Introduction to openCV-Python

openCV was initailly developed in c++ a very fast compiled language as compared to the slow nature of python. But due to the simplicity of python and the ease of use a wrapper was developed for python around the original openCV c++ code. Which allowed the users to make the best of both world. A wrapper in simple terms means that the user will be writing the code in python but in the backend the code will be executed in c++ which made the code execution faster.

Installing openCV

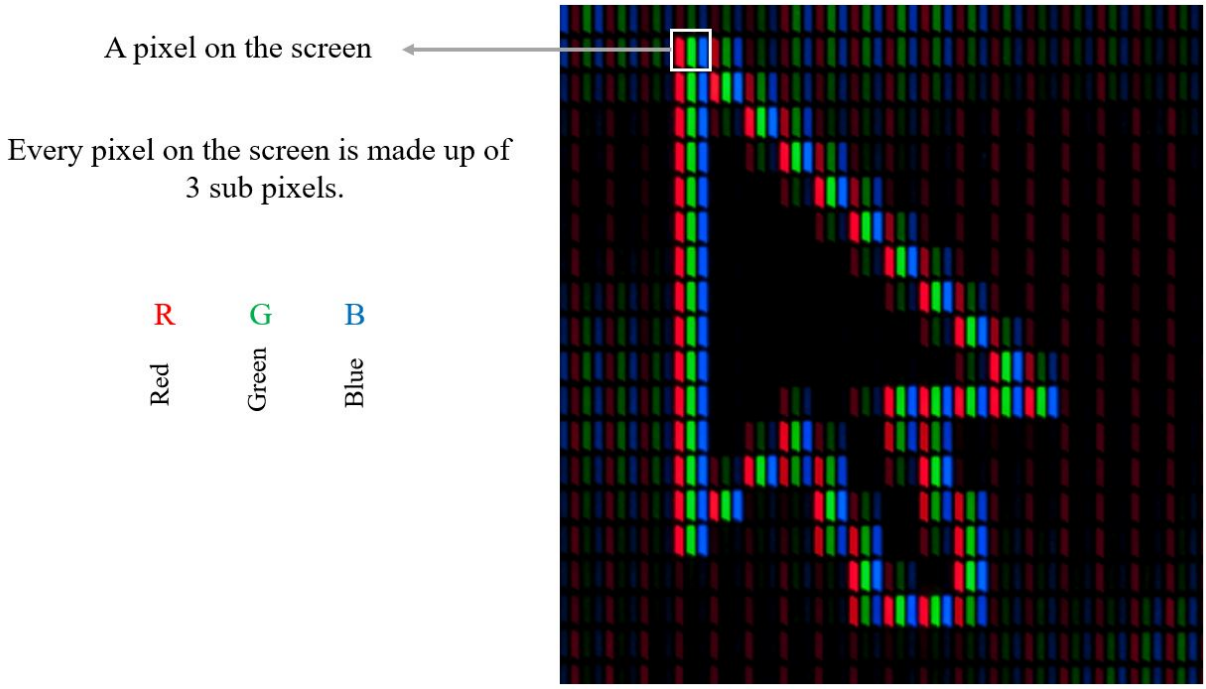
- for windows users
  `pip install opencv-python`

- for mac users
  `pip3 install opencv-python`

Understanding Images

Before we start with image processing we must understand what an image is.

Below Image shows a zoomed in image of a mouse pointer with white edges on the screen.

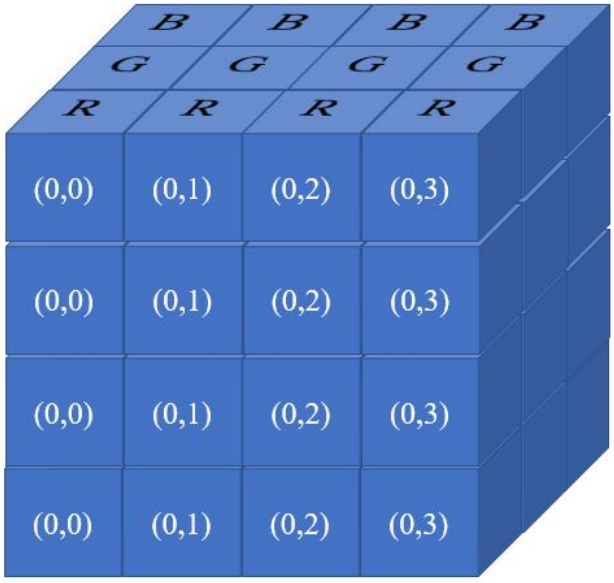


Wait but the edges of the mouse is not white!!
The white color is the result of the RGB sub-pixel colors mixing together as they are close to each other. Red Green and Blue being the primary colors we can create any color on the screen by manipulaing the brightness of each sub-pixel.

- A standard monitor supports 256 levels of brightness for every sub-pixel pixel. So by manipulating the RGB sub-pixels we can create a total number of 256^3 colors
- A subpixel brightness of 0 denotes that the subpixel is entierly turned off while a value off 255 denotes the subpixel is completely turned on.

You can use the below link to check out a RGB color picker which allows you to chose the RGB brightness to create a new color. https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html

An image on the screen can be represented using a 3D numpy array having the number of rows and columns equal to the height and the width of the image and the third dimension representing the RGB subpixel values

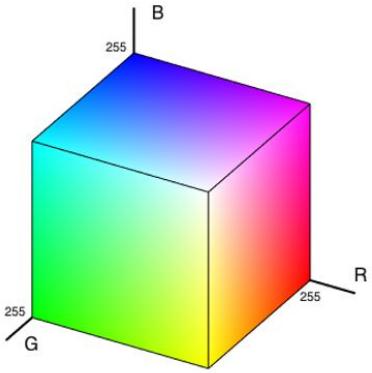


NOTE : OpenCV uses the BGR color model to stroe the image data. The BGR color model is converted to RGB just before it is rendered on the screen as the screen does not understang any other color model.

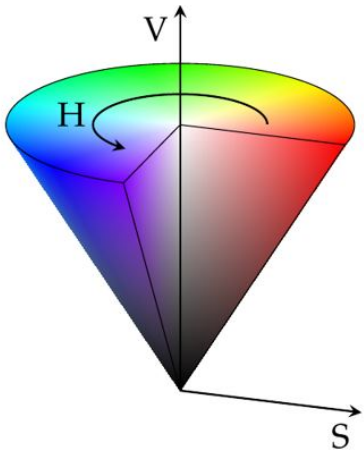
Color models are just the way in which image data is stored in memory. BGR and RGB are few of the most used color models. but not the most convenient ones. From a users point of view it is really hard to select and use a color in RGB color model as you would have noticed from the above color picker link. This lead to emergence of newer eaiser to use color models and one of the most used one is the **HSV color model.**

- **H - Hue**
- **S - Saturation**
- **V - Value**

Below image shows the RGB and the HSV color model



RGB Colour Model



HSV Colour Model

- **Biggest advantage of using the HSV color model us that we can select a color by chosing the Hue(H) and then get all the shades of the same color by manipulating the Saturation(S) and the Valeu(v).**
- **To acheive the same result with BGR model we will have to simultaneously manipulate all the R, G and the B values which is very difficult**

Image Formats

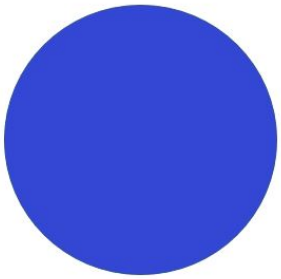
Images comes in all types of format two of the most famous ones are JPEG and PNG. Format decides a lot of properties of the image but one of the most important property decided by the format is the **number of channels**.

- JPEG images support only 3 channels (RGB)
- PNG images has an optional additional Alpha channel (RGBA)

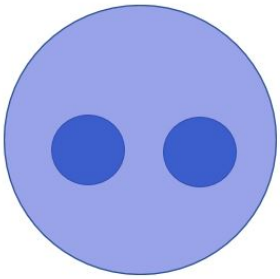
The Alpha channel decides the Trasperancy of a pixel. The Alpha channel has to do nothing with how images are actually represented on the instead they are something that are simulated by

software.

Below Image shows how power point uses alpha channel to show transperancy



Transparency – 0%



Transparency – 50%

NOTE : It is not mandatory for PNG images to have a fourth alpha channel

But if the images has an aplha channel the numpy array created as a result of reading the image will a 3D array with 4 channels.

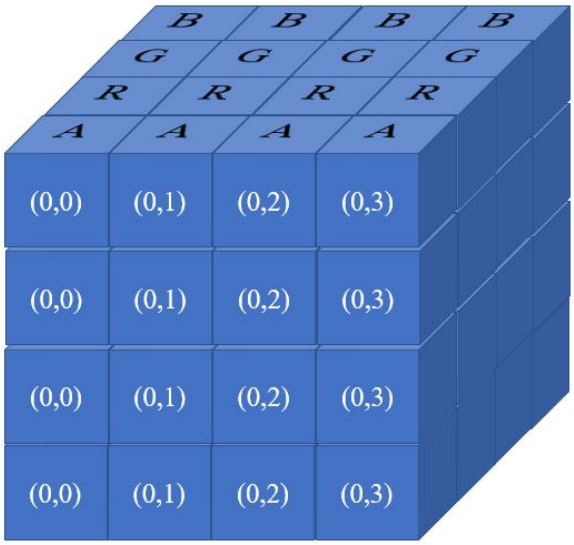


Image Reading

The image methods in cv2 starts with `im` most of the times
we use the `imread` method to read images. this takes in 2 parameters the first the file name the second parameter is the channles to read from the image

In [ ]:

```
from cv2 import cv2
import numpy

image = cv2.imread("images\\lena.jpg",1)
```

Second argument is a flag which specifies the way image should be read.

- cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.
- cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode
- cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel

The cv2 imread wont complain if the image file is not rpesent instead the image object which is infact a numpy array return NONE. else it would return the actual numpy array.

In [ ]:
```
from cv2 import cv2
import numpy

image = cv2.imread("images\\lenasa.jpg",1)
print(image)
```

We use the `imshow` a to show images

In [ ]:
```
from cv2 import cv2
import numpy as np

image = cv2.imread("images\\lena.jpg",1)
cv2.imshow('image',image)
```

The below code will only show the image for a split second. The reason being the program ends after showing the image. TO show the image longer we need to make the program execution to pasue. We can do this by adding a keyboard interrupt.

We add the `waitKey()` function to wait for a keyboard interrupt or a timeout event. The timeout has to be provided in milliseconds. If the timeout is `0` then the function waits forever unless a keyboard stroke is recorded

In [ ]:
```
from cv2 import cv2
import numpy as np

image = cv2.imread("images\\lena.jpg",1)
cv2.imshow('image',image)
cv2.waitKey(5000)
```

The reason why `waitkey()` works is because of the fact that it is a blocking function which means it stops the execution of the program until an event occurs.

It is always recomended to destroy a window object explicitly once its use is over.

In [ ]:
```
from cv2 import cv2
import numpy as np

image = cv2.imread("images\\lena.jpg",1)
cv2.imshow('image',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Writing Images

we can write/create images using the `imwrite()` method

In [ ]:
```
import numpy as np
import cv2

image = cv2.imread('images\\lena.jpg',0)
cv2.imshow('image',image)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('images\\lena_copy.png',image)
    cv2.destroyAllWindows()
```

The problem with the above code is that even if we press any other key (apart from escape) the program exits which is not desirable. so we can solve the above problem by using a while loop.

In [ ]:
```
import numpy as np
import cv2

image = cv2.imread('images\\lena.jpg',0)
cv2.imshow('image',image)

while True:
    k = cv2.waitKey(0)
    if k == 27:         # wait for ESC key to exit
        break
    elif k == ord('s'): # wait for 's' key to save and exit
        cv2.imwrite('images\\lena_copy.png',image)
        break
    else:
        pass
cv2.destroyAllWindows()
```

Getting the sequence correct

In [ ]:
```
%matplotlib qt
#just to make matplotlib to make a separate window for showing image
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('images\\lena.jpg',0)
plt.imshow(img)
plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
plt.show()
```

The problem is that the matplotlib uses RGB format while opencv uses GRB format to solve the above problem

Since `img` object created using cv2 is a numpy array we can solve this problem with numpy indexing and slicing.

In [ ]:
```
#just to make matplotlib to make a separate window for showing image
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('images\\lena.jpg',0)
# cv2.imshow('image',img)

img = img[:,:,::-1]

plt.imshow(img)
# plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
plt.show()
```

In [ ]:
```
#just to make matplotlib to make a separate window for showing image
#Images consist of X,Y and BGR values. By slicing the last index you only inverteded th
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('images\\lena.jpg',1)
# cv2.imshow('image',img)

img = img[:,::-1,::-1]

plt.imshow(img)
#plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
plt.show()
```

HOMEWORK

1. Read any image and show on the screen for unlimited time
2. Save image after changing color of image

HOMEWORK SOLUTION

In [ ]:
```
# TASK 1

from cv2 import cv2
import numpy as np

image = cv2.imread("images\\lena.jpg",1)
cv2.imshow('image',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [ ]:
```
#TASK 2

import numpy as np
import cv2
```

```
img = cv2.imread('images\\lena.jpg',1)
cv2.imshow('image',img)
while True:
    k = cv2.waitKey(0)
    if k == 27:
        break
    elif k == ord('s'):
        img = img[:,:,::-1]

        cv2.imwrite('images\\lena_copy.png',img)
        break
    else:
        pass
cv2.destroyAllWindows()
```

In [ ]: