

# Image Processing

သတိ: စာကိုမကျက်ပါနှင့်။  
နားလည်အောင်ဖတ်ပြီးစဉ်းစားပါ။

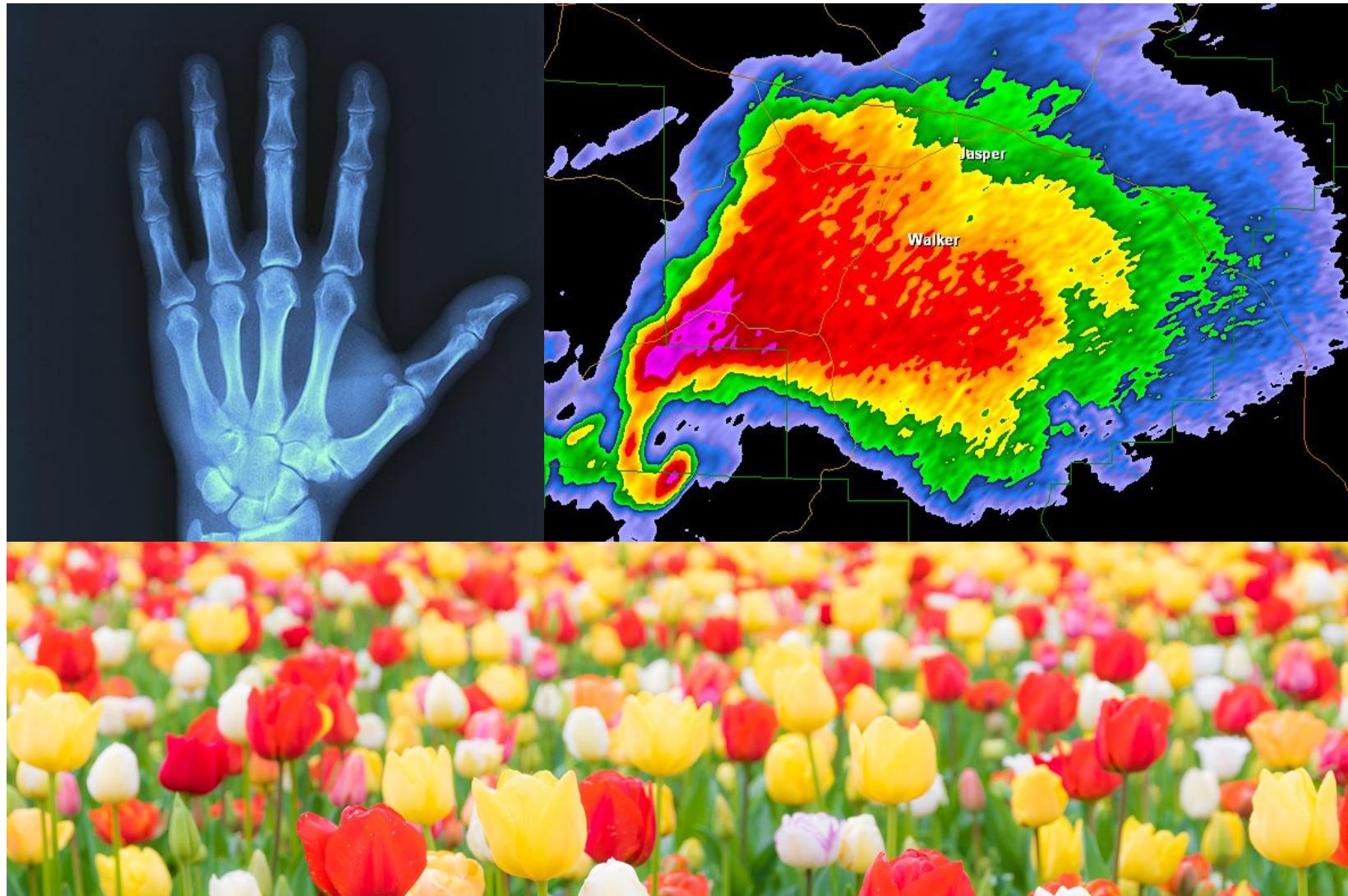
# Introduction

- Signal Processing ကို ကျွန်တော်တို့ သီအိရိများ ဖြင့်လေ့လာခဲ့ပြီးပါပြီ။
- သီအိရိများသည် သီအိရိများသာ ဖြစ်ပါသည်။ ဒါကို လက်တွေ့ပြန်အသုံးချိနိုင်ဖို့ အခါ ကျွန်တော်တို့ Signal Processing ကို Image Processing အနေဖြင့် လက်တွေ့လုပ်ဆောင်ကြည့်မည် ဖြစ်သည်။
- အမှန်တော့ Image Processing သည် Signal Processing ၏ Branch တစ်ခုဖြစ်ပါသည်။ ထို့ကြောင့်ပင် Image Processing ကိုနားလည်ဖို့ Signal Processing အခြေခံကို နားလည်ထားဖို့လိုပါသည်။
- Image Processing Lab ဟုဆိုသော်လည်း အလျဉ်းသင့်သလို လိုအပ်သော Image Processing သီအိရိများကိုလည်း ဆွေးနွေးသွားပါမည်။

# Image

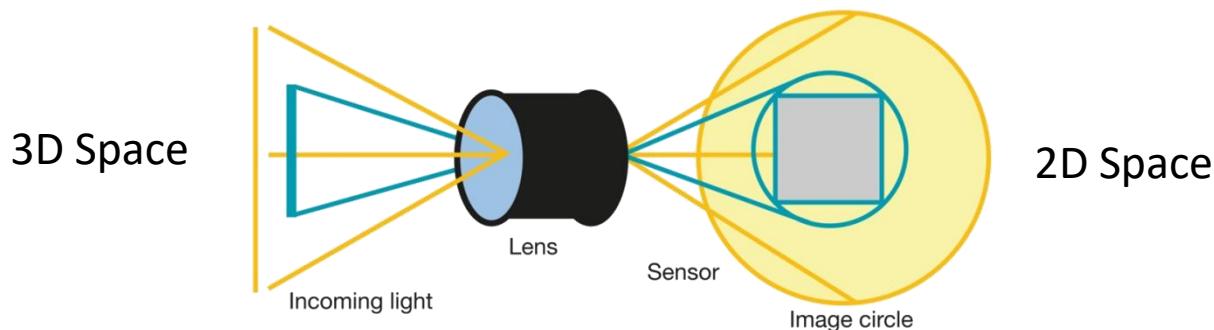
- Image ဆိုတာဘာလဲ။ ဘာကို Image လို့ ခေါ်တာလဲ ပေါ့။
- အမှန်တော့ Image လို့ခေါ်တဲ့ ဓာတ်ပုံများ ပေါ်လာသည်က သိပ်မကြာသေးပါ။ အမှန်တော့ Engine များထက်တောင် နောက်ကျပါသည်။ Digital Image များပေါ်သည်က ဆယ်စုနှစ် အနည်းငယ် လောက်ပဲ ရှိပါသေးသည်။
- အမှန်တော့ Image တစ်ခုသည် Light (Electromagnetic) Radiation (အလင်းဖြာခြင်းကြောင့်) ဖြစ်ပေါ်လာသော Light Intensity များ၏ Reflection နှင့် Reaction တစ်ခုသာ ဖြစ်ပါသည်။
- တစ်နည်းအားဖြင့် Image တစ်ခုသည် Light Radiation ၏ Intensity Record တစ်ခု ဖြစ်သည်လို့ အကြမ်းဖျဉ်းပြောနိုင်ပါသည်။
- ဒါကြာင့် Light (Electromagnetic) Radiation ပေါ်မှုတည်ပြီး Image မျိုးစုံရှိပါသည်။ Visible Light ပေါ်မှုတည်တဲ့ Camera Image, X-Ray ပေါ်မှုတည်တဲ့ X-Ray Image, Radio Wave ပေါ်မှုတည်တဲ့ Radar Image စသည်ဖြင့် အများကြီး ဖြစ်ပါသည်။

# Images of Different Radiation

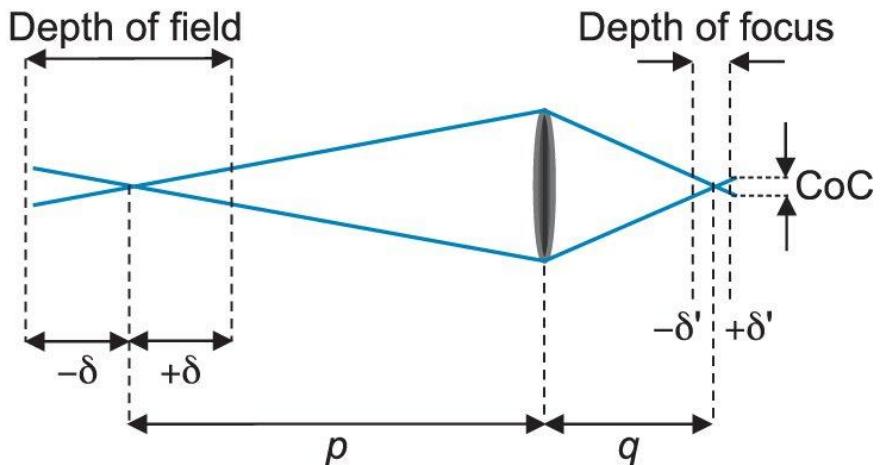
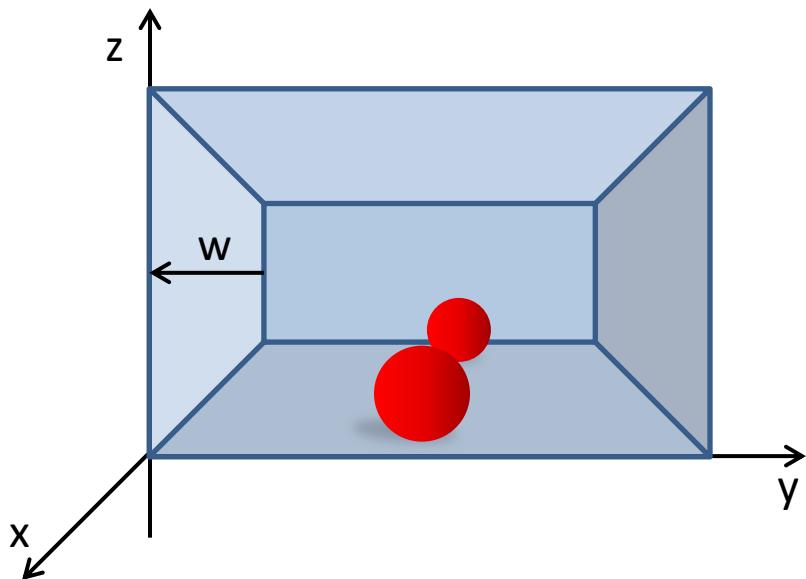


# Camera Image

- Image တွေအတဲ့မှာမှ ကျွန်တော်တို့ Visible Light ကြောင့် ဖြစ်တဲ့ Camera Image များကို လေ့လာပါမည်။
- Camera သည် မျက်လုံးကို တုပထားသော Device တစ်ခုဖြစ်ပြီး Lens နှင့် Light (Visible) Sensors များပါဝင်သည်။
- Lens ၏ အဓိကတာဝန်သည် အလင်းများကို စုပေးဖို့ ဖြစ်သည်။ Mathematically ပြောရင်တော့ Lens သည် Linear Transformation တစ်ခုဖြစ်ပြီး 3D Space ကို 2D Space သို့ Map လုပ်ပေးပါသည်။
- Sensors များကတော့ Light Intensity များကို Record လုပ်ပါသည်။
- ဤနည်းဖြင့် ကျွန်တော်တို့ 3D World ကို ဖော်ပြုသော 2D Image တစ်ခုကို ရရှိပါသည်။ ဒါကြောင့် Image တစ်ခု၏ Quality သည် Lens နှင့် Sensors များပေါ်မှုတည်ပါသည်။

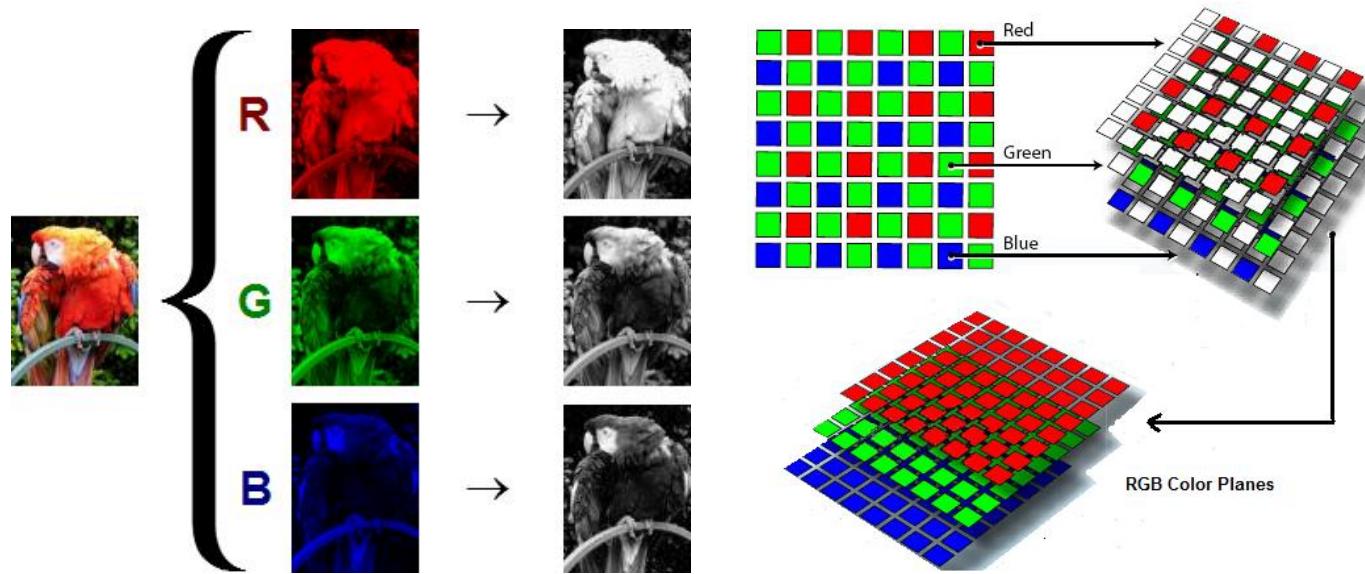


# Homogeneous Coordinates



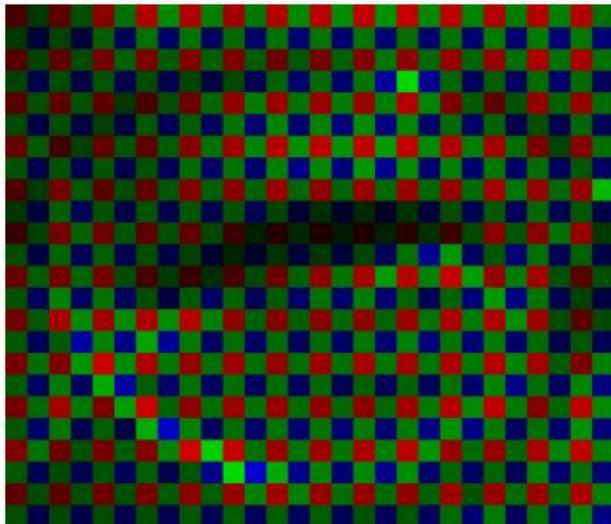
- အမှန်တော့ Camera Image များသည် 2D Homogenous Space ဖြစ်သည်။ 2D Homogenous Space သည် Depth Information ပါဝင်သော 2D Space ဖြစ်သည်။ Depth Information သည် Camera Lens ၏ Focal Length ပေါ်မူတည်သည်။
- ထိုအပြင် Lens ၏ Focal Length ပေါ်မူတည်ပြီး Depth Information များကွဲပြားမည် ဖြစ်သည်။
- ထိုကြောင့် Focal Length သည် Image တစ်ခု၏ Depth ကို ဖော်ပြုပါသည်။

# Light Intensity

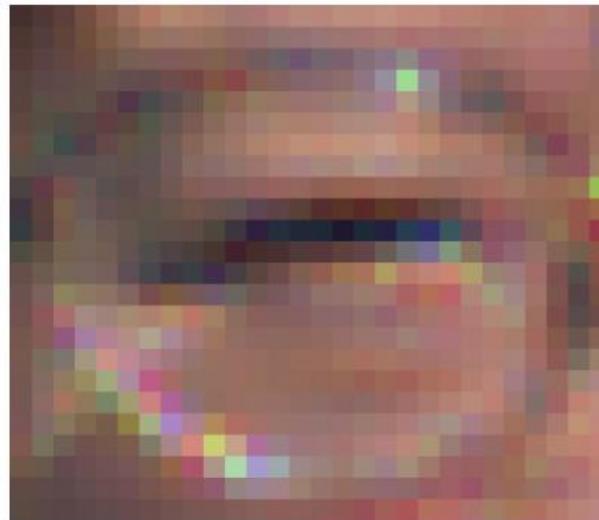


- Light Sensors များတွင် Red, Green နှင့် Blue (Light Frequency) Sensors များပါဝင်ပြီး Grid များဖြင့် Detect လုပ်ပါသည်။
- ပြီးရင် Light Intensity များကို RGB Channel Grid များဖြင့် ဖော်ပြပါသည်။ RGB Channel Grid များဖြင့် ဖော်ပြထားသော Image များသည် Raw Image File များဖြစ်ပါသည်။
- Digital Camera အများစုံမှာ Raw Image File များကို ရနိုင်သည်ဟု ပြောကြသည်။ (ကျွန်တော်တော့ မစမ်းဖူးသေးပါ။)

# Demosaicing



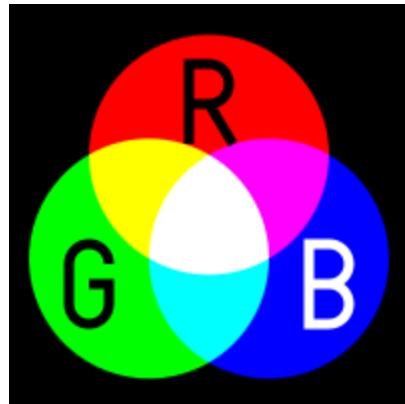
Raw Image



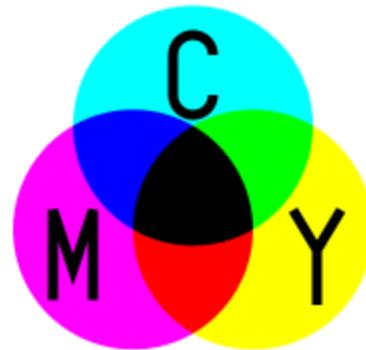
Processed Image

- Raw Image File ကို Demosaicing လုပ်လိုက်သည့် အခါမှာတော့ ကျွန်တော်တို့ တွေ့နေကြ Color Image များကို ရမည်ဖြစ်ပါသည်။
- အမှန်တော့ Color Image မှ RGB Channels များကို ပြန်ခဲ့ထုတ်လို ရပါသည်။ ဒီအကြောင်းတော့ နောက်မှ ပြောပါမည်။

# Types of Color Space



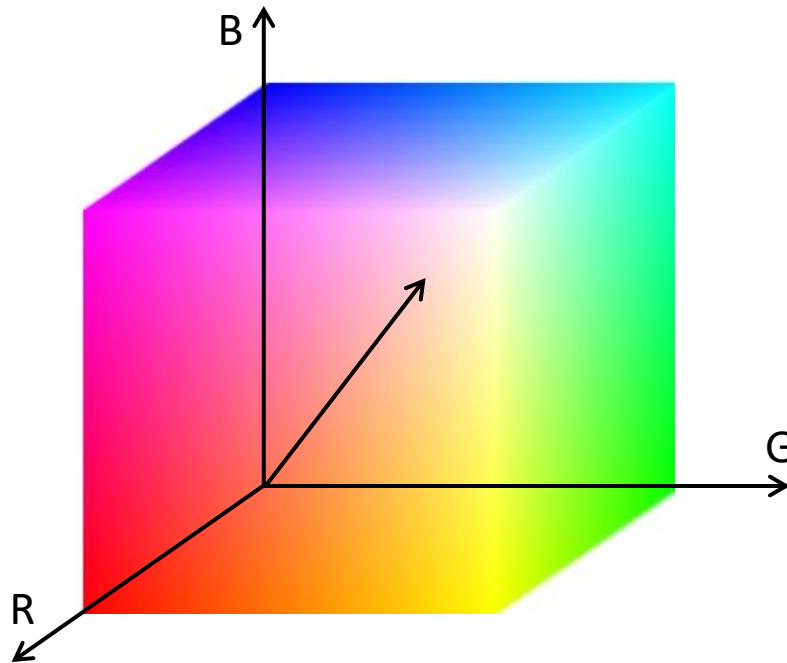
Primary Color of Light



Primary Color of Pigment

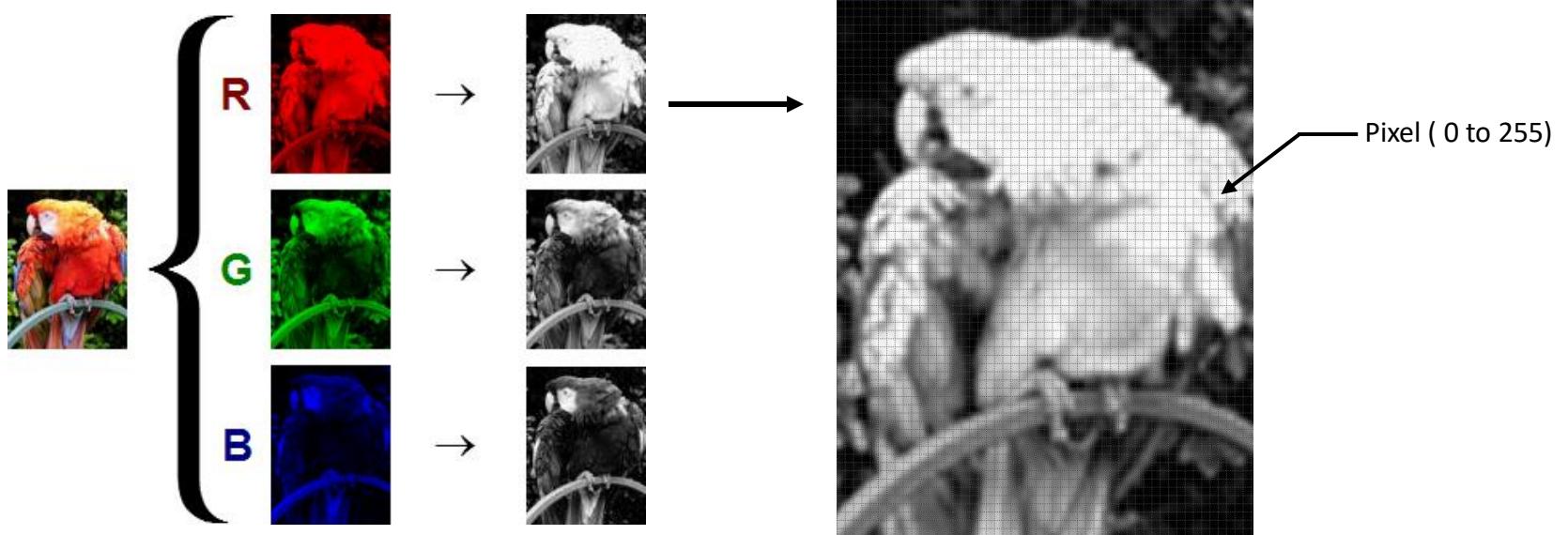
- Color Space နှစ်မျိုးရှိပါသည်။ Primary Color of Light လို့ ခေါ်တဲ့ အလင်းရောင် (Light Source) ကြောင့်ဖြစ်သော အရောင်များနှင့် Primary Color of Pigments လို့ ခေါ်တဲ့ အလင်းပြန်ခြင်း (Light Reflection) ကြောင့် ဖြစ်သော အရာင်များ ဆိုပြီး 2 မျိုးရှိပါသည်။
- Primary Color of Light ကို Light Source များမှာ သုံးပါသည်။ ဥပမာ Monitor Screen, TV Screen။
- Primary Color of Pigments များကို Printing မှာသုံးပါသည်။ ဥပမာ ပန်းချီဆွဲခြင်း၊ ပုံနှိပ်ခြင်း။

# Color Space



- Color Space တစ်ခုသည် Orthogonal Color Vector ( $R, G, B$ ) များ၏ Orthogonal Vector Space ဖြစ်သည်။
- Orthogonal Basis Color Vector များဖြင့် ကြိုက်တဲ့ Color ကို ဖော်ပြလို့ ရပါသည်။
- တစ်နည်းအားဖြင့် Color များသည် Orthogonal Basis Color Vector များ၏ Linear Combination ဖြစ်သည်။

# Grayscale Image



- RGB Channel တစ်ခုခြင်းစီမှု ပုံများကို ယူ၍သော်လည်းကောင်း၊ သို့မဟုတ် RBG Channel အားလုံး၏ Average ကိုရှု၍သော်လည်းကောင်း ကျွန်တော်တို့ Grayscale Image ကိုရရှိမှာ ဖြစ်ပါသည်။
- Grayscale Image ကို Pixel များဖြင့် ဖွဲ့စည်းထားပြီး Pixel တစ်ခုသည် ( $0 = \text{Black}$ ,  $255 = \text{White}$ ) တန်ဖိုးရှုမည်ဖြစ်သည်။
- ထိုကြောင့် Grayscale Image တစ်ခုသည်  $n \times m$  Matrix တစ်ခု ဖြစ်သည်။ Color Image သည် Grayscale Image များကို ထပ်ထားသော  $n \times m \times p$  Tensor တစ်ခုဖြစ်သည်။

# Python OpenCV



<https://pypi.org/project/opencv-python/>

- OpenCV သည် Powerful Computer Vision Library ဖြစ်ပါသည်။ ကျန်တော်တို့ရဲ့ Lab အတွက် OpenCV ကိုသုံးပါမည်။ OpenCV သည် C++, C#, Java နှင့် Python တို့အတွက် Library များရှိပါသည်။ သို့သော် ကျန်တော်တို့ Lab အတွက်တော့ Python ပဲ သုံးမည် ဖြစ်သည်။

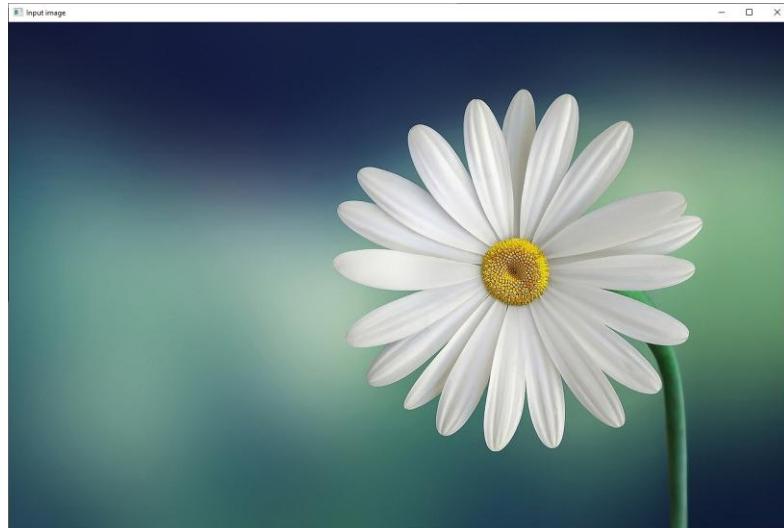
```
pip install opencv-contrib-python
```

# Our First Image

- ပုံများကိုတော့ ကျွန်တော် <https://pixabay.com/> မှရပါသည်။ ကျွန်တော်တို့ ဒီလိုဆိုရင် ပုံကိုမြင်ရမှာပါ။

```
import cv2

img = cv2.imread('./images/flower.jpg')
cv2.imshow('Input image', img)
cv2.waitKey()
```

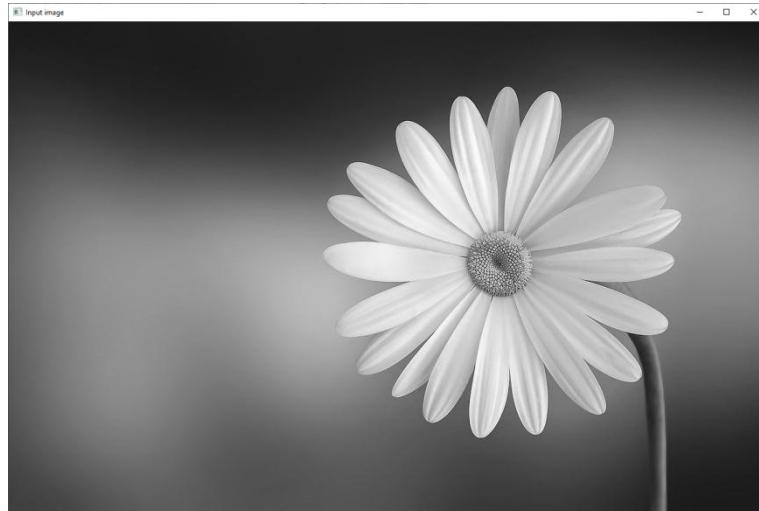


# Grayscale Image

- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် Grayscale ပုံကိုမြင်ရမှာပါ။

```
import cv2

img = cv2.imread('./images/flower.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Input image', img)
cv2.waitKey()
```



# RGB Channel

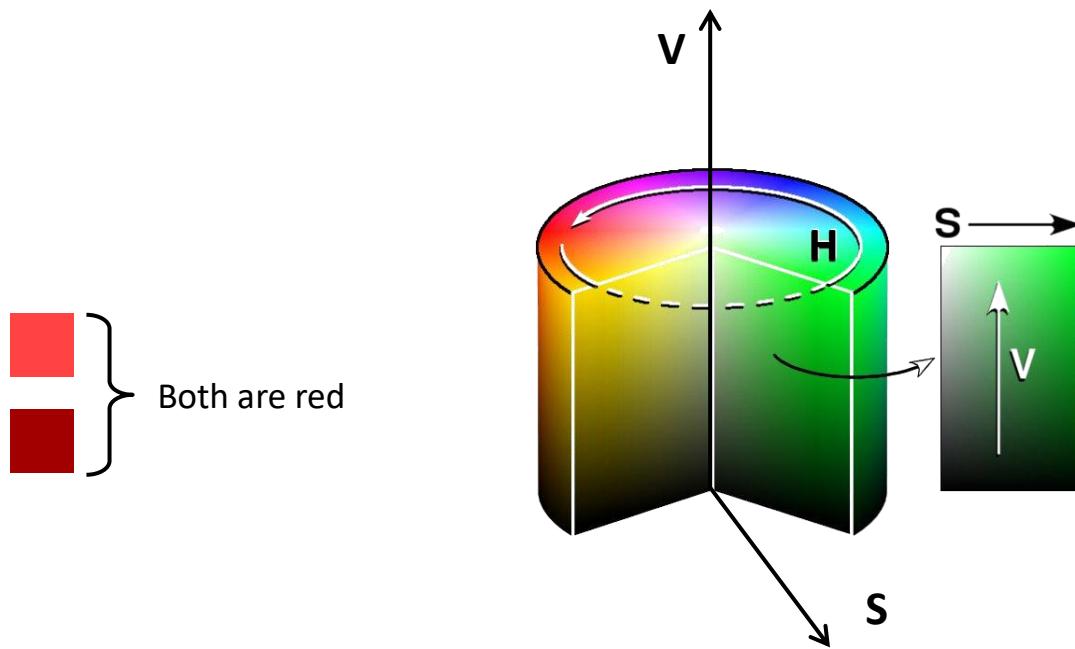
- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် လိုအပ်တဲ့ Color Channel ကိုခွဲထုတ်လိုရမှာပါ။

```
import cv2

img = cv2.imread('./images/flower.jpg')
#R = 0, G = 1, B = 2
cv2.imshow('R Channel', img[:, :, 0])
cv2.waitKey()
```

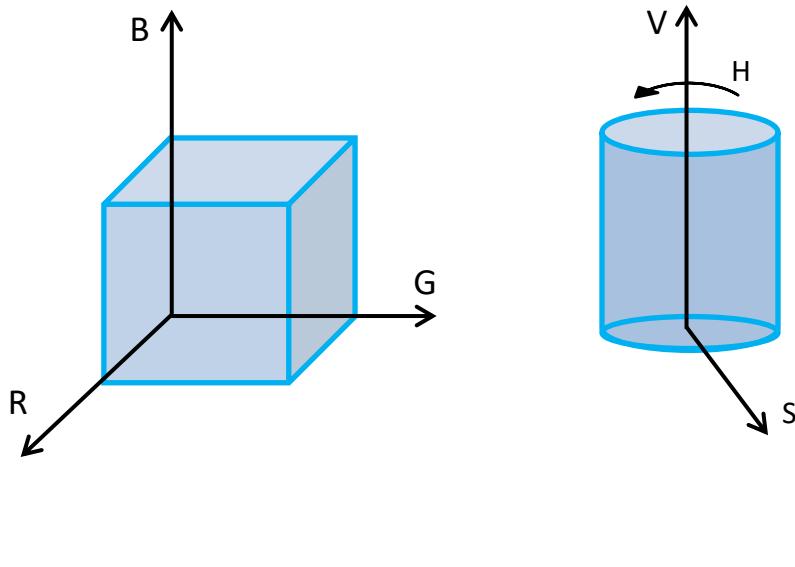


# HSV Color Space



- RGB Color Space သည် Euclidean Space ဖြစ်သည်။ သို့သော် ကျွန်တော်တို့ မျက်စိကတော့ အရောင်များကို အများကြီး မခဲ့နိုင်ပါ။ ဥပမာ အနီရောင် နှစ်မျိုးလုံးသည် အနီရောင်ဖြစ်သည်။ အနှုန်းဆုံး အရင်သာကွာသည်။
- ထို့ကြောင့် လူမျက်စိနှင့်တူအောင် ကျွန်တော်တို့ HSV Color Space ကို လိုပါသည်။ HSV သည် Cylindrical Space ဖြစ်သည်။
- H (Hue) က အရောင်ကို ပြောသည်၊ S (Saturation) က မီးခိုး (Gray) ပါဝင်မှုကို ပြောပြီး V (Value) က အလင်း (Light) ပါဝင်မှုကို ပြောသည်။

# Transforming Color Space



$$\begin{aligned}R' &= R / 255 \\G' &= G / 255 \\B' &= B / 255\end{aligned}$$

$$\begin{aligned}C_{max} &= \max(R', G', B') \\C_{min} &= \min(R', G', B') \\C_{\Delta} &= C_{max} - C_{min}\end{aligned}$$

$$H = \begin{cases} 0^\circ, \Delta = 0 \\ 60^\circ \times \left( \frac{G' - B'}{\Delta} \text{ (mod 6)} \right), C_{max} = R' \\ 60^\circ \times \left( \frac{B' - R'}{\Delta} + 2 \right), C_{max} = G' \\ 60^\circ \times \left( \frac{R' - G'}{\Delta} + 4 \right), C_{max} = B' \end{cases}$$

$$S = \begin{cases} 0, C_{max} = 0 \\ \frac{\Delta}{C_{max}}, C_{max} \neq 0 \end{cases}$$

$$V = C_{max}$$

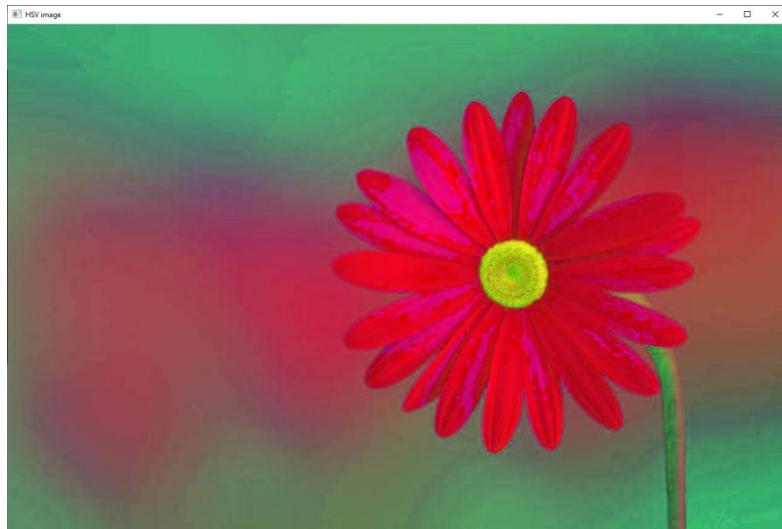
- ကျွန်ုတ်တို့ အနီရောင်ရှိသော အဝတ်အထည်များကို ရှာမည်ဆိပါတော့။
- ဒါဆိုရင် HSV Color Space က RGB Color Space ထက်လိုပြီး အဆင်ပြေမှာ ဖြစ်ပါသည်။

# HSV Color Image

- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် HSV Color Image ပုံကိုမြင်ရမှာပါ။ ဒီမှာ ပြသော အရောင်များသည် တကယ့်အရောင်မဟုတ်ပါ။ HSV ၏ Projected Color သာသည် ဖြစ်သည်။

```
import cv2

img = cv2.imread('./images/flower.jpg')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow('HSV image', hsv_img)
cv2.waitKey()
```

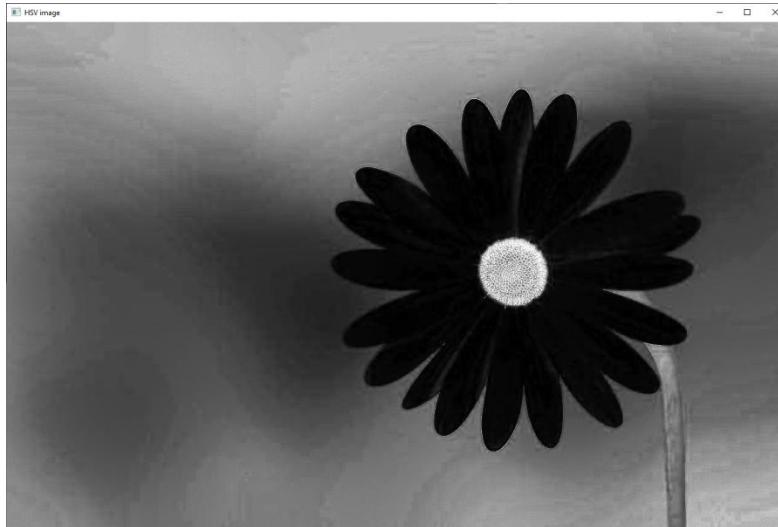


# HSV Channel

- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် HSV Saturation Image ပုံကိုမြင်ရမှာပါ။

```
import cv2

img = cv2.imread('./images/flower.jpg')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
#H = 0, S = 1, V = 2
cv2.imshow('HSV image', hsv_img[:, :, 1])
cv2.waitKey()
```

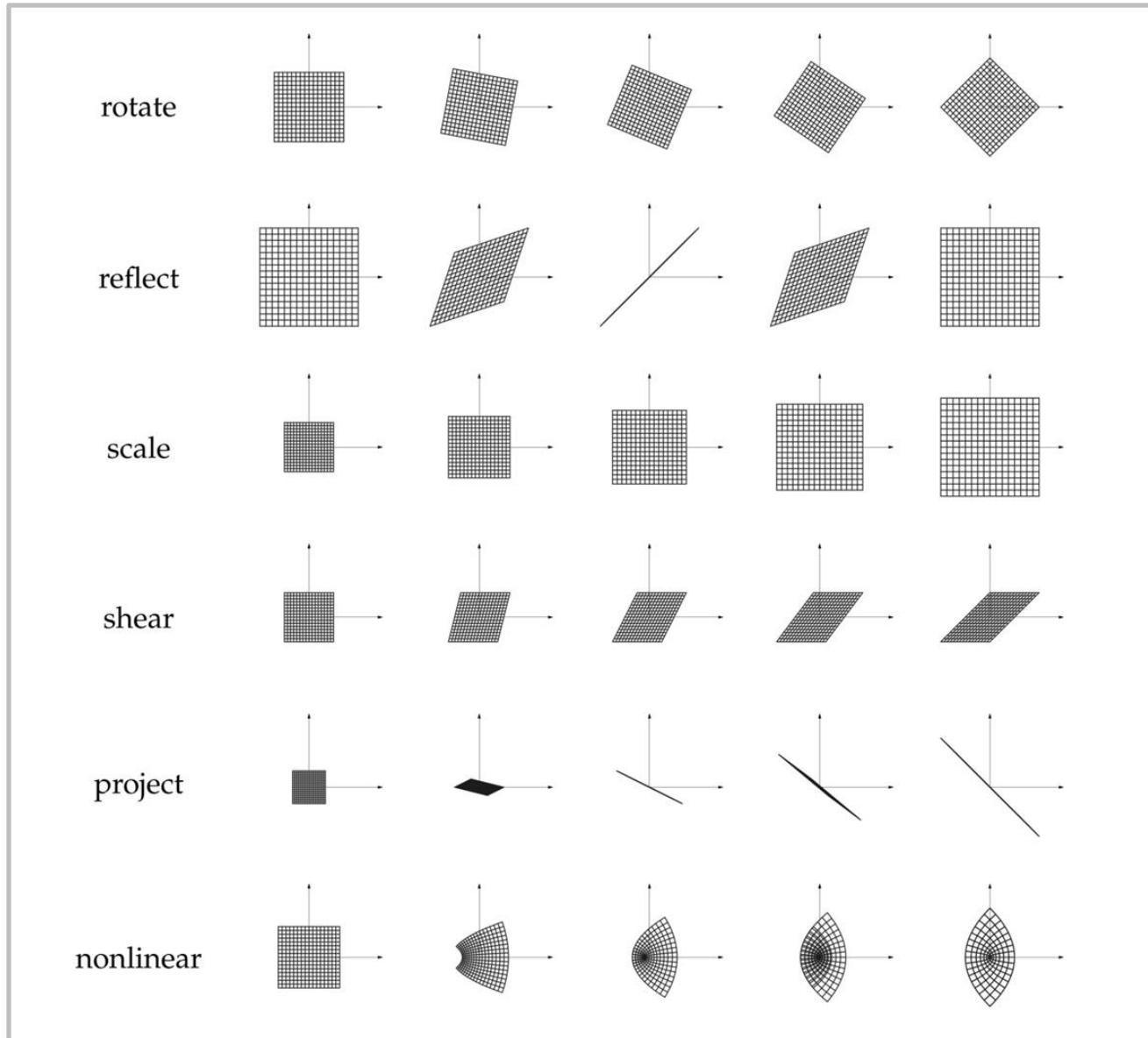


# Linear Transformation

$$\mathbf{y} = \mathbf{Ax} + \mathbf{b}$$

- Linear Transformation တစ်ခုကို ဒီလို ဖော်ပြကြပါသည်။ အမှန်တော့ ကျွန်တော်တို့ Linear Transformation ကို သုံးပြီး Image များကို Transform လုပ်လို ရပါသည်။
- Linear Transformation ကို စဉ်းစားရင် အချက် 2 ချက်ကို သတိထားရပါမည်။
  - Additivity : Linear Entity များကို အချင်းချင်း ပေါင်းလျှင် အမြဲ Linear Entity ပဲ ပြန်ရပါသည်။
  - Scalability : Linear Entity တစ်ခုကို Scalar ဖွင့် မြှောက်လျှင်လည်း အမြဲ Linear Entity ပဲ ပြန်ရပါသည်။
- Image များကို ယော့ယျှအားဖြင့် Linear Tensor များအနေဖြင့် ယူဆလို ရပါသည်။ (ဒါကိုတော့ သတိထားပါ။ ဒီလို ယူဆခြင်းသည် အမြဲမှန်ပါ။ Image များသည် အမှန်တော့ 2D Homogenous Space ဖြစ်ပါသည်။ Flat 2D မဟုတ်ပါ။ Depth Information သည် အရေးမကြီးဘူးလို ယူဆမှ မှန်ပါသည်။)
- တစ်ခါ Image များကို Scale လုပ်လျှင်လည်း ကျွန်တော်တို့ Resolution ပြသုနာ ရှိပါသည်။ Image တစ်ခုကို ပုံကြီးခဲ့လျှင် ပိုဝင်း (Blur) လာမည် ဖြစ်သည်။ ဒါအကြောင်းကို ကျွန်တော်တို့ နောက်မှ ဆွေးနွေးပါမည်။
- အခုလောလောဆယ်တော့ Image များကို ယော့ယျှအားဖြင့် Linear Tensor များအနေဖြင့် ယူဆထားပါ။

# Linear Image Transformation



# Linear Image Transformation

Rotate



Reflect



Scale



Shear



Project



# Non Linear Image Transformation



# Image Transformation

$$Y = \text{Transformation Matrix} \times \text{Image}$$

- Image Transformation များကို Matrix Multiplication များဖြင့် လုပ်ဆောင်နိုင်ပါသည်။
- Euclidean Transformation များကို Affine Transformation ဟု ခေါ်ပါသည်။ Affine Transformation တွင် Space Dimension များ မပြောင်းပါ။ Origin Point မပြောင်းပါ။ Line များ မပြောင်းပါ။ (ဆိုလိုသည် Line များသည် Curve မဖြစ်သွားပါ။) Rotation, Reflection, Shearing, Scaling တို့သည် Affine Transformation များဖြစ်ကြသည်။
- Perspective Transformation များတွင် Space Dimension များပြောင်းသွားပါသည်။ 2D Space မှ 3D Space သို့ပြောင်းသွား နိုင်သည်။
- Non Linear Transformation များတွင် အကုန်လုံး ပြောင်းနိုင်ပါသည်။
- Image တစ်ခုကို Transformation Matrix များနှင့် Matrix Multiplication လုပ်ခြင်းဖြင့် Image Transformation များလုပ်ဆောင်နိုင်ပါသည်။

# Shearing

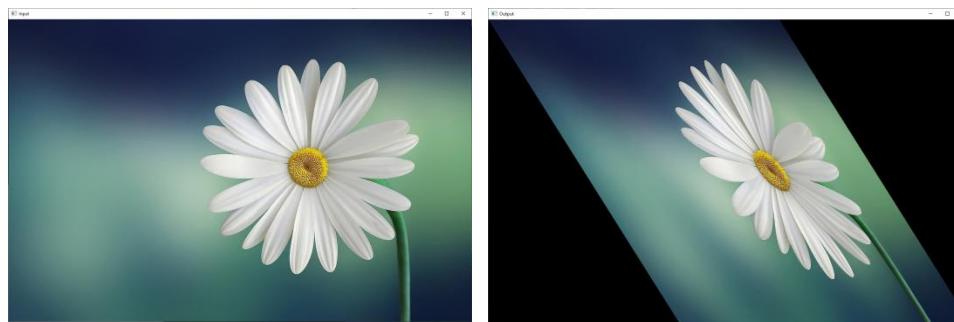
- ကျွန်ုတ်တို့ ဒီလိုခိုရင် Shearing ပုံကိုဖြင့်ရမှာပါ။

```
import cv2
import numpy as np

img = cv2.imread('./images/flower.jpg')

rows, cols = img.shape[:2]
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[0,0], [int(0.6*(cols-1)),0], [int(0.4*(cols-1)),rows-1]])
affine_matrix = cv2.getAffineTransform(src_points, dst_points)
img_output = cv2.warpAffine(img, affine_matrix, (cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.waitKey()
```



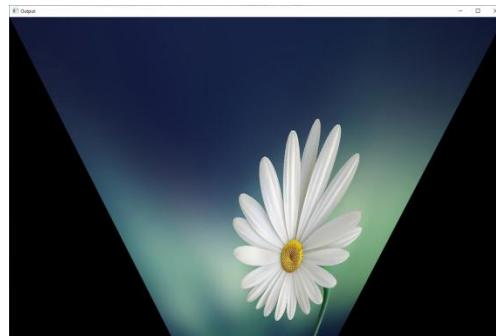
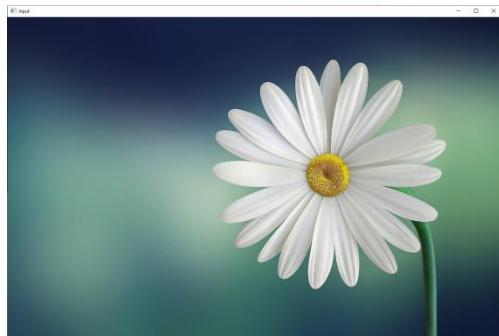
# Perspective Projection

- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် Projection ပုံကိုဖြင့်ရမှာပါ။

```
import cv2
import numpy as np

img = cv2.imread('./images/flower.jpg')

rows, cols = img.shape[:2]
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1], [cols-1,rows-1]])
dst_points = np.float32([[0,0], [cols-1,0], [int(0.33*cols),rows-1], [int(0.66*cols),rows-1]])
projective_matrix = cv2.getPerspectiveTransform(src_points, dst_points)
img_output = cv2.warpPerspective(img, projective_matrix, (cols,rows))
cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.waitKey()
```



# Non Linear

- Non Linear Transformation အတွက်တော့ ပုံသေမရှိပါ။ Algorithm များအပေါ်မူတည်ပါသည်။ ကျန်တာတွေကိုစာအုပ်ထဲကအတိုင်းစမ်းကြည့်ပါ။

```
import cv2
import numpy as np
import math

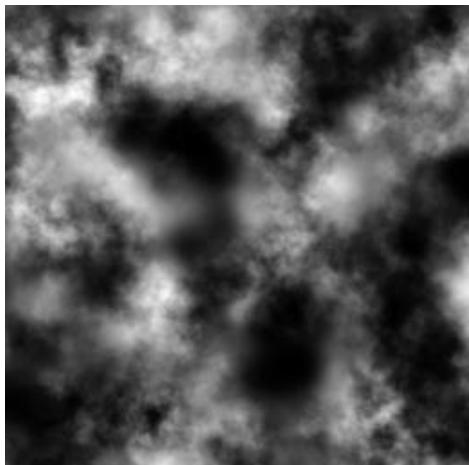
img = cv2.imread('./images/flower.jpg', cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape
# Vertical wave
img_output = np.zeros(img.shape, dtype=img.dtype)

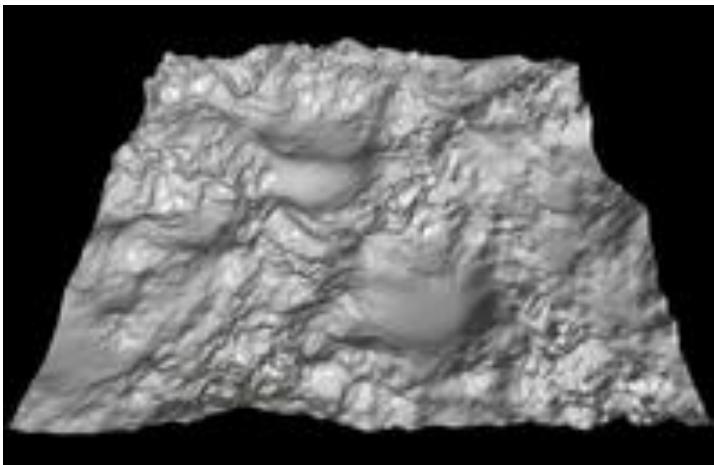
for i in range(rows):
    for j in range(cols):
        offset_x = int(25.0 * math.sin(2 * 3.14 * i / 180))
        offset_y = 0
        if j+offset_x < rows:
            img_output[i,j] = img[i,(j+offset_x)%cols]
        else:
            img_output[i,j] = 0
cv2.imshow('Input', img)
cv2.imshow('Vertical wave', img_output)

cv2.waitKey()
```

# Height Map of an Image



Height Map



3D Terrain

- ကျွန်ုတ်တို့သည် Grayscale Image တစ်ခုကို အမှန်တော့ Height Map တစ်ခုအနေဖြင့် ယူဆလို ရပါသည်။ 0 = Black သည် အနိမ့်ဆုံး မြပ်ပြန်ဖြစ်ပြီး 255 = White သည် အမြင့်ဆုံး တောင်ထိပ်ဖြစ်မည်။
- ဒါဆိုရင် Grayscale Image တစ်ခုသည် မြပ်ပြန်အနိမ့်အမြင့် ပါဝင်သော Terrain တစ်ခုကို ဖော်ပြပါသည်။ (ဒီအချက်ပေါ်မှတ်ည်ပြီး Procedural Terrain များကို Game များ၊ Computer Simulation များမှာ ဆွဲကြပါသည်။)
- ဒီနေရာ Height Map ကို Depth Map ဖြင့် မရောထွေးစေခဲ့ပါ။ Depth Map က 2D Homogenous Space ၏ Depth Information ဖြစ်ပြီး Height Map က Pixel Value များ၏ Variation (Height) ကို ဆိုလိုပါသည်။

# Depth Map of an Image



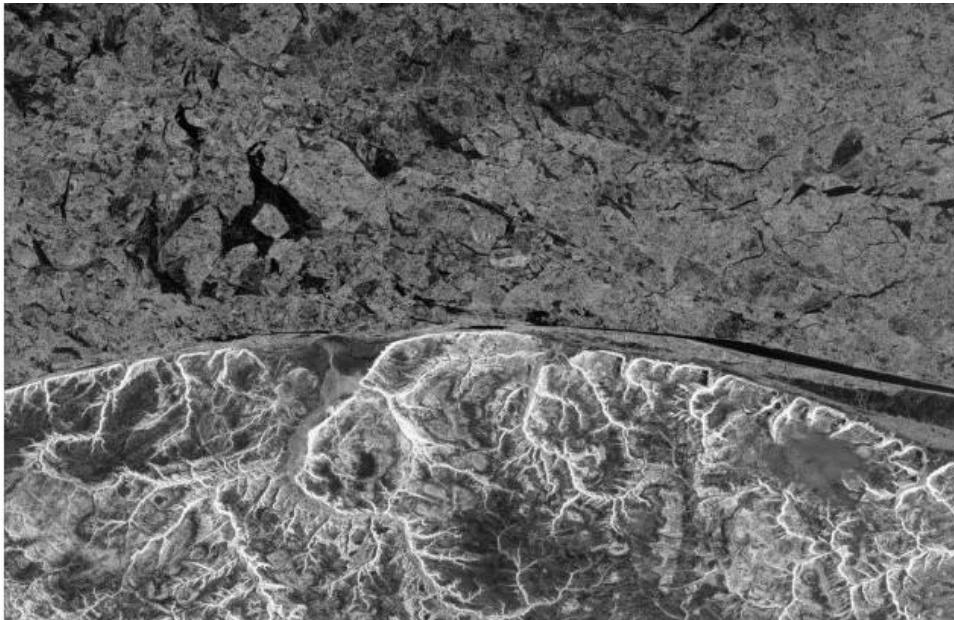
(a) Real-image



(b) Depth-map

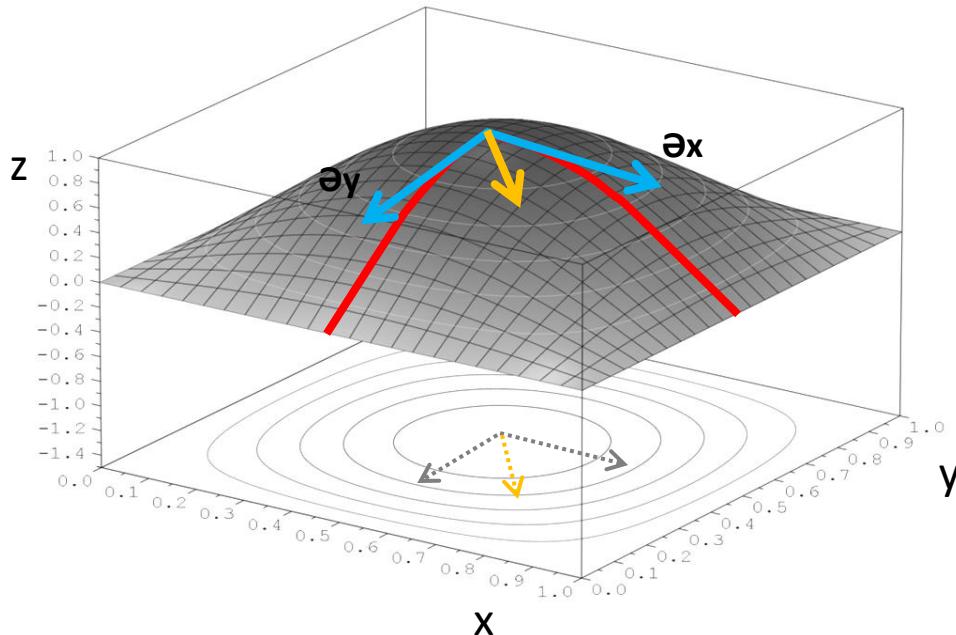
- Depth Map က 2D Homogenous Space ၏ Depth Information ဖြစ်ပါသည်။
- Depth Map တစ်ခုကို Grayscale Image ဖြင့်ဖော်ပြသည့် အခါ White (255) သည် Camera နှင့် အနီးဆုံးမှာ ရှိပြီး Black (0) သည် အဝေးဆုံးမှာ ရှိပါသည်။
- ပုံမှန် Camera များတွင် Depth Information ပါသော်လည်း အများအားဖြင့် Blur (Due to Focal Length) ဖြစ်လေ့ရှိပါသည်။ ထိုကြောင့် Depth Information ကို ပိုမြဲး အသေးစိတ်သိလိုပါက 3D Camera များ (ဥပမာ Microsoft Kinect )ကိုသုံးကြပါသည်။

# Aerial Photo



- Aerial Photo (Satellite Image) များတွင်မူ Height Map နှင့် Depth Map တို့တူတဲ့ ဖြစ်ပါသည်။ အမြင့်ဆုံးသည် Camera နှင့် အနီးဆုံးဖြစ်၍ ဖြစ်သည်။
- ထိုကြောင့် Real 3D Terrain ကို ကျန်တော်တို့ Generate လုပ်မည်ဆိုပါက လုပ်လို့ ရပါသည်။ သော်ချာချာကြည့်ပါက တောင်တိုင်များသည် အဖြူရောင် (255) ဖြစ်ပြီး ချောက်များ၊ တွင်းများသည် (0) ဖြစ်ပါသည်။
- အမှန်တော့ Aerial Photo မှမဟုတ် Grayscale Image တိုင်းကို Height Map အနေဖြင့် ယူဆလို့ရပါသည်။
- သို့သော် Grayscale Image တိုင်းတွင်မူ Height Map နှင့် Depth Map တို့တူမည် မဟုတ်ပါ။

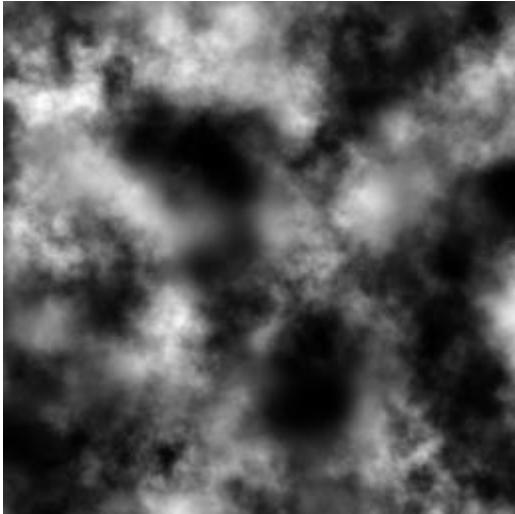
# Gradient



- $z = f(x, y)$  ဆိုတဲ့ 2 Variable Scalar Function မှာ  $\frac{\partial z}{\partial x}$  နှင့်  $\frac{\partial z}{\partial y}$  ဆိုပြီ: Partial Derivative နစ်ခုရှိပါသည်။
- ဒီ Partial Derivative နစ်ခုကို Gradient Vector တစ်ခု လိုပြောခဲ့ပါသည်။ ဒါဆိုရင် Grayscale Image တစ်ခုရဲ့ Gradient ဆိုရင်ရော ဘယ်လို့ဖြစ်မလဲ။

$$\nabla f = \left\langle \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right\rangle$$

# Gradient of a Grayscale Image



$$z \text{ (height)} = f(x, y)$$

$$\nabla f = \left\langle \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \right\rangle$$

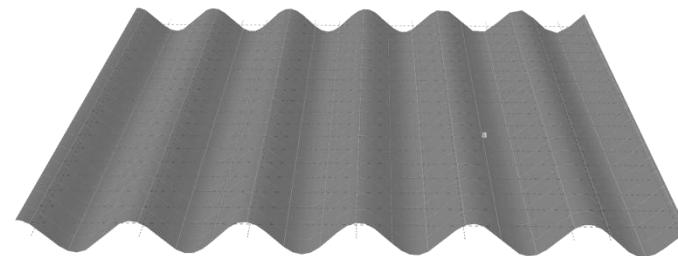
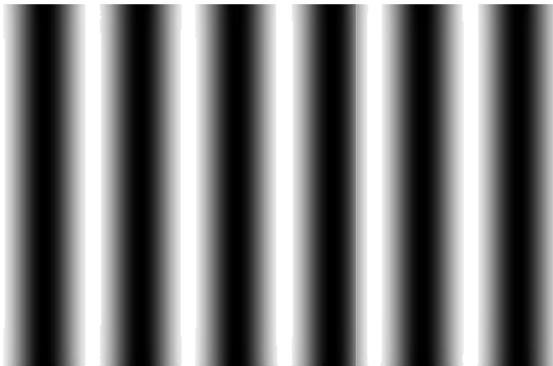
Grayscale Image (Height Map)

- Grayscale Image တစ်ခုသာ Height Map တစ်ခုသာ ဆိုရင် Gradient of Height Map သည် Height Map မှာရှိတဲ့ တောင်းကုန်များ၊ ချောက်များ၏ လွှာဗောက် (Slope) များဖြစ်ကြမည်။
- ဟုတ်ပါပြီ၊ ဒါဆိုရင် ကျွန်တော်တို့ Grayscale Image တစ်ခု၊ Gradient ကို ဘယ်လိုရှာမလဲ။

# Spatial Wave

$y = f(t)$  , Temporal Signal

$z = f(x, y)$  , Spatial Signal



- တခါကျွန်တော်တို့ Grayscale Image များကို Spatial Wave များအနေဖြင့် ယူဆလို့ရပါသည်။
- ဒါဆိုရင် Image များရဲ့ Intensity (Grayscale) များ၏ ပြောင်းလဲမှုသည် Frequency နှင့် ဆက်စပ်နေမည် ဖြစ်သည်။ တစ်နည်းအားဖြင့် Change of Intensity (Gradient) များထဲမှု Frequency များလာမည်ဖြစ်သည်။

# Convolution

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t - \tau)d\tau$$

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n - k]$$

- ကျွန်ုတ်တို့ Convolution ကို LTI System ရဲ့ Zero State Response အနေဖြင့် ပြောခဲ့ပါသည်။ ဒီလိုယူဆရင် ဒါသည် Temporal Convolution ဖြစ်သည်။ Temporal Convolution သည် Signal တစ်ခု၏ Time Domain Filter ဖြစ်သည်။
- Image Processing မှာတော့ ကျွန်ုတ်တို့ Spatial Convolution ကိုစဉ်းစားကြည့်ပါမည်။ တစ်နည်းအားဖြင့် Temporal Wave Signal များ၏ Convolution အစား Spatial Wave Signal များ၏ Convolution အနေဖြင့်စဉ်းစားကြည့်မည် ဖြစ်သည်။
- ထိုအခါ Spatial Convolution သည် Image တစ်ခုရဲ့ Spatial Filter တစ်ခု ဖြစ်သွားပါသည်။

# Spatial Convolution

$$y[n, m] = x[n, m] * h[n, m] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x[i, j]h[n - i, m - j]$$

- Grayscale Image တစ်ခု၏ Spatial Convolution သည် Spatial Filter တစ်ခုဖြစ်ပြီး ထို Image ၏ Spatial Information များကို Filter လုပ်ပေးပါသည်။
- Spatial Information ဆိုသည်မှာ Image တစ်ခု၏ Position နင့် Intensity (Grayscale Value 0 to 255) ကို ဆိုလိုပါသည်။
- ထို့ကြောင့် Spatial Convolution ကိုသုံးပြီး Spatial Filter များကို လုပ်ဆောင်နိုင်ပါသည်။ Spatial Filter များကို Kernel ဟုခေါ်ပါသည်။
- ထိုအပြင် Spatial Convolution များသည် Neighborhood Operations များဖြစ်သည်။ ဆိုလိုသည်က Image တစ်ခုလုံး အပေါ်မှာ လုပ်ဆောင်သော Operation မဟုတ်ဘဲ Pixel တစ်ခု၏ ပတ်ဝန်းကျင်နားကို လုပ်ဆောင်သော Operation များဖြစ်သည်။
- Convolution ကို ဘယ်လိုလုပ်သည်ကို Signal Processing မှာပြောပြီးသား ဖြစ်သည်။

# Sobel Filter

$$\frac{\partial z}{\partial x} = G_x =$$

-1	0	1
-2	0	2
-1	0	1

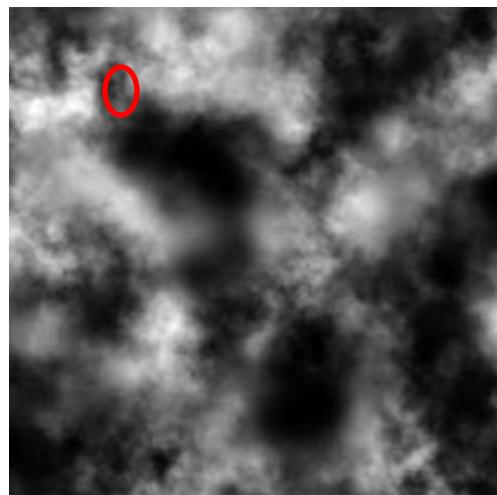
\* Image

$$\frac{\partial z}{\partial y} = G_y =$$

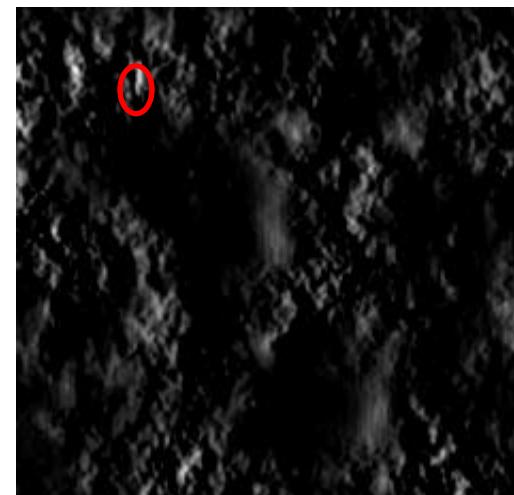
1	2	1
0	0	0
-1	-2	-1

\* Image

- Sobel Filter ကရလာသော Image များသည် Gradient of an Image ဖြစ်ပါသည်။ ဒုအပြင် ထို Gradient သည် Image တစ်ခု၏ Edge များကို ပြောပါသည်။
- စဉ်းစားကြည့်ပါ။ Image တစ်ခု၏ Gradient သည် Edge များအနားမှာ အကြီးဆုံးဖြစ်မည်၊ ထို့ကြောင့် တန်ဖိုးအကြီးဆုံး (255) ဖြစ်မည်။ Image တစ်ခု၏ Gradient သည် အပြန်များတွင် အသေးဆုံး ဖြစ်မည်။ ထို့ကြောင့် တန်ဖိုးအသေးဆုံး (0) ဖြစ်မည်။



Height Map



Gradient Map

# Sobel ( $G_x$ ) Kernel

- ကျွန်ုတ်တို့ ဒီလိုဆိုရင် Gradient of the Image in the direction of X ပုံကိုမြင်ရမှာပါ။

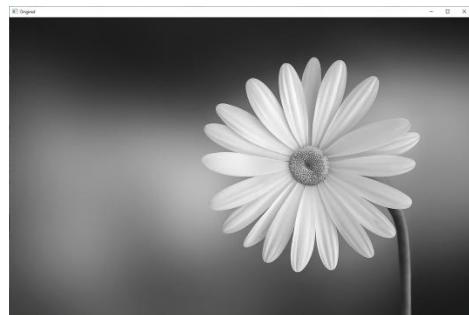
```
import cv2
import numpy as np

img = cv2.imread('./images/flower.jpg', cv2.IMREAD_GRAYSCALE)

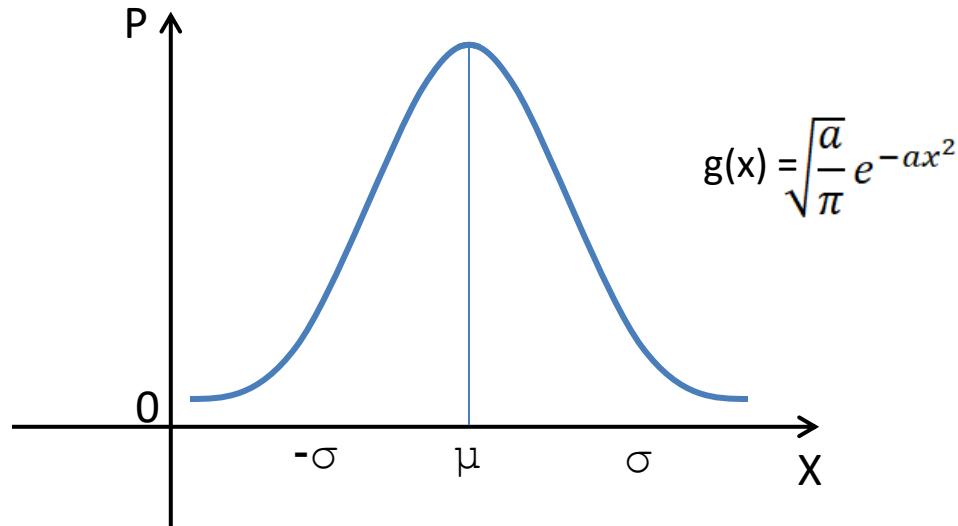
rows, cols = img.shape[:2]
kernel_sobel = np.array([[-1,0,1], [-2,0,2], [-1,0,1]])

cv2.imshow('Original', img)
output = cv2.filter2D(img, -1, kernel_sobel)
cv2.imshow('Sobel filter', output)

cv2.waitKey()
```



# Gaussian Function



- Gaussian Function ကို ကျွန်တော်တိ Probability and Statistics မှာတွေခဲ့ဖူးပါသည်။ Gaussian Function ကိုအခြားနေရာများစွာမှာလည်း သုံးပါသည်။
- Signal Processing တွင်မူ Gaussian Function ကို Impulse Response  $H(t)$  အနေဖြင့်ယူဆကြပါသည်။
- ထိုကြောင့် Gaussian Function ကို Spatial Convolution အနေဖြင့်လဲသုံးပါသည်။ Gaussian Function သည် Normal Distribution ဖြစ်သည့်အတွက် Pixel ၏ Average များကို ရှာပေးပါသည်။

# Gaussian Filter

$$G = \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} * \text{Image}$$

- Gaussian Filter မှရလာသောပုံများသည် Blur (Smooth) ဖြစ်သွားသည်။ ထို့ကြောင့် Gaussian Filter ကို Camera ရှိ Beauty Filter များမှာ သုံးပါသည်။ (မိန်းကလေးများ Gauss ကြီးကို ကျေးဇူးတင်သင့်ပါသည်။)
- Gaussian Filter များသည် High Frequency Component (အစက်အပျောက်) များကို ဖယ်ရှားလိုက်သည့် အတွက် သူ့ကို Low Pass Filter လိုလည်း ခေါ်ပါသည်။ ဒါကိုတော့ ကျွန်တော်တို့ Frequency Domain မှာ ထပ်ပြောပါမည်။

Gaussian ကြီးလိမ်းပြီး 3 စက်နဲ့ အကြောတွင်



Before

After

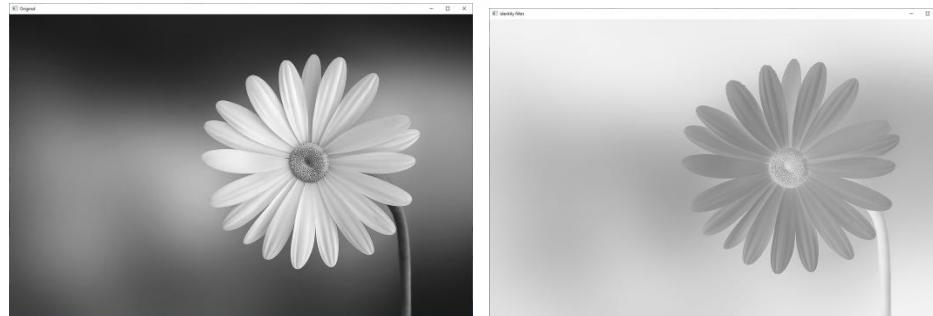
# Difference of Guassian

- Difference of Guassian වියේ තොර්තොර්ඩාව්ස් යෝජන කිරීමෙහි Function තුළු ප්‍රක්ෂේප වියේ සෑයුරු ඇතුළත් නොවැනු ඇතුළත් වූ.

```
import cv2
import numpy as np

img = cv2.imread('./images/flower.jpg', cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape[:2]
kernel_g1= np.array([[0,0,0], [0,0.3,0], [0,0,0]])
kernel_g2= np.array([[0,0,0], [0,0.8,0], [0,0,0]])
cv2.imshow('Original', img)
output1 = cv2.filter2D(img, -1, kernel_g1)
output2 = cv2.filter2D(img, -1, kernel_g2)
output = output1 - output2
cv2.imshow('DoG filter', output)
cv2.waitKey()
```



# Laplacian Function

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$$

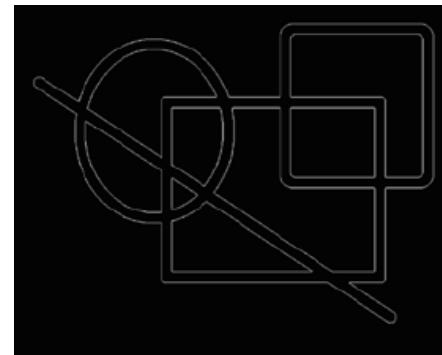
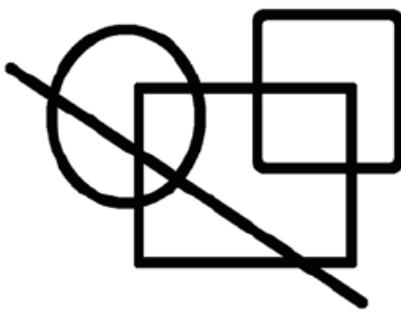
$$\Delta f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}$$

- Laplacian သည် အမှန်တော့ Divergence of the Gradient of a function ဖြစ်သည်။ တစ်နည်းအားဖြင့် Laplacian သည် Linear Combination of Second Order Partial Derivative ဖြစ်သည်။
- ဒါဆိုရင် Grayscale Image တစ်ခုရဲ့ Laplacian ဆိုရင် ဘယ်လို အဓိပ္ပာယ်ရမလဲ။ ကျွန်တော်တို့ Sobel Filter ကို 2 ခု Apply လုပ်ပြီး ပေါင်းထားခြင်း ဖြစ်သည်လို့ ဆိုနိုင်ပါသည်။
- သို့သော် Laplacian သည် Highly Sensitive to High Frequency Components (အစက်အပျောက်) ဖြစ်သည့်အတွက် Laplacian ကို Gaussian Filter ဖြင့် တွဲသုံးလေ့ရှိပါသည်။ Gaussian Filter ကို Apply လုပ်ပြီးမှ Laplacian သုံးလေ့ရှိပါသည်။ ဒါကို LoG (Laplacian of Gaussian) ဟူခေါ်ပါသည်။
- အမှန်တော့ အသုံးအများဆုံး Edge Detector ၏ Canny Detector ဖြစ်သည်။

# Laplacian Filter

$$L = \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} * \text{Image}$$

- Laplacian သည် High Frequency Components များနှင့် Random Noise များမပါလျှင် Edge Detect လုပ်ဖို့ အင်မတန်သင့်တော်ပါသည်။



# Canny and Laplacian

- ကျွန်ုပ်တော်တို့ Laplacian နှင့် Canny Filter တို့၏ လုပ်ဆောင်ချက်များကို တွေ့ရှိနိုင်ပါသည်။

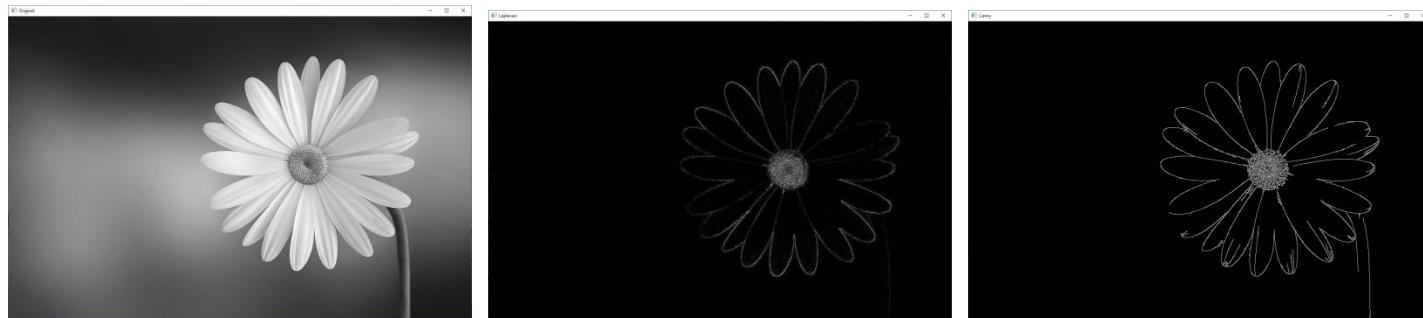
```
import cv2
import numpy as np

img = cv2.imread('./images/flower.jpg', cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape[:2]
kernel_1= np.array([[0,-1,0], [-1,4,-1], [0,-1,0]])

cv2.imshow('Original', img)
output1 = cv2.filter2D(img, -1, kernel_1)
output2 = cv2.Canny(img, 50, 240)
cv2.imshow('Laplacian', output1)
cv2.imshow('Canny', output2)

cv2.waitKey()
```



# Binary Image

- အခါထိ ကျွန်တော်တို့ ပြောခဲ့သော Image များတွင် Color Image, Grayscale Image များသာ ဖြစ်ပါသည်။ တစ်ခါတစ်လေ Binary Image များကိုလည်း အသုံးလိုပါသည်။ ဥပမာ၊ Image Segmentation များ၊ Morphology များ တွင် Binary Image များသည် အသုံးဝင်ပါသည်။
- ထိုအပြင် Binary Image များကို Connected Region များရှာဖွေရာတွင်လည်း အသုံးဝင်ပါသည်။ Binary Image များကို Thresholding လုပ်ခြင်းဖြင့်လည်း ရရှိပါသည်။

Color



Grayscale



Binary

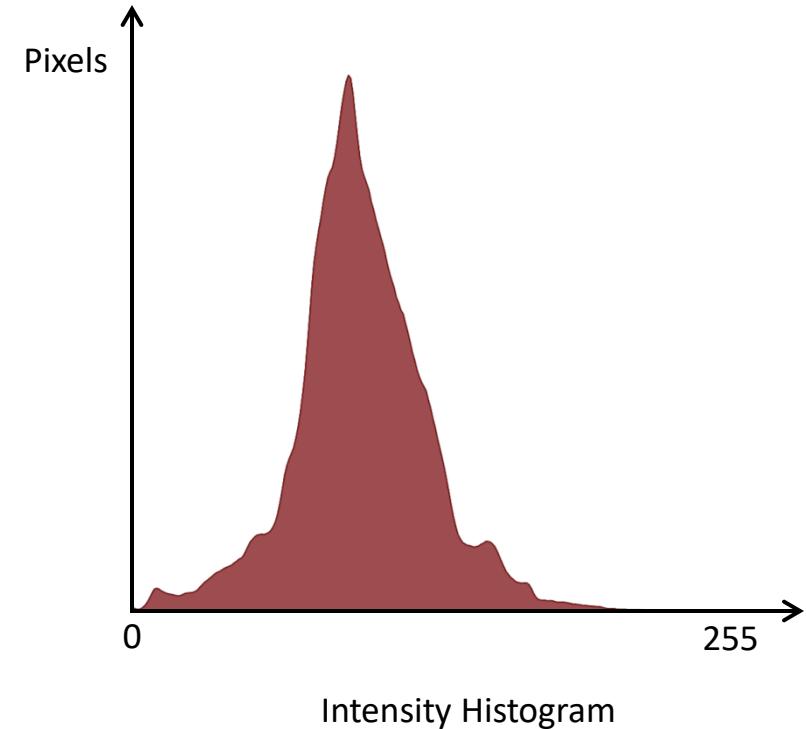


# Image Histogram

- Image Histogram သည် Image များရဲ့ Intensity များ၏ Pixel အရေအတွက် ဖြစ်သည်။ Image Histogram သည် အင်မတန် အသုံးများပါသည်။
- Image Histogram သည် Image တစ်ခု၏ Intensity Distribution ကို ဖော်ပြပါသည်။



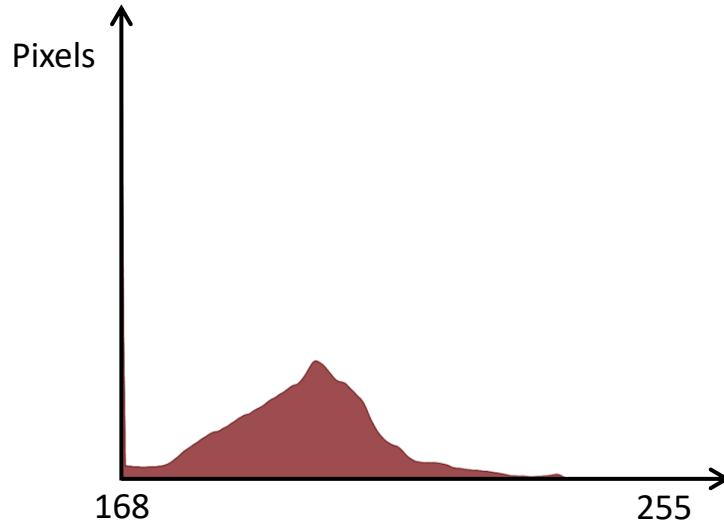
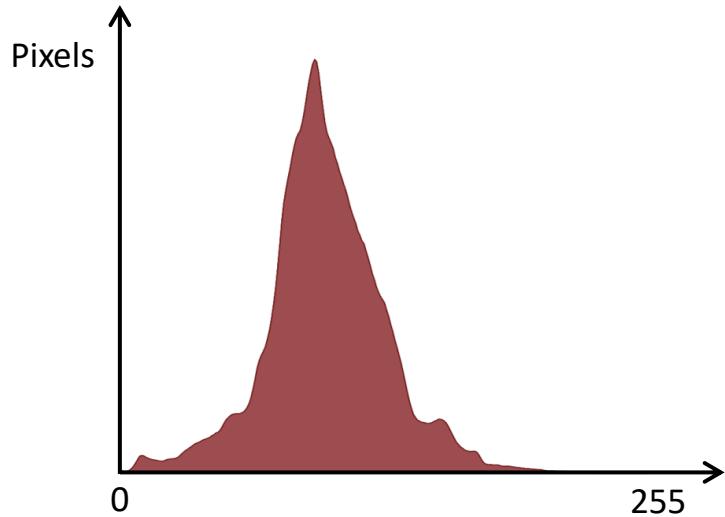
Grayscale



Intensity Histogram

# Intensity Thresholding

- Image Intensity များသည် အလင်းအမှာင်ပေါ်မှတည်ပါသည်။ ထိုကြောင့် Image Intensity များကို Threshold လုပ်ခြင်း (လိုအပ်သော Intensity များကိုသာ ယူဖြီးကျန်တဲ့ Intensity များကို Interpolate (Cut off) လုပ်ခြင်း) ဖြင့် လိုအပ်သော အလင်းအမှာင် Component များကို ရပါသည်။



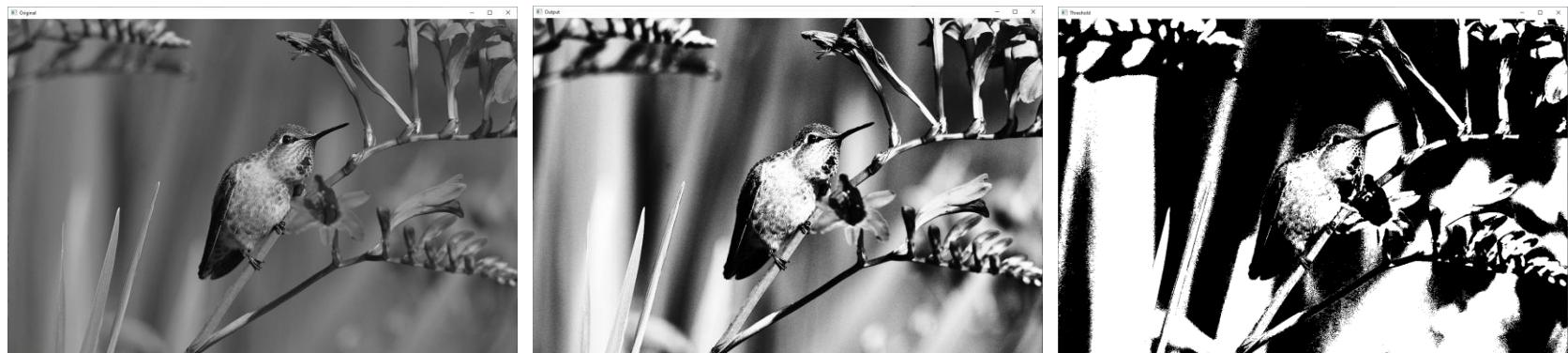
# OTSU Thresholding

- အများအားဖြင့် Thresholding ကို Binary Image များအနေဖြင့် ပြောင်းရှာတွင်လည်း သုံးပါသည်။ Thresholding Technique များထဲမှ OTSU Method ကို စမ်းကြည့်ပါမည်။

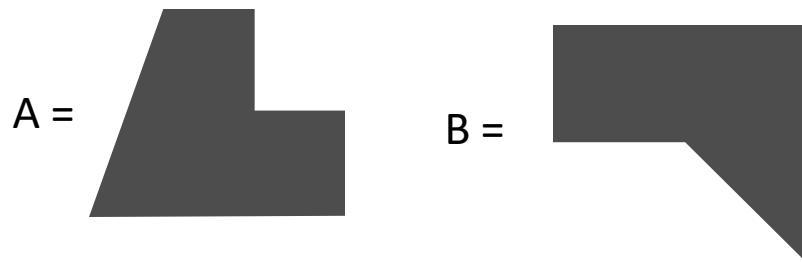
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('./images/bird.jpg', cv2.IMREAD_GRAYSCALE)
(T, thresh) = cv2.threshold(img, 100, 255, cv2.THRESH_OTSU);
hist = cv2.calcHist([img], [0], None, [255], [0,255])
output = cv2.equalizeHist(img)
plt.hist(img.ravel(),256,[0,256]); plt.show()
cv2.imshow('Original', img)
cv2.imshow('Output', output)
cv2.imshow('OTSU', thresh)

cv2.waitKey()
```



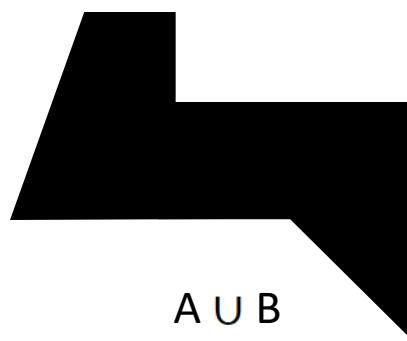
# Morphology



- Morphology သည် Set Theory အပေါ် အခြေခံပါသည်။
- တကယ်လို့ Binary Image တစ်ခုကို 2 Dimensional Binary Set Region တစ်ခု အဖြစ်ယူဆပါက



$A \cap B$   
A AND B

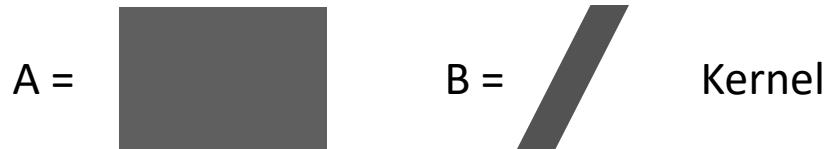


$A \cup B$   
A OR B



$A - B$   
A AND NOT B

# Dilation



- Dilation သည် Spatial Convolution နှင့် ဆင်တူပါသည်။ Region B ကို Region A အလိုက် တိုးလိုက်ခြင်းဖြစ်သည်။



- Dilation ကို Grayscale Image များမှာလည်း သုံးလို့ ရပါသည်။ Dilation ကိုသုံးပြီး စာလုံးများကို ကိုပြီးထင်ရှားအောင် လုပ်လို့ ရပါသည်။

# Erosion

$$A = \begin{matrix} \text{[Gray Square]} \end{matrix} \quad B = \begin{matrix} \text{[Dark Gray Vertical Bar]} \end{matrix} \quad \text{Kernel}$$

- Erosion သည် Dilation နှင့် ဆန်ကျင့်ဖက် ဖြစ်ပါသည်။ Region B ကို Region A အလိုက် နှုတ်လိုက်ခြင်း ဖြစ်သည်။

$$A \ominus B = \begin{matrix} \text{[Gray Square]} \\ \text{[Dark Gray Vertical Bar]} \\ \text{[Gray Square]} \end{matrix} = \begin{matrix} \text{[Black Square]} \end{matrix}$$

- Erosion ကို Grayscale Image များမှာလည်း သုံးလို ရပါသည်။ Erosion ကိုသုံးပြီး စာလုံးများ၏ Skeleton ကိုရှာလို ရပါသည်။

# Dilation and Erosion

- ကျွန်ုပ်တော်တို့ Erosionနှင့် Dilationတို့၏ လုပ်ဆောင်ချက်များကို တွေ့ရှိနိုင်ပါသည်။ တစ်ခါတလေ မသဲကွဲသော စာလုံးများကို Dilationသို့မဟုတ် Erosionလုပ်ခြင်းဖြင့် ထင်ရှားအောင်လုပ်နိုင်ပါသည်။

```
import cv2
import numpy as np

img = cv2.imread('./images/mmcursivetxt.jpg', cv2.IMREAD_GRAYSCALE)

kernel = np.ones((5,5), np.uint8)

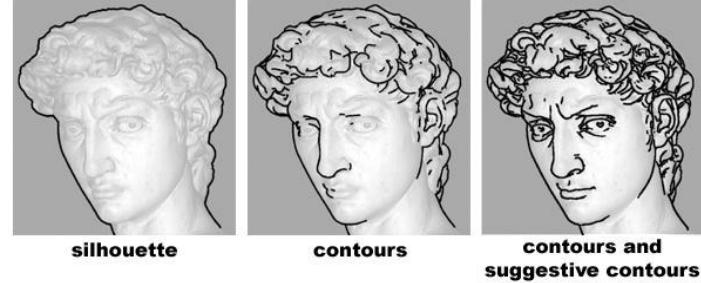
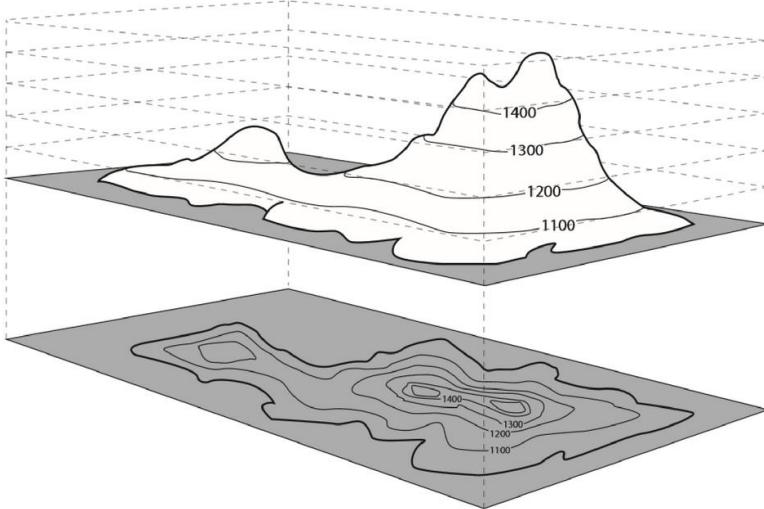
output1 = cv2.erode(img, kernel, iterations=1)
output2 = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Original', img)
cv2.imshow('Erosion', output1)
cv2.imshow('Dilation', output2)

cv2.waitKey()
```



# Image Contours



$$f(x_1, x_2, x_3, \dots, x_n) = k = \text{constant}$$

- အမှန်တော့ Contour များသည် အမြင့်တူသော Curve များ၏ Region တစ်ခု ဖြစ်ပြီး ထို Region ၏ Shape ကို Boundary ဖြင့် Define (သတ်မှတ်) လုပ်ပါသည်။
- Advanced Calculus မှာ ပြောခဲ့ပြီးသည့်အတိုင်း Contours နှင့် Gradient များသည် အမြဲ Orthogonal ဖြစ်ပါသည်။
- Contours များကို သုံးပြီး Image များ (အထူးသဖြင့် Binary Image များ) ၏ Shape ကို သိနိုင်ပါသည်။ ဒီအချက်သည် နောက်ပိုင်း အဆင့်မြင့် Recognition များလုပ်ရာတွင် အင်မတန် အရေးကြီးလာပါသည်။

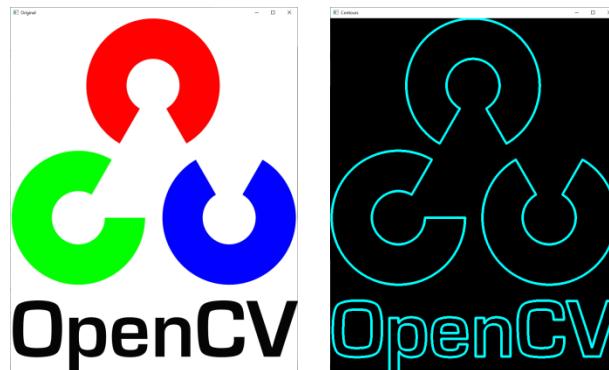
# Contours

- Contours များသည် အဆင့်ဆင့် (အတွင်း၊ အပြင်) ထပ်နေသည့် အတွက် OpenCV တွင် Contour Hierarchy ရှိပါသည်။ အပြင်အကျခုံး Contour သည် Hierarchy အမြင့်ဆုံး ဖြစ်သည်။

```
import cv2
import numpy as np

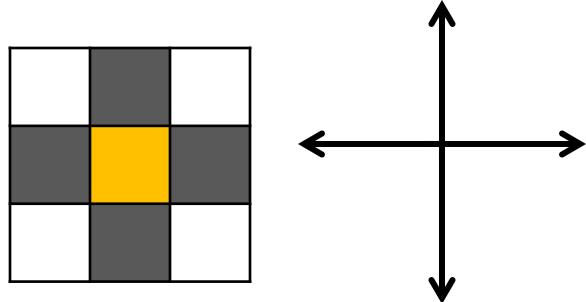
input = cv2.imread('./images/logo.png')

img = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)
(ret, thresh) = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours
(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
blank = np.zeros((input.shape[0],input.shape[1],3), np.uint8)
output= cv2.drawContours(blank, contours, -1, (255, 255, 0), 5)
cv2.imshow('Original', input)
cv2.imshow('Contours', output)
cv2.waitKey()
```

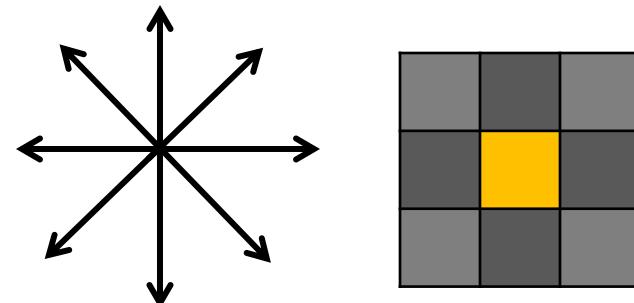


# Connected Components

- Connected Components များသည် Contours များအပေါ် အခြေခံပါသည်။ တစ်နည်းအားဖြင့် Contour များက Define လုပ်ထားသော Region များသည် တစ်ခုနှင့် တစ်ခု ဘယ်လို ဆက်စပ်နေသည်ကို ဖော်ပြပါသည်။
- Connected Components များသည် Object Detection လုပ်ရာတွင်လည်းကောင်း၊ Object Recognition လုပ်ရာတွင် လည်းကောင်း အင်မတန် အရေးပါပါသည်။
- Object Detection သည်အများအားဖြင့် Foreground Detection ဖြစ်သည်။ Background Detection များကိုတော့ Image Segmentation မှာ ထပ်ပြောပါမည်။
- အများအားဖြင့် Component များ၏ Connection ကို 4 (Neighborhood) Direction သို့မဟုတ် 8 (Neighborhood) Direction ဖြင့် Define လုပ်ပါသည်။



4 (Neighborhood) Direction



8 (Neighborhood) Direction

# Object Detection

- ကျန်တော်တို့ Connected Component များကို သုံးပြီး Myanmar License Plate များကို Detect လုပ်ကြည့်မည် ဖြစ်သည်။ အခုက် Image Detection သာ ဖြစ်ပါသည်၊ Image Recognition ရှိကတော့ Image Classification with Machine Learning လိုပါသည်။
- Example က ဒီကရပါသည်။ <https://www.pyimagesearch.com/2021/02/22/opencv-connected-component-labeling-and-analysis/>

```
import cv2
import numpy as np

input = cv2.imread('./images/mplate.jpg')
img = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)

(ret, thresh) = cv2.threshold(img, 180, 255, cv2.THRESH_OTSU);

#use 8 Directional Connected Components
(numLabels, labels, stats, centroids) = cv2.connectedComponentsWithStats (thresh, 8
, cv2.CV_32S)
```

# Object Detection

```
# loop over the number of unique connected component labels
for i in range(0, numLabels):

    x = stats[i, cv2.CC_STAT_LEFT]
    y = stats[i, cv2.CC_STAT_TOP]
    w = stats[i, cv2.CC_STAT_WIDTH]
    h = stats[i, cv2.CC_STAT_HEIGHT]
    area = stats[i, cv2.CC_STAT_AREA]
    (cX, cY) = centroids[i]

    output = input.copy()
    cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 3)
    cv2.circle(output, (int(cX), int(cY)), 4, (0, 0, 255), -1)

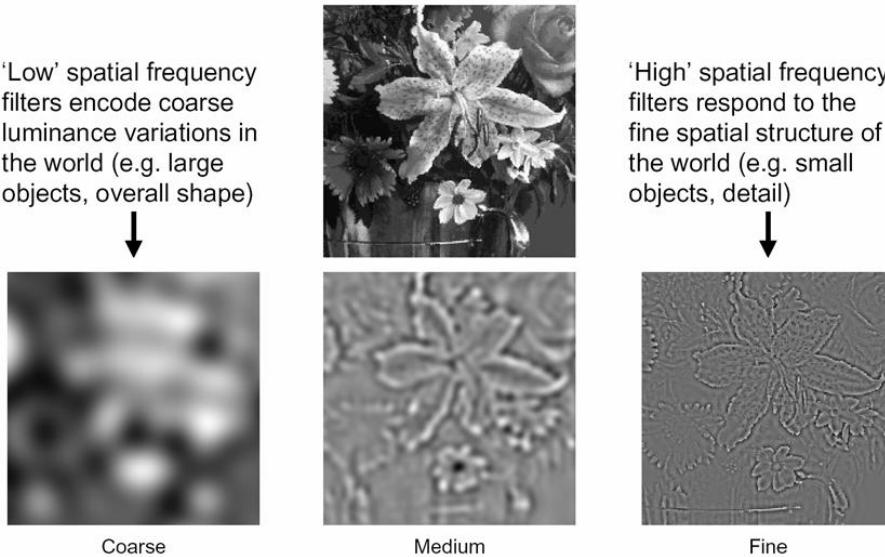
    componentMask = (labels == i).astype("uint8") * 255
    # show our output image and connected component mask
    cv2.imshow("Output", output)
    cv2.imshow("Connected Component", componentMask)
    cv2.waitKey(0)
```

# Object Detection



# Frequency Domain Analysis

- ကျွန်တော်တိ အခုအထိ Image ရဲ Space Domain ကို အဓိကထားပြီး ဆွဲးနွေးခဲ့ပါသည်။
- ဒီတစ်ခါတော့ Image ကို Frequency Domain မှလေ့လာကြည့်ကြပါစို့။
- အမျှန်တော့ Color Image များကို Light Spectrum များ၏ Intensity များဖြင့် ဖော်ပြထားခြင်း ဖြစ်သည်။ ထို့ကြောင့် Frequency Component များသည် အရေးကြီးသော အခန်းမှ ပါဝင်ပါသည်။
- တစ်ကျွန်တော်တိ Grayscale Image ကို ကျွန်တော်တိ Spatial Wave များအနေဖြင့် ယူဆလို့ရသည့်အတွက် ထိ Spatial Wave များ၏ Frequency Components များသည်လည်း အရေးကြီးပါသည်။



# Fourier Transform

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt$$

- Signal Processing မှာ ကျွန်တော်တိ Fourier Transform အကြောင်းကို ပြောခဲ့ပါသည်။
- အမှန်တော့ Fourier Transform တွင် Continuous Time Signal များကို Transform လုပ်သော Fourier (FT) Transform သာမက Continuous Time Signal များကို Sample လုပ်ထားသော Discrete Time Signal များကိုပါ Transform လုပ်သော Discrete Time Fourier Transform (DTFT) လည်းရှိပါသည်။
- ကျွန်တော်တိသည် Digital Signal (Digital Sound, Digital Image) များကို Process လုပ်နေသည်ဖြစ်၍ Discrete Time Fourier Transform ကိုသုံးရပါသည်။ သို့သော DTFT ၏ Fourier Spectrum သည် အများအားဖြင့် Continuous Spectrum ဖြစ်သည့်အတွက် လက်တွေ့အားဖြင့် အဆင်မပြေပါ။ ဒါကြောင့် DTFT ကို Approximate လုပ်ထားသော Discrete Spectrum ရှိသည့် Discrete Fourier Transform (DFT) ကိုယုံးကြပါသည်။
- Discrete Fourier Transform (DFT) ကို Implement လုပ်ထားသော Algorithm များစွာ ရှိပါသည်။ ထို Algorithm များကို Fast Fourier Transform (FFT) ဟူခေါ်ကြပါသည်။
- ကျွန်တော်တိ Image များကို Frequency Domain သို့ Transform လုပ်သည့်အခါ DFT ကို သုံးကြပါသည်။

# Discrete Time Fourier Transform

$$X(j\omega) = \sum_{-\infty}^{\infty} (x(nT_s)e^{-j\omega nT_s}) \times T_s$$

$$X(e^{j\hat{\omega}}) = \sum_{-\infty}^{\infty} x[n]e^{-j\hat{\omega}n}$$

- Discrete Time Signal များသည် Continuous Time Signal များကို Sample Rate ( $F_T = 1/T_s$ ) ဖွင့် Sample လုပ်ထားခြင်းဖြစ်သည်။
- ထိုကြောင့် DTFT သည် Discrete Time Signal များ၏ Fourier Transform ဖြစ်သည်။ DTFT တွင် Continuous Frequency Spectrum ရှိပါသည်။
- ထိုကြောင့် လက်တွေ့ တွက်ဖို့ (အထူးသဖွင့် Computer များဖြင့် တွက်ဖို့) အဆင်မပြေပါ။

# Discrete Fourier Transform

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j \frac{2\pi}{N} kn}$$

- Discrete Time Fourier Transform (DTFT) ကို Discrete Fourier Transform (DFT) အနေဖြင့် Reduce လုပ်လို့ရပါသည်။
- Discrete Time Signal များသည် Periodic ဖြစ်ပါသည် (Sample Rate ဖြင့် Sample လုပ်ထား၍ ဖြစ်သည်)။
- ထိုကြောင့် Discrete Time Signal များမှ N Samples (Enough Samples) ကို ယူလိုက်မည့်ဆိုလျှင် Signal ၏ Information အားလုံးကို သိနိုင်မည် ဖြစ်သည်။
- Discrete Fourier Transform (DFT) သည် Discrete Time Fourier Transform (DTFT) များမှ N Samples များ ပြန်ယူထားခြင်း ဖြစ်သည်။ ထိုကြောင့် N Samples များ၏ Frequency Component များသာ ပါဝင်တော့သည်။
- ထိုကြောင့် Signal Length ရှည်လေ၊ Frequency Resolution ပိုကောင်းလေ ဖြစ်သည်။

# DFT with Numpy

- DFT କି ଗ୍ରହଣତ୍ତ୍ଵ ନାମରେ Numpy ପ୍ରଦାନ କରିଛି।

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

input = cv2.imread('./images/logo.png')

img = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
cv2.waitKey()
```

Input Image



Magnitude Spectrum



# DFT with OpenCV

- DFT ကို ကျန်တော်တို့ OpenCV ဖြင့် လုပ်ကြည့်ပါမည်။

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

input = cv2.imread('./images/logo.png')

img = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)
dft = cv2.dft(np.float32(img), flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log (cv2.magnitude(dft_shift[:,:,:0],dft_shift[:,:,:1]))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
cv2.waitKey()
```

Input Image



Magnitude Spectrum



# IDFT with OpenCV

- IDFT (Inverse DFT) ကို ကျွန်တော်တို့ Numpy ဖြင့် လုပ်ကြည့်ပါမည်။ ကျွန်တော်တို့ High Frequency များကို ဖယ်လိုက်သည့်အတွက် Blur ဖြစ်သွားမည် ဖြစ်သည်။

```
rows, cols = img.shape
crow,ccol = math.ceil(rows /2) , math.ceil(cols/2)
# create a mask first, center square is 1, remaining all zeros
mask = np.zeros((rows,cols,2),np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1
# apply mask and inverse DFT
fshift = dft_shift*mask
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0],img_back[:, :, 1])
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
cv2.waitKey()
```

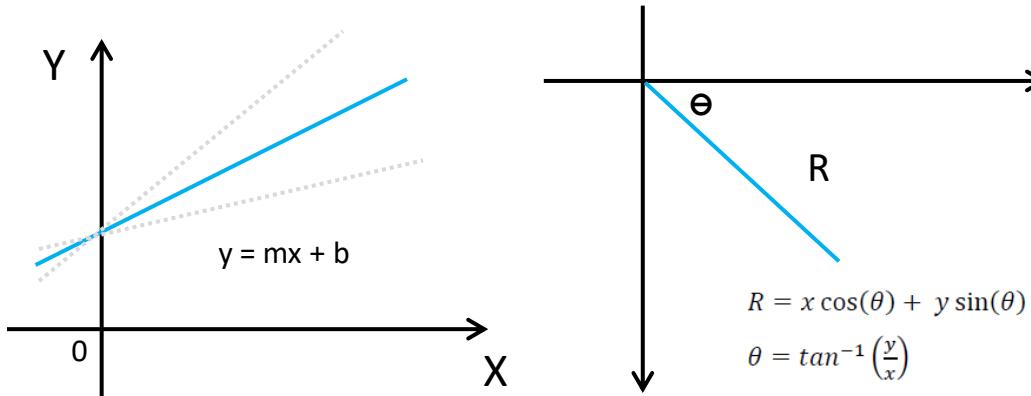
Input Image



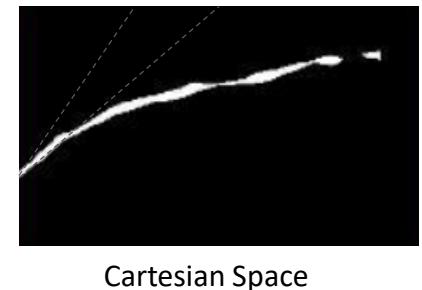
Magnitude Spectrum



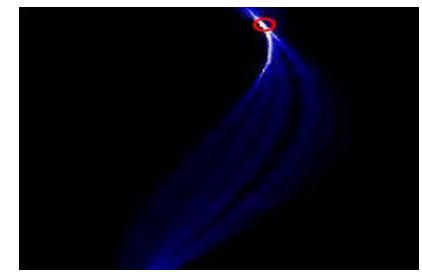
# Hough Transformation



- ကျွန်တော်တို့သည် Image တစ်ခုရှိ Point များကို ဖြတ်သွားသော Line များကို Hough Transform ဖြင့် ဖော်ပြပါက Point ( $R, \Theta$ ) များ အနေဖြင့် ရရှိမည် ဖြစ်သည်။
- Hough Space တွင် အများဆုံးဆုံးသော Point သည် Image တစ်ခုရှိ Point များဆုံးကို ဖြတ်သွားသော Line တစ်ခုဖြစ်မည် ဖြစ်သည်။
- ထိအချက်ပေါ်မူတည်ပြီး ကျွန်တော်တို့ Line Detection ကိုလုပ်ဆောင်လို ရပါသည်။
- Hough Transform သည် အင်မတန် အသုံးဝင်သော Transformation တစ်ခုဖြစ်ပါသည်။



Cartesian Space



Hough Space

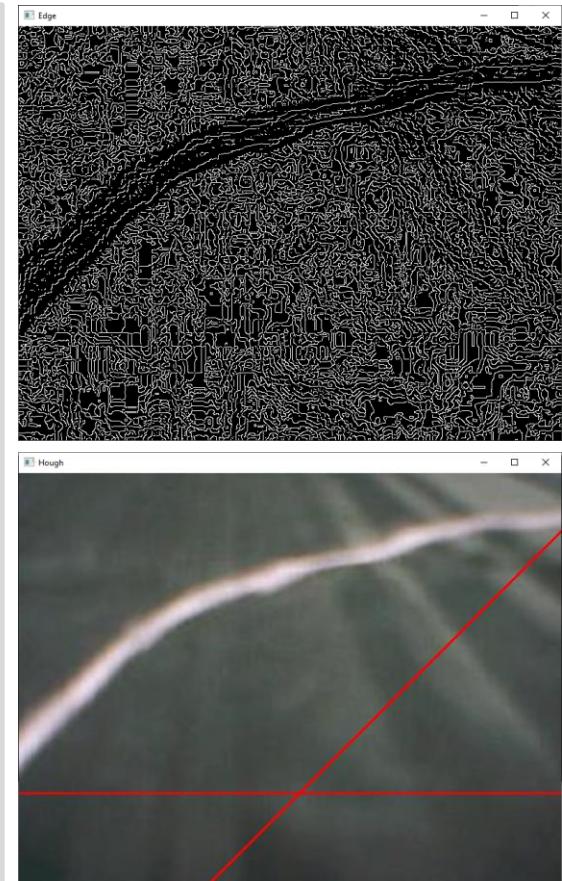
# Hough Transform

- ကျွန်တော်တိ Hough Transformation ကိုသုံးပြီးတော့ အဖူရောင် Curve ၏ Curvature ကို တွက်ကြည့်ပါမည်။ Line 2 ခုကြားထဲက Angle သည် Curvature ဖြစ်သည်။

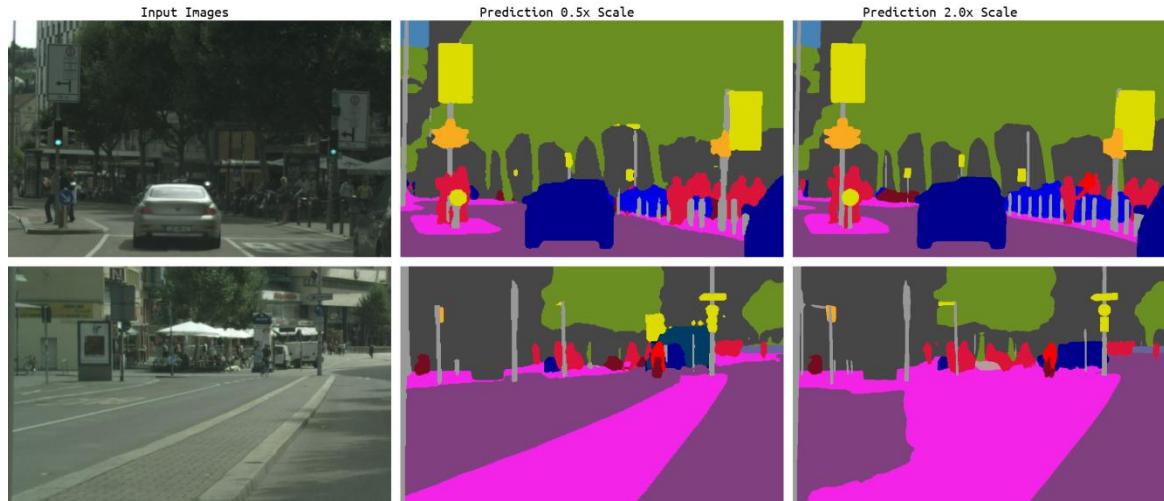
```
import cv2
import numpy as np

input = cv2.imread('./images/line.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 0, 255, apertureSize = 7)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
r = 0
for line in lines:
    rho,theta = line[0]
    if (rho - r > 150):
        r = rho
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(img, (x1,y1), (x2,y2), (0,0,255), 2)

cv2.imshow('Hough', img)
cv2.waitKey()
```

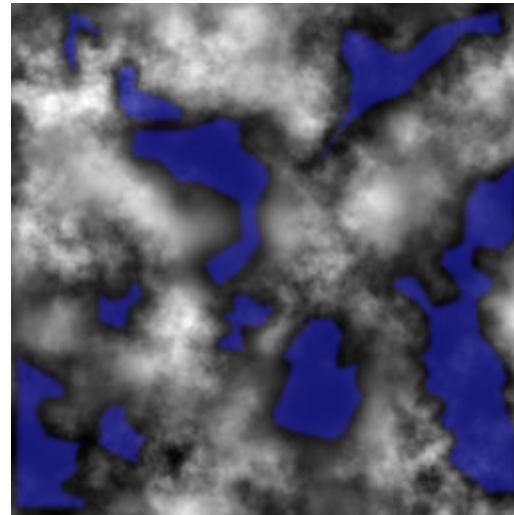
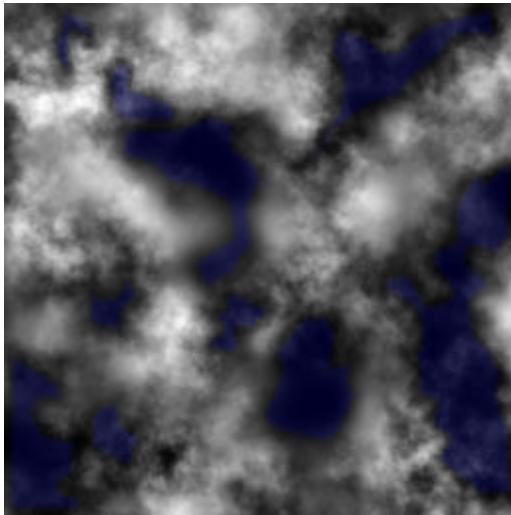


# Image Segmentation



- ကျွန်ုတ်တို့ Connected Components များကို သုံးပြီးတော့ Object Detection လုပ်လိုဂေကြာင်း ပြောခဲ့ပါသည်။
- အမှန်တော့ Image Segmentation သည် ပိုပြီးတော့ General ဖြစ်သော Method ဖြစ်ပြီး Computer Vision မှာ အသုံးအများဆုံးဖြစ်ပါသည်။ Image Segmentation လုပ်သော နည်းစနစ်ပေါင်းများစွာ ရှိပါသည်။
- ထိုအထဲကမှ ကျွန်ုတ်တို့ အခြေခံအကျဆုံး ဖြစ်သော Watershed Method ကိုစမ်းကြည့်ပါမည်။

# Watershed Segmentation



- ကျွန်တော်တိ Terrain တစ်ခကို Flood လုပ်လိုက်မည်ဆိုလျှင် ခီ၏၏၏များတွင် ရေပြည့်သွားမည် ဖြစ်သည်။ တစ်နည်းအားဖြင့် ရေများသည် Local Minimum Region များသို့ စီးဝင်သွားမည် ဖြစ်သည်။ Watershed Segmentation သည် ဒါ Idea အပေါ် အခြေခံထားပါသည်။
- Local Minimum Region များကို အများအားဖြင့် Low Intensity Mask များဖြင့် Define လုပ်ပါသည်။ Low Intensity Mask များကို Morphological Operation များလုပ်ခြင်းဖြင့် ရရှိမည် ဖြစ်သည်။ ထိုနောက် Foreground နှင့် Background ကို ခွဲပါသည်။ ထိုစပ်နေသော Region များကို Distance Transform မှတည်ပြီး ခွဲထုတ်ပါသည်။
- ထိုနောက် ရလာသော Mask ပေါ်မှတည်၍ Connected Components များပြန်ရှာသော အခါ Segmented Image ကိုရရှိမည် ဖြစ်သည်။

# Watershed Segmentation

- ကျွန်တော်တို့ Watershed ကိုသုံးပြီးတော့ Tennis ဘောလုံးများကို Segment လုပ်ကြည့်ပါမည်။

```
import cv2
import numpy as np

img = cv2.imread('./images/balls.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
(ret, thresh) = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

border = cv2.dilate(thresh, None, iterations=3)
border = border - cv2.erode(border, None)

# noise removal
kernel = np.ones((3,3),np.uint8)
closing = cv2.morphologyEx(border, cv2.MORPH_CLOSE, kernel, iterations = 2)

# sure background area
sure_bg = cv2.dilate(closing, kernel, iterations=1)

cv2.imshow('BG', sure_bg)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(closing, cv2.DIST_L2, 3)
ret, sure_fg = cv2.threshold(dist_transform, 0.001 * dist_transform.max(), 255, 0)
```

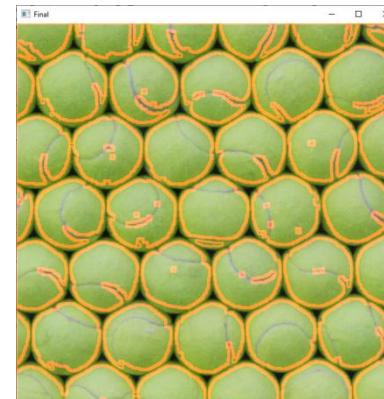
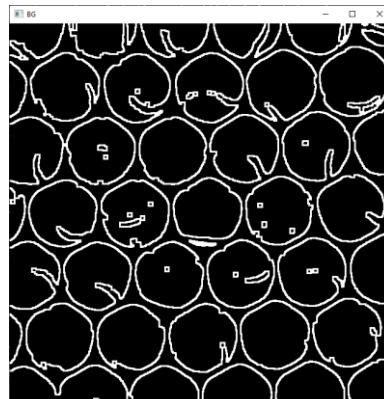
# Watershed Segmentation

```
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)

# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0

markers = cv2.watershed(img, markers)
img[markers == -1] = [0,0,255]

img = cv2.dilate(img, None)
cv2.imshow('Final', img)
cv2.waitKey()
```



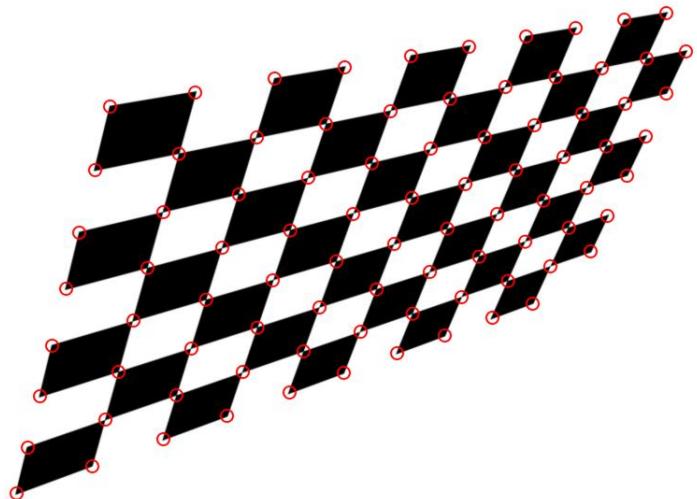
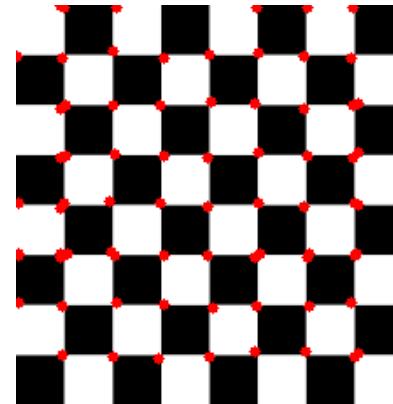
# Image Features



- ကျွန်တော်တိ ဒါ Image 3 ခုကို ကြည့်ပါ။ အရွယ်ရောက်ပြီးသော ယောက်၍သာ တစ်ယောက်ဖြစ်ကြောင်း၊ ပုံအားလုံးသည် တစ်ယောက်ထဲကိုပင် ဖော်ပြေကြောင်း သိပါသည်။
- ပုံတွေ တစ်ခုနှင့် တစ်ခုမတူဘဲနဲ့ ကျွန်တော်တိ ပုံအားလုံး အတူတူ ဖြစ်ကြောင်း ဘယ်လိုသိလဲ။
- အမှန်တော့ ကျွန်တော်တိ Image များ၊ Feature များကို Compare လုပ်ကြည့်လို သိတာ ဖြစ်ပါသည်။ ဒါဆိုရင် Image များ၊ Feature ဆိုတာ ဘာလဲ။
- အကြမ်းဖျဉ်းပြောရင် ကျွန်တော်တိ ပြောနိုင်တဲ့ Image များ၊ Feature ဆိုတာ ဆံပင်၊ မျက်လုံး၊ နာခေါင်း၊ မျက်နှာ၊ မျက်ခုံး၊ မေးစွဲ တို့ရဲ့ အရွယ်အစား၊ အချိုးအစား၊ တည်နေရာ ကိစ္စလိုတာ ဖြစ်ပါသည်။ အမှန်တော့ ကျွန်တော်တိ အရာဝါတ္ထာတစ်ခု၊ လူတစ်ယောက်ကို Recognize ဖြစ်သည်ဆိုတာ ထို Feature များကို Recognize ဖြစ်ခြင်းပင် ဖြစ်ပါသည်။
- ထိုအတူ Image များတူ၊ မတူ ဆိုတာလဲ ထို Feature များ ဘယ်လောက်တူလဲ ပေါ်မှုတည်ပါသည်။

# Image Features

- အမျန်တော့ Image တစ်ခု၏ Feature များကို အောက်ပါ အချက်များ  
ဖြင့် Define လုပ်ပါသည်။
  - Line နှင့် Edge များ
  - Contour နှင့် Region များ
  - Corner များ
- သို့သော Image များသည် အလင်းအမြှင်၊ Geometric Transformation (ဥပမာ၊ Rotation) ပေါ်မှုတည်ပြီး Line နှင့် Edge များ၊ Contour နှင့် Region များသည် ပြောင်းသွားနိုင်ပါသည်။ ဥပမာ၊ Horizontal Line သည် Rotate လုပ်လိုက်လျှင် Vertical Line ဖြစ်သွားပါမည်။
- ထိုကြောင့် Image များ၏ Feature များကို Scale Invariant နှင့် Rotation Invariant စသည့် အချက်များပေါ်မှုတည်ပြီး ခွဲခြားကြပါသည်။
- Scale Invariant နှင့် Rotation Invariant ဖြစ်သော Feature သည် Corner များ ဖြစ်ပါသည်။ Image တစ်ခုကို Scale လုပ်လုပ်၊ Rotate လုပ်လုပ် Corner များသည် အမြဲ Corner များ ဖြစ်သည်။



# Feature Detection and Extraction

- အမျန်တော့ Image တစ်ခု၏ Feature များကို Detect, Extract, Match လုပ်နိုင်သော Algorithm များစွာ ရှိပါသည်။
  - Harris Detector ကို Corner များကို Detect၊ Extract လုပ်ဖို့သုံးပါသည်။
  - SIFT (Scale Invariant Feature Transform) ကို Blob (Region) များကို Detect၊ Extract လုပ်ဖို့သုံးပါသည်။
  - SURF (Speed Up Robust Features) ကို Blob (Region) များကို Detect၊ Extract လုပ်ဖို့သုံးပါသည်။
  - FAST (Features from Accelerated Segment Test) ကို Corner များကို Detect၊ Extract လုပ်ဖို့သုံးပါသည်။
- အခြား Feature Detectionနှင့် Extraction Method များ ရှိပါသေးသည်။
- Extract လုပ်ပြီးသော Feature များကို အောက်ပါအချက်များဖြင့် Define လုပ်ပါသည်။
  - pt
  - size
  - angle
  - response
  - octave
  - class\_id

# Harris Detector

- Shirt Image හි Corner මුදල් Harris Detect ලදී කෙතුවා මෙයි!!

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

input = cv2.imread('./images/shirt.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Gray', gray)

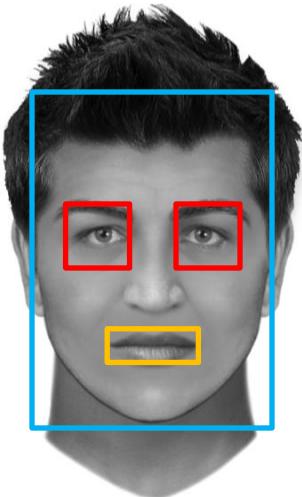
dst = cv2.cornerHarris(gray, 2, 15, 0.04)
img[dst > 0.01 * dst.max()] = [0, 255, 255]

img = cv2.dilate(img, None, 2)
cv2.imshow('Corners', img)

cv2.waitKey()
```



# Haar Cascade



- လူတစ်ယောက်၏ မျက်နှာကို Detect လုပ်ဖို့ ကျွန်တော်တို့ Haar Cascade ကို သုံးပါသည်။
- အမှန်တော့ Haar Cascade များသည် Feature Detector များကို ဆင့်ကာဆင့်ကာ သုံးပြီး အရေးကြီးသော Feature များကိုသာ ဆွဲထွက်သော Technique တစ်ခု ဖြစ်ပါသည်။
- OpenCV မှာ Object Detect လုပ်သော Haar Cascade မျိုးစုံရှိပါသည်။
- <https://github.com/opencv/opencv/tree/master/data/haarcascades>
- ထိုအပြင် Haar Cascade ကို Video နှင့် Live Camera များမှာလည်း သုံးနိုင်ပါသည်။

# Haar Cascade

- ကျွန်တော်တို့ Blackpink ၏ မျက်နှာများကို Detect လုပ်ကြည့်ပါမည်။

```
import cv2
import numpy as np

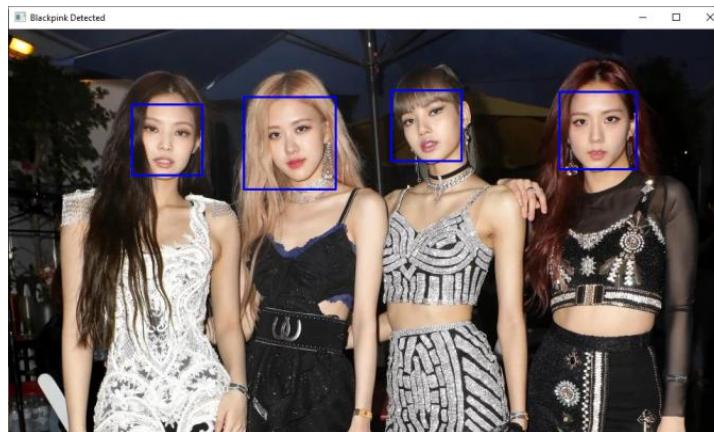
input = cv2.imread('./images/blackpink.jpg')
face_cascade = cv2.CascadeClassifier('./models/haarcascade_frontalface_default.xml')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:
    img = cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
cv2.imshow('Blackpink Detected', img)

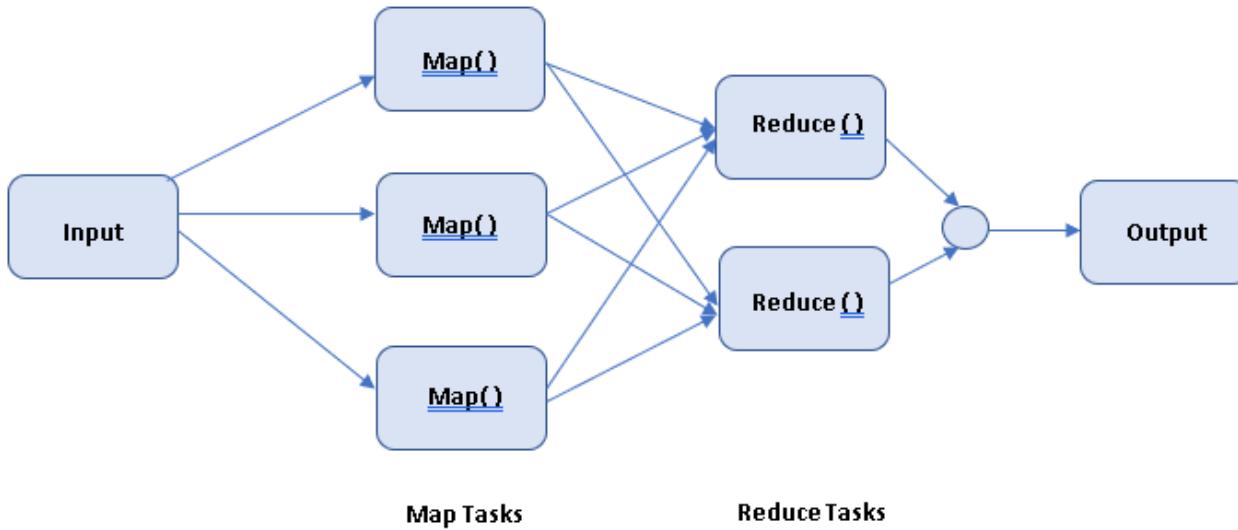
cv2.waitKey()
```



# Summary

- အခုခိုရင်တော့ ကျွန်တော်တို့ Image Processing ကို အခြေခံလောက်တော့ သိသွားမည် ဖြစ်သည်။
- အမှန်တော့ Image Processing ၊ အထူးသဖြင့် Signal Processing သည် အလွန်ကျယ်ဝန်း နက်ရှိင်းသော နည်းပညာနယ်ပယ် ဖြစ်ပါသည်။
- သို့လော် အခြေခံ Image Processing ကို သိလျှင် နောက်ပိုင်း AI နှင့် Machine Learning များလုပ်ဆောင်ရာမှာ ပိုပြီး လွယ်ကူလာပြီး ဘာကြောင့် ဘာလို့ ဒီလိုလုပ်ရသည်ကို သိလာမည် ဖြစ်သည်။
- တစ်ခု သတိထားဖိုက Image Processing သည် Computing Power အရမ်းလိုအပ်ပြီး Fast ဖြစ်ဖို့ လိုပါသည်။ ဆိုပါတော့ ကျွန်တော်တို့ 1024 x 860 Image မှာဆိုလျှင် Pixel ပေါင်း 880,640 ရှိပါသည်။ For Loop ကိုသုံးပြီး ကျွန်တော်တို့ Pixel တစ်ခုချင်းကို Update လုပ်ရင်တောင် 880,640 ခါလုပ်ရမည် ဖြစ်သည်။ ဒီလို Image ပေါင်း 1000 ဆုရင် Computing Power အရမ်းလိုပါသည်။
- ဒါကြောင့် Image Processing Library များကို C Language ဖြင့် ရေးလေ့ရှိပြီး အခြား Language များက Wrapper API အနေဖြင့် ခေါ်သုံးကြပါသည်။
- ဒီအပြင် ပိုမြန်အောင် Parallel Computing များကို အသုံးချဖို့ လိုပါသည်။ ဒါကြောင့် Map-Reduce ကို Appendix အနေဖြင့် ထည့်ထားပေးပါသည်။

# Appendix A – Map Reduce



- အခါ ကျွန်တော်တို့ Computer များသည် Multi-Core (More than 1 processor) ဖြစ်ပါသည်။
- ထိုကြောင့် Computer တစ်ခုမှာ Processor 10 ခုရှိရင် ပဲ 1,000 ကို Process လုပ်မည်ဆိုပါက Processor 1 ခုက ပဲ 100 လုပ်ခြင်းဖြင့် ကျွန်တော်တို့ 10 ဆ နီးပါး (Overhead Delay များရှိပါသည်) ပဲမြန်လာမည် ဖြစ်သည်။
- Map Reduce သည် Data Parallelism Pipeline အတွက် အင်မတန် အရေးကြီးပါသည်။