# Extracting Bit planes

**Rajapandi Nadar**
UID: 192585
Roll : 360

**Derron colaco**
UID: 192408
Roll :358

**Department of Physics, St. Xavier's College(Autonomous), Mumbai-400001**

Course code: SPHY05AC

Digital Image processing

Group 21

September 2021

**Abstract:**

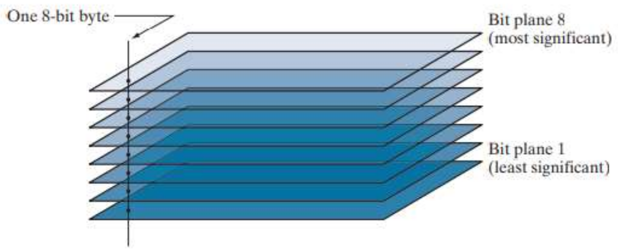
In this project we will explore the concept of bit planes of a 8-bit image with the help of python programming language we will write code for extracting different bit plane of a 8 bit image and by plotting all the bit planes simultaneously we will compare that with the original one and by comparing we will conclude about which bit planes carries how much information regarding the image

**Index**

**<u>Introduction</u>:**

As we all know that each pixel value of the images are integers which is composed of bits the integers are converted into binary format and each bits of the binary is assigned to different bit planes for example values in a 256- level gray-scale are composed of 8-bits, here instead of highlighting the intensity-level ranges as intensity-level slicing we would highlight the contribution made to total image appearance by specific bits

This figure illustrated an 8-bit image may be considered as being composed of eight one-bit planes, with plane 1 containing the lowest-order bit of all pixels in the image and plane 8 all the highest-order bits



**<u>Background</u>:**

**<u>Bit-plane slicing</u>**

The gray level of each pixel in a digital image is stored as one or more bytes in a computer. For an 8-bit image,0 is encoded as 00000000 and 255 is encoded as 11111111. Any number between 0 to 255 is encoded as one byte. The bit on the far left side is referred to as the most significant bit because a change in that bit would significantly change the value encoded by the byte. The bit in the far right is referred to as the least significant bt, because a change in this bit does not change the encoded gray value much. The bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel. One cal use only most significant bit to represent teh pixel, which reduces the original gray level to a binary image the three main goals of bit plane slicing is

- Converting a gray level image to a binary image
- Representing an image with fewer bits and corresponding the image to a smaller size
- Enhancing the image by focusing

**Theory :**

For an image A having dimension i*j which consists of i*j terms each and every entry in this matrix will be an integer for every $A_{i,j}$ there will be an corresponding binary conversion, if we consider the maximum gray scale value in the image matrix is ‘7’ then the image will be an 3-bit image similarly if the maximum gray scale value is ‘63’ then the image will be a 6-bit image

Consider a matrix

| | | | | |
|---|---|---|---|---|
| 6 | 7 | 6 | 6 | 7 |
| 0 | 0 | 0 | 1 | 2 |
| 1 | 1 | 1 | 2 | 3 |
| 4 | 5 | 5 | 4 | 2 |
| 6 | 6 | 6 | 7 | 7 |

Since in the given image the maximum gray scale is 7, it is a 3-bit image, we will convert the image into binary and separate the bit planes

| | | | | |
|---|---|---|---|---|
| 110 | 111 | 110 | 110 | 111 |
| 000 | 000 | 000 | 001 | 010 |
| 001 | 001 | 001 | 010 | 011 |
| 100 | 101 | 101 | 100 | 010 |
| 110 | 110 | 110 | 111 | 111 |

Separating the bit planes, we obtain

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

MSB plane

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Centre bit plane

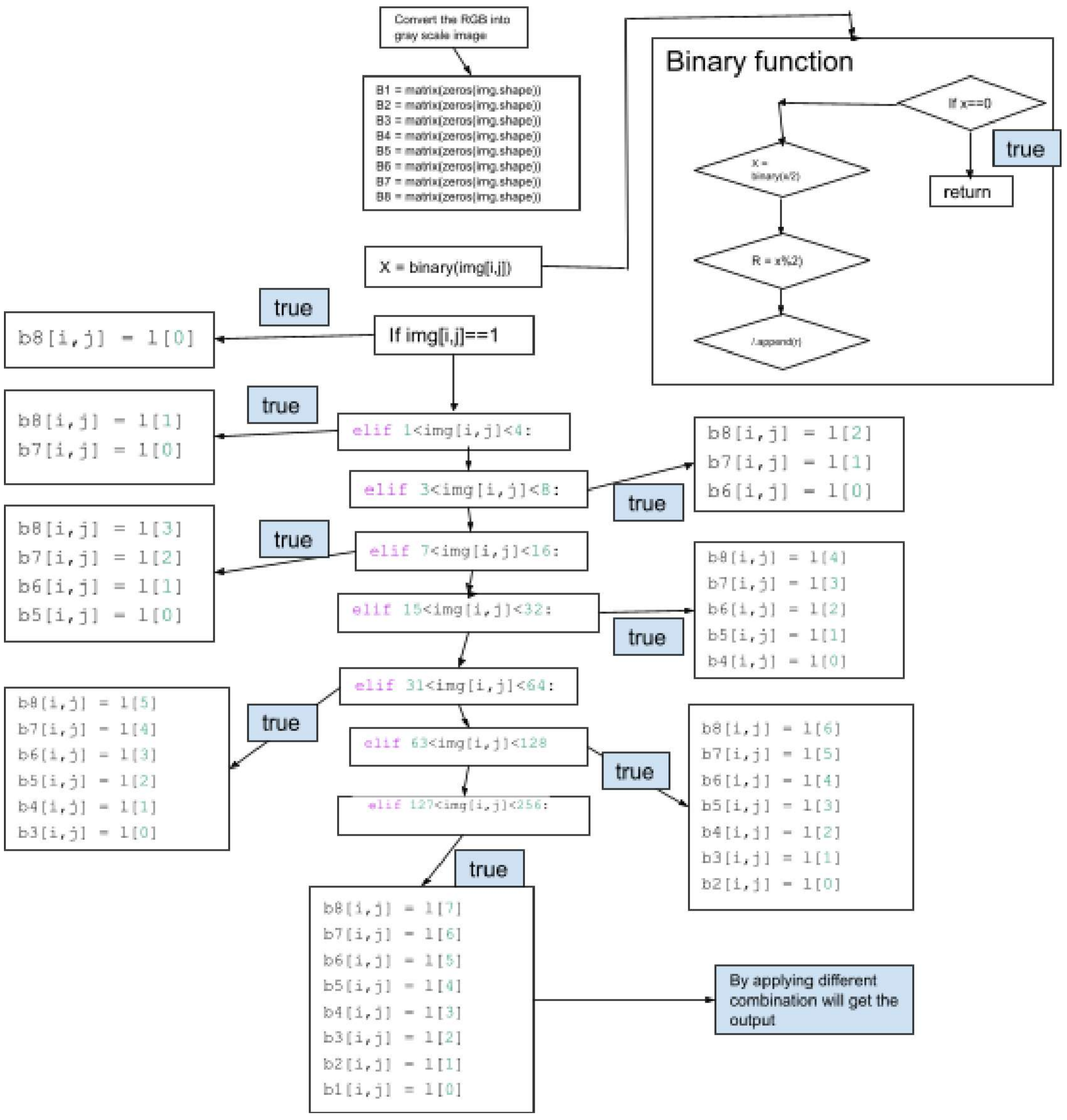
| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |

LSB plane

**<u>Computational Work</u>:**

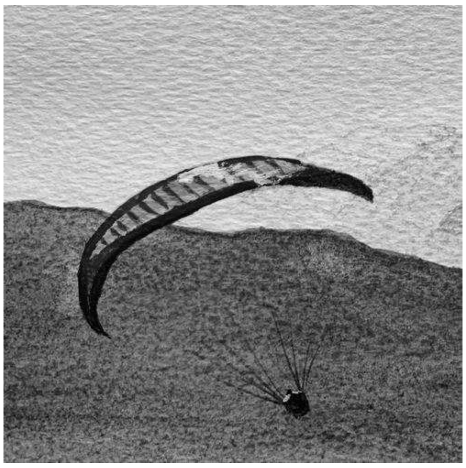
**<u>Flowchart of the program</u> :**

- Read the image provided and convert it into a 8-bit gray scale image
- We will create 8 zero matrix of the same dimension same as that of the original 8-bit image
- Will define a function for converting the integer into the binary format will append each and every bit in a empty list created
- Will start a for loop and a nested for loop
- Will compare the entry in the image matrix with the values and each condition will have a specific way of filling the bit planes [the zero matrix that was created earlier]



**Results** :

After obtaining 8 different bit planes using them we have obtained different images firstly we have contained image of each and every bitplanes separately which are as follows



Original 8-bit gray scale image

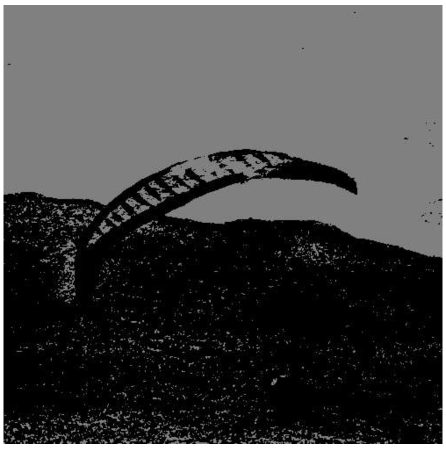


Image of eighth bit plane

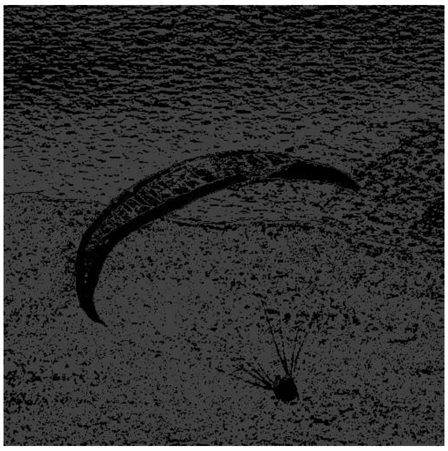


Image of seventh bit plane

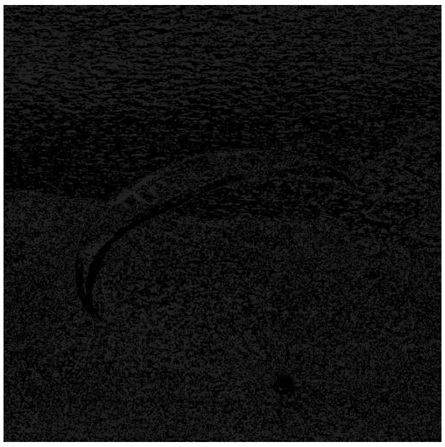


Image of sixth bit plane

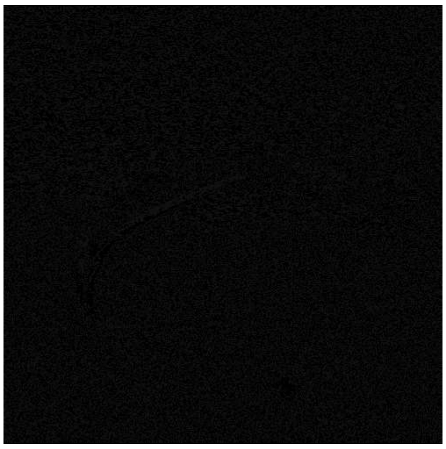


Image of fifth bit plane

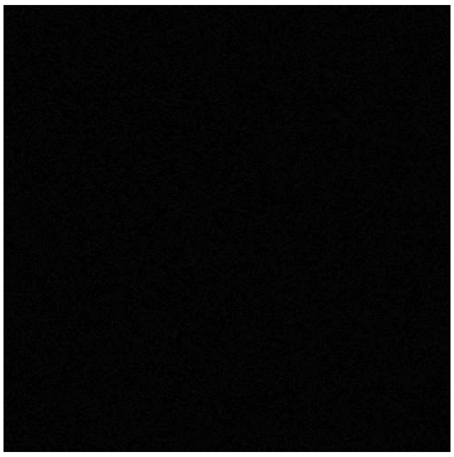


Image of fourth bit plane

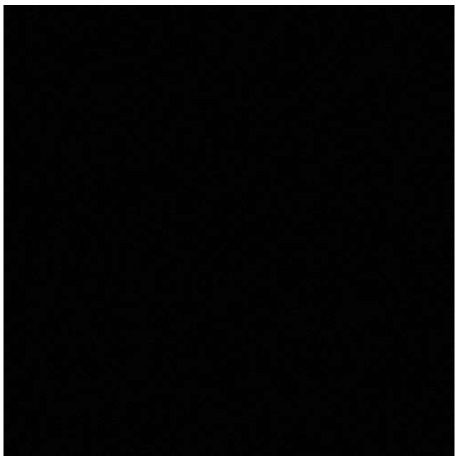


Image of third bit plane

Image of second bit plane

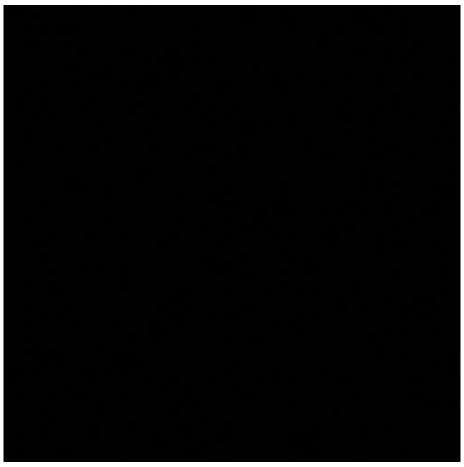


Image of first bit plane

From all the above image it is clear that the most significant bit planes consist the majority of the information of the image

We have obtained image by eliminating the most significant bit plane of the image and compared it with the original image

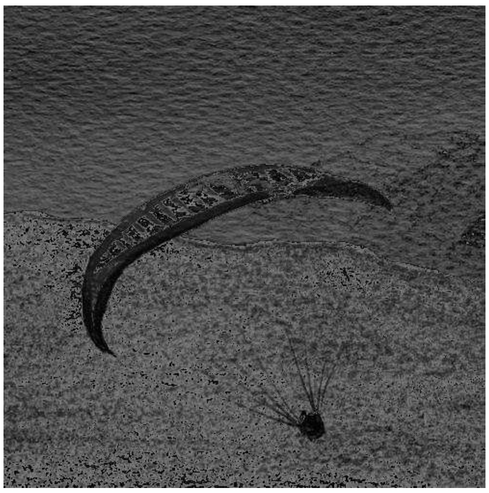
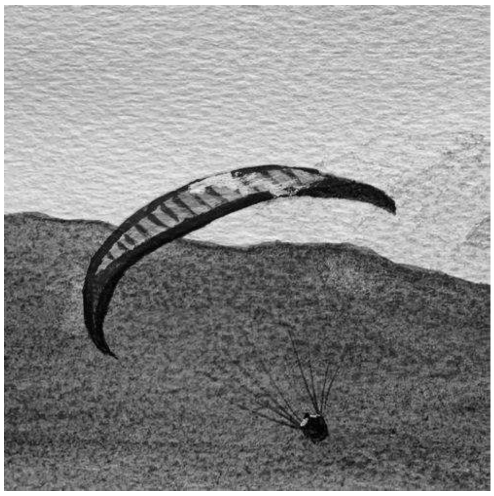


Image after setting the
most significant bit plane to zero



Original image

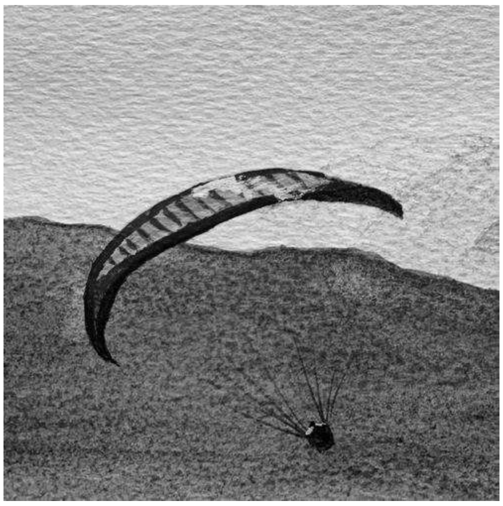
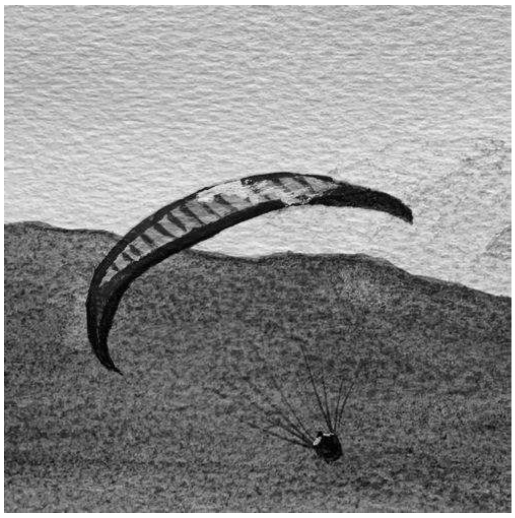


Image after setting the
least significant bit plane to zero



Original image

Using the above images we can conclude that the most significant bit plane has the most effect on the image as we can see the image after setting the least significant bit plane to zero is very much similar to the original image while the image that is created after setting the most significant bit plane to zero some informations (detailing are lost) when the most significant bit plane was set to zero

## <u>**Results and conclusion**</u> :

We successfully extracted the bit planes from the 8 bit gray scale and with the help of the bit planes obtained we have created images of individual bit planes and compared it with the original image and concluded that the highest bit planes has the maximum information about the image we have also tried to create image by making the most and the least significant and concluded again the same that the most significant bit planes has majority if the information

**Acknowledgement:**
We would like to thank the Physics Department of St. Xavier's College, Mumbai, who remained determined enough to impart us practical "hands-on" knowledge even during this pandemic. We would also like to thank our families for cooperating and helping throughout this project which has been a journey during COVID-19.

**Reference :**

- **Digital image processing by Rafael C. Gonzalez & Richard E. Woods**

**Source Code:**

```
from numpy import *
from matplotlib.pyplot import *
from PIL import Image
import cv2
from google.colab.patches import cv2_imshow
from google.colab import drive
drive.mount('/content/drive')
        #Step 1 : Extracting the image and converting the image into grayscale
img1 = imread('/content/drive/MyDrive/DIP/pic.jpg')
img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
print('Original Image')
cv2_imshow(img)
print('\n')

        #Step 2 : Formaing empty matrix which will be furter converted to bit
planes

b1 = matrix(zeros(img.shape))
b2 = matrix(zeros(img.shape))
b3 = matrix(zeros(img.shape))
b4 = matrix(zeros(img.shape))
b5 = matrix(zeros(img.shape))
b6 = matrix(zeros(img.shape))
b7 = matrix(zeros(img.shape))
b8 = matrix(zeros(img.shape))
```

```
    #Step 3 : Defining a function for conveting the numerics to binary

def binary(x):
  if(x==0):
    return
  else:
    binary(int(x/2))
    r=x%2
    l.append(r)



for i in range(500):
  for j in range(500):
    l=[]

    binary(img[i,j])
    if img[i,j] == 1:
      b8[i,j] = l[0]
    elif 1<img[i,j]<4:
      b8[i,j] = l[1]
      b7[i,j] = l[0]
    elif 3<img[i,j]<8:
      b8[i,j] = l[2]
      b7[i,j] = l[1]
      b6[i,j] = l[0]
```

```
elif 7<img[i,j]<16:
      b8[i,j] = l[3]
      b7[i,j] = l[2]
      b6[i,j] = l[1]
      b5[i,j] = l[0]
    elif 15<img[i,j]<32:
      b8[i,j] = l[4]
      b7[i,j] = l[3]
      b6[i,j] = l[2]
      b5[i,j] = l[1]
      b4[i,j] = l[0]
    elif 31<img[i,j]<64:
      b8[i,j] = l[5]
      b7[i,j] = l[4]
      b6[i,j] = l[3]
      b5[i,j] = l[2]
      b4[i,j] = l[1]
      b3[i,j] = l[0]
    elif 63<img[i,j]<128:
      b8[i,j] = l[6]
      b7[i,j] = l[5]
      b6[i,j] = l[4]
      b5[i,j] = l[3]
      b4[i,j] = l[2]
      b3[i,j] = l[1]
      b2[i,j] = l[0]
```

```
elif 127<img[i,j]<256:
    b8[i,j] = l[7]
    b7[i,j] = l[6]
    b6[i,j] = l[5]
    b5[i,j] = l[4]
    b4[i,j] = l[3]
    b3[i,j] = l[2]
    b2[i,j] = l[1]
    b1[i,j] = l[0]

      #Getting different types of images with the help of the bit planes obtained

print('image having most significant bit plane to zero')
h = pow(2,6)*b2 + pow(2,5)*b3 + pow(2,4)*b4 + pow(2,3)*b5 + pow(2,2)*b6 +
pow(2,1)*b7 + pow(2,0)*b8
cv2_imshow(h)
print('\n')
print('image having least significant bit palne to zero')
h1 = pow(2,7)*b1 + pow(2,6)*b2 + pow(2,5)*b3 + pow(2,4)*b4 + pow(2,3)*b5 +
pow(2,2)*b6 + pow(2,1)*b7
cv2_imshow(h1)
print('\n')
print('image of eighth bit plane')
h2 = pow(2,7)*b1
cv2_imshow(h2)
print('\n')
print('image of seventh bit plane')

h3 = pow(2,6)*b2

cv2_imshow(h3)

print('\n')

print('image of sixth bit plane')

h4 = pow(2,5)*b3

cv2_imshow(h4)

print('\n')

print('image of fifth bit plane')

h5 = pow(2,4)*b4

cv2_imshow(h5)

print('\n')

print('image of fourth bit plane')

h6 = pow(2,3)*b5

cv2_imshow(h6)

print('\n')

print('image of third bit plane')

h7 = pow(2,2)*b6

cv2_imshow(h7)

print('\n')

print('image of second bit plane')

h8 = pow(2,1)*b7

cv2_imshow(h8)

print('\n')

print('image of first bit plane')

h9 = pow(2,0)*b8

cv2_imshow(h9)
```