Tarefa 1: Análise e processamento de imagens

Antonio Pilan. NUSP: 10562611

Ferramenta: OpenCV

Nesse trabalho, vou usar a ferramenta OpenCV.

A biblioteca OpenCV é programada em C/C++ e encapsuladas em Python o que gera maior performance. Isso porque ela tem como base a biblioteca Numpy, que trabalha com o processamento em C encapsulado em Python.

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Abrindo a imagem

Na biblioteca OpenCV a imagem é salva como numpy array.

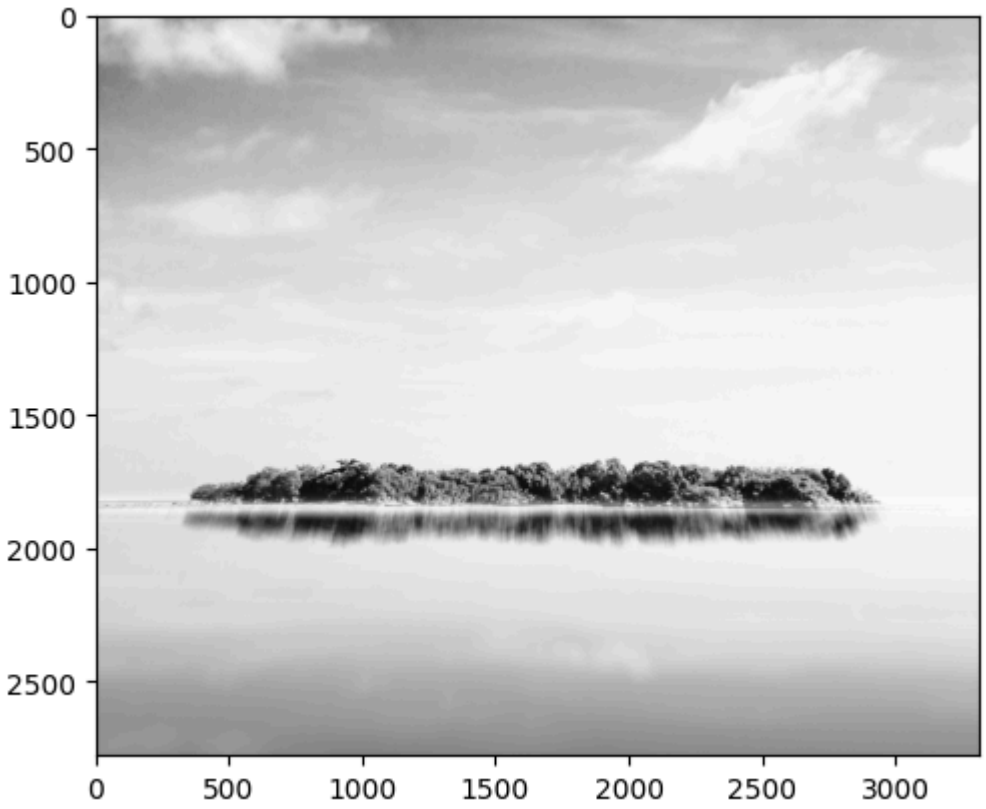
In [ ]:
```
img = cv2.imread('img_tarefa1.jpg')

print("Primeira linha: ")
display(img[0]) #printando a primeira linha x=0

print("Primeiro pixel: ")
display(img[0][0]) #printando a primeira linha x=0
```

```
Primeira linha:
array([[177, 177, 177],
       [176, 176, 176],
       [174, 174, 174],
       ...,
       [193, 193, 193],
       [195, 195, 195],
       [196, 196, 196]], dtype=uint8)
Primeiro pixel:
array([177, 177, 177], dtype=uint8)
```

Então, usamos o módulo para ler a imagem e usamos o Matplotlib pra plotar a figura

In [ ]:
```
plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1947f7b2840>
```



Analisando dimensões da imagem

In [ ]:
```
height, width, channels = img.shape

print(f"altura: {height}, largura: {width}, numero de canais: {channels}")
```

```
altura: 2778, largura: 3316, numero de canais: 3
```

Podemos ver que temos uma imagem 3316x2778. Porém, existem 3 canais na imagem (RGB). O que podemos verificar novamente visualizando a matriz da imagem:

In [ ]:
```
print("Primeira linha: ")
display(img[0]) #printando a primeira linha x=0

print("Primeiro pixel: ")
display(img[0][0]) #printando a primeira linha x=0
```

```
Primeira linha:
array([[177, 177, 177],
       [176, 176, 176],
       [174, 174, 174],
       ...,
       [193, 193, 193],
       [195, 195, 195],
       [196, 196, 196]], dtype=uint8)
Primeiro pixel:
array([177, 177, 177], dtype=uint8)
```

Os três elementos do pixel (RGB) armazenam o mesmo valor, isso acontece porque a imagem está em escala de cinza. Tendo isso em mente, não é necessário ter a matriz nesse formato, basta reduzir os valores de RGB para um único valor, o de escala de cinza.

Portanto, poderíamos usar apenas o valor da primeira posição de cada pixel para nossas análises.

Mas o pacote Numpy salva nossa vida novamente, verificando o valor de mínimo e máximo em toda a array:

Cálculos

In [ ]:
```
max_value = img.max()
min_value = img.min()

print(f"Max: {max_value}\nMin: {min_value}")
```

```
Max: 255
Min: 8
```

In [ ]:
```
bit_depth = np.log2(max_value - min_value)
print(bit_depth)
```

```
7.95
```

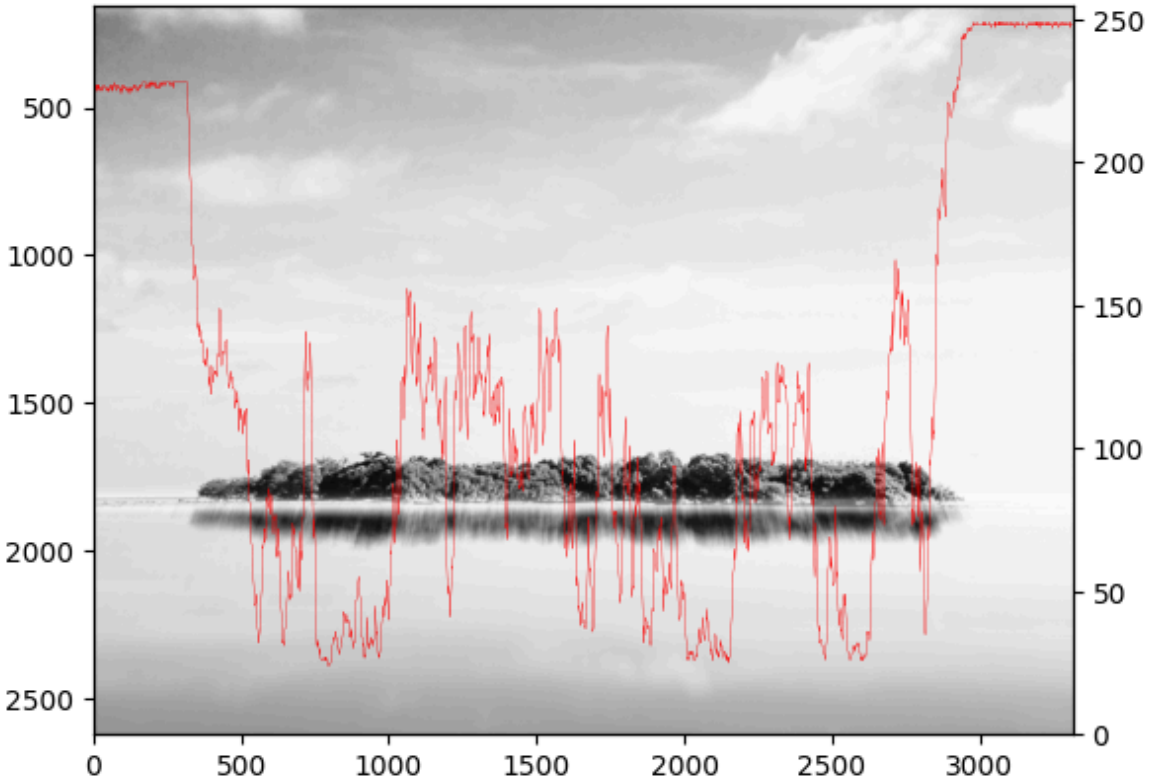
Portanto, o pixel de maior valor é 255 e o de menor valor é 8.

A partir disso, sabemos que temos variações de cinza entre 8 e 255... 247 tons de cinza pra ser mais específico. Ao calcular a profundidade, tempos 7.95 bits de profundidade na foto, então sabemos que pra convenções computacionais, essa é uma imagem 8 bits.

Visualizando intensidade de cinza

In [ ]:
```
x = 1900
line = img[x]

fig, ax1 = plt.subplots()
ax1.imshow(img)

ax2 = ax1.twinx()
ax2.set_ylim(0, 255)
ax2.plot(line,c='r',linewidth=0.09)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x1940407cb00>,
 <matplotlib.lines.Line2D at 0x194040a00b0>,
 <matplotlib.lines.Line2D at 0x1940407ce00>]
```



Procurei lugares perto do meio da ilha. No plot, é possível identificar a ilha no centro da figura, aonde os tons de cinza são menos intensos.

perto das extremidades temos o céu (ou reflexão do mar dependendo do valor de x), aonde o tom de cinza se aproxima do branco.