Universidade Estadual do Norte do Paraná

Centro de Ciências Tecnológicas

Curso: Ciência da Computação
Disciplina: Computação Gráfica
Prof. Bruno Miguel Nogueira de Souza

Notas de Aula – Computação Gráfica

Espaços de Cor

RGB, YCbCr e YUV

Exercício – Construa um algoritmo que converta os valores RGB de um pixel para o espaço de cor YCbCr de acordo com a matriz a seguir:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0{,}2568 & 0{,}5041 & 0{,}0979 \\ -0{,}1482 & -0{,}2910 & 0{,}4392 \\ 0{,}4392 & -0{,}3678 & -0{,}0714 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Conversão RGB para CMYK, HSI, HSL e HSV.

HSI:

$$\theta = cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\right\}$$

$$H = \begin{cases} \theta \; se \; B \leq G \\ 360º - \; \theta \; se \; B > G \end{cases}$$

$$I = \frac{R+G+B}{3}$$

$$S = 1 - \frac{3}{R+G+B} * [\min(R,G,B)]$$

HSV:

$$Dado \; que \; MAX = \max(R,G,B) \; e \; MIN = \min(R,G,B), então:$$

$$H = \begin{cases} 60 * \left(\frac{G-B}{MAX-MIN}\right) + 0, se\ MAX = R\ e\ G \geq B \\ 60 * \left(\frac{G-B}{MAX-MIN}\right) + 360, se\ MAX = R\ e\ G < B \\ 60 * \left(\frac{B-R}{MAX-MIN}\right) + 120, se\ MAX = G \\ 60 * \left(\frac{R-G}{MAX-MIN}\right) + 240, se\ MAX = B \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

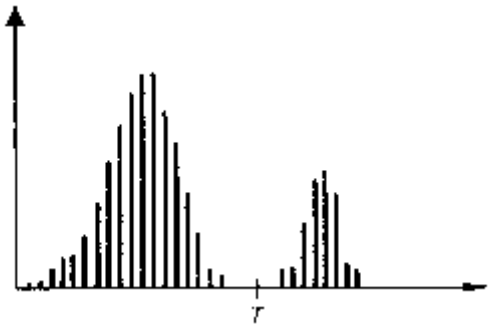
HSL (H e S do HSV)

$$L = V - \frac{S}{2}$$

Binarização ou Limiarização

A conversão de uma imagem com níveis de cinza para uma imagem com representação binária (dois tons) é importante para uma série de objetivos, tais como:

- identificar objetos e separá-los do fundo da imagem;
- quando analisar a forma da imagem é mais importante que a intensidade dos píxeis; e
- apresentar a imagem em um dispositivo de saída que tem somente um bit de resolução de intensidade, ou seja, um dispositivo de dois níveis, como uma impressora.



O diagrama da figura 1 representa a um histograma típico de uma imagem de cor mais clara sob uma funda mais escuro. Ele é bi-modal, uma transformação da intensidade da imagem para que ela passe a ter só dois níveis distintos,neste caso pode separar o objeto do fundo.

Esta transformação é chamada binarização, e pode ser descrita através da aplicação da função: s = T(r).

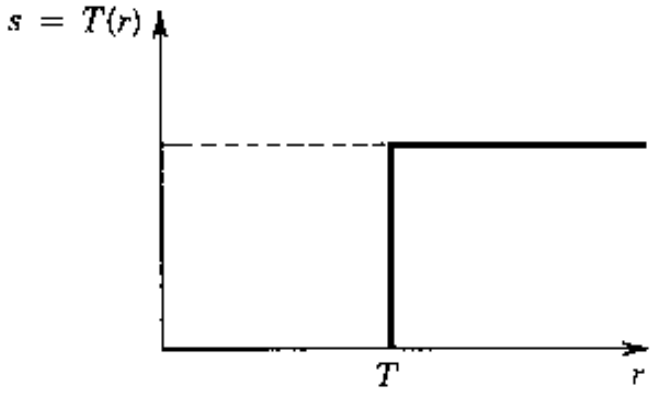


Figura 1

A função T(r) compara o sinal de entrada com um valor de *threshold* (T), escolhido como referência para a separação dos níveis de cinza. O sinal de saída, apresentado é obtido pela relação:

$$s = \begin{cases} 1\ para\ r \geq T \\ 0\ para\ r < T \end{cases}$$

O histograma da imagem, após sua binarização, terá apenas dois tons com número de pixels diferentes de zero.

Algoritmos clássicos de Limiarização:

- Otsu
- Johansen

Limiarização de Otsu

$$P_i = \frac{n_i}{n} \qquad \varpi_1 = 1 - \varpi_0$$

$$\mu_t = \sum_{i=0}^{t} iP_i \qquad \varpi_0 = \sum_{i=0}^{t} P_i$$

$$\sigma_B^2 = \omega_0 \omega_1 (\mu_1 \mu_0)^2$$

$$\mu_0 = \frac{\mu_t}{\varpi_0} \qquad \mu T = \sum_{i=0}^{l-1} iP_i$$

$$\mu_1 = \frac{\mu T - \mu_t}{1 - \mu_0} \qquad \sigma_T^2 = \sum_{i=0}^{l-1} (i - \mu T)^2 P_i$$

$$\eta = \frac{\sigma_B^2}{\sigma_T^2}$$

Onde n é o número total de pixels na imagem;

Pi = Probabilidade de um pixel da imagem ser de uma cor i

T = max(η)

Implementação de limiarização de OTSU

```
Algoritmo de OTSU

Dado Pi o histograma da cor i
max = -∞
Para t=0 ate 255
      Para i = 0 até t
            w0 = w0+Pi
            μt = μt + i*Pi
            Se (i!=t)
                  μT = μT + i*Pi
      fimPara
      w1 = 1 – w0
      μ0 = μ1/w0
      μ1 = (μT – μt)/(1- μ0)
      σb2 = w0*w1*(μ1*μ0)2
      Para i=0 até (t-1)
            σT2 = σT2 + (i-μT)2*Pi
      fimPara
      η = σb2/ σT2
      if (η >max)
            max = η
            maxi = t
      fimIf
fimPara
return maxi
```

Aula 04/04/2019

4-Vizinhança e 8-Vizinhança de um pixel

Filtro de Suavização – Média

Implementação de média em 4-Vizinhança e 8-Vizinhança

Aula 09/04/2019

Operação de Convolução em Imagens

Utilizando Máscara como operador;

Processamento de Imagens – Filtros de Realce e de Suavização

Filtros de Suavização

- Filtro da média

- Filtro da Mediana

Filtros de Realce

- Detecção de Bordas (Máscara de Detecção de laplace)

Aula 11/04/2019

- Instalação OpenCV e Scikit-Image no Python

Instalar Anaconda -> CONDA PROMPT

Criar ambiente para instalação do OPENCV

>> conda create env -n opencv-env python=3.5

>>activate opencv-env

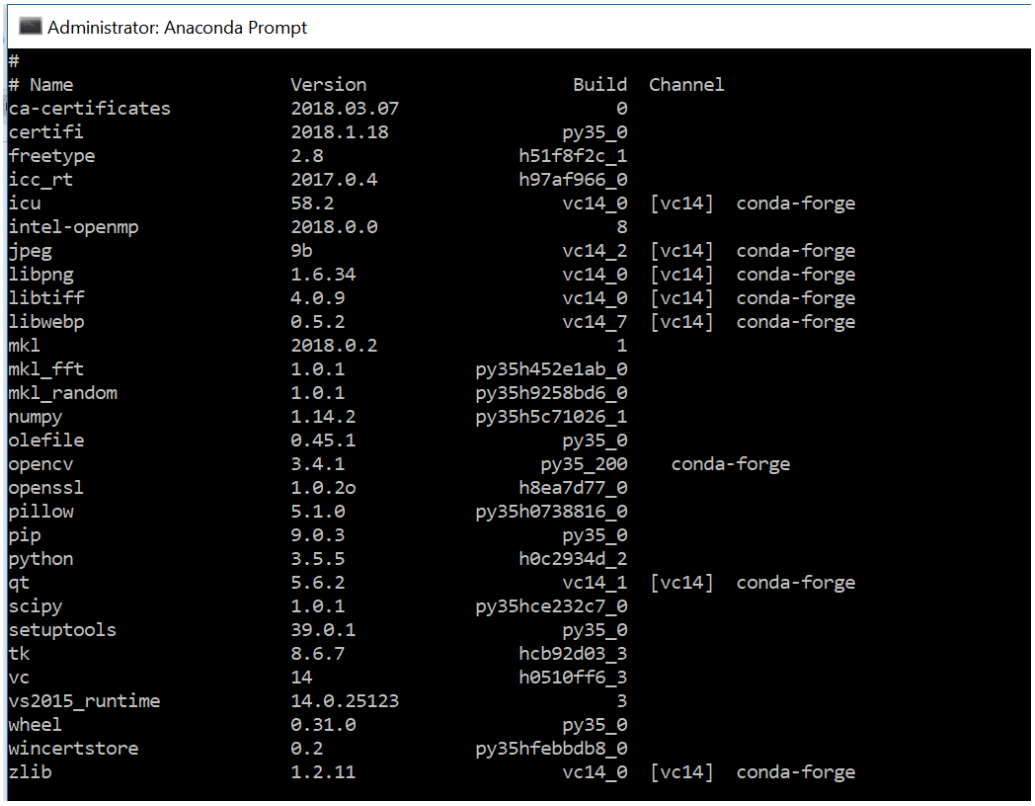
>>`conda install -c conda-forge opencv`

>>conda install scipy

>> conda install pillow

>>conda install scikit-image

Dando o comando conda-list, o ambiente deve estar com estas bibliotecas



Códigos de teste:

```
import os.path as osp
import cv2

path = "C:\\imagens\\imagem.jpg"
pastaResult = "C:\\imagens\\resultados"
imgMatrix = cv2.imread(path)

width = len(imgMatrix)
height = len(imgMatrix[0])
#imprime tamanho da imagem
print("w:" + str(width) + " h:" + str(height))
#Pega o nome do arquivo da imagem
imgname = osp.basename(img)
#Redimensiona Imagem para 200x200
newpatch = cv2.resize(imgMatrix, (200, 200))
cv2.imwrite(pastaResult + '\\rs' + imgname, newpatch)
#aplica filtro de deteccao de bordas de Canny, usando limiares 200 e
70
edges = cv2.Canny(newpatch,200,70)
cv2.imwrite(pastaResult + '\\borda_' + imgname, edges)
```

Lendo Pixel a pixel em Python

```
def lerPixels(matrixImg):
    cont = 0
    #Cria uma nova matriz igual a matriz de imagem
    newimg = matrixImg
    for row in (range(len(matrixImg))):
        for col in (range(len(matrixImg[row]))):
            red = matrixImg[row, col, 0]
            green = matrixImg[row, col, 1]
            blue = matrixImg[row, col, 2]
            #Fazer alteracoes na imagem... alterando
            # os valores de cada pixel na nova imagem
            newimg[row, col, 0] = red
            newimg[row, col, 1] = blue
            newimg[row, col, 2] = green

    return newimg
```