CSE 4/573 Fall 2010 v2 Oct 31 2010

Homework #3 due Wednesday November 3

1. Construction of a granulometry function: submit an m-file GranFn.m with first line

```
function G_psi = GranFn(InIm,B)
```

where

InIm: A type uint8 logical binary image whose foreground pixels are 1

B : Structuring element for the granulometry, an nx2 type uint8 logical binary matrix whose foreground pixels are 1 and in which B(i,1) is the x-coord and B(i,2) is the y-coord of the ith pixel in B. The first opening uses B_1=B. The second opening uses B_2 defined as all foreground pixels in B_1 plus all pixels that are 4-connected to foreground pixels in B_1. For each n, B_{n+1} consists of all foreground pixels in B plus all pixels 4-connected to foreground pixels in B_n.

G_psi: The resulting granulometry function, a unit8 output image where each object pixel Gpsi(i,j) takes as its value the index n such that (i,j) survives the opening with B_n but not with B_{n+1}. Each background pixel in InIm should be given the value 0 in G_psi.

2. This problem explores how you might fuse foreground objects in a color image together. Submit an m-file RGBConn.m whose first line is

```
function OutIm = RGBConn(InIm,c_type)
```

where

InIm: An input type double normalized RGB image
c_type: A number which must be either 4 or 8
OutIm: The output type double normalized RGB image

For each pixel in InIm, define Br=sqrt(R^2+G^2+B^2) where R, G and B are the red, green and blue color values of that pixel, and Br is its brightness. Assume that your input image InIm consists of object pixels with Br>=0.50 and background pixels with Br<0.50. What RGBConn.m should do is repeatedly close the image until one of two things happens: either all foreground pixels are linked together into a single connected object, or no further changes occur with additional closing operations. You may use any of the Matlab image processing toolbox functions you like, such as imclose, bwmorph, bwlabel, etc. But note that none of these functions operate on RGB images. You will have to decide how to extend the functions that you use to RGB. Note: there is no unique way to extend the morphological operations from gray level to color images. Use your imagination in deciding how you want to define the closing of a color image.

3. The chord distribution (see p 239 ed 2, p 338 ed 3) of a blob can have very high computationally complex, i.e. take a lot of computing power to compute. A useful "Monte Carlo" approximation can be computed more efficiently, here is how. Pick two points on the boundary of the blob at random with all points having equal probability of being selected. Compute the chord (Euclidean distance between these two boundary points). Repeat many times. The distribution of the corresponding chord values will accurately approximate the true chord distribution. With that in mind, here is what you are asked to do:

Submit a m-file MCCD.m whose first line is

```
Function CD=MCCD(CC,n)
```

where

CC: the 8-connected Freeman chain code (see p. 236 ed 2, p 335 ed 3) for the blob's boundary

n: the number of bins in the chord distribution.

For instance, if n=20 and the largest chord you found was 65.6 pixels long, then CD should be a row vector of 20 components where the first component is the fracton of random chords you found which were between length 0 and length 65.6/20, the second is the fraction with Euclidean lengths between 65.6/20 and (2*65.6)/20, etc. Note: 1. If there are n bins and m pixels in CC, you should use n*m randomly selected pairs of points to estimate the chord distribution.

4. Write a Matlab script Moment_script.m that will:

1. Prompt for the name of a bitmap (.bmp) file. This file may be assumed to be a .bmp intensity image which contains a single 8-connected object against background pixels all equal to zero.
2. Then prompt for the maximum desired moment indices (M,N).
3. Compute and display an MxN type double matrix $\underline{\boldsymbol{\mu}}$ whose (i,j)th element is the (i,j)th central moment of the object.

To help you test your code, recall that the central moments are shift-invariant. So if you generate $\underline{\boldsymbol{\mu}}$ for any image test.bmp of the type described above, and then shift the object to a different location in the image creating a second image test_shifted.bmp, the $\underline{\boldsymbol{\mu}}$ for these two images should be identical (up to small round-off differences). You should not submit these test images, just submit Moment_script.m.