University of Nottingham
UK | CHINA | MALAYSIA

COMP-2032

Introduction to Image Processing

Lecture 7

Segmentation

Learning Outcomes

IDENTIFY

1. What is Segmentation?
2. Region-based Segmentation
3. Edge-based Segmentation

What is Segmentation?

Image Segmentation

?

A common task in image analysis & computer vision

To identify **meaningful** regions

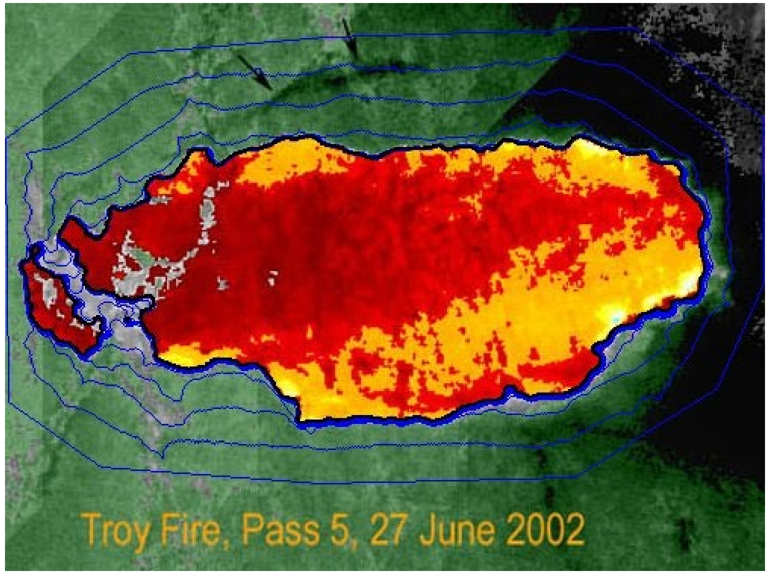
Why do it?

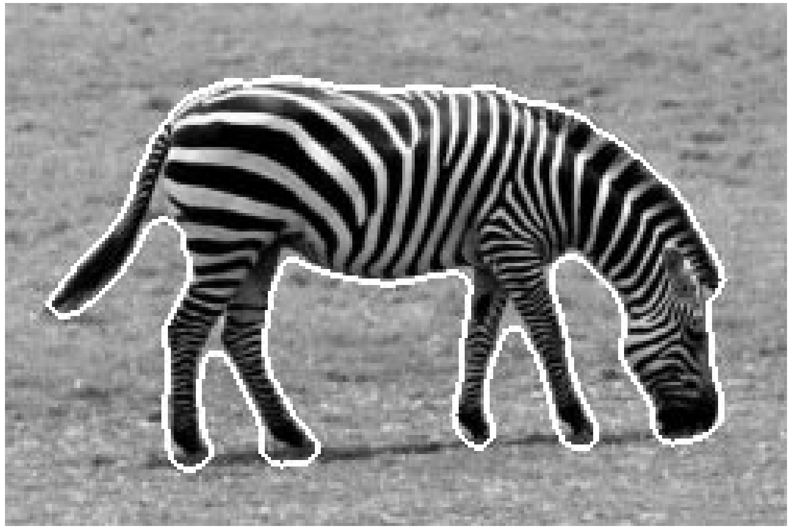
We can partition or group pixels according to local image properties

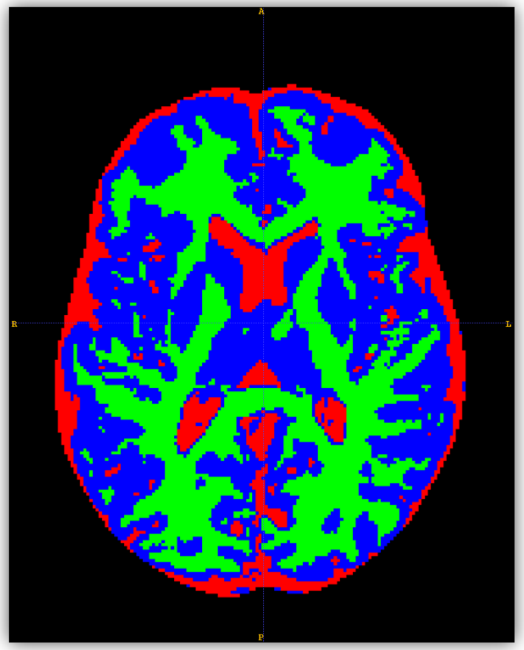
- Intensity or colour from original images, or computed values based on image operators
- Textures or patterns that are unique to each type of region
- Spectral profiles that provide multidimensional image data

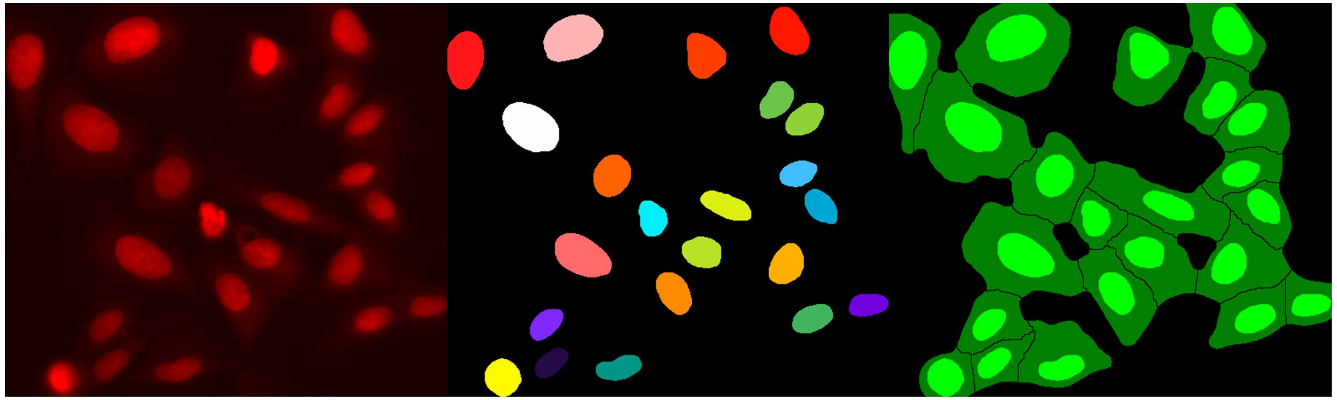
Elaborate systems may use a combination of properties

Applications









Approaches

Many different approaches have been taken to the segmentation problem

Clustering

Seeks groups of similar pixels, with no regard for where they are – views images as uncorrelated data

Region-based

- Focus on finding physically connected sets of pixels
- E.g., region growing, split and merge

Edge-based

- Emphasise the boundaries between regions
- E.g., watersheds

Note

Thresholding + connected components is a form of segmentation, but treats grey/colour and spatial information independently

Region-Based Segmentation

Region-based Segmentation

We want smooth regions in the image

- We still want the pixels in each region to be similar, and those in adjacent regions to be different
- One way to do this is to work with regions rather than pixels

Region Growing

Start with a small 'seed' and expand by adding similar pixels

Split & Merge

- Splitting divides regions that are inconsistent
- Merging combines adjacent regions that are consistent

Region Growing

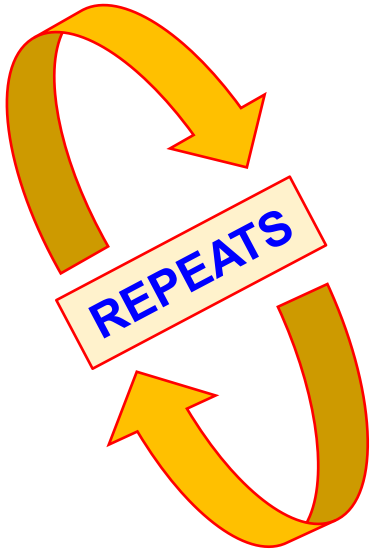
Region growing starts with a small patch of seed pixels

- Compute statistics about the region
- Check neighbours to see if they can be added
- Recompute the statistics

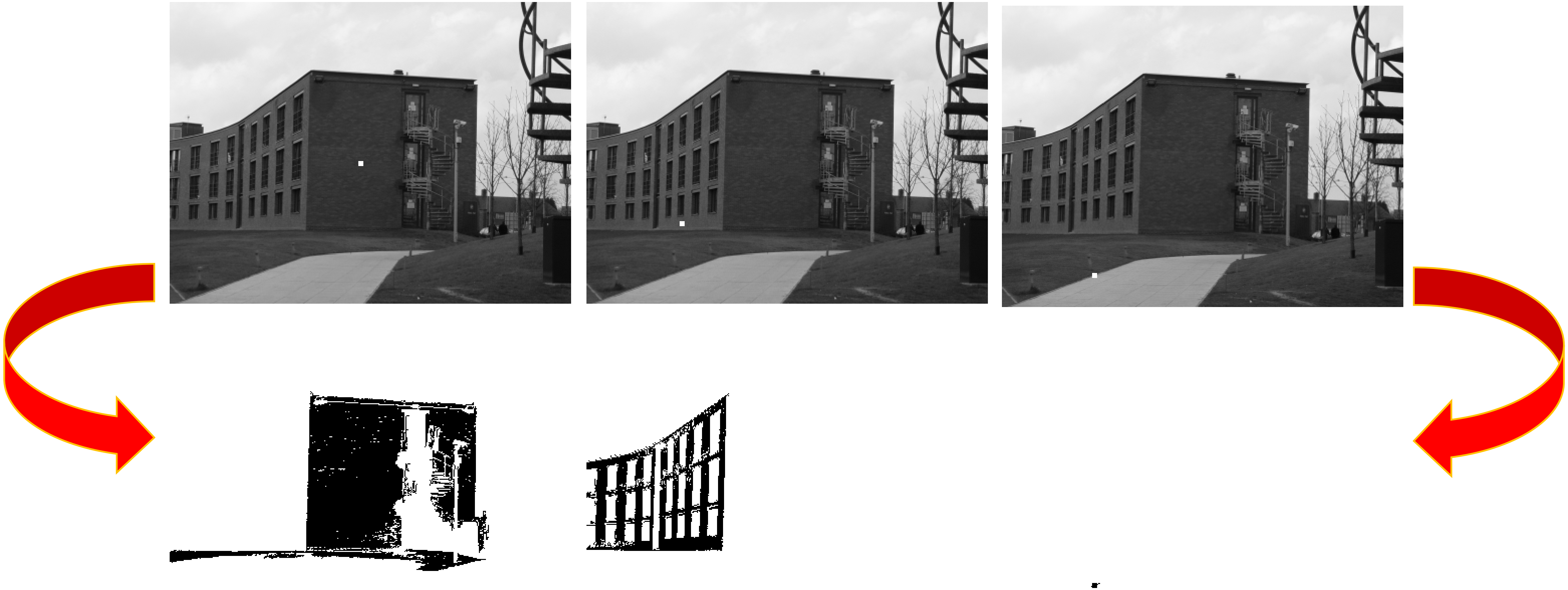
Algorithm

This procedure repeats until the region stops growing

- Simple example: *we compute the mean grey level of pixels in the region*
- Neighbours are added if their grey level is near the average



Region Growing Example



Output 1 Output 2 Output 3

Split and Merge - Split

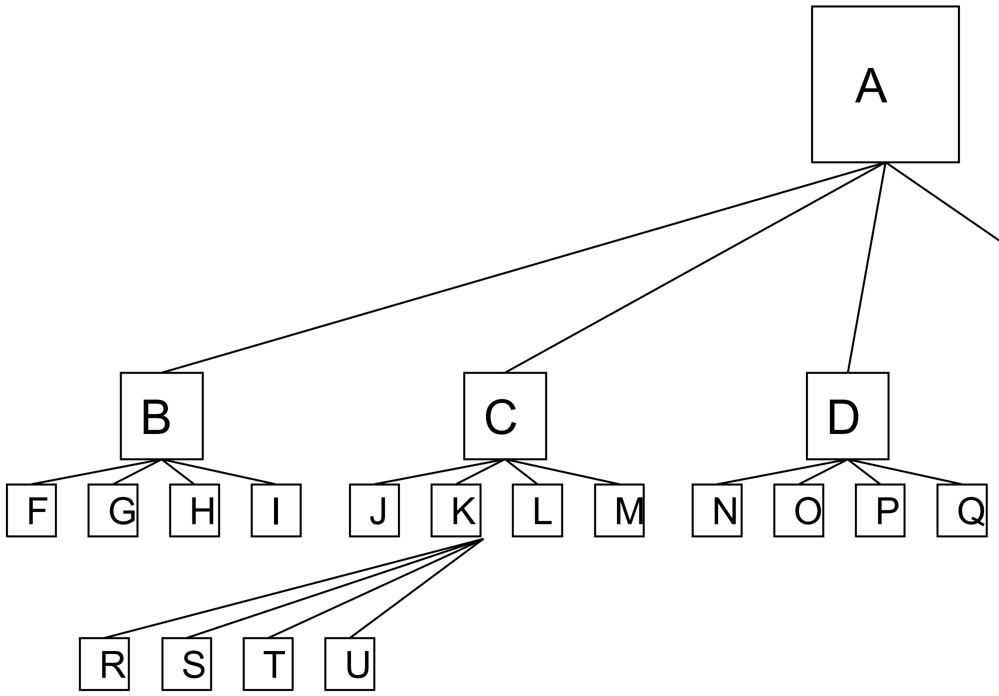
We start by taking the whole image to be one region

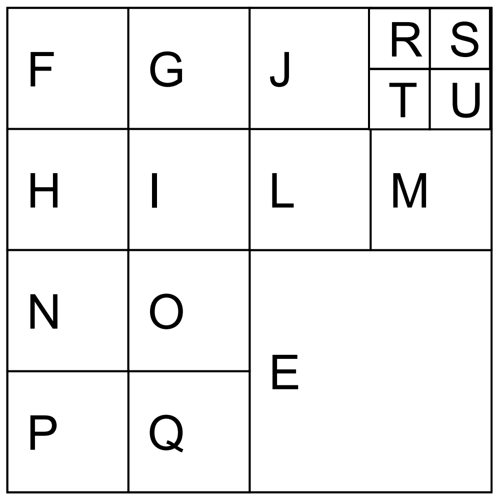
- We compute some measure of internal similarity
- If this indicates there is too much variety, we divide the region
- Repeat until no more splits, or we reach a minimum region size

Some details are needed

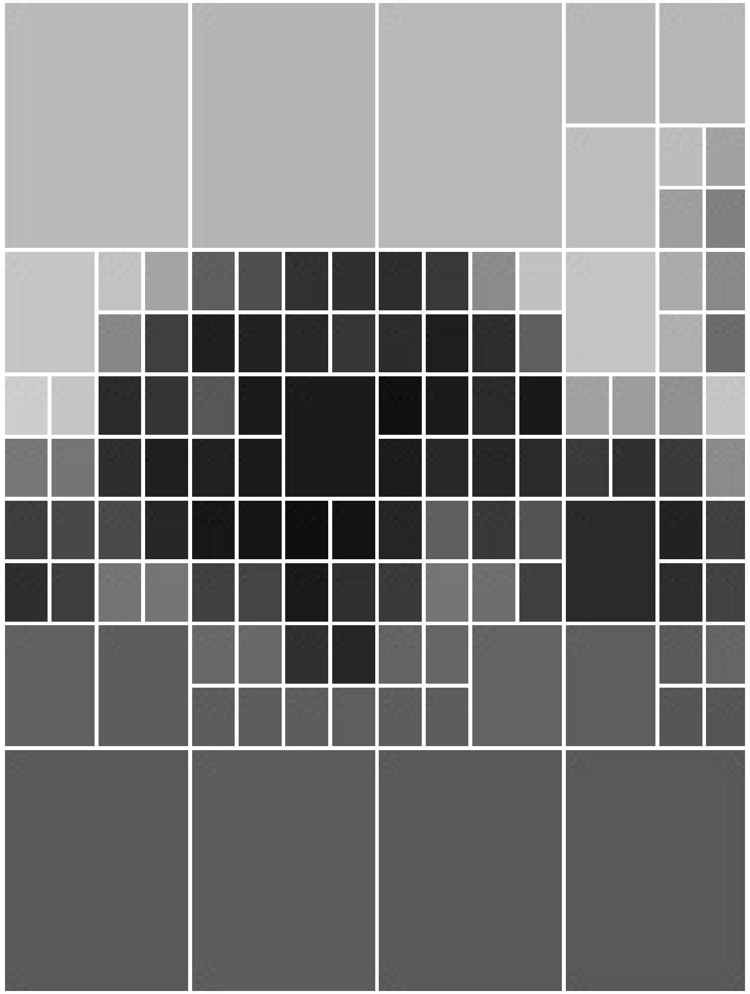
- How to we measure similarity? – standard deviation are commonly used
- How do we determine whether to split or not? – thresholding is easy
- How do we split regions? – *quadtrees* are a common method

Quadtrees





Example - **Splitting**



We'll use the tree image again

- Splitting based on intensity (*could use something else*)
- Splitting based on standard deviation, with a threshold of 25
- Split using quadtree with a maximum of 5 level

Split and Merge - Merge

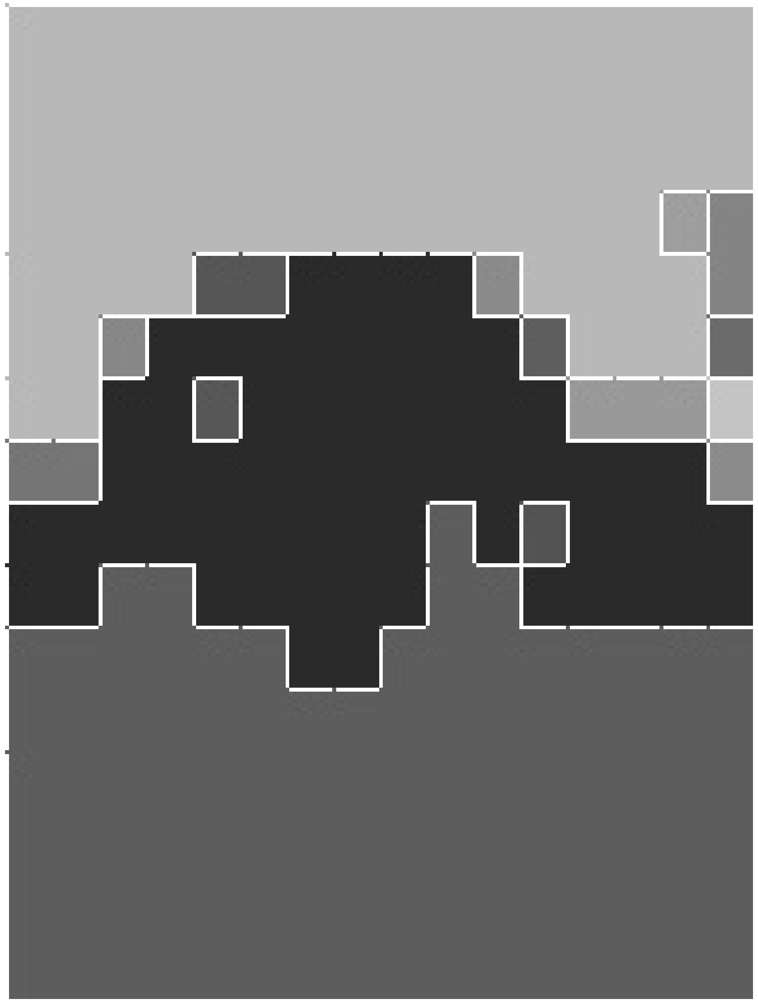
Splitting give us…

- Regions that are small, consistent, or both
- Rather too many regions, as adjacent ones may be very similar
- We can now combine adjacent regions to make bigger ones

Merging

- We merge two regions if they are adjacent and similar
- Need a measure of similarity – can compare their mean grey level, or use statistical tests
- Repeat the merging until you can do no more

Example - Merging



We consider merging adjacent regions

- Two regions are merged if their mean grey levels differ by less than 25
- This leads to less regularly shaped regions, but they are larger and still consistent

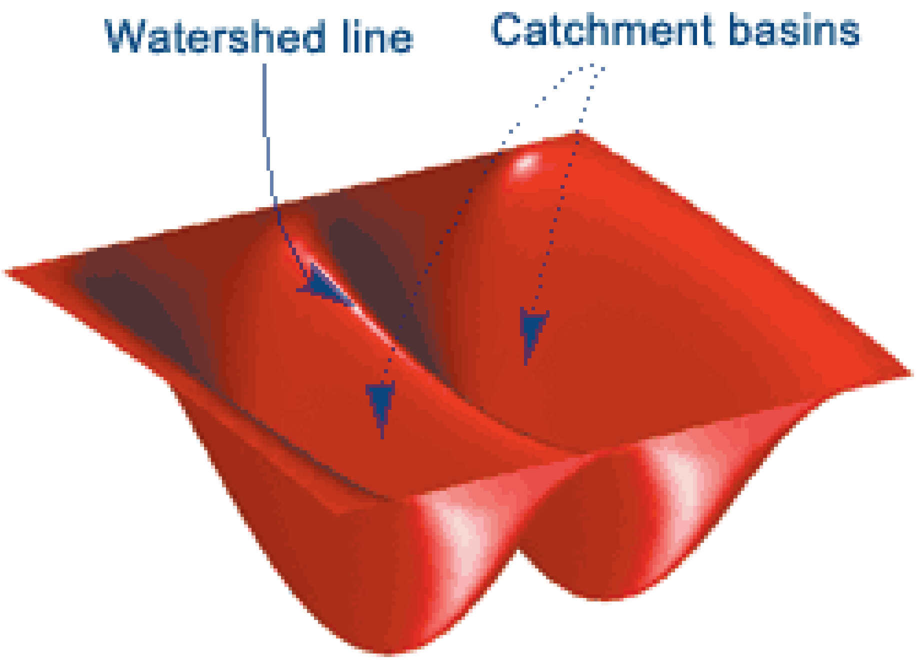
Break

University of Nottingham
UK | CHINA | MALAYSIA

Edge-based Segmentation (*Watersheds*)

Edge-based Segmentation

- Do region-based methods focus too much on regions?
- Edge represent discontinuities in image intensity
- Regions should then be areas without edges, and should be bounded by edges
- One class of edge-based segmentation uses **watersheds**



In geography, a watershed is a ridge which divides rainfall into basins on either side

Catchments in images

We can view the ***gradient*** image as a terrain

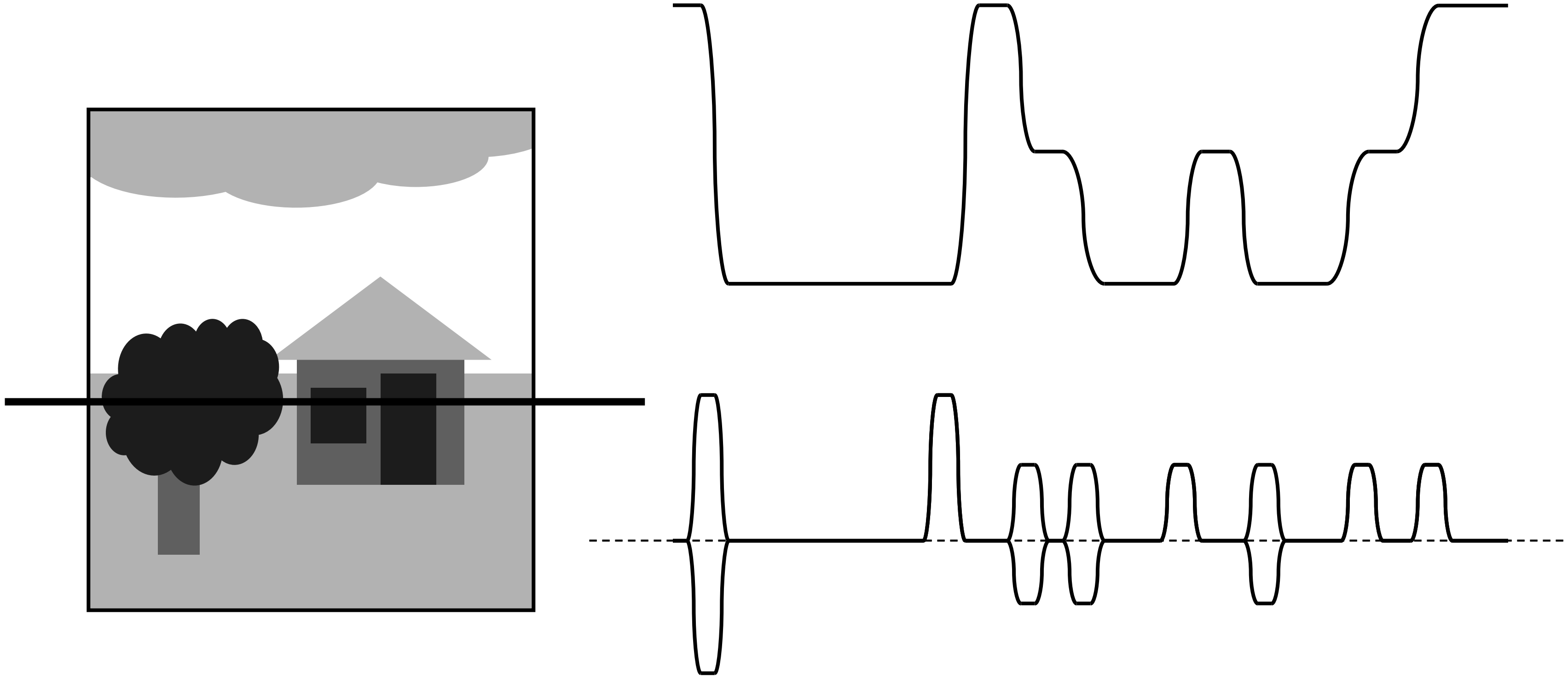
- Areas of high gradient are high points on the terrain
- Catchment basins are regions in the image
- Watersheds are the lines dividing them

We don't have to use the usual intensity gradient

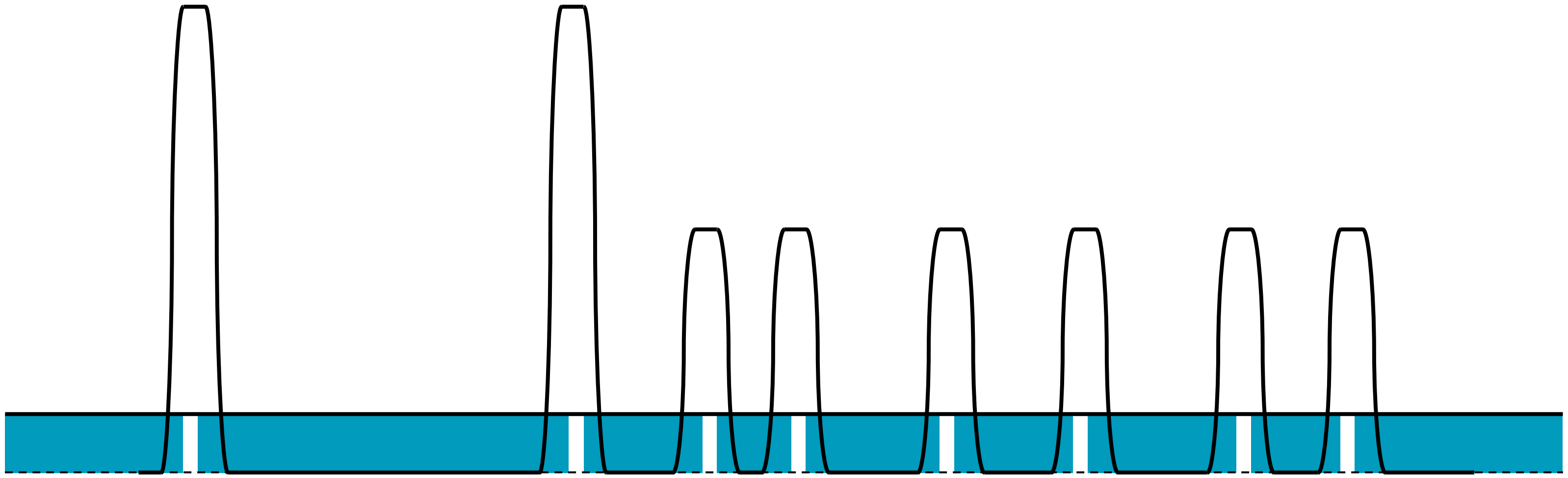
- Gradients can be computed from hue etc., if we want
- Any value that is low within a region and high at boundaries could be used

Using gradients is common, though….

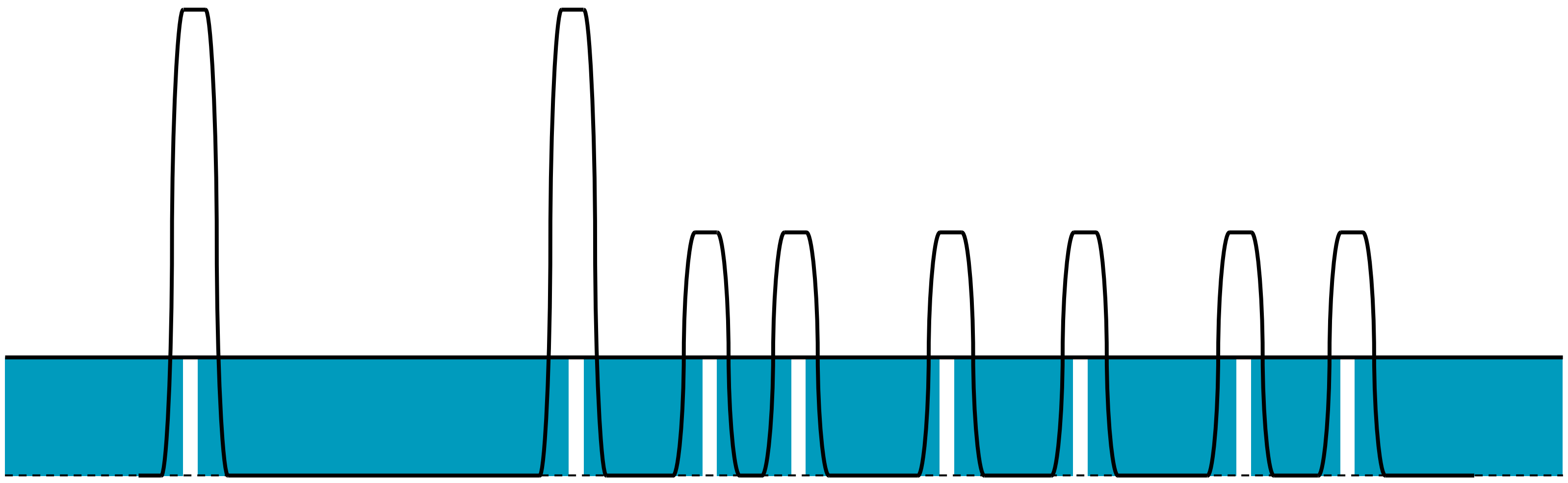
Gradients in Highlight Edges



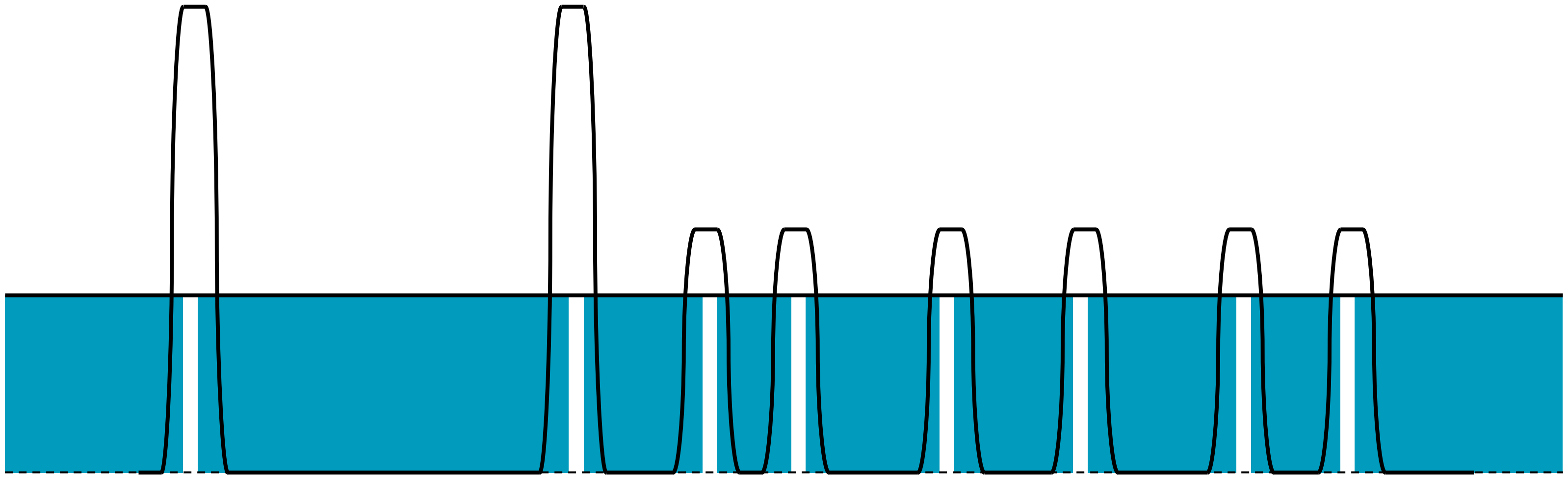
Immersion to Find Regions



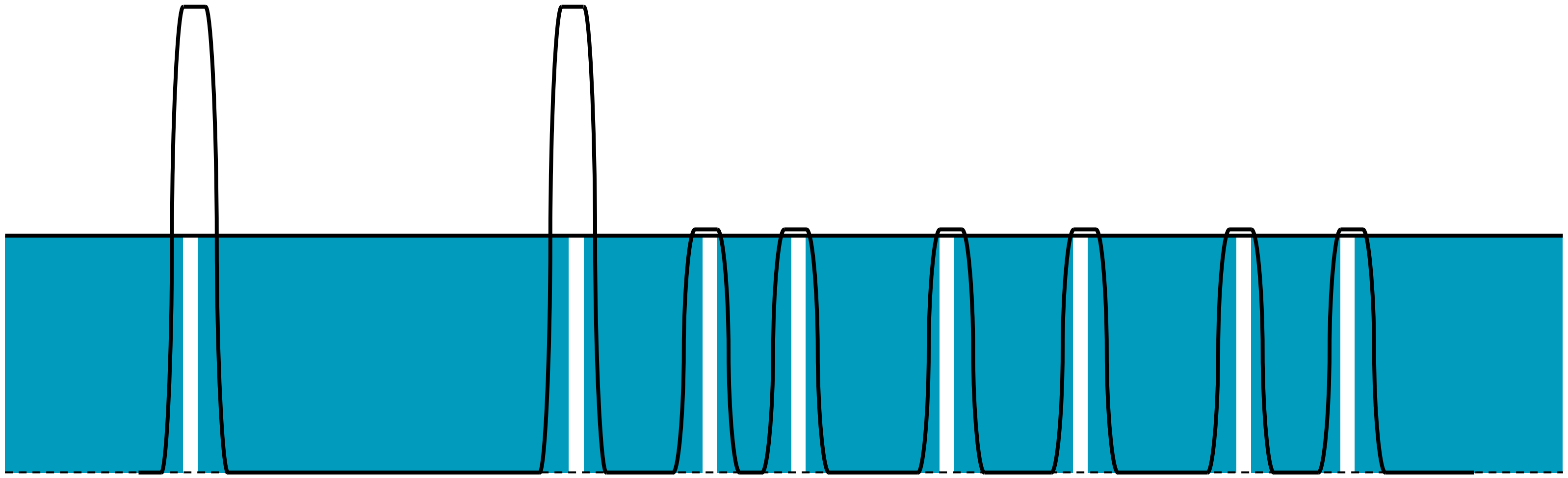
Immersion to Find Regions



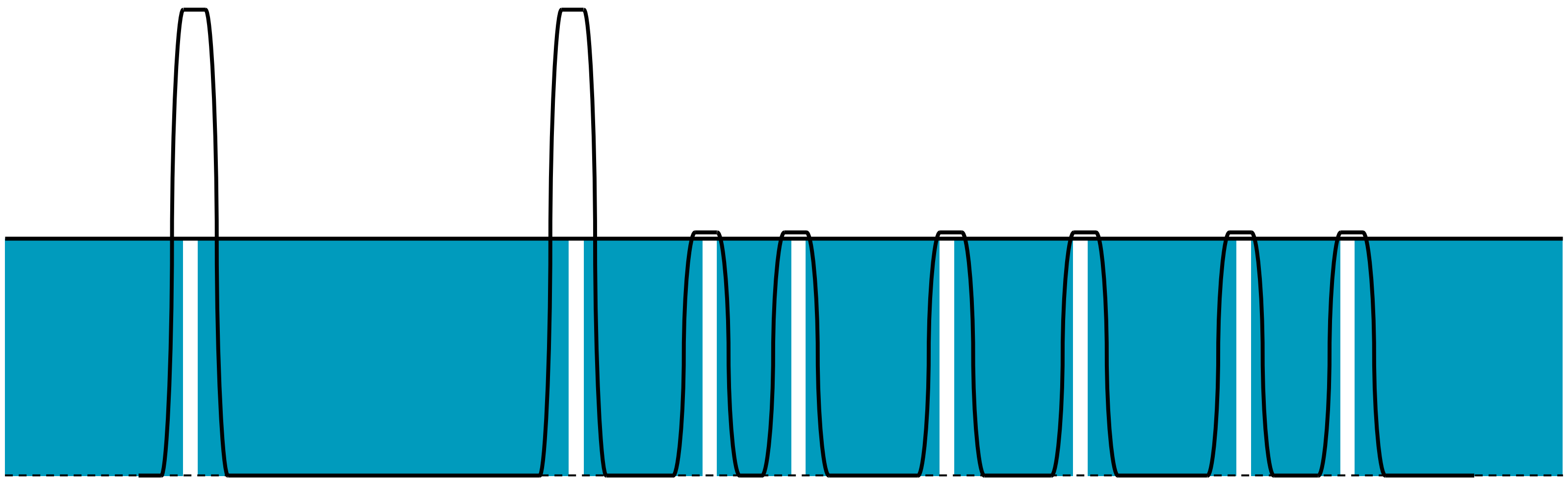
Immersion to Find Regions



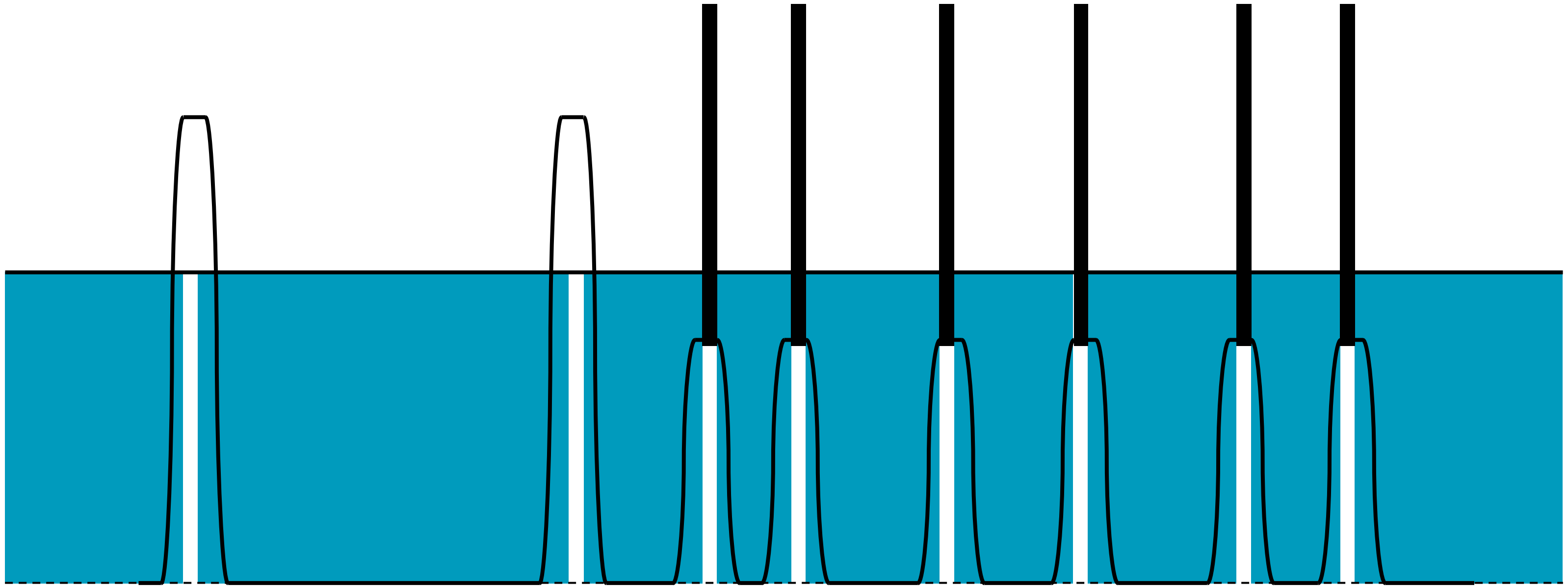
Immersion to Find Regions



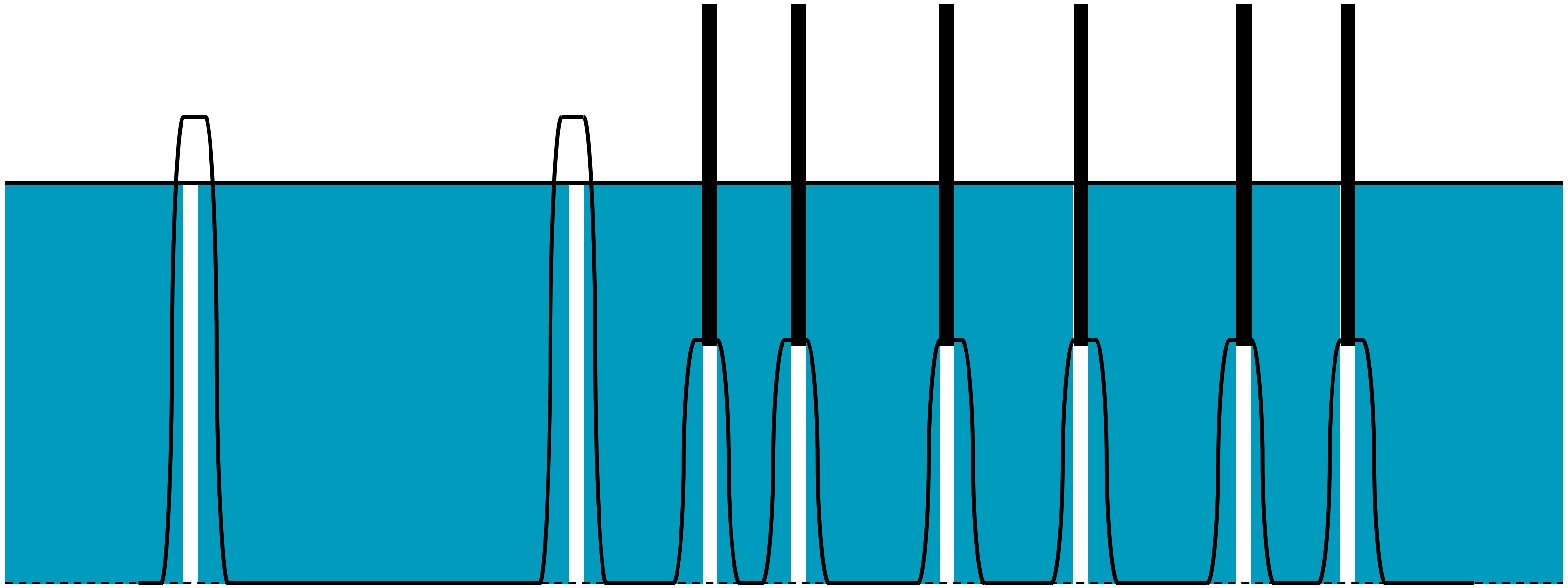
Immersion to Find Regions



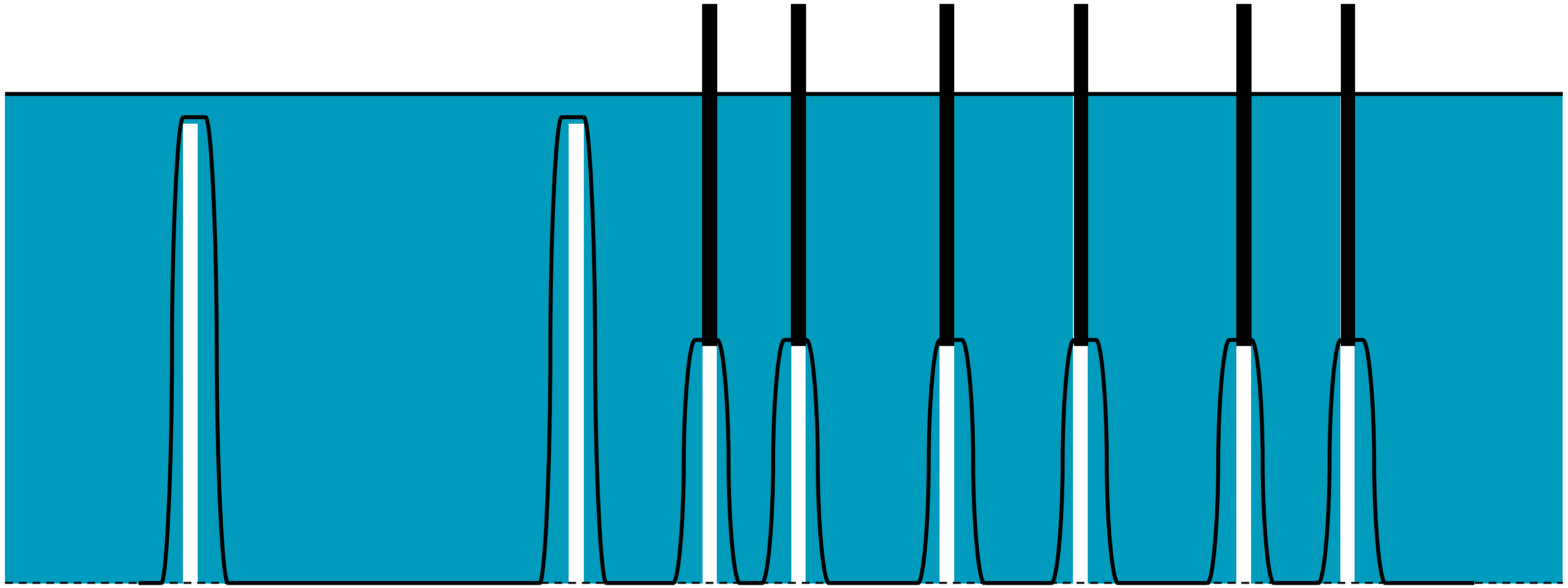
Immersion to Find Regions



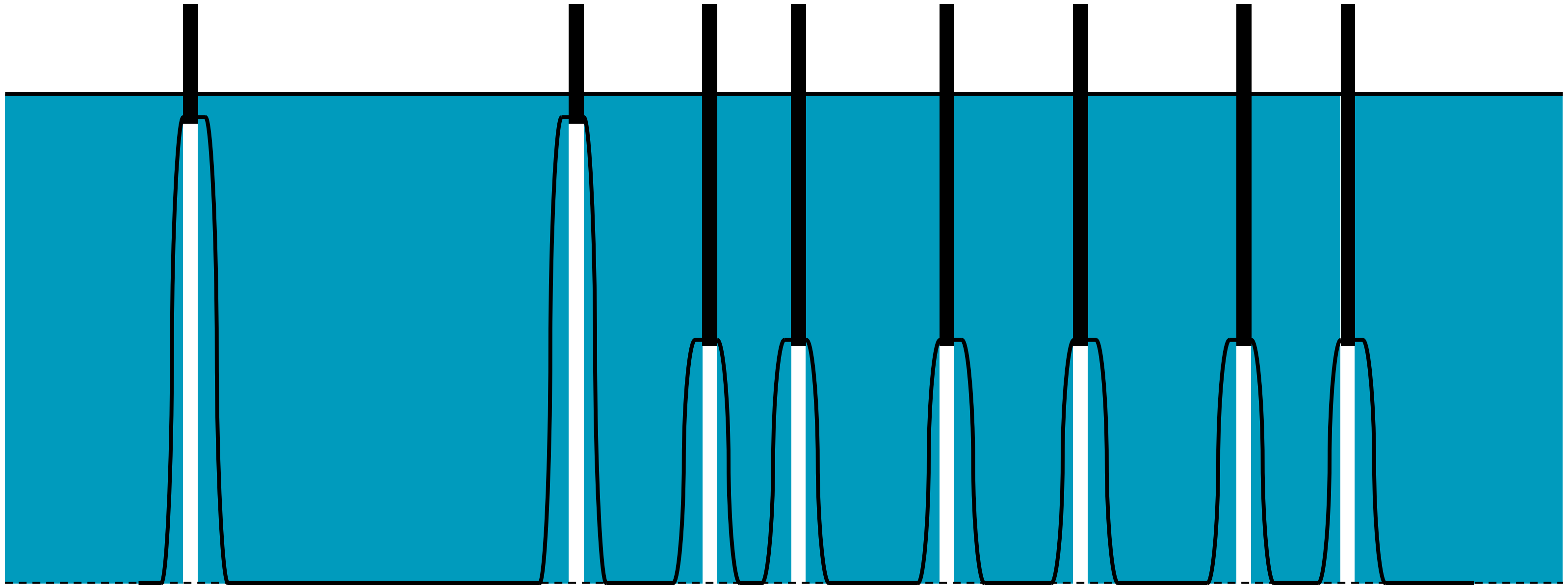
Immersion to Find Regions



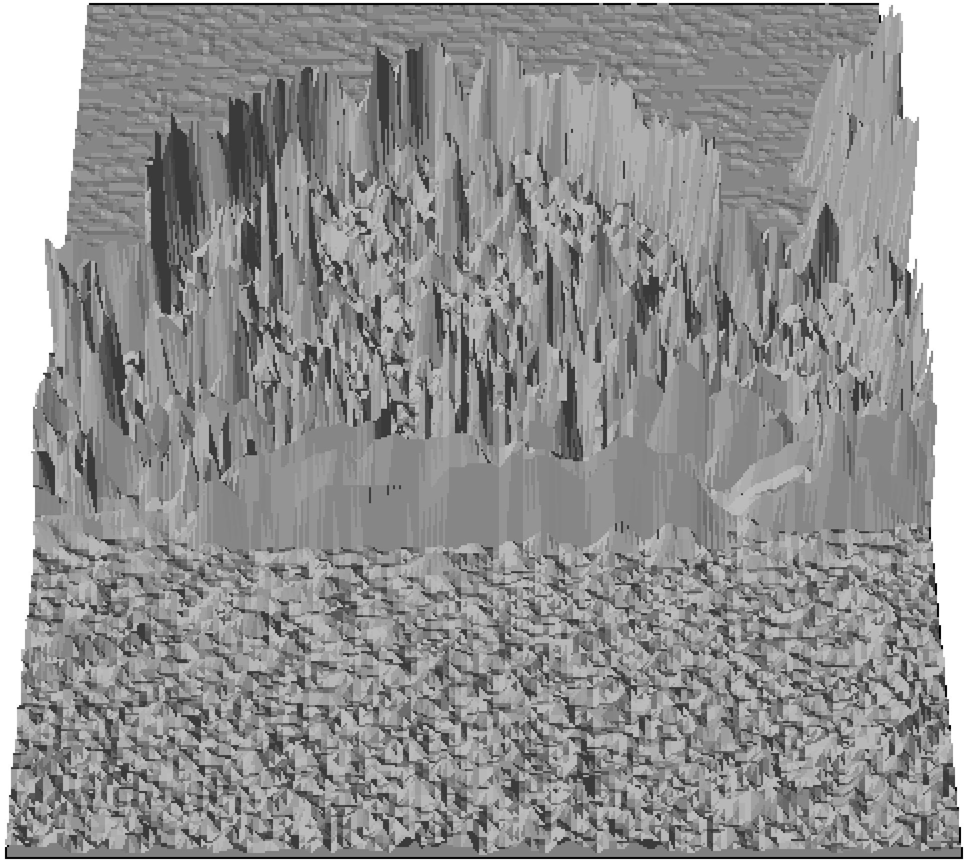
Immersion to Find Regions



Immersion to Find Regions



Watersheds in Images

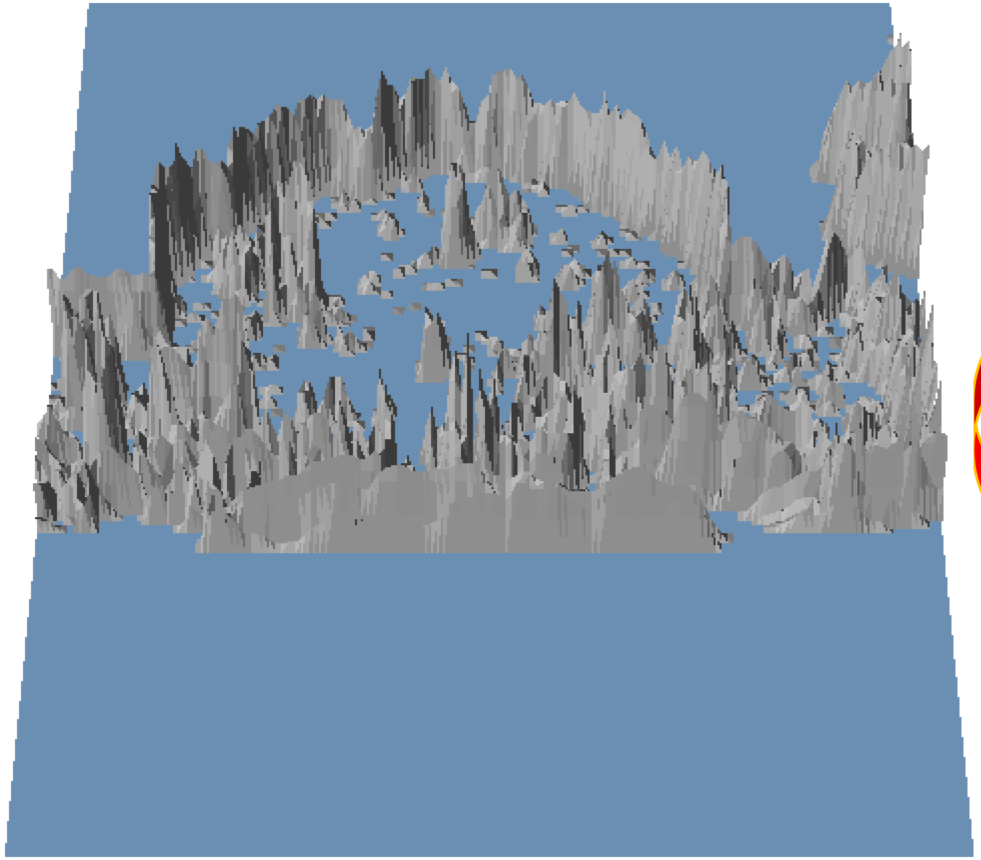


We start by finding images gradients

- Using methods like Sobel operators, we get a value for the gradient magnitude
- This can be viewed as a 3D 'terrain'

$$\sqrt{I_x^2 + I_y^2}$$

Watersheds in Images



We then slowly flood the terrain

- Flat areas of the image become areas of low gradient, so are valleys in the terrain
- Edges in the image have high gradient and so are ridges in the terrain

Watershed Algorithm

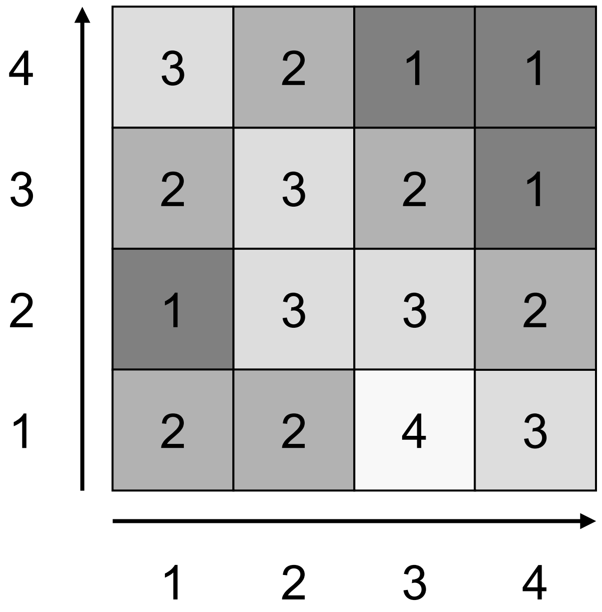
1. Sort the pixels: **low to high**
2. For each pixels

- If it's neighbours are all unlabelled, give it a new label
- If it has neighbours with a single label, it gets that label
- If it has neighbours with two or more labels, it is a watershed

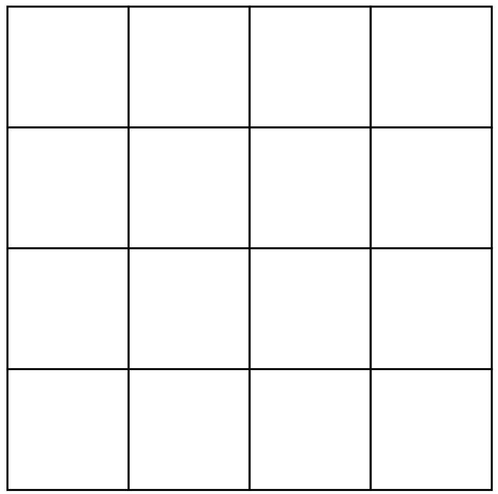
This is a very basic version

- It has certain problems in that it can give 'thick' watersheds rather than fine lines
- It is sensitive to noise and so can generate lots of small regions
- It does show the basic plan, though

An Example



Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



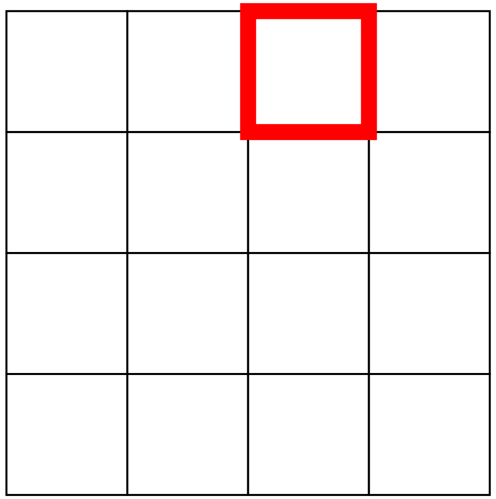
Labels

An Example

	1	2	3	4
4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3

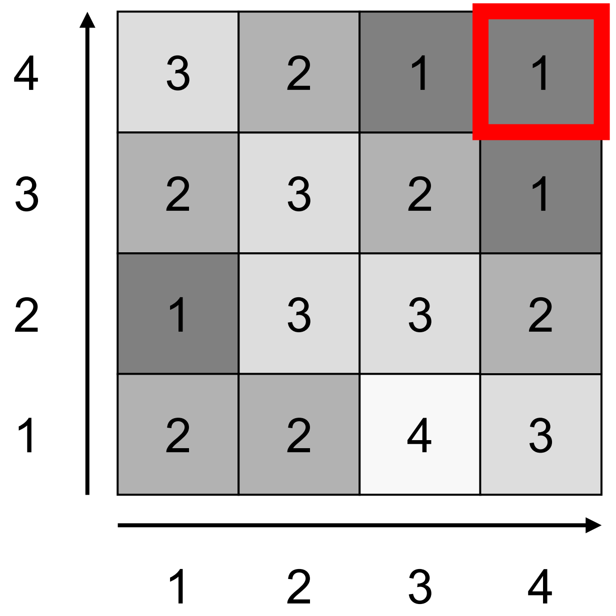
Sorted list:

(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

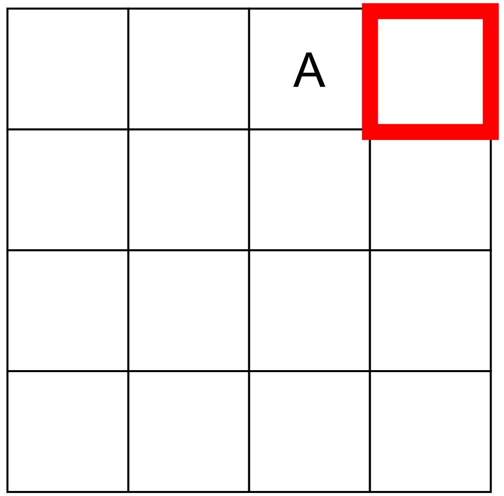


Labels

An Example

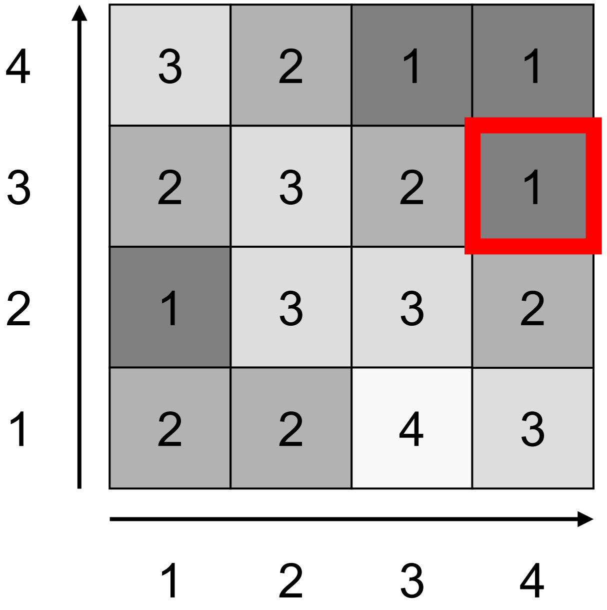


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

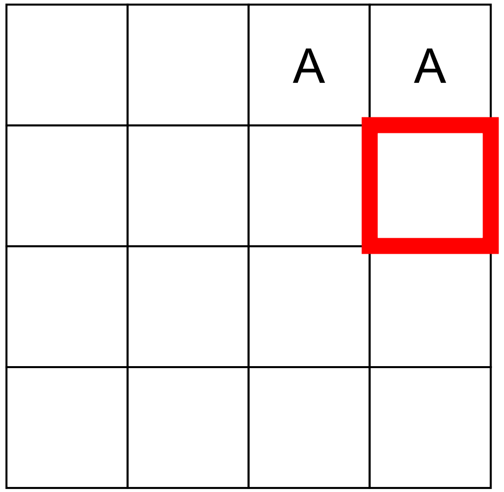


Labels

An Example

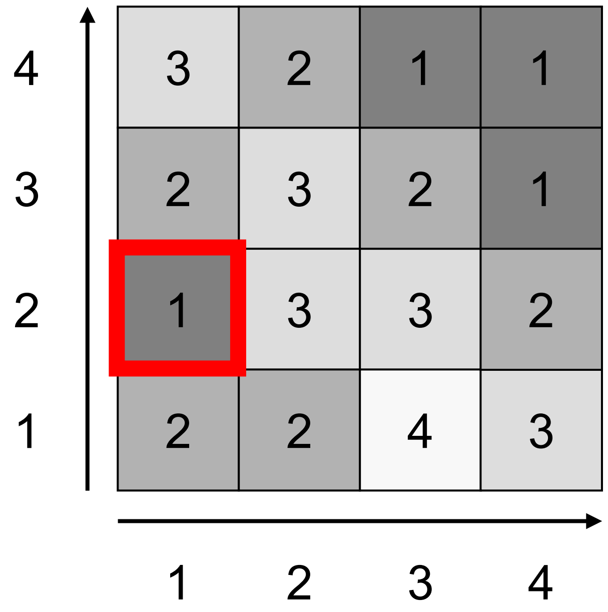


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

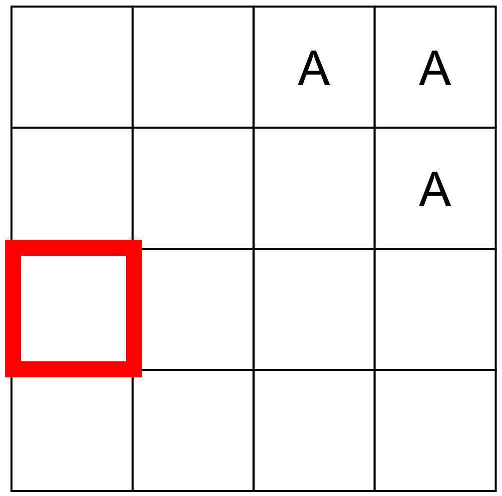


Labels

An Example

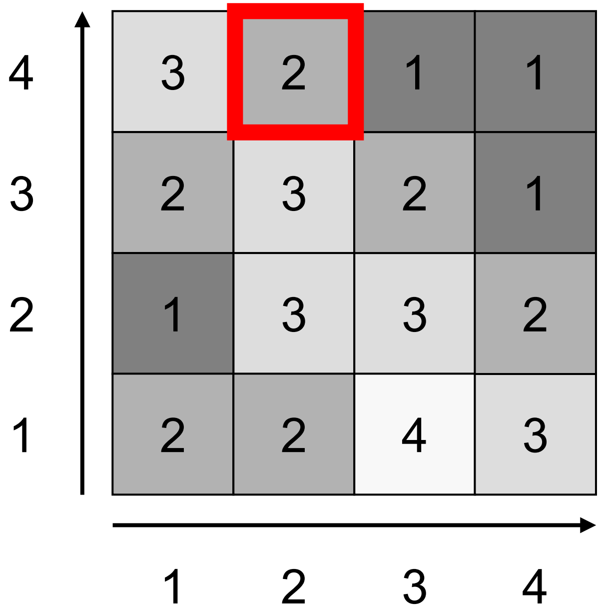


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

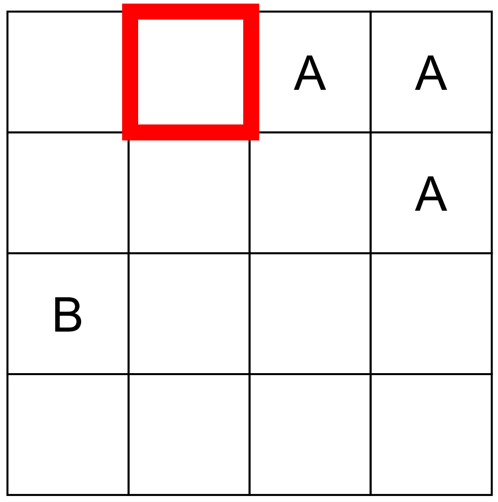


Labels

An Example

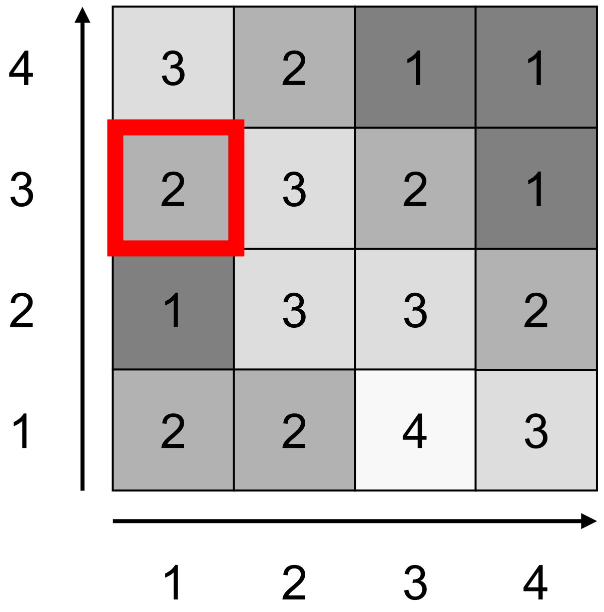


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

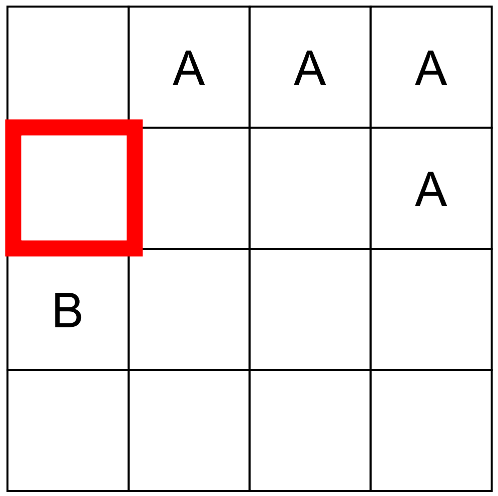


Labels

An Example

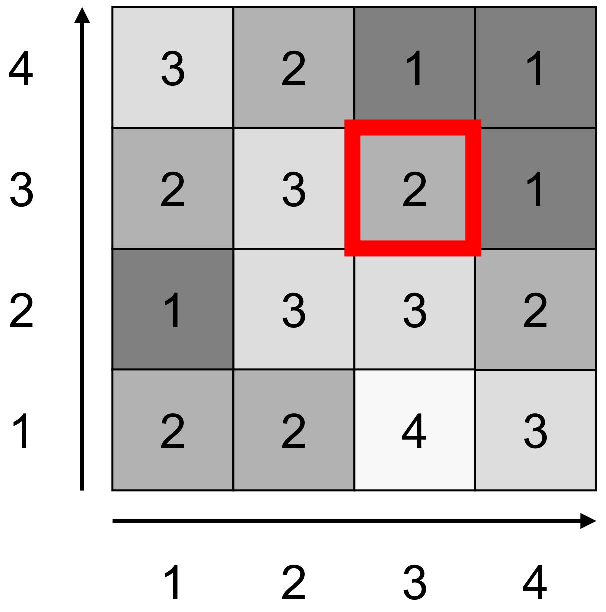


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

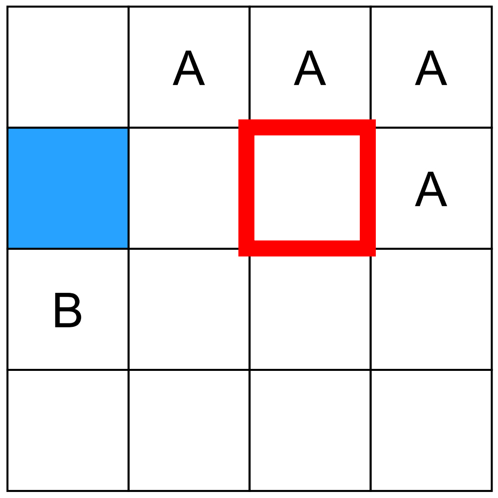


Labels

An Example

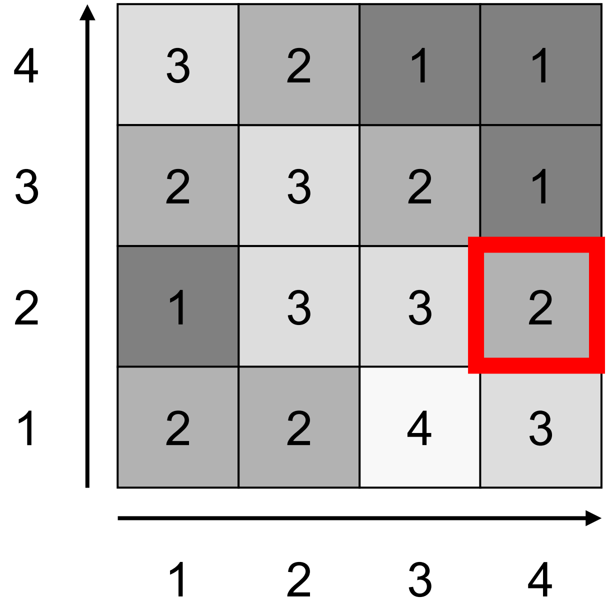


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



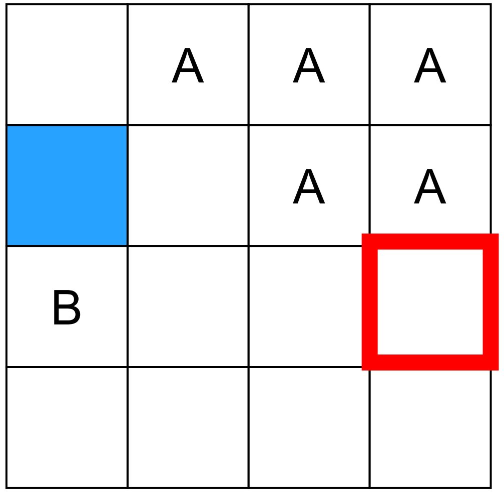
Labels

An Example



Sorted list:

(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



Labels

An Example

4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3
	1	2	3	4

Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

	A	A	A
		A	A
B			A

Labels

An Example

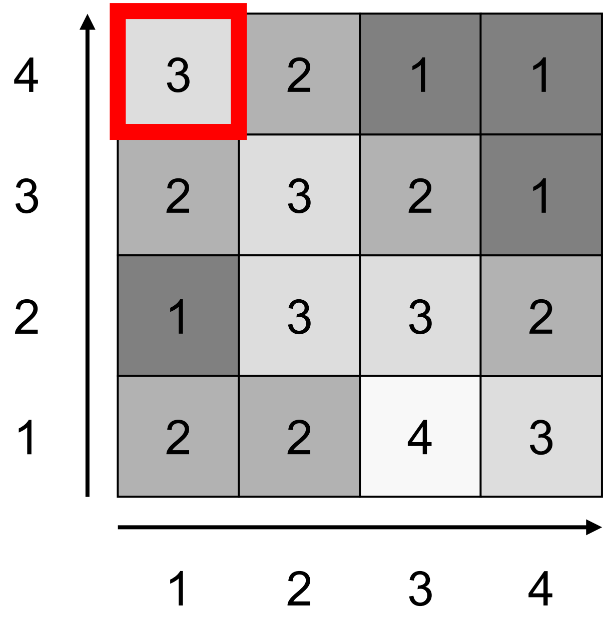
4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3
	1	2	3	4

Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

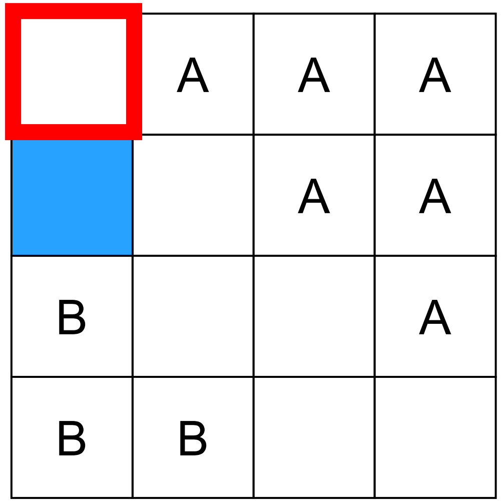
	A	A	A
		A	A
B			A
B			

Labels

An Example

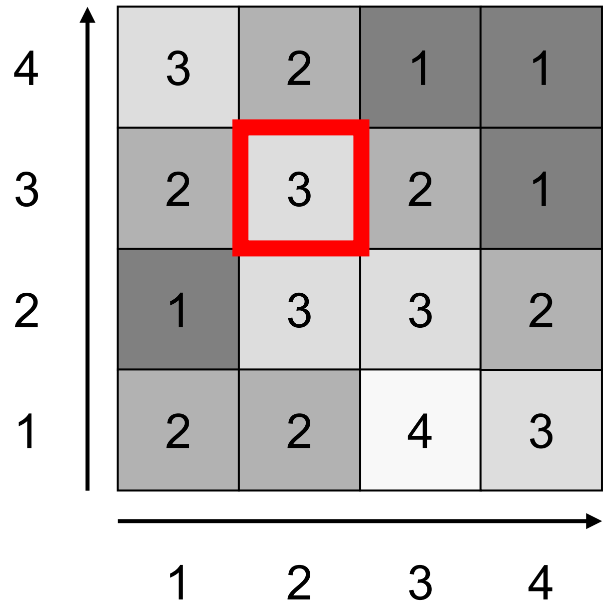


Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

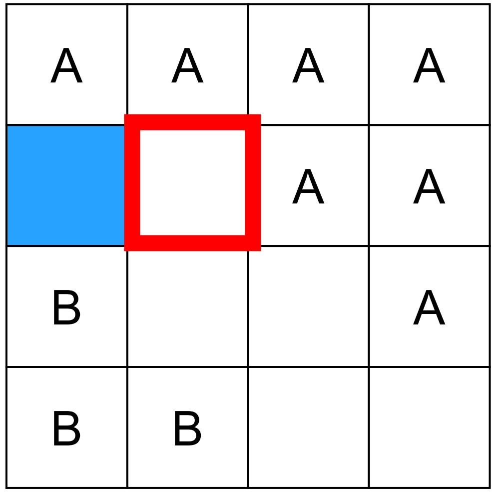


Labels

An Example



Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



Labels

An Example

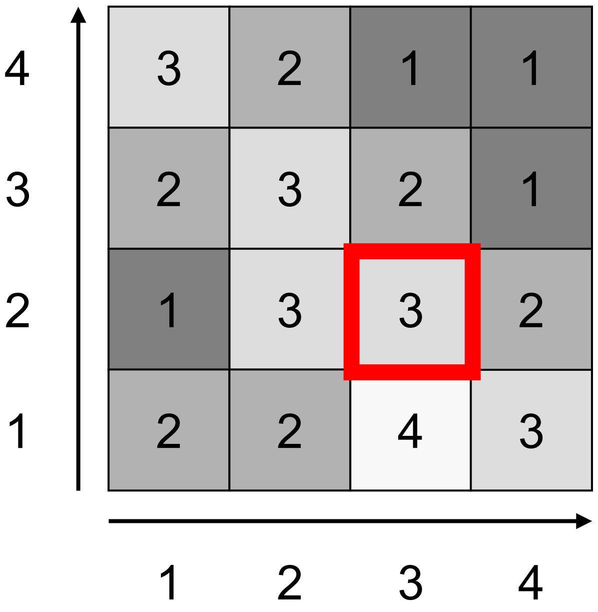
	1	2	3	4
4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3

Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

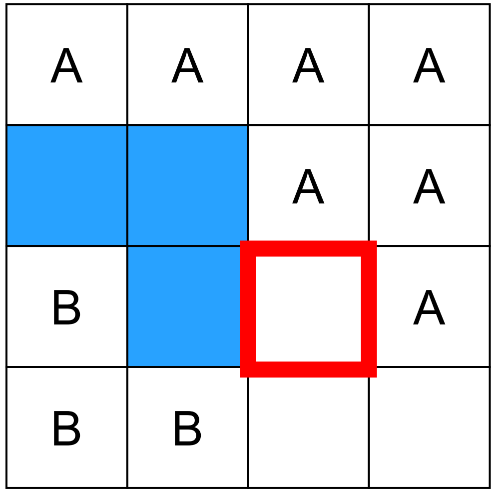
A	A	A	A
		A	A
B			A
B	B		

Labels

An Example



Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



Labels

An Example

4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3
	1	2	3	4

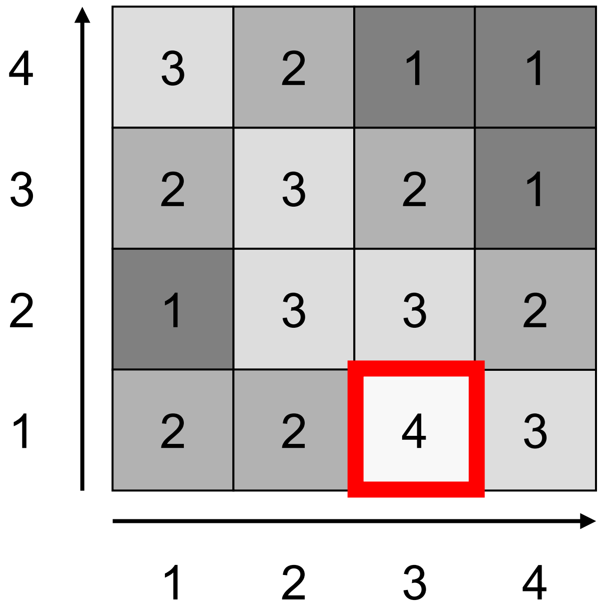
Sorted list:

(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

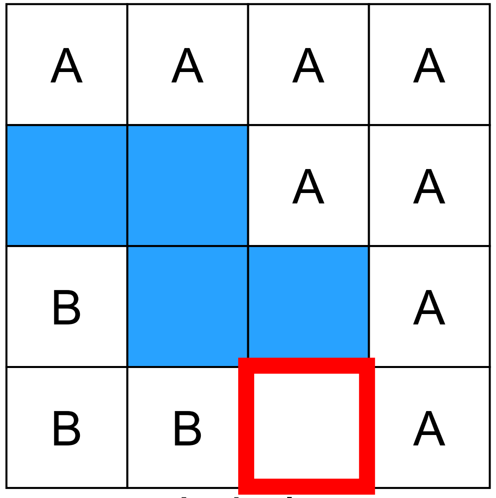
A	A	A	A
		A	A
B			A
B	B		

Labels

An Example



Sorted list:
(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4



Labels

An Example

4	3	2	1	1
3	2	3	2	1
2	1	3	3	2
1	2	2	4	3
	1	2	3	4

Sorted list:

(3,4) = 1
(4,4) = 1
(4,3) = 1
(1,2) = 1
(2,4) = 2
(1,3) = 2
(3,3) = 2
(4,2) = 2
(1,1) = 2
(2,1) = 2
(1,4) = 3
(2,3) = 3
(2,2) = 3
(3,2) = 3
(4,1) = 3
(3,1) = 4

A	A	A	A
		A	A
B			A
B	B		A

Labels

Computing Watersheds

Watershed based segmentations can be very efficient

- It is possible to implement it in *O(n)* time, where *n* is the number of pixels
- Since it takes *O(n)* time to read or write an image, this is as good as it can get in most situations

To implement watersheds we need to sort the pixels

- They need to be sorted from highest to lowest gradient
- Sorting is *O(n* log*(n))*, so how do we get an *O(n)* algorithm?

Sorting in Linear Time

In any situation where we have

- A large number of values, and
- Those values are drawn from a smaller set of possibilities

We can sort in linear time with a bin sort

1. Make a bin (*a list, queue or stack*) for each possible value
2. For each item: put it in the appropriate bin

Bin Sort

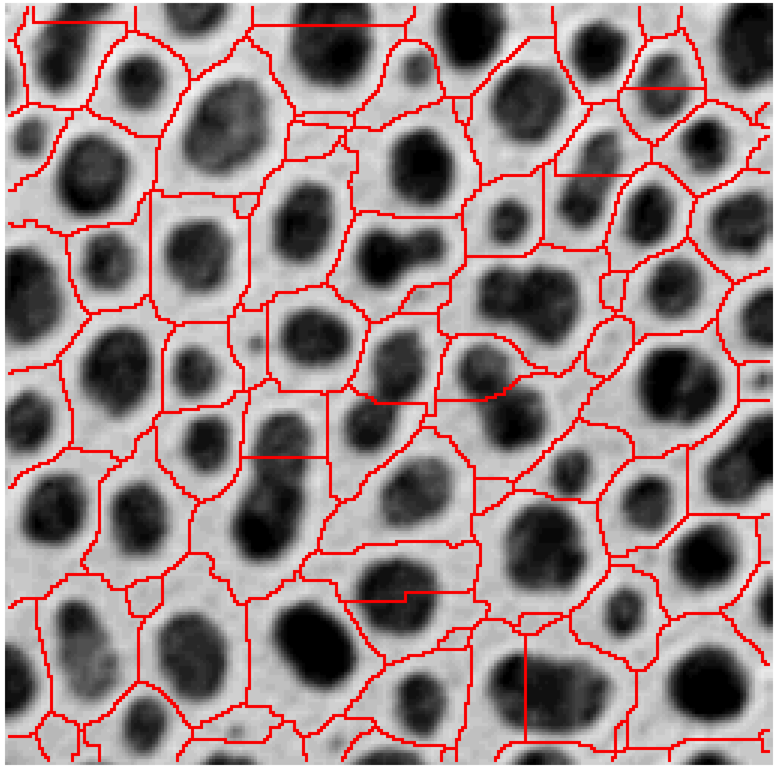
The items are now SORTED!

Example - Watershed

Segmenting the tree image

- The basic algorithm has been modified to avoid the effects of noise
- The gradient has been quantised to remove small variations
- Above a threshold water level, no more new segments are introduced as the water rises

Example - Watershed



- Watersheds can also be applied to some greyscale images directly
- For example, in many medical and biological images the regions are dark or light regions against a light or dark background

Summary

1. What is Segmentation?
2. Region-based Segmentation
3. Edge-based Segmentation

University of Nottingham
UK | CHINA | MALAYSIA

Questions

University of Nottingham

UK | CHINA | MALAYSIA

NEXT:

Interactive Segmentation

COMP2005

Segmentation and Superpixels

An Alternative? Superpixels

- Segmentation has motivated development of some useful techniques, but:
  - ‘segmentation’ is poorly defined
  - trying to achieve meaningful, semantically correct results without knowledge of the application domain is optimistic at best
  - segmentation methods really just divide the image into similar regions
- So let’s accept that and forget
  the semantics…….

Simple Linear Iterative Clustering

- High-quality, compact, nearly uniform superpixels
- Simple, efficient algorithm based on K-means
- Only parameter is number of superpixels required (K)

1. Initialize cluster centers on pixel grid in steps S

- Image has N pixels, you want K superpixels
- Each superpixel is a roughly square area of roughly N/K pixels
- Each superpixel is roughly sqrt(N/K) by sqrt(N/K)
- S = sqrt(N/K)

2. Move centres to the position in a 3x3 window with the <u>smallest</u> intensity (or colour) gradient

- Move centres away from edges, onto flattest area available
- Only a small move, these are still initial positions

SLIC

3. Compare each pixel to all cluster centres within 2S pixels and assign it to the best matching centre

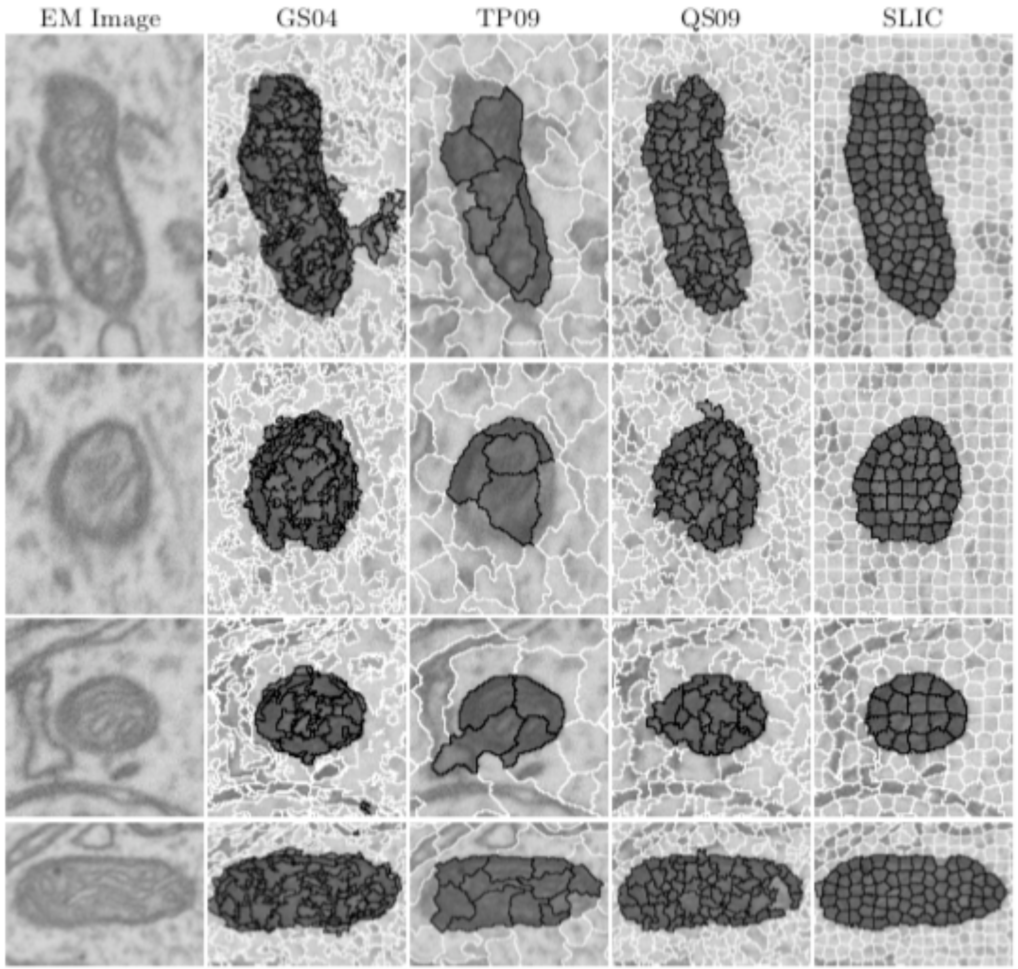
- Best matching = nearby and similar in colour
- Distance measure is sum of colour distance and image plane distance

See the paper on Moodle for details

4. Recompute cluster centres as mean colour and position of the pixels belonging to each cluster

5. Repeat 3 and 4 until total change made to position and colour of centres is below a threshold, or for a fixed number of iterations

SLIC



- Evaluated on
  - similarity of pixels in superpixels vs variation of values between adjacent superpixels
  - proportion of object boundaries marked by a superpixel boundary
- E.g. EM images of brain mitochondria (linked to degenerative diseases)
- Segmentation meets edge detection?

SLIC in Matlab

A = imread('kobi.png’);

[L,N] = superpixels(A,500);

// L is a label image

// N number of superpixels actually produced

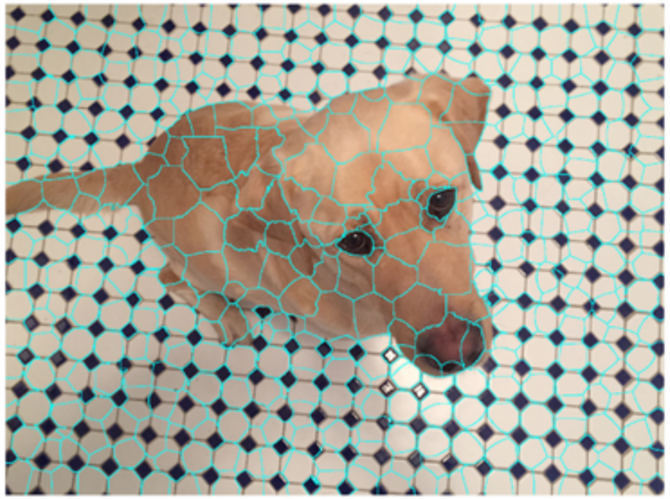
figure

BW = boundarymask(L);

// marks transitions from one label to another

imshow(imoverlay(A,BW,'cyan’),
'InitialMagnification’,67);

//overlays boundary mask on original image
in cyan



https://www.mathworks.com/help/images/ref/superpixels.html