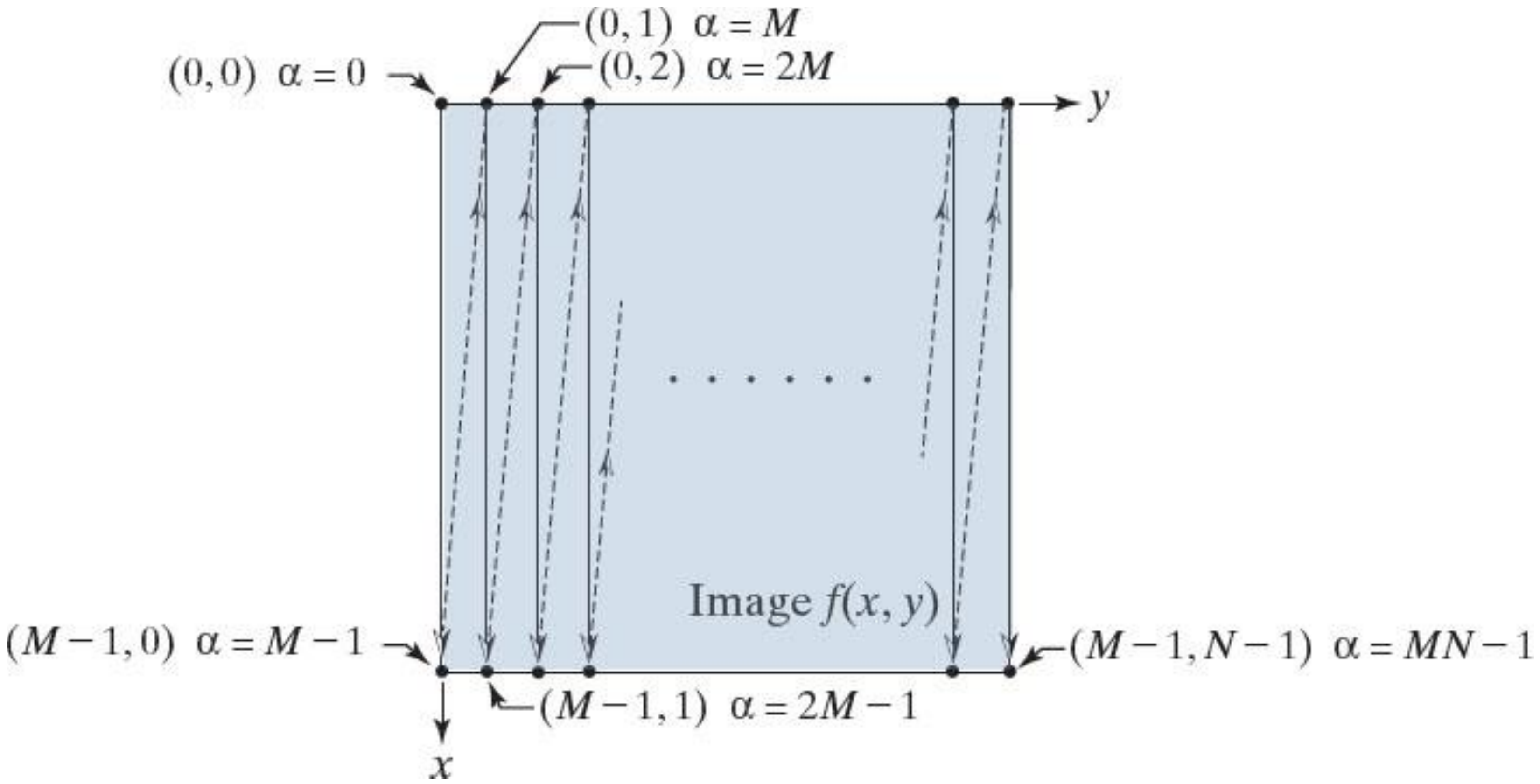
Some attributes of the imaging system/images

- *Dynamic range* of an imaging system: ratio between the maximum and minimum detectable intensity level of the system.
- *Saturation*: highest value beyond which intensity levels are clipped (to a constant value)
- *Noise:* grainy texture pattern
- Contrast: difference in intensity between the highest and lowest intensity level in an image



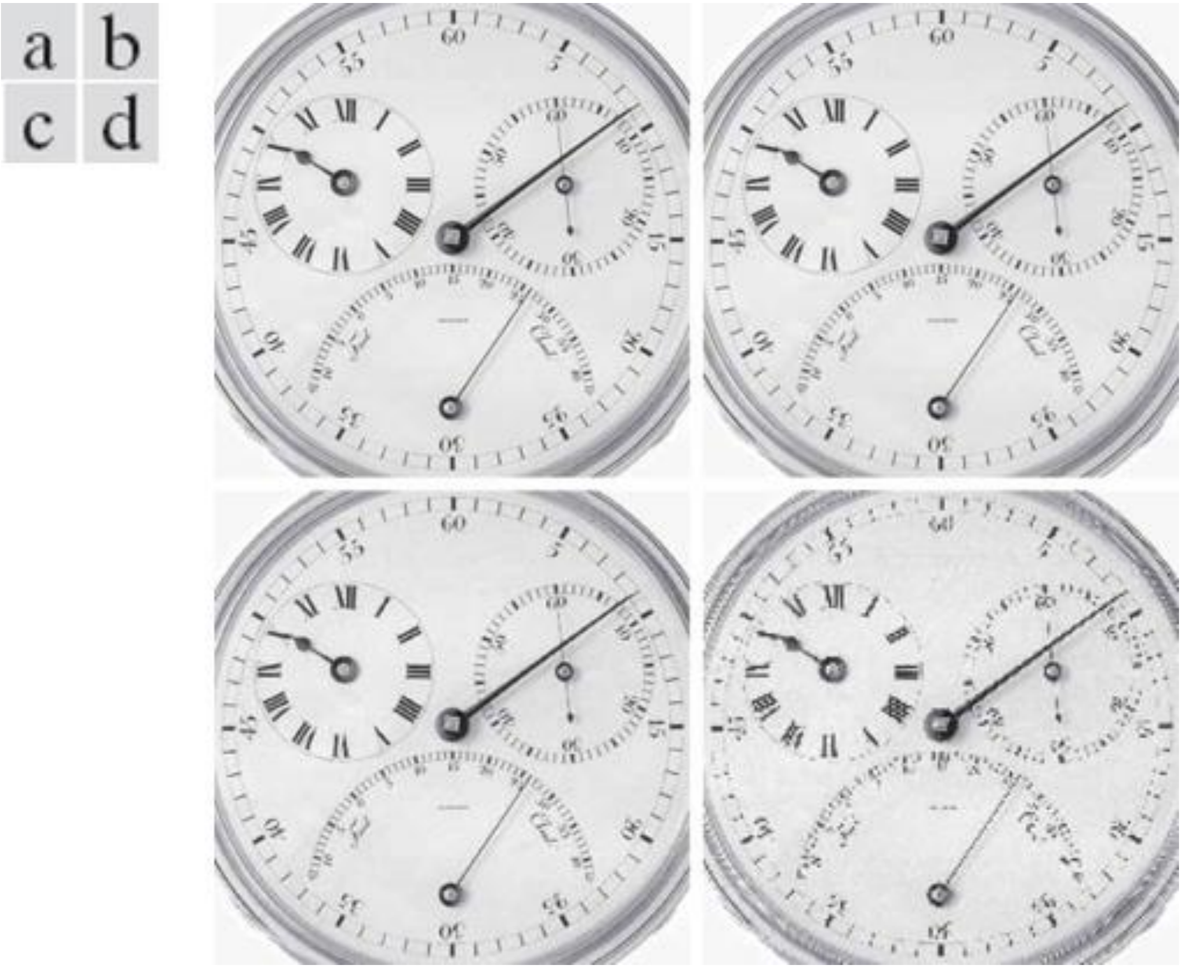
Linear vs. coordinate indexing

- $\alpha = My + x$
  - $x = \alpha \bmod M$
  - $y = (\alpha - x)/M$



Spatial and intensity resolution

- *DPI : dots per inch*
- *Spatial resolution:*
  is the measure of the smallest discernible detail in an image.
- Relates number of pixels to spatial dimension of the image.
- High spatial resolution: very detailed image
- Low spatial resolution: poor detailed image



Effects of reducing spatial resolution. The images shown are at: (a) 930 dpi, (b) 300 dpi, (c) 150 dpi, (d) 72 dpi.

Intensity resolution

- *Intensity resolution:* spatial resolution fixed, reduce *k,* the number of intensity levels.
  - [0, L-1]=[0, 2^k]
- Low intensity resolution might result in false contours

a b
c d

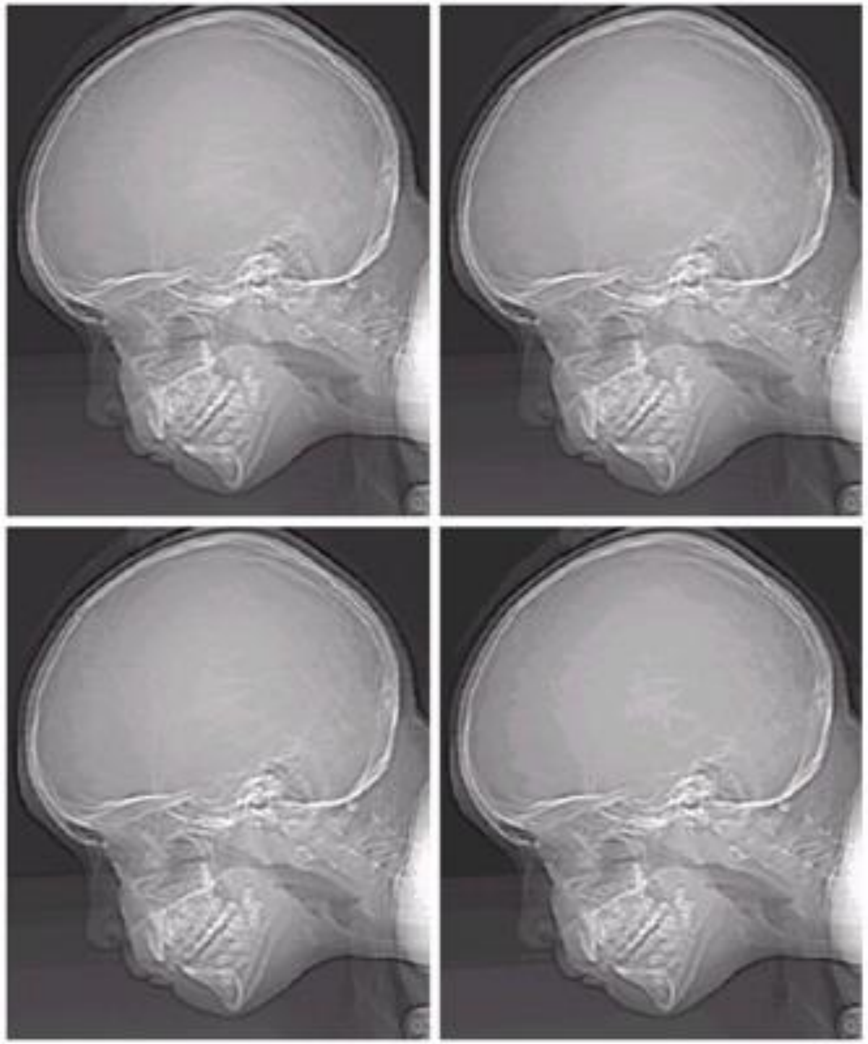


FIGURE 2.21 (a) 452 × 374, 256-level image. (b)–(d) Image displayed in 128, 64, and 32 gray levels, while keeping the spatial resolution constant.

Intensity resolution

- *Intensity resolution:* spatial resolution fixed, reduce *k,* the number of intensity levels.
  - [0, L-1]=[0, 2^k]
- Low intensity resolution might result in false contours

e f
g h

FIGURE 2.21 *(Continued)* (e)–(h) Image displayed in 16, 8, 4, and 2 gray levels. (Original courtesy of Dr. David R. Pickens, Department of Radiology & Radiological Sciences, Vanderbilt University Medical Center.)

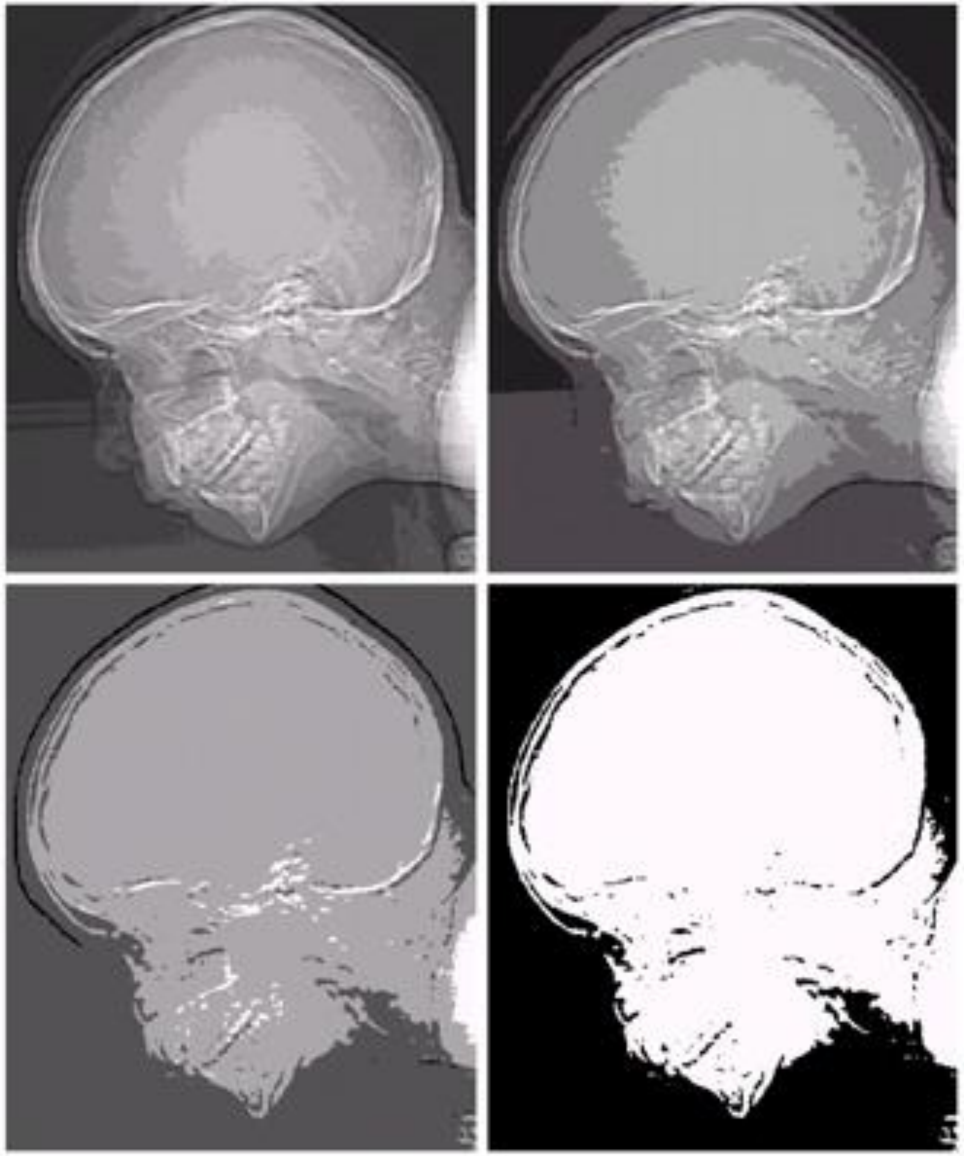


Image with various levels of detail

- What are the optimal values of N and k ?
  - No general rule, might depend on the level of detail of the image

a b c

(a) Image with a low level of detail. (b) Image with a medium level of detail. (c) Image with a relatively large amount of detail.

Image interpolation

- Is used for zooming, shrinking, rotating, geometric corrections, etc. (resampling methods)
- *Interpolation*: estimate values at unknown locations using known data values.

a b c

(a) Image reduced to 72 dpi and zoomed back to its original 930 dpi using nearest neighbor interpolation. (b) Image reduced to 72 dpi and zoomed using bilinear interpolation. (c) Same as (b) but using bicubic interpolation.

Image interpolation

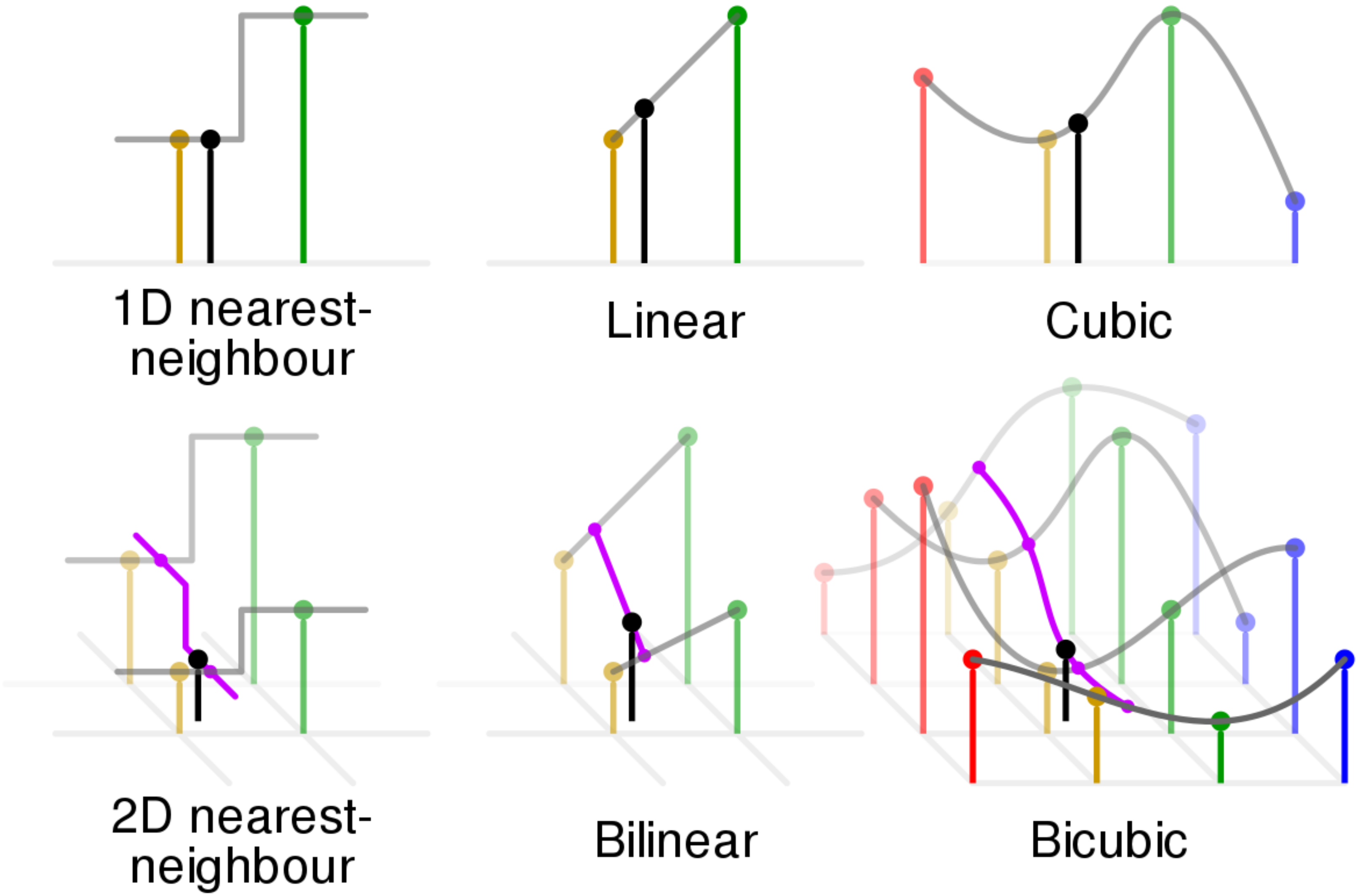
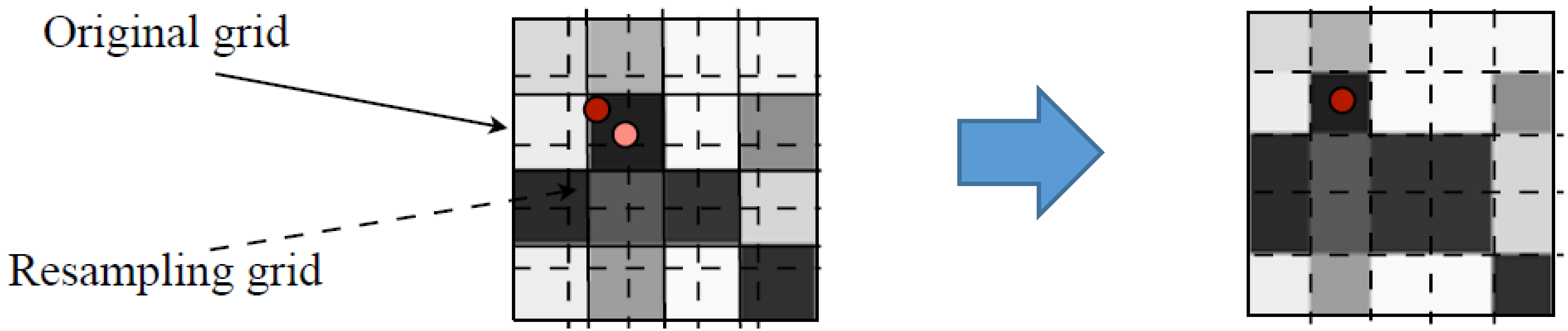
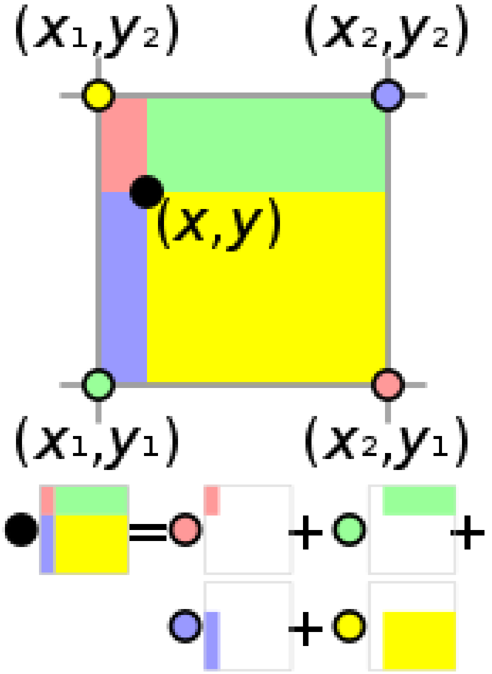


Image interpolation

- Nearest neighbor
  - find the closest pixel in the original grid and assign its value



- *Bilinear interpolation:* $v(x, y) = ax + by + cxy + d$
  - the coefficients *a, b, c, d* are computed using 4 neighbors



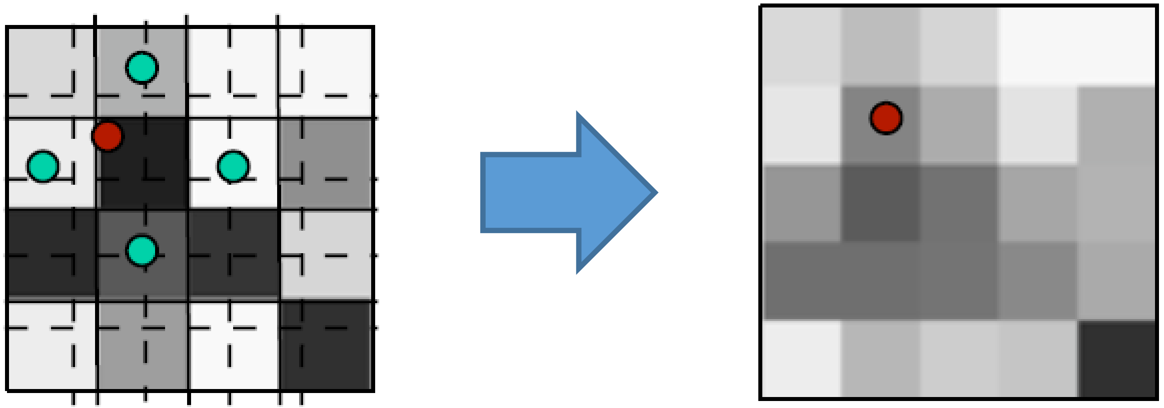
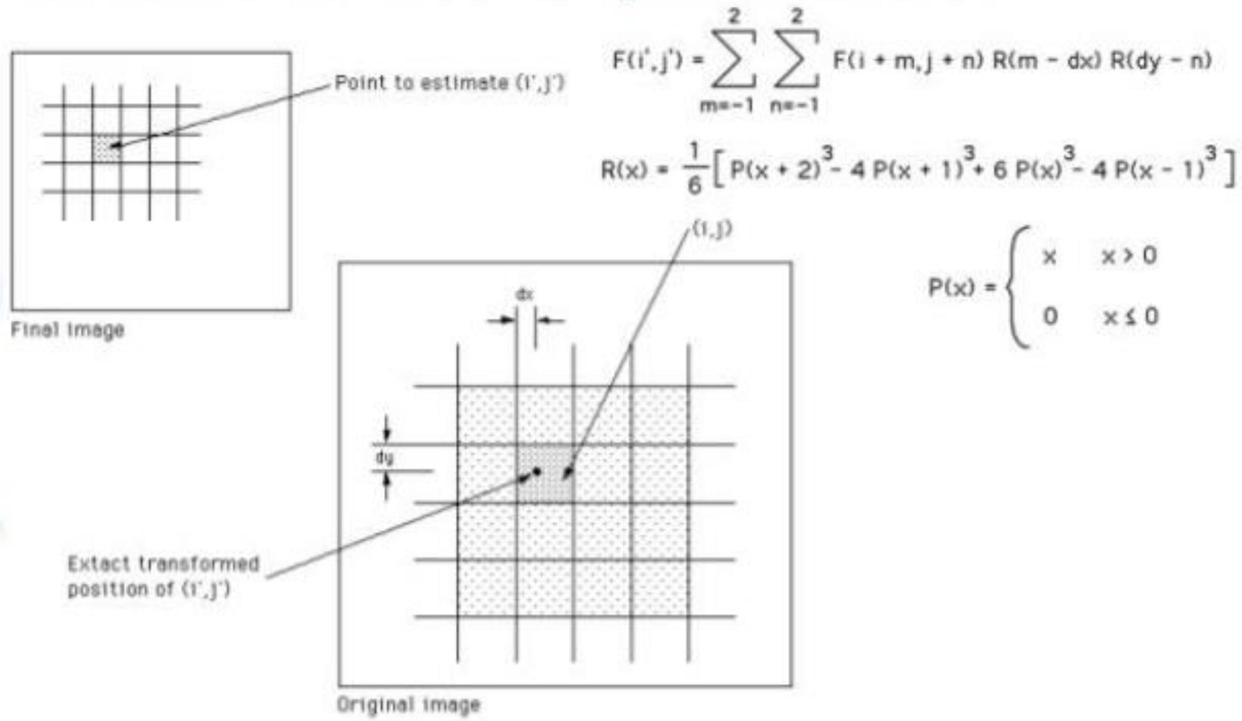
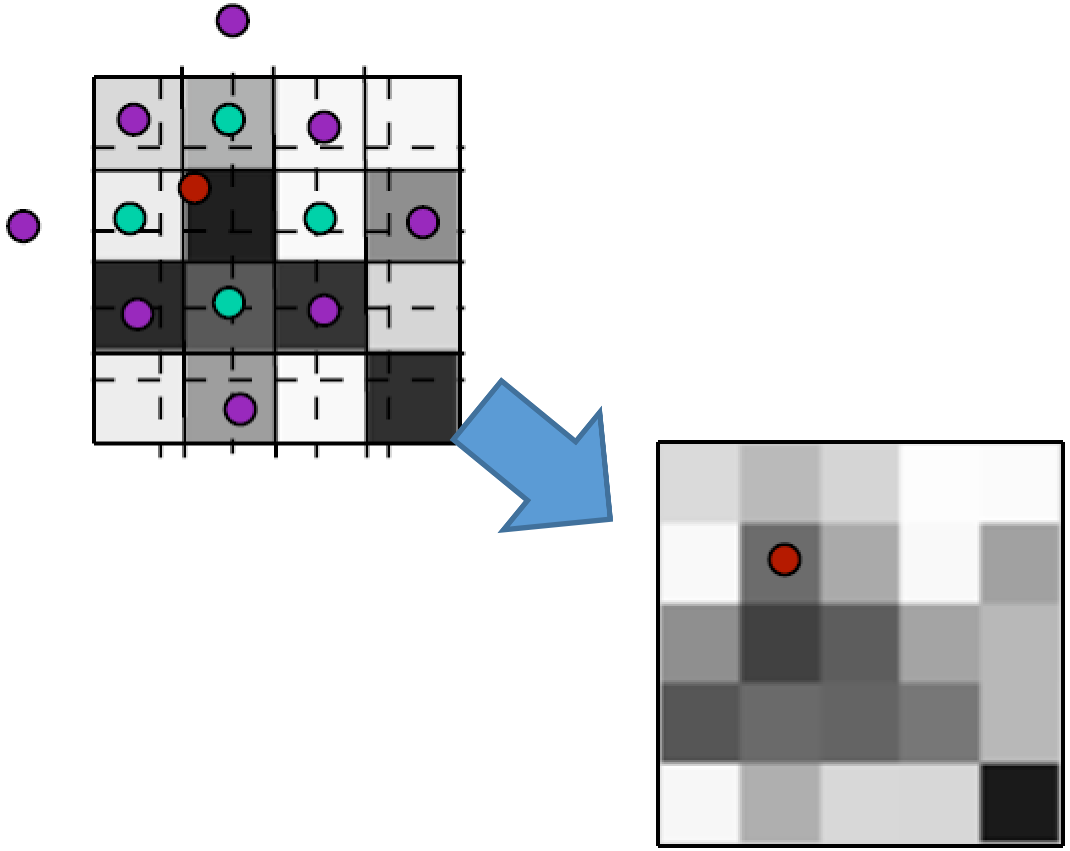


Image interpolation

- *Bicubic interpolation:* $v(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{i,j} x^i y^j$
- the coefficients $a_{i,j}$ are computed using 16 nearest neighbors





Reading image from file

```python
import argparse

import cv2

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
params = parser.parse_args()

img = cv2.imread(params.path)

# Check if image was successfully read.
assert img is not None

print('read {}'.format(params.path))
print('shape:', img.shape)
print('dtype:', img.dtype)

img = cv2.imread(params.path, cv2.IMREAD_GRAYSCALE)
assert img is not None
print('read {} as grayscale'.format(params.path))
print('shape:', img.shape)
print('dtype:', img.dtype)
```

OpenCV C++ Mat structure

```cpp
class CV_EXPORTS Mat
{
public:
    // ... a lot of methods ...
    ...

    /*! includes several bit-fields:
         - the magic signature
         - continuity flag
         - depth
         - number of channels
     */
    int flags;
    //! the array dimensionality, >= 2
    int dims;
    //! the number of rows and columns or (-1, -1) when the array has more tha
    int rows, cols;
    //! pointer to the data
    uchar* data;

    //! pointer to the reference counter;
    // when array points to user-allocated data, the pointer is NULL
    int* refcount;

    // other members
    ...
};
```

```cpp
for (int y = 0; y < height; y++) {

        for (int x = 0; x < width; x++) {

                uchar b = img_color.at<Vec3b>(y, x)[0];
                uchar g = img_color.at<Vec3b>(y, x)[1];
                uchar r = img_color.at<Vec3b>(y, x)[2];

                img_grayscale.at<uchar>(y, x) = (r + g + b) / 3.0;
        }
}
```

```cpp
for (int y = 0; y < height; y++) {

        uchar *data_output = img_grayscale.data;


        for (int x = 0; x < width; x++) {

                uchar b = data_input[y * width * 3 + x * 3];
                uchar g = data_input[y * width * 3 + x * 3 + 1];
                uchar r = data_input[y * width * 3 + x * 3 + 2];


                data_output[width * y + x] = (r + g + b) / 3.0;
        }
}
```

OpenCV numpy.ndarray structure

```
>>> import numpy as np
>>> import cv2 as cv

>>> img = cv.imread('messi5.jpg')
```

You can access a pixel value by its row and column coordinates. For BGR image, it returns an array of Blue, Green, Red values. For grayscale image, just corresponding intensity is returned.

```
>>> px = img[100,100]
>>> print( px )
[157 166 200]

# accessing only blue pixel
>>> blue = img[100,100,0]
>>> print( blue )
157
```

You can modify the pixel values the same way.

```
>>> img[100,100] = [255,255,255]
>>> print( img[100,100] )
[255 255 255]
```

OpenCV numpy.ndarray structure

```
# accessing RED value
>>> img.item(10,10,2)
59

# modifying RED value
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

```
>>> print( img.shape )
(342, 548, 3)
```

Note

If an image is grayscale, the tuple returned contains or
grayscale or color.

Total number of pixels is accessed by `img.size`:

```
>>> print( img.size )
562248
```

Image datatype is obtained by `img.dtype`:

```
>>> print( img.dtype )
uint8
```

```
>>> ball = img[280:340, 330:390]
>>> img[273:333, 100:160] = ball
```

Check the results below:

image

Resizing, Flipping

```python
import argparse
import cv2

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
params = parser.parse_args()
img = cv2.imread(params.path)
print('original image shape:', img.shape)

width, height = 128, 256
resized_img = cv2.resize(img, (width, height))
print('resized to 128x256 image shape:', resized_img.shape)

w_mult, h_mult = 0.25, 0.5
resized_img = cv2.resize(img, (0, 0), resized_img, w_mult, h_mult)
print('image shape:', resized_img.shape)

w_mult, h_mult = 2, 4
resized_img = cv2.resize(img, (0, 0), resized_img, w_mult, h_mult, cv2.INTER_NEAREST)
print('image shape:', resized_img.shape)

img_flip_along_x = cv2.flip(img, 0)
img_flip_along_x_along_y = cv2.flip(img_flip_along_x, 1)
img_flipped_xy = cv2.flip(img, -1)

# check that sequential flips around x and y equal to simultaneous x-y flip
assert img_flipped_xy.all() == img_flip_along_x_along_y.all()
```

`cv2.resize`*(img, dsize, fx, fy, interpolation)*

Parameters:

- **img** – Image
- **dsize** – Manual Size. 가로, 세로 형태의 tuple(ex; (100,200))
- **fx** – 가로 사이즈의 배수. 2배로 크게하려면 2. 반으로 줄이려면 0.5
- **fy** – 세로 사이즈의 배수
- **interpolation** – 보간법

- 0, for flipping the image around the x-axis (vertical flipping);
- > 0 for flipping around the y-axis (horizontal flipping);
- < 0 for flipping around both axes.

Saving image using lossy and lossless compression

```
import argparse
import cv2

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
parser.add_argument('--out_png', default='../data/Lena_compressed.png',
                    help='Output image path for lossless result.')
parser.add_argument('--out_jpg', default='../data/Lena_compressed.jpg',
                    help='Output image path for lossy result.')
params = parser.parse_args()
img = cv2.imread(params.path)

# save image with lower compression - bigger file size but faster decoding
cv2.imwrite(params.out_png, img, [cv2.IMWRITE_PNG_COMPRESSION, 0])

# check that image saved and loaded again image is the same as original one
saved_img = cv2.imread(params.out_png)
assert saved_img.all() == img.all()

# save image with lower quality - smaller file size
cv2.imwrite(params.out_jpg, img, [cv2.IMWRITE_JPEG_QUALITY, 0])
```

Showing image in OpenCV window

```
import argparse
import cv2

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
parser.add_argument('--iter', default=50, help='Downsampling-upsampling iterations number.')
params = parser.parse_args()
orig = cv2.imread(params.path)
orig_size = orig.shape[0:2]

cv2.imshow("Original image", orig)
cv2.waitKey(2000)

resized = orig

for i in range(params.iter):
    resized = cv2.resize(cv2.resize(resized, (256, 256)), orig_size)
    cv2.imshow("downsized&restored", resized)
    cv2.waitKey(100)

cv2.destroyWindow("downsized&restored")

cv2.namedWindow("original", cv2.WINDOW_NORMAL)
cv2.namedWindow("result")
cv2.imshow("original", orig)
cv2.imshow("result", resized)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Scrollbars in OpenCV window

```
import cv2, numpy as np

cv2.namedWindow('window')

fill_val = np.array([255, 255, 255], np.uint8)


def trackbar_callback(idx, value):
    fill_val[idx] = value


cv2.createTrackbar('R', 'window', 255, 255, lambda v: trackbar_callback(2, v))
cv2.createTrackbar('G', 'window', 255, 255, lambda v: trackbar_callback(1, v))
cv2.createTrackbar('B', 'window', 255, 255, lambda v: trackbar_callback(0, v))

while True:
    image = np.full((500, 500, 3), fill_val)
    cv2.imshow('window', image)
    key = cv2.waitKey(3)
    if key == 27:
        break

cv2.destroyAllWindows()
```

Drawing 2D primitives

```python
import argparse
import cv2, random

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
params = parser.parse_args()
image = cv2.imread(params.path)
w, h = image.shape[1], image.shape[0]


def rand_pt(mult=1.):
    return (random.randrange(int(w * mult)),
            random.randrange(int(h * mult)))


cv2.circle(image, rand_pt(), 40, (255, 0, 0))
cv2.circle(image, rand_pt(), 5, (255, 0, 0), cv2.FILLED)
cv2.circle(image, rand_pt(), 40, (255, 85, 85), 2)
cv2.circle(image, rand_pt(), 40, (255, 170, 170), 2, cv2.LINE_AA)
cv2.line(image, rand_pt(), rand_pt(), (0, 255, 0))
cv2.line(image, rand_pt(), rand_pt(), (85, 255, 85), 3)
cv2.line(image, rand_pt(), rand_pt(), (170, 255, 170), 3, cv2.LINE_AA)
cv2.arrowedLine(image, rand_pt(), rand_pt(), (0, 0, 255), 3, cv2.LINE_AA)
cv2.rectangle(image, rand_pt(), rand_pt(), (255, 255, 0), 3)
cv2.ellipse(image, rand_pt(), rand_pt(0.3), random.randrange(360), 0, 360, (255, 255, 255), 3)
cv2.putText(image, 'OpenCV', rand_pt(), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 3)

cv2.imshow("result", image)
key = cv2.waitKey(0)
```

`cv2.circle`*(img, center, radian, color, thickness)*

Parameters:

- **img** - 그림을 그릴 이미지
- **center** - 원의 중심 좌표(x, y)
- **radian** - 반지름
- **color** - BGR형태의 Color
- **thickness** - 선의 두께, -1 이면 원 안쪽을 채움

`cv2.line`*(img, start, end, color, thickness)*

Parameters:

- **img** - 그림을 그릴 이미지 파일
- **start** - 시작 좌표(ex; (0,0))
- **end** - 종료 좌표(ex; (500. 500))
- **color** - BGR형태의 Color(ex; (255, 0, 0) -> Blue)
- **thickness** (*int*) – 선의 두께. pixel

`cv2.putText`*(img, text, org, font, fontSacle, color)*

Parameters:

- **img** – image
- **text** - 표시할 문자열
- **org** – 문자열이 표시될 위치. 문자열의 bottom-left corner점
- **font** – font type. CV2.FONT_XXX
- **fontSacle** – Font Size
- **color** – fond color

Handling user input from keyboard

```python
import argparse
import cv2, numpy as np, random

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
params = parser.parse_args()
image = cv2.imread(params.path)
image_to_show = np.copy(image)
w, h = image.shape[1], image.shape[0]


def rand_pt():
    return (random.randrange(w),
            random.randrange(h))

finish = False
while not finish:
    cv2.imshow("result", image_to_show)
    key = cv2.waitKey(0)
    if key == ord('p'):
        for pt in [rand_pt() for _ in range(10)]:
            cv2.circle(image_to_show, pt, 3, (255, 0, 0), -1)
    elif key == ord('l'):
        cv2.line(image_to_show, rand_pt(), rand_pt(), (0, 255, 0), 3)
    elif key == ord('r'):
        cv2.rectangle(image_to_show, rand_pt(), rand_pt(), (0, 0, 255), 3)
    elif key == ord('e'):
        cv2.ellipse(image_to_show, rand_pt(), rand_pt(), random.randrange(360), 0, 360, (255, 255, 0), 3)
    elif key == ord('t'):
        cv2.putText(image_to_show, 'OpenCV', rand_pt(), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 3)
    elif key == ord('c'):
        image_to_show = np.copy(image)
    elif key == 27:
        finish = True
```

Handling user input from mouse

```python
import argparse
import cv2, numpy as np

parser = argparse.ArgumentParser()
parser.add_argument('--path', default='../data/Lena.png', help='Image path.')
params = parser.parse_args()
image = cv2.imread(params.path)
image_to_show = np.copy(image)

mouse_pressed = False
s_x = s_y = e_x = e_y = -1


def mouse_callback(event, x, y, flags, param):
    global image_to_show, s_x, s_y, e_x, e_y, mouse_pressed

    if event == cv2.EVENT_LBUTTONDOWN:
        mouse_pressed = True
        s_x, s_y = x, y
        image_to_show = np.copy(image)

    elif event == cv2.EVENT_MOUSEMOVE:
        if mouse_pressed:
            image_to_show = np.copy(image)
            cv2.rectangle(image_to_show, (s_x, s_y),
                          (x, y), (0, 255, 0), 1)

    elif event == cv2.EVENT_LBUTTONUP:
        mouse_pressed = False
        e_x, e_y = x, y
```

```python
cv2.namedWindow('image')
cv2.setMouseCallback('image', mouse_callback)

while True:
    cv2.imshow('image', image_to_show)
    k = cv2.waitKey(1)

    if k == ord('c'):
        if s_y > e_y:
            s_y, e_y = e_y, s_y
        if s_x > e_x:
            s_x, e_x = e_x, s_x

        if e_y - s_y > 1 and e_x - s_x > 0:
            image = image[s_y:e_y, s_x:e_x]
            image_to_show = np.copy(image)
    elif k == 27:
        break

cv2.destroyAllWindows()
```

Playing frame stream from video

```
import cv2

capture = cv2.VideoCapture('../data/drop.avi')

while True:
    has_frame, frame = capture.read()
    if not has_frame:
        print('Reached end of video')
        break

    cv2.imshow('frame', frame)
    key = cv2.waitKey(500)
    if key == 27:
        print('Pressed Esc')
        break

cv2.destroyAllWindows()
```