Digital Image Processing

Image segmentation

Erchan Aptoula

Spring 2016-2017

Contents

- What is image segmentation, and why is it so problematic?
- Histogram based methods
- Clustering based methods
- Region growing and splitting based methods
- Morphological segmentation
- Graph partitioning based segmentation
- Partition based tree representations

Segmentation

Segmentation is the process of partitioning a digital image into semantically meaningful regions.

Why? Because, images very often contain redundant things along with the object(s) of interest. We want to separate the interesting from the uninteresting.

More formally, the outcome of an image's segmentation is a **partition**.

Let's assume that E is the definition domain of the image f. A partition of f is a family of sets $P = \{P_i\}$ such that:

- $\bigcup P_i = E$
- $\forall\, i \neq j, P_i \cap P_i = \emptyset$
- $\forall\, i, H(P_i) = \text{true}$
- $\forall\, i \neq j, P_i$ adjacent to $P_j, H(P_i \cup P_j) = \text{false}$

where H represents a homogeneity predicate.

Ok, so we just need to determine which pixel sets are homogeneous w.r.t. the rest.

Segmentation is difficult. Because:

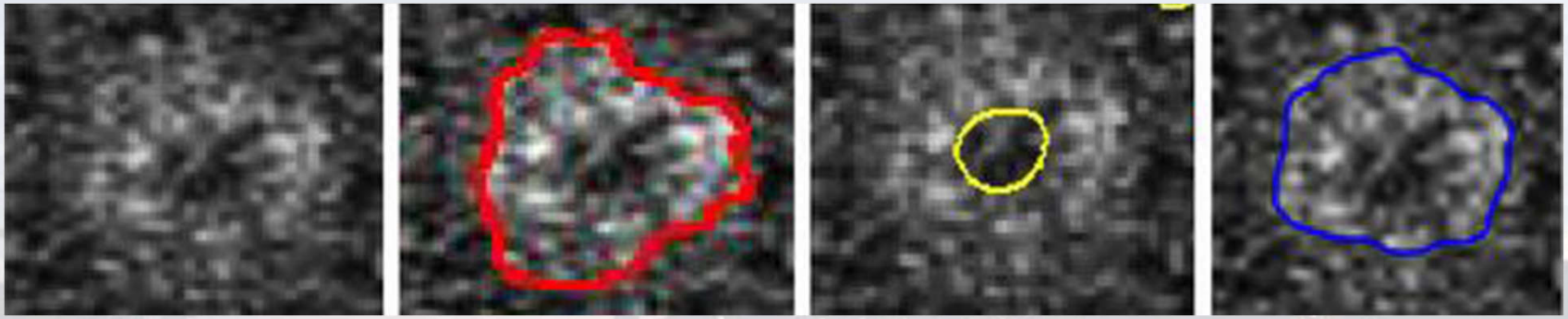
- What's "interesting" and "uninteresting" depends on the application.
- The concept of "semantically meaningful" is subjective.

Which is why segmentation is known as an **ill-posed problem**.

Metin: sky and hills (2 regions)
Ali: sky, hills, trees (3 regions)
Feyyaz: sky, clouds, hills, river, buildings, trees, bridge (7 regions)

Who is right?

Or the experts might agree on the number of regions, but disagree on the positions of the region border



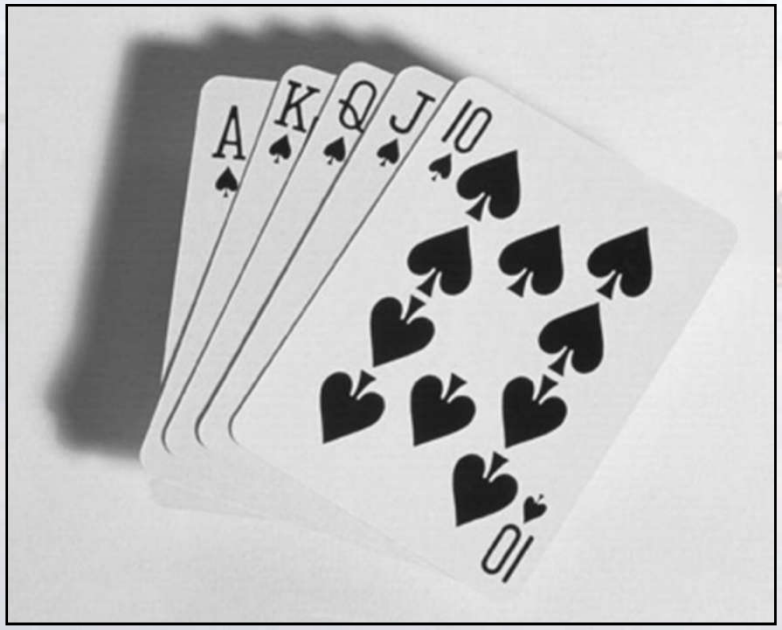
Adapted from Chen et al, QIMS, 2015

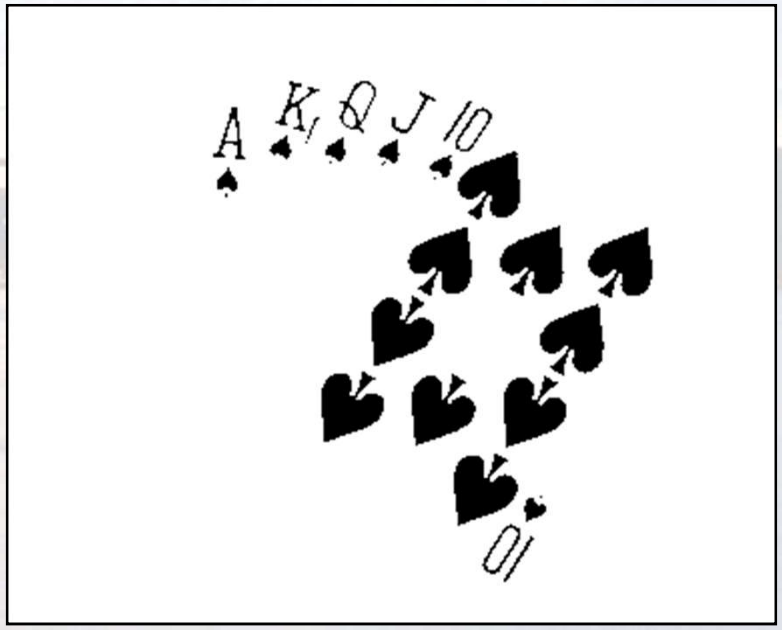
As a result, there is no single segmentation method that works with all applications and all images.

The simplest segmentation method is **thresholding**:

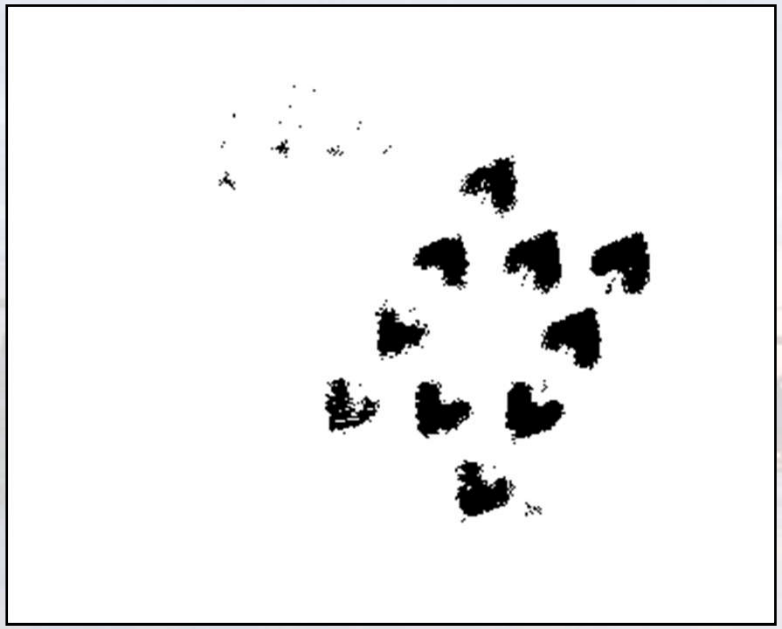
Given a threshold T, pixels are set to either white or black depending on whether they are above or below T.

$$g(x,y) = \begin{cases} 255, \text{if } f(x,y) \geq T \\ 0, \text{otherwise} \end{cases}$$

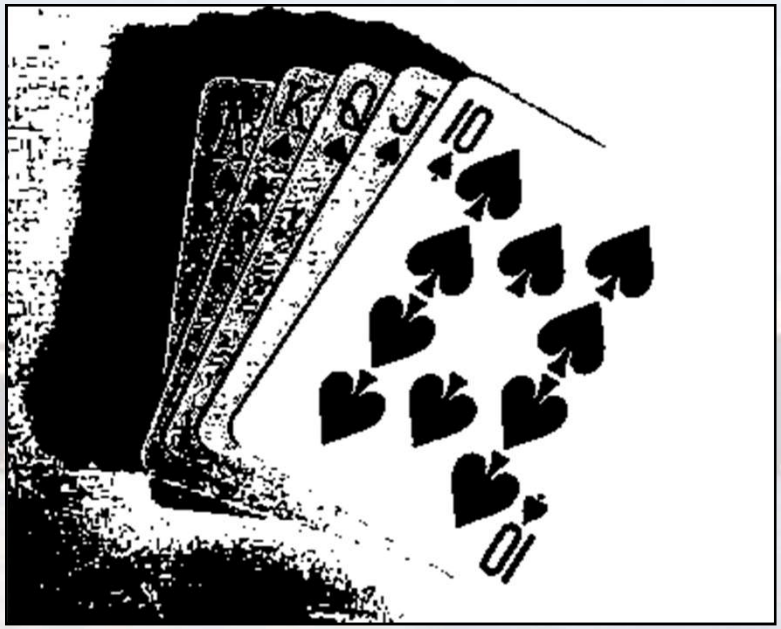




Thresholding is very **fast**, but depends critically on the choice of the threshold.



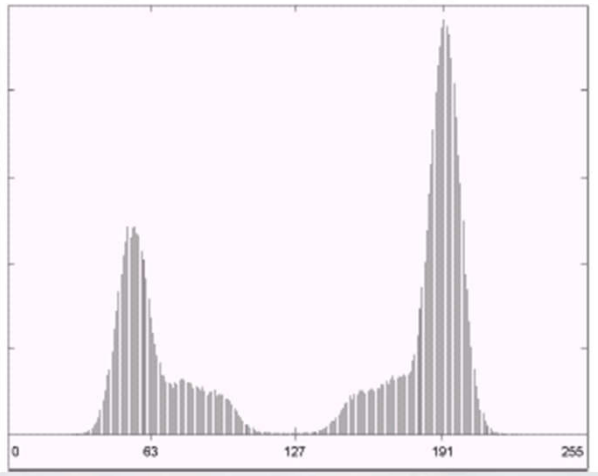
Threshold Too Low



Threshold Too High

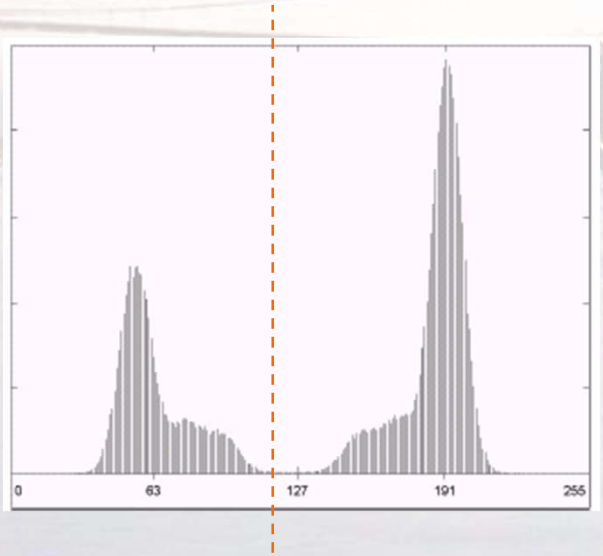
There are MANY algorithms for threshold selection. Many of them rely on the histogram of the image under consideration.

If the image has been acquired under controlled conditions, then it is not rare for it to contain the object(s) of interest on a semi-uniform background, leading very often to a **bi-modal** histogram.



If the histogram is bi-modal, then you could do the following:

1) Start with an initial threshold estimation T (usually the image's mean value)
2) Threshold the image with T and determine the pixels of the two sets A and B
3) Compute their mean values: μ_A and μ_B
4) Compute the new threshold value $T = (\mu_A + \mu_B)/2$
5) Repeat steps 2, 3, 4 until the difference between successive T values drops below a predefined level ε



Alternatively you can also use the famous **Otsu's method** (1979):

Idea: assuming a bi-modal histogram the **optimal** threshold is the one minimizing within-class variance. So it's just a question of trying every threshold until you find the optimal one.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Normalized histogram

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \qquad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

With a few tricks it can be proven that this is the same thing as maximizing inter-class variance:

$$\sigma^2{}_B(t) = \sigma^2(t) - \sigma^2{}_W(t)$$

$$\sigma^2{}_B(t) = q_1(t)(\mu(t) - \mu_1(t))^2 + q_2(t)(\mu(t) - \mu_2(t))^2$$

or simpler

$$\boxed{\sigma^2{}_B(t) = q_1(t)q_2(t)(\mu_1(t) - \mu_2(t))^2}$$

Between-class variance is both **computationally simpler** and can be obtained by sweeping across the histogram **recursively**:

$$q_1(t+1) = q_1(t) + P(t+1)$$

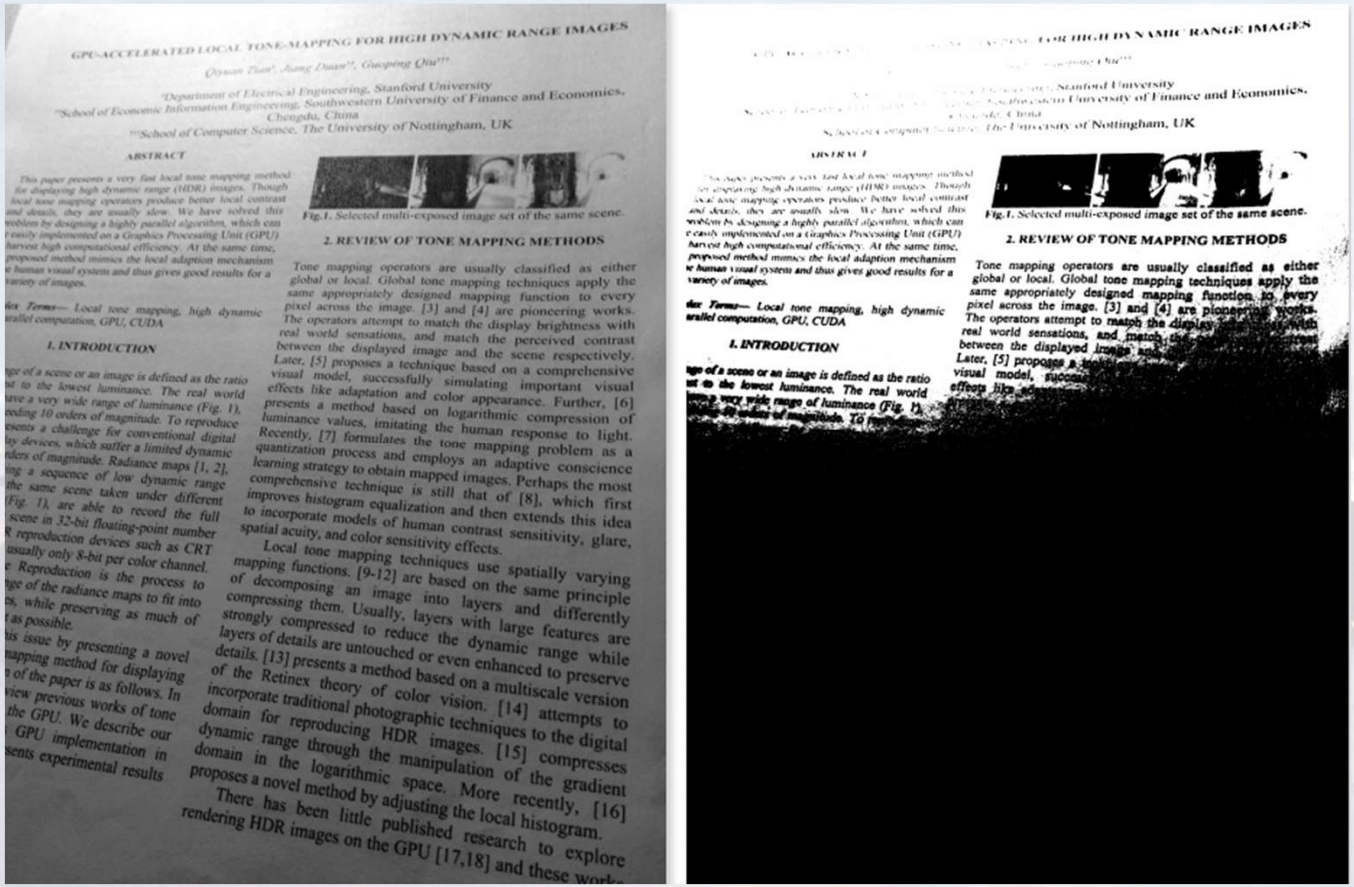
$$\mu_1(t+1) = \frac{q_1(t)\mu_1(t) + (t+1)P(t+1)}{q_1(t+1)}$$

$$\mu_2(t+1) = \frac{\mu - q_1(t+1)\mu_1(t+1)}{1 - q_1(t+1)}$$

Segmentation

Thresholding however is also very sensitive to illumination conditions

Original



Otsu

Even if your histogram is bimodal, uneven illumination can mess up everything. Which filter can deal with it?

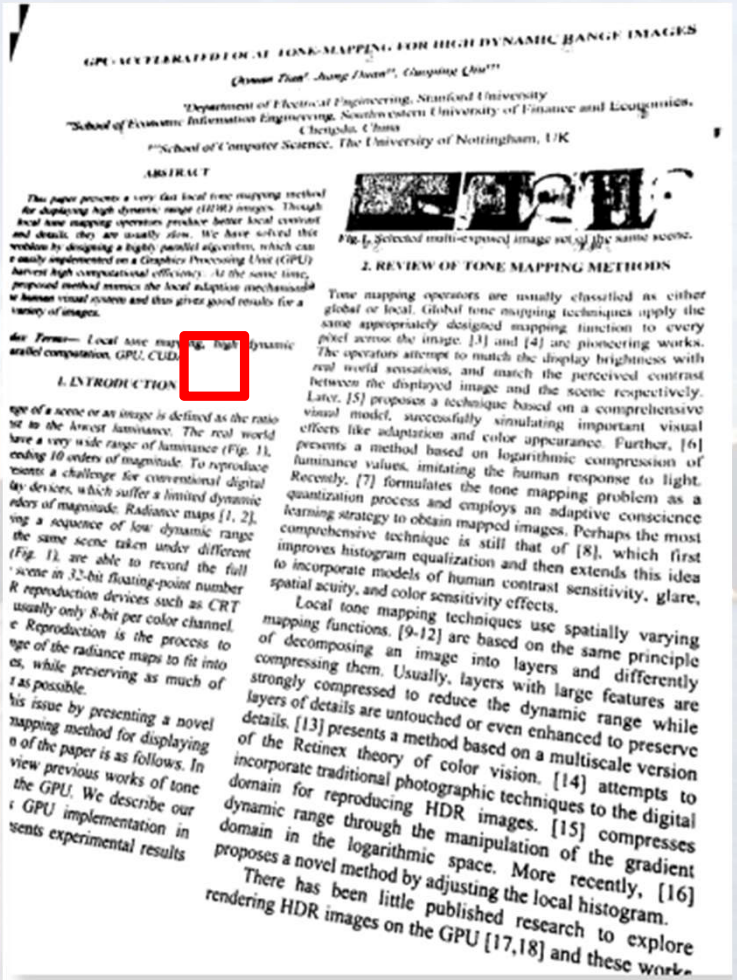
Adapted from Bernd Girod, Stanford U.

Or, alternatively you can also cope with it through **adaptive thresholding**. Even if the image's illumination may be globally uneven, locally it can be quite uniform.

So you determine a window size W,
slide the window on top of the image,
and threshold every window independently;
e.g. using Otsu's method.

Be careful not to threshold uniform areas!
Just set them to black or white by checking their variance.

Of course determining the suitable W size might require some trial and error.



GPU-ACCELERATED LOCAL TONE MAPPING FOR HIGH DYNAMIC RANGE IMAGES

Qiuwen Zhan*, Aung Thant**, Changqing Zou***

*Department of Electrical Engineering, Stanford University
**School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu, China
***School of Computer Science, The University of Nottingham, UK

ABSTRACT

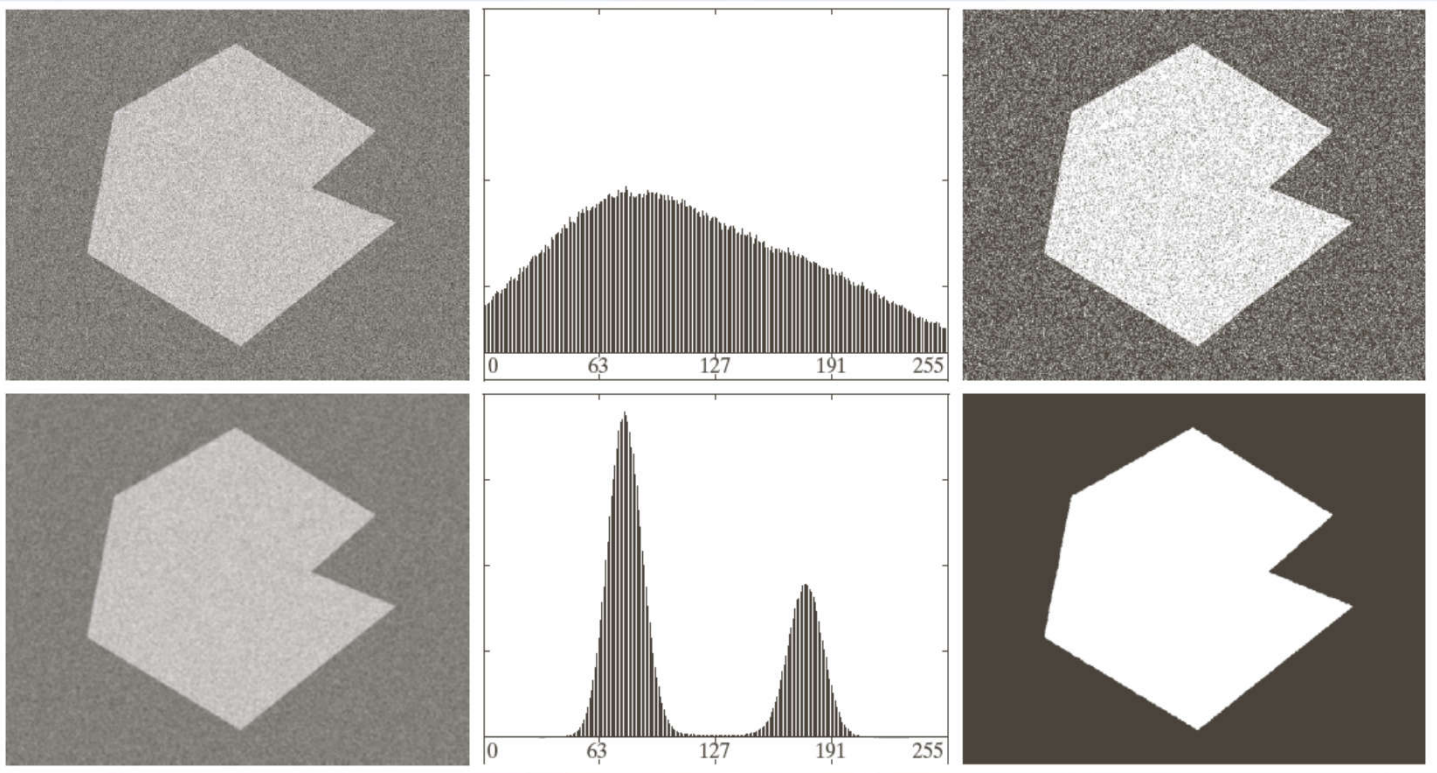
2. REVIEW OF TONE MAPPING METHODS

Tone mapping operators are usually classified as either global or local. Global tone mapping techniques apply the same appropriately designed mapping function to every pixel across the image. [3] and [4] are pioneering works. The operators attempt to match the display brightness with real world sensations, and match the perceived contrast between the displayed image and the scene respectively. Later, [5] proposes a technique based on a comprehensive visual model, successfully simulating important visual effects like adaptation and color appearance. Further, [6] presents a method based on logarithmic compression of luminance values, imitating the human response to light. Recently, [7] formulates the tone mapping problem as a quantization process and employs an adaptive conscience learning strategy to obtain mapped images. Perhaps the most comprehensive technique is still that of [8], which first improves histogram equalization and then extends this idea to incorporate models of human contrast sensitivity, glare, spatial acuity, and color sensitivity effects.

Local tone mapping techniques use spatially varying mapping functions. [9-12] are based on the same principle of decomposing an image into layers and differently compressing them. Usually, layers with large features are strongly compressed to reduce the dynamic range while layers of details are untouched or even enhanced to preserve details. [13] presents a method based on a multiscale version of the Retinex theory of color vision. [14] attempts to incorporate traditional photographic techniques to the digital domain for reproducing HDR images. [15] compresses dynamic range through the manipulation of the gradient domain in the logarithmic space. More recently, [16] proposes a novel method by adjusting the local histogram. There has been little published research to explore rendering HDR images on the GPU [17,18] and these works

Noise is yet another disruptive factor for thresholding.

If the image is noisy, forget thresholds, smooth it first!

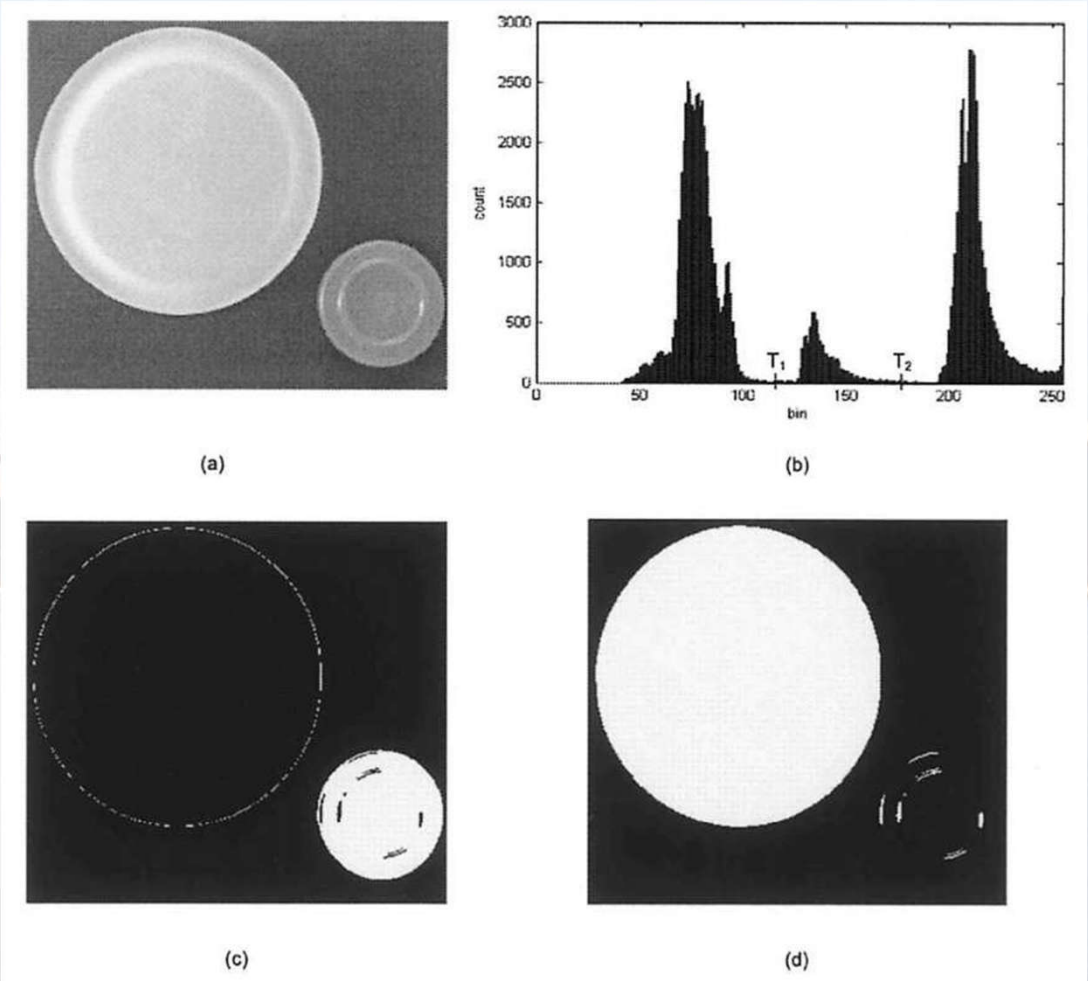


a b c
d e f

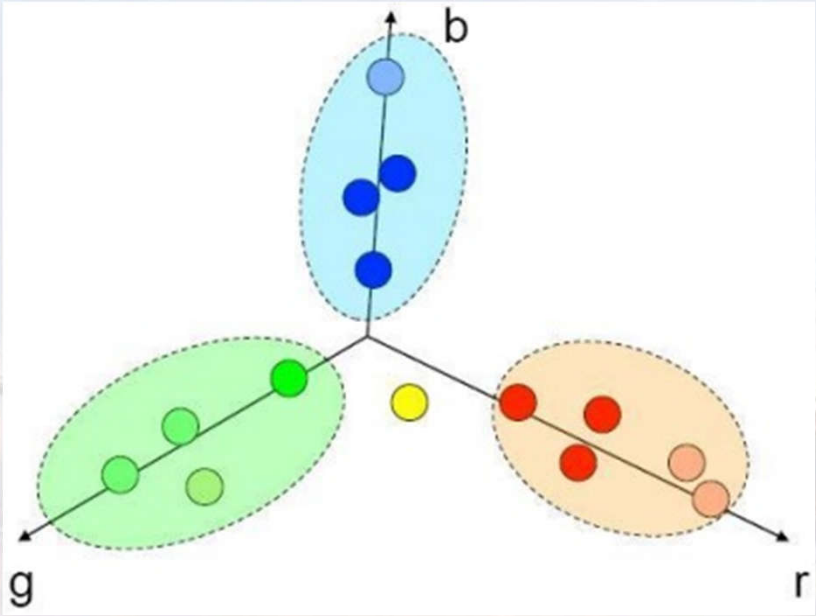
FIGURE 10.40 (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5 × 5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method.

Of course not all images have bi-modal histograms, there can be 3 or more.

In which case, you'll either try to calculate each threshold necessary to separate the histogram modes, or…
…use some other segmentation method.



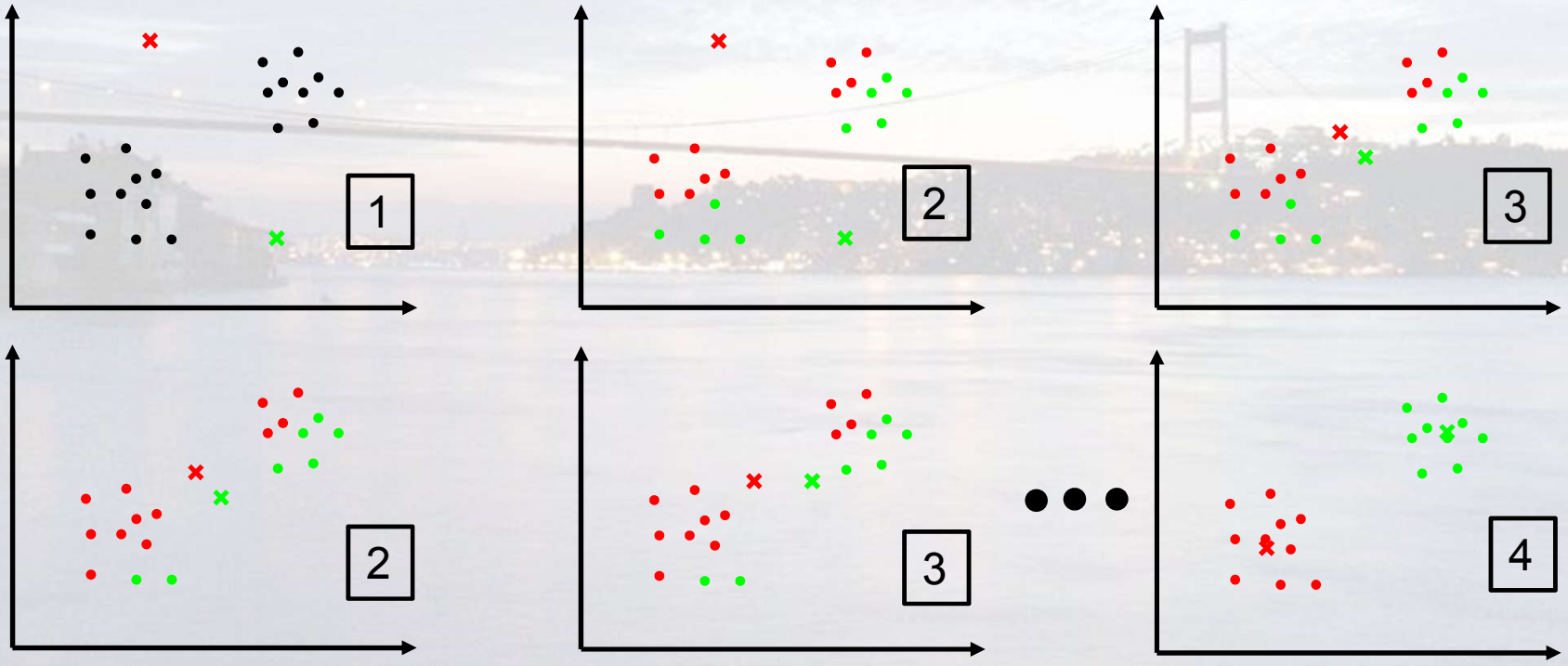
Another popular approach for segmentation is **clustering**. It's a concept we borrow from machine learning, and it describes the process of *grouping together similar things*.



Assuming that the semantically meaningful regions have similar pixel values, we could discover them by clustering the discrete space of pixel values.

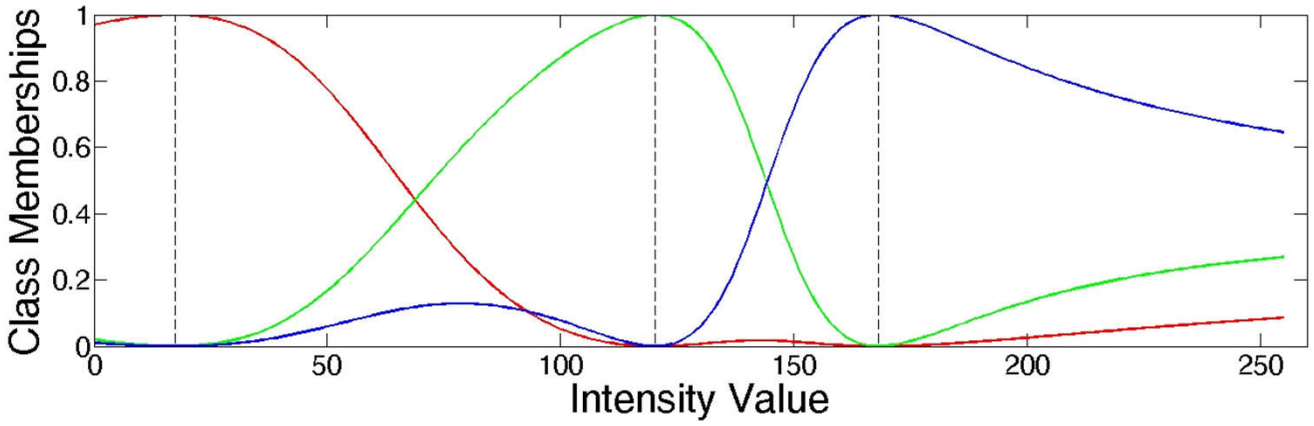
One of the simplest clustering algorithms is **K-means**.

1) Select K random cluster centers
2) Associate every pixel value with the cluster center closest to it.
3) Calculate the center of every cluster.
4) Repeat 2,3 until the cluster centers move less than a predefined threshold ε



Segmentation

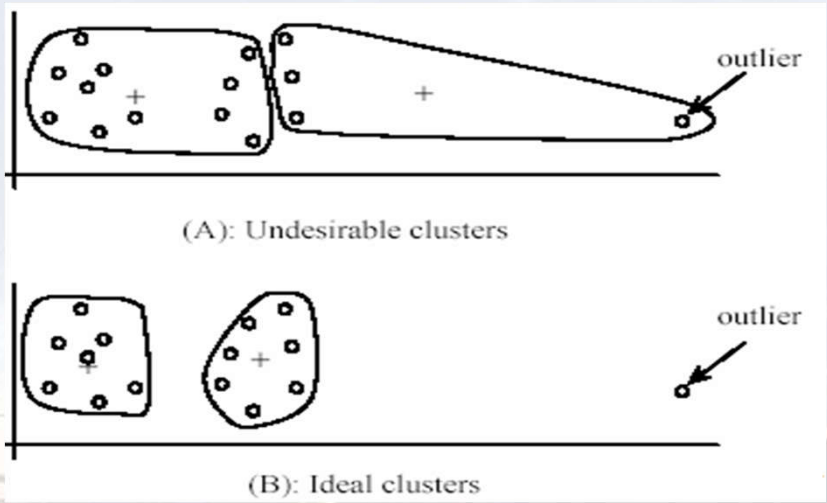
Segmentation



+ K-means based image segmentation is **fast** and **easy** to implement.

Keep in mind that you should use a suitable color similarity metric.

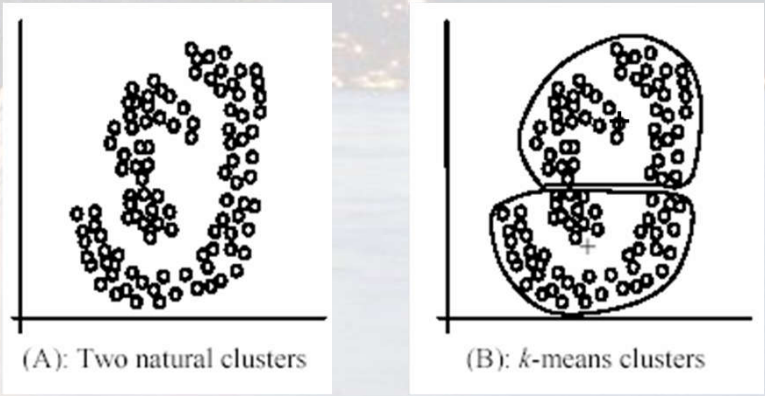
Ideas?



- However, it requires a priori information on the number of regions (K = ?); which is often not available.

- It is sensitive to outliers.

- And it produces only spherical clusters…



Adapted from Kristen Grauman

Another clustering method commonly used for segmentation is **meanshift** (1997).

Its main advantage is that it does not require a priori the number of regions.

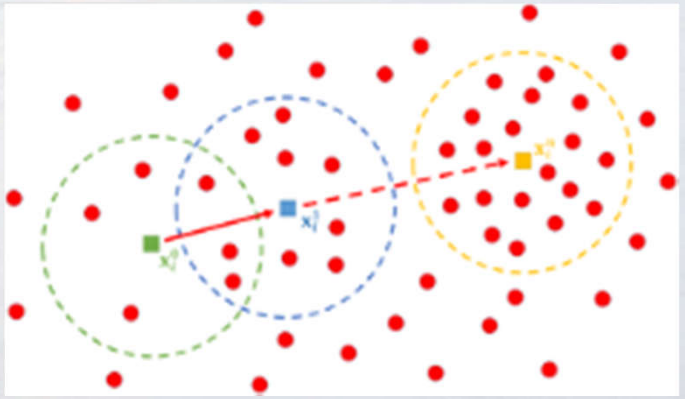
Given the discrete space of pixel values, it detects locations of maximal density (i.e. modes).

1) For every point p of our samples, calculate the weighted mean $m(p)$ of its "nearby" neighbors (taking place within a circle of predefined radius):

$$m(p) = \frac{\sum_{p_i \in N(p)} K(p_i - p) p_i}{\sum_{p_i \in N(p)} K(p_i - p)}$$

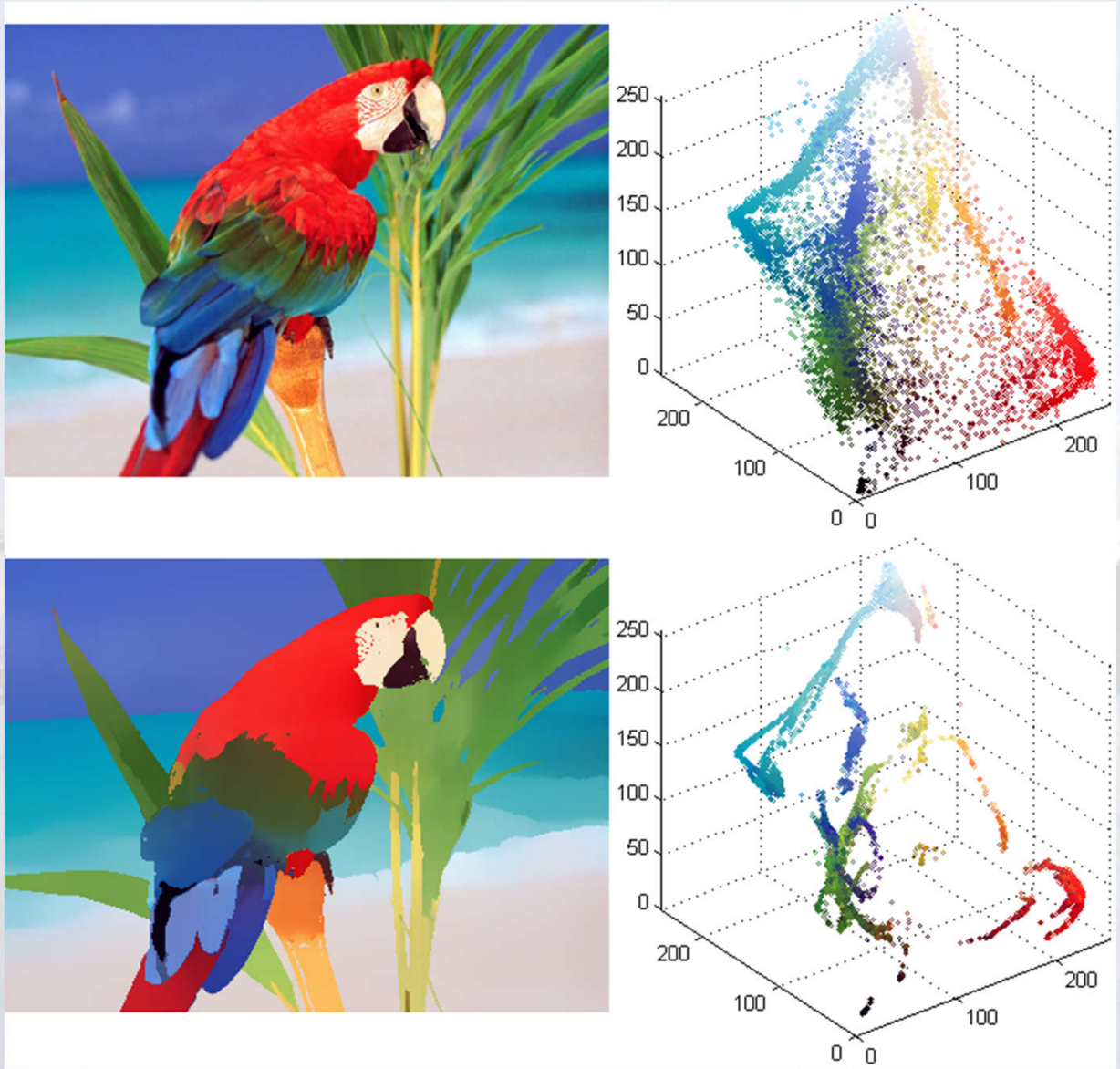
where $K(s)$ is a kernel function, typically Gaussian $K(s) = e^{-cs^2}$
and $N(p)$ is the neighborhood of p, i.e. the points r for which $K(r - p) \neq 0$

2) Then we repeat (1) for $m(p)$, up until the weighted mean converges to a point s; which now represents a cluster center.



Segmentation

Segmentation



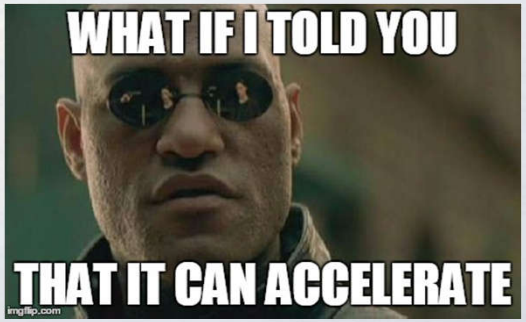
Meanshift

+ Robust to outliers
+ Does not require the number of modes, it finds them on its own
+ Can work with arbitrarily shaped clusters

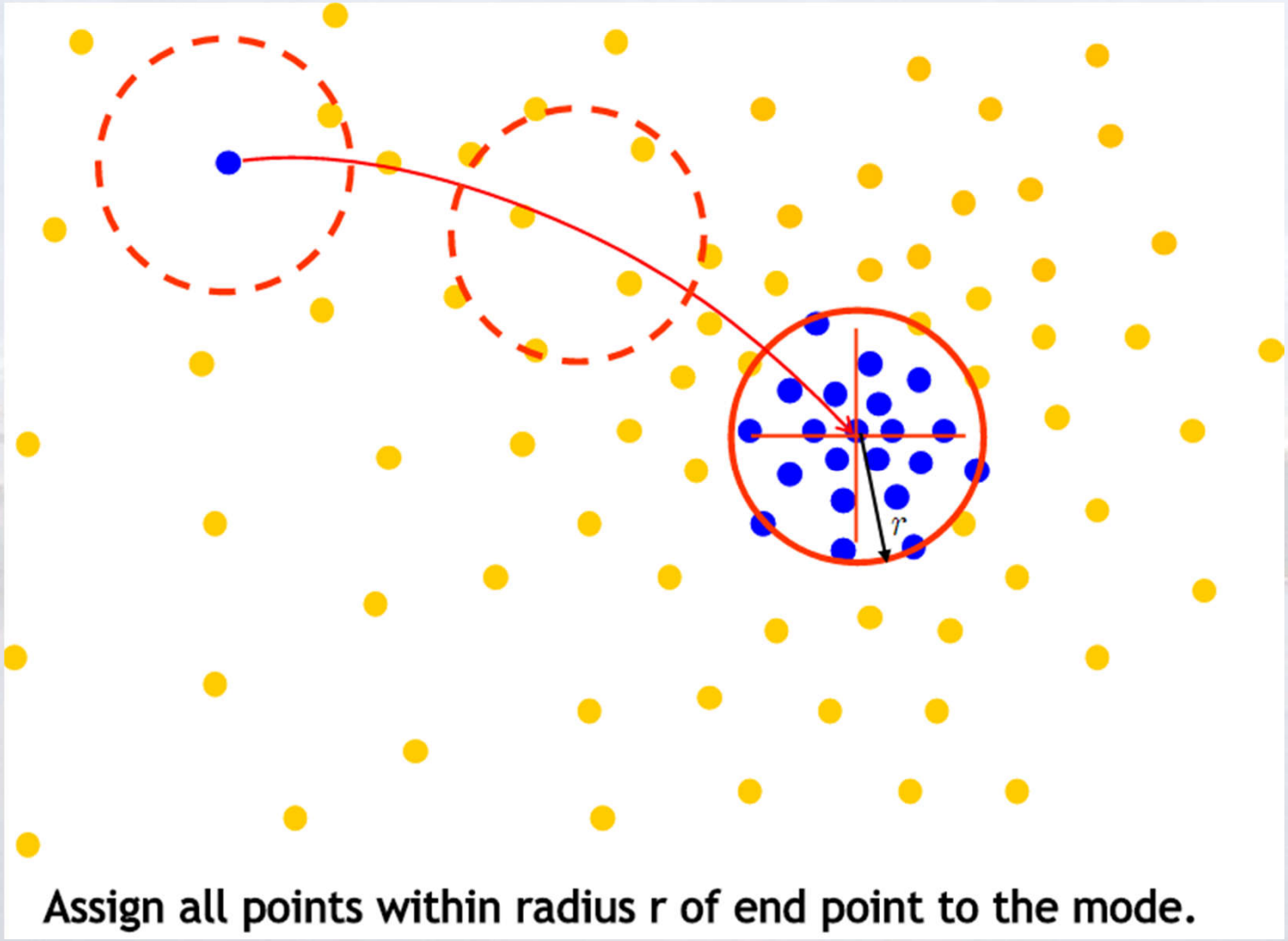
It disregards spatial information if you work on pixel values; this can be rectified by working on the discrete space of **pixel values and coordinates**; e.g. (r, g, b, x, y)

However,

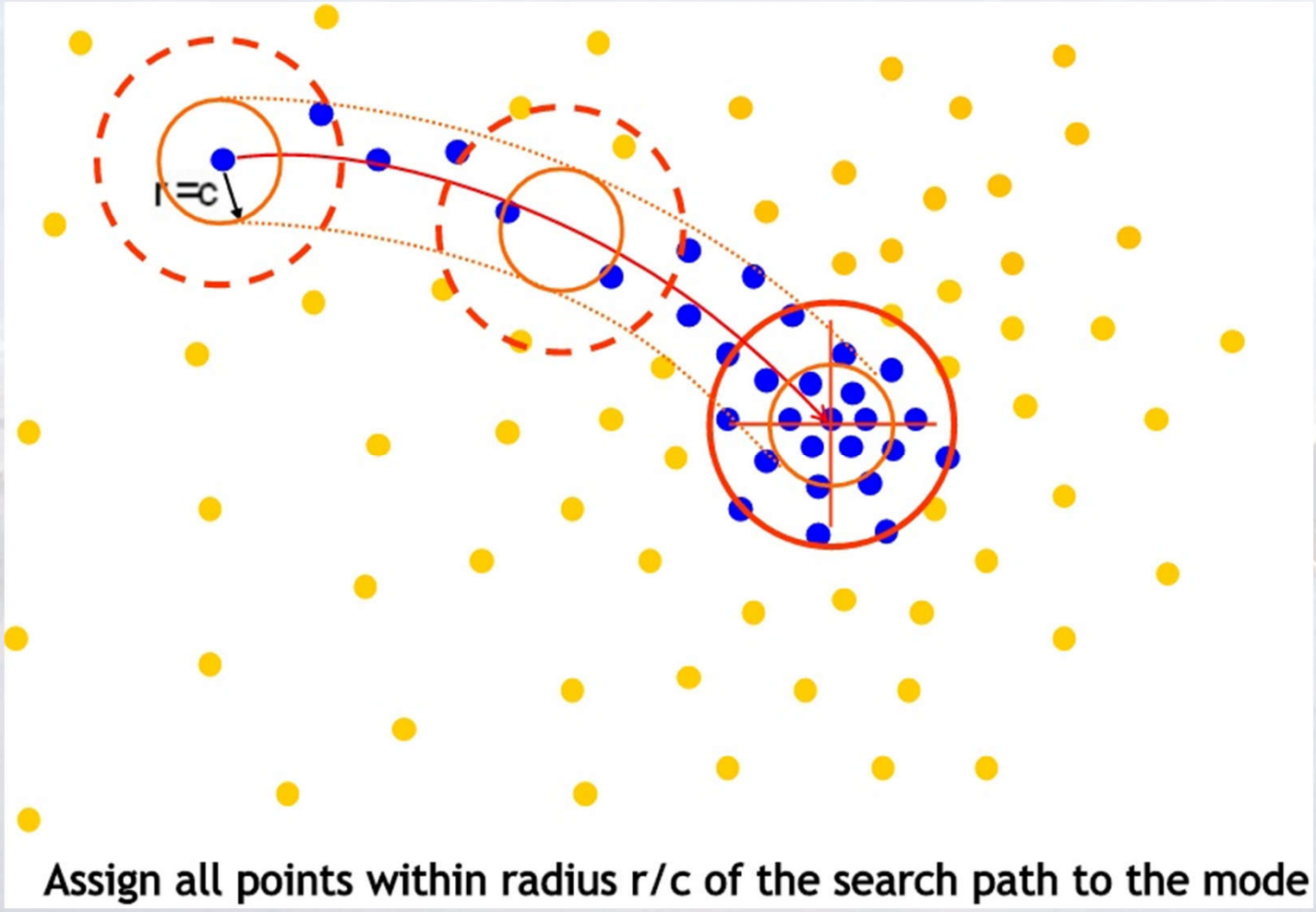
- The window size and kernel type affect the outcome
- It's computationally intensive, but can be accelerated!



Acceleration tip #1



Acceleration tip #2



Region growing

The idea is to start with some “seed” pixels at distinct regions, and to grow them by grouping similar neighboring pixels, up until the entire image is covered.

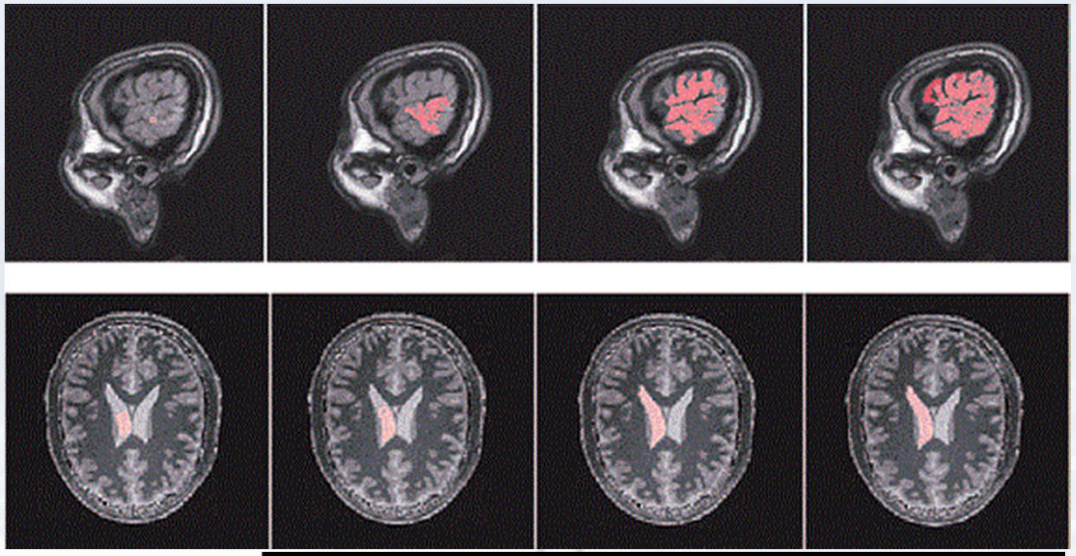
It requires seeds and a similarity measure for deciding whether a pixel is similar enough to a region to be added to it.

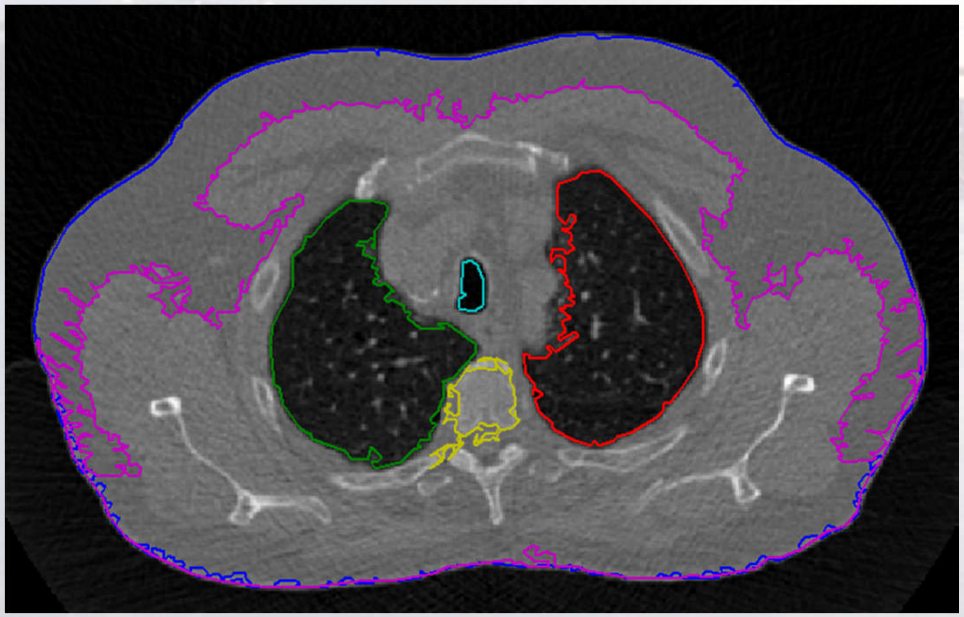
It provides a high degree of flexibility; since the growing decision can be made based on arbitrary criteria:

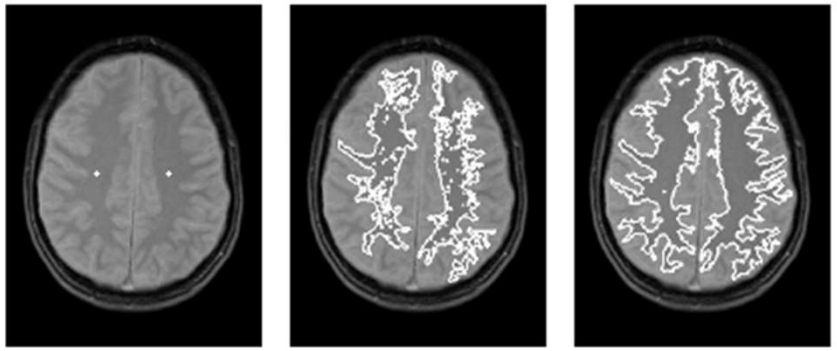
- Two regions (or a region and a pixel) are merged if their means are similar
- A region can continue growing as long its standard deviation is below a threshold

…and more.

Examples of segmentation by region growing



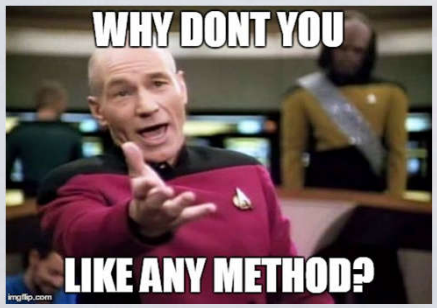


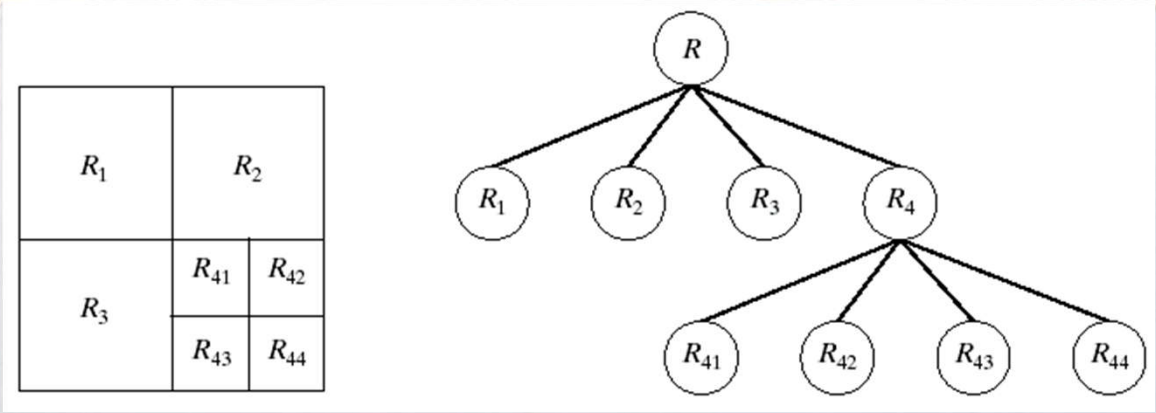


Region splitting (quad tree)

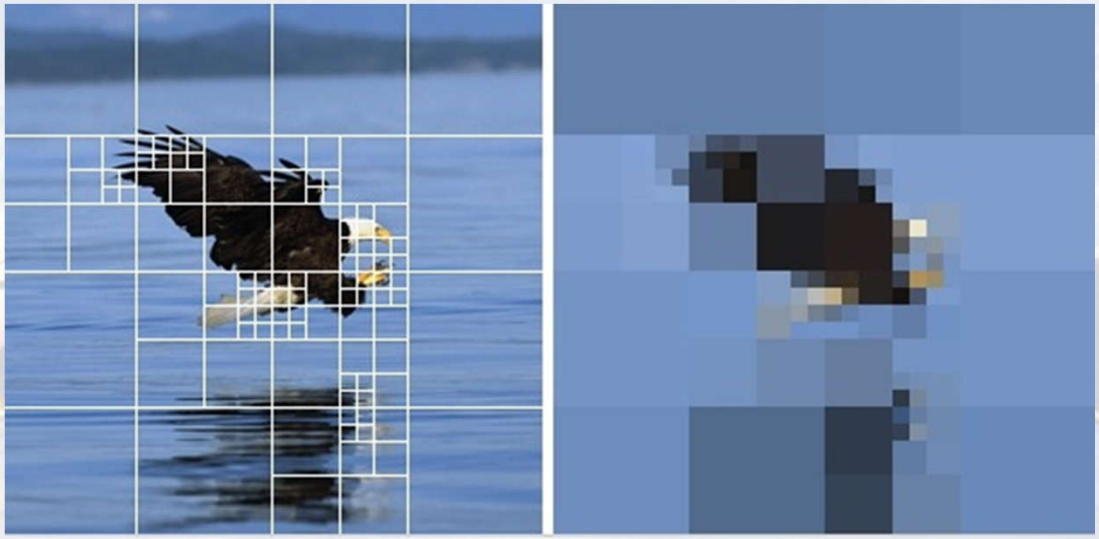
This is the opposite idea of region growing. We start with the entire image, and subdivide it into quadrants, and then again each region into quadrants, and stop only when a certain homogeneity predicate is achieved; e.g. low standard deviation.

It's fast and quite effective, but the output is blocky.

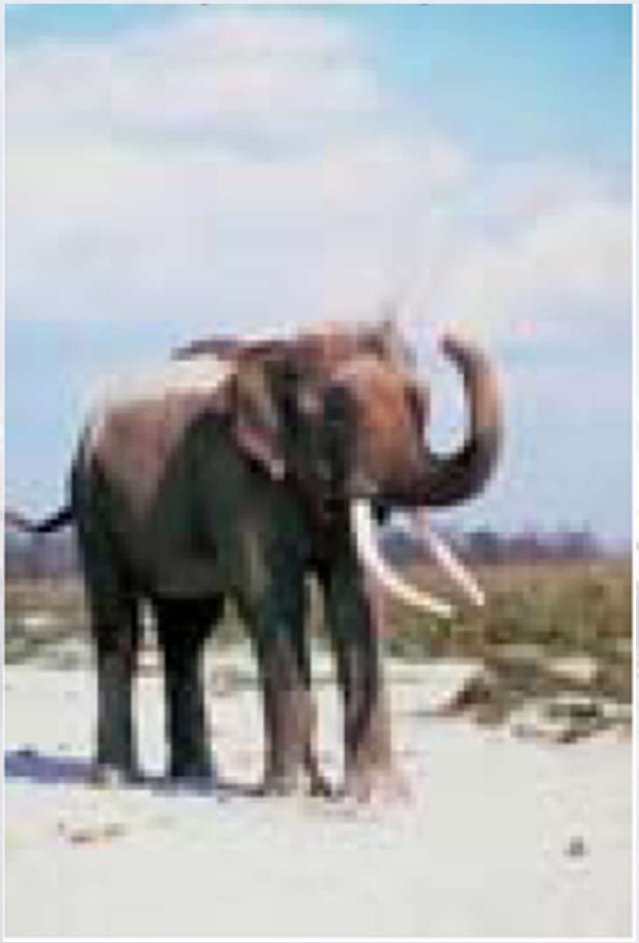


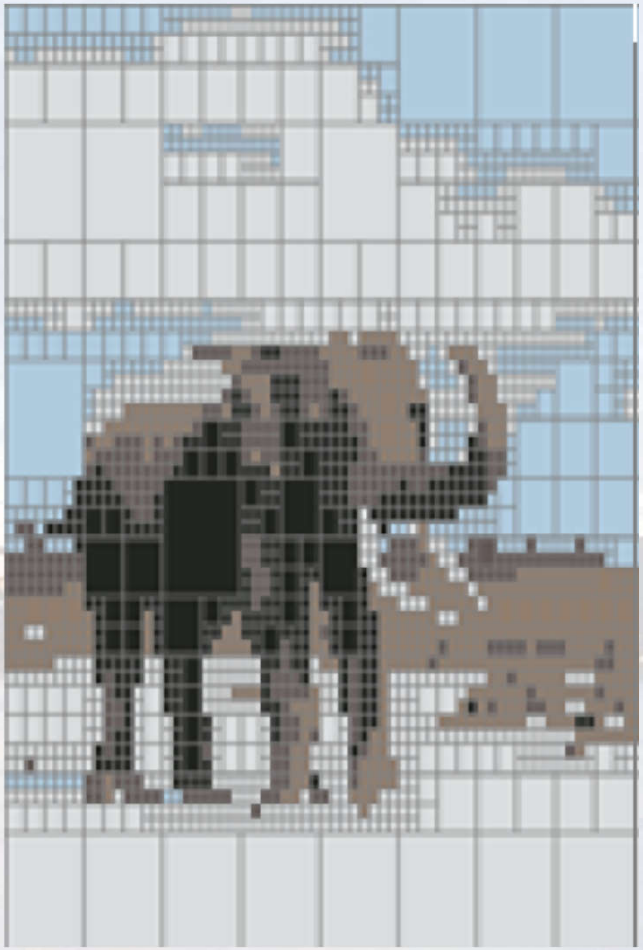


Segmentation results with quad trees



Segmentation

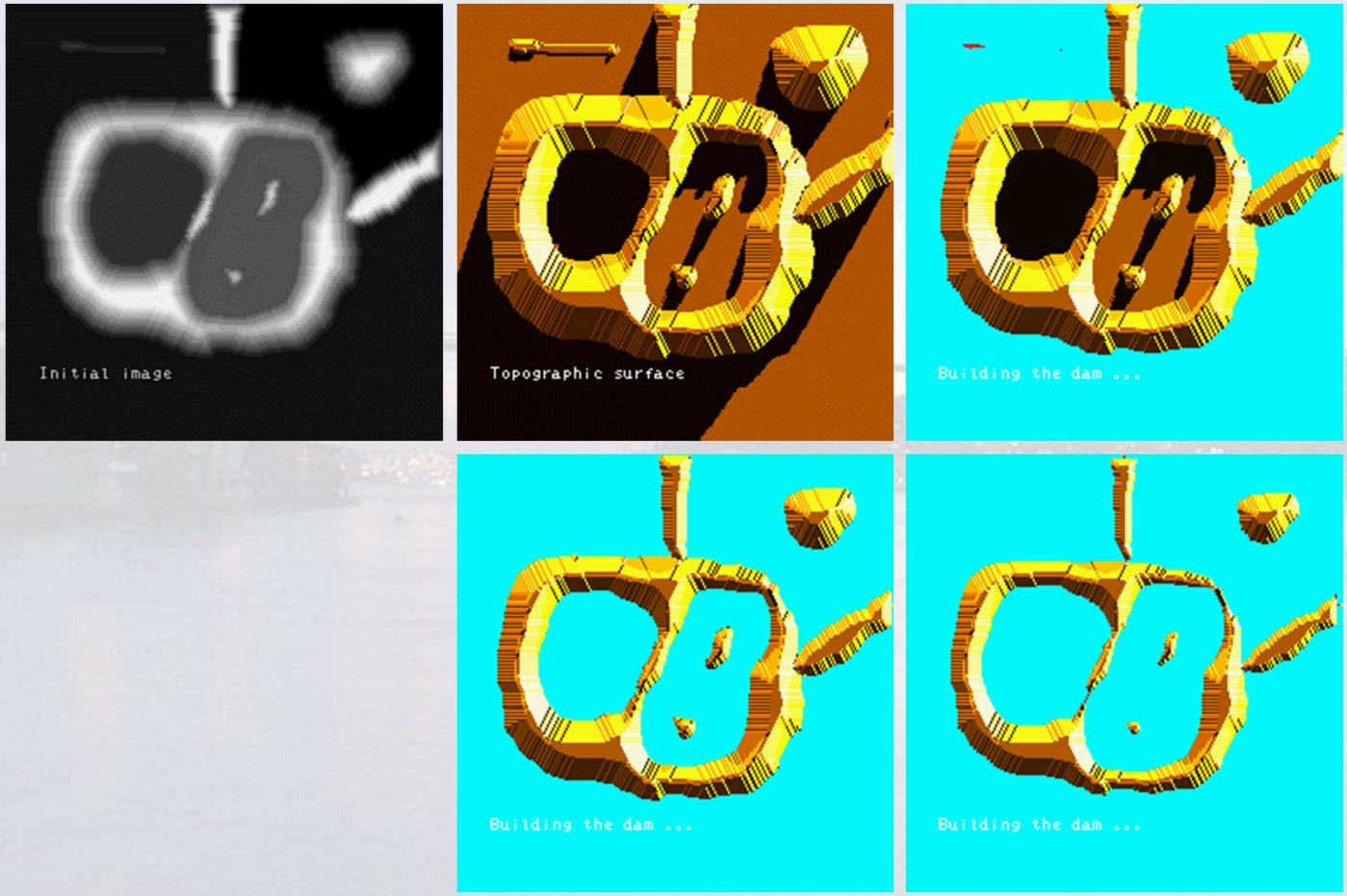




Images from Selim Aksoy, Bilkent U.

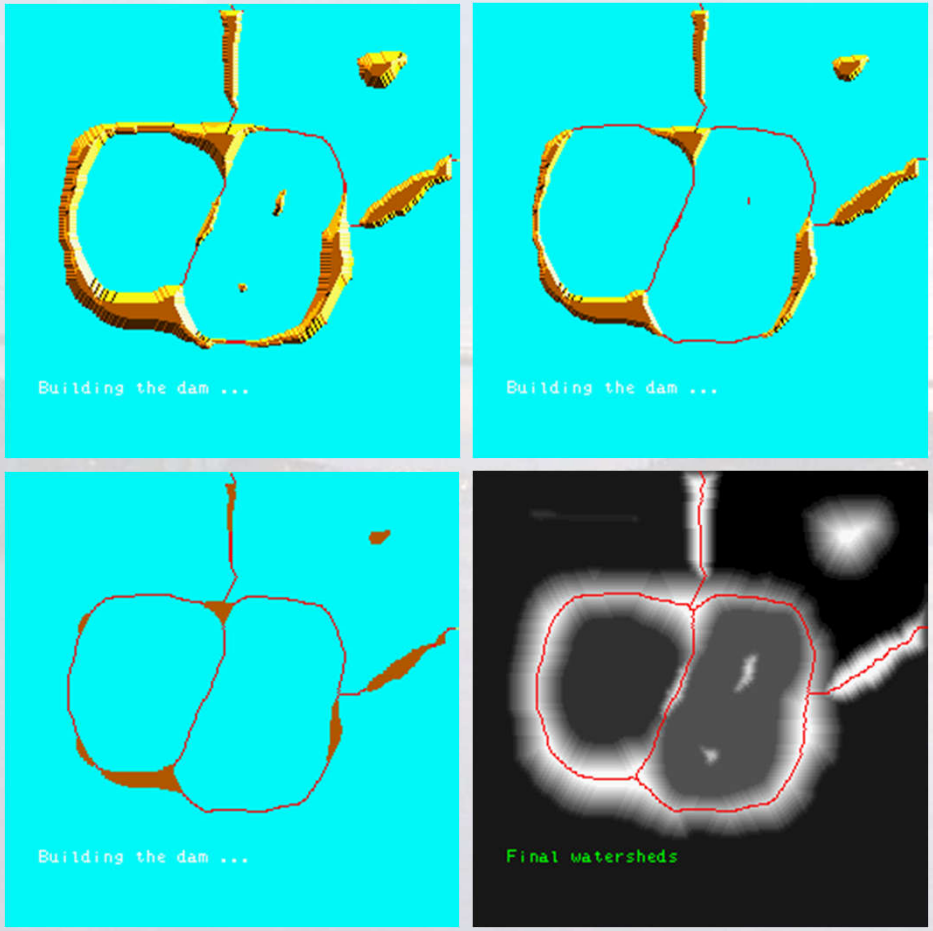
The watershed transform (1990s)**:** the main morphological tool for segmentation.

Main idea: flooding of the topographic surface from regional minima!



The watershed transform (1990s)**:** the main morphological tool for segmentation.

Main idea: flooding of the topographic surface from regional minima!

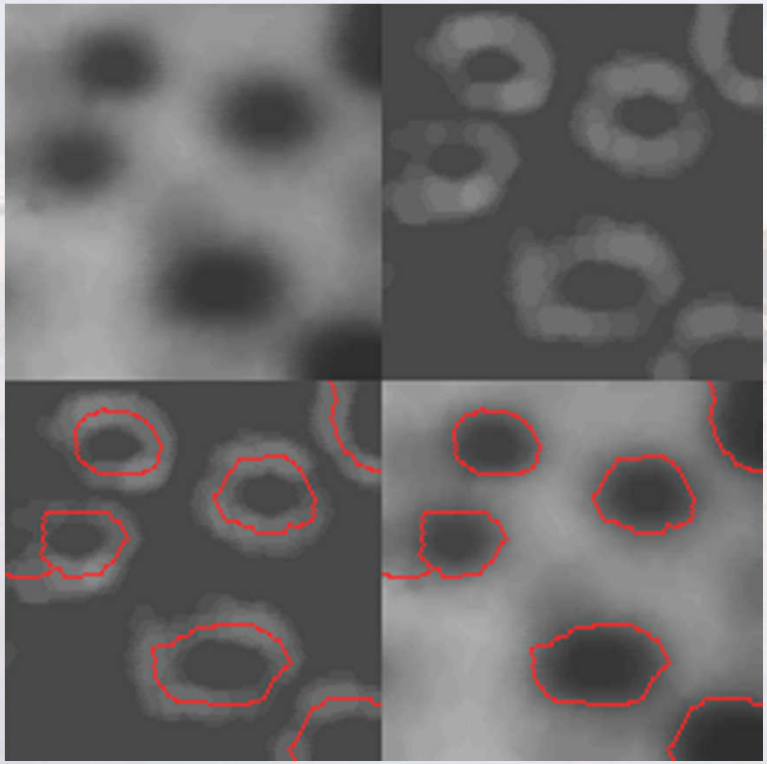


Adjacent basins are not allowed to merge!

For the watershed transform to work, the region borders need to correspond to elevated parts of the topographical surface; i.e. the edges need to be bright, and the uniform areas need to be dark.

The watershed transform must be applied **on the gradient of a grayscale input!**

original



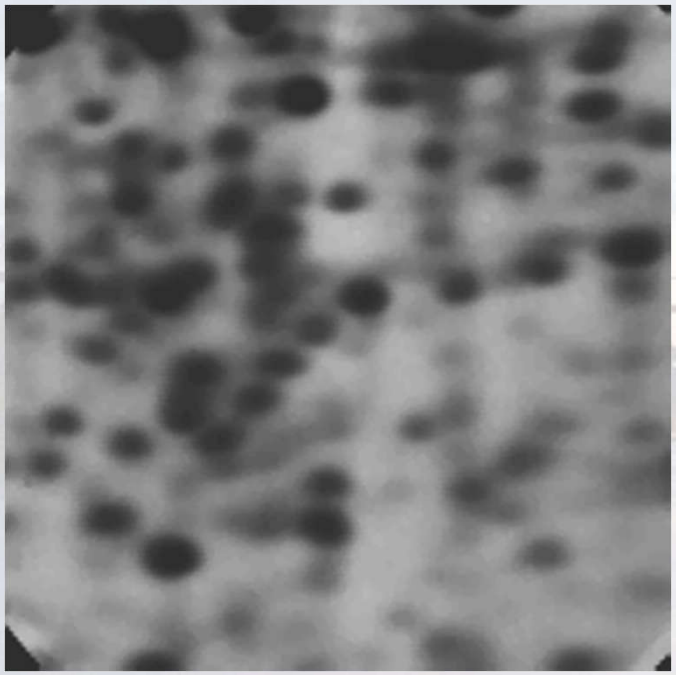
gradient

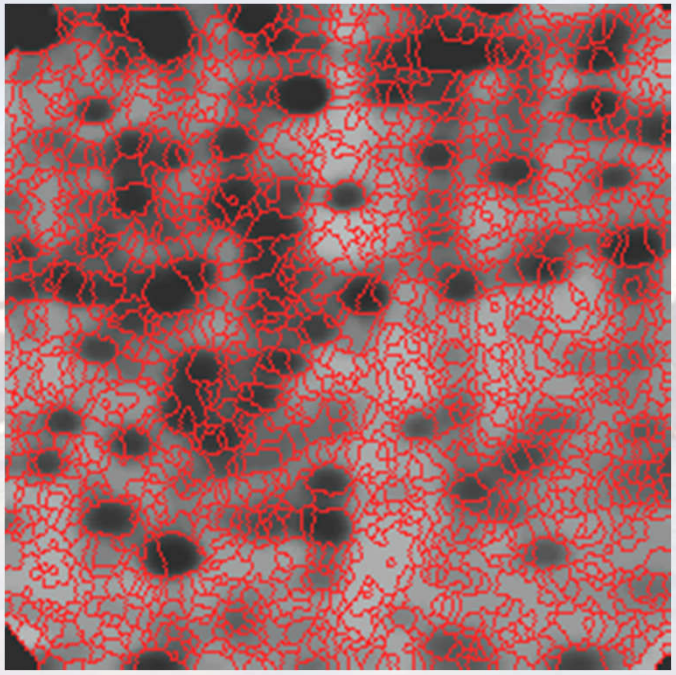
Borders superimposed on the gradient

Borders superimposed on the original

The watershed transform is **fast** and **fully automatic**!

The problem is that if applied directly on a given gradient image, most probably it will lead to an **over-segmentation**, i.e. a partition that is finer than desired.





Why? Because gradients have many minima, and the WT gives one region per minimum!

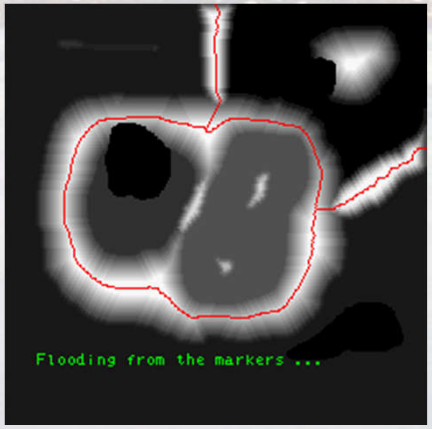
This can be dealt with using the **marker based watershed transform**!

Same logic as WT, but the sources of flooding are user-defined, i.e. the markers.

Number of markers

=

Number of regions!



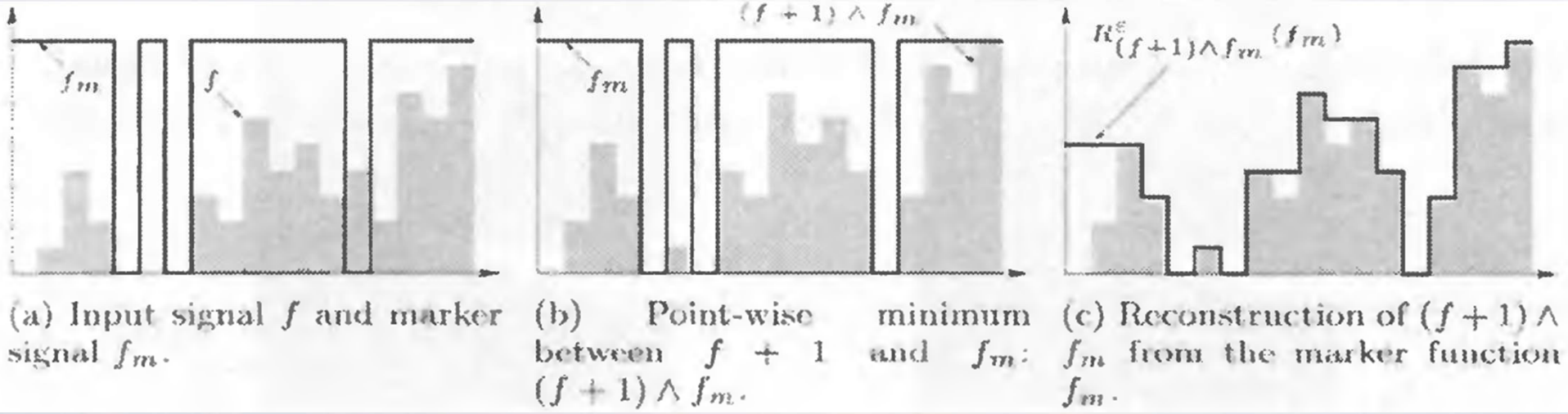
No more

over-segmentation!

Minimum imposition

$$f_m(p) = \begin{cases} 0, \text{if it}^{'}\text{s supposed to be a minimum} \\ t_{max}, \text{ otherwise} \end{cases}$$

$$R^{\varepsilon}(f_m, f_m \wedge (f+1))$$



(a) Input signal f and marker signal f_m.

(b) Point-wise minimum between $f+1$ and f_m: $(f+1) \wedge f_m$.

(c) Reconstruction of $(f+1) \wedge f_m$ from the marker function f_m.

Adapted from P. Soille

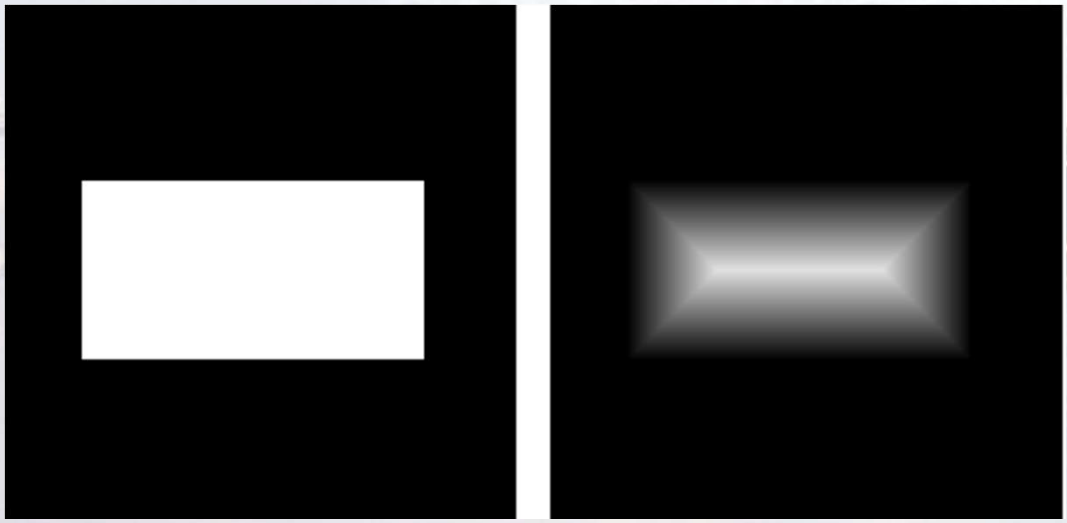
Marker based watershed transform

From the Prometheus Project

Of course the main issue now is the selection of the markers…

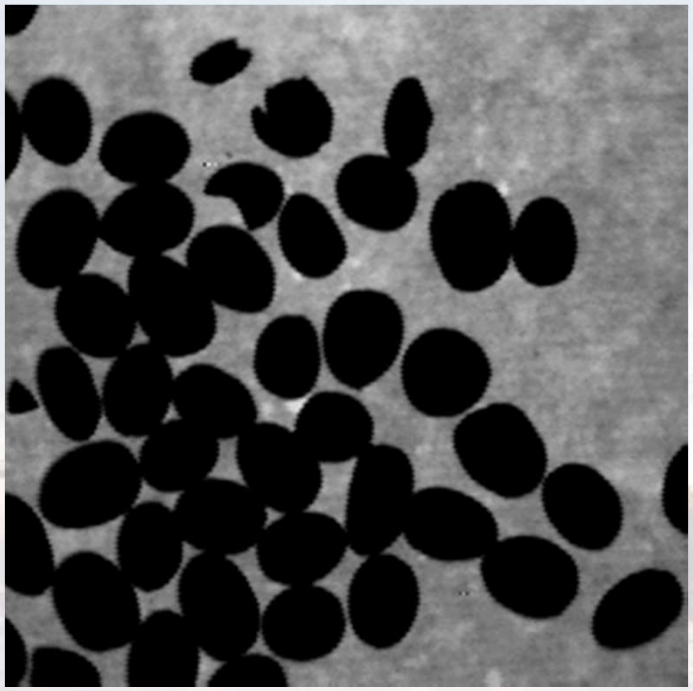
The **distance transform** is an interesting tool, that is very useful with the watershed transform family.

Given a binary image, a distance transform replaces every pixel with its distance to the closest (w.r.t. a metric) background pixel.



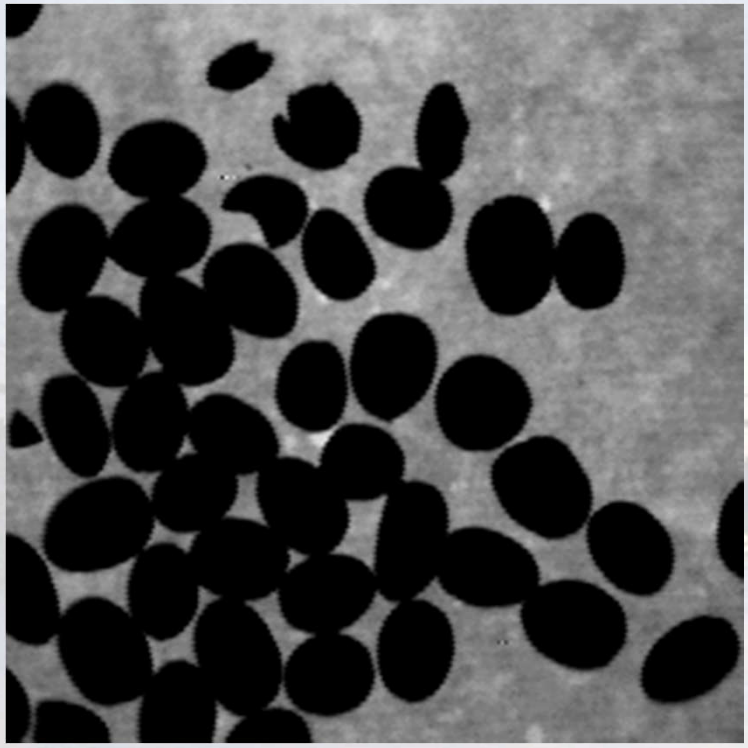
Does it remind you anything?

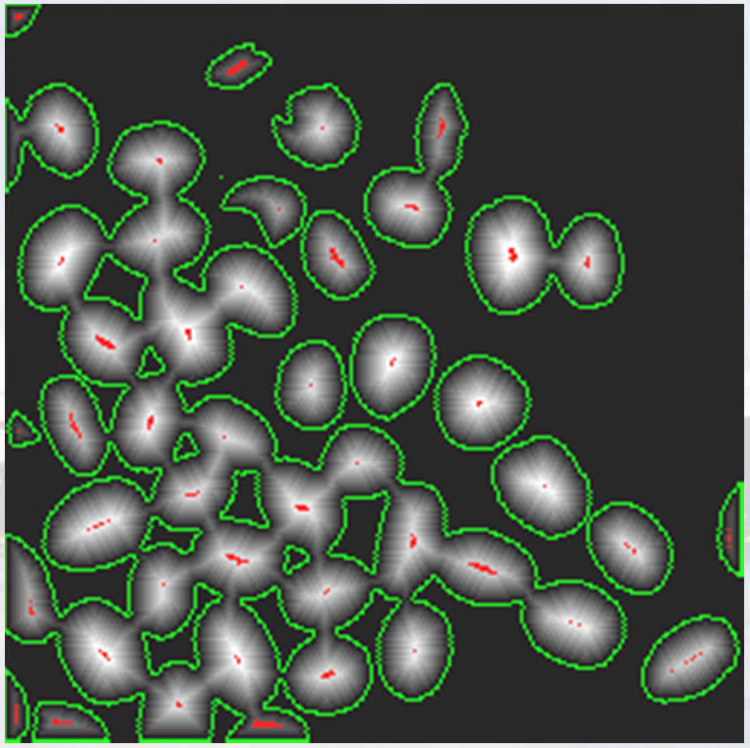
Example: coffee bean separation



Touching objects. A common problem especially with industrial vision applications.

First we take the distance transform of the beans, and detect their regional maxima; i.e. pixel of which no neighbor has greater value. They represent bean centers!

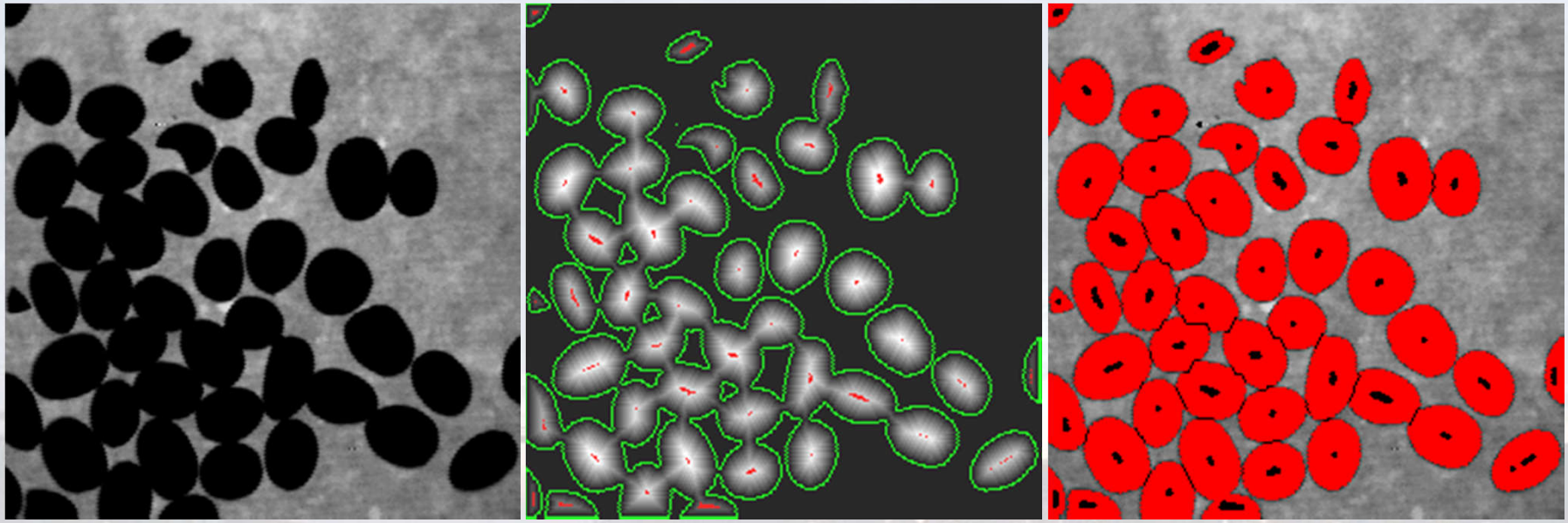




Then we perform marker based watershed with those centers as markers.

The resulting borders represent the external borders of the beans.

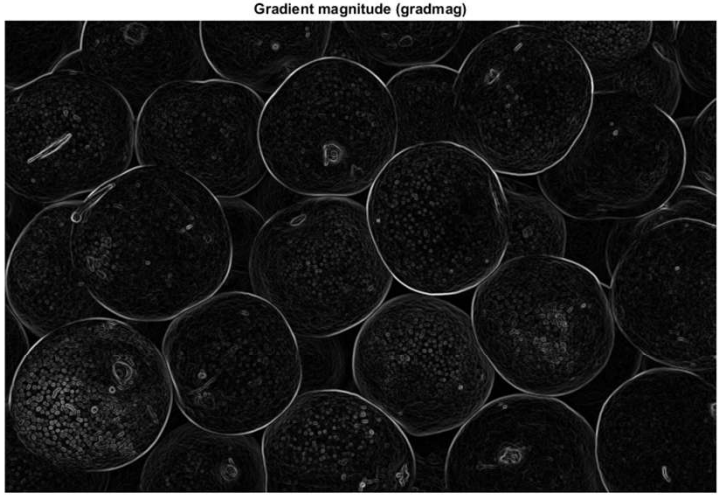
And the finale..

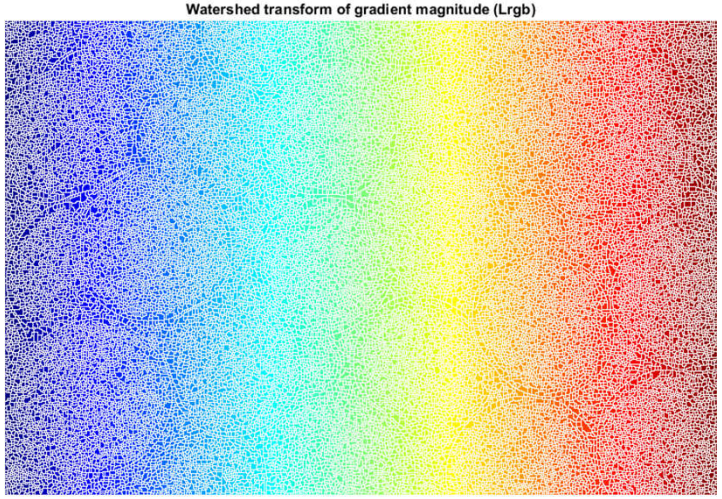


We perform once more marker based watershed, using as markers both the bean centers (**internal markers**) and their borders (as **external markers**).

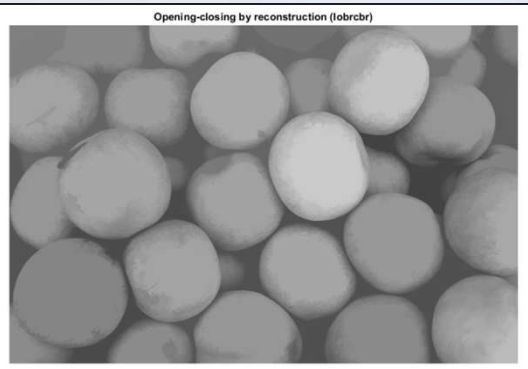
The beans are now separated!

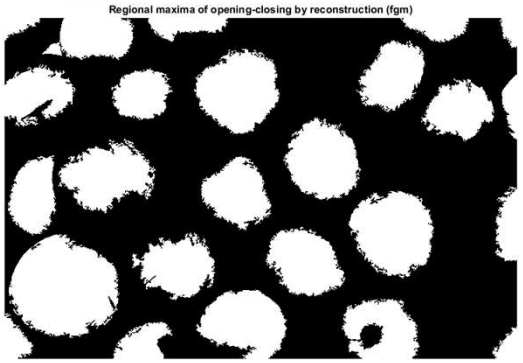
Same concept with apples; over-segmentation with WT

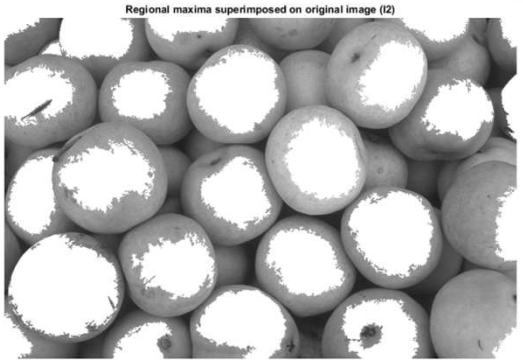


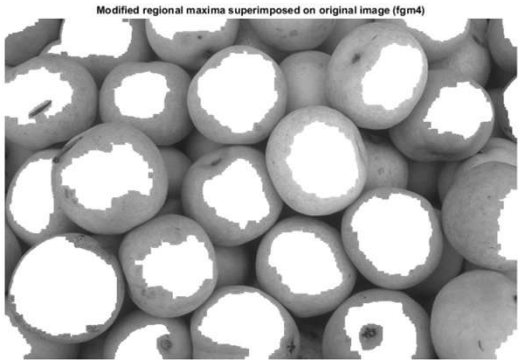


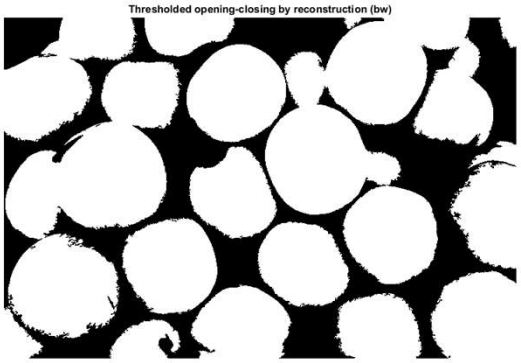
Segmentation

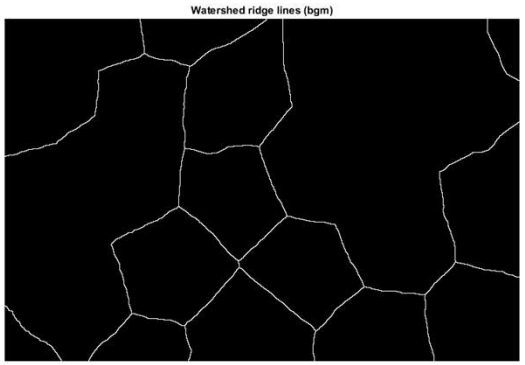


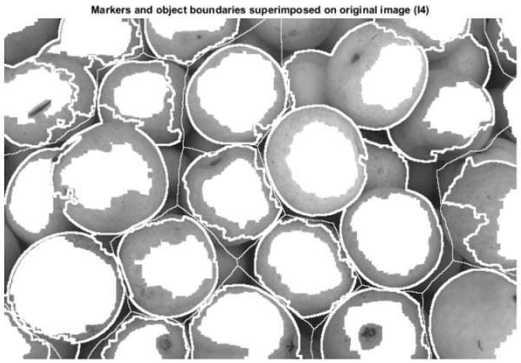


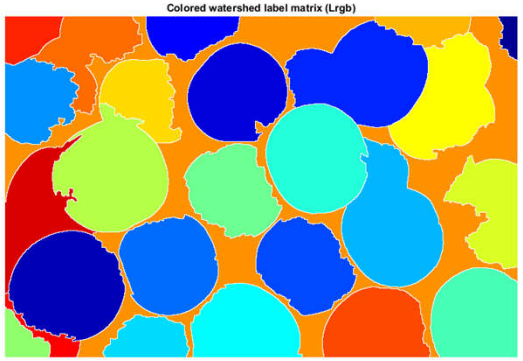


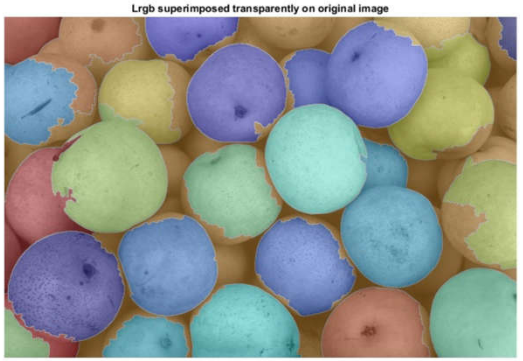








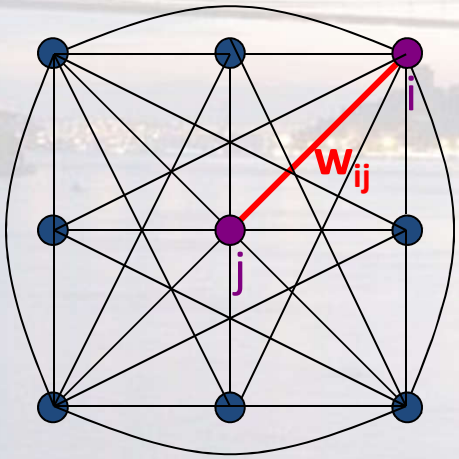


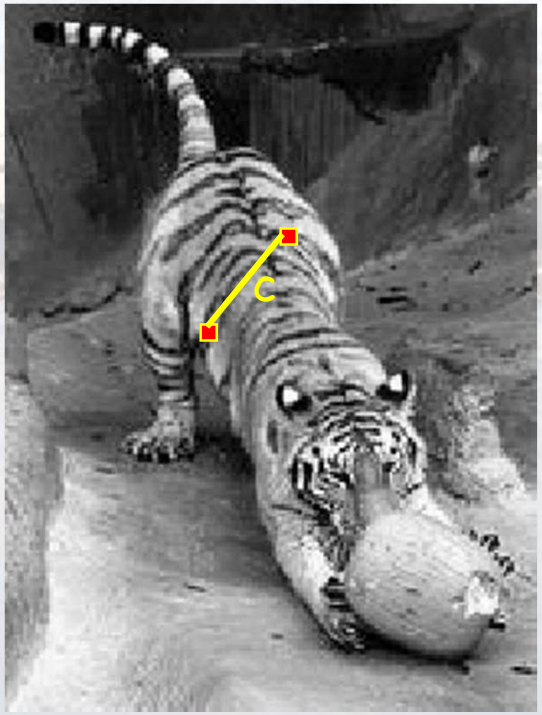


Graph based segmentation

The idea is to represent an image as a weighted graph, where nodes represent pixels or at least groups of pixels (e.g. flat zones), and the edges denote the similarity between nodes.

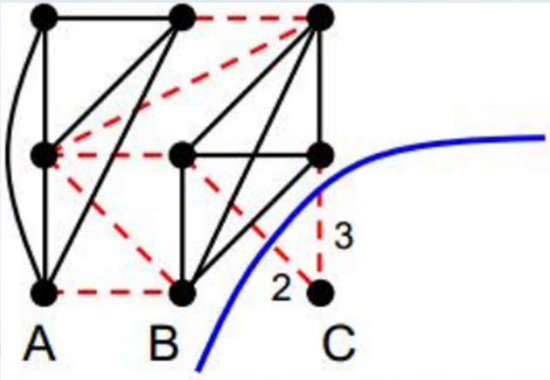
Graph based segmentation transforms the problem of segmentation into a graph partitioning problem.





Adapted from D. Holem, S. Seitz

In order to partition the graph, we need to “cut” it into disjoint sets that make “sense”.



We should remove the edges of low cost;

i.e. of low similarity.

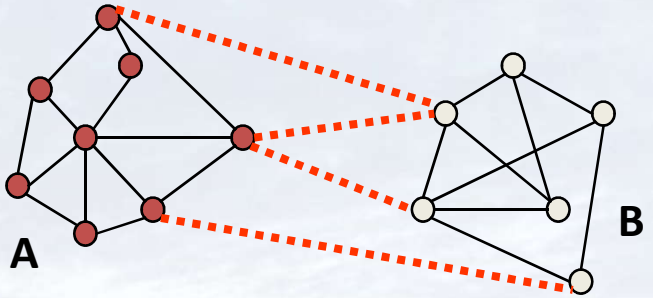
Let’s say $G = (V, E)$ is our graph, where an edge (p, q) has a weight w_{pq}

Adapted from S. Seitz

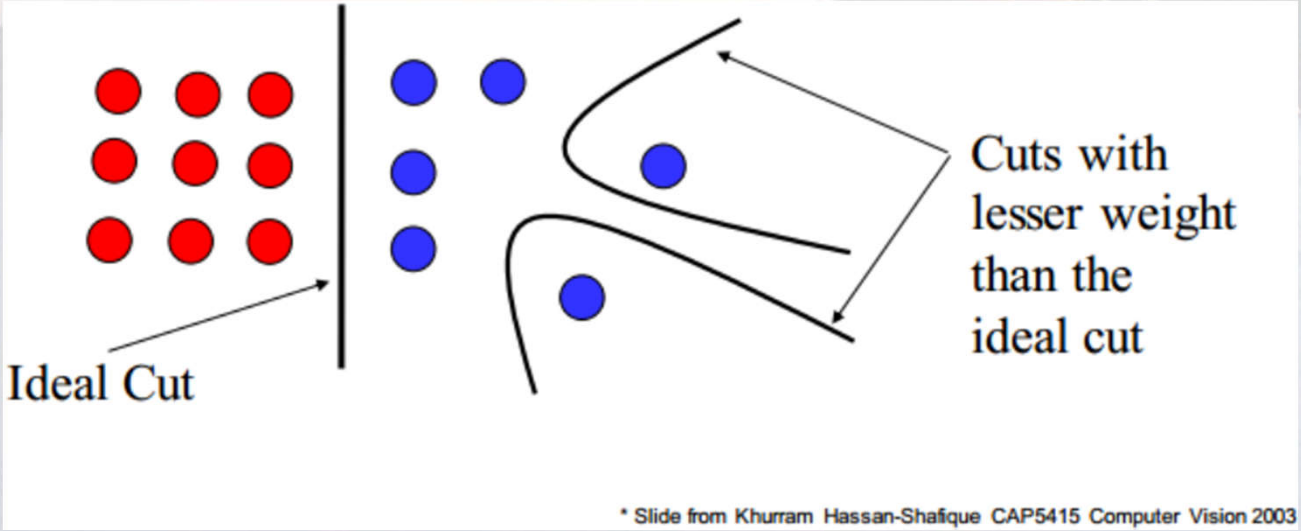
A **cut** is a set of edges whose removal makes a graph disconnected.

Its cost is the sum of weights of the removed edges:

$$cut(A,B) = \sum_{p \in A, q \in B} w_{p,q}$$



There are fast algorithms for this, but **min-cuts** tend to isolate components…



* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

To remedy this problem Shi and Malik (2000) proposed **normalized cuts**:

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(A,V)}$$

Where $assoc(A,V)$ is the sum of all edge weights touching A.

Normalized cuts are normalized by the size of the segments under consideration.

$$Ncut(A,B) = \frac{3}{21} + \frac{3}{16}$$

How to calculate $Ncuts$:

- Assume you have N nodes.
- $\vec{x}$ is an N-dimensional indicator for a set of nodes A, such that $x_i = 1$ is $node_i \in A$, and -1 otherwise.
- $d_i = \sum_j w_{i,j}$ is the total connection cost from $node_i$ to all the other nodes.
- D is the diagonal $N \times N$ matrix of connection costs, and W the matrix of weights.

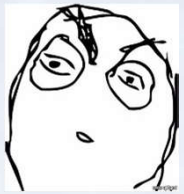
$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(B,A)}{assoc(B,V)}$$
$$= \frac{\sum_{(x_i>0, x_j<0)} -w_{ij}x_i x_j}{\sum_{x_i>0} d_i} + \frac{\sum_{(x_i<0, x_j>0)} -w_{ij}x_i x_j}{\sum_{x_i<0} d_i}$$

After a few operations, it can be converted into something like:

$$\frac{y^T(\mathbf{D}-\mathbf{W})y}{y^T\mathbf{D}y}$$

where $y = (1+x) - \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}(1-x).$ is a continuous approximation of x.

$\frac{y^T(\mathbf{D}-\mathbf{W})y}{y^T\mathbf{D}y}$ this (plus a few constraints) is **Rayleigh's Quotient**!

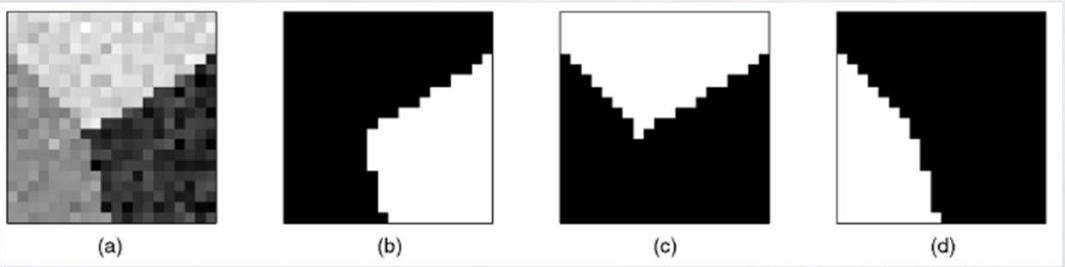


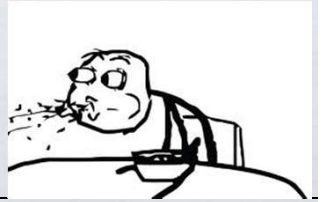
Given $R(A,v)=\frac{v^tAv}{v^tBv}$ where B is a positive definite hermitian matrix, then

the expression is minimized for the minimal eigenvector of A; solve: $A\vec{v}=\lambda B\vec{v}$

In other words, we need to solve $(\mathbf{D}-\mathbf{W})y=\lambda\mathbf{D}y.$

Long story short, use the eigenvector with the 2nd smallest eigenvalue to bipartition the graph, and continue repartitioning recursively if necessary.





Summary

1. Given features, construct the graph
2. Solve $(\mathbf{D}-\mathbf{W})y=\lambda \mathbf{D}y.$ for eigenvectors with the smallest eigenvalues
3. Use the eigenvector with the 2nd smallest eigenvalue to bipartition the graph
4. Recursively repartition the segmented parts if necessary

Weights usually include the pixel coordinates X as well as the spectral information F so that distant pixels have zero weights. F can be intensity, color, texture, etc.

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}(i)-\boldsymbol{F}(j)\|_2^2}{\sigma_I}} * \begin{cases} e^{\frac{-\|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2^2}{\sigma_X}} & \text{if } \|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise,} \end{cases}$$

Normalized cuts are **flexible** and **robust**, but also **computationally complex** and **you need to know when to stop.**

Many more segmentation methods are out there

- Markov random fields
- Grab cuts

etc.