Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

Ciência da Computação

Processamento Digital de Imagens (9789)

Cubo RGB

Discente

Nome	**RA**
Eduardo Andrade Manfre	116409

Maringá - PR - 2024

1. Introdução

O cubo RGB de 24 bits é uma representação tridimensional das cores que podem ser formadas pela combinação dos três canais de cor primários: vermelho (Red), verde (Green) e azul (Blue). Cada canal de cor é representado por 8 bits, permitindo 256 níveis diferentes de intensidade (de 0 a 255). Combinando esses três canais, é possível representar aproximadamente 16,7 milhões de cores (256^3), o que constitui uma vasta gama de cores perceptíveis pelo olho humano.

2. Cores Primárias e Secundárias

No modelo RGB, as cores primárias são o vermelho, o verde e o azul. Essas cores são chamadas de primárias porque não podem ser criadas pela mistura de outras cores. Em um sistema aditivo de cores, como o RGB, essas cores são combinadas em diferentes intensidades para criar outras cores.

Quando duas cores primárias são combinadas em intensidades iguais, formam-se as cores secundárias:

- Vermelho + Verde = Amarelo
- Vermelho + Azul = Magenta
- Verde + Azul = Ciano

3. Estrutura do Cubo RGB

O cubo RGB é uma forma visual de representar como as cores primárias se misturam para criar todas as outras cores. Cada eixo do cubo representa um dos três canais de cor (R, G, B):

- O eixo X representa o canal vermelho (R).
- O eixo Y representa o canal verde (G).
- O eixo Z representa o canal azul (B).

No cubo RGB, o ponto (0, 0, 0) representa a cor preta, onde todas as intensidades de cor são zero. No canto oposto, o ponto (255, 255, 255) representa a cor branca, onde todas as intensidades de cor são máximas.

4. Formando o Cubo RGB

Para visualizar o cubo RGB, imagine um espaço tridimensional onde cada ponto é definido por uma tripla de valores (R, G, B). Movendo-se ao longo dos eixos,

as combinações desses valores criam diferentes cores. Quando combinamos essas intensidades, obtemos uma matriz tridimensional de cores que formam o cubo RGB.

Em suma, o cubo RGB de 24 bits oferece uma representação completa e precisa das cores possíveis em dispositivos digitais, proporcionando uma base fundamental para a manipulação e a compreensão das cores em um mundo digital.

5. Implementação

Python

```python
import numpy as np
import cv2

def cubo_RGB(face, i):
    if face < 1 or face > 6:
        raise ValueError("Face deve ser um valor entre 1 e 6")
    if i < 0 or i > 255:
        raise ValueError("O valor de i deve estar entre 0 e 255")

    # Criar uma matriz de zeros com forma 256x256x3 para representar a
imagem
    image = np.zeros((256, 256, 3), dtype=np.uint8)

    if face == 1:
        image[:, :, 1] = np.arange(256).reshape(1, 256)
        image[:, :, 2] = np.arange(256).reshape(256, 1)
        image[:, :, 0] = i

    elif face == 2:
        image[:, :, 0] = np.arange(256).reshape(256, 1)
        image[:, :, 2] = np.arange(255,-1,-1).reshape(1, 256)
        image[:, :, 1] = i
        image = np.flipud(image)

    elif face == 3:
        image[:, :, 1] = np.arange(256).reshape(1, 256)
        image[:, :, 2] = np.arange(256).reshape(256, 1)
        image[:, :, 0] = 255-i

    elif face == 4:
        image[:, :, 0] = np.arange(256).reshape(256, 1)
        image[:, :, 2] = np.arange(255,-1,-1).reshape(1, 256)
        image[:, :, 1] = 255-i
        image = np.flipud(image)
```

```
    elif face == 5:
        image[:, :, 0] = np.arange(256).reshape(256, 1)
        image[:, :, 1] = np.arange(256).reshape(1, 256)
        image[:, :, 2] = 255-i
        image = np.flipud(image)

    elif face == 6:
        image[:, :, 0] = np.arange(256).reshape(256, 1)
        image[:, :, 1] = np.arange(256).reshape(1, 256)
        image[:, :, 2] = i
        image = np.flipud(image)

    return image

# Parâmetros de entrada
face = 6  # Número da face (1 a 6)
i = 0     # Valor da i-ésima fatia (0 a 255)

# Gerar a fatia do cubo RGB com a face escolhida
image = cubo_RGB(face, i)
image1 = cubo_RGB(face, i+85)
image2 = cubo_RGB(face, i+170)
image3 = cubo_RGB(face, i+255)

cv2.imshow(f'face {face}, fatia 0', image)
cv2.imshow(f'face {face}, fatia 85', image1)
cv2.imshow(f'face {face}, fatia 170', image2)
cv2.imshow(f'face {face}, fatia 255)', image3)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

explicação do código

Foi utilizado a biblioteca opencv.

A princípio verifica-se os parâmetros (**‘i’** e **‘face’**) passados estão dentro do esperado, caso contrário é apresentado um erro.

Caso os dados de entrada estiverem dentro do esperado é inicializado uma matriz de zeros da forma (256, 256, 3), onde 256 corresponde a altura e largura da imagem, e 3 corresponde aos três canais de cores RGB. Após a inicialização da matriz e de acordo com a face de entrada, o programa entra em um dos if(s) executando as seguintes operações:

‘np.arange(256).reshape(1, 256)’;

A qual cria um array com valores de 0 a 255, o remodela para uma linha (1x256) e o salva em um canal RGB específico de acordo com cada if.

E ‘**np.arange(256).reshape(256, 1)’**.
Que também cria um array com valores de 0 a 255 mas o remodela para uma coluna (256x1), e também o salva em um canal RGB específico de acordo com o if em questão.

Em dois casos foi utilizado valores diferentes dos já mencionados sendo eles:

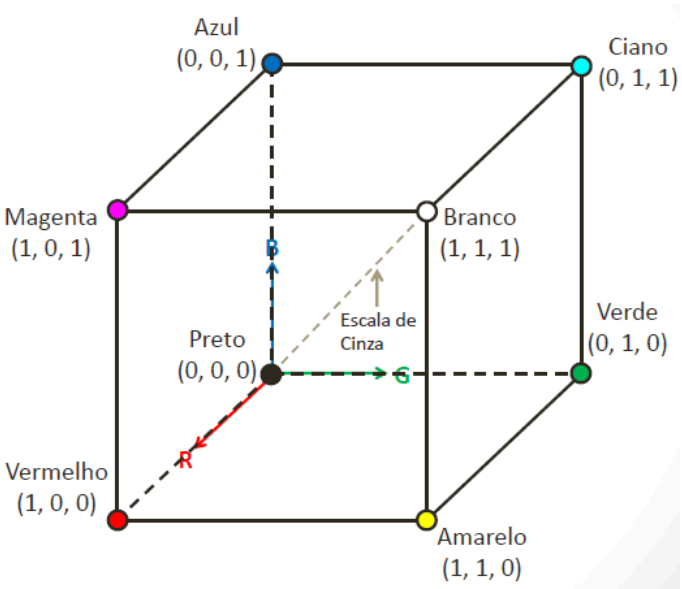
‘np.arange(255, -1, -1).reshape(1, 256)’.
Com esses valores a matriz será preenchida de forma decrescente, foi utilizado para corrigir a orientação das imagens de saída junto com **‘np.flipud’** que é utilizado para flipar a imagem e deixá-la na orientação correta.

Por fim o **‘i’** é utilizado para definir o valor que será fixo, o qual define a fatia do cubo, após executar essas operações é retornada a matriz **‘image’** com a imagem resultante.

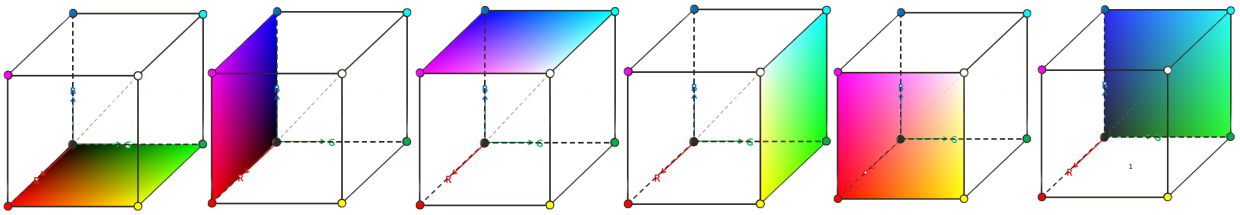
No final do programa é gerado 4 fatias a partir da face escolhida.

6. Resultados

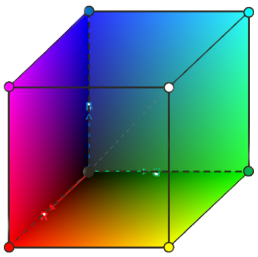
Foi utilizado como orientação de qual vértice fixar cada cor o cubo rgb apresentado em aula, sendo ele:



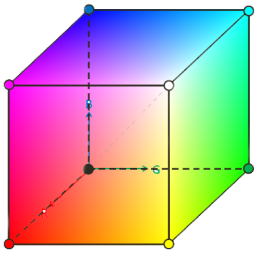
Faces obtidas:



face 1 face 2 face 3 face 4 face 5 face 6



face 1, face 2 e face 6

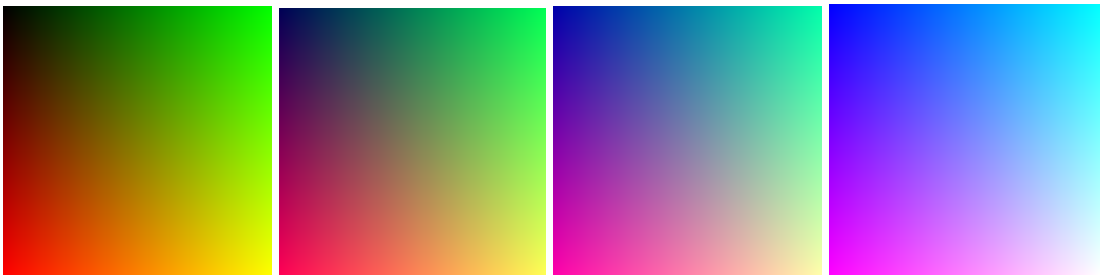


face 3, face 4 e face 5.

Face e fatiamentos:

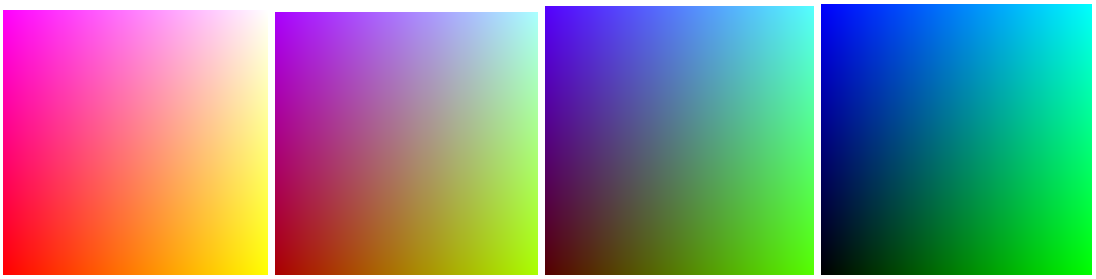
Face 1, fatiamento:

0, **85,** **170,** **255.**



Face 5, fatiamento:

0, **85,** **170,** **255.**



Observa-se que o último fatiamento do cubo com valor 255 é a face oposta à face de entrada.

7. Bibliografia

PROCESSAMENTO DIGITAL DE IMAGENS - PROF. DR. FRANKLIN CÉSAR FLORES, SLIDE - FUNDAMENTOS DE COR.