Richard Ngo

Code Information:

To replicate the super pixel process, my code follows the algorithm stated in the State-of-the-Art Super pixel Methods report, included on the assignment page:

```
Algorithm 1. SLIC superpixel segmentation
  /* Initialization */
  Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling
  pixels at regular grid steps $S$.
  Move cluster centers to the lowest gradient position in a $3 \times 3$
  neighborhood.
  Set label $l(i) = -1$ for each pixel $i$.
  Set distance $d(i) = \infty$ for each pixel $i$.

  repeat
   /* Assignment */
   for each cluster center $C_k$ do
     for each pixel $i$ in a $2S \times 2S$ region around $C_k$ do
       Compute the distance $D$ between $C_k$ and $i$.
       if $D < d(i)$ then
         set $d(i) = D$
         set $l(i) = k$
       end if
    end for
    end for
    /* Update */
    Compute new cluster centers.
    Compute residual error $E$.
  until $E \leq$ threshold
```

It is very similar to the K means algorithm, with the exception being that Distance between each K mean is calculated using some extra parameters, and that each K center now has a limited region of pixels to search in.

S, a variable representing the search space of each K center, was calculated to be equal to the square root of the number of pixels of the image, divided by the number of centers to be found

S=Sqrt(N/K)

To calculate the distance, the following formula also included in the article was used

$$d_c = \sqrt{(l_j - l_i)^2 + (a_j - a_i)^2 + (b_j - b_i)^2},$$

$$d_s = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2},$$

$$D' = \sqrt{\left(\frac{d_c}{N_c}\right)^2 + \left(\frac{d_s}{N_s}\right)^2}.$$

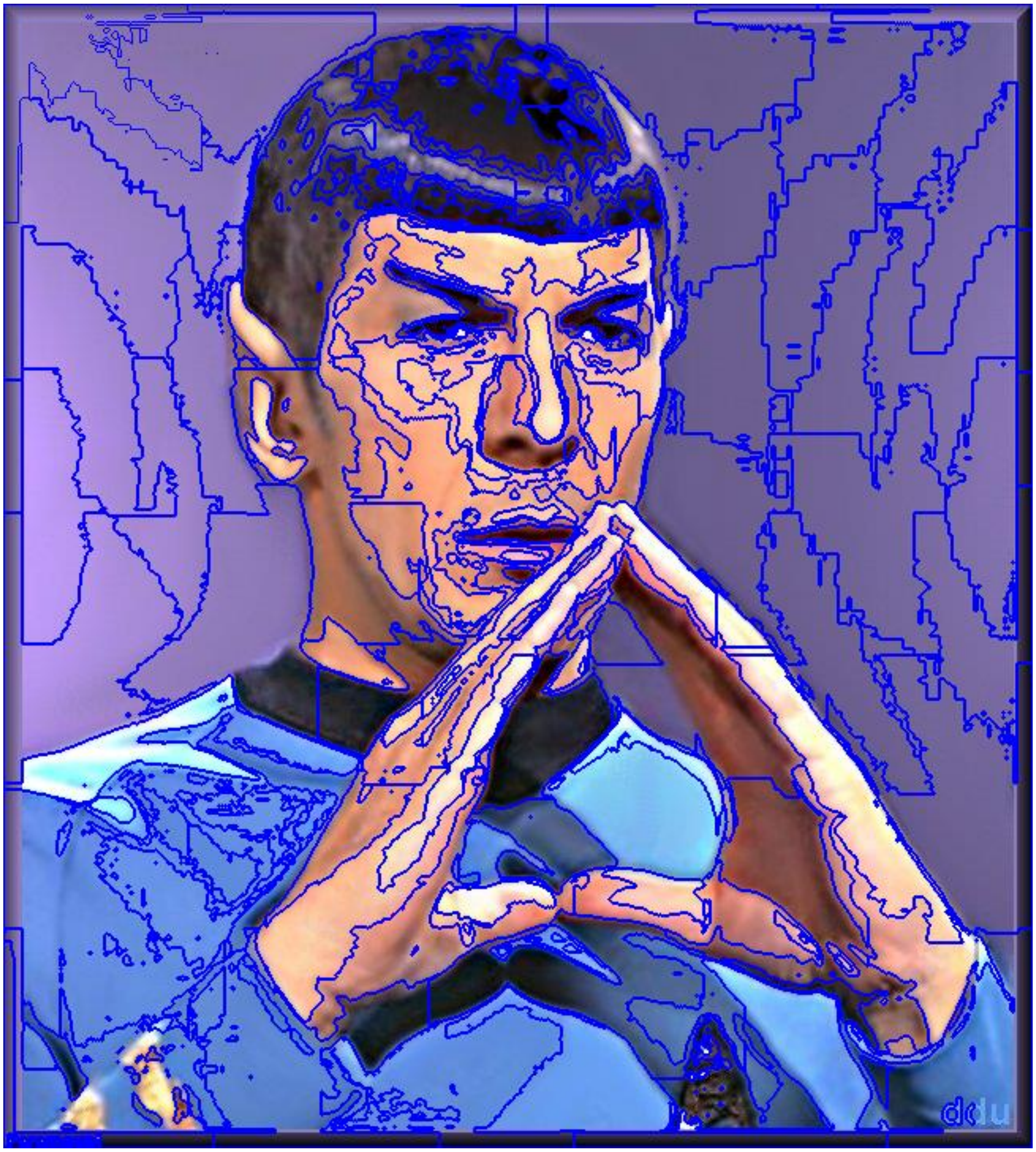
However, instead of using CIELAB values to calculate the distance of the color space, the RGB values of the image was used instead. The main thing that differentiates this distance function from the K means distance function is the inclusion of individual weights for its color and spatial features, what are seen as variables Nc and Ns respectively. The value of Ns was chosen to be equal to the S variable calculated above. The value of Nc was chosen to be equal to be m variable specified by the user.

Overall, the algorithm chooses K amount of centers, updates the centers based on the values pixels most closest to it within a certain range with each iteration, and then finalizes the centers when they stop moving. The pixels belong to the centers that they had the closest distance with over the iterations.

Influence of m

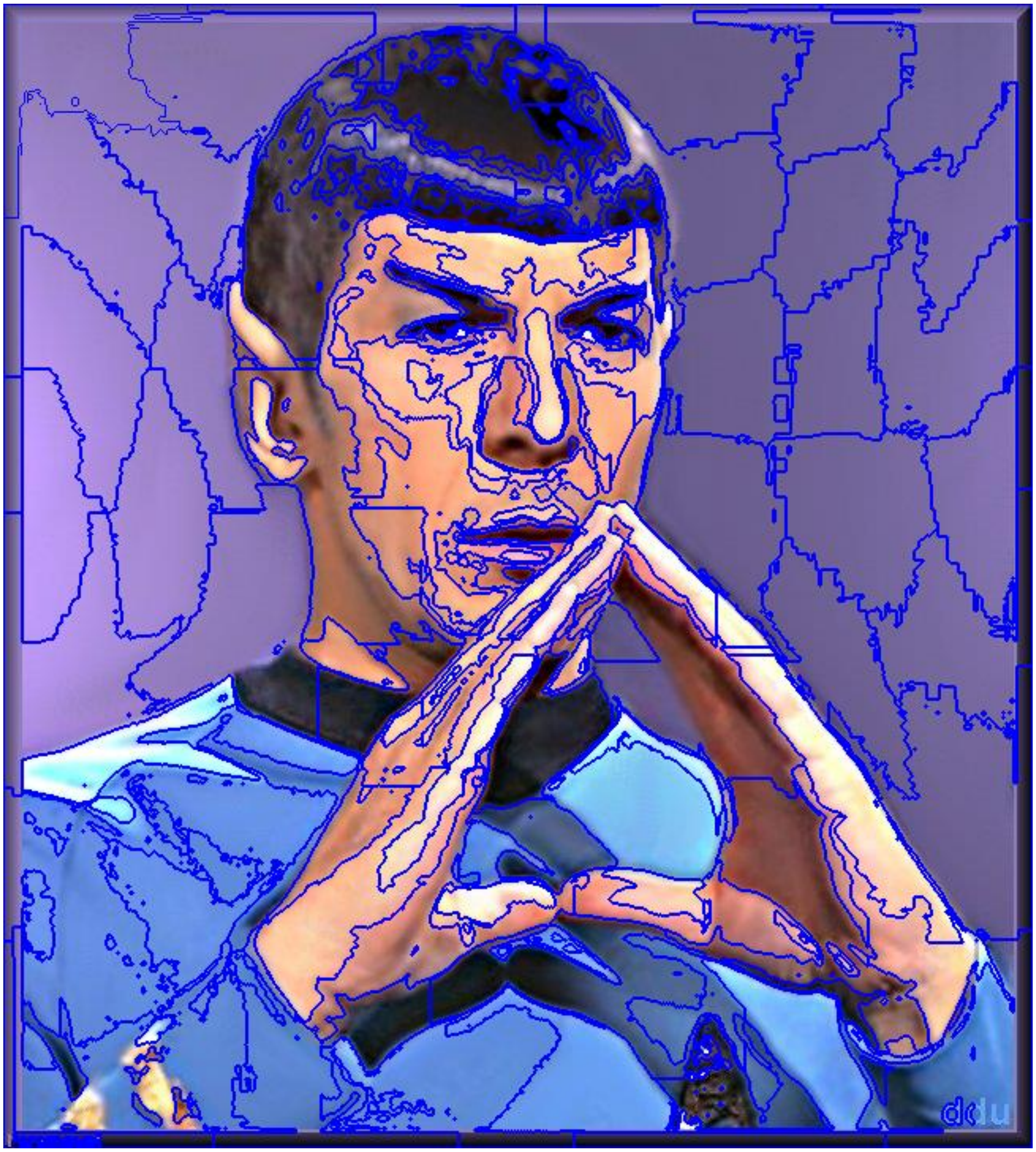
The influence of m, or the weight on the color features that was observed was that, the smaller the m, the more ridged and complex or spikey each of the segments became, and the larger then, the more smoother and rounded the edges were able to be, given that the weight on the color features, S was equal to sqrt(N/K). The values of m that were tested was were m = 5, 10 15, and the resulting images are included below. The influence of S can also be viewed by comparing the super pixels of images of different K center numbers, given that it that S is dependent on K, and was observed to have an effect on the results similar to that of m (larger K center numbers result in less spikey or complex segments).

K = 64, m = 5

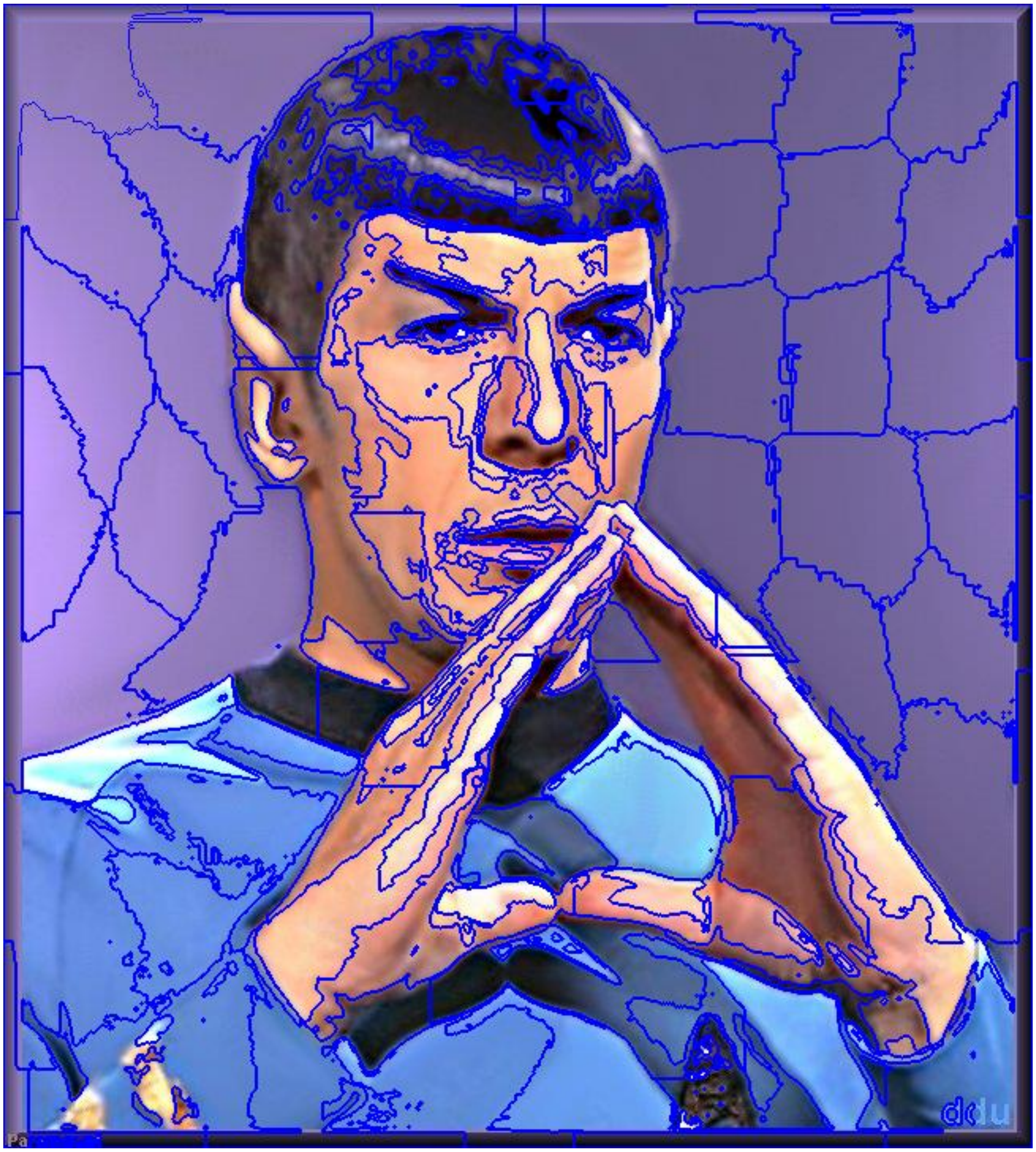


Richard Ngo

k = 64, m = 10



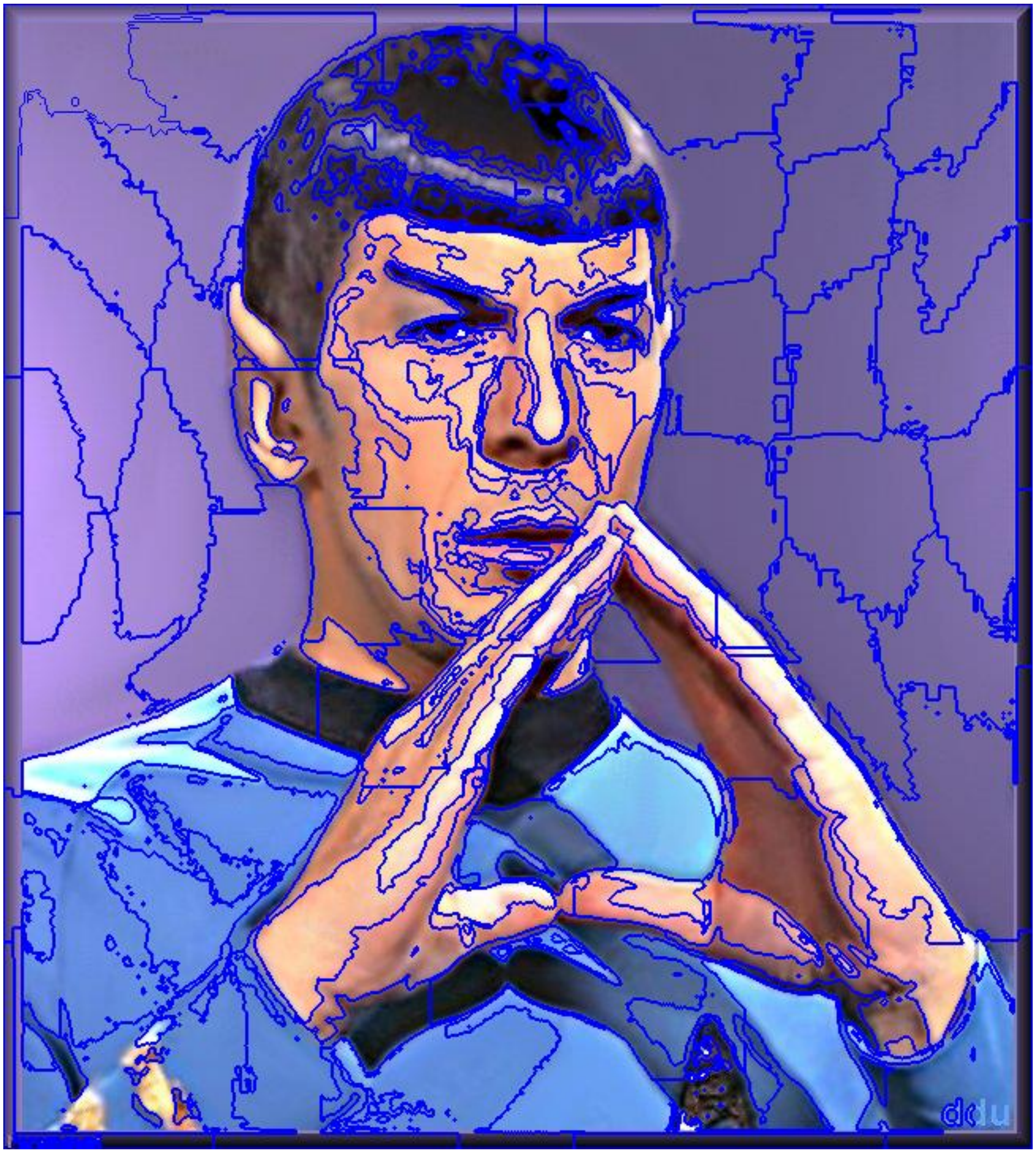
K = 64, M = 15



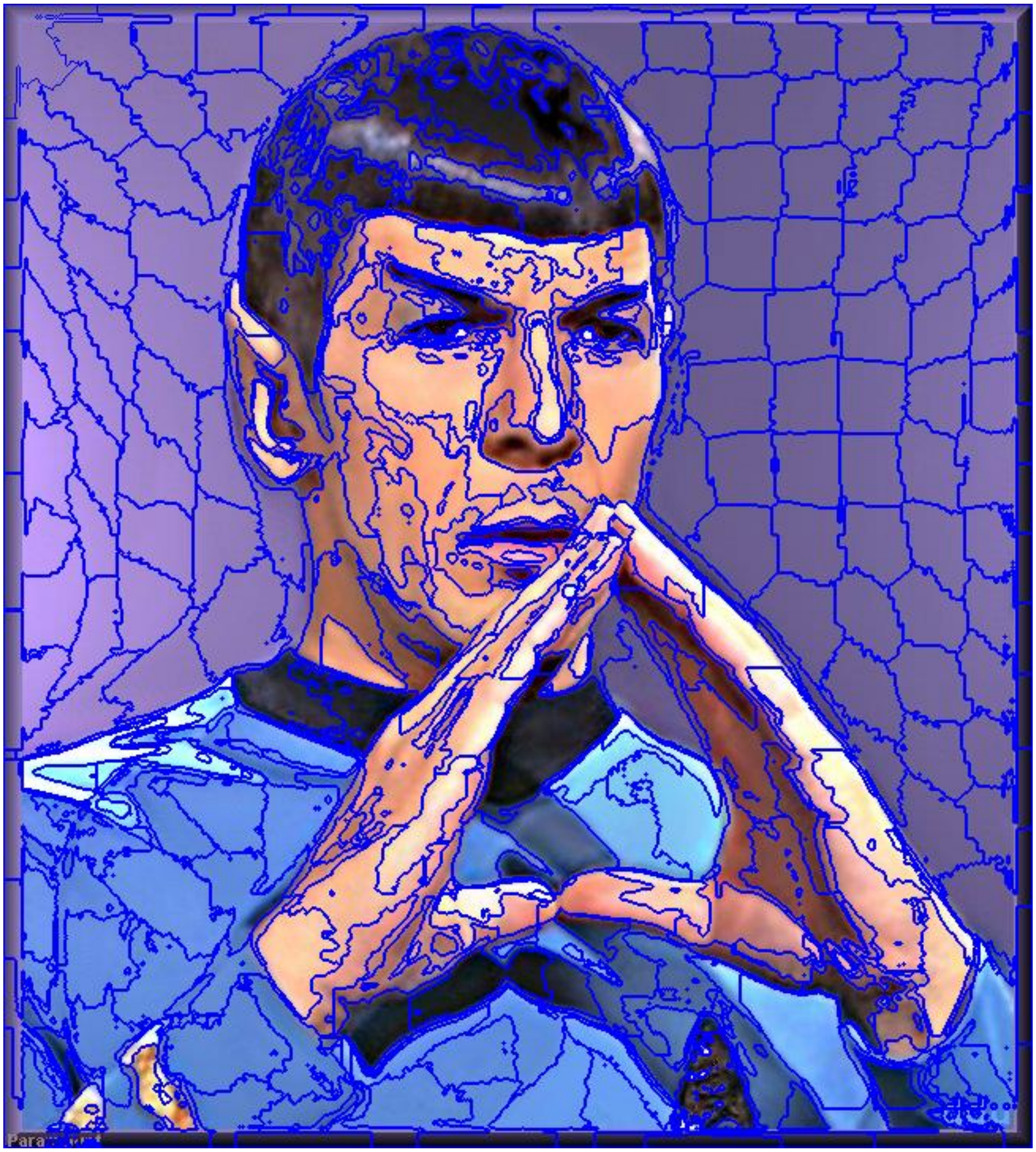
Richard Ngo

Spock.png Results

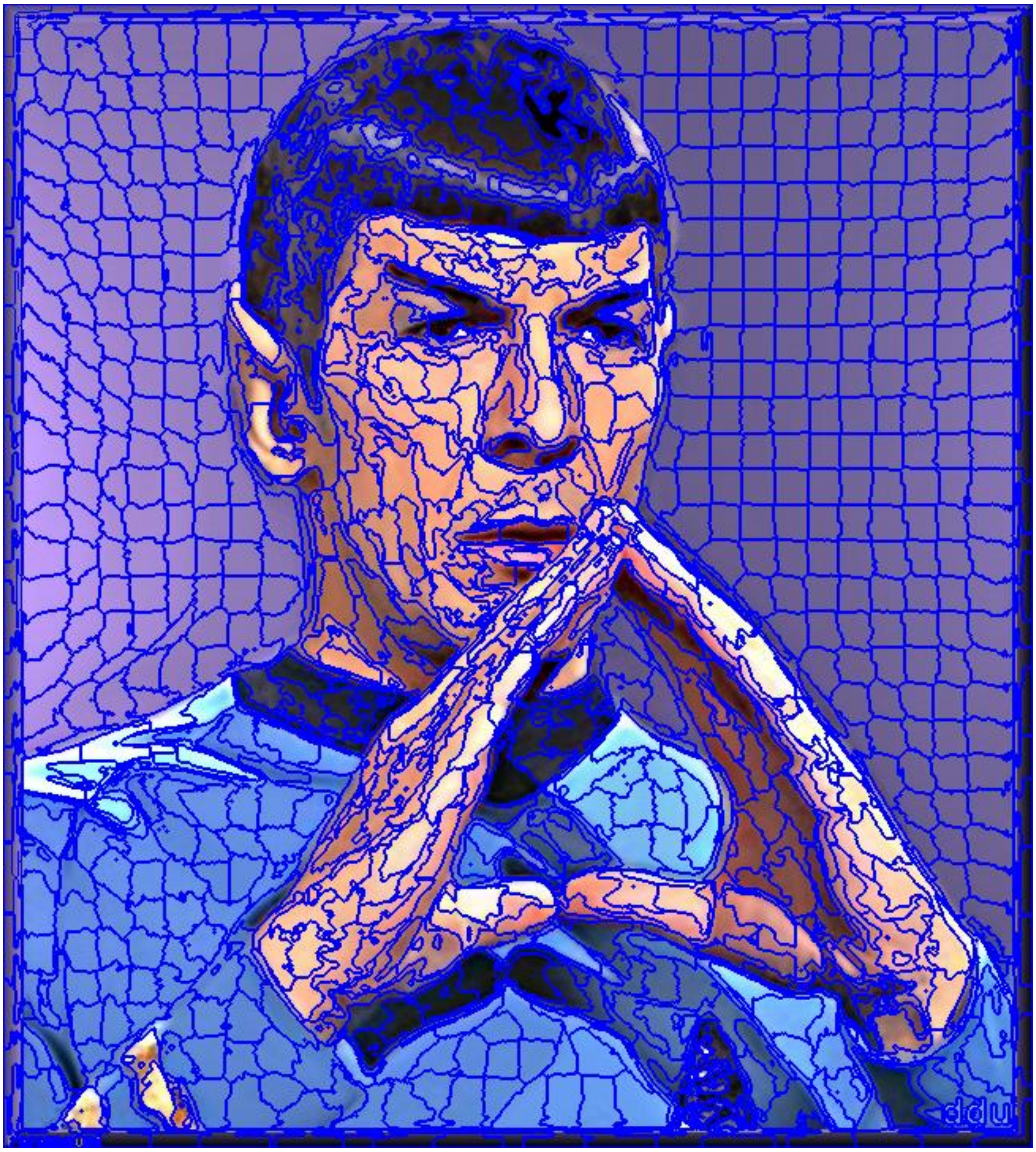
K = 64, m = 10



K = 256, m = 10

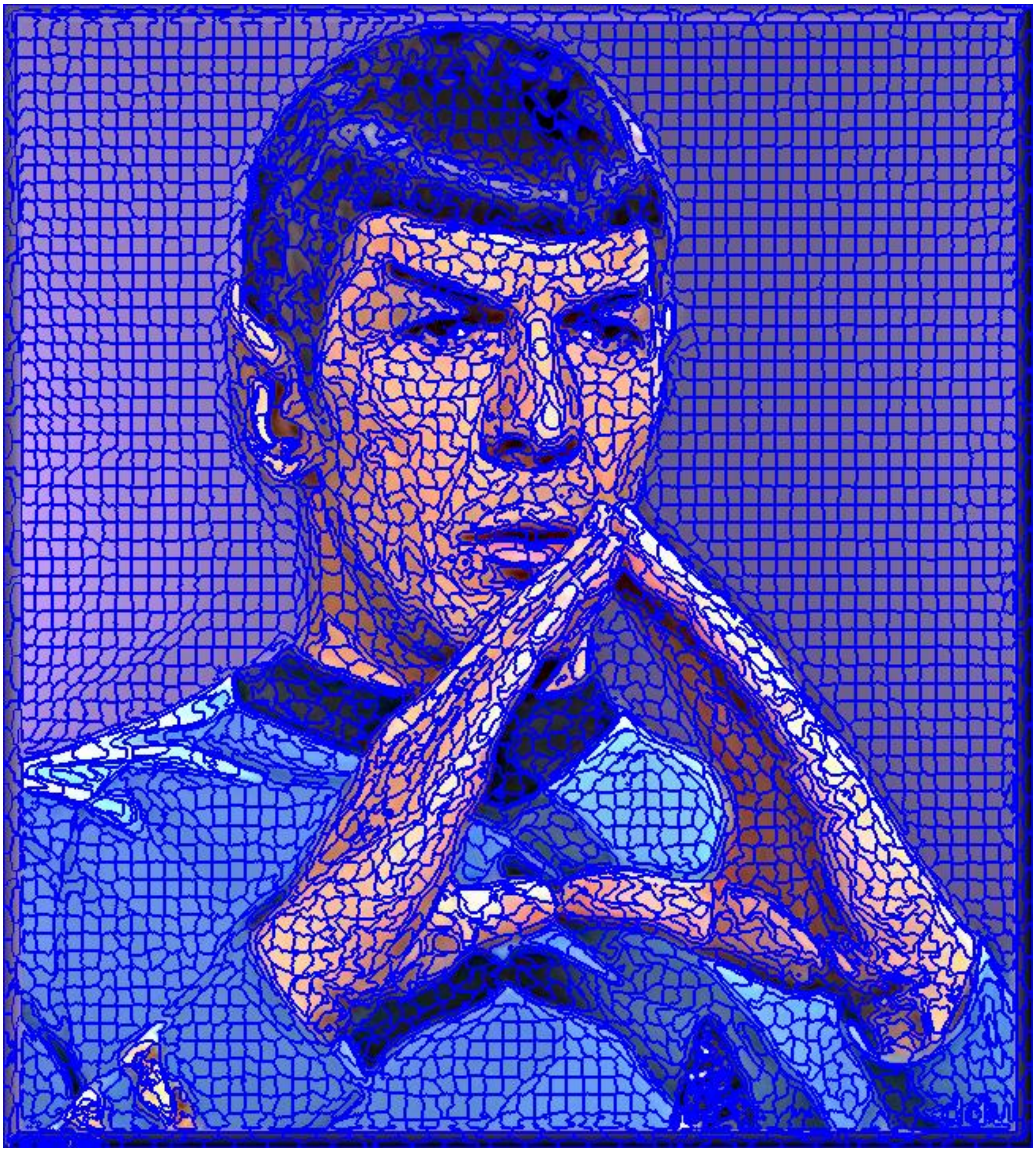


K = 1024, m = 10



K = 4096, m = 10

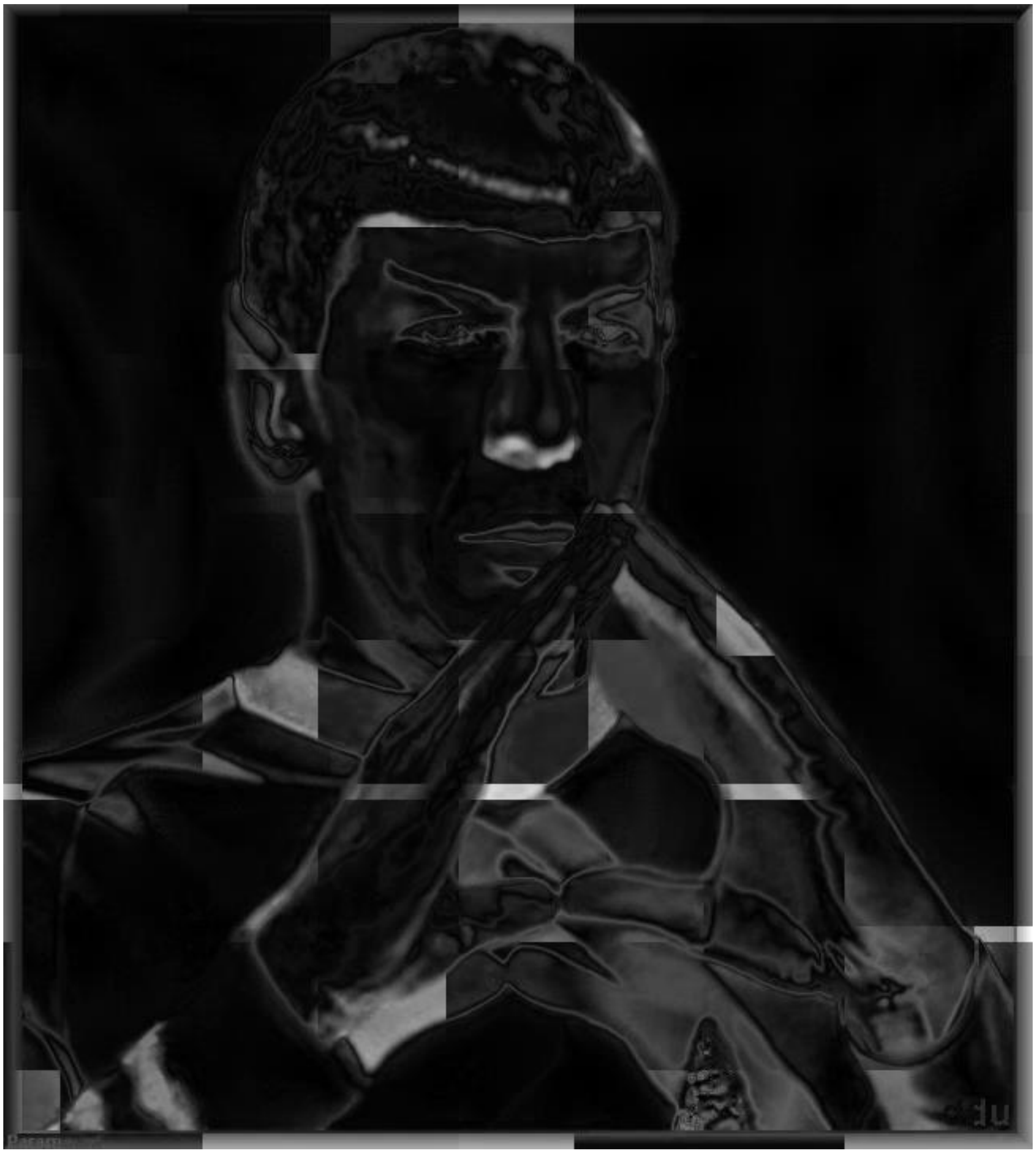
Richard Ngo



Richard Ngo

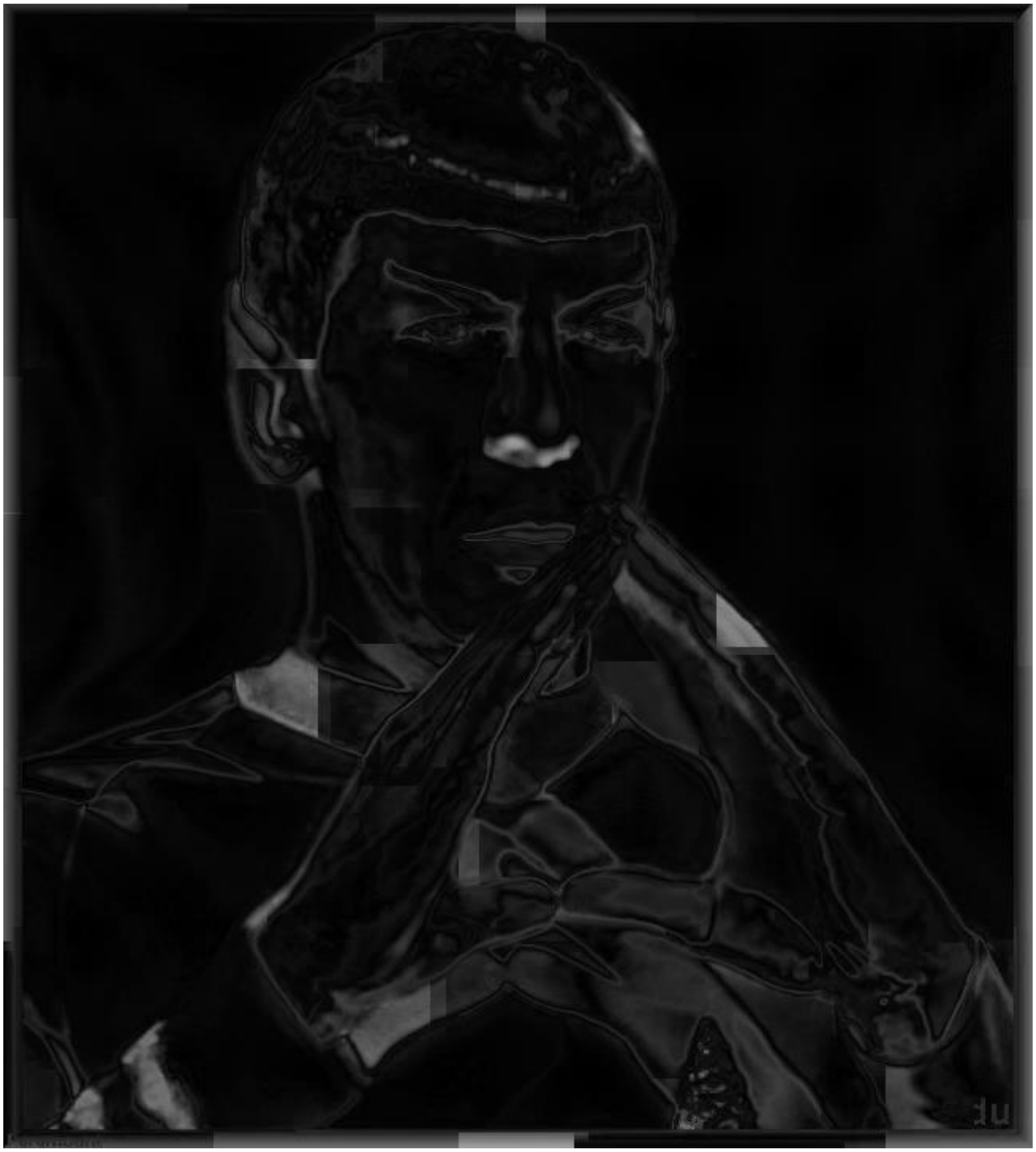
Error map of spock.png with K = 64, m=10 (Max intensity set at distance error 40.01)

Initial Error Map



Richard Ngo

Final Error Map



Code used:

```
# -*- coding: utf-8 -*-


# CompVisHwExCredit.py - Implimentation of Super Pixel Algorithm

# Created on 12/12/19

# @author: Richard Ngo


import numpy as np

import cv2

import math


def showimg(img):

    cv2.namedWindow('showimg', flags=cv2.WINDOW_NORMAL)

    cv2.imshow('showimg', img)

    #cv2.resizeWindow('showimg', (int(len(img[0])*2), int(len(img)*2)))

    #print(img)

    cv2.waitKey(0)

    cv2.destroyAllWindows()




def superPixelFind(R, nclus, m):

    imgH = np.size(R, axis = 0)

    imgW = np.size(R, axis = 1)

    imgD = np.size(R, axis = 2)


    N = imgH*imgW

    S = np.int32(pow(N/nclus,1/2))
```

```
Kcenter = np.zeros((1,nclus,imgD), dtype=np.float64)

KcenterP = np.copy(Kcenter)

KPos = np.zeros((1,nclus,2), dtype=np.float64)

KPosum = np.zeros((1,nclus,2), dtype=np.float64)

Ksum = np.zeros((1,nclus,imgD), dtype=np.float64)

Kcount =np.zeros((1,nclus,imgD), dtype=np.float64)

stop = 0


imgHin = imgH/pow(nclus,1/2)

imgWin = imgW/pow(nclus,1/2)

for k in range(nclus):

    #Kcenter[0,k,:] =
R[np.int32(imgH/nclus*k+imgH/(nclus*2)),np.int32(imgW/nclus*k+imgW/(nclus*2)),:]

    xpos = np.int32(np.int32(k*imgWin)%imgW+imgWin/2)

    ypos = np.int32(math.floor(k*imgWin/imgW)*imgHin+imgHin/2)

    #xpos = np.int32(k*(N/nclus))%imgW

    #ypos = math.floor(k*(N/nclus)/imgW)

    KPos[0,k,0] = ypos

    KPos[0,k,1] = xpos

    Kcenter[0,k,:] = R[ypos,xpos,:]


#outtest = np.zeros((imgH,imgW), dtype=np.uint8)

#for k in range(nclus):

#    print(KPos[0,k,0],KPos[0,k,1])

#    outtest[np.int32(KPos[0,k,0]),np.int32(KPos[0,k,1])] = 200

#return outtest


Kset = np.zeros((imgH,imgW,2), dtype=np.float64)

Kset[:,:,0]=-1
```

```
errorHold = np.zeros((imgH,imgW,2), dtype=np.float64)
started = 0
while stop == 0:
    for k in range(nclus):
        voidU = min(np.int32(KPos[0,k,0]-S),0)
        voidD = max(np.int32(KPos[0,k,0]+S+1)-imgH,0)
        voidL = min(np.int32(KPos[0,k,1]-S),0)
        voidR = max(np.int32(KPos[0,k,1]+S+1)-imgW,0)
        for y in range(np.int32(KPos[0,k,0]-S-voidU),np.int32(KPos[0,k,0]+S+1-voidD)):
            for x in range(np.int32(KPos[0,k,1]-S-voidL),np.int32(KPos[0,k,1]+S+1-voidR)):
                checkintensity=pow(np.sum(np.power(np.subtract(R[y,x,:],Kcenter[0,k,:]),2)),1/2)
                checkdistance=pow(pow(KPos[0,k,0]-y,2)+pow(KPos[0,k,1]-x,2),1/2)
                checkmin = pow(pow(checkintensity/m,2)+pow(checkdistance/S,2),1/2)
                if Kset[y,x,1]>checkmin or Kset[y,x,0]==-1:
                    Kset[y,x,1] = checkmin
                    Kset[y,x,0] = k
    for y in range(imgH):
        for x in range(imgW):
            if Kset[y,x,0] != -1:
                Ksum[0,np.int32(Kset[y,x,0]),:]+=R[y,x,:]
                KPosum[0,np.int32(Kset[y,x,0]),0]+=y
                KPosum[0,np.int32(Kset[y,x,0]),1]+=x
                Kcount[0,np.int32(Kset[y,x,0]),:]+=1
    print("Group's member count: ",Kcount[0,:,0])
    Kcount[Kcount==0]=1
    Kcenter=np.divide(Ksum,Kcount)
    KPos=np.divide(KPosum,Kcount[:,:,0:2])
    KPosum*=0
```

```
        Ksum*=0

        Kcount*=0


        changed = np.sum(np.abs(np.subtract(Kcenter,KcenterP)))

        print("Change in centers: ",changed)


        if (changed<=0):

            stop = 1

        #KcenterP=np.copy(Kcenter)

        KcenterP[:,:,:]=Kcenter[:,:,:]

        if started == 0:

            errorHold[:,:,0] = Kset[:,:,1]

            started = 1


    errorHold[:,:,1] = Kset[:,:,1]

    print("Kmeans Selected")


    pcolor = np.copy(R)

    pavg = np.copy(R)


    for y in range(imgH):

        for x in range(imgW):

            if Kset[y,x,0] != -1:

                pavg[y,x,:] =
R[np.int32(KPos[0,np.int32(Kset[y,x,0]),0]),np.int32(KPos[0,np.int32(Kset[y,x,0]),1]),:]


    for i in range(nclus-1):

        ret,thresh1 = cv2.threshold(np.uint8(Kset[:,:,0]),i,255,cv2.THRESH_BINARY_INV)

        ret,thresh2 = cv2.threshold(np.uint8(Kset[:,:,0]),i+1,255,cv2.THRESH_BINARY)
        SPix = thresh1+thresh2
        trash, contours, trash2 = cv2.findContours(SPix, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        pcolor=cv2.drawContours(pcolor, contours, -1, (255,0,0), 0)

    return pcolor, pavg, errorHold

K = [64,256,1024,4096]
M = [5,15]
inputimg = cv2.imread("spock.png", cv2.IMREAD_COLOR)

for m in M:
    outputimg, outputimgFill, errorHold = superPixelFind(inputimg, 64, m)
    cv2.imwrite("spock_Seg_64_M_"+str(m)+"_.png", outputimg)
    cv2.imwrite("spock_Seg_64_M_"+str(m)+"_Filled.png", outputimgFill)

for k in K:
    outputimg, outputimgFill, errorHold = superPixelFind(inputimg, k, 10)
    cv2.imwrite("spock_Seg_"+str(k) + ".png", outputimg)
    cv2.imwrite("spock_Seg_"+str(k) + "Filled.png", outputimgFill)
    maxHold = np.amax(errorHold[:,:,0])
    errorSNorm = np.uint8((errorHold[:,:,0]/maxHold)*255)
    errorFNorm = np.uint8((errorHold[:,:,1]/maxHold)*255)
    cv2.imwrite("spock_Seg_"+str(k) + "_InitialErr"+str(maxHold)+".png", errorSNorm)
    cv2.imwrite("spock_Seg_"+str(k) + "_FinalErr.png", errorFNorm)

K2 = [64,256,1024]
for I in range(300100):
    if I > 0:
        inputimg = cv2.imread(str(I)+".jpg", cv2.IMREAD_COLOR)
```

```
if inputimg is not None:
  for k in K2:
    outputimg, outputimgFill, errorHold = superPixelFind(inputimg, k, 10)
    cv2.imwrite(str(I)+"_"+str(k) + ".jpg", outputimg)
    cv2.imwrite(str(I)+"_"+str(k) + "Filled.jpg", outputimgFill)
```