# Image processing #3

**고급건설재료학**

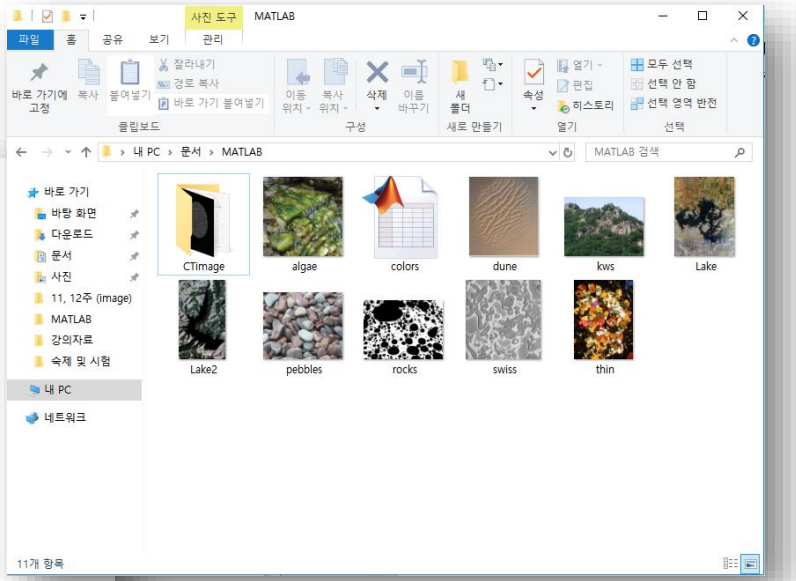
서울대 건설환경공학부 문주혁 교수

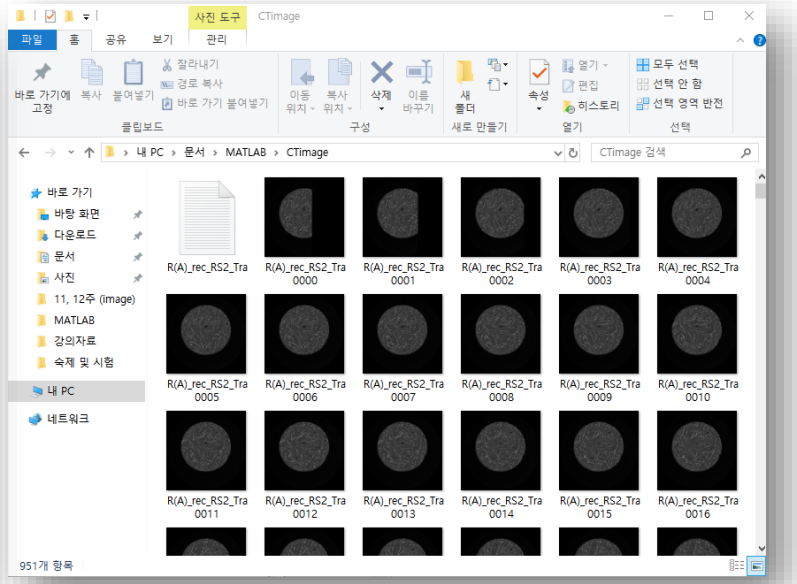
# Contexts

- #1. Introduction and Examples
- #2. Basics of Matlab, Image Processing Toolbox
- **#3. 3D input, Segmentation, Edge detection & Transformation**

# 3D data input in Matlab

```
>> clear all; close all
>> listing=dir('CTimage/*jpg');
>> nFiles=numel(listing);
>> ls=imread(['CTimage' '/' listing(1).name]);
>> [nRows nCols nLyrs] = size(ls);
nPix = nRows * nCols;
>> resize = 494;
cRow = round(nRows/2);
sRow = cRow-resize/2;
cCol = round(nCols/2);
sCol = cCol-resize/2;
>> for k = 1:nFiles;
  I = imread(['CTimage' '/' listing(k).name]);
  I2{k} = I(sRow:sRow+resize-1,sCol:sCol+resize-1);
end
```




변수 - I2

편집기 - fullcode.m

| | SE | SE.Neighborhood | SE.Dimensionality | L0 | L | listing |
|---|---|---|---|---|---|---|

1x950 cell

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 494x494 uint8 | 494x494 ui... | 494x494 ui... | 494x494 ui... | 494x494 ui... | 494x494 ui... |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |
| 6 | | | | | | |

# Segmentation #1

- Measuring the size of a lake
  - Google Earth RGB image

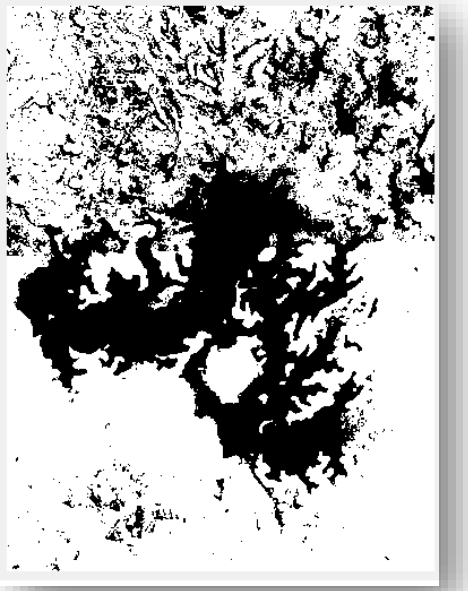
```
>> I=imread('lake.jpg');
>> G=rgb2gray(I);
>> level=graythresh(I);
>> bw=im2bw(I,level);
>> figure, imshow(bw)
```

```
>> cc=bwconncomp(1-bw,4);
>> graindata = regionprops(cc,'basic');
>> grain_areas = [graindata.Area];
>> [max_area, idx]=max(grain_areas)
```

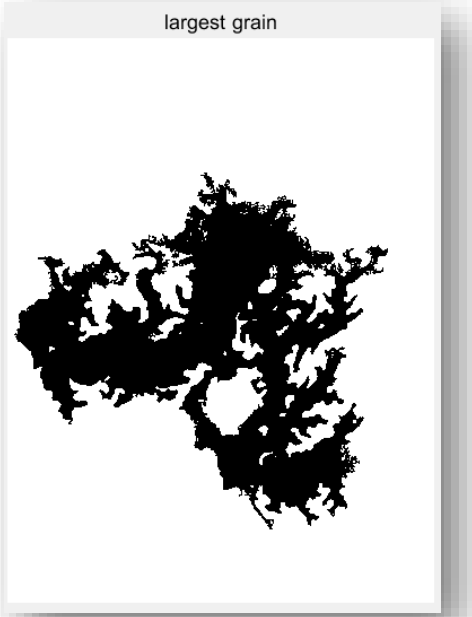
```
>> grain = false(size(bw));
>> grain(cc.PixelIdxList{idx}) = true;
>> figure, imshow(1-grain);title('largest grain')
```

Lake.JPG

Level = 0.3608 (Otsu's method)

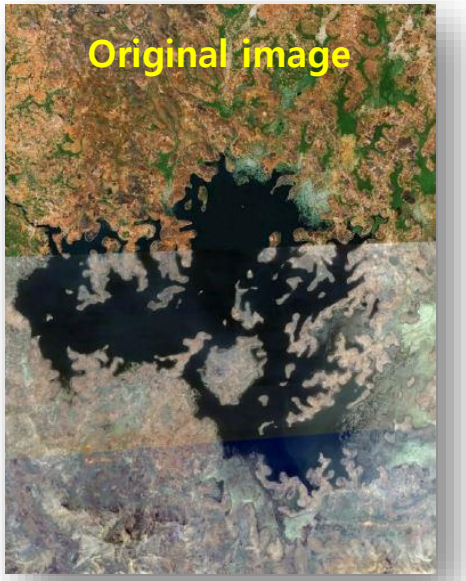


Max_area = 41190

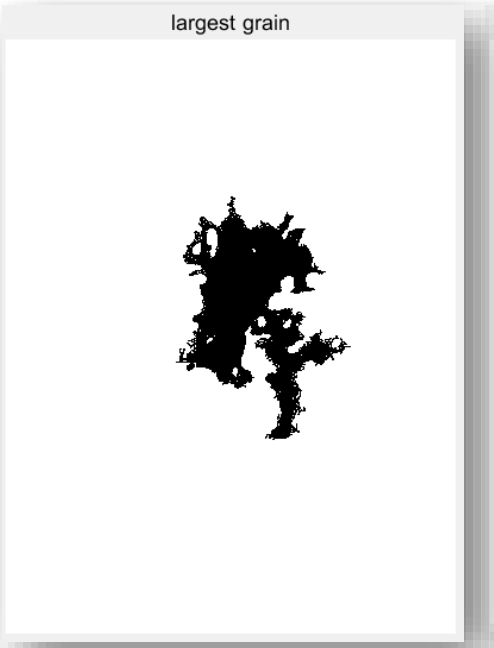


# Segmentation #1

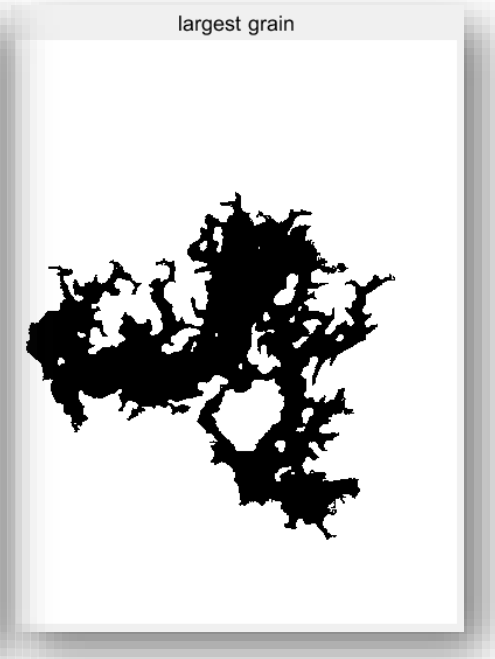
- Measuring the size of a lake
  - Different size depending different value of threshold



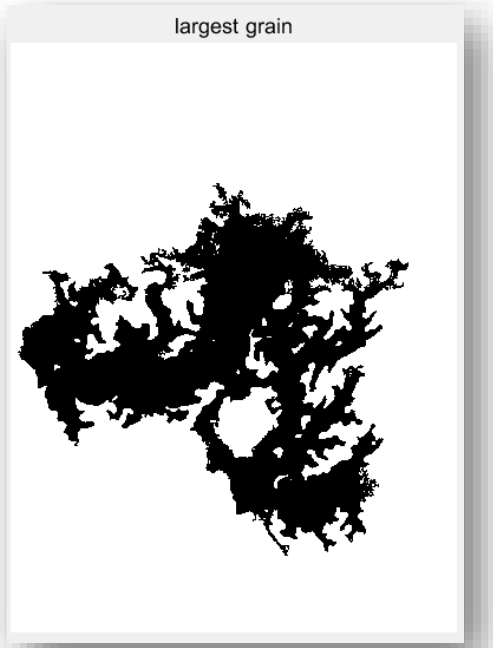

Level = 0.1
Max_area = 11921




Level = 0.2
Max_area = 33118

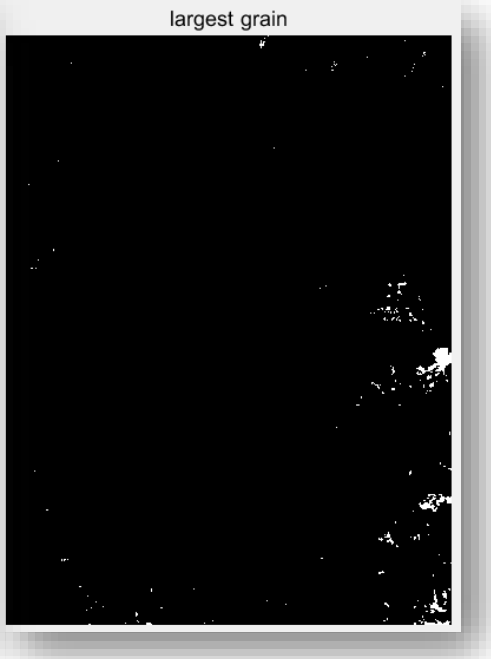



Level = 0.3608 (threshold fn.)
Max_area = 41190




Level = 0.6
Max_area = 156086


Level = 0.8
Max_area = 187979

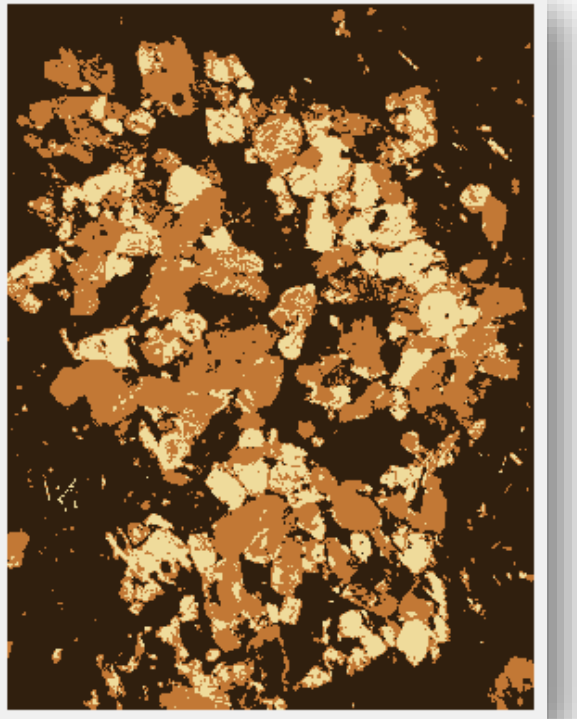


# Segmentation #1

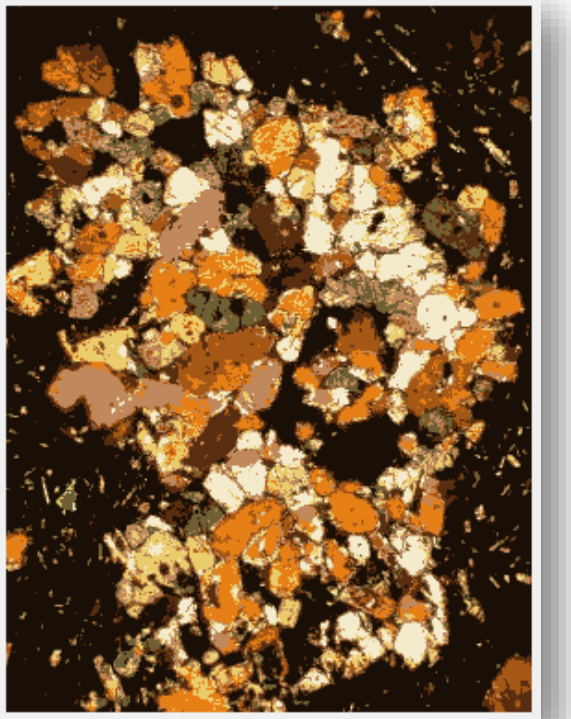
- Classifying minerals in thin section

```
>> clear all; close all
>> I=imread('thin.jpg');
>> nColors=3;
>> [ind map]=rgb2ind(I, nColors, 'nodither');
>> figure, imshow(ind, map)
```

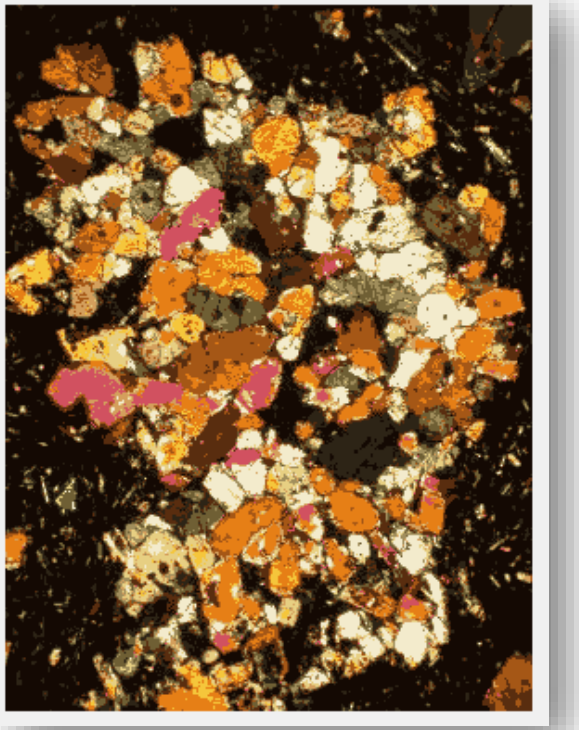

nColors=3



nColors=8



nColors=12



# Segmentation #1

- Classifying minerals in thin section
- Selecting N colors by N mouse clinks (hand)

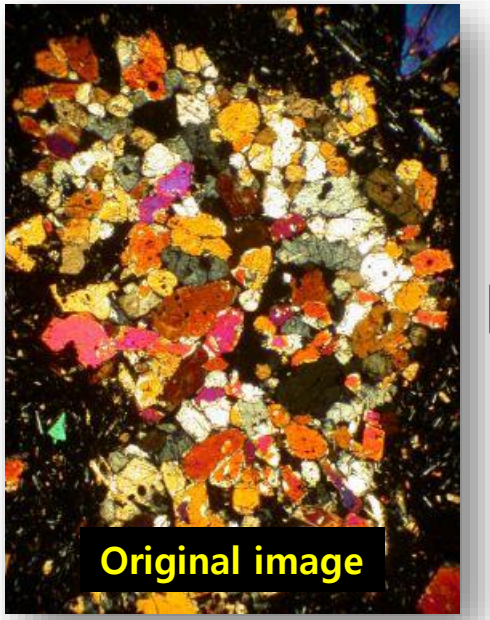
```
>> figure, imshow(I)
>> colors=impixel; save colors;
>> nColors=size(colors,1);
>> [ind map] = rgb2ind(I, colors/256, 'nodither');
>> figure, imshow(ind, map);
```

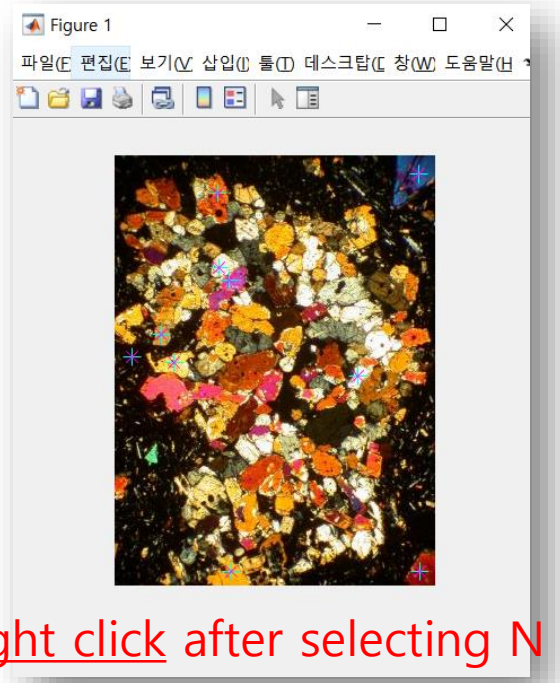
변수 - colors

cColors | colors

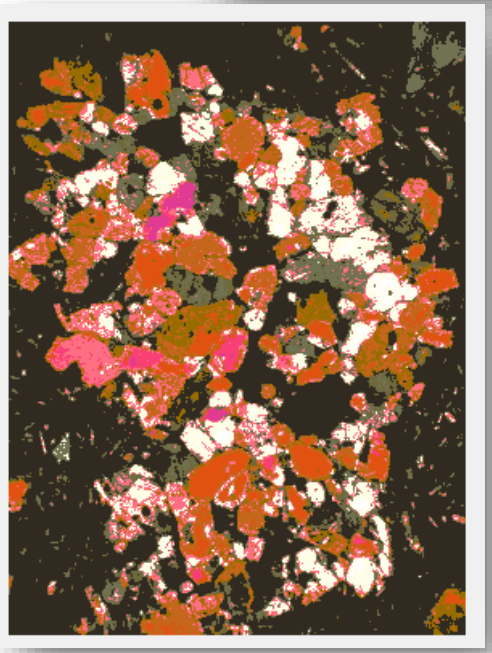
11x3 double

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 50 | 43 | 33 | |
| 2 | 255 | 112 | 132 | |
| 3 | 249 | 65 | 119 | |
| 4 | 153 | 110 | 0 | |
| 5 | 197 | 103 | 29 | |
| 6 | 226 | 59 | 147 | |
| 7 | 226 | 83 | 17 | |
| 8 | 255 | 255 | 232 | |
| 9 | 112 | 105 | 61 | |
| 10 | 255 | 255 | 253 | |
| 11 | 114 | 115 | 83 | |
| 12 | | | | |




Right click after selecting N colors



# Segmentation #2 (Filtering)

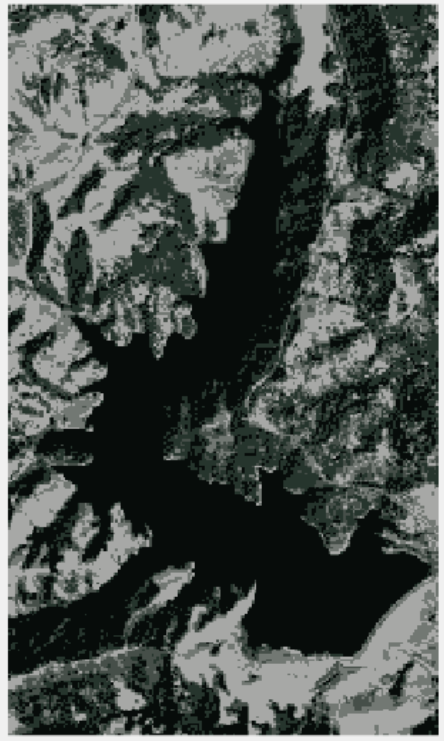
- Filtering (basic threshold)

```
>> clear all; close all;
>> I=imread('lake2.jpg');
>> figure, imshow(I)
```

```
>> [ind map]=rgb2ind(I, 5, 'nodither');
>> figure, imshow(ind, map)
```

```
>> level = 4/256;
>> bw = im2bw(I,level);
>> figure, imshow(bw)
```

```
>> bw=1-bw;
>> cc=bwconncomp(bw,4);
>> graindata = regionprops(cc,'basic');
>> grain_areas = [graindata.Area];
>> [max_area, idx]=max(grain_areas)
>> grain=false(size(bw));
>> grain(cc.PixelIdxList{idx}) = true;
>> figure, imshow(1-grain)
```

# Segmentation #2

- Filtering (taking average)

변수 - h

+1 | colors | lr | l | ind | map | bw | h

5x5 double

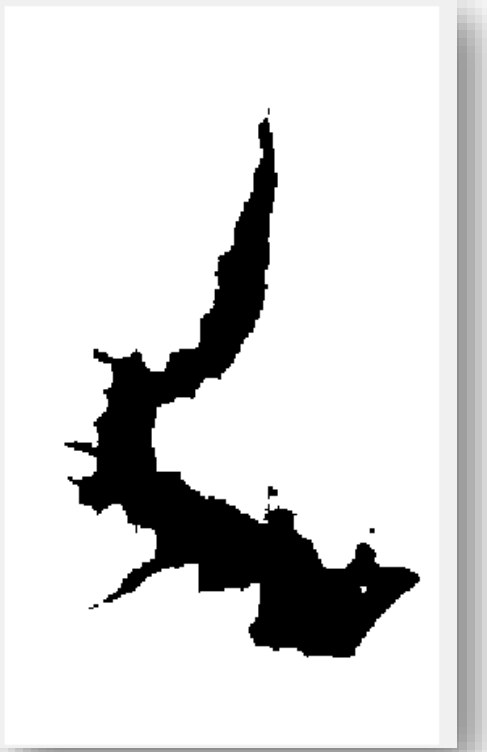
| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | |
| 2 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | |
| 3 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | |
| 4 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | |
| 5 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | 0.0400 | |
| 6 | | | | | | |

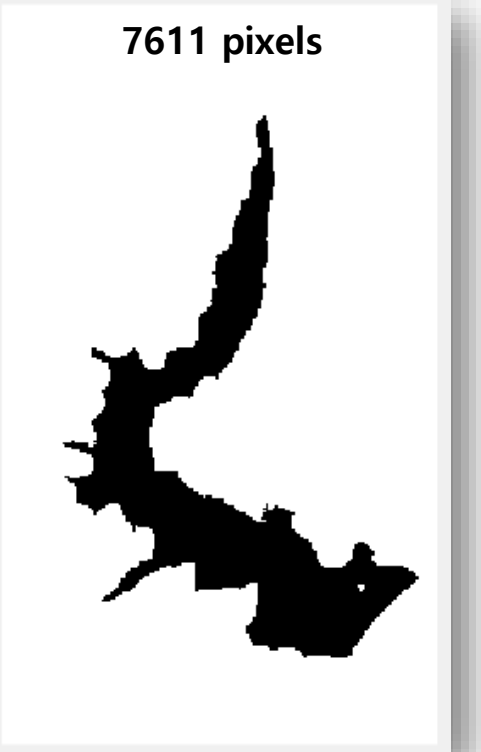
```
>> clear all; close all;
>> I=imread('lake2.jpg');
>> figure, imshow(I)
```

```
>> n=5;
>> h=ones(n,n)/(n*n);
>> I=imfilter(I,h);
>> figure, imshow(I)
```

```
>> level = 4/256;
>> bw = im2bw(I,level);
>> figure, imshow(bw)
```

```
>> bw=1-bw;
>> cc=bwconncomp(bw,4);
>> graindata = regionprops(cc,'basic');
>> grain_areas = [graindata.Area];
>> [max_area, idx]=max(grain_areas)
>> grain=false(size(bw));
>> grain(cc.PixelIdxList{idx}) = true;
>> figure, imshow(1-grain)
```



# Segmentation #2 (Filtering)

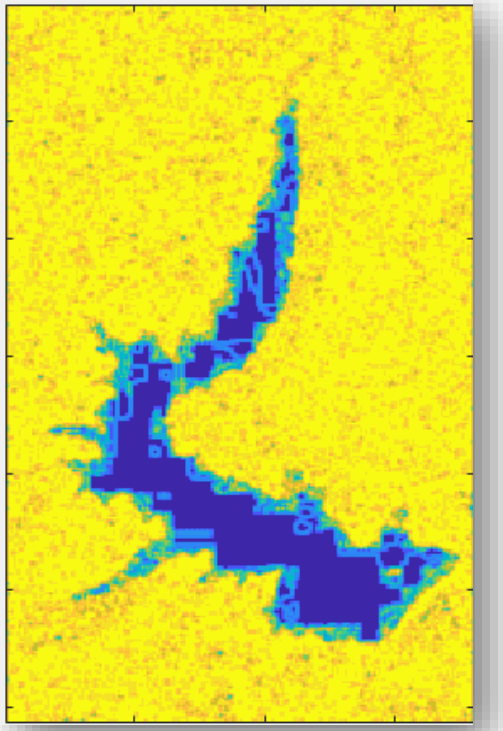
- Filtering (Entropy filtering)

```
>> clear all; close all;
>> I=imread('lake2.jpg');
>> figure, imshow(I)
```

```
>> fSize=3; ngood=true(fSize);
>> G=rgb2gray(I);
>> E=entropyfilt(G,ngood);
>> figure, imagesc(E)
```

```
>> L=E<0.8;
>> imshow(L)
>> cc=bwconncomp(L,4);
>> graindata=regionprops(cc,'basic')
>> grain_areas=[graindata.Area];
>> max(grain_areas)
>> imshow(1-L)
```

```
>> L=E<1;
>> imshow(L)
>> cc=bwconncomp(L,4);
>> graindata=regionprops(cc,'basic');
>> grain_areas=[graindata.Area];
>> max(grain_areas)
>> imshow(1-L)
```

3474 pixels

4628 pixels

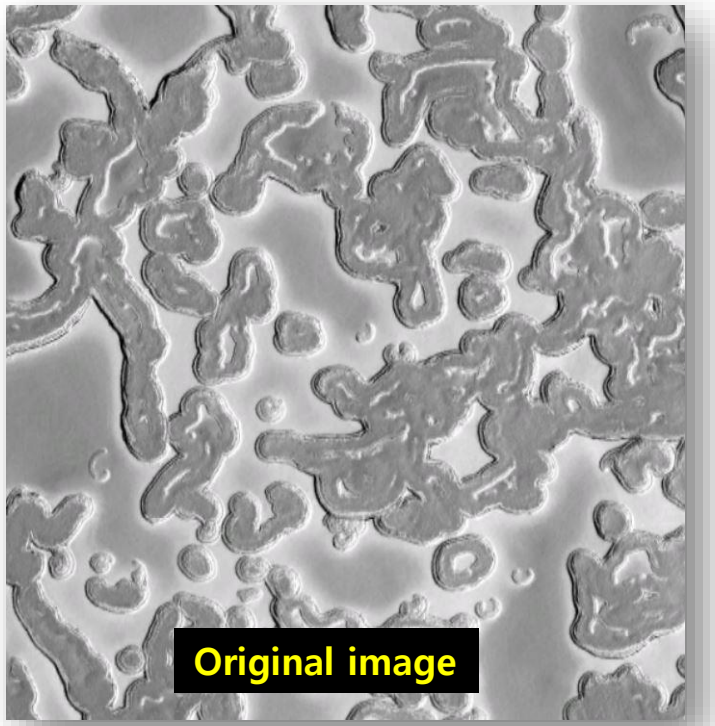
# Segmentation #2 (Filtering)

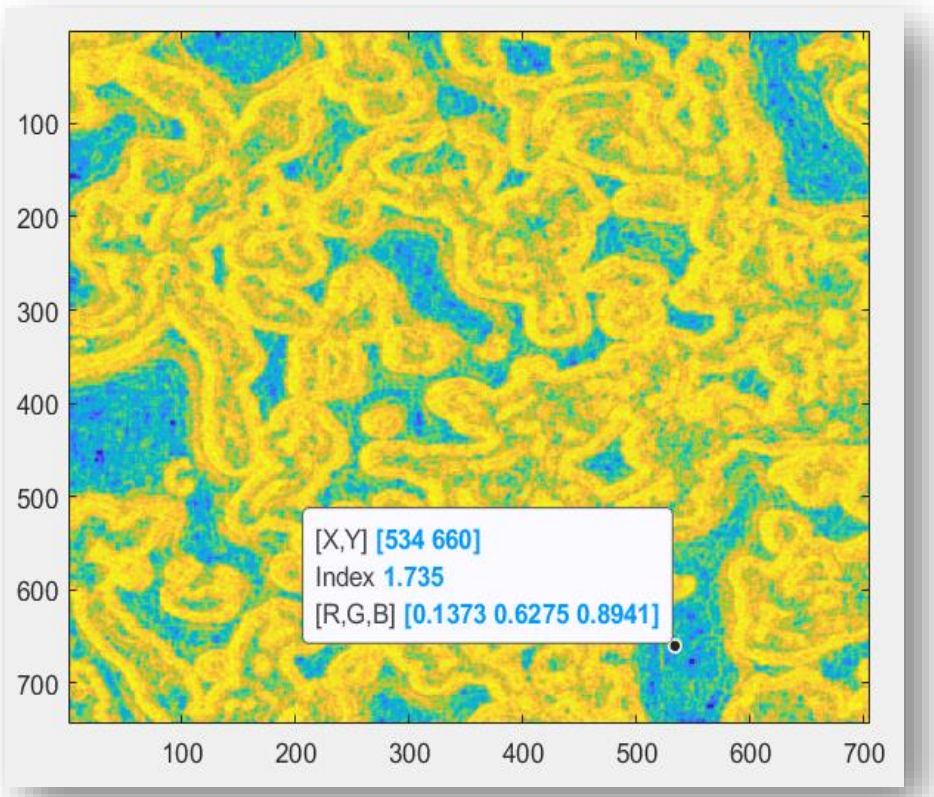
- Filtering (Entropy filtering)

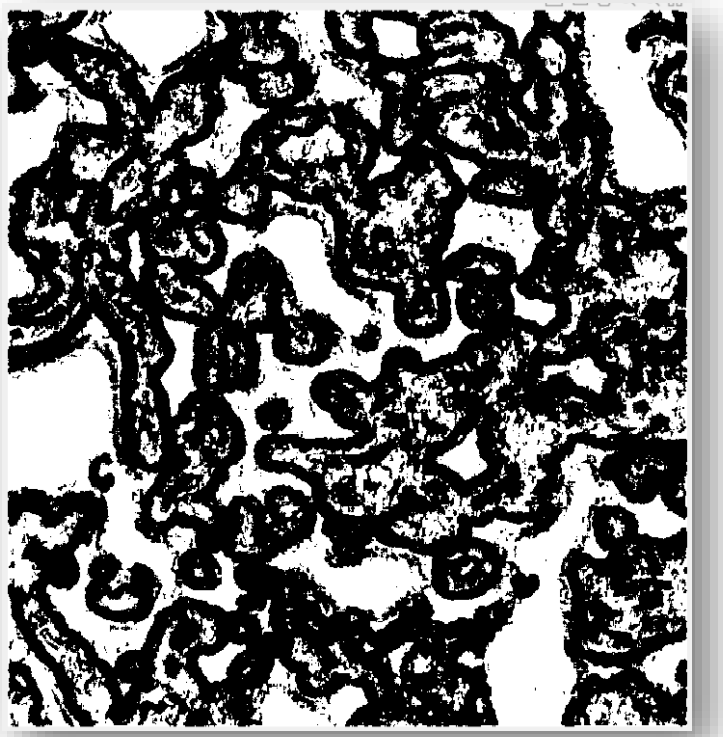
```
>> clear all, close all
>> I=imread('swiss.jpg');
>> G=rgb2gray(I);
```

```
>> fSize=5, nhood=true(fSize);
>> E=entropyfilt(G,nhood);
>> figure, imagesc(E)
```

```
>> L=E<3.5;
>> figure, imshow(L)
```



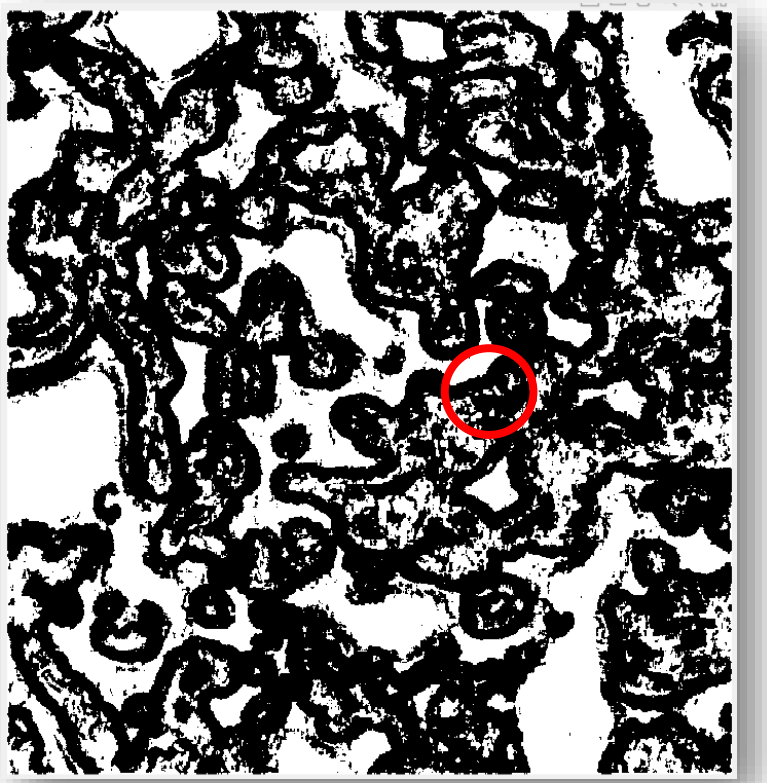
# Segmentation #2 (Filtering)

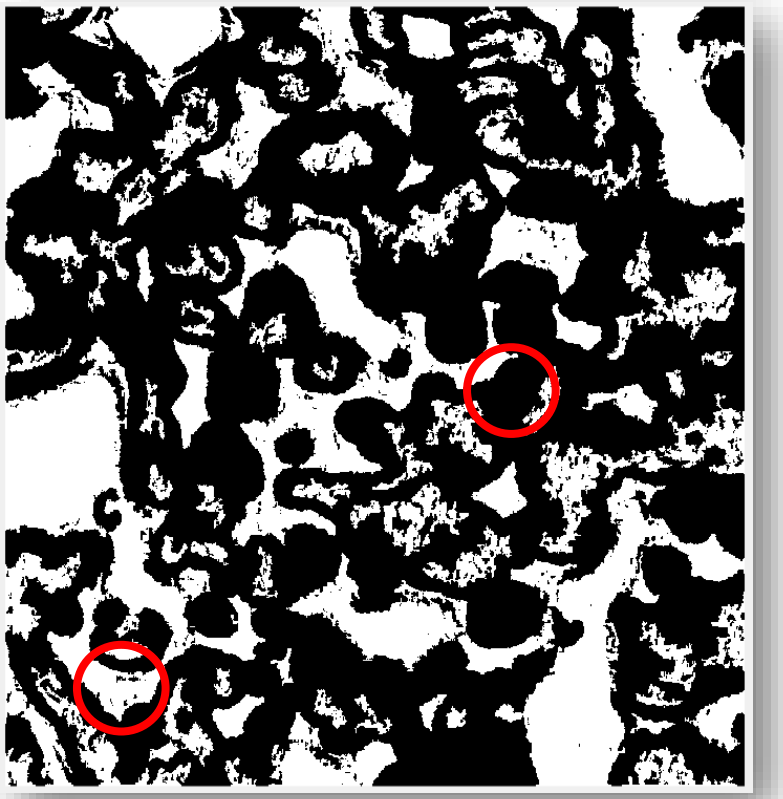
- Entropy Filter + Gray threshold + Closing (removing)

```
>> L=E<3.5;
>> figure, imshow(L)
```

```
>> L2=bwareaopen(L, 200);
>> figure, imshow(L2)
```

```
>> L3=bwareaopen(1-L2, 500);
>> figure, imshow(1-L3)
```



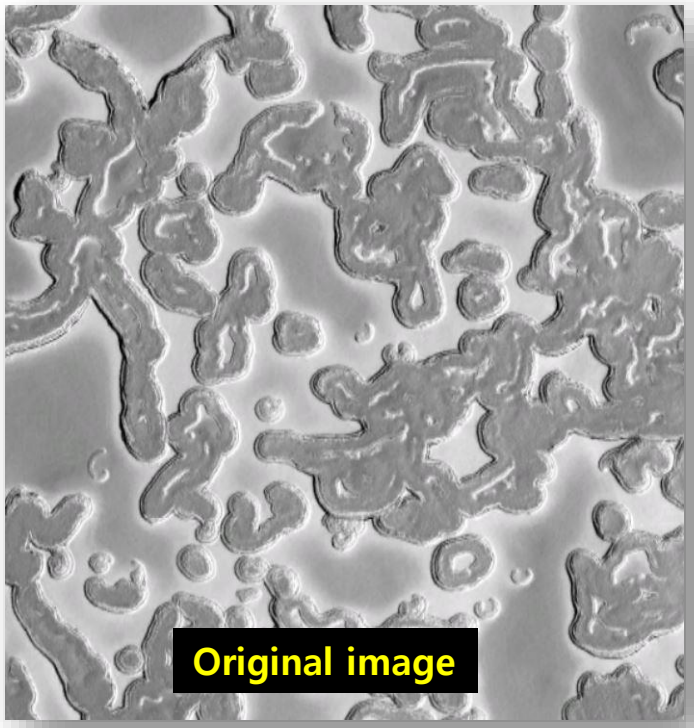


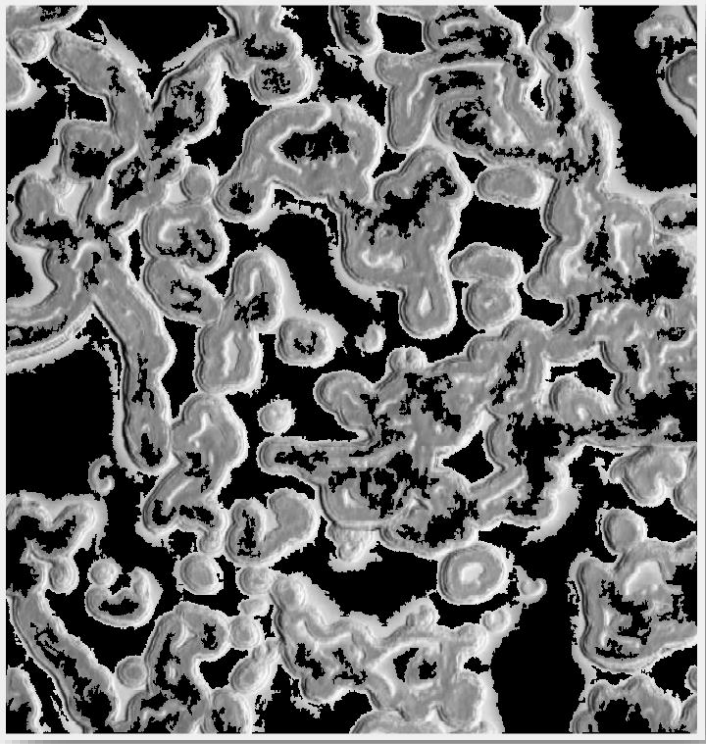
# Segmentation #2 (Filtering)

- Entropy Filter + Gray threshold + Closing (removing)

```
>> L3=bwareaopen(1-L2, 500);
>> figure, imshow(1-L3)
```

```
>> G2=G;
>> G2(L3==0)=0;
>> figure, imshow(G2)
```



# Segmentation #2 (Filtering)

- Entropy, Standard Deviation Filter, Gradients

```
>> clear all; close all;
>> I=imread('lake2.jpg');
>> figure, imshow(I)
```

```
>> fSize = 3; nHood = true(fSize);
>> G=rgb2gray(I);
>> E=entropyfilt(G,nHood);
>> figure, imagesc(E), colorbar
```

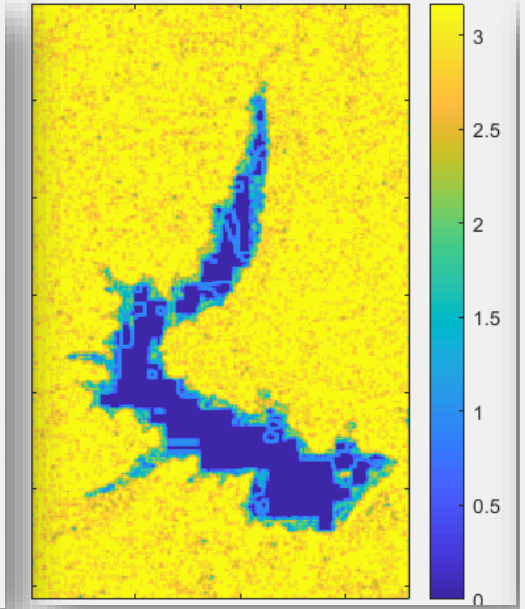
```
>> E1=rangefilt(G,nHood);
>> figure, imagesc(E1), colorbar
```

```
>> E2=stdfilt(G,nHood);
>> figure, imagesc(E2), colorbar
```

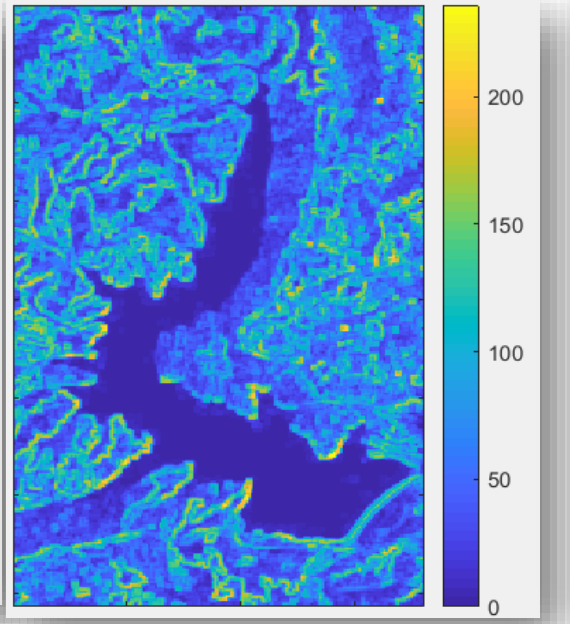
```
>> [Gx Gy] = gradient(double(G));
>> Gr = sqrt(Gx.^2 + Gy.^2);
>> figure; imshow(Gr);
>> figure; imagesc(Gr), colorbar
```

**Original**

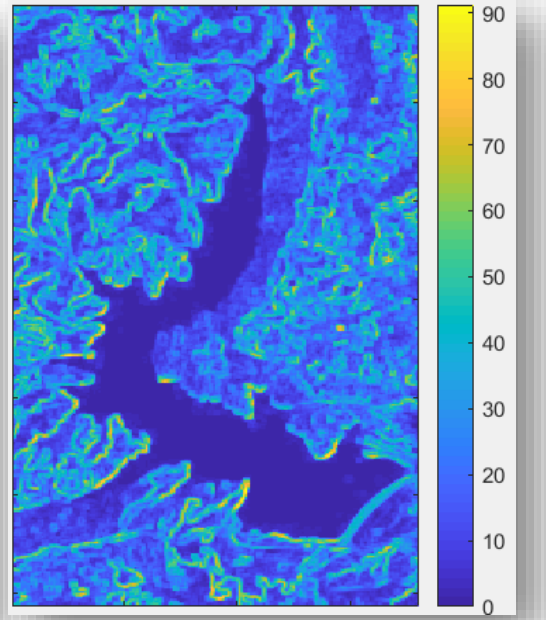
**Entropy**



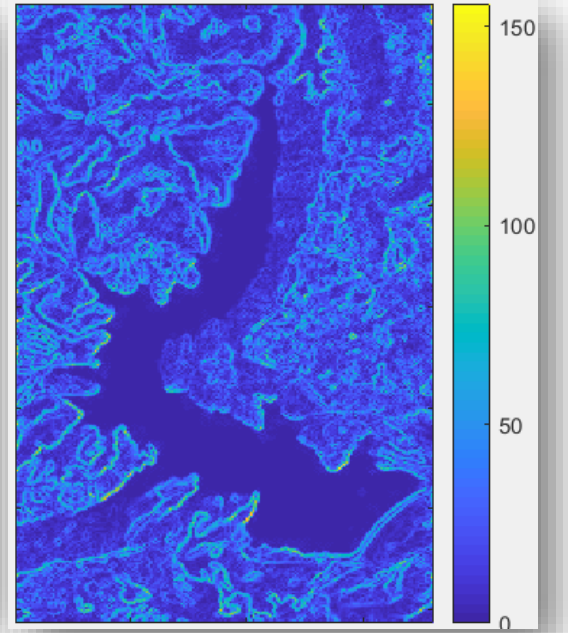

**Range Filter**




**Standard Deviation Filter**




**Gradients**



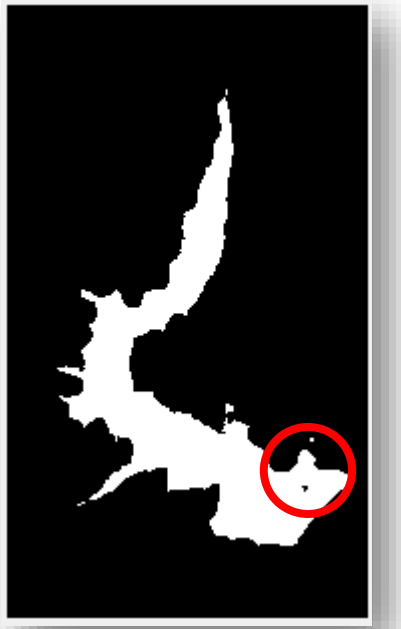
# Segmentation #3 (Morphological operations)

- Matlab likes to operate on objects
- Example of Closing

```
>> clear all; close all;
>> I=imread('lake2.jpg');
```


```
n=5; h=ones(n,n)/(n*n);
I=imfilter(I,h);
figure, imshow(I)
level=4/256;
bw=im2bw(I,level);
figure, imshow(bw)
```

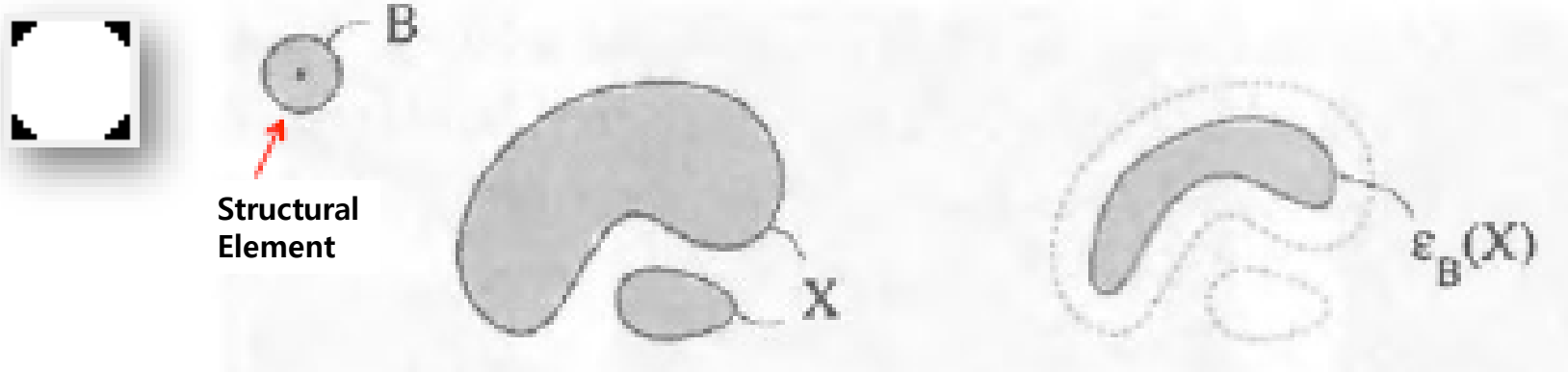
```
>> bw=1-bw;
>> figure, imshow(bw)
```



```
>> SE=strel('disk',10);
>> L1=imclose(bw, SE);
>> figure, imshow(L1)
```

# Segmentation #3 (Morphological operations)

- Morphological operations (**Image Erosion**, Image Dilation, Image Opening, Image Closing)



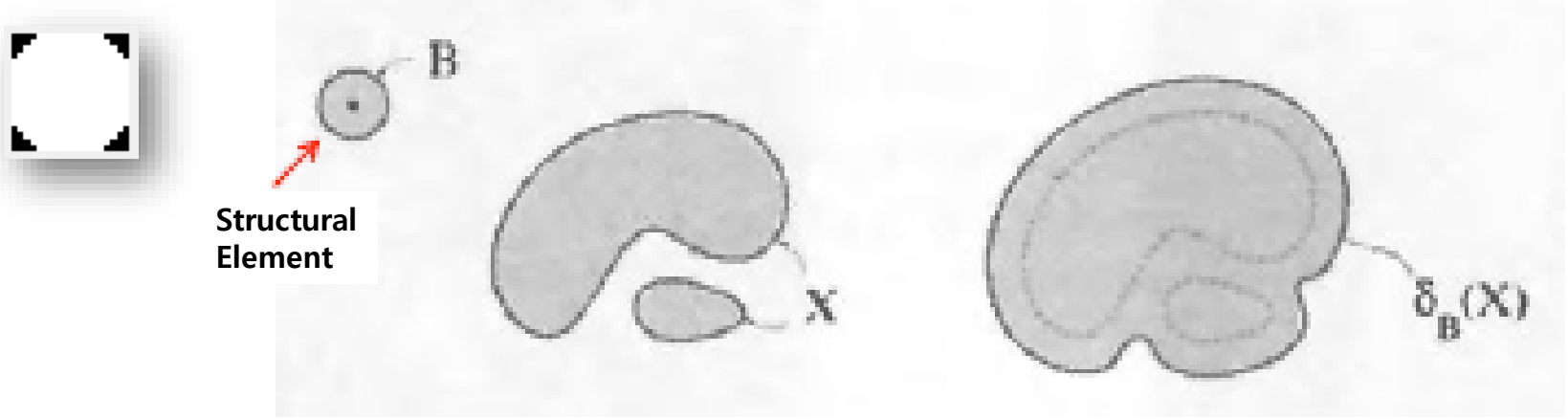

- Does B "fit" in the set X?
- Retain all points (i,j) in X such that when B is centered at (i,j), B is contained in X
- Mathematically:

$$\varepsilon_B(X) = \left\{ X \middle| B_X \subseteq X \right\}$$

- **In Matlab: imerode(Img,Elt)**

# Segmentation #3 (Morphological operations)

- Morphological operations (Image Erosion, **Image Dilation**, Image Opening, Image Closing)




- Does B "touch" in the set X?
- Add to X all points (i,j) in B such that when B is centered at (i,j), B is overlapping with X
- Mathematically:

$$\delta_B(X) = \bigcup_{b \in B} X\_b$$

- **In Matlab: imdilate(Img,Elt)**

# Segmentation #3 (Morphological operations)

- Morphological operations (Image Erosion, Image Dilation, **Image Opening**, Image Closing)

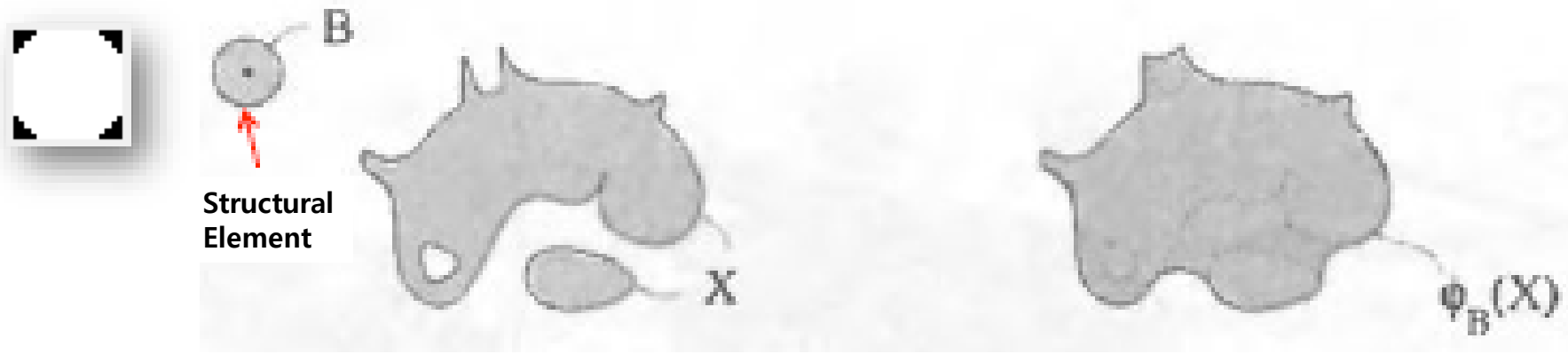

- Does B "fit" in the set X?
- Like with erosion, retain all points (i,j) in X such that when B is centered at (i,j), B is contained in X
- When the previous is true, also keep all of B
- Mathematically:

$$\gamma_B(X) = \bigcup_X \left\{ B_X \,\middle|\, B_X \subseteq X \right\}$$

- **In Matlab: imopen (Img,Elt)**

# Segmentation #3 (Morphological operations)

- Morphological operations (Image Erosion, Image Dilation, Image Opening, **Image Closing**)

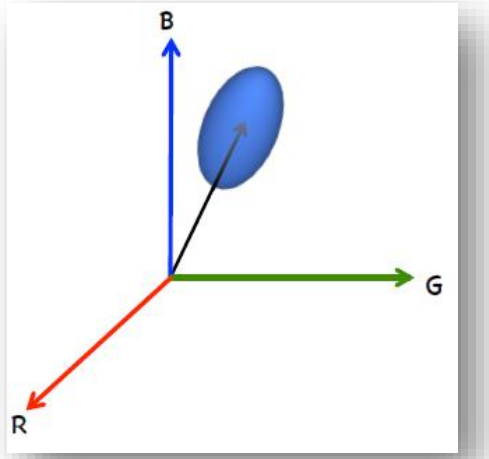



- Does B "fit" in the backround of the set X?
- When the previous is true, all of B belongs to the background
- The complement of the new background define X
- Mathematically:

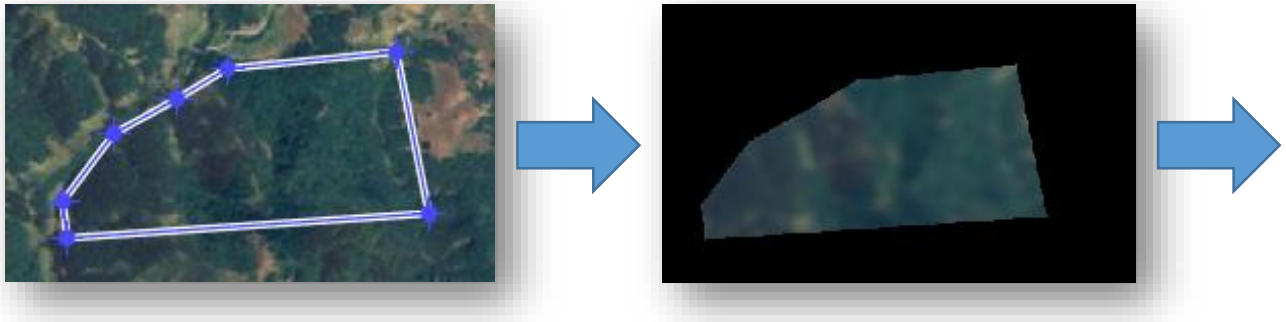
$$\phi_B(X) = \left[ \bigcup_X \left\{ B_X \,\middle|\, B_X \subseteq X^c \right\} \right]^c$$

- **In Matlab: imclose(Img,Elt)**

# Segmentation #4 (Mean & Covariance)




- Compute the mean and the covariance matrix
- Then similarity implies being with a weighted distance
- The pixels that are within an ellipsoid centered at the mean color are similar



myMean

1x3 double

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 43.7137 | 69.1717 | 74.3852 |

Mean

$$+\begin{pmatrix} 1135.9 & 1072.1 & 380.2 \\ 1072.1 & 1031.7 & 364.7 \\ 380.2 & 364.7 & 335.6 \end{pmatrix}$$

Covariance Matrix

Covariance: $$\mathrm{cov}(A,B) = \frac{1}{N-1}\sum_{i=1}^{N}(A_i-\mu_A)^*(B_i-\mu_B)$$

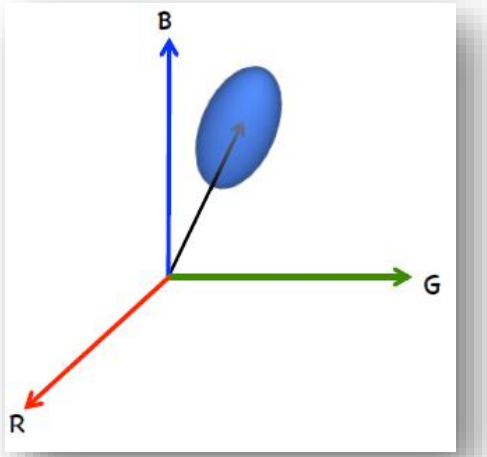
Covariance Matrix: $$C = \begin{pmatrix} \mathrm{cov}(A,A) & \mathrm{cov}(A,B) \\ \mathrm{cov}(B,A) & \mathrm{cov}(B,B) \end{pmatrix}$$

Variance: $$V = \frac{1}{N-1}\sum_{i=1}^{N}|A_i-\mu|^2$$

Average: $$\mu = \frac{1}{N}\sum_{i=1}^{N}A_i.$$

# Segmentation #4 (Mean & Covariance)

- Mahalanobis distance




## Mahalanobis Distance

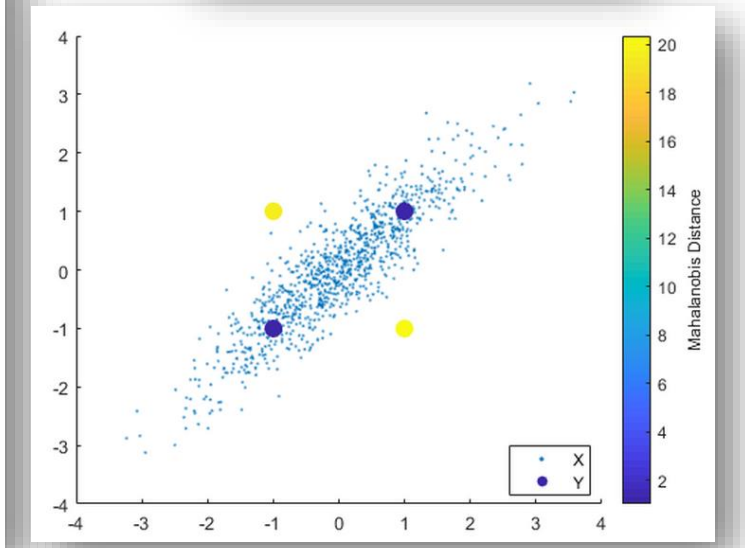
The Mahalanobis distance is a measure between a sample point and a distribution.

The Mahalanobis distance from a vector $y$ to a distribution with mean $\mu$ and covariance $\Sigma$ is

$$d = \sqrt{(y-\mu)\sum^{-1}(y-\mu)'}.$$

This distance represents how far $y$ is from the mean in number of standard deviations.

`mahal` returns the squared Mahalanobis distance $d^2$ from an observation in Y to the reference samples in X. In the `mahal` function, $\mu$ and $\Sigma$ are the sample mean and covariance of the reference samples, respectively.




(From Matlab Help)

# Segmentation #4 (Mean & Covariance)

- Compute Forest area in the Photo
- Input & Smoothening

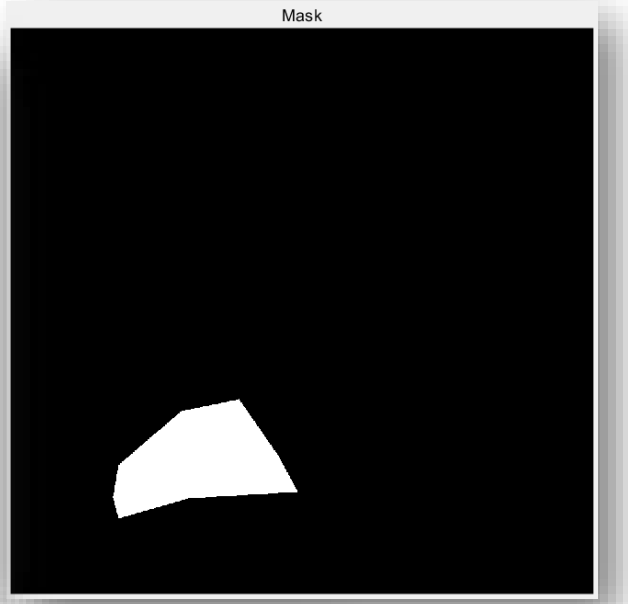
```
clear all; close all;
imName = 'naip1.jpg';
rgbImg = imread(imName);
[nRows nCols nLyrs] = size(rgbImg);
nPix = nRows * nCols;
figure; imshow(rgbImg); title(['RGB image ' imName]);
```

```
%Smooth
filtSize = 10; %set filter size
myFilt = ones(filtSize) ./ filtSize^2;
smoothImg = imfilter(rgbImg, myFilt);
figure; imshow(smoothImg); title('smoothed RGB image');
```

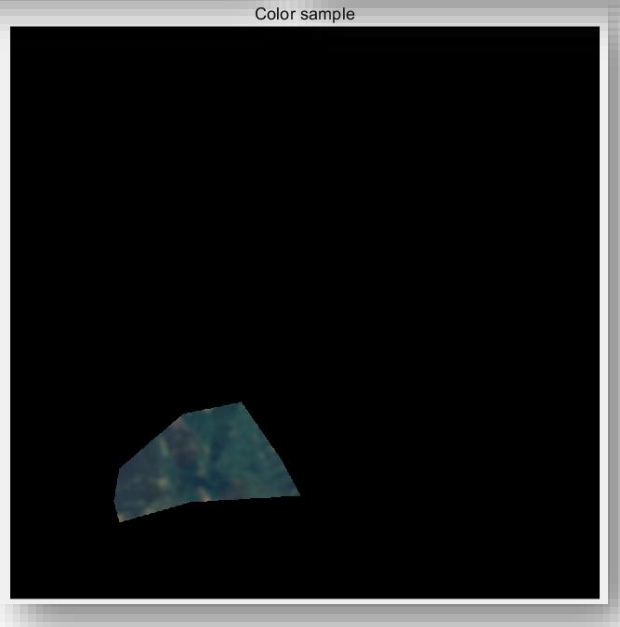

# Segmentation #4 (Mean & Covariance)

- Use roipoly() for selecting Region of Interest (ROI)

```
figure;
roiMask = roipoly(rgbImg);
%right clink on the polygon and select the option "Create Mask"
figure; imshow(roiMask); title('Mask');
```



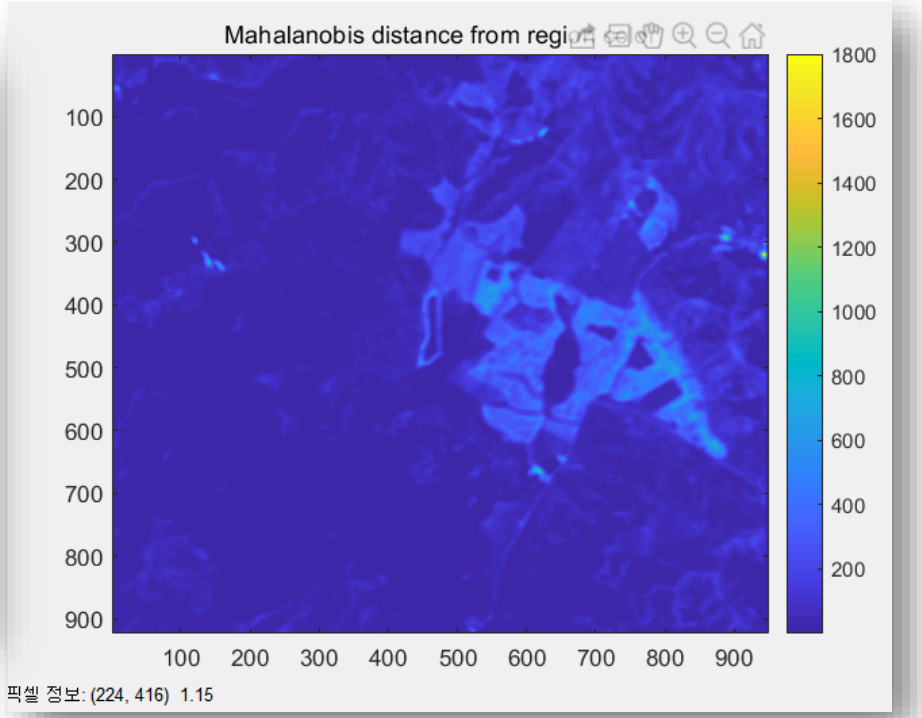

```
red=immultiply(roiMask, smoothImg(:,:,1));
green=immultiply(roiMask, smoothImg(:,:,2));
blue=immultiply(roiMask, smoothImg(:,:,3));
roiImg = cat(3, red, green, blue);
figure; imshow(roiImg); title('Color sample');
```



# Segmentation #4 (Mean & Covariance)

- Compute Mahalanobis distance for all pixels

```
%compute mean patch color and std
imgPix = double(reshape(smoothImg, nPix, 3));
roiPix = double(reshape(roiImg, nPix, 3));
roiIdx = find(roiMask);
roiPix = roiPix(roiIdx, 1:3);
myMean = mean(roiPix);
myStd = max(std(roiPix));

%compute Mahalanobis dist.
mahDist = mahal(imgPix, roiPix);
myDist = reshape(mahDist, nRows, nCols);
figure; imagesc(myDist);
title('Mahalanobis distance from region color');
colorbar;
impixelinfo
```

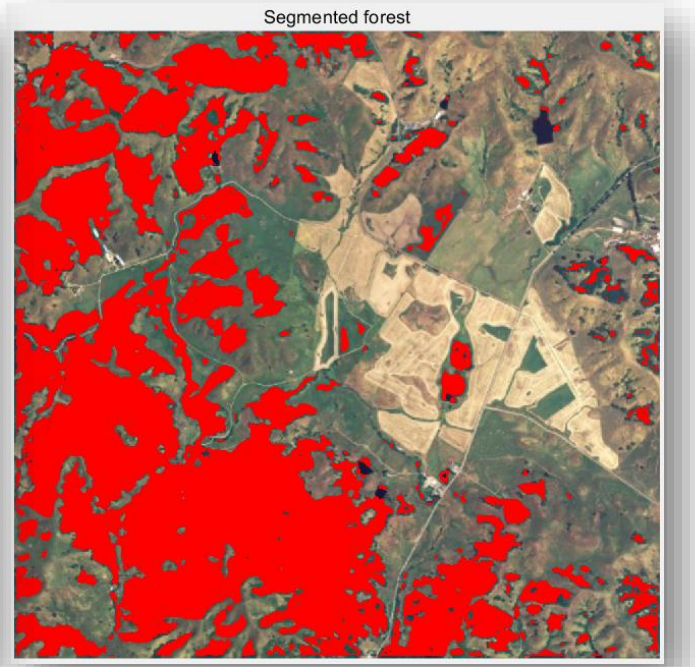



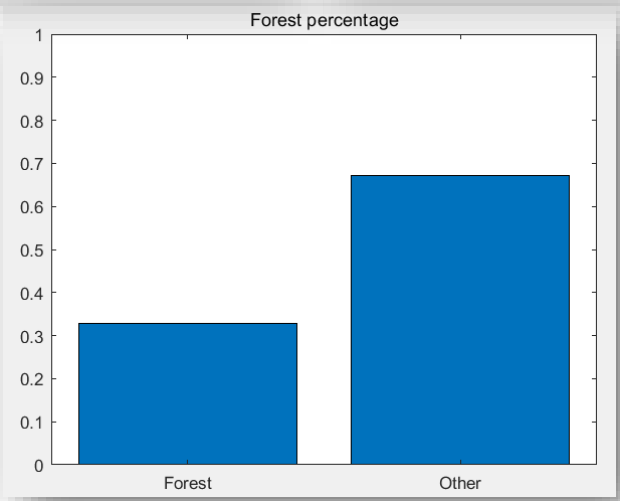
# Segmentation #4 (Mean & Covariance)

- Segmentation & Histogram

```
%segmentation
kThresh = 1;
mySegm = false(nRows, nCols);
mySegm(myDist <= kThresh * myStd) = true;

lr=rgbImg(:,:,1);
lg=rgbImg(:,:,2);
lb=rgbImg(:,:,3);

lr(mySegm==1) = 255;
lb(mySegm==1) = 0;
lg(mySegm==1) = 0;

l=cat(3,lr,lg,lb);
figure;imshow(l);title('Segmented forest')
```

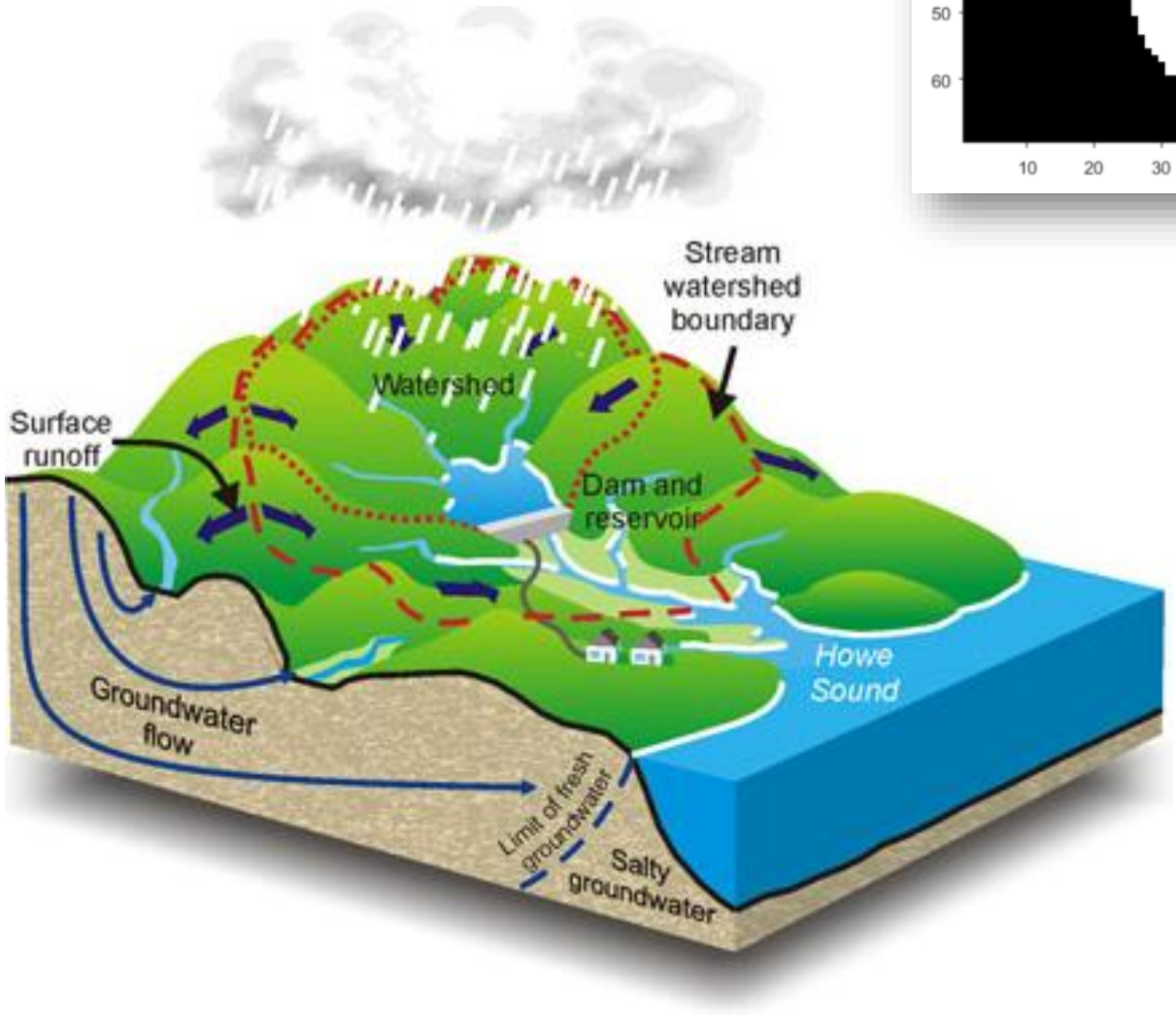
```
%histogram
myCount=sum(mySegm(:));
myDistr = [myCount nPix-myCount] ./ nPix;
figure; bar(myDistr);
axis([0.5 2.5 0 1]);
set(gca, 'XTickLabel', {'Forest', 'Other'});
title('Forest percentage');
```

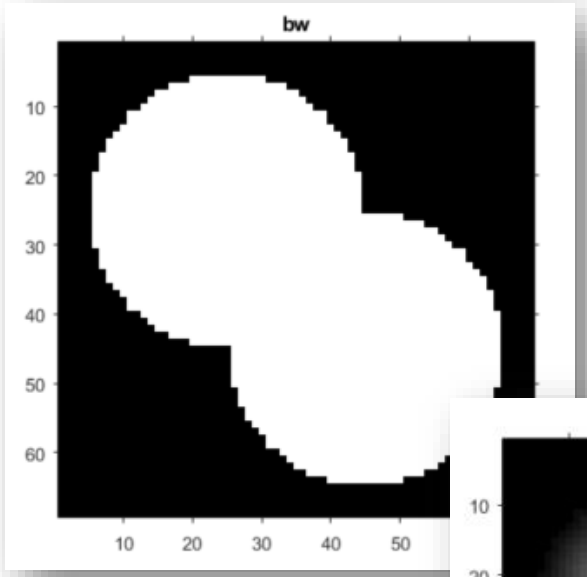



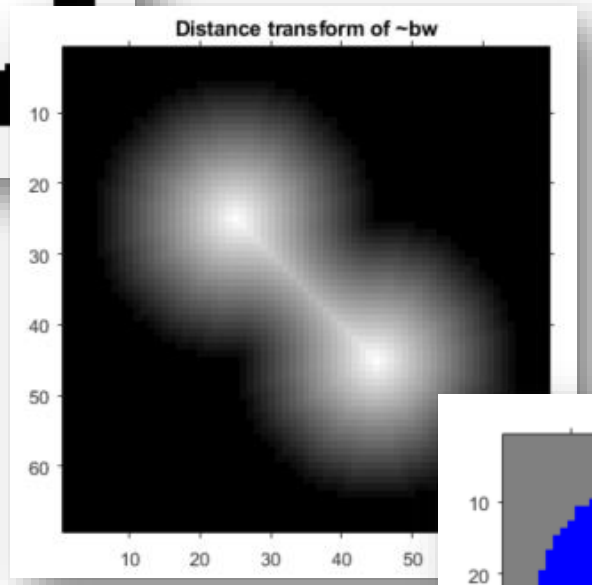



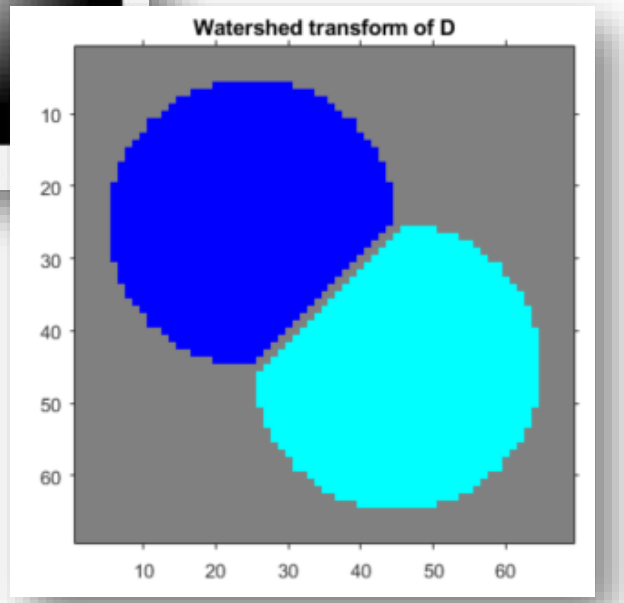
# Edge Detection #1

- Watershed Transformation



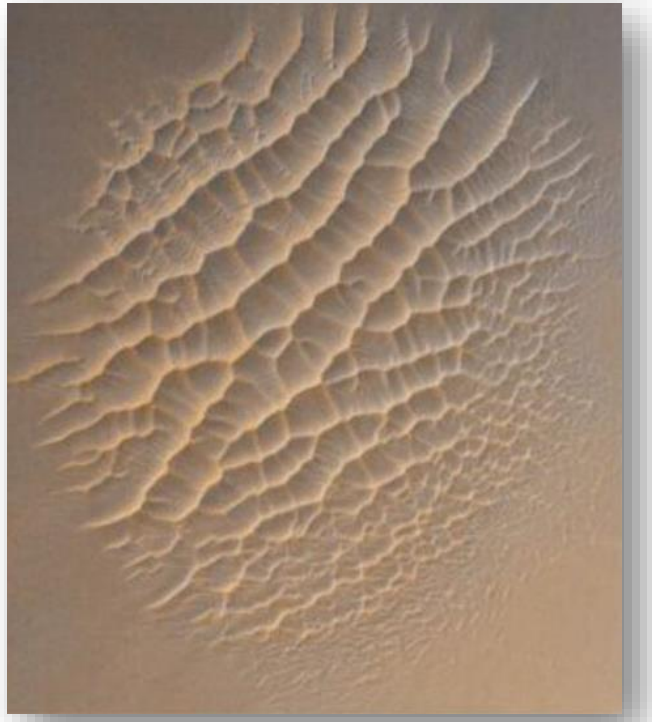



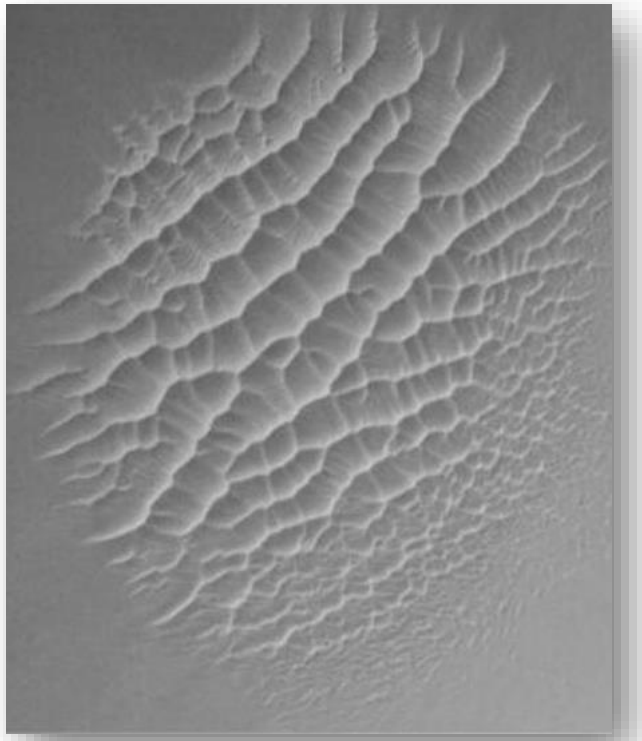
# Edge Detection #1

- Watershed Transformation

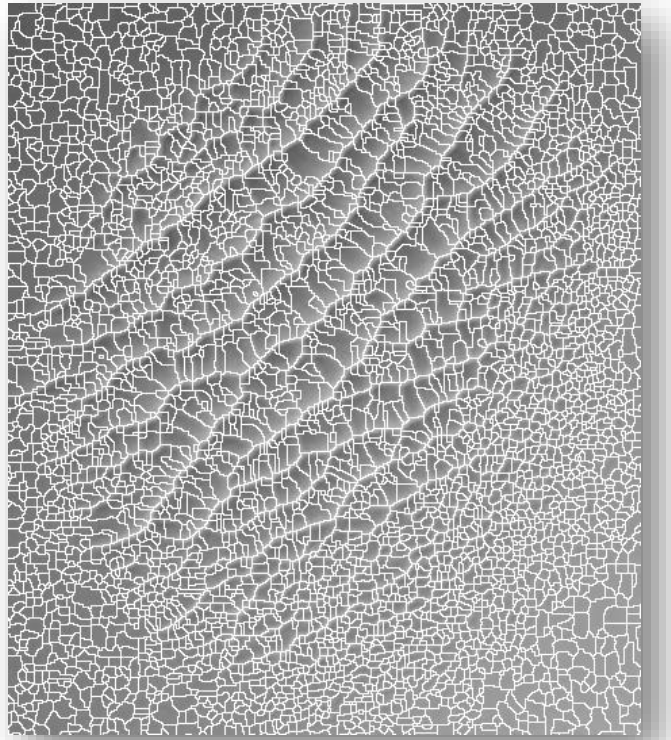
```
>> clear all; close all;
>> I=imread('dune.jpg');
>> figure, imshow(I)
```



```
>> G=rgb2gray(I);
>> figure; imshow(G)
```



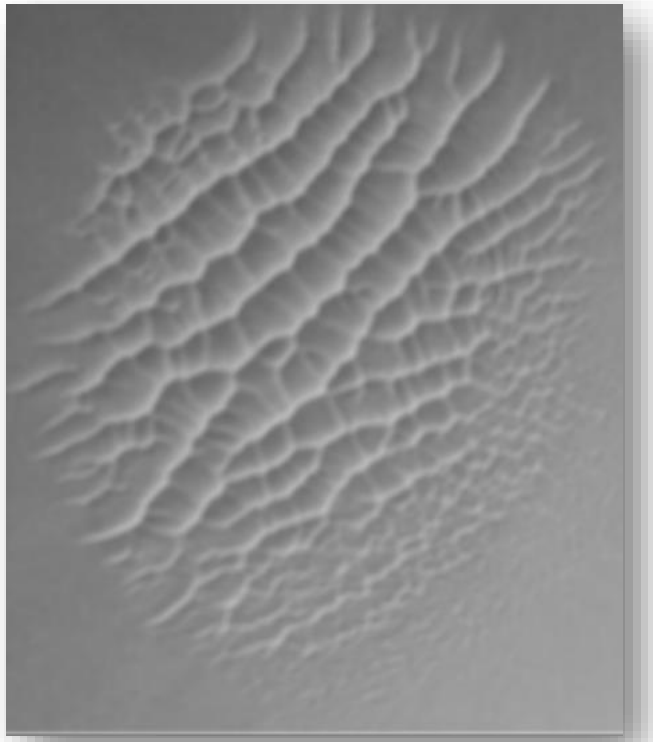
```
>> L=watershed(G);
>> G2=G;
>> G2(L==0)=255;
>> figure; imshow(G2)
```



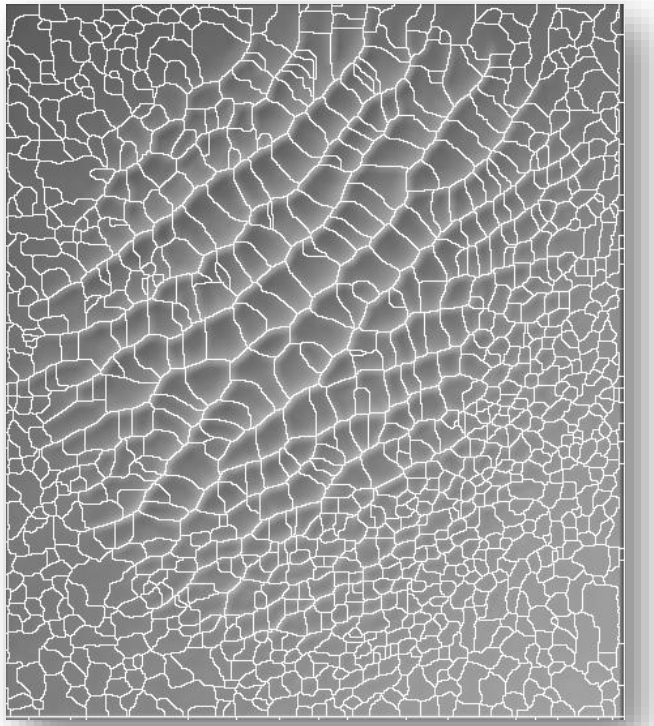
# Edge Detection #1

- **Smoothed** + Watershed Transformation

```
>> fSize = 6;
>> h=ones(fSize,fSize)/fSize^2;
Sm=imfilter(G,h);
figure;imshow(Sm)
```



```
>> L=watershed(Sm);
>> L0=(L==0);
>> G2=Sm;
>> G2(L0)=255;
>> figure, imshow(G2);
```

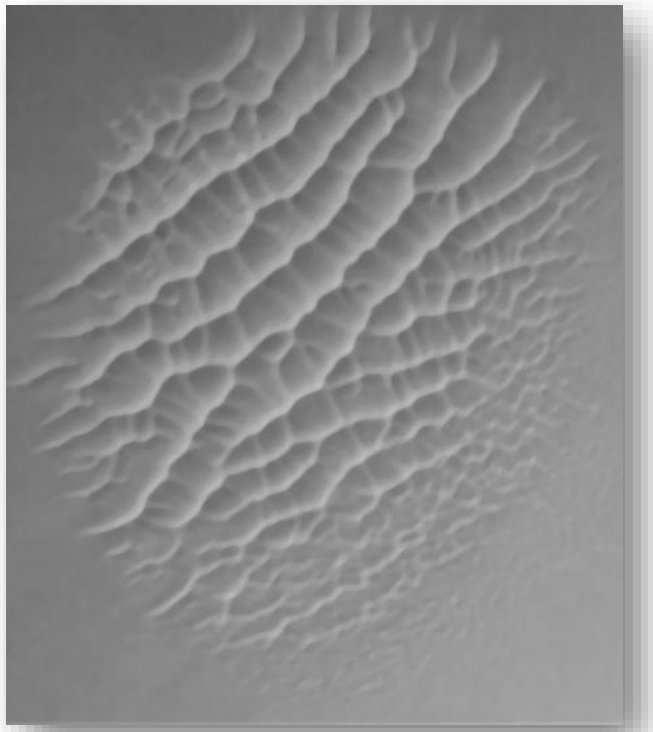


```
>> I2=I(:,:,2);I2(L0)=0;
>> I1=I(:,:,1);I1(L0)=0;
>> I2=I(:,:,2);I2(L0)=0;
>> I3=I(:,:,3);I3(L0)=0;
>> II=cat(3,I1,I2,I3);
>> figure, imshow(II)
```

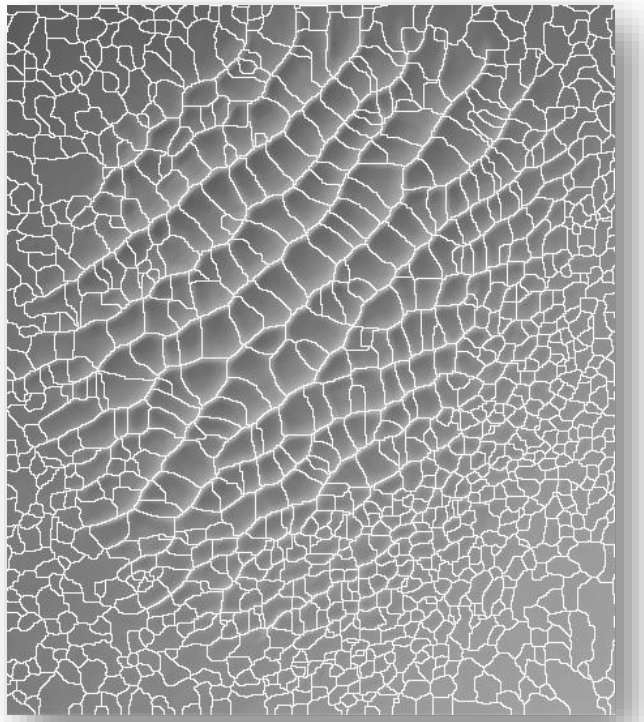
# Edge Detection #1

- **Median Filter** + Watershed Transformation

```
>> fSize = 6;
>> Sm=medfilt2(G, [fSize fSize]);
>> figure; imshow(Sm)
```



```
>> L=watershed(Sm);
>> L0 = (L==0);
>> G2 = Sm;
>> G2(L0) = 255;
>> figure; imshow(G2)
```

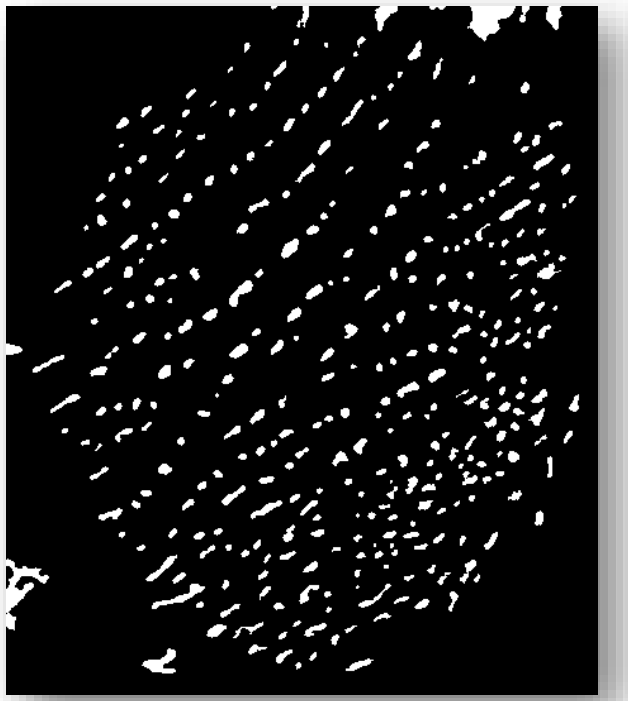


```
>> I1=I(:,:,1);I1(L0)=0;
>> I2=I(:,:,2);I2(L0)=0;
>> I3=I(:,:,3);I3(L0)=0;
>> II=cat(3,I1,I2,I3);
>> figure; imshow(II)
```

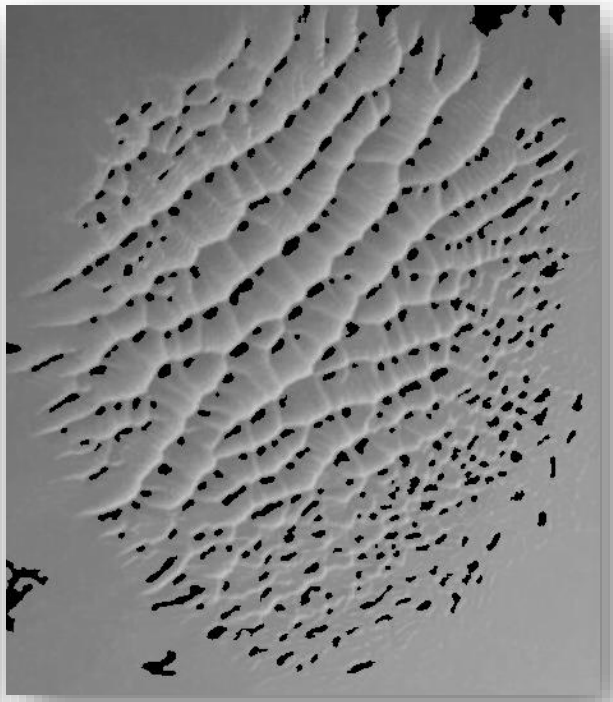
# Edge Detection #1

- **Regional Extended Minima** + Watershed Transformation

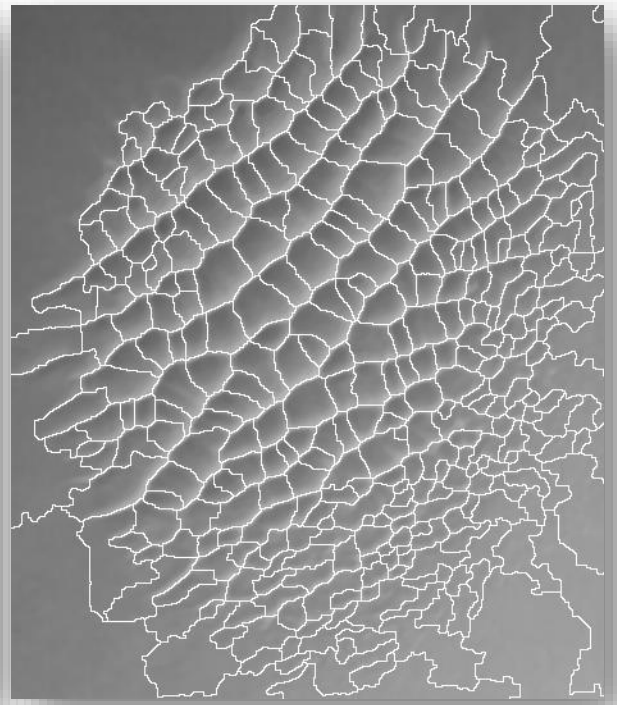
```
>> thresh=3;
>> Imin=imextendedmin(Sm, thresh);
>> figure; imshow(Imin)
```



```
>> G2=imimposemin(G, Imin);
>> figure; imshow(G2)
```



```
>> L2 = watershed(G2);
>> L0 = (L2==0);
>> G3=Sm;
>> G3(L0)=255;
>> figure; imshow(G3)
```



```
>> I1=I(:,:,1);I1(L0)=0;
>> I2=I(:,:,2);I2(L0)=0;
>> I3=I(:,:,3);I3(L0)=0;
>> II=cat(3,I1,I2,I3);
>> figure; imshow(II)
```

# Edge Detection #2

1. Canny -> Dilation -> Skeletonization
2. Bottom-hat -> Distance -> Watershed
3. gPb -> Distance -> Watershed
4. gPb -> OWT -> UCM

*Mathematical Geology, Vol. 37, No. 1, January 2005 (© 2005)*
DOI: 10.1007/s11004-005-8745-x

**Automated Sizing of Coarse-Grained Sediments: Image-Processing Procedures[1]**

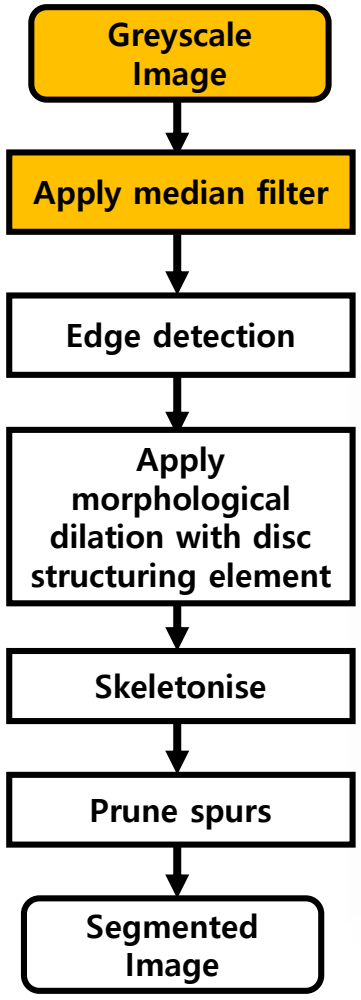
**David J. Graham,[2] Ian Reid,[2] and Stephen P. Rice[2]**

*This is the first in a pair of papers in which we present image-processing-based procedures for the measurement of fluvial gravels. The spatial and temporal resolution of surface grain-size characterization is constrained by the time-consuming and costly nature of traditional measurement techniques. Several groups have developed image-processing-based procedures, but none have demonstrated the transferability of these techniques between sites with different lithological, clast form and textural characteristics. Here we focus on image-processing procedures for identifying and measuring image objects (i.e. grains); the second paper examines the application of such procedures to the measurement of fluvially deposited gravels. Four image-segmentation procedures are developed, each having several internal parameters, giving a total of 416 permutations. These are executed on 39 images from three field sites at which the clasts have contrasting physical properties. The performance of each procedure is evaluated against a sample of manually digitized grains in the same images, by comparing three derived statistics. The results demonstrate that it is relatively straightforward to develop procedures that satisfactorily identify objects in any single image or a set of images with similar sedimentary characteristics. However, the optimal procedure is that which gives consistently good results across sites with dissimilar sedimentary characteristics. We show that neighborhood-based operations are the most powerful, and a morphological bottom-hat transform with a double threshold is optimal. It is demonstrated that its performance approaches that of the procedures giving the best results for individual sites. Overall, it out-performs previously published, or improvements to previously published, methods.*

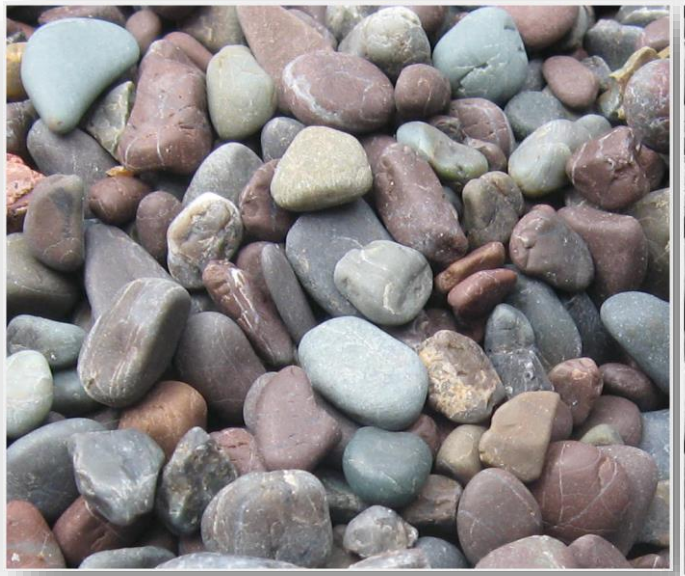
**KEY WORDS:** grain-size analysis, mathematical morphology, segmentation, bottom-hat transform, rivers, fluvial.

# Edge Detection #2

1. Canny -> Dilation -> Skeletonization




```
>> I=imread('pebbles.png');
>> imshow(I);
```

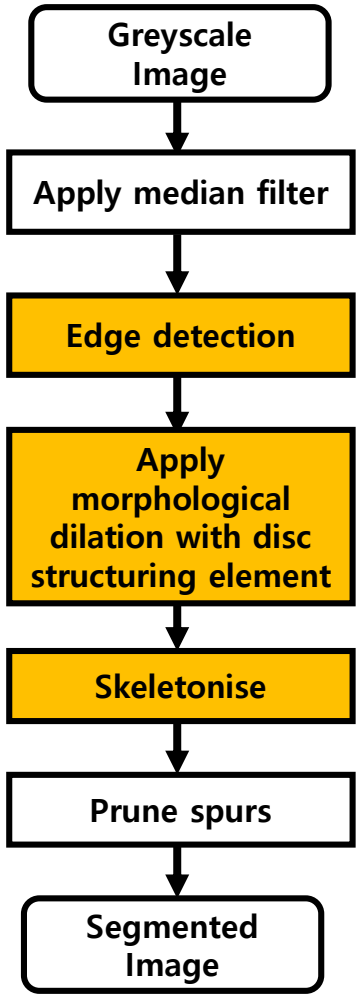


```
>> G=rgb2gray(I);figure;
>> imshow(G)
```

```
>> fSize=7;
>> S=medfilt2(G, [fSize fSize]);
>> figure; imshow(S);
```

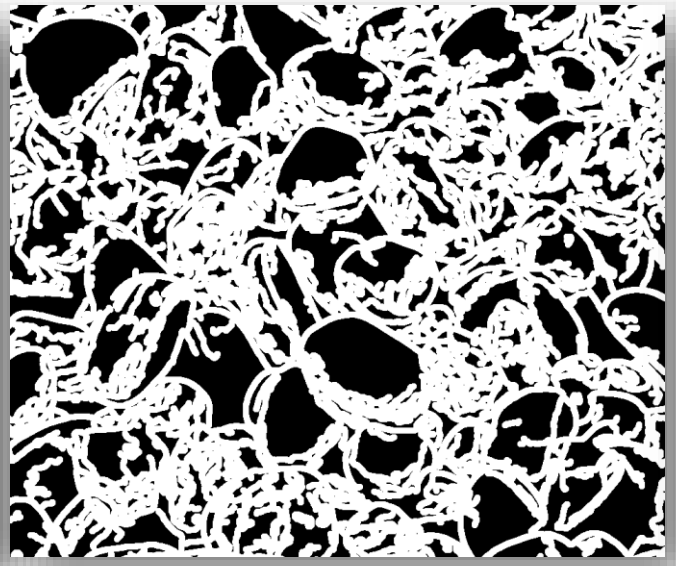
# Edge Detection #2

1. Canny -> Dilation -> Skeletonization

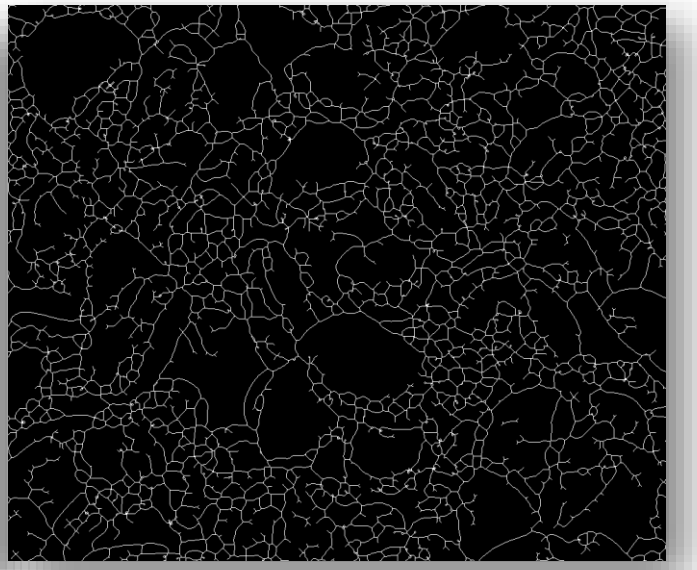



```
>> E2=edge(S, 'canny');
>> figure; imshow(E2)
```

```
>> radius=5;
>> h=strel('disk',radius);
>> E2=imdilate(E2,h);
>> figure; imshow(E3)
```

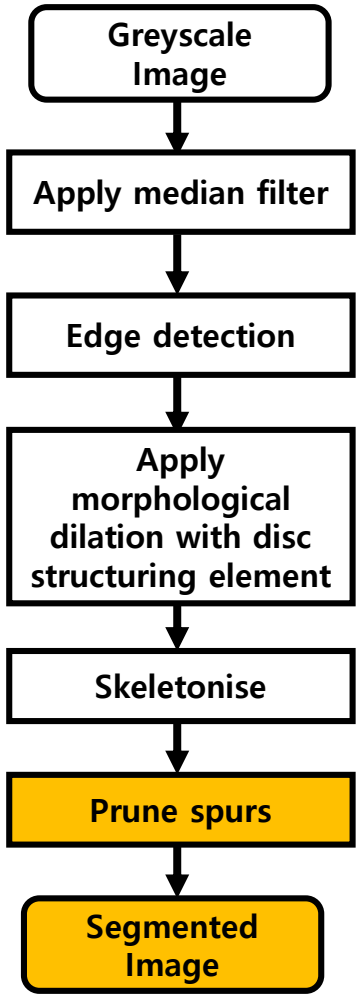


```
>> E4=bwmorph(E3, 'skel',Inf);
>> figure; imshow(E4)
```

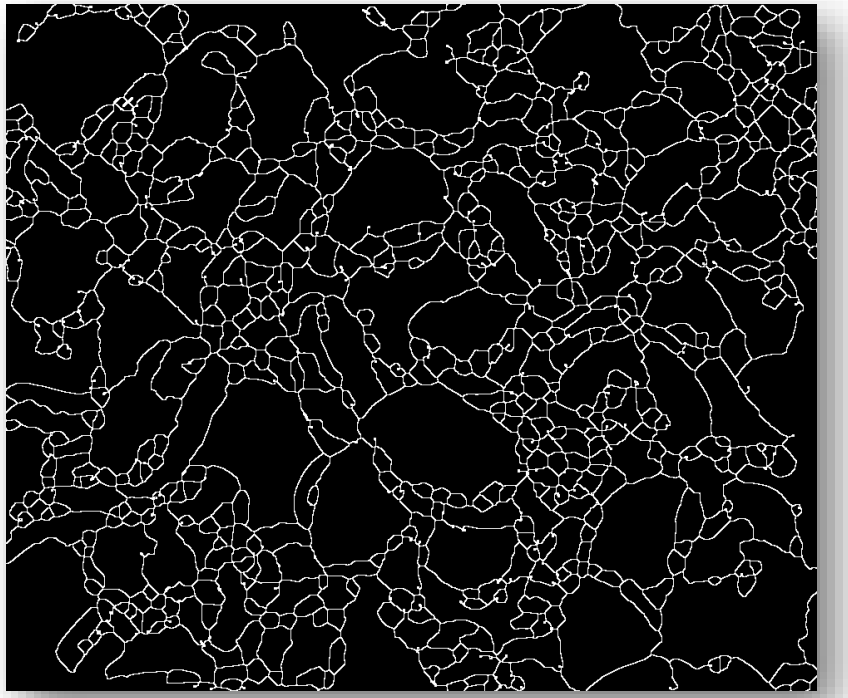


# Edge Detection #2

1. Canny -> Dilation -> Skeletonization



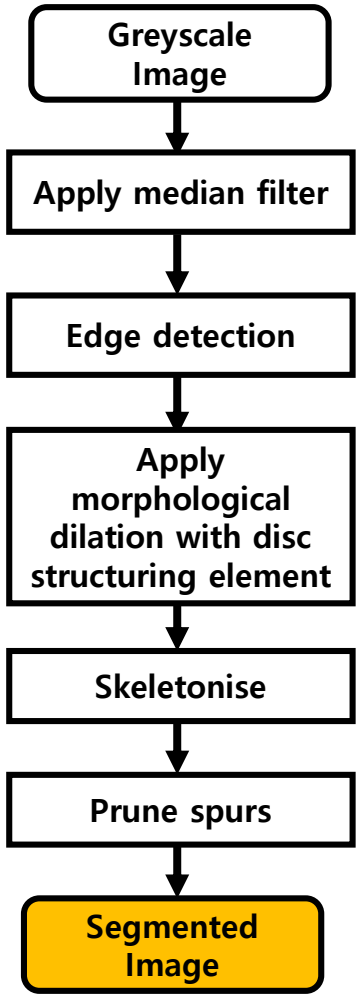

```
>> E5=bwmorph(E4, 'shrink', Inf);
>> E6=bwmorph(E5, 'spur',Inf);
>> E7=bwmorph(E6, 'clearn',Inf);
>> E7=bwmorph(E6, 'clean',Inf);
>> E8=bwmorph(E7,'diag',Inf);
>> figure; imshow(E8);
```



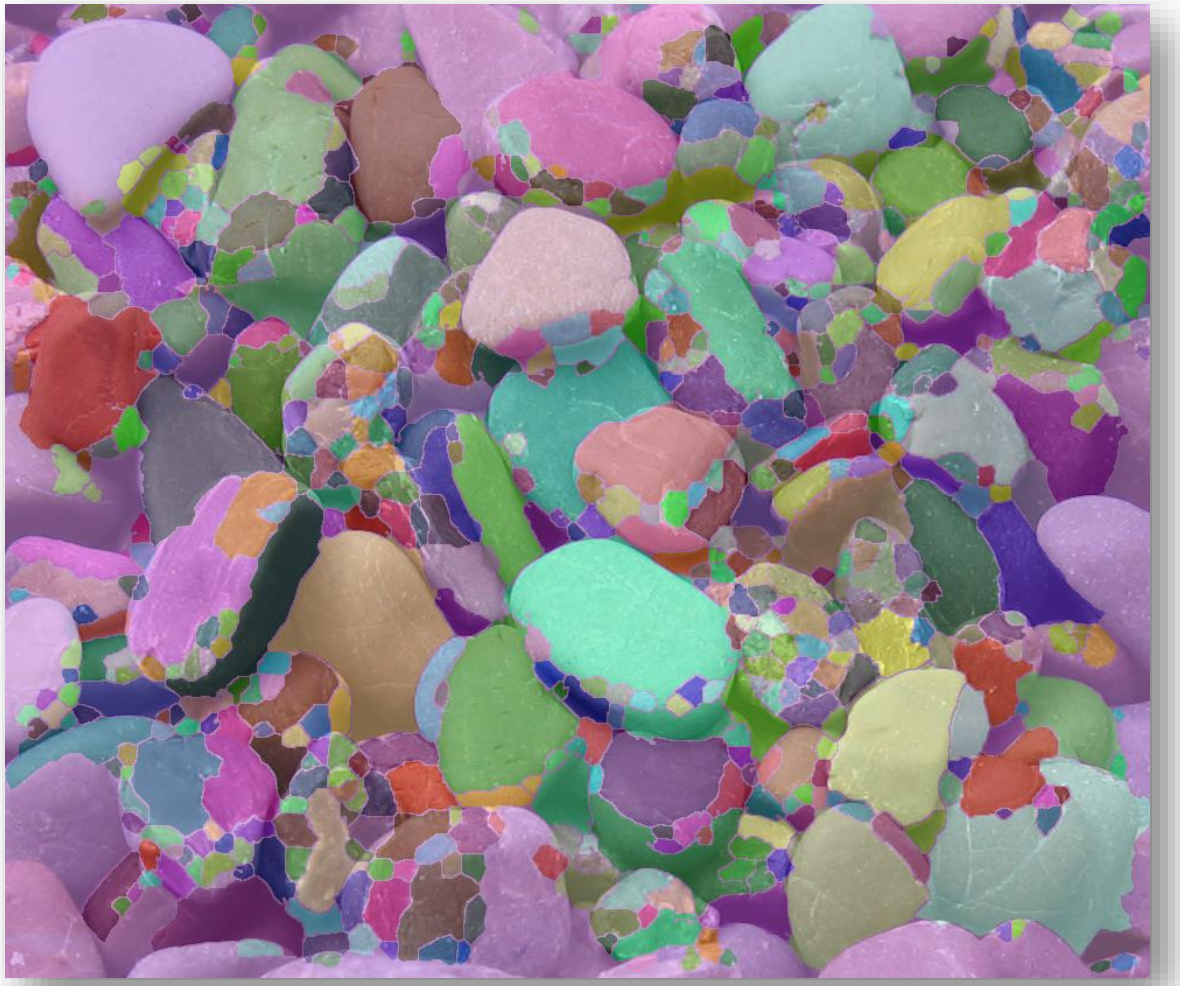
```
>> I1=I(:,:,1); I1(E8==1)=255;
>> I2=I(:,:,2); I2(E8==1)=0;
>> I3=I(:,:,3); I3(E8==1)=255;
>> II=cat(3, I1, I2, I3);
>> figure; imshow(II)
```

# Edge Detection #2

1. Canny -> Dilation -> Skeletonization

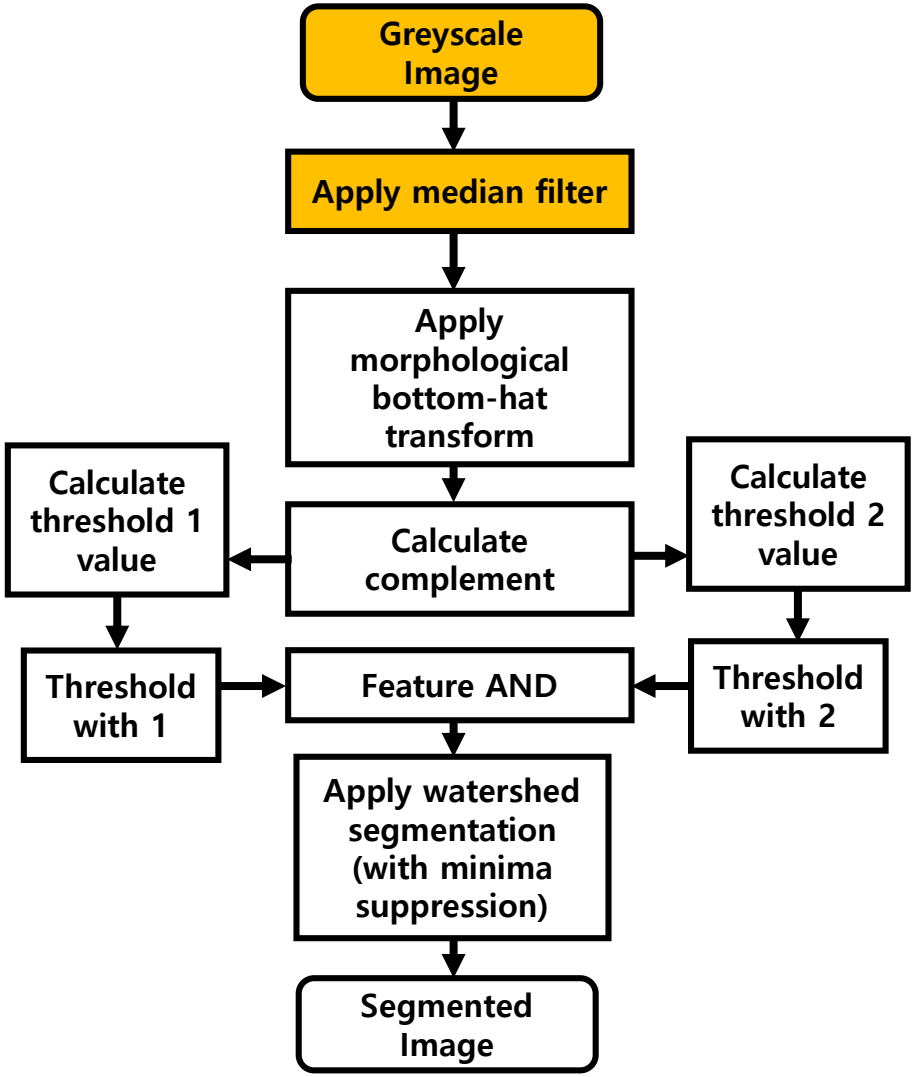



```
>> C=imcomplement(E8);
>> [L numCC]=bwlabel(C);
>> map=rand(numCC, 3);
>> transp=0.5;
>> mask=zeros(size(L))+transp;
>> figure; imshow(I);
>> hold on
>> h=imshow(L,map);
>> set(h, 'AlphaData',mask);
>> hold off
```



# Edge Detection #2

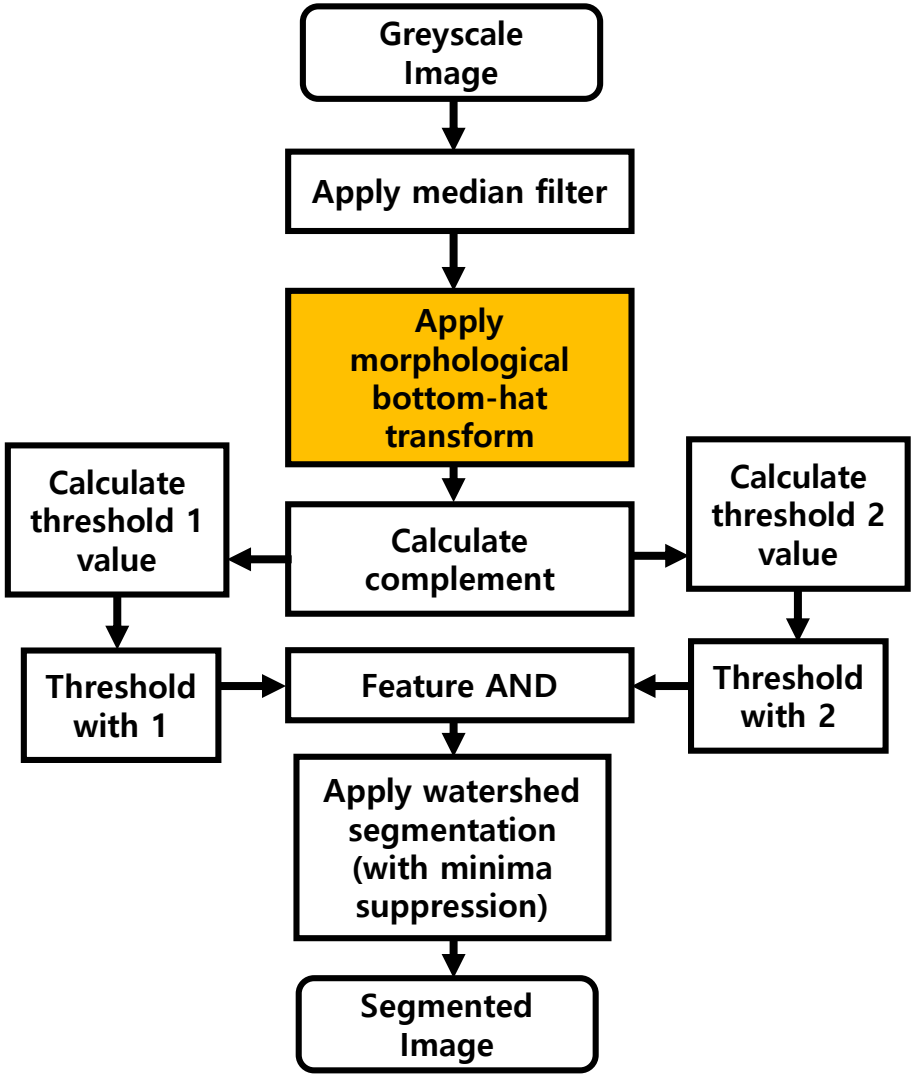
2. Bottom-hat -> Distance -> Watershed




```
clear all; close all;
I=imread('pebbles.png');
imshow(I);
G=rgb2gray(I);
figure; imshow(G);
fSize = 7;
S=medfilt2(G, [fSize fSize]);
figure; imshow(S);
```

# Edge Detection #2

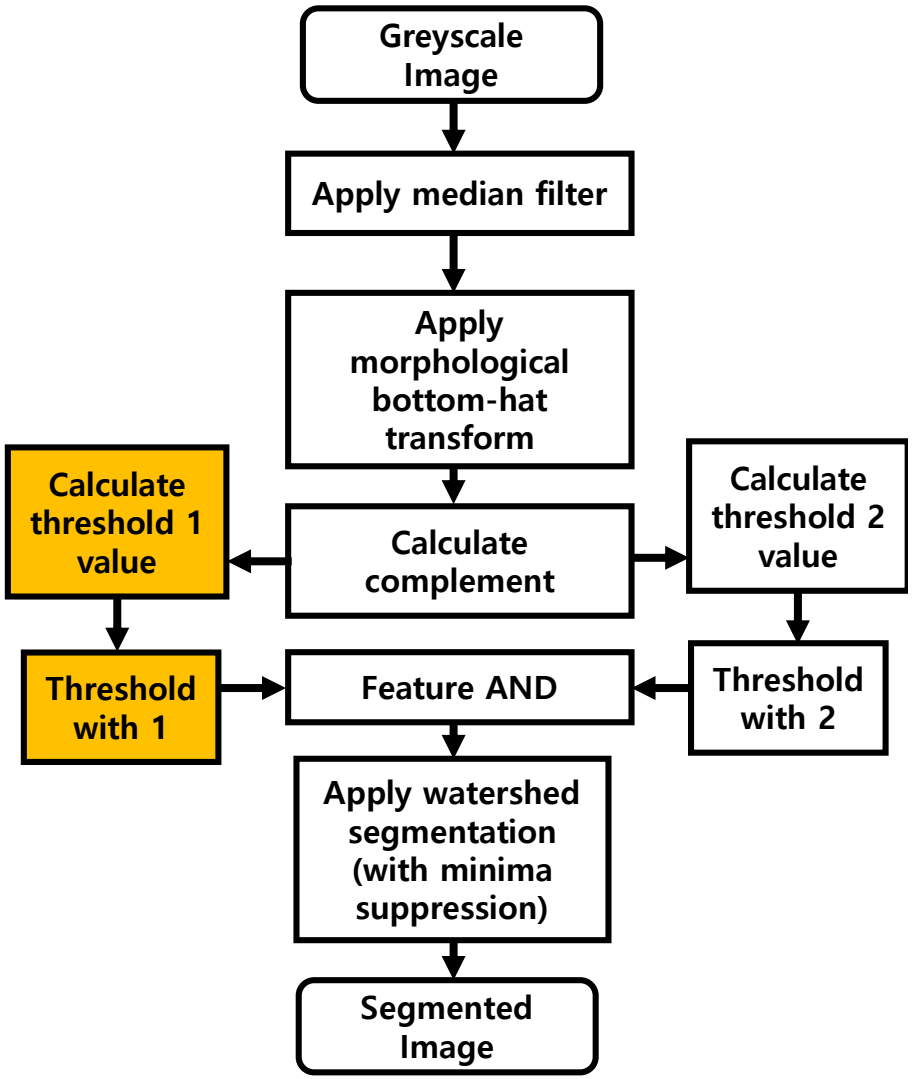
2. Bottom-hat -> Distance -> Watershed



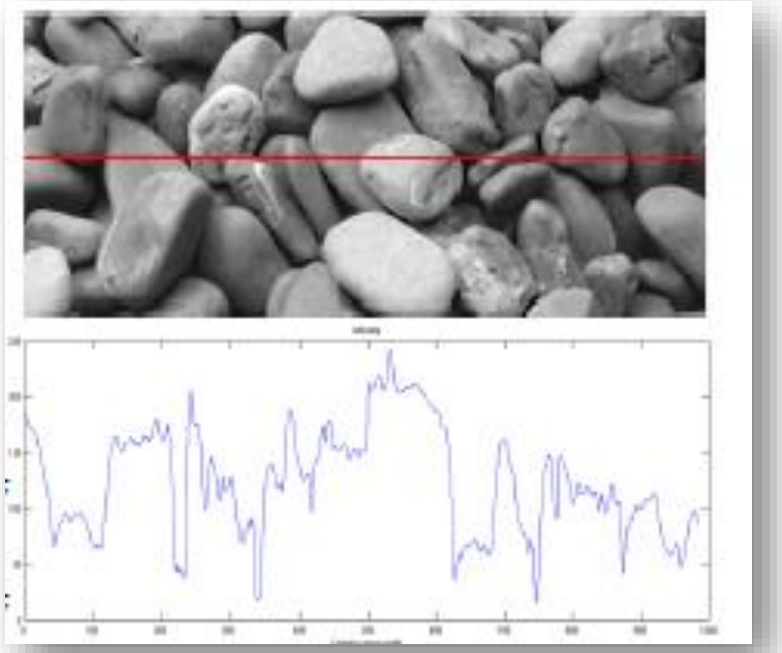

```
bh_radius = 5;
SE = strel('disk',bh_radius);
S2 = imbothat(S, SE);
figure, imshow(imadjust(S2));
BH=imsubtract(S,S2);
figure, imshow(BH);
```

# Edge Detection #2

## 2. Bottom-hat -> Distance -> Watershed

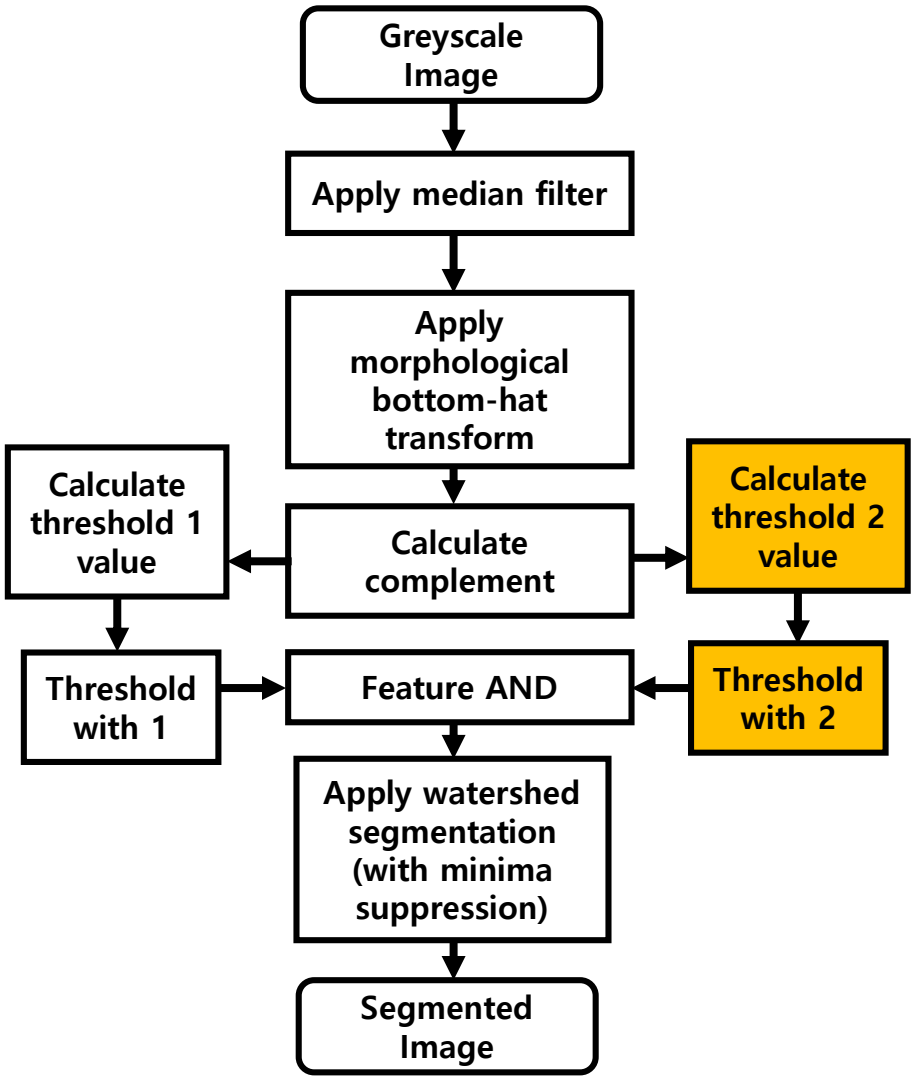



```
II=BH;
low_thresh = 3;
[counts, intensity] = imhist(II);
cumcounts = cumsum(counts);
cumpercent = ...
    cumcounts/max(cumcounts)*100;
cum1 = cumpercent(1:end-1);
cum2 = cumpercent(2:end);
if cumpercent(1) > low_thresh
    lowindex = 1;
else
    lowindex = find(cum1<=low_thresh & ...
        cum2 > low_thresh);
end
LB = (II <= intensity(lowindex));
figure; imshow(LB);
```

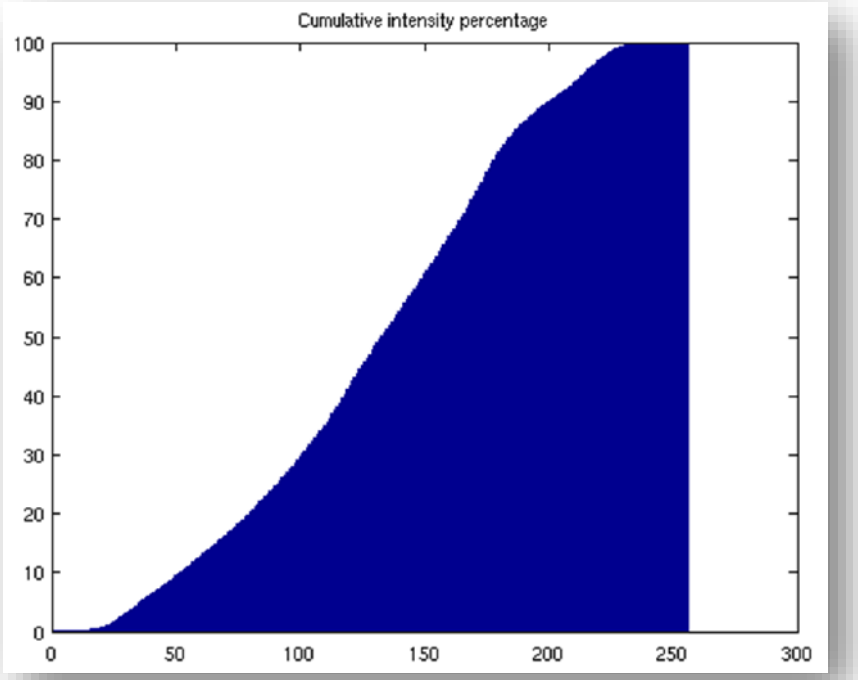


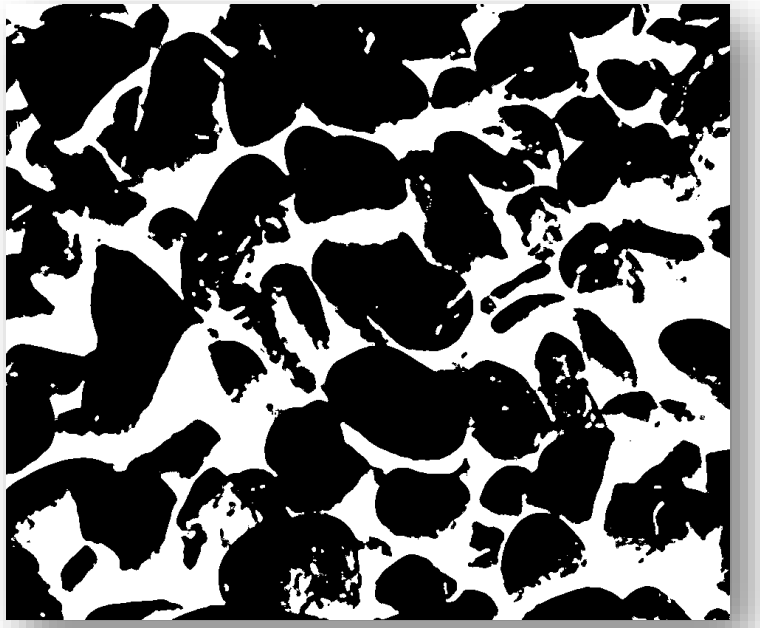
# Edge Detection #2

## 2. Bottom-hat -> Distance -> Watershed



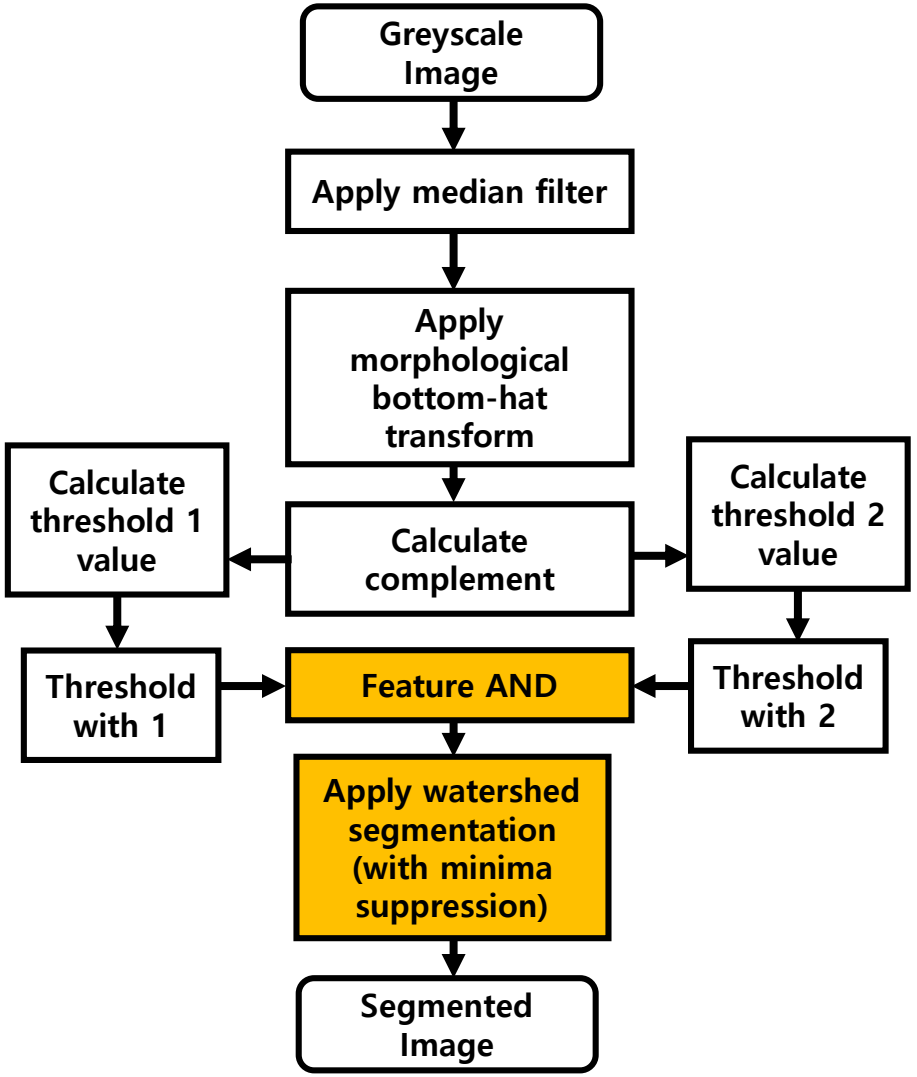

```
high_thresh = 35;
if cumpercent(1) > high_thresh
    highindex = 1;
else
    highindex = find(cum1 <= high_thresh & ...
        cum2 > high_thresh);
end
HB=(II <= intensity(highindex));
figure; imshow(HB)
```



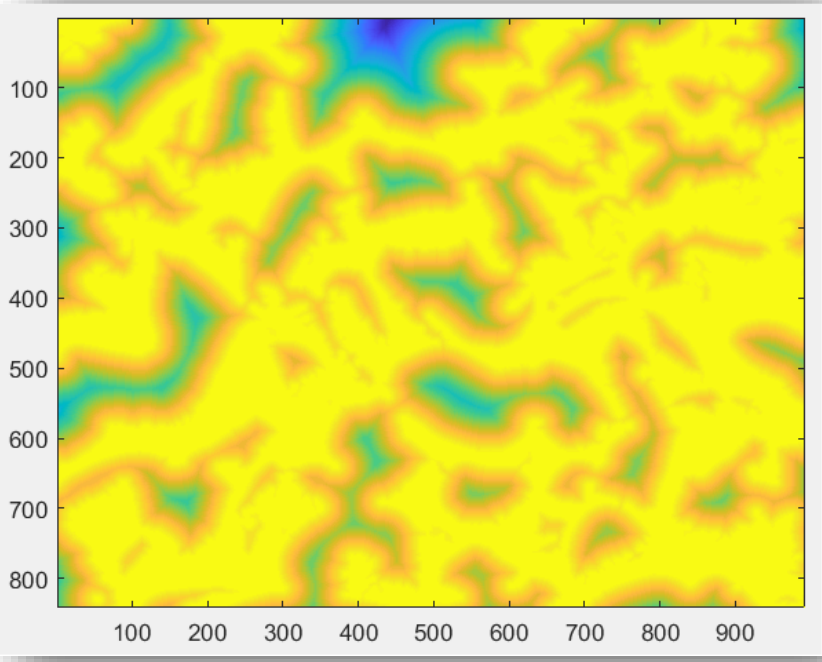



# Edge Detection #2

## 2. Bottom-hat -> Distance -> Watershed

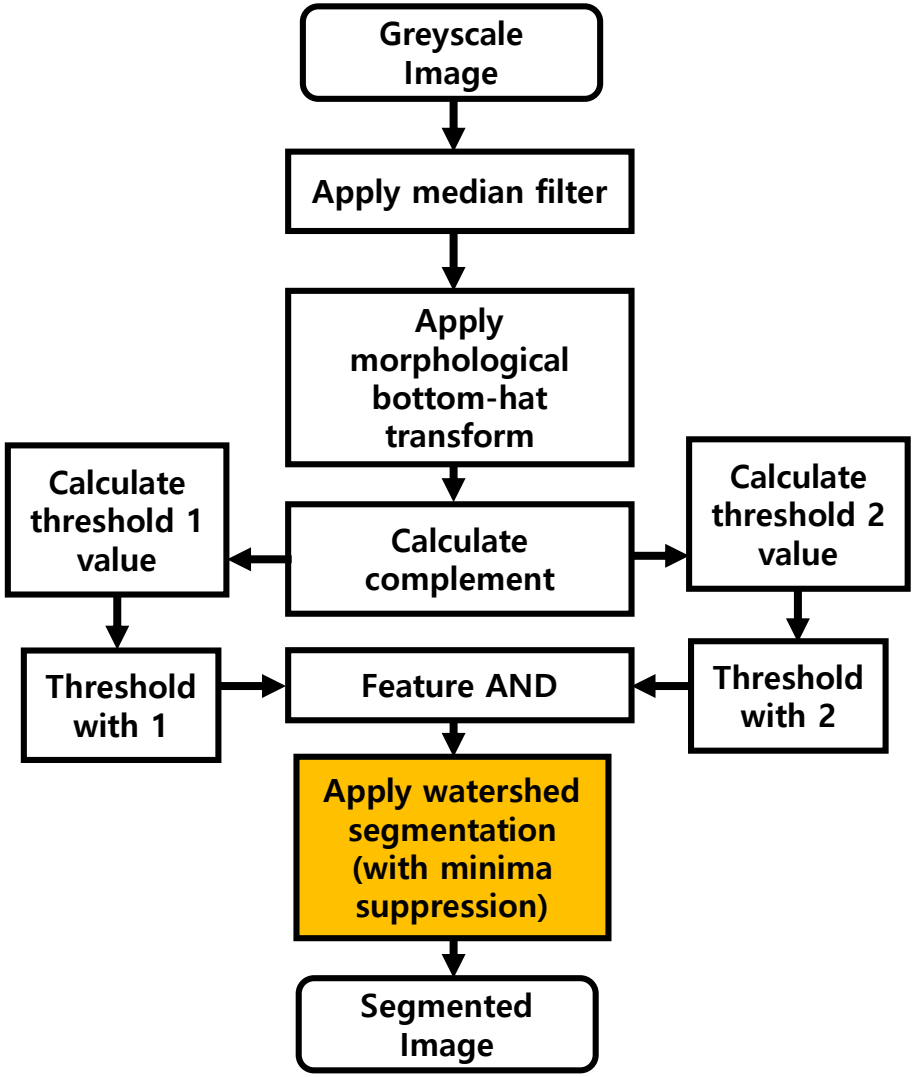



```
[indI, indJ] = find(LB > 0);
E=bwselect(HB, indI, indJ, 8);
figure, imshow(E);
DT = bwdist(E);
DT2 = -DT + max(DT(:));
hm_thresh = 2;
DT3 = imhmin(DT2, hm_thresh);
figure, imagesc(DT3);
```



# Edge Detection #2

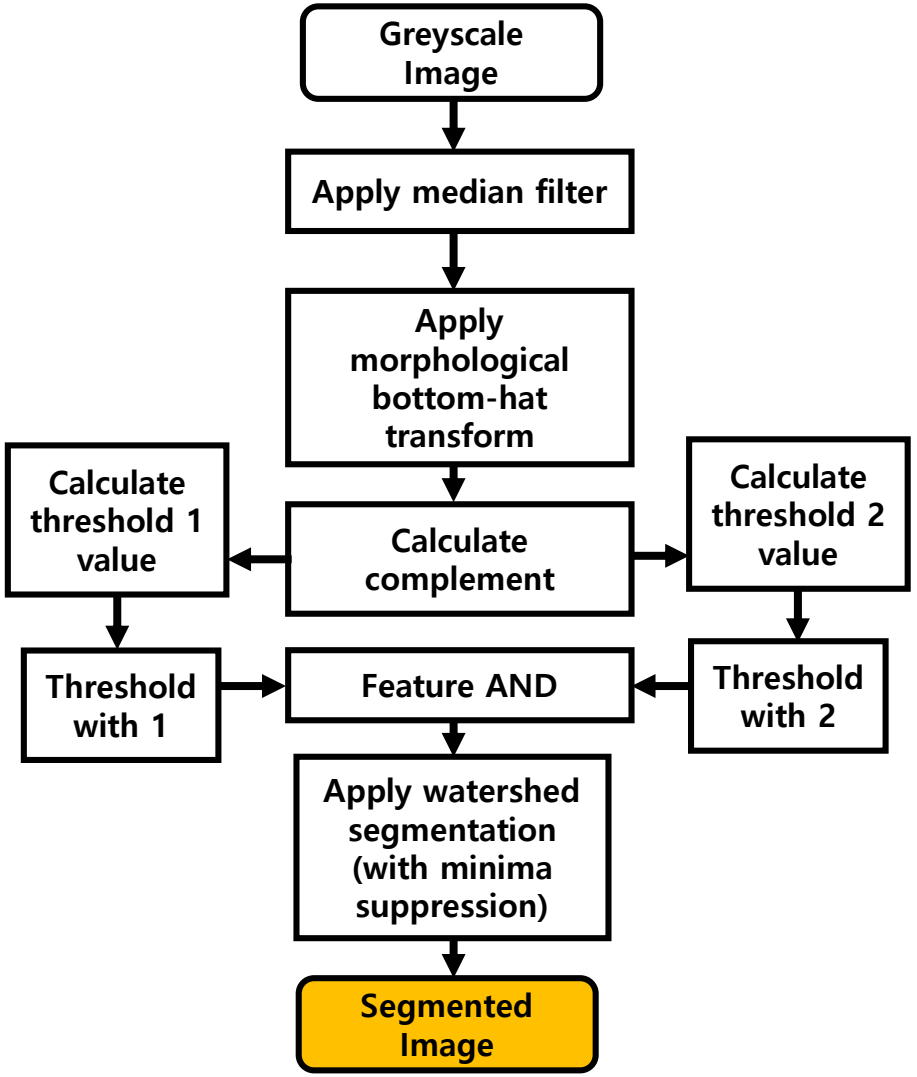
2. Bottom-hat -> Distance -> Watershed




```
L=watershed(DT3);
LO = (L==0);
G2 = G;
G2(LO) = 255;
figure; imshow(G2);
I1=I(:,:,1); I1(LO)=255;
I2=I(:,:,2); I2(LO)=0;
I3=I(:,:,3); I3(LO)=255;
II=cat(3,I1,I2,I3);
figure; imshow(II)
```

# Edge Detection #2

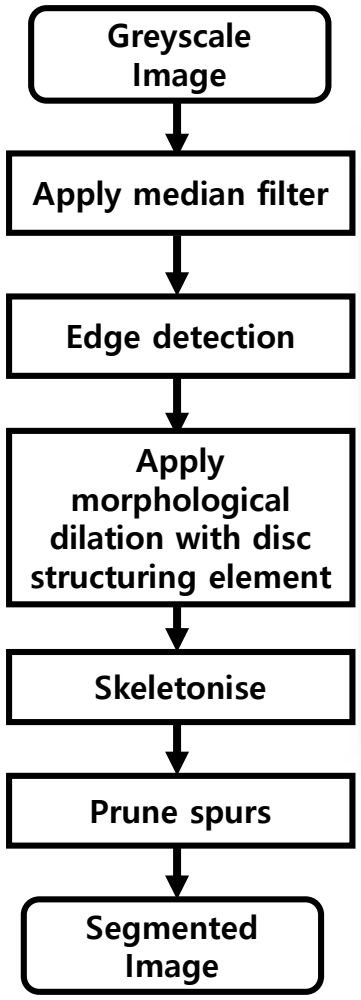
## 2. Bottom-hat -> Distance -> Watershed

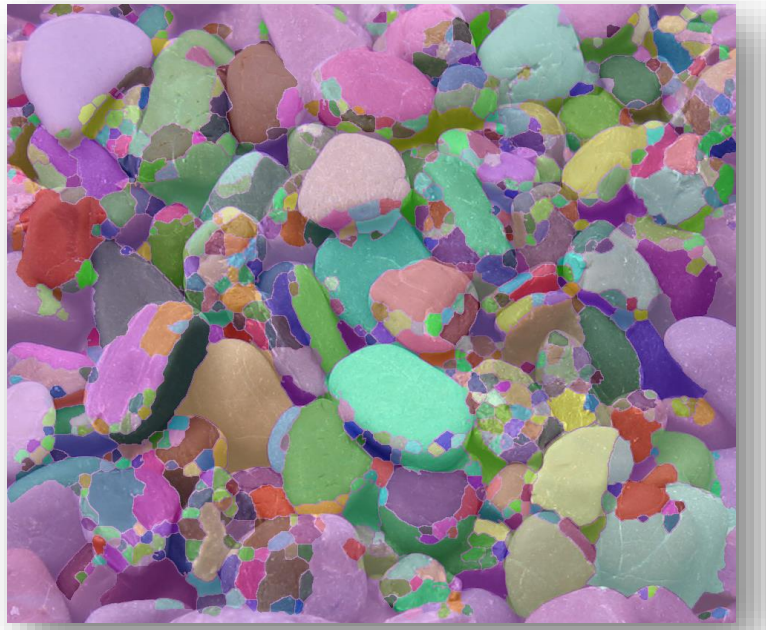



```
map=rand(max(L(:)), 3);
transp=0.5;
mask=zeros(size(L))+transp;
figure; imshow(I);
hold on
h=imshow(L,map);
set(h,'AlphaData',mask);
hold off
```

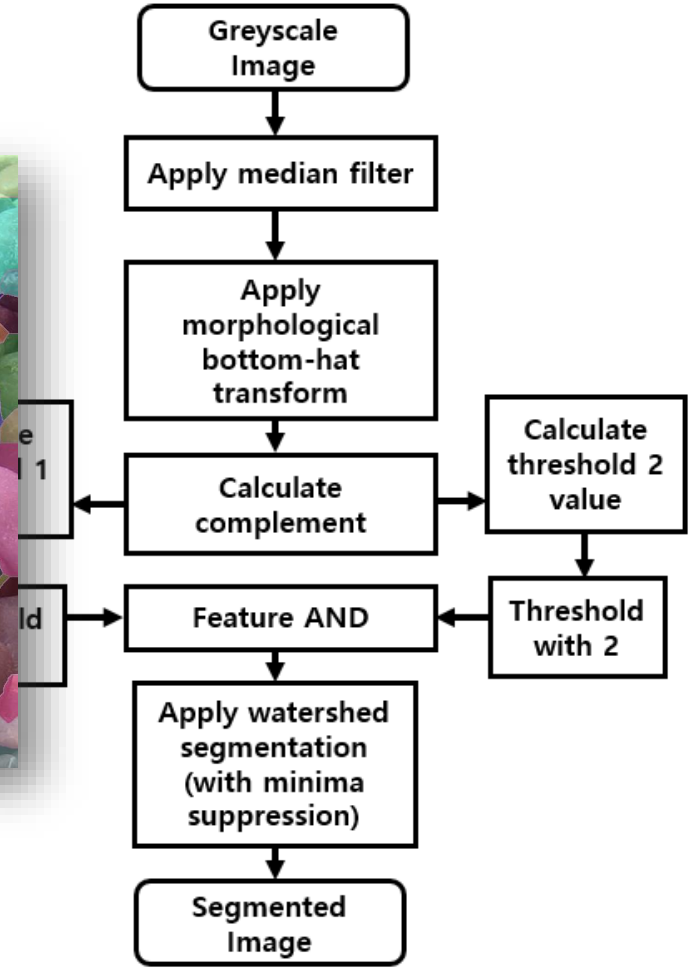
# Edge Detection #2

1. Canny -> Dilation -> Skeletonization



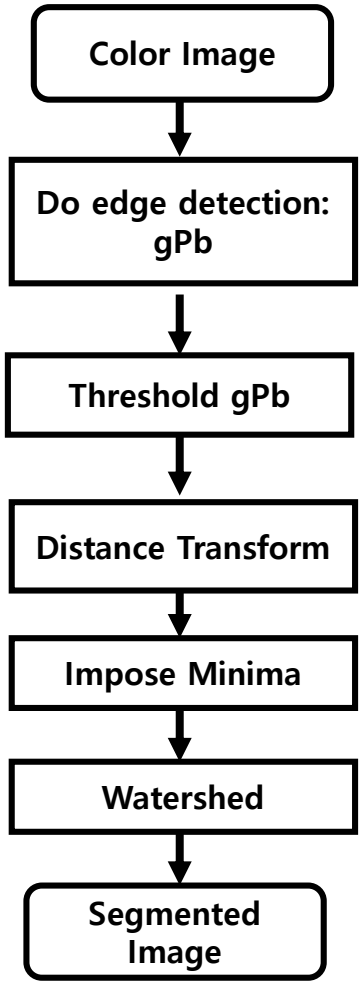



2. Bottom-hat -> Distance -> Watershed

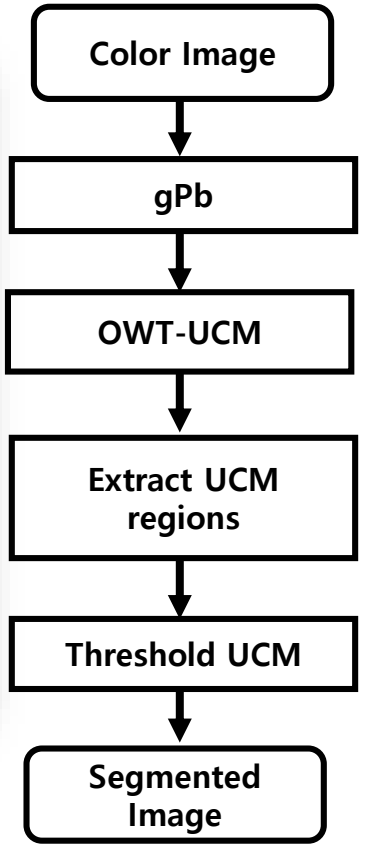


# Edge Detection #2

3. gPb -> Distance -> Watershed




4. gPb -> OWT -> UCM



# Transformation

**Fourier Transform**

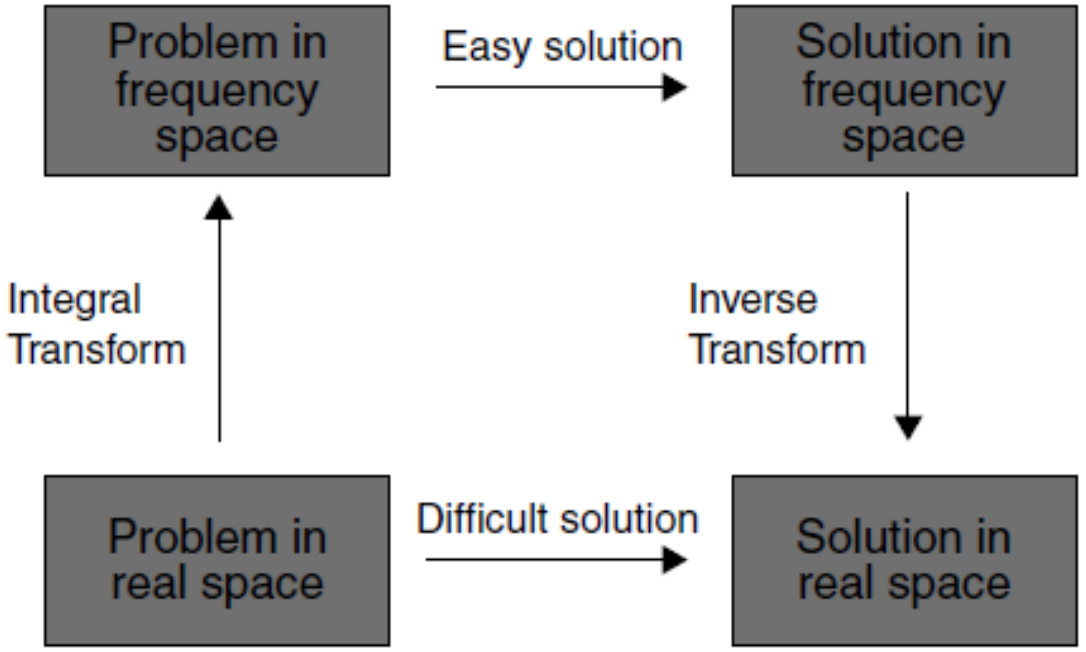
Wavelet Transform

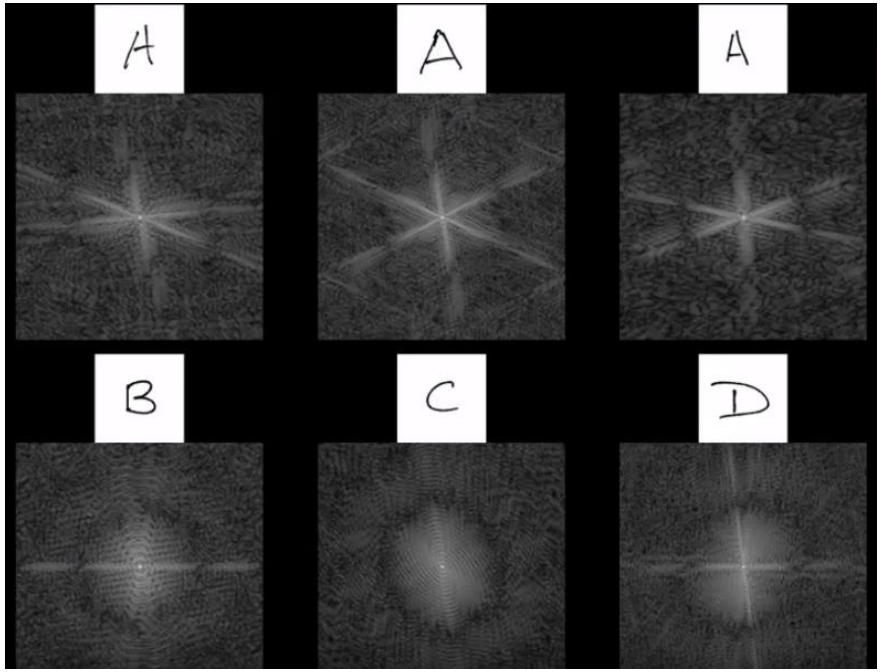
Hough Transform

- FT is a particular type of integral transform that enables us to view imaging and image processing from an alternative viewpoint by transforming the problem to another space.

- We are usually concerning with 2-D spatial distributions of intensity or color which exist in **real space** (i.e., 2-D Cartesian space in which the axes define units of length)

- The FT operates on such a function to produce an entirely equivalent form which lies in an abstract space called **frequency space.**



# Transformation (Fourier Transform)

**Harmonic content of signals.** Any signal can be expressed as a weighted linear combination of harmonic (i.e., sine and cosine) functions having different periods or frequencies. These are called the (spatial) frequency components of the signal.

Time series

$$V(t) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n t}{T}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi n t}{T}\right)$$

$$= \sum_{n=0}^{\infty} a_n \cos(\omega_n t) + \sum_{n=1}^{\infty} b_n \sin(\omega_n t)$$

Note: n-> infinite for exact reproducibility

*Fourier basis functions* $\sin(k_n x)\ \cos(k_n x)$

*Spatial frequency* $k_n = 2\pi n/\lambda$

*Fourier or frequency spectrum* $\{a_n \text{ and } b_n\}$

Spatial series

$$V(x) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n x}{\lambda}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi n x}{\lambda}\right)$$

$$= \sum_{n=0}^{\infty} a_n \cos(k_n x) + \sum_{n=1}^{\infty} b_n \sin(k_n x)$$

# Transformation (Fourier Transform)

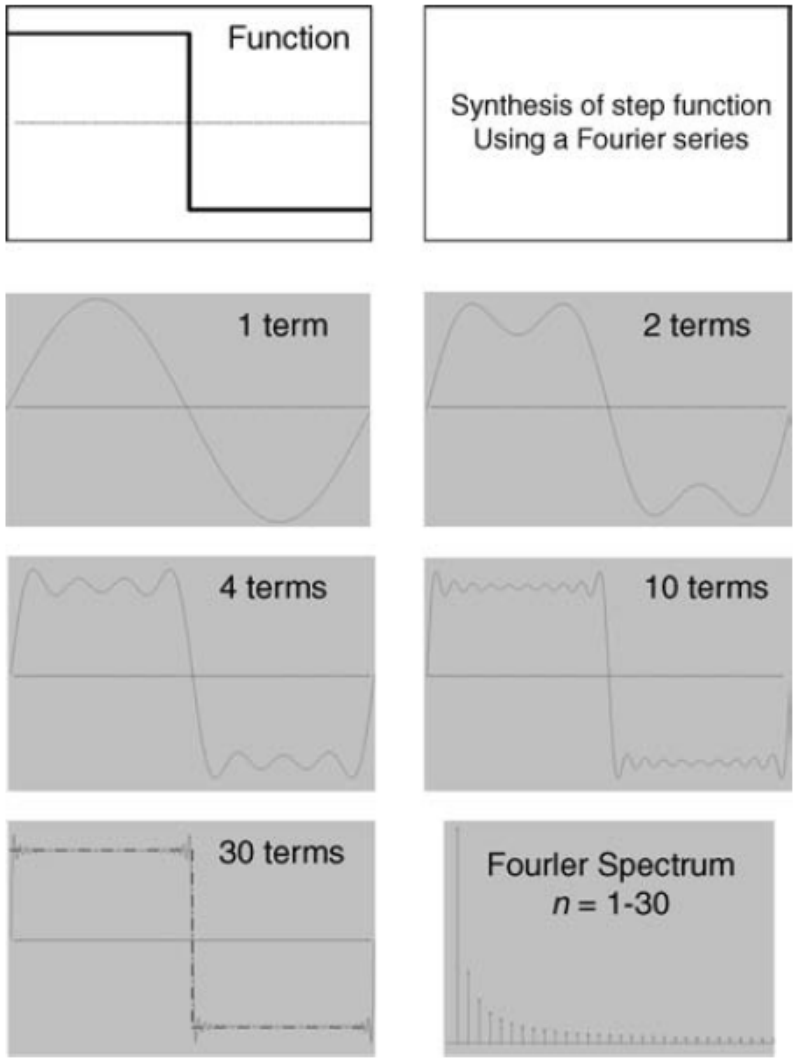
$$V(x) = \sum_{n=0}^{\infty} a_n \cos(k_n x) + \sum_{n=1}^{\infty} b_n \sin(k_n x)$$

$$\sin(k_n x) = \sin(2\pi n x / \lambda)$$

Fourier representation of a signal as a weighted combination of harmonic functions of different frequencies, the assigned weights constitute the **Fourier spectrum**. It is a complete and valid, alternative representation of the signal.

**The space domain and the Fourier domain are reciprocal.**
Harmonic terms with high frequencies (short periods) are needed to construct small-scale (i.e. sharp or rapid) changes in the signal.
Conversely, smooth features in the signal can be represented by harmonic terms with low frequencies (long periods). The two domains are thus reciprocal.

**Fourier series expansion and Fourier transform**
The difference is that the Fourier series breaks down a periodic signal into harmonic functions of discrete frequencies. Whereas, the Fourier transform breaks down a nonperiodic signal into harmonic functions of continuously varying frequencies.



# Transformation (Fourier Transform)

**Calculation of the Fourier spectrum** $\{a_n \text{ and } b_n\}$

*Harmonic function form*

$$V(x) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n x}{\lambda}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi n x}{\lambda}\right)$$

$$= \sum_{n=0}^{\infty} a_n \cos(k_n x) + \sum_{n=1}^{\infty} b_n \sin(k_n x)$$

$$a_n = \frac{2}{\lambda}\int_{\lambda/2}^{\lambda/2} f(x)\cos(k_n x)\,\mathrm{d}x$$

$$b_n = \frac{2}{\lambda}\int_{\lambda/2}^{\lambda/2} f(x)\sin(k_n x)\,\mathrm{d}x$$

where $k_n = 2\pi n/\lambda$.

*Complex exponential form*

$$f(x) = \sum_{n=-\infty}^{\infty} c_n \exp\left(\frac{\mathrm{i}2\pi n x}{\lambda}\right)$$

$$= \sum_{n=-\infty}^{\infty} c_n \exp(\mathrm{i}k_n x)$$

$$c_n = \frac{1}{\lambda}\int_{\lambda/2}^{\lambda/2} f(x)\exp(\mathrm{i}k_n x)\,\mathrm{d}x$$

where $c_k = a_k + ib_k$

# Transformation (Fourier Transform)

**Fourier transform of f(x):**

$$F(k_x) = \frac{1}{2\pi} \int_{\infty}^{\infty} f(x) \exp(-\mathrm{i}k_x x)\, \mathrm{d}x$$

**Inverse Fourier transform:**

$$f(x) = \int_{\infty}^{\infty} F(k_x) \exp(\mathrm{i}k_x x)\, \mathrm{d}k_x$$

Specific weight (contribution) assigned to the harmonic (complex exponential) functions are given by the function $F(k_x)$

**Fourier transform**

$$F(k_x) = |F(k_x)| \exp \mathrm{i}\varphi(k_x)$$

where

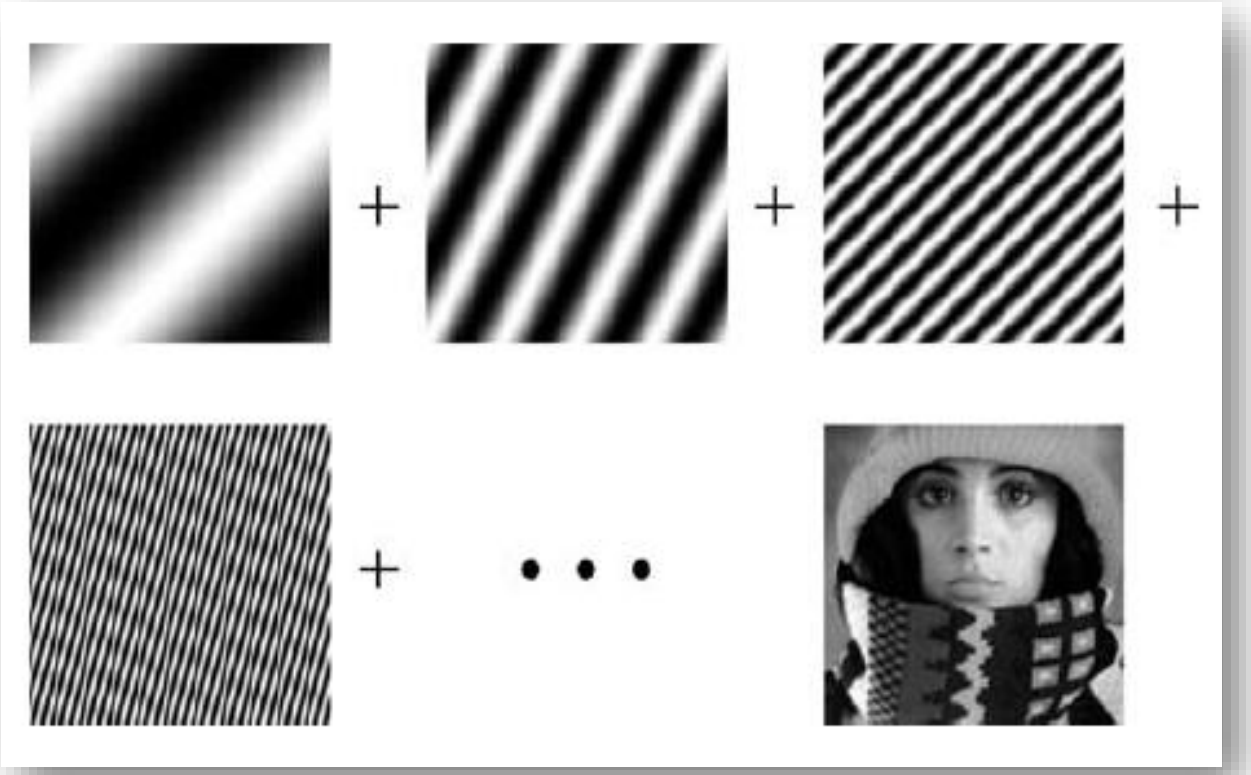
$$|F(k_x)|^2 = [\mathrm{Re}\{F(k_x)\}]^2 + [\mathrm{Im}\{F(k_x)\}]^2 \qquad \varphi(k_x) = \tan^{1}\left[\frac{\mathrm{Im}\{F(k_x)\}}{\mathrm{Re}\{F(k_x)\}}\right]$$

# Transformation (Fourier Transform)

**Fourier method in Image processing**

(1) Images are not, in general, periodic functions

(2) Images are typically 2-D spatial functions of finite support



$$f(x,y) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{\infty} \int_{\infty}^{\infty} F(f_x, f_y) \exp[2\pi i(f_x x + f_y y)] \, df_x \, df_y$$

*2-D Fourier transform*

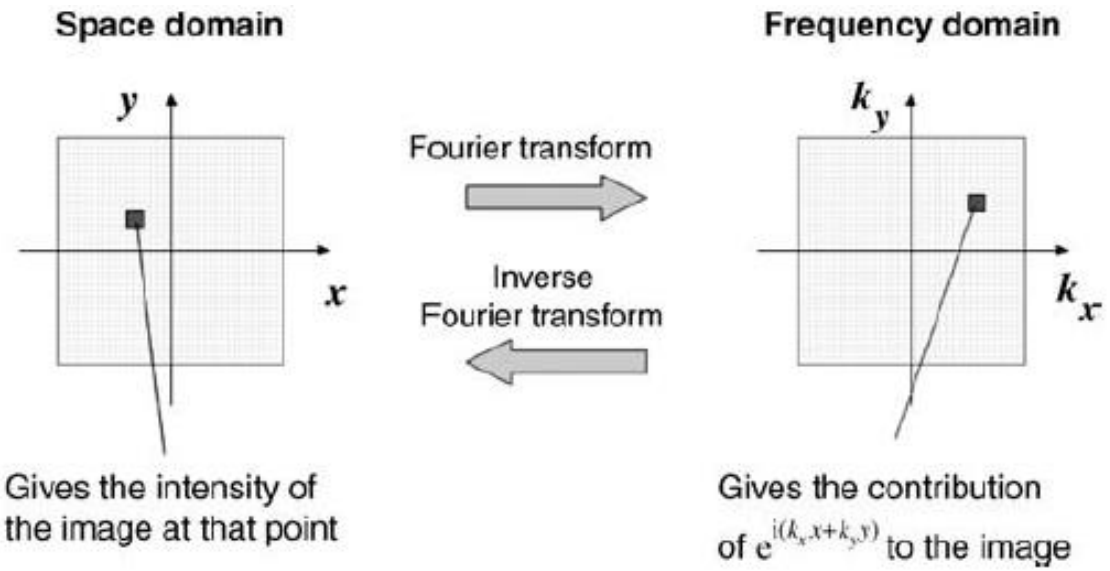
$$F(f_x, f_y) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{\infty} \int_{\infty}^{\infty} f(x,y) \exp[-2\pi i(f_x x + f_y y)] \, dx \, dy$$

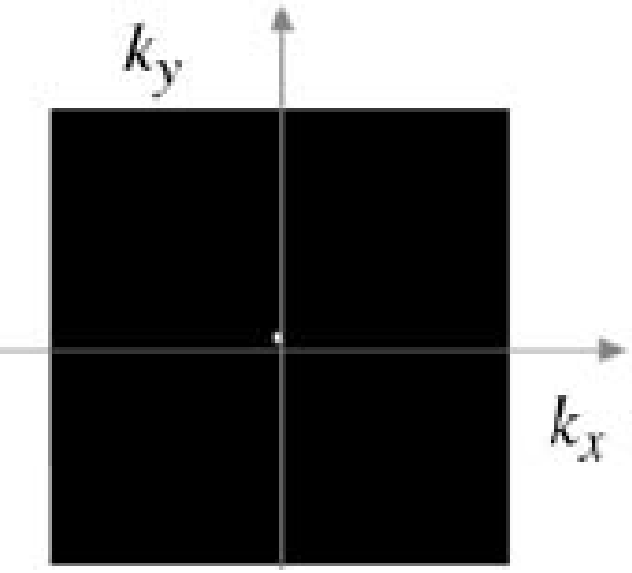
(Images from Fundamentals of Digital Image Processing, Solomon and Breckon 2010)

# Transformation (Fourier Transform)

## Fourier method in Image processing

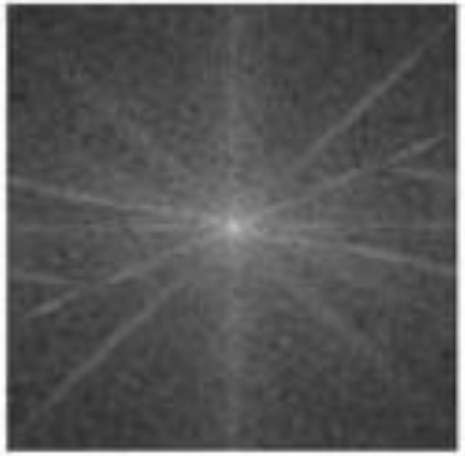
$$F(f_x, f_y) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{\infty} \int_{\infty}^{\infty} f(x, y) \exp[-2\pi i(f_x x + f_y y)] \, dx \, dy$$



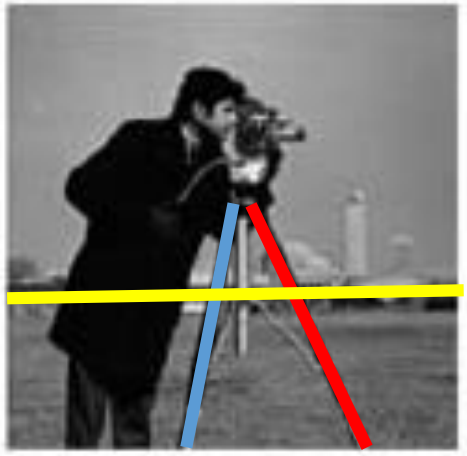




*Frequency domain*

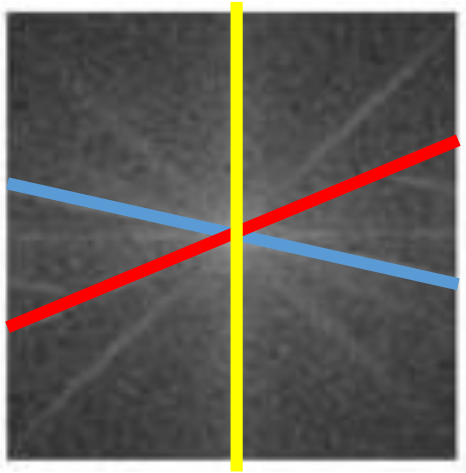
*Original image*



*FT of image*



*Original image*



*FT of image*

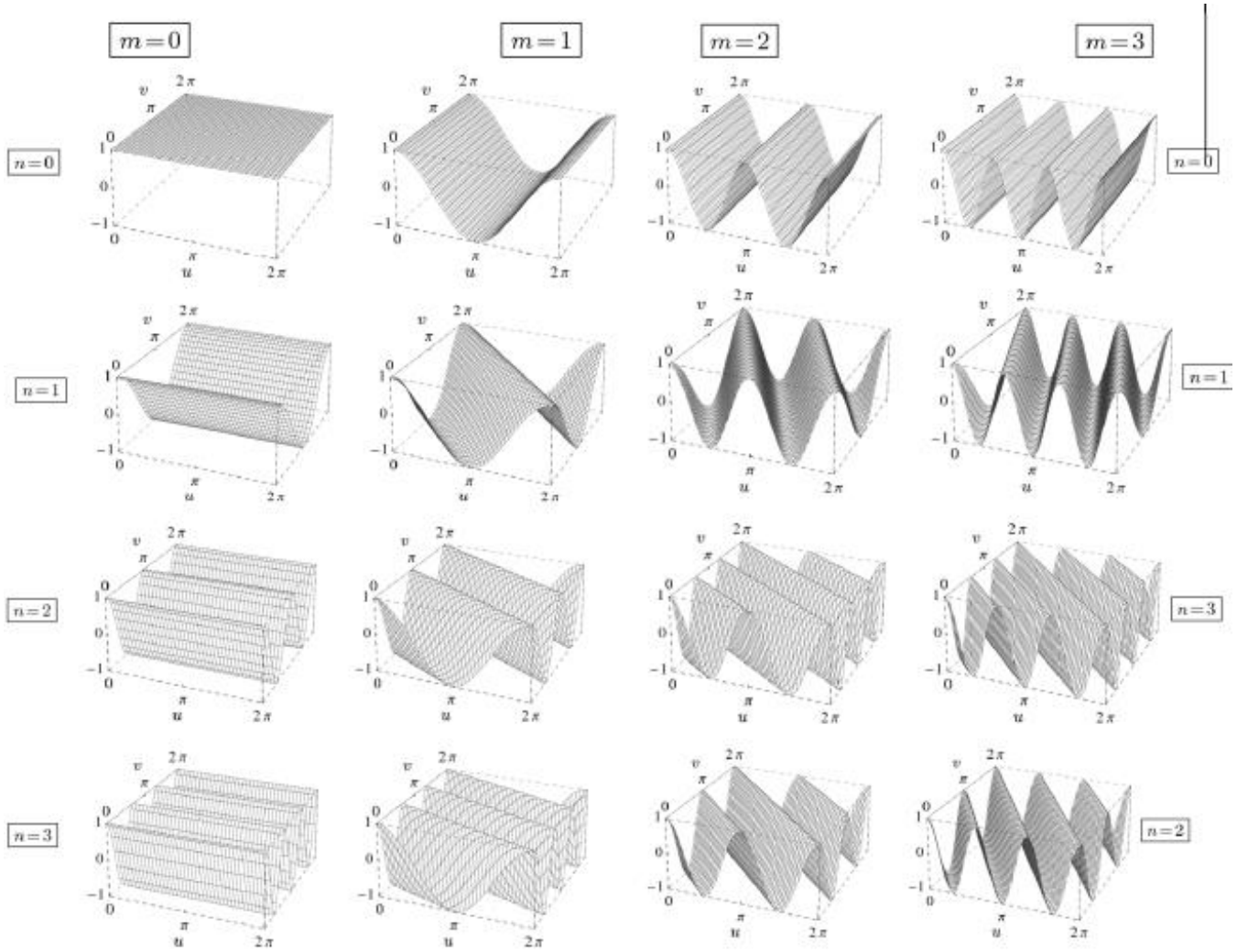
# Transformation (Fourier Transform)

## Discrete Fourier Transform (DFT)

For M x N matrix, forward and inverse Fourier transform can be:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left[-2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right].$$

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\exp\left[2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right]$$



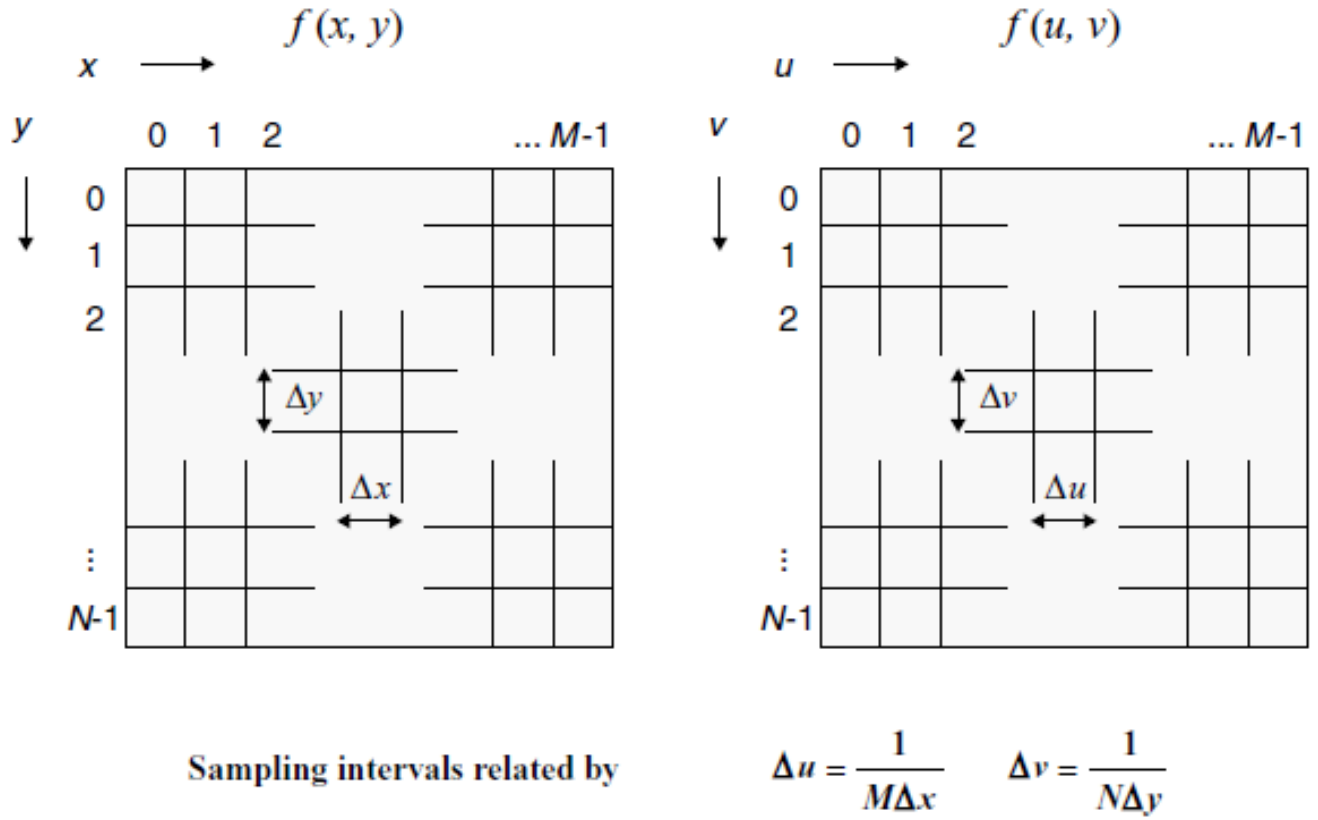
# Transformation (Fourier Transform)

## Discrete Fourier Transform (DFT)

For M x N matrix, forward and inverse Fourier transform can be:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left[-2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right]$$

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\exp\left[2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right]$$



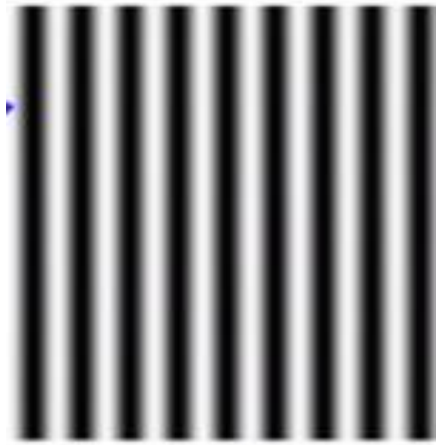
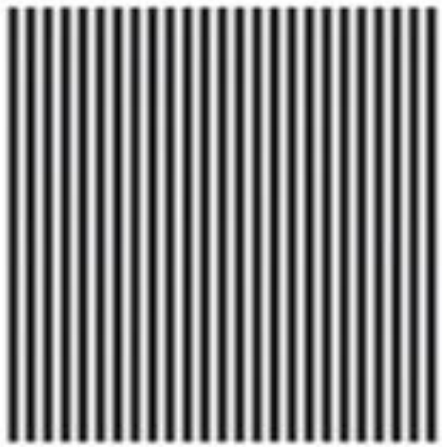

Sampling intervals related by $\Delta u = \frac{1}{M\Delta x}$ $\Delta v = \frac{1}{N\Delta y}$

(Images from Fundamentals of Digital Image Processing, Solomon and Breckon 2010)

# Transformation (Fourier Transform)

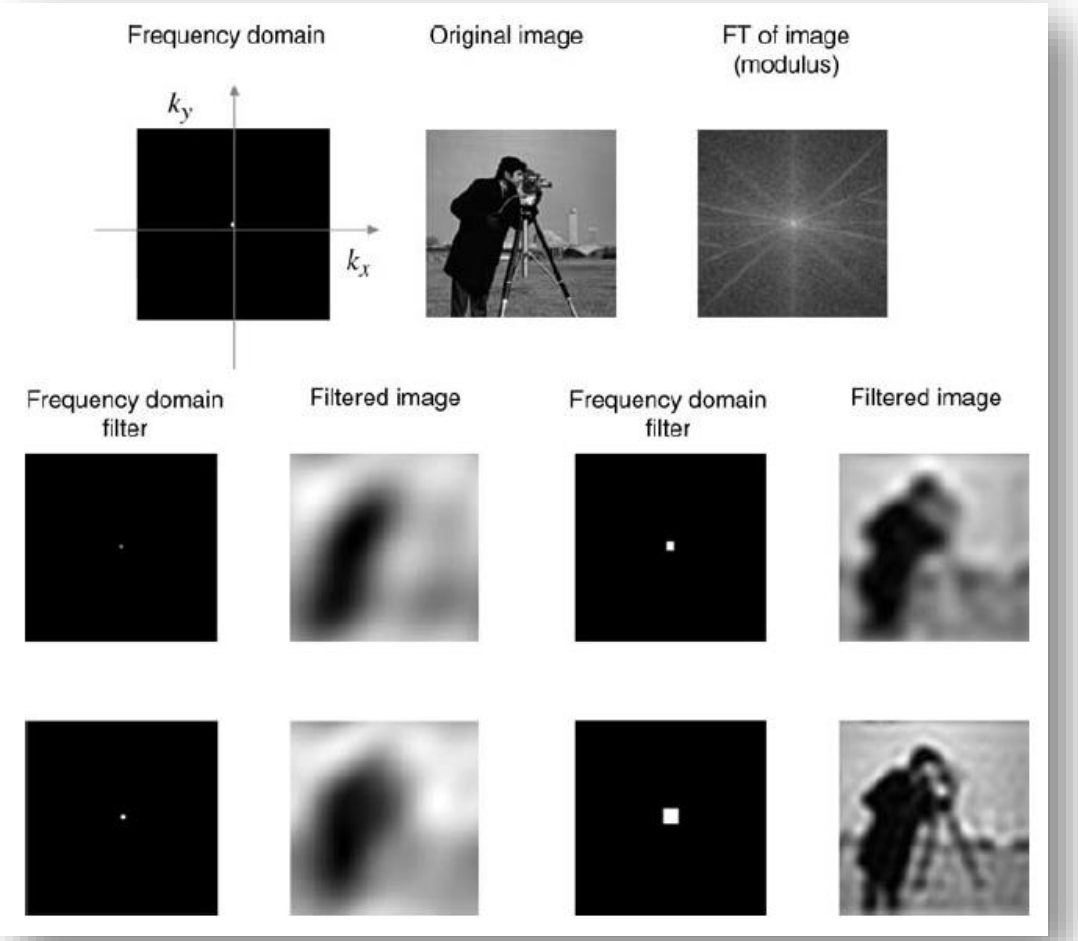
## Discrete Fourier Transform (DFT)

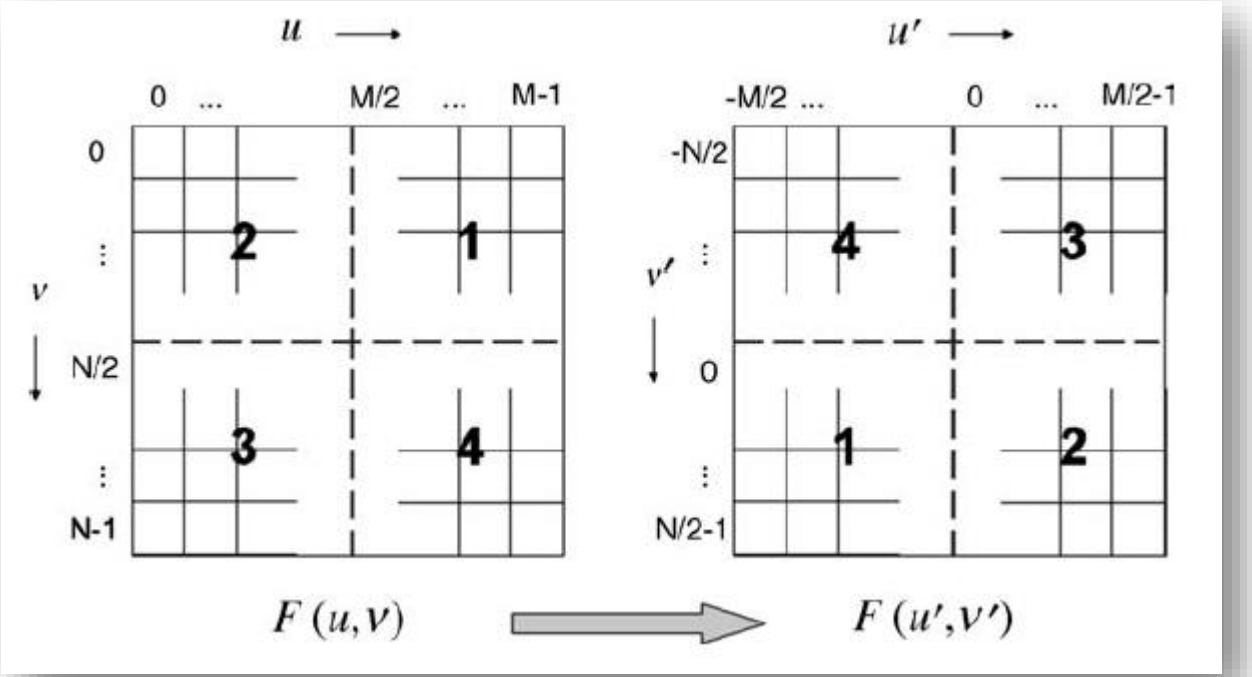
$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left[-2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)\right].$$

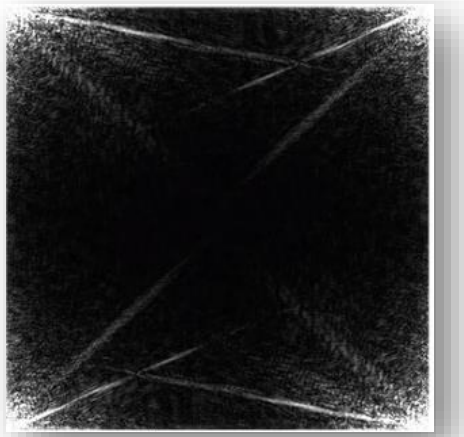



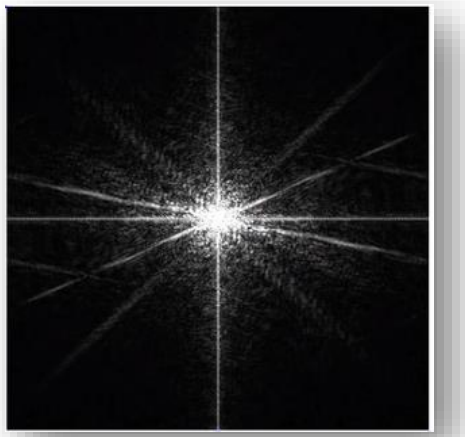

# Transformation (Fourier Transform)

## Discrete Fourier Transform (DFT)





(Images from Fundamentals of Digital Image Processing, Solomon and Breckon 2010)

# Transformation (Fourier Transform)

**Examples**

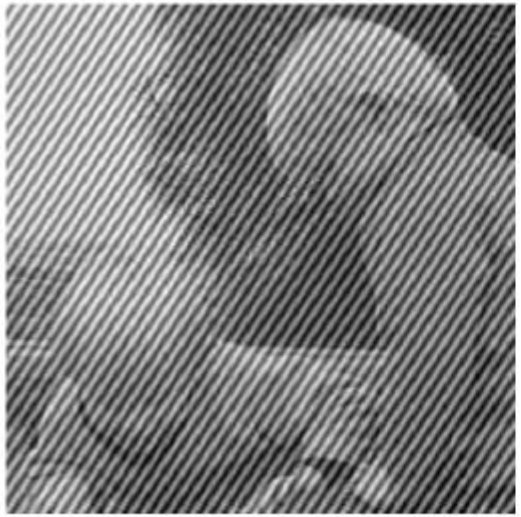
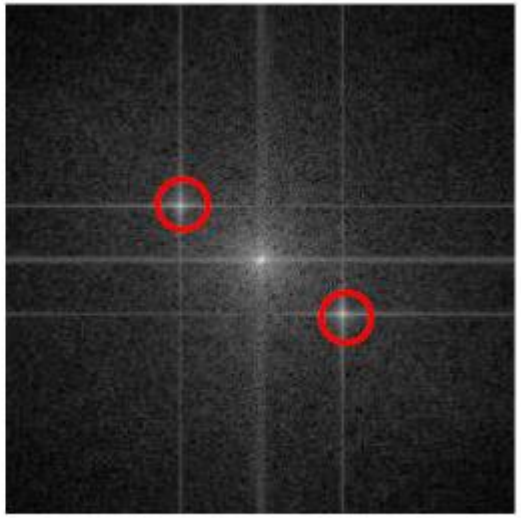
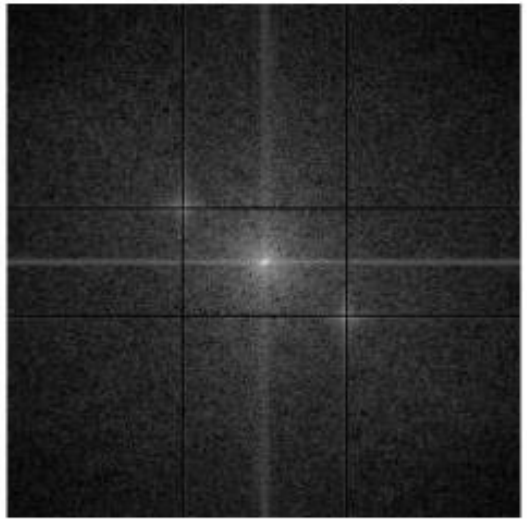


Image with periodic Noise

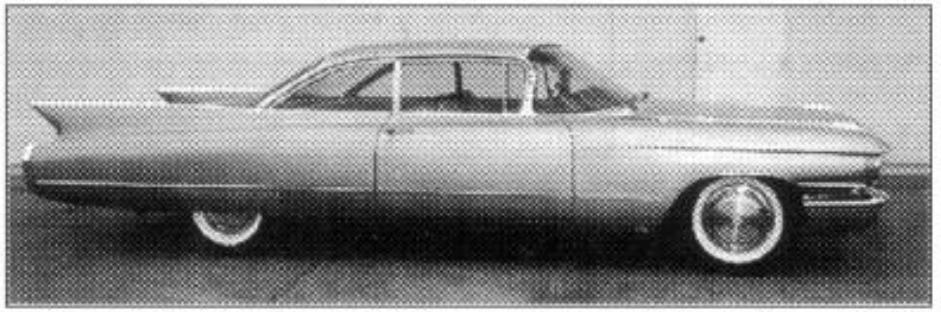


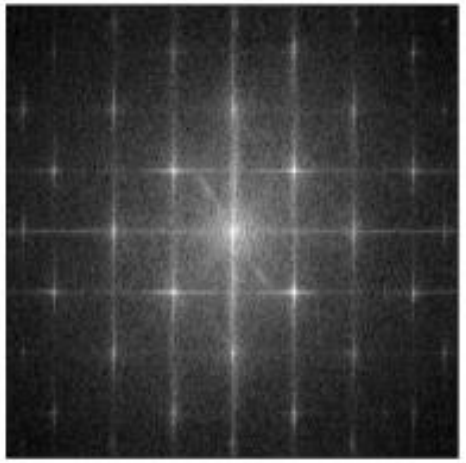
DFT of Image



Notch Filter

Result after notch filter applied then inverted



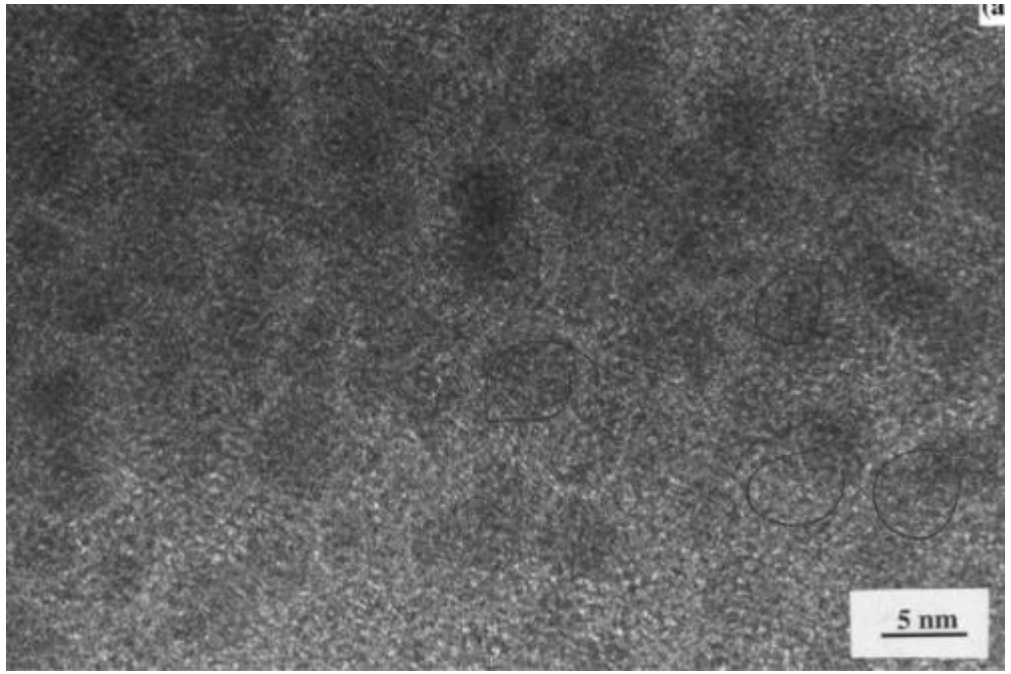


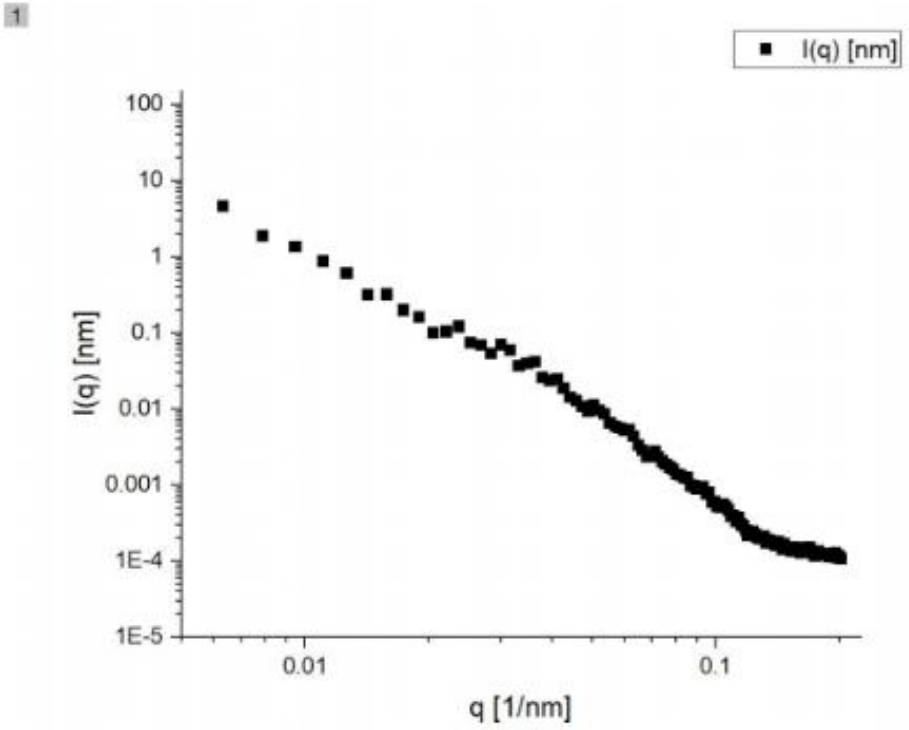
Images from

# Transformation (Fourier Transform)

**Examples**

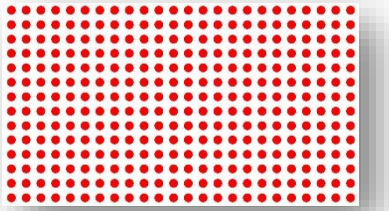
#7 Fourier Transformation of TEM image ★★★



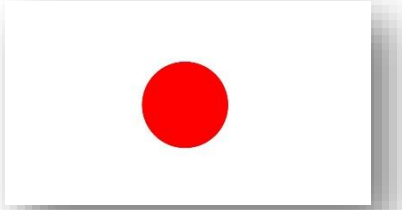




Lattice images of nanocrystalline regions in C-S-H in OPC specimen 28 d old

# Transformation



many balls.jpg



one ball.jpg

```
close all; clear all;
I=imread('many balls.jpg');
figure; imshow(I);
xg = rgb2gray(I);
figure; imshow(xg);
XG = fft2(xg, 1000, 1000);
XGM=(abs(XG));
figure; imshow(XGM*0.00005);
XGMS = fftshift(XGM);
figure; imshow(XGMS*0.00005);
```