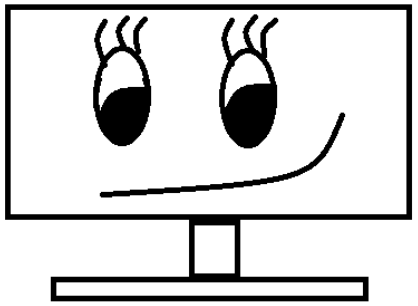
Seneca



CVI620/ DPS920
Introduction to Computer Vision

Simple Image Processing and Drawing tools

Seneca College

Savita Seharawat

Overview

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

Simple Image Processing

Image Processing

- Not the main focus in Computer Vision
- However, used as a pre-processing step for computer vision applications
- Examples:
  - Increase brightness
  - Reduce image noise
  - Rotate image
  - Crop image

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

- 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 or blurring

Pixel Transforms

- Assuming one color channel for simplicity
- $I_{\text{out}}(i,j) = f(I_{in}(i,j))$, for some function f

- Examples:
  - Addition with a constant - Brightness adjustment

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

  - Multiplication with a constant – Contrast adjustment

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

Addition & Subtraction (scalar)

- 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 (scalar)

```
// 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)
```

$\propto = 1, \beta = 0$

$\propto = 0.6, \beta = 0.4$

$\propto = 0.4, \beta = 0.6$

$\propto = 0, \beta = 1$

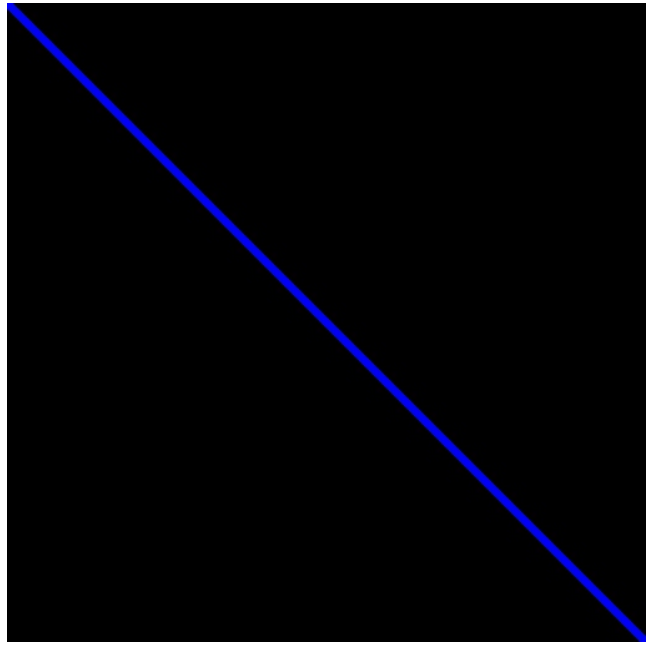
OpenCV: Drawing tools

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)
```



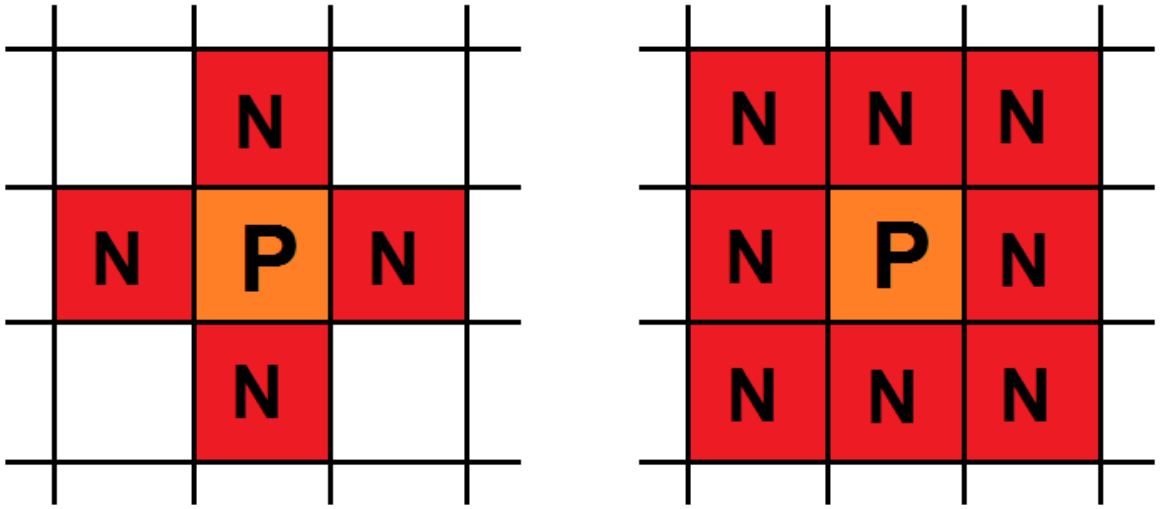
Draw a simple line

```
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
```

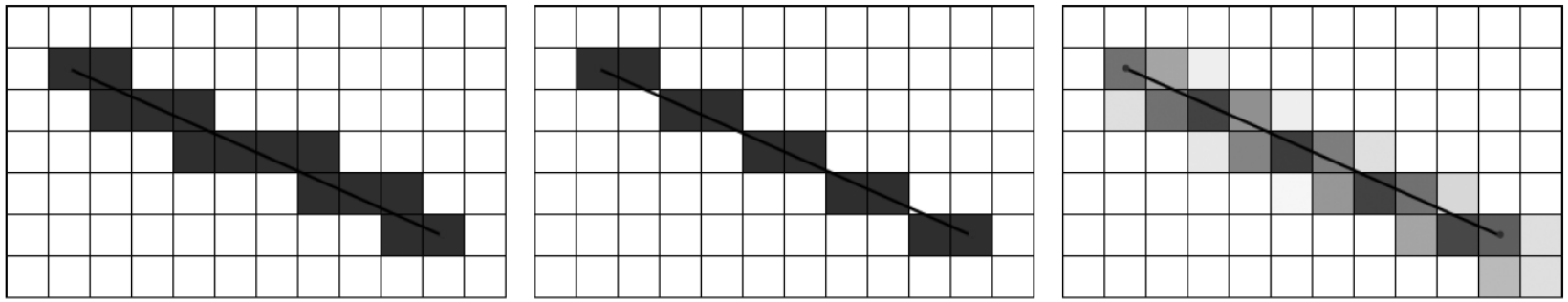
Connectivity

- 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)



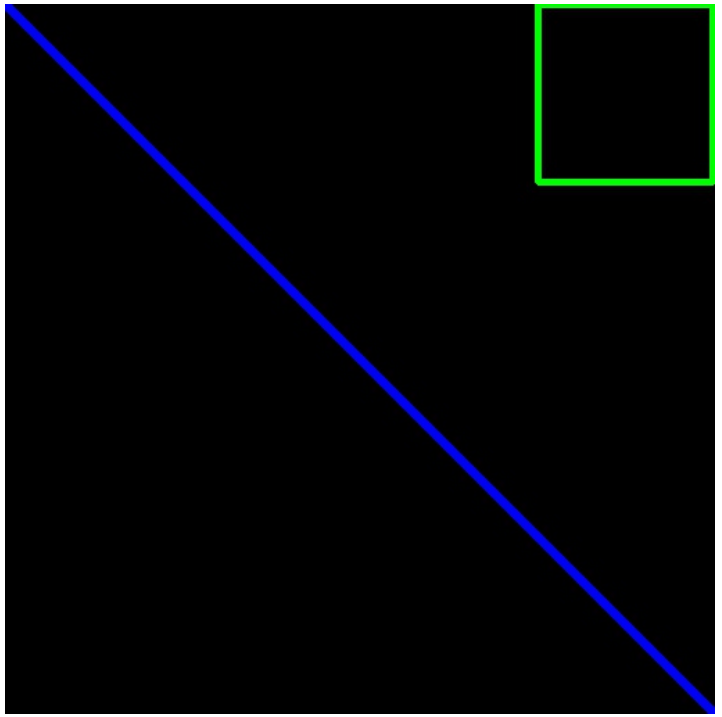
(a) 4-connected Bresenham Algorithm

(b) 8-connected Bresenham Algorithm

(c) Anti-aliased Line With Gaussian Smoothing

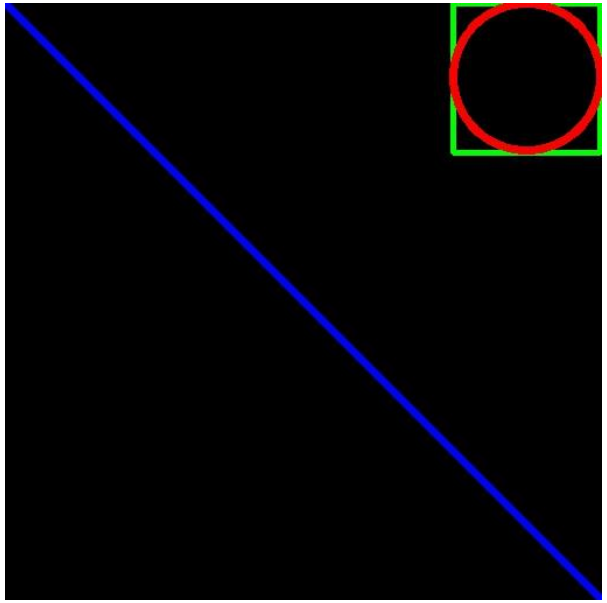
Draw a simple rectangle

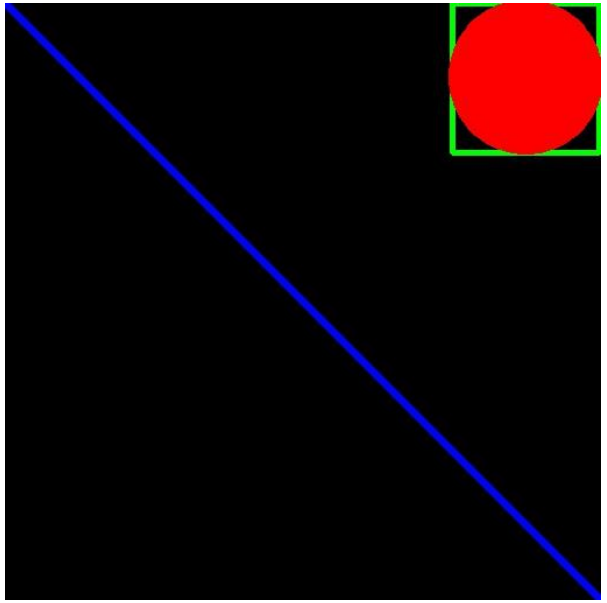
- `cv.rectangle(img,(384,0),(510,128),(0,255,0),3)`



Drawing a circle

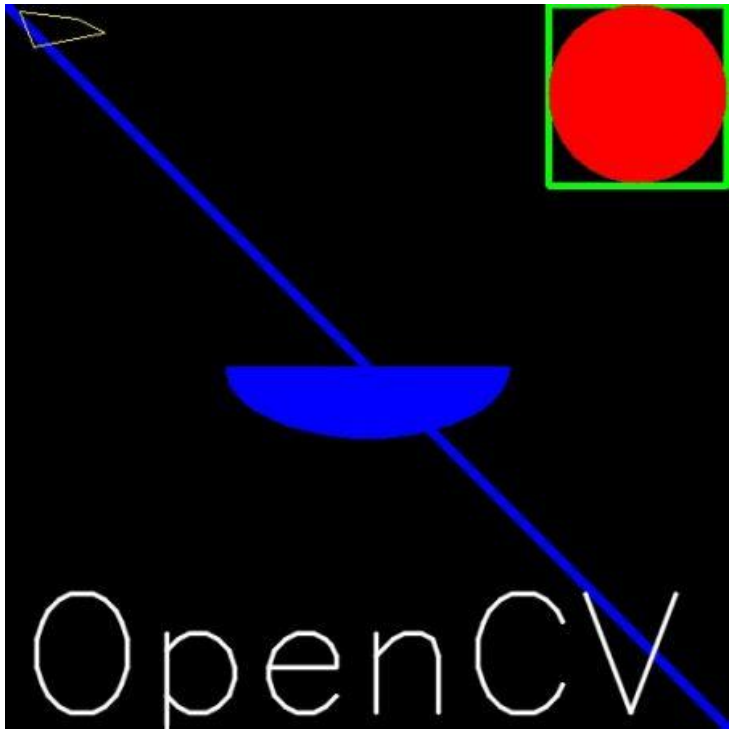
- `cv.circle(img,(447,63), 63, (0,0,255),5)`
- `cv.circle(img,(447,63), 63, (0,0,255), cv.FILLED)`





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: Mouse as a Paint-Brush]

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.
);
```

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()
```

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, R. Szeliski (http://szeliski.org/Book)

[2] Learning OpenCV 3, A. Kaehler & G. Bradski

- Available online through Safari Books, Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51532449200003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US

[3] Practical introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe

- Available through Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA51428109500003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US