UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GUSTAVO GINO SCOTTON

PROCESSAMENTO DE IMAGEM – MODOS DE SELEÇÃO

Araranguá
2018

GUSTAVO GINO SCOTTON

PROCESSAMENTO DE IMAGEM – MODOS DE SELEÇÃO

Relatório realizado para a disciplina de Tópicos Especiais III da Universidade Federal de Santa Catarina, com o conteúdo de processamento de imagem, para obtenção de uma das notas referentes aos trabalhos.

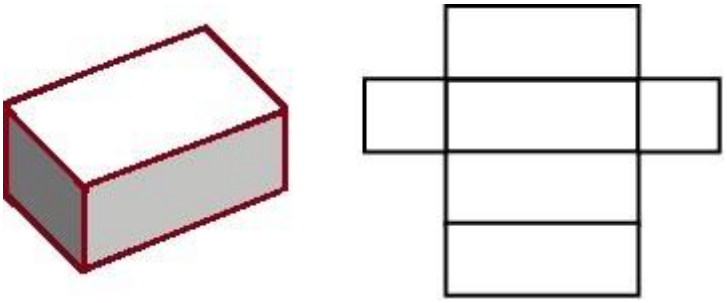
Professor: Dr. Antonio Carlos Sobieranski

Araranguá

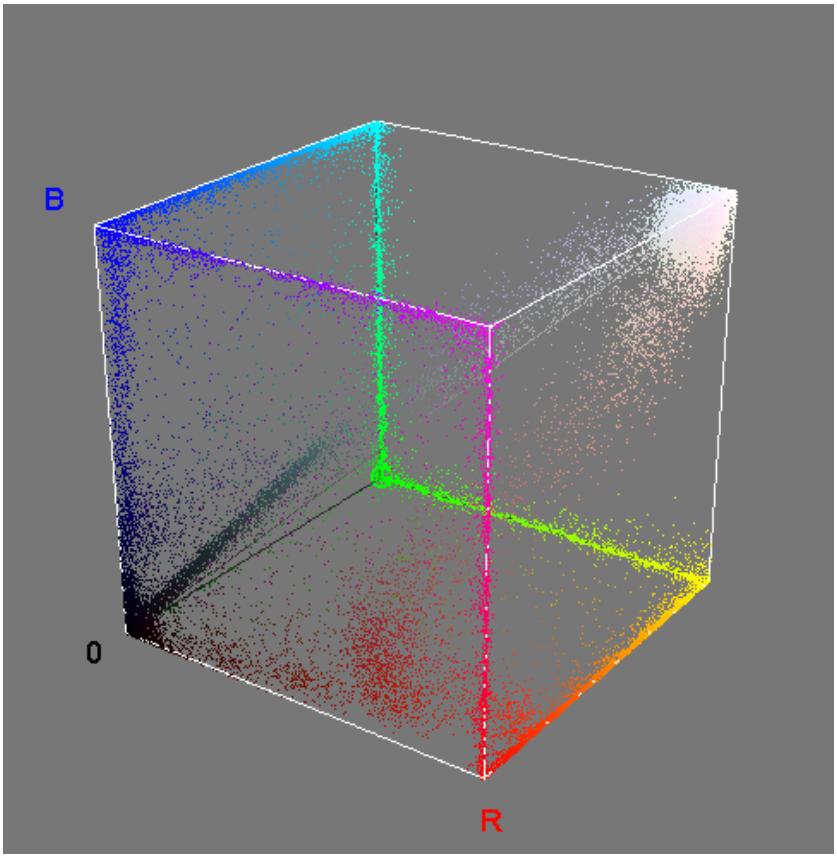
2018

1. SELECIONANDO COM UM CUBO / PARALELEPIPEDO

Todos os sólidos geométricos são formados pela união de figuras planas. Para exemplificar melhor, confira abaixo a planificação do paralelepípedo reto:



E temos nosso cubo de distribuição de cores, inspecionado pelo Color Inspector pode-se notar o cubo tridimensional, R para RED, G para GREEN, e B para BLUE. O zero identifica onde é o local mais escuro e se canto oposto o local mais claro.



A ideia aqui é criar esta forma geométrica dentro do cubo de cores, para isto, é necessário utilizar uma linguagem de programação, na minha preferência, decidi usar Python 3. O algoritmo para realizar isto pode ser visualizado a seguir.

```python
import cv2
import numpy as np

img = cv2.imread("estrela.jpg")
#print ("dados da imagem: ",img.shape)
altura=img.shape[0]
largura=img.shape[1]
canais=img.shape[2]
red = float(input("Digite o valor do ponto RED max: "))
red1 = float(input("Digite o valor do ponto RED min: "))
green = float(input("Digite o valor do ponto GREEN max: "))
green1 = float(input("Digite o valor do ponto GREEN min: "))
blue = float(input("Digite o valor do ponto BLUE max: "))
blue = float(input("Digite o valor do ponto BLUE min: "))

for i in range(altura):
    for j in range(largura):
        if (img[i,j,2] < red and img[i,j,2]  > red1 and img[i,j,1] < green and mg[i,j,1] > green1  and img[i,j,0] < blue and img[i,j,0] > blue1):

            img[i,j,2] = 255
            img[i,j,1] = 255
            img[i,j,0] = 255

cv2.imwrite("estrela-cubo.jpg",img)
print("Imagem gerada com sucesso!")

```

Como havia conversado com o professor após uma aula, poderia utilizar o OpenCV, para abrir as imagens e realizar algumas operações. Então realizo a importação desta biblioteca.

Leio a imagem, defino sua altura, largura em pixels. Faço a leitura dos extremos do cubo ou paralelepípedo que o usuário deseja criar, e por meio de dois “for” consigo fazer a varredura da imagem, verificando se o pixel atual se encontra dentro ou fora do espectro definido pelo usuário, se sim, os valores de RGB do pixel recebem o máximo, neste caso, imagem de 8bits, 255.

Após isto, é necessário apenas salvar a imagem alterada, e para questão de informação para o usuário, imprime-se uma mensagem de sucesso na tela.

O resultado obtido nesta etapa, pode ser visualizado a seguir.

Utilizando as configurações:

```
Digite o valor do ponto RED max: 300
Digite o valor do ponto RED min: 130
Digite o valor do ponto GREEN max: 255
Digite o valor do ponto GREEN min: 100
Digite o valor do ponto BLUE max: 255
Digite o valor do ponto BLUE min: 70
Imagem gerada com sucesso!
```

2. SELECIONANDO COM UMA ESFERA

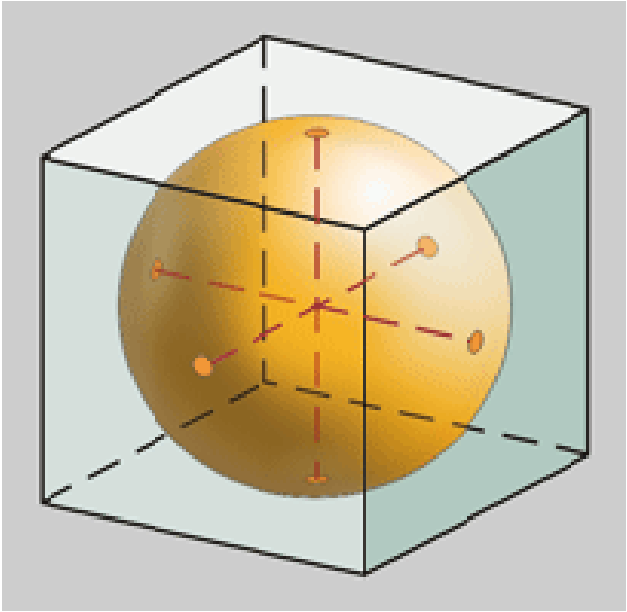
Uma esfera pode ser definida como "uma sequência de pontos alinhados em todos os sentidos à mesma distância de um centro comum". É tida também como um sólido geométrico formado por uma superfície curva contínua, cujos pontos estão equidistantes de um outro fixo e interior, chamado centro, ou seja, é uma superfície fechada de tal forma que todos os pontos dela estão à mesma distância de seu centro.

Para fins de calcular, temos que, uma esfera em coordenadas retangulares é representada pela equação:

$$(x-a)^2 + (y-b)^2 + (z-c)^2 \leq r^2$$

Onde, a, b, c são as coordenadas do centro da esfera nos eixos x, y, z respectivamente, e r é o raio da esfera.

Para nossa aplicação, a, b, c, são os Pontos R,G,B do centro da esfera no cubo RGB, x,y e z, são nossos eixos, neste caso o próprio RGB, e o r é o raio da esfera, que no algoritmo deverá ser definido pelo usuário, quanto maior o raio, maior a área de pixels a ser abordada pelo algoritmo.



O código para descrever esta função pode ser visto na figura a seguir.

```python
import cv2
import numpy as np

img = cv2.imread("estrela.jpg")
#print ("dados da imagem: ",img.shape)
altura=img.shape[0]
largura=img.shape[1]
canais=img.shape[2]

raio = input("Digite o valor do raio:")
raio2 = float(raio)**2
r=input("Digite o valor red: ")
g=input("digite o valor green: ")
b=input("Digite o valor blue: ")

for i in range(altura):
    for j in range(largura):
        dist=(float(img[i,j,2])-float(r))**2 + (float(img[i,j,1])-float(g))**2 + (float(img[i,j,0])-float(b))**2
        #print("dist = ",dist) #debug
        if (float(dist) < float(raio2)):
            img[i,j,2] = 255
            img[i,j,1] = 255
            img[i,j,0] = 255


cv2.imwrite("estrela-esfera.jpg",img)
cv2.imshow("Estrela esfera",img)
cv2.waitKey()
print("Imagem gerada com sucesso!")

```

Partindo do mesmo princípio do código anterior, se abre a imagem desejada, lê-se os dados da imagem (altura, largura e canais de cor), pede-se para o usuário digitar o raio, e calcula-se o raio quadrático. Com isso, pede-se também os valores RGB, do centro da esfera. Agora sim, podemos realizar a formula explicada anteriormente, fazendo com que todos os pixels da varredura que se encontrarem dentro da esfera definida pelo usuário sejam mudados para branco.

Com isso, temos os seguintes resultados de imagens:

Os valores para gerar esta imagem foram:

```
Digite o valor do raio:3672
Digite o valor red: 2958
digite o valor green: -2182
Digite o valor blue: 505
```

3 SELECIONANDO COM A DISTÂNCIA DE MAHALANOBIS

A distância de Mahalanobis é uma medida de distância introduzida pelo matemático indiano Prasanta Chandra Mahalanobis em 1936. É baseada nas correlações entre variáveis com as quais distintos padrões podem ser identificados e analisados. É uma estatística útil para determinar a similaridade entre uma amostra desconhecida e uma conhecida. Distingue-se da distância euclidiana já que tem em conta as correlações do conjunto de dados e é invariante à escala, ou seja, não depende da escala das medições.

Formalmente, a distância de Mahalanobis entre um grupo de valores com média $\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_p)^T$ e matriz de covariância **S** para um vector multivariado $x = (x_1, x_2, x_3, \ldots, x_p)^T$ é definida como:

$$D_M(x) = \sqrt{(x-\mu)^T S^{-1} (x-\mu)}.$$

Onde nossa matriz de covariância é dada por:

```
m_cov=np.zeros((3,3), dtype=np.float64)
z=int(0)
for z in range(nponto):
    m_cov[0,0]= ((vred[z]-m[0])*(vred[z]-m[0])) + m_cov[0,0]
    m_cov[1,0]= ((vred[z]-m[0])*(vgreen[z]-m[1])) + m_cov[1,0]
    m_cov[2,0]= ((vred[z]-m[0])*(vblue[z]-m[2])) + m_cov[2,0]
    m_cov[2,1]= ((vgreen[z]-m[1])*(vblue[z]-m[2])) + m_cov[2,1]
    m_cov[1,1]= ((vgreen[z]-m[1])*(vgreen[z]-m[1])) + m_cov[1,1]
    m_cov[2,2]= ((vblue[z]-m[2])*(vblue[z]-m[2])) + m_cov[2,2]

m_cov[0,2]= m_cov[2,0]
m_cov[1,2]= m_cov[2,1]
m_cov[0,1]= m_cov[1,0]
```

O vetor do meu ponto de cor – a média do ponto de cor, multiplicado pelo vetor do ponto de cor, menos a média. A media dos valores dá o ponto da matriz de covariância.

Meu X, ou vetor multivalorado é dado por:

```
21  for w in range(nponto):
22      pred=input("Digite o ponto red: ")
23      pgreen=input("Digite o ponto green: ")
24      pblue=input("Digite o ponto blue: ")
25      vred[w]=float(pred)
26      vgreen[w]=float(pgreen)
27      vblue[w]=float(pblue)
```

Onde pred,pgreen e pblue são pontos digitados pelo usuário, criando o vetor de cores.

Meu vetor de média é dado por:

```
28      m[0] = float(m[0])+float(vred[w])
29      m[1] = float(m[1])+float(vgreen[w])
30      m[2] = float(m[2])+float(vblue[w])
31
32  m[0] = m[0] / float(nponto)
33  m[1] = m[1] / float(nponto)
34  m[2] = m[2] / float(nponto)
```

Onde, m[0] é a média do Red, m[1] a média do Green e m[2] a média do Blue.

Por fim, o cálculo dá distância é dado pela transposta vezes a matriz inversa, sendo efetuado pelo seguinte algoritmo:

```
53 ▼ for i in range(2):
54       for j in range(2):
55           m_cov[i,j]= (m_cov[i,j] / n)
56   i_cov = inv(m_cov)
57   vet=[0]*3
58 ▼ for g in range(altura):
59 ▼     for f in range(largura):
60           vet[0]=float(img[g,f,2])-float(m[0])
61           vet[1]=float(img[g,f,1])-float(m[1])
62           vet[2]=float(img[g,f,0])-float(m[2])
63           tvet=np.transpose(vet)
64           aux = np.dot(i_cov,vet)
65           dist=np.dot(tvet,aux)
66           d=float(dist)
67 ▼         if(d < raiooo):
68               #print("D: ",d)
69               img[g,f,2] = 255
70               img[g,f,1] = 255
71               img[g,f,0] = 255
```

Os resultados da distância de mahalanobis podem ser conferidos abaixo:

Os valores para os pontos foram:

```
Quantos pontos você deseja introduzir? 4
Qual o tamanho do raio que deseja? 25
Digite o 0º ponto red: 185
Digite 0º o ponto green: 78
Digite 0º o ponto blue: 57
Digite o 1º ponto red: 255
Digite 1º o ponto green: 217
Digite 1º o ponto blue: 152
Digite o 2º ponto red: 162
Digite 2º o ponto green: 65
Digite 2º o ponto blue: 38
Digite o 3º ponto red: 154
Digite 3º o ponto green: 55
Digite 3º o ponto blue: 13
Programa finalizado
```

O

4 SELECIONANDO COM K-VIZINHOS

O K-Vizinhos foi utilizado como base o algoritmo da esfera, porém agora, com inúmeros pontos, o usuário decide quantos pontos deseja inserir (quantas esferas), cada uma com seu centro e raio distintos. Para melhor entendimento, observe o código a seguir.

```python
import cv2
import numpy as np
import sys

img = cv2.imread("estrela.jpg")

altura=img.shape[0]
largura=img.shape[1]
canais=img.shape[2]

resp = "y"
while resp == "y" or resp=="Y":
    resp = input("Deseja adicionar um ponto? [Y] [N]")
    if (resp == "n" or resp == "N"):
        print("Programa finalizado")
        cv2.imshow("Estrela Vizinho",img)
        cv2.waitKey()
        sys.exit()

    pred=input("Digite o ponto red: ")
    pgreen=input("Digite o ponto green: ")
    pblue=input("Digite o ponto blue: ")
    raio=input("Digite o raio: ")
    raio2=float(raio)**2

    for i in range(altura):
        for j in range(largura):
            distancia=(float(img[i,j,2])-float(pred))**2 + (float(img[i,j,1])-float(pgreen))**2 + (float(img[i,j,0])-float(pblue))**2
            #print("dist = ",dist) #debug
            if (float(distancia) < float(raio2)):
                img[i,j,2] = 255
                img[i,j,1] = 255
                img[i,j,0] = 255

    cv2.imwrite("estrela-vizinho.jpg",img)
    img =cv2.imread("estrela-vizinho.jpg")
    print("Ponto adicionado com sucesso!")

```

Realiza-se a leitura de vários pontos com seus respectivos raios, e com isso, gera-se várias esferas dentro do cubo RGB, mesmo procedimento do método por esfera, porém agora, utilizando inúmeras esferas, com diferentes raios.

Abaixo pode-se ver o resultado deste método de seleção.

Os valores dos pontos RGB utilizados foram:

```
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 444
Digite o ponto green: -82
Digite o ponto blue: 61
Digite o raio: 352
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 586
Digite o ponto green: 367
Digite o ponto blue: 0
Digite o raio: 408
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 1033
Digite o ponto green: -456
Digite o ponto blue: -153
Digite o raio: 1081
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 255
Digite o ponto green: 238
Digite o ponto blue: 192
Digite o raio: 36
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 90
Digite o ponto green: 37
Digite o ponto blue: 21
Digite o raio: 21
Ponto adicionado com sucesso!
```

```
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 240
Digite o ponto green: 217
Digite o ponto blue: 163
Digite o raio: 31
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 197
Digite o ponto green: 178
Digite o ponto blue: 110
Digite o raio: 5
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]Y
Digite o ponto red: 213
Digite o ponto green: 192
Digite o ponto blue: 129
Digite o raio: 5
Ponto adicionado com sucesso!
Deseja adicionar um ponto? [Y] [N]N
Programa finalizado
```

5 CONCLUSÃO

Com isso, podemos concluir que o melhor método pelos testes realizados é o método de mahalanobis, o trabalho serviu para nos fazer aprender como trabalhar com esses métodos e também a melhorar o entendimento do cubo RGB, e como manipula-lo. Contudo, sinto que o aprendizado foi de grande importância, não somente para a continuidade da matéria, mas também para implementações no projeto final e em projetos futuros.

6 IMAGENS DE RESULTADO