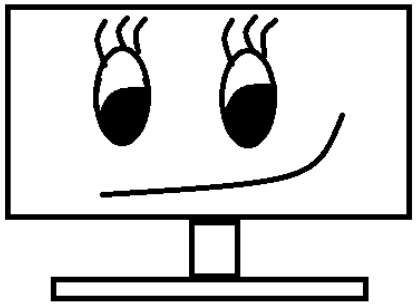
Seneca



Computer Vision

Simple Image Processing and Drawing Tools

Seneca Polytechnic

Credit: Vida Movahedi

Overview

- Simple Image Processing
- Histogram Equalization
- Drawing Tools
- Mouse Callback

Simple Image Processing

What is Computer Vision? (Reminder)

- "The science of **creating a similar capability** [as human vision] in computers and, if possible, **to improve upon it**"
- "Computer vision is the automatic analysis of images and videos by computers in order to gain some understanding of the world … and to **emulate the capabilities of human vision**"
- "The transformation of data from a still or video camera into either a **decision or a new representation** … to **achieve some particular goal**"

Digital Image Processing

- Digital image processing is the use of a digital computer to process digital images through an algorithm **to improve the quality of images**.
- It is a subcategory or a subfield of **digital signal processing**
- Many of the techniques of digital image processing started developing in the 1960s, at Bell Laboratories, the Jet Propulsion Laboratory, Massachusetts Institute of Technology, University of Maryland, and a few other research facilities
- We saw some examples last week: DSP, Image Compression, Color Models

Image Processing

- Not the main focus in Computer Vision
- However, used as a **pre-processing step** for computer vision applications to improve the quality of images, to prepare images for AI/ML algorithms, …
- Examples:
  - Increase brightness
  - Reduce image noise
  - Rotate image
  - Crop image
  - Resize image
  - Filtering and smoothing

Point & Neighborhood Operators [1]

- **Point Operators:**

  The value of each pixel in the **output** depends **only** on the value of the **same pixel** in the input (and possibly **some global information** or some parameters)

  Example: Brightness Adjustment

- **Neighborhood Operators:**

  The value of each pixel in the **output** depends on the value of the **pixel** **and** the value of **its neighbors** in the input

  Example: Smoothing, Blurring, Filtering

Point Operators [1]

Point Operators:

The value of each pixel in the **output** depends **only** on the value of the **same pixel** in the input (and possibly **some global information** or some parameters)

Example: brightness adjustment

Example: Assuming **one color channel** for simplicity

- $I_{\text{out}}(i,j) = f(I_{in}(i,j))$, for some function f

Point Operators [1]

Brightness Adjustment

- **Addition** with a constant, positive or negative – (Brightness changes the **overall** lightness of the image)

$$I_{\text{out}}(i,j) = I_{in}(i,j) + b$$

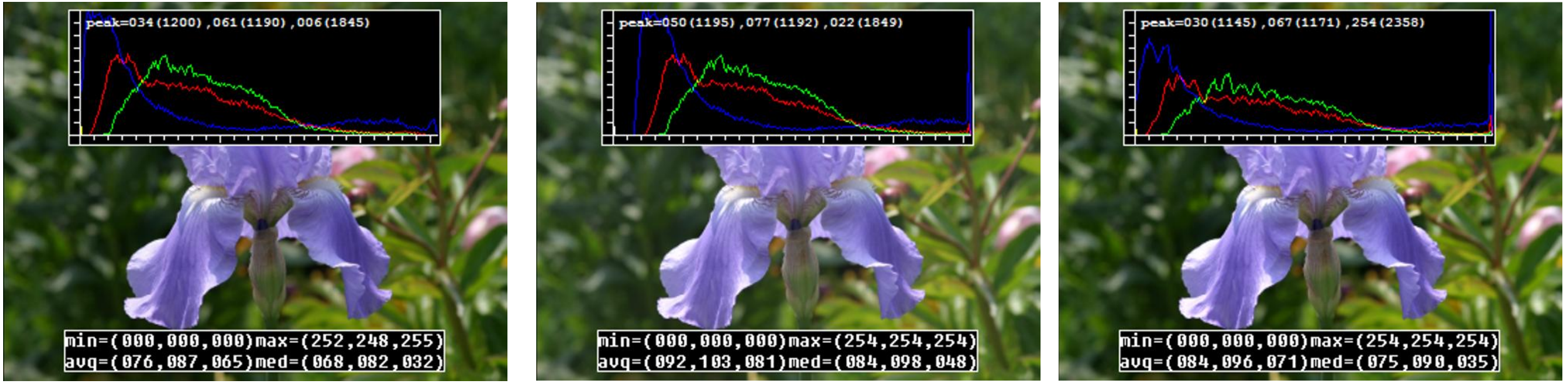
Contrast Adjustment

- **Multiplication** with a constant, smaller than 1 or greater than 1 – (Contrast adjusts the difference between the darkest and lightest colors)

$$I_{\text{out}}(i,j) = aI_{in}(i,j)$$

Contrast & Brightness Adjustment [1]

- Left: original image with its three color (per-channel) histograms;
- Middle: brightness increased (additive offset, b = 16);
- Right: contrast increased (multiplicative gain, a = 1.1);



Addition & Subtraction (Brightness Adjustment)

- Using arithmetic operations in Numpy (does not work when values can go above 255 or lower than 0)
- Use **OpenCV Functions** instead

```
// Make brighter
img2 = cv.add(img,
              np.ones(img.shape, dtype = "uint8") * 50)

// Make darker
img3 = cv.subtract(img,
              np.ones(img.shape, dtype = "uint8") * 100)
```

Multiply & Divide (Contrast Adjustment)

```
// Make bright colors brighter- use scale > 1
img2 = cv.multiply(img,
                np.ones(img.shape, dtype = "uint8"),
                scale = 1.4)

// Make dark colors darker- use scale < 1
img3 = cv.multiply(img,
                np.ones(img.shape, dtype = "uint8"),
                scale =  0.6)
```

Linear Blend (Weighted Image Addition)

- **Two input images**, *img1* and *img2*
- $dst = \alpha.img1 + \beta.img2 + \gamma$

```
img1 = cv.imread("Trillium.jpg")
img2 = cv.imread("flower.jpg")
img2 = cv.resize(img2, (img1.shape[1], img1.shape[0]))
img3 = cv.addWeighted(img1, 0.6 , img2, 0.4, 0)
```

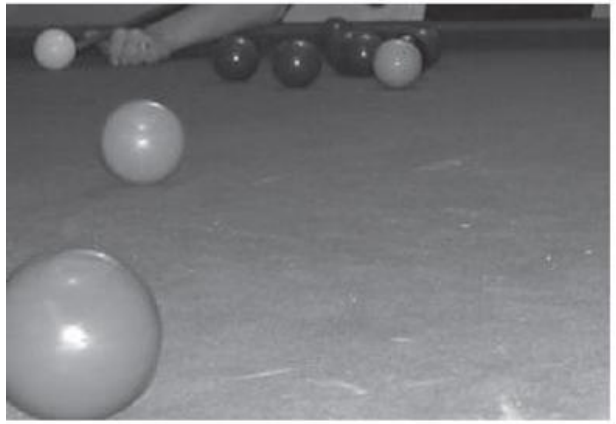
$\propto = 1, \beta = 0$ $\propto = 0.6, \beta = 0.4$ $\propto = 0.4, \beta = 0.6$ $\propto = 0, \beta = 1$

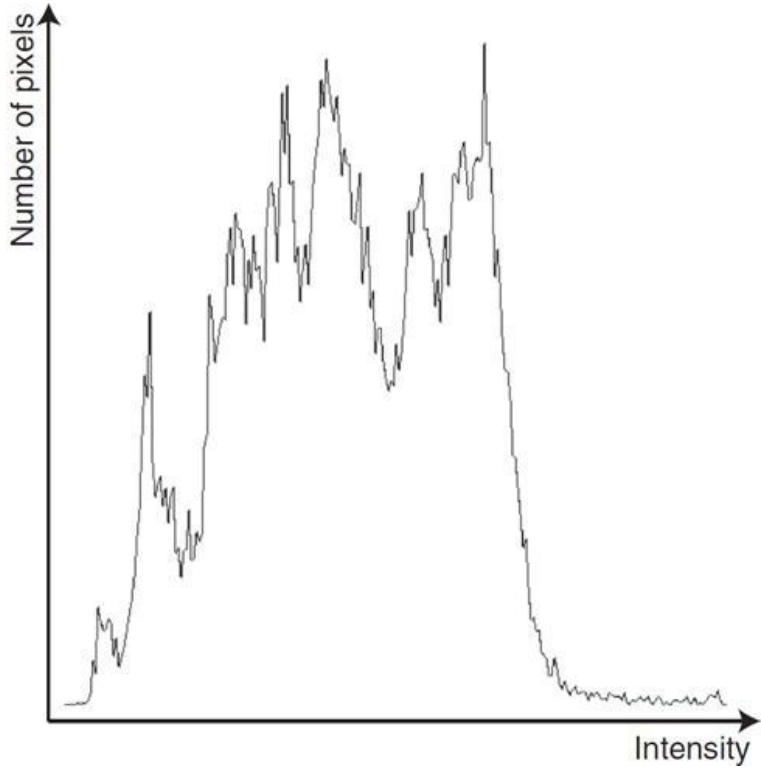
[OpenCV: Arithmetic Operations on Images]

Histogram Equalization

Histogram

- A **visual representation** of an image where the frequency of each image value (brightness/intensity) is determined
- The **count** of the number of pixels at each value (**frequency** of each intensity value)
- X axis: **Intensity** values (e.g. 0 to 255)
- Y axis: **Number of Pixels** in the image having that value





A grey-scale image and the histogram derived from it [3]

- The algorithm to derive such a **histogram h(g)** from a grey-scale image f(i,j) is:

Algorithm 3.1

```
// Initialise the histogram
for (g = 0; g <= 255; g++)
  h(g) = 0
// Compute the histogram
for (i = 0; i < MAX_column;i++)
  for (j = 0; j < MAX_row; j++)
    h(f(i,j))++
```

[3]

- The above code assumes 256 bins (0 to 255), and counts the number of pixels with the corresponding value

Histogram

- Histogram contains **global information** about the image and that information is completely independent of the **position** and the **orientation** of objects in the scene

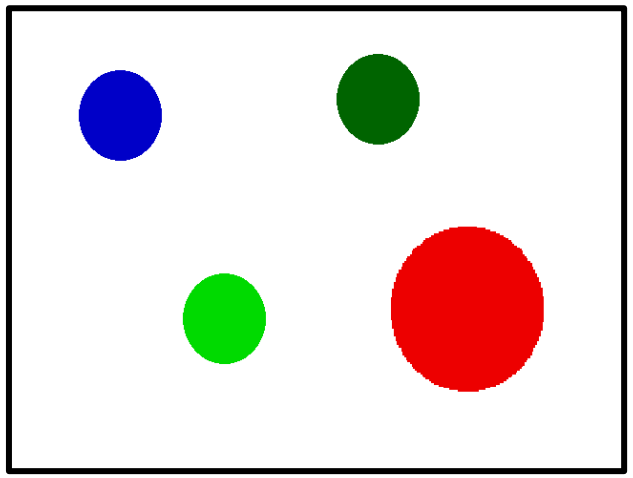
- The number of **bins** is often set to smaller numbers, for example 4

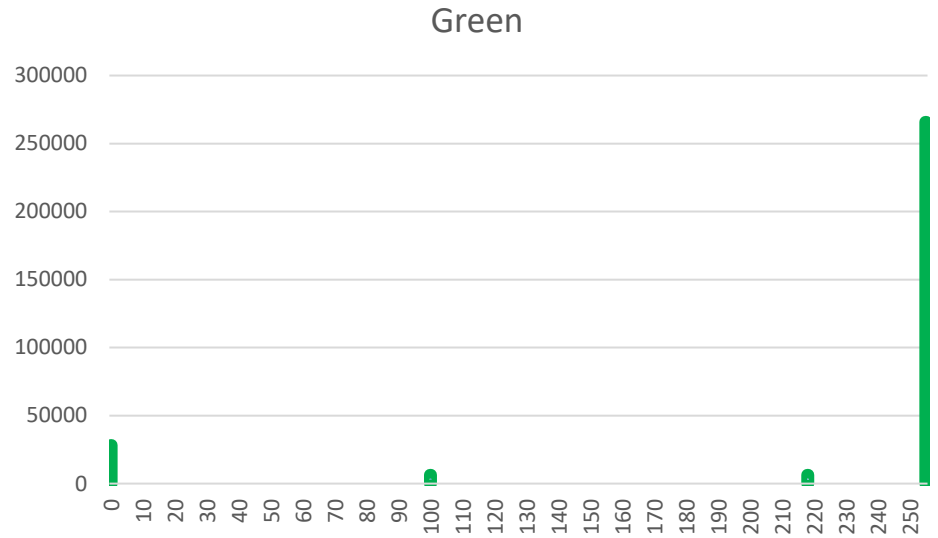
- **Normalized histogram**: Divide all numbers by the total number of pixels in the image, therefore

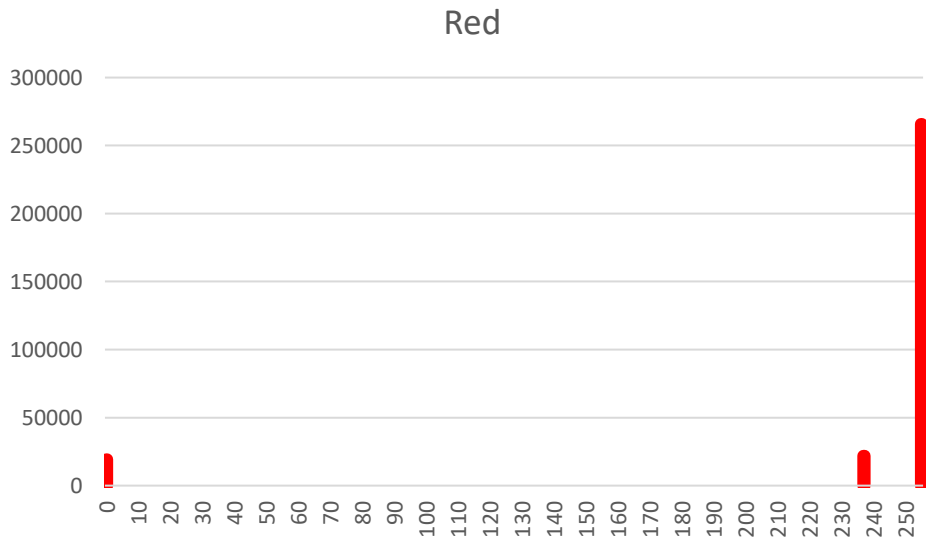
$$\Rightarrow \sum_{g=0}^{255} h(g) = 1$$

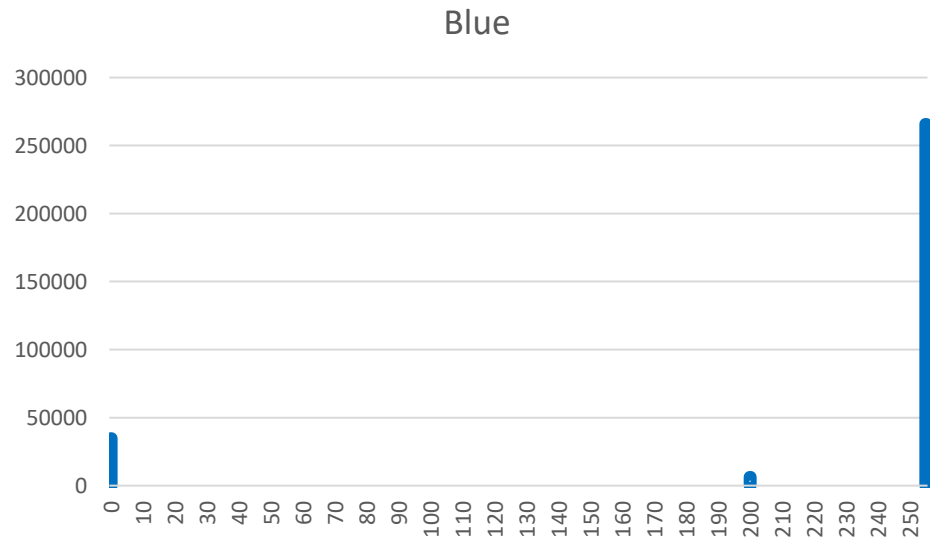
Color Histograms

One Histogram per Color Map or Each Channel

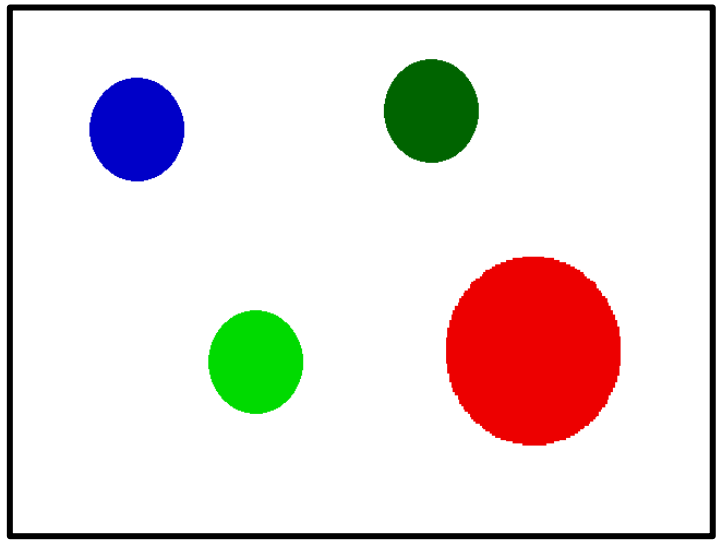


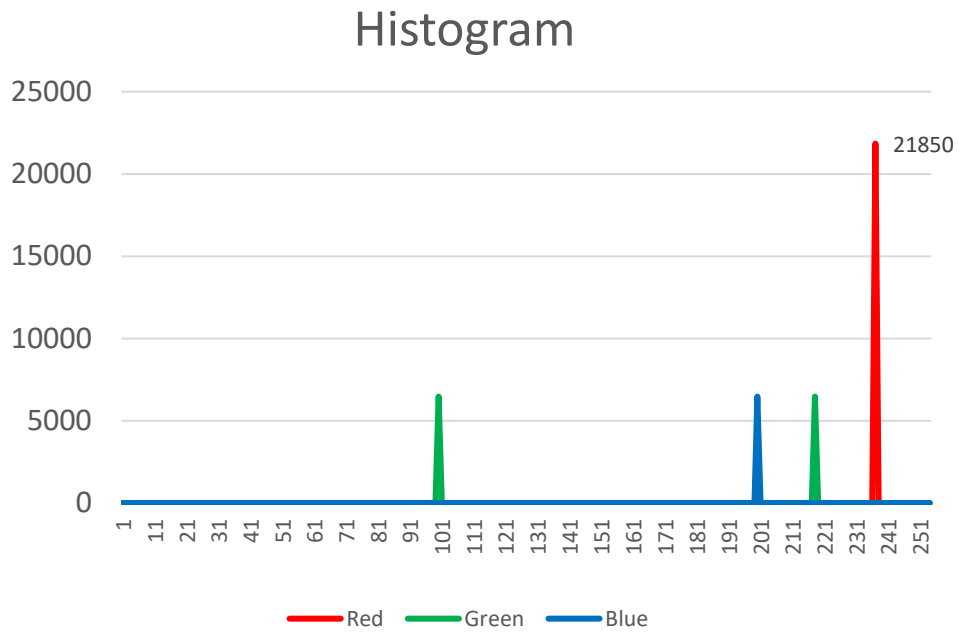






Color Histograms

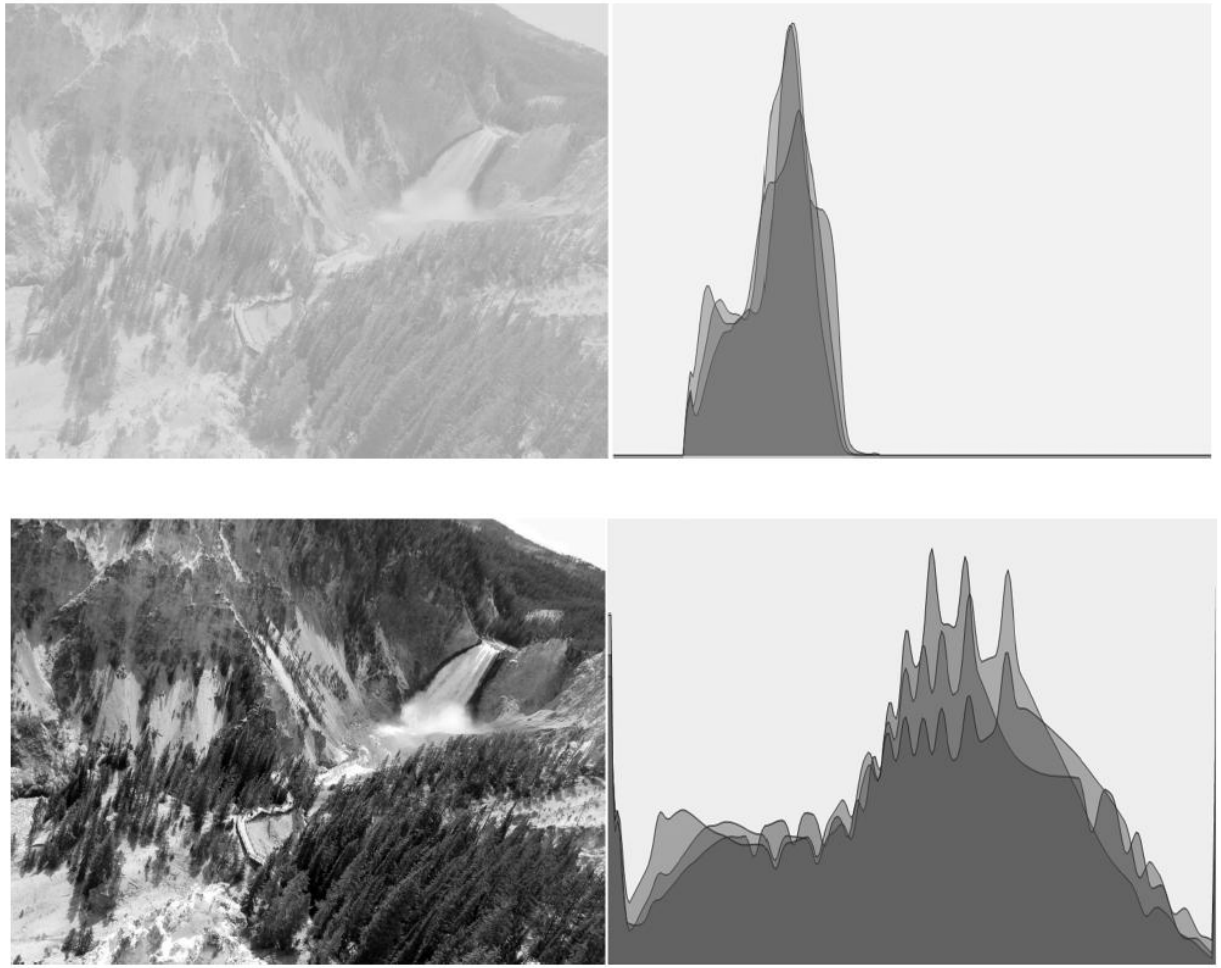




Overlapping the three histograms on top of each other for **comparing** the **size** of the objects with specific colors
(This histogram **does not include** counts for values 0 and 255)

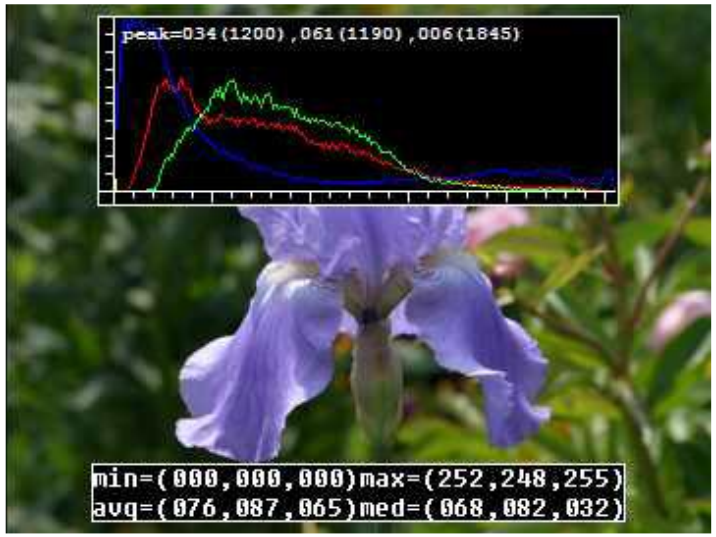
Histogram Equalization [2]

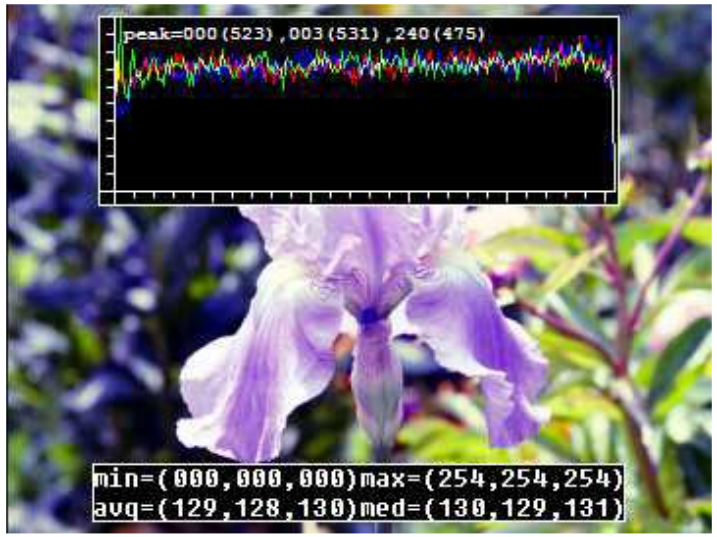
- When there is an **excess of pixels** in a certain range, we use Histogram Equalization.
- A technique for improving the distribution of grey-scales in an image
- Points of some common grey-scales in the input are mapped to **multiple different grey-scales** in the output

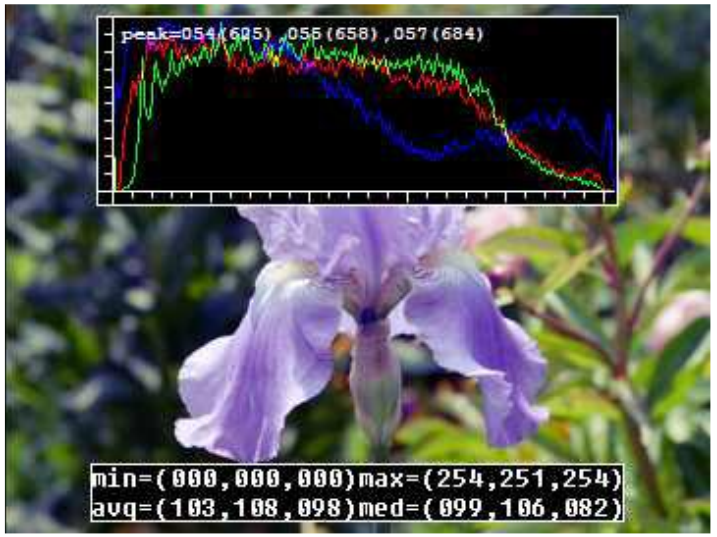


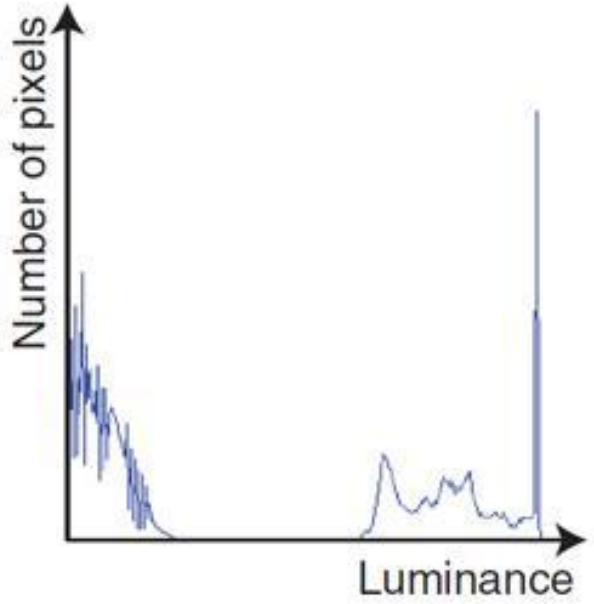
Histogram Equalization [1]

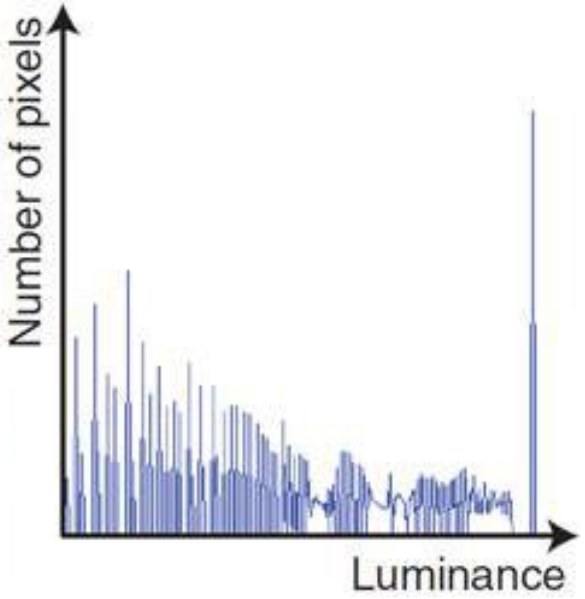
- Mapping the values in a way that results in a **flatter distribution** → The trick to finding such intensity mapping function is the same one to generate random samples from a probability density function.











- When equalizing a color image, we generally only equalize the luminance channel as otherwise the colors can become distorted.
- Already implemented in OpenCV (cv::equalizeHist)[2] for **single channel image**
- Example: A color image and its luminance histogram (top), together with a histogram equalized version of the image and the resultant luminance histogram (bottom)**[3]**

Beyond Histogram Equalization

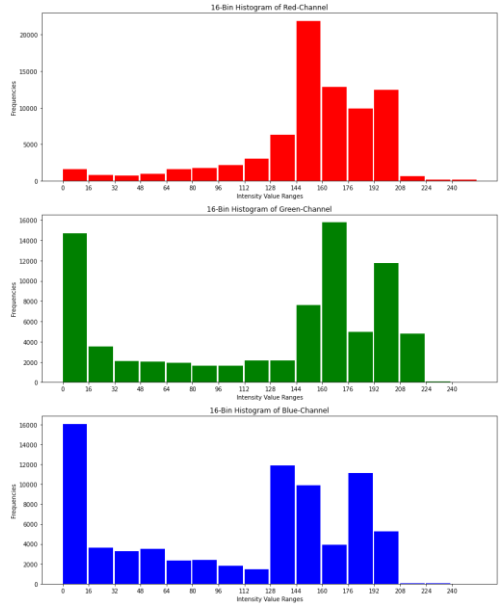
- Histogram is fast and easy to compute.
- Size can easily be normalized so that different image histograms can be compared → The normalized histogram gives the percentage of pixels that have gray tone i.
- Histogram can be used to match color histograms for database query or classification.

Beyond Histogram Equalization

Retrieving images that are similar to a given image or that contain a particular content (Content Based Image Retrieval – CBIR).

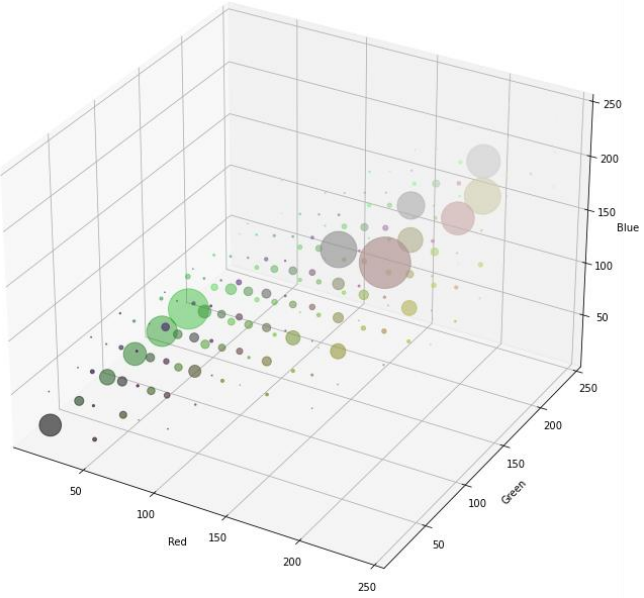
- It is possible to analyze the color distribution in an image by comparing histograms derived from the images.
  - Compare all pixels? Computationally expensive
  - Compare their histograms first; Calculate the histogram difference or distance
- Image search engines do this mostly by using meta-data tags associated with each image

Beyond Histogram Equalization



- Finding the most similar image in the database involves identifying the 3D color histogram that best matches the 3D color histogram of the query image.

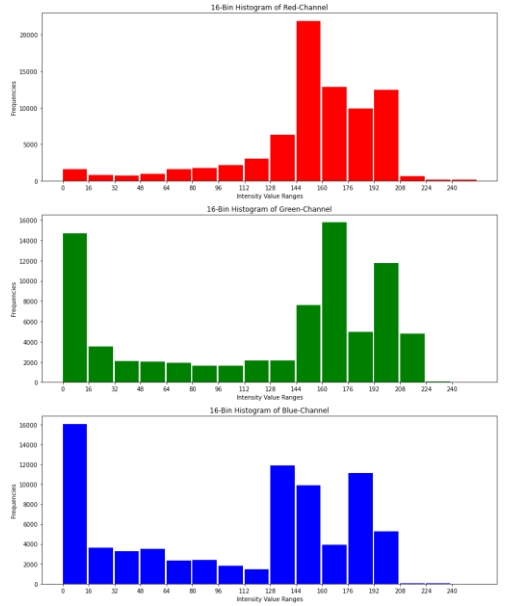
- In multidimensional histograms the bins are not 1-dimensional but e.g., 3-dimensional for 3-channel images

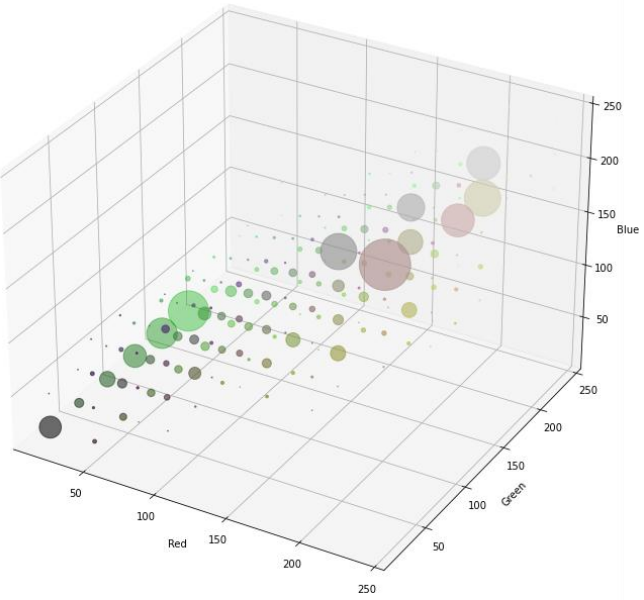


Slide Credit: Johannes Maucher

Beyond Histogram Equalization

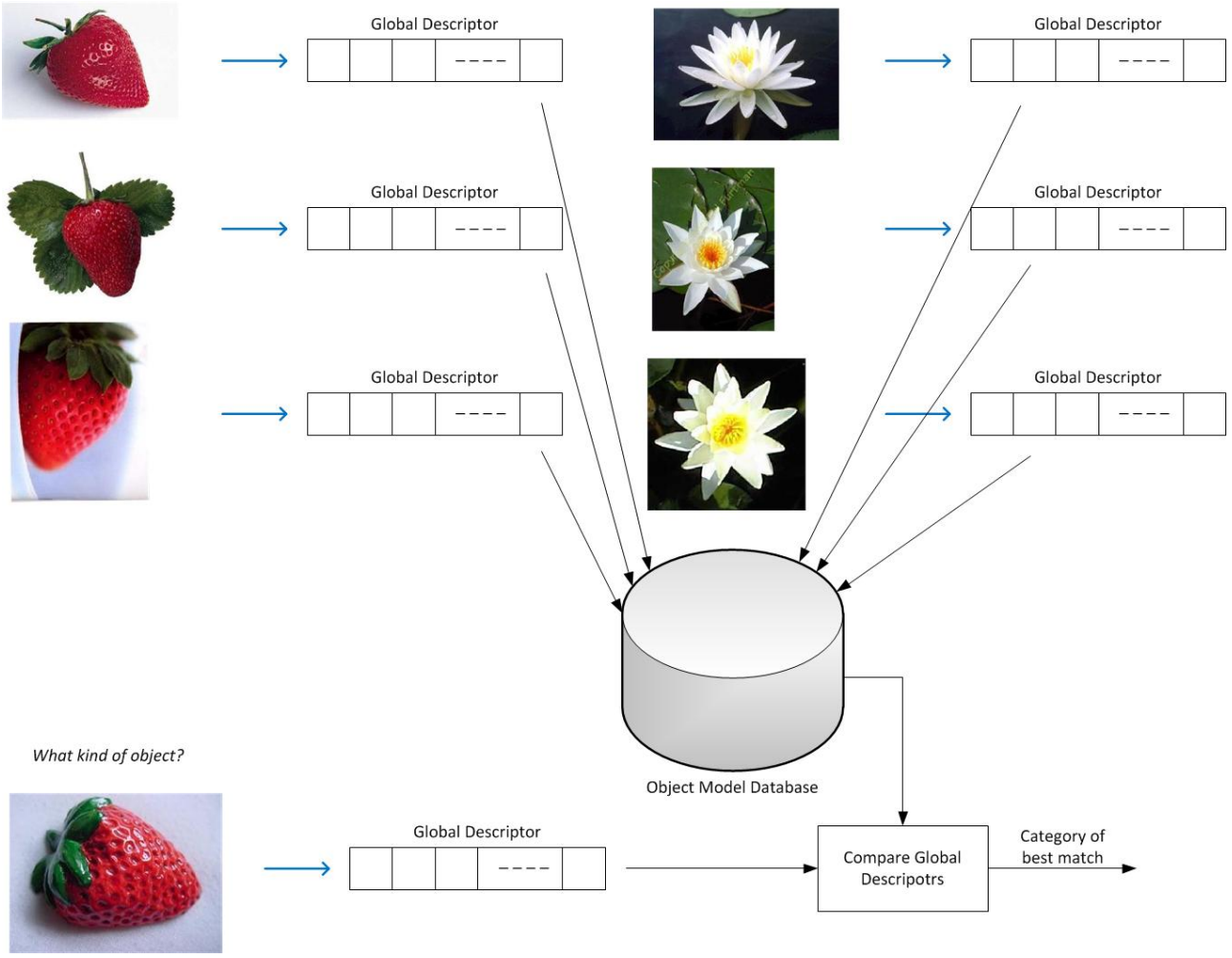
- For an 8-bin histogram, the 3D histogram will have 512 bins
  - The bin [0,0,0] counts the number of pixels in the image, for which the values of the red, green and blue channel are in the range between 0 and 15 (inclusive).
  - The bin [7,7,7] counts the number of pixels in the image, for which the values of the red, green and blue channel are in the range between 240 and 255 (inclusive).





Slide Credit: Johannes Maucher

Beyond Histogram Equalization



Slide Credit: Johannes Maucher

Histogram Comparison [3]

Compare their histograms → Calculate the histogram similarity or difference/distance

- $D_1 = \sum_i \left(h_1(i) - h_2(i)\right)$

Other **distance/similarity functions** for comparing histograms exist that perform better than the above function such as correlation, Chi-Square, intersection, Earth Mover's Distance, …

Best matching images (based on histogram comparison). The reference image is the one on the top left (which gets a perfect matching score). The metrics shown in red are the **correlation** scores **[3]**

Summary

- Simple image processing can be done by image arithmetic, such as adding, subtracting, multiplying, or dividing by a constant, or adding or subtracting two images.
- A histogram is a visual representation of the count of pixels (frequency) at each intensity or color value. Histogram equalization is a technique for mapping values to a flatter distribution and often results in an improvement to the image.
- Two types of connectivity in an image are: 4-connectivity and 8-connectivity.
- OpenCV has many tools and functions for drawing shapes on an image and listening to mouse events.

References

[1] **Computer Vision: Algorithms and Applications** by R. Szeliski

Computer Vision: Algorithms and Applications, 2nd ed. (szeliski.org)

[2] **Learning OpenCV 3** by A. Kaehler & G. Bradski
Available online via Seneca Libraries: Learning OpenCV 3 : computer Vision in C++ with the OpenCV Library - Seneca (exlibrisgroup.com)

[3] **A Practical Introduction to Computer Vision with OpenCV** by Kenneth Dawson-Howe

Available online via Seneca Libraries: A Practical Introduction to Computer Vision with OpenCV. - Seneca (exlibrisgroup.com)

Readings

Chapter 3 [1]

Chapters 6, 7, 9 & 13 [3]

- [1] **A Practical Introduction to Computer Vision with OpenCV**
  by Kenneth Dawson-Howe
  Available online via Seneca Libraries: A Practical Introduction to Computer Vision with OpenCV. - Seneca (exlibrisgroup.com)

- [2] **Learning OpenCV 4 Computer Vision with Python 3**
  by J. Howse & J. Minichino
  Available online via Seneca Libraries: Learning OpenCV 4 Computer Vision with Python 3 : get to grips with tools, techniques, and algorithms for computer vision and machine learning - Seneca (exlibrisgroup.com)

- [3] **Learning OpenCV 3**
  by A. Kaehler & G. Bradski
  Available online via Seneca Libraries: Learning OpenCV 3 : computer Vision in C++ with the OpenCV Library - Seneca (exlibrisgroup.com)

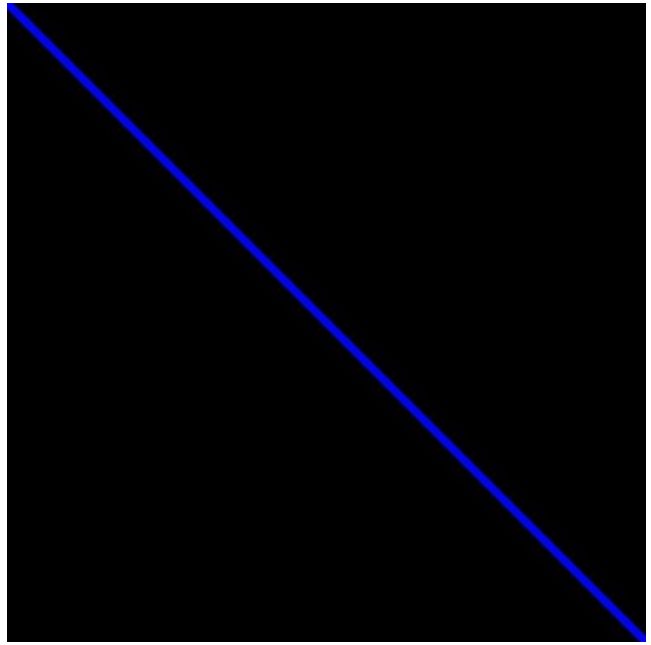
OpenCV: Drawing Tools

Draw a Simple Line

```
import numpy as np
import cv2 as cv

# Create a black image
img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
cv.line(img, (0,0), (511,511), (255,0,0), 5)
```



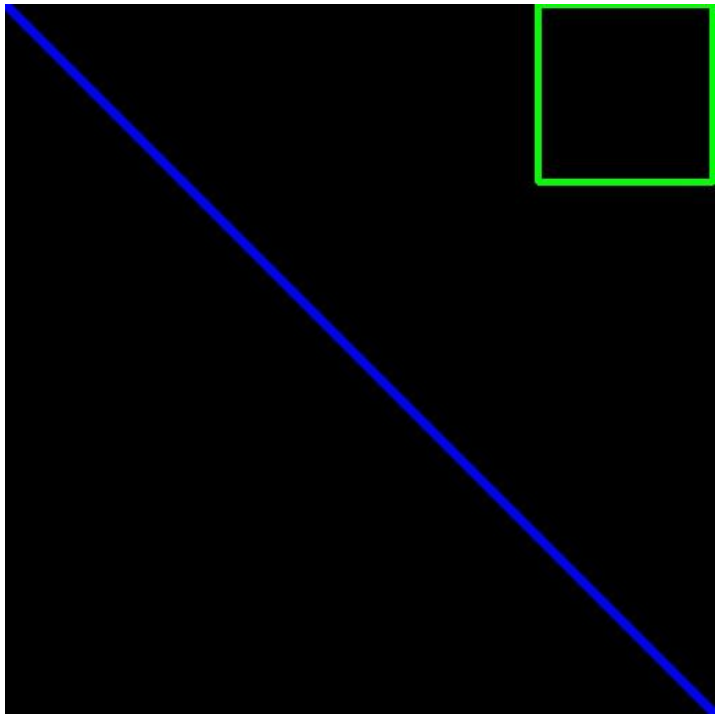
Draw a Simple Line

```
cv.line(
            img,          # Image to be drawn
            pt1,          # Starting point
            pt2,          # Endpoint
            color[,       # Color BGR form
            thickness[,   # Thickness of line
            lineType[,    # Connectedness, 4, 8, or cv.LINE_AA
            shift]]]      # Bits of radius to treat as fraction

        ) ->     img
```

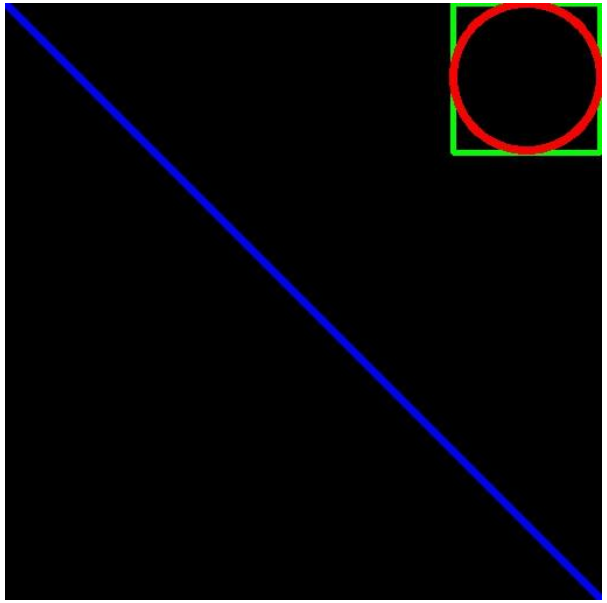
Draw a Simple Rectangle

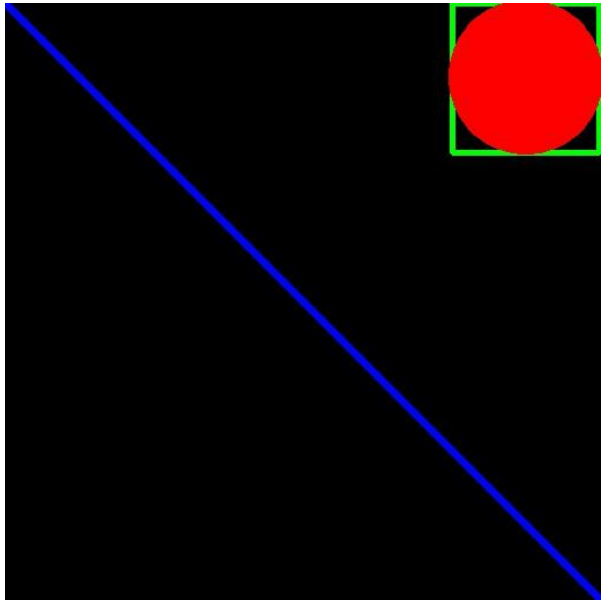
- `cv.rectangle(img,(384,0),(510,128),(0,255,0),3)`



Drawing a circle

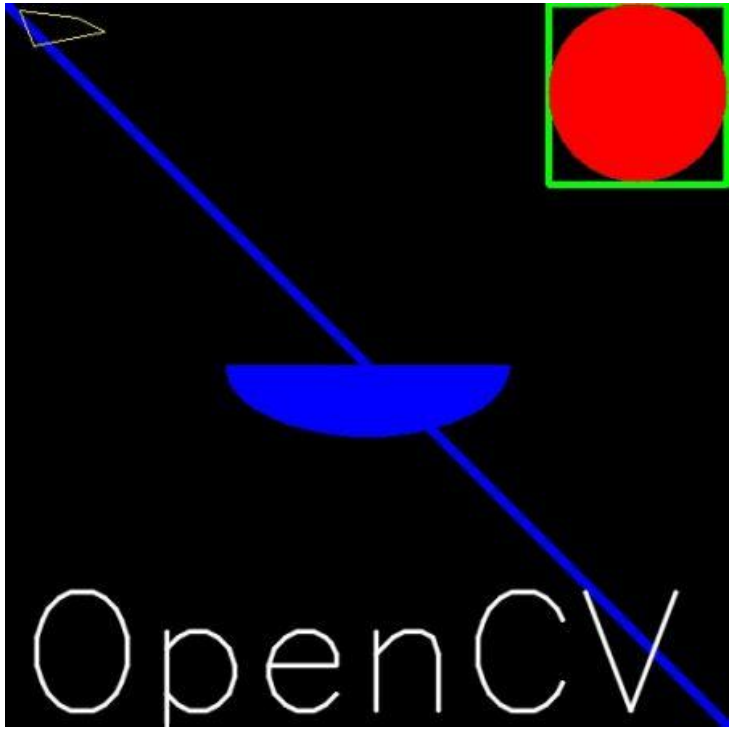
- `cv.circle(img,(447,63), 63, (0,0,255),5)`
- `cv.circle(img,(447,63), 63, (0,0,255), cv.FILLED)`





More Drawing Functions

- See OpenCV: Drawing Functions in OpenCV




```
cv.ellipse(img,(256,256),(100,50),0,0,180,(255,0,0),-1)

pts = np.array([[10,5],[20,30],[70,20],[50,10]], np.int32)
pts = pts.reshape((-1,1,2))
cv.polylines(img,[pts],True,(0,255,255))

font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(img,'OpenCV',(10,500), font, 4,(255,255,255),2,cv.LINE_AA)
```

Draw with Your Mouse

OpenCV: Drawing Functions in OpenCV

Draw a Circle When Double Click

```
import numpy as np
import cv2 as cv

# mouse callback function
def draw_circle(event, x, y, flags, param):
    if event == cv.EVENT_LBUTTONDBLCLK:
        cv.circle(img,(x,y),100,(255,0,0),-1)

# Create a black image, a window and bind the function to window
img = np.zeros((512,512,3), np.uint8)
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)

while(1):
    cv.imshow('image',img)
    if cv.waitKey(20) & 0xFF == 27:
        break
cv.destroyAllWindows()
```

Listen to Mouse Events

1. Define a callback function
   - A function that OpenCV calls whenever a mouse events occurs
   - For example, when the left button is pressed (down) or released (up), when the mouse is moved

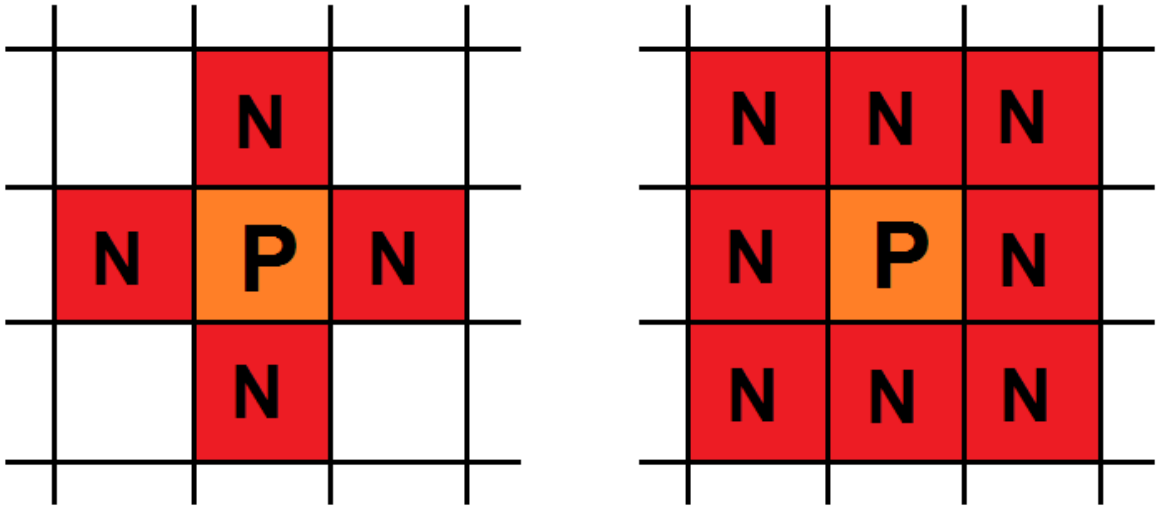
```
void your_mouse_callback(
  int   event,                    // Event type (see Table 9-1)
  int   x,                        // x-location of mouse event
  int   y,                        // y-location of mouse event
  int   flags,                    // More details on event (see Table 9-2)
  void* param                     // Parameters from cv::setMouseCallback()
);
```

2. Register the callback function for the window

```
void cv::setMouseCallback(
  const string&     windowName,          // Handle used to identify window
  cv::MouseCallback your_mouse_callback, // Callback function
  void*             param     = NULL     // Additional parameters for callback fn.
);
```

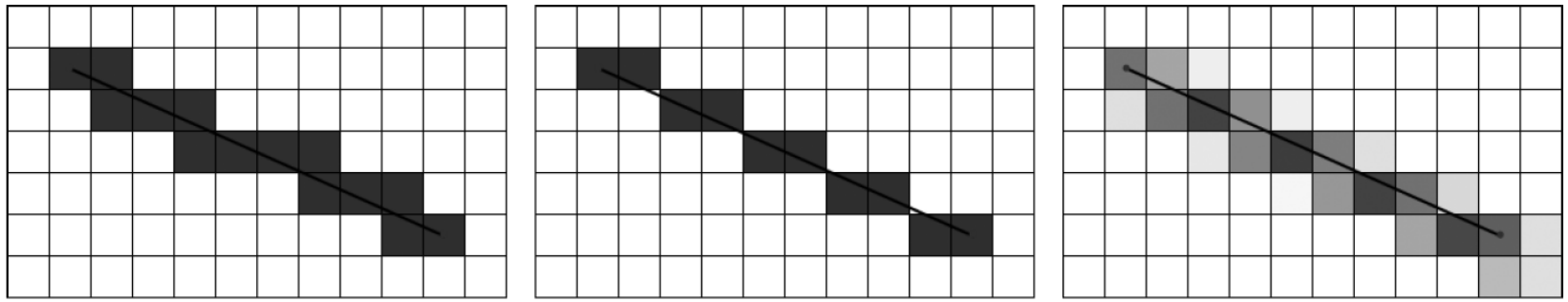
Connectivity

- 4- connectivity: A pixel P is considered connected to 4 neighbors
- 8-connectivity: A pixel P is considered connected to 8 neighbors



Drawing on images [2]

- lineType
  - Integer (4, 8 (default) , or cv::LINE_AA)
  - 8 is the default for lineType



(a) 4-connected Bresenham Algorithm

(b) 8-connected Bresenham Algorithm

(c) Anti-aliased Line With Gaussian Smoothing