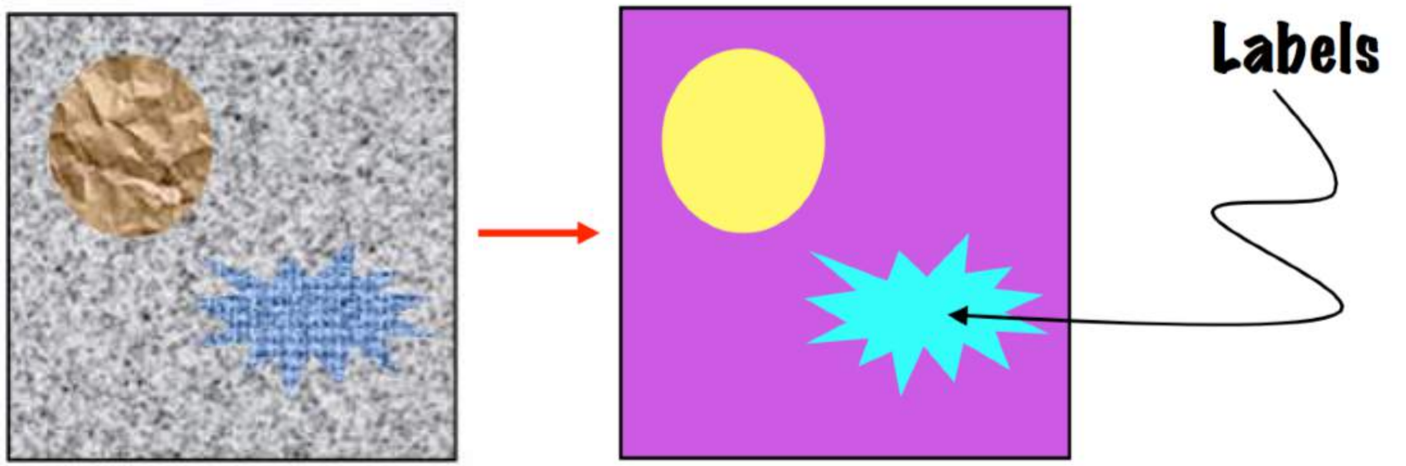
Segmentation

Part I

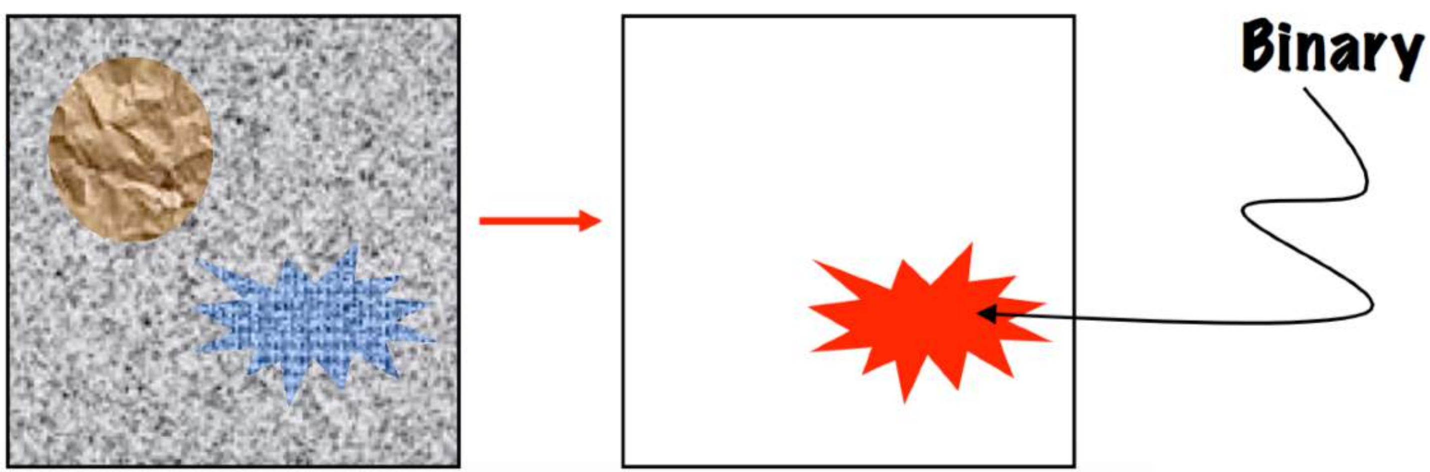
Introduction

What is segmentation?

- Partitioning images/volumes into meaningful pieces



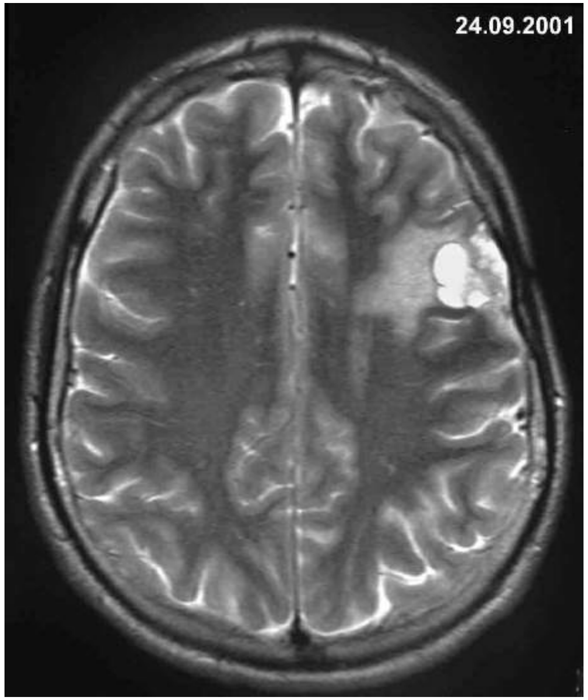
- Isolating a specific region of interest

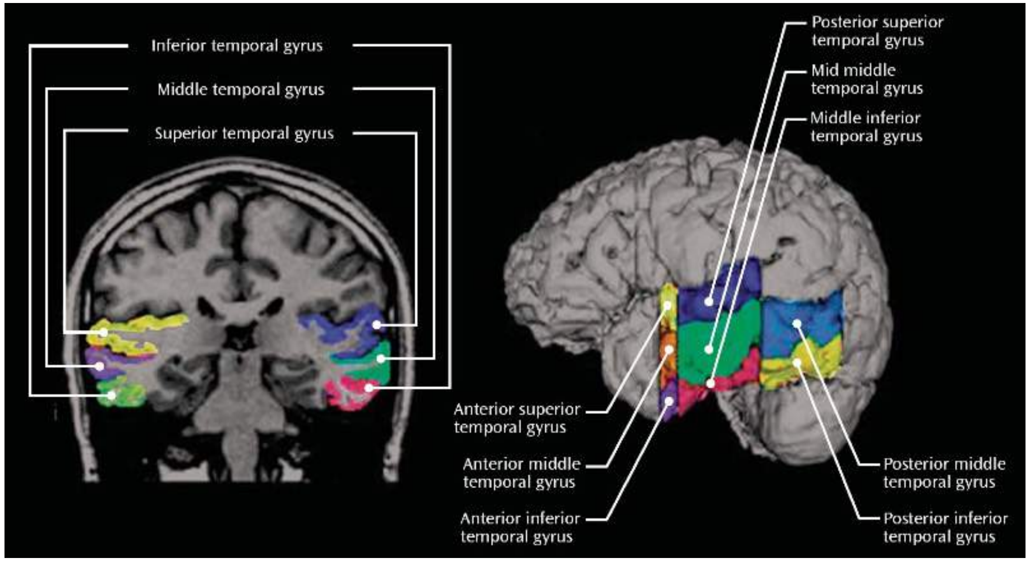


Why is segmentation interesting?

- Detection/recognition of objects
  - Where is the vehicle?
  - Which type of vehicle is it?

- Quantifying object properties
  - How big is a tumour? Is it expanding or shrinking?
  - Statistical analysis of sets of biological volumes





Main categories of algorithms

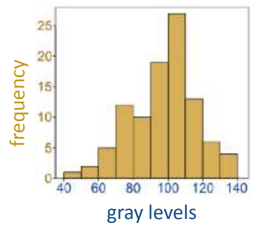
- Threshold based approaches
- Region based approaches
- Morphological watershed
- Active contours

Threshold based approaches

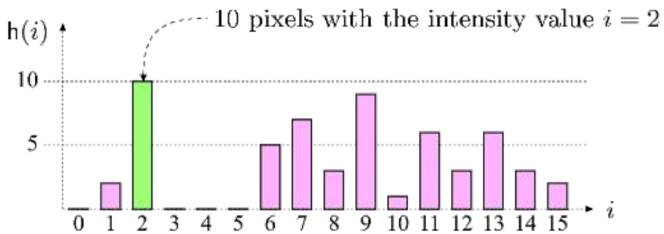
recall Histogram of an image

- A **histogram** is a function $h(i)$ that gives the *frequency of each intensity* i that occur in an image
  - Given an image $I : \Omega \to [0 \ldots K-1]$, its histogram is the function:

$$h(i) = \mathrm{card}\,\{\ (u,v)\ |\ I(u,v) = i\ \}$$

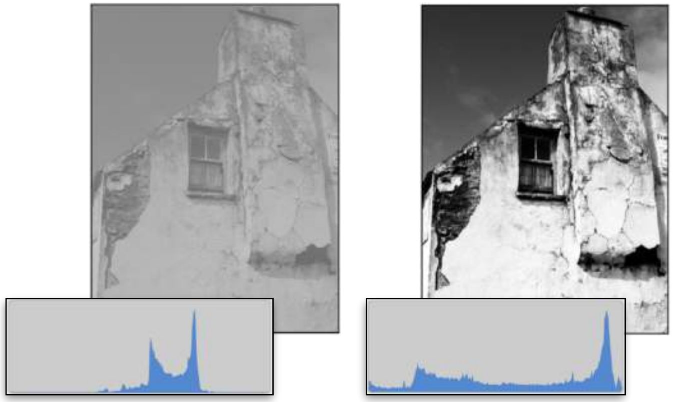


- In other words
  - $h(i)$ = number of pixels with intensity i



h(i)	0	2	10	0	0	0	5	7	3	9	1	6	3	6	3	2
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Notes
  - *Low-contrast* image ➜ histogram is *narrow*
  - *High-contrast* image ➜ histogram is *spread out*
  - In general, *image processing* alters the histogram

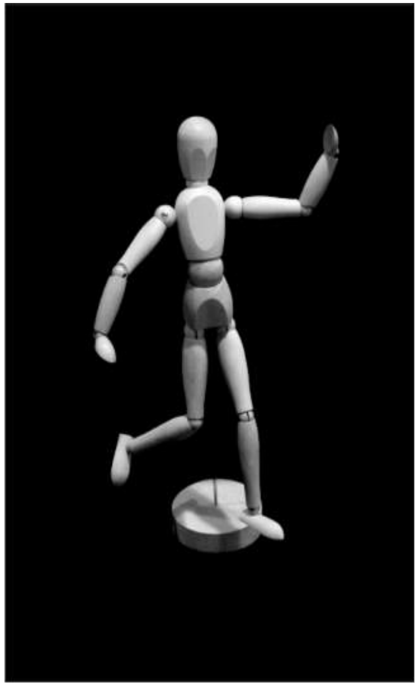


recall Useful functions

Clamping (or clipping)

- *Limit intensities* to a given interval $[a, b]$

$$f(p) = \begin{cases} a & \text{if } p < a \\ p & \text{if } a \leq p \leq b \\ b & \text{if } p > b \end{cases}$$



Windowing

- *Clamping* followed by *intensity stretching* to fill the full possible range $[0, M]$

$$f(p) = \begin{cases} 0 & \text{if } p < a \\ M \times \frac{p-a}{b-a} & \text{if } a \leq p \leq b \\ M & \text{if } p > b \end{cases}$$

recall Useful functions

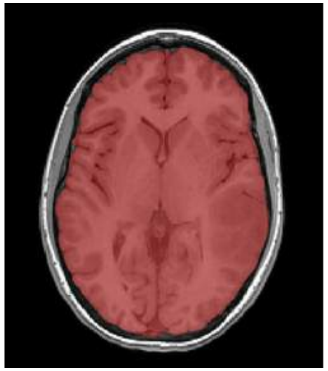
Thresholding

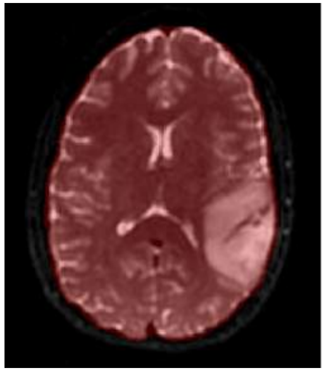
- Also called *image binarization*

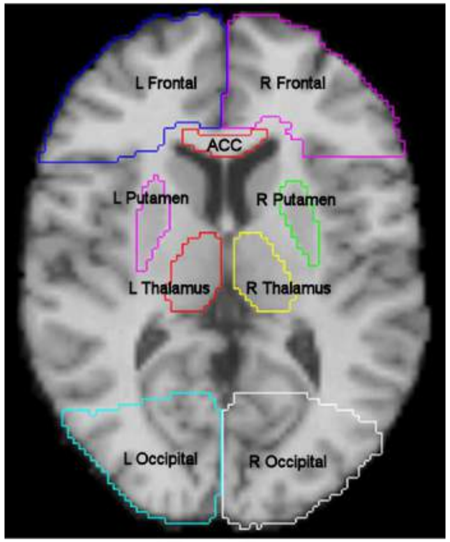
$$f(p) = \begin{cases} 0 & \text{if } p \leq a \\ 1 & \text{if } p > a \end{cases}$$

Notes

- Despite their simplicity, **binary images** are widely used in medical image processing
- Examples
  - *Constrain the processing* to a given portion of the image, e.g brain
  - *Regions-of-interest* (ROI) analysis

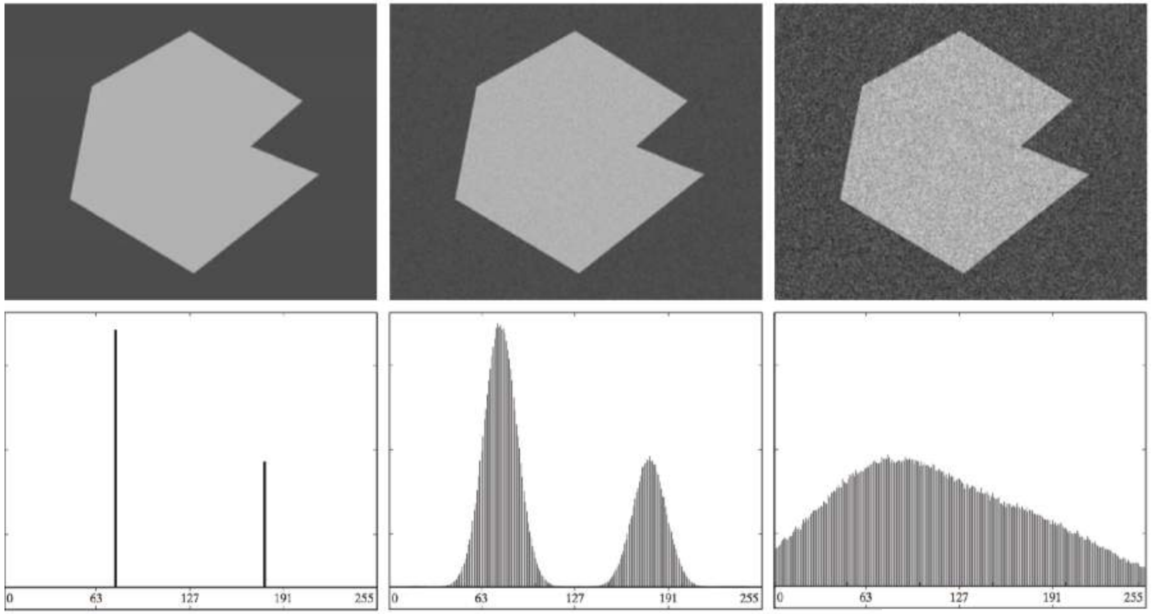


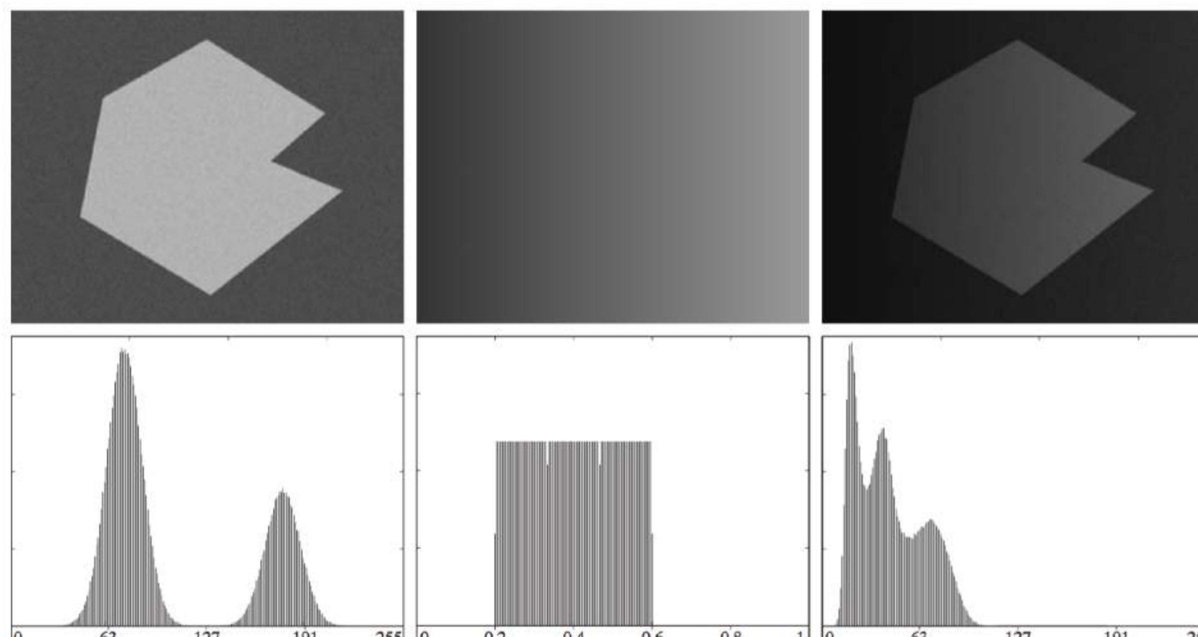




Noise and illumination

- Noise and illumination affect the histograms



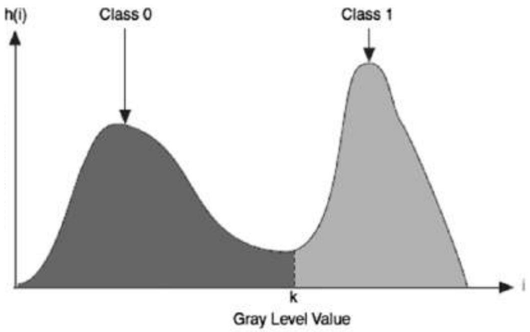


recall Automatic thresholding

Difficult to find the **proper threshold**

- Usually, there are **not only two regions**, e.g. foreground and background
- We will see **advanced segmentation tools** for the more general case

OTSU's method



- *Very basic* algorithm to **automatically binarize an image**
- Assumes that the image contains *exactly two regions* (i.e. bimodal histogram)
  - Actually, extensions exist for multiple regions
- Searches for the threshold that **minimizes the intra-class variance**:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

  - ω_0 and ω_1 are the *probabilities of the two classes* separated by a *threshold t*
  - σ^2_0 and σ^2_1 are *variances* of these two classes

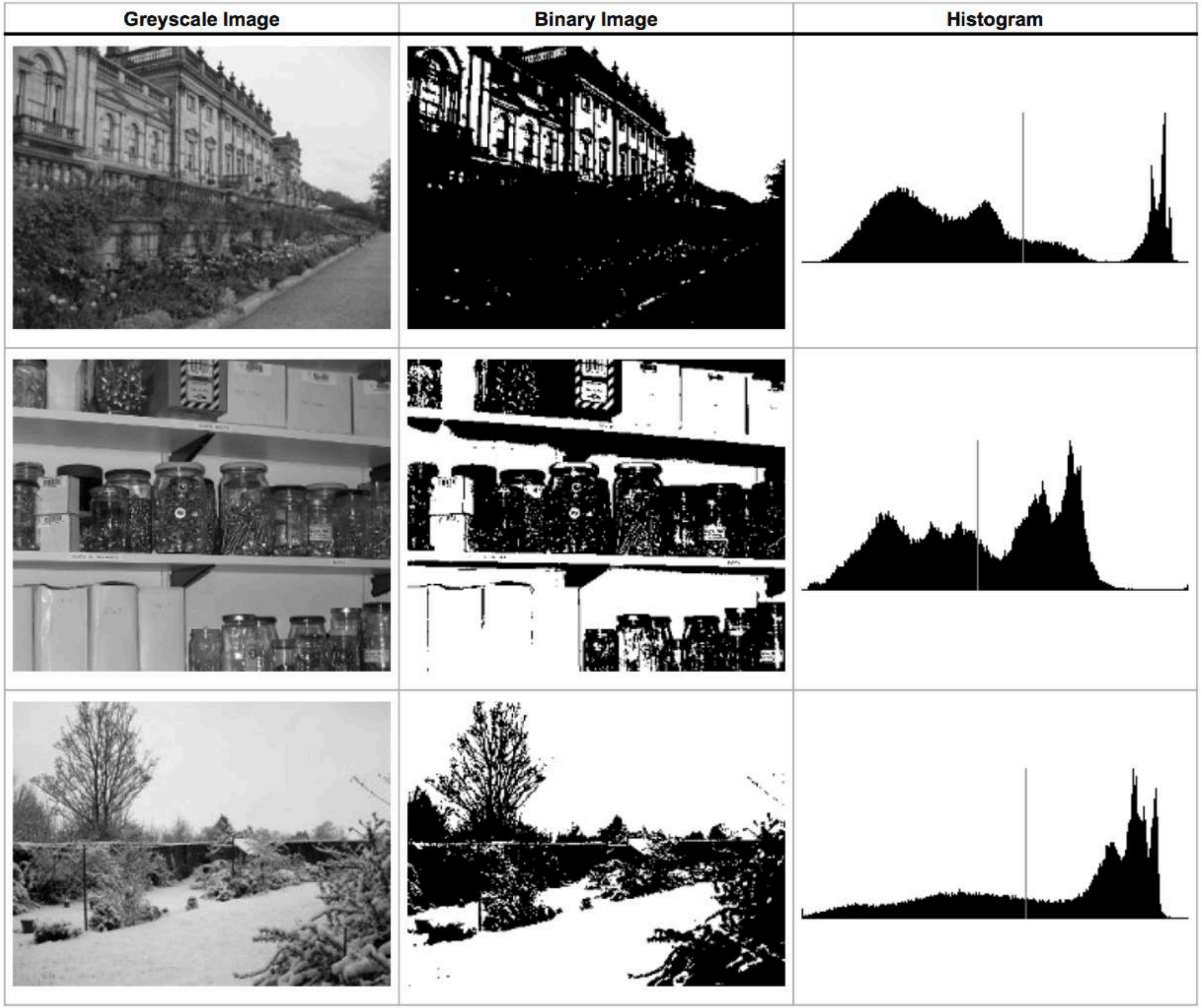
$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

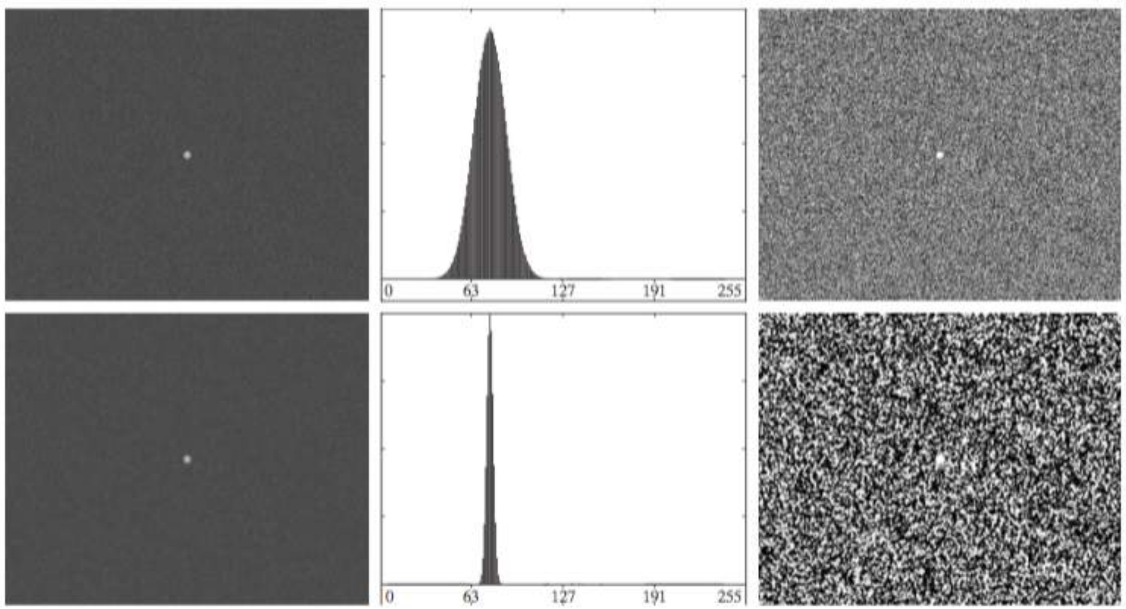
- **NB**: iterates through all the possible threshold values

recall Automatic thresholding

- Examples



Use edges to improve global thresholding



- **Why?**

Chances of selecting a "good" threshold are enhanced considerably if the histogram peaks are <u>tall</u>, <u>narrow</u>, <u>symmetric</u>, and <u>separated by deep valleys</u>.

- **Idea**

Use only the pixels near the edges between objects and background to construct the histogram —> the peaks will have approximately the same height.

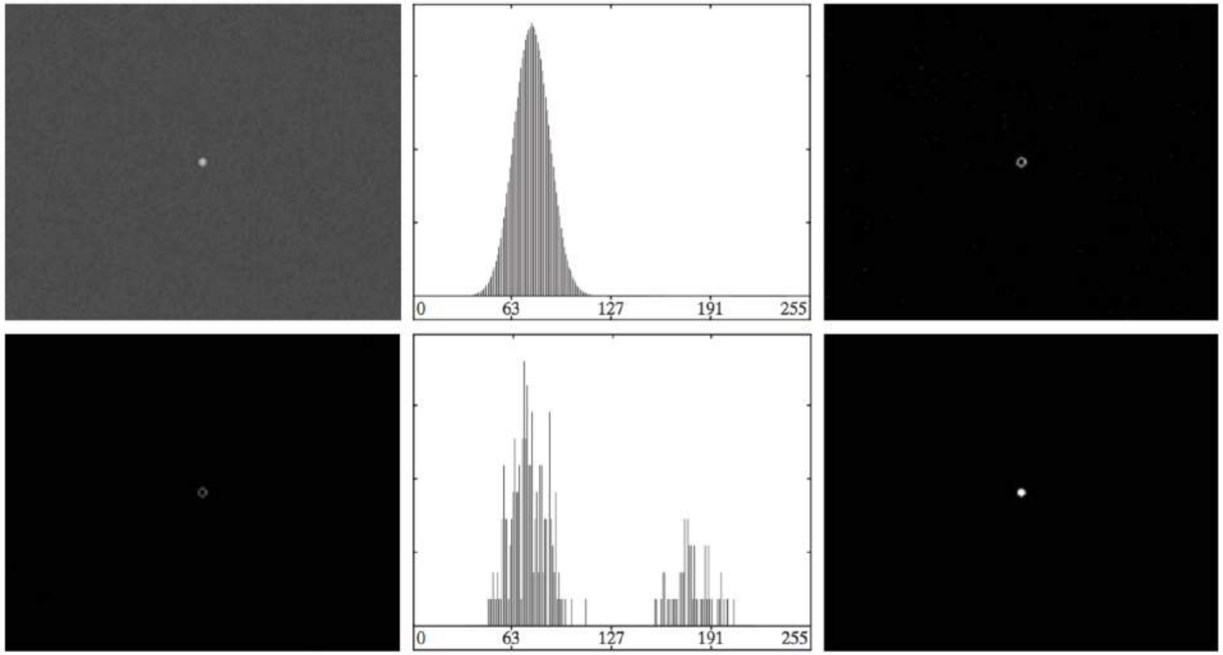
- **Problem**

We don't know the edges! —> We use the average value of the Laplacian is 0 at the transition of an edge.

Otsu's method using edges information

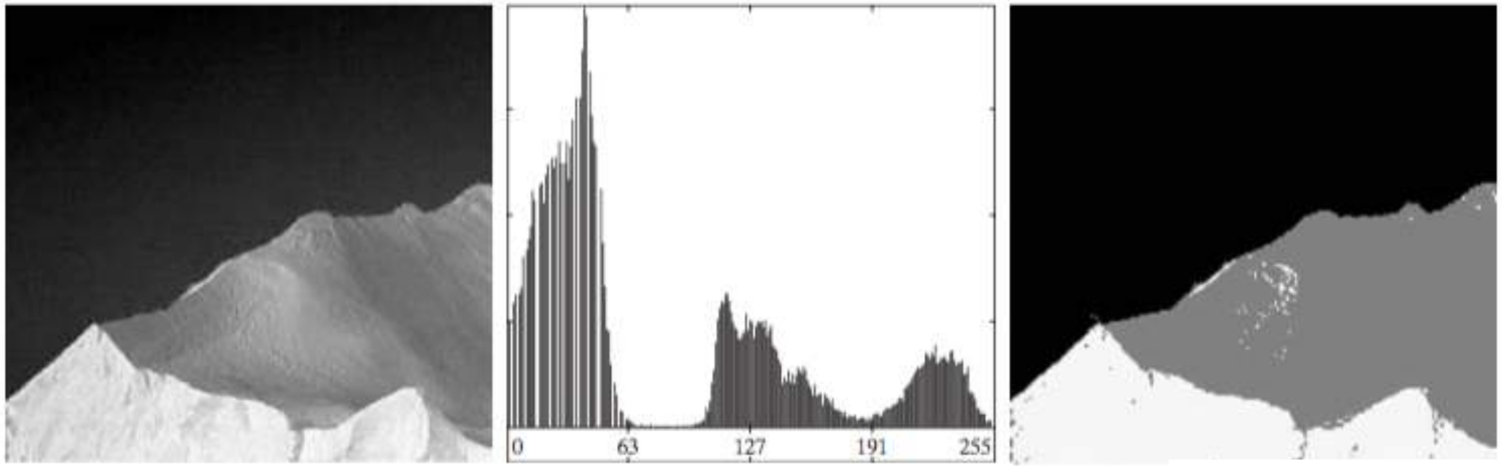
Algorithm

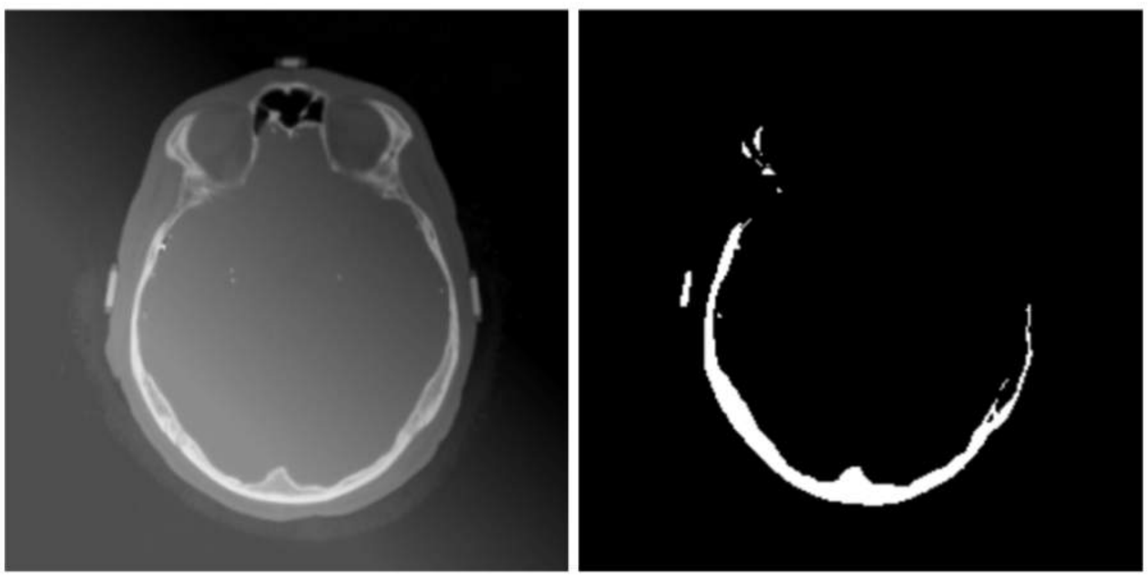
1. Compute an edge image of $f(x,y)$ using any method you have seen
2. Specify a threshold value T
3. Threshold the image of Step1 using the threshold T to produce a binary image $g_T(x,y)$ (*mask image*)
4. Compute an histogram using only the pixels in $f(x,y)$ that correspond to the locations of the 1-valued pixels in $g_T(x,y)$
5. Use the histogram in Step4 to segment $f(x,y)$ globally using Otsu's method.

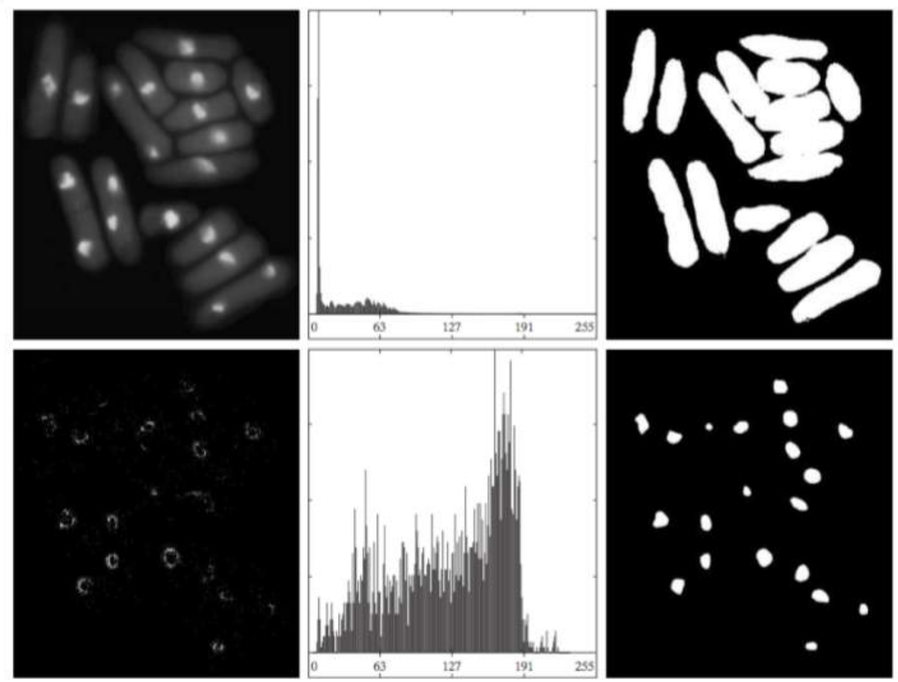


Noise and illumination

- Noise and not uniform illumination play a major role in the performance of a thresholding algorithm







Multiple thresholds

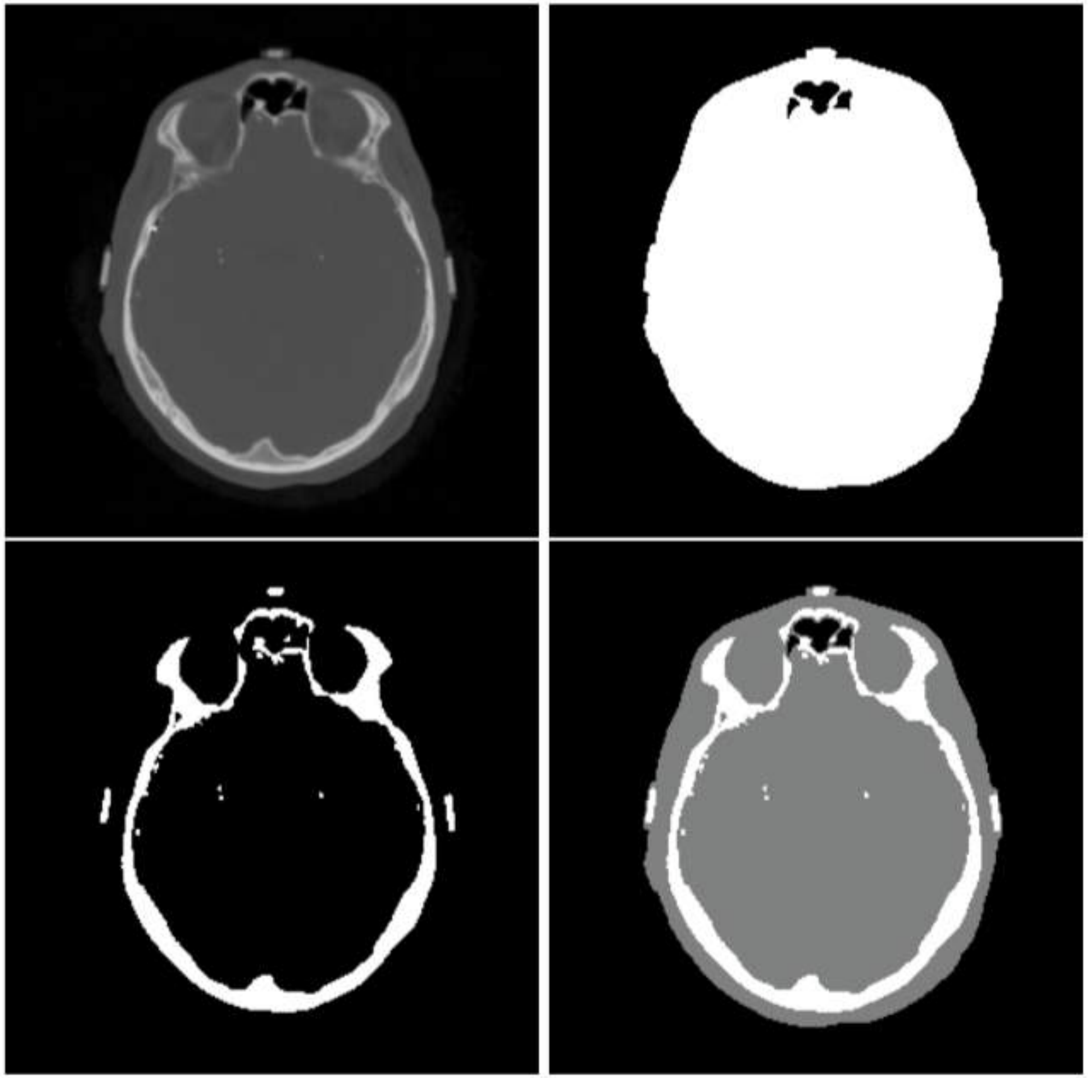
Why?

- To capture more features

How?

- Set more than one threshold simultaneously using the property of separability measure

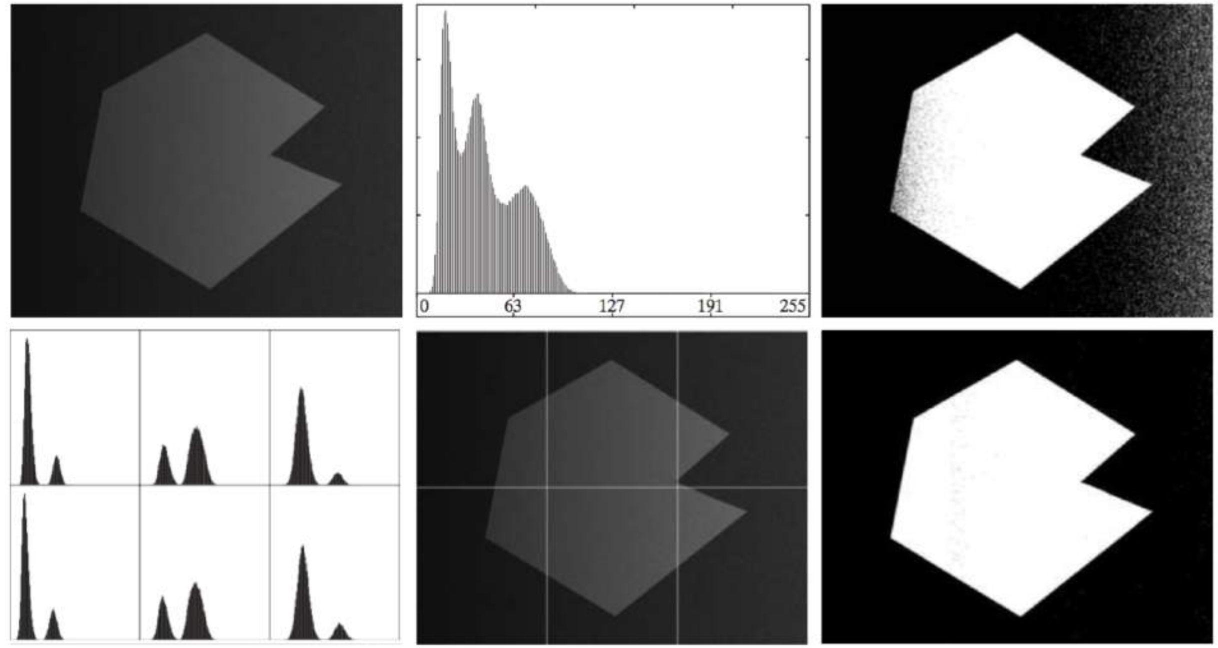
$$g(v) = \begin{cases} 0 & \text{if} \quad v < t_1 \\ 1 & \text{if} \quad t_1 \leqslant v < t_2 \\ 2 & \text{if} \quad t_2 \leqslant v < t_3 \\ \vdots & \vdots \quad \vdots \\ n & \text{if} \quad t_n \leqslant v. \end{cases}$$



Variable Thresholding

Image subdivision

- To compensate for non-uniformities in illumination and/or reflectance



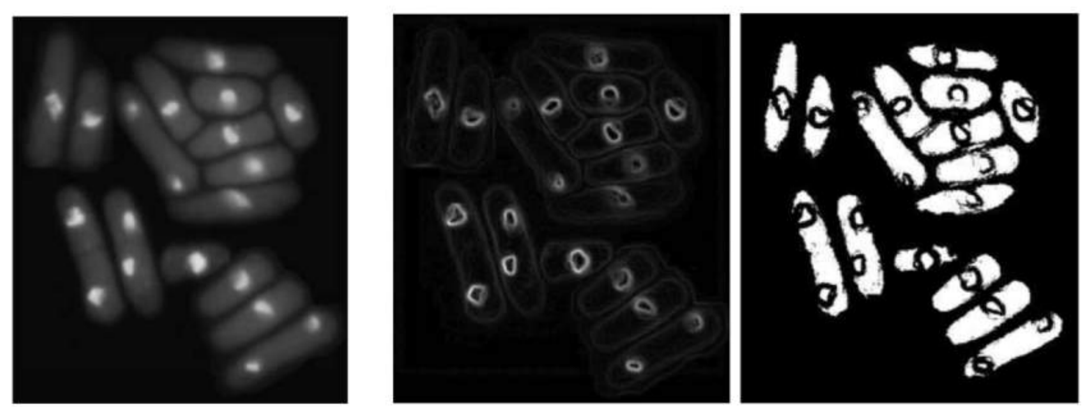
Local image properties

Find a threshold for every point (x, y) using local properties (such as mean and standard deviation) of the set of pixels contained in the neighbourhood of (x, y)

Example:

$$T_{xy} = a\sigma_{xy} + bm_{xy}$$

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T_{xy} \\ 0 & \text{if } f(x, y) \leq T_{xy} \end{cases}$$



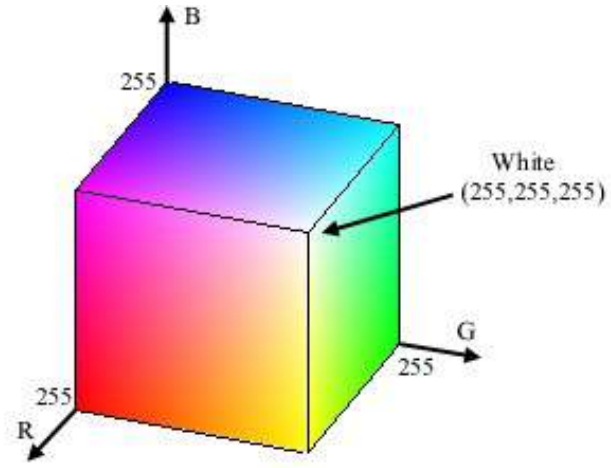
Multivariable thresholding

- If we consider a color image, then we have more than one variable to characterize each pixel (e.g. **RGB**).

multivariable thresholding

Each pixel is represented as a 3D vector $\mathbf{z} = (z_1, z_2, z_3)$. In order to perform a color threshold we then introduce the notion of **distance** $D(\mathbf{z}, \mathbf{a})$

- $D(\mathbf{z}, \mathbf{a}) > 0$ for all **z** not equal **a**
- $D(\mathbf{z}, \mathbf{a}) = 0$ if **z=a**
- $D(\mathbf{z}, \mathbf{a}) \leq D(\mathbf{z}, \mathbf{x}) + (\mathbf{x}, \mathbf{a})$
- $D(\mathbf{z}, \mathbf{a}) = T$ defines a volume

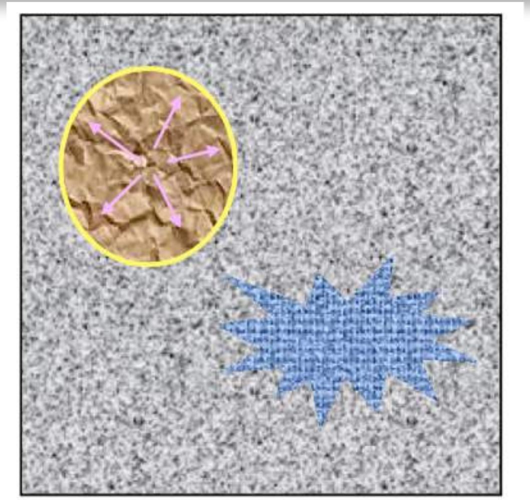


Example: Euclidean distance

$$D(\mathbf{z}, \mathbf{a}) = ||\mathbf{z} - \mathbf{a}|| = \left[(\mathbf{z} - \mathbf{a})^T \cdot (\mathbf{z} - \mathbf{a})\right]^{1/2}$$

Region based approaches

Region-based approach

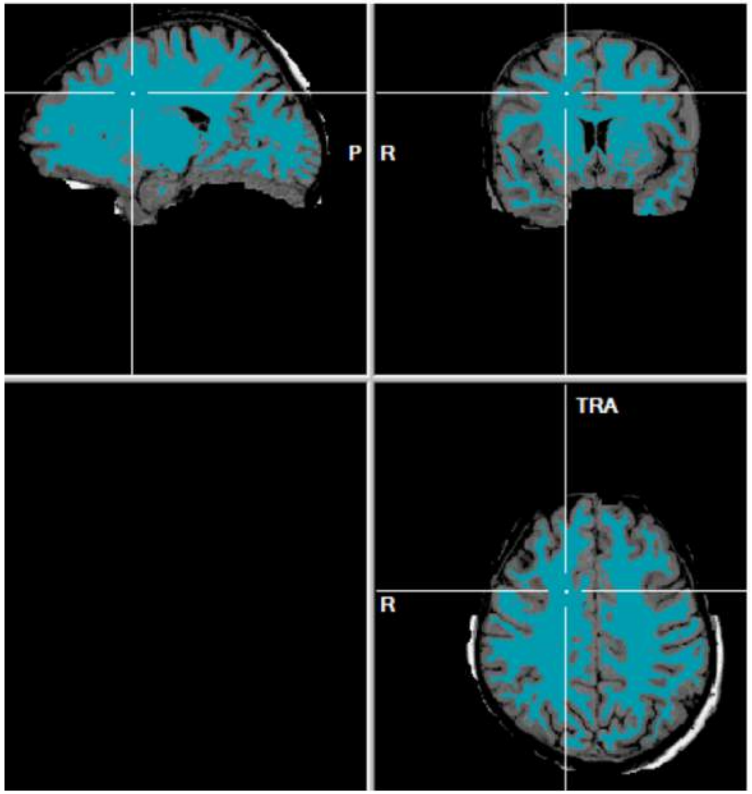


What is a region?

A group of connected pixels with **similar** properties

Idea

Pixels that correspond to an object are grouped together and marked



Principles

- Similarity
  - Gray value differences
  - Gray value variance
- Spatial proximity
  - Euclidean distance
  - Compactness of a region

Region-based segmentation

Main Goal:

Partition an image *I* into regions R_i

Formulation:

- Completeness. Every pixel must be in a region $\bigcup_{i=1}^{n} R_i$
- Connectedness. The points of a region must be connected in some sense
- Disjointness. Region must be disjoint $R_i \bigcap R_j = \emptyset \quad \forall i = 1, 2, ..., n$
- Satisfiability. Pixels of a region must satisfy at least one common property P $\quad P(R_i) = TRUE \quad \forall i = 1, 2, ..., n$
- Segmentability. Different regions satisfy different properties! $\quad P(R_i \bigcup R_j) = FALSE \quad \forall i = 1, 2, ..., n$

Main methods of Region based Segmentation

- Region Growing

- Split and Merge

- Clustering

Region growing segmentation

Principle:

To group pixels or sub-regions into larger regions based of pre-defined criteria.

Method:

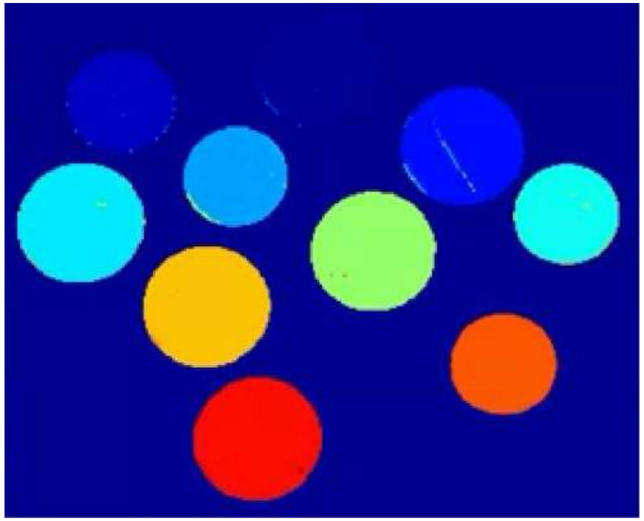
Select a set of pixels ("**seed points**") of potential regions and try to grow them by appending to each seed point those neighbouring pixels that have ***similar*** properties (such as gray level, texture, color, shape, …) till the pixels being compared are too dissimilar.

Region growing algorithm

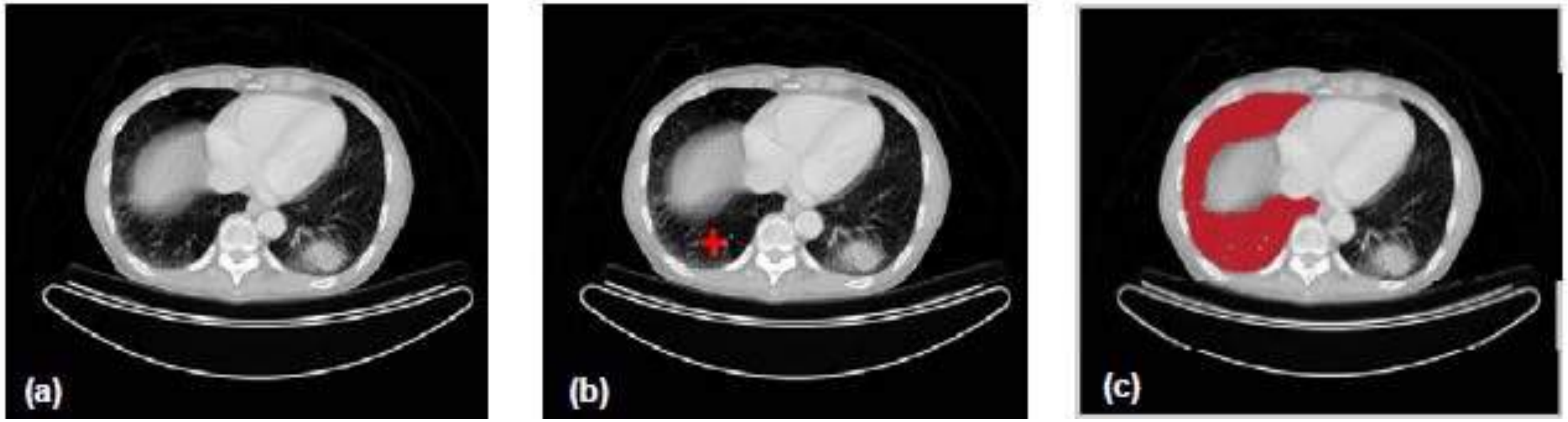
Algorithm based on 8-connectivity

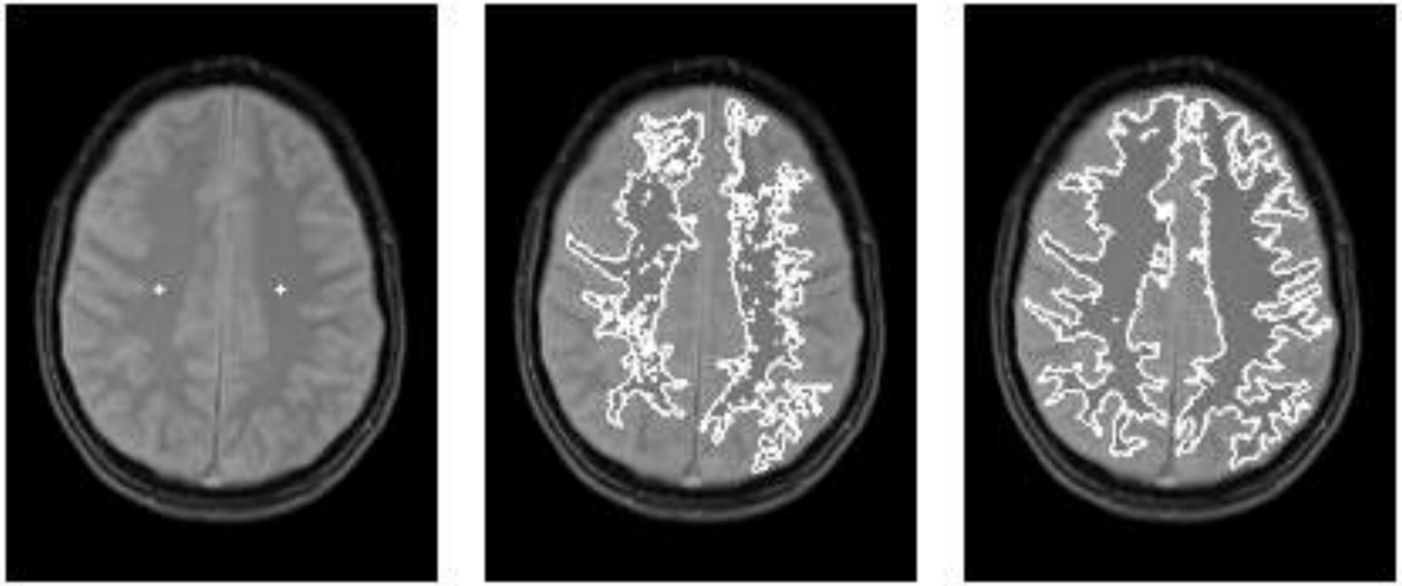
$f(x,y)$ - the input image
$S(x,y)$ - seed array containing 1 at the location of seed points and 0 elsewhere (size(*S*)=size(*f*))
P - predicate to be applied at each location (x,y)

1. Find all connected components in $S(x,y)$ and erode each connected component to one pixel
2. Form an image $f_P(x,y)$ such that at a pair coordinates (x,y) $f_P(x,y)=1$ if the input image satisfies the given predicate P, otherwise $f_P(x,y)=0$
3. Let $g(x,y)$ be an image formed by appending to each seed point in $S(x,y)$ all the 1-valued points in $f_P(x,y)$ that are 8-connected to that seed point
4. Label each connected component in $g(x,y)$ with a different region label obtaining the **segmented image**



Region growing examples





Region growing segmentation comments

Advantages

- It is a **fast** method
- It is conceptually **simple**

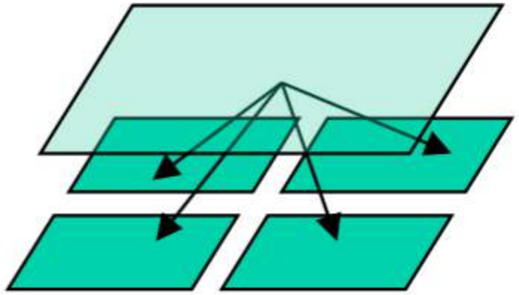
Disadvantages

- **Local method**: no global view of the problem
- **Application specific**: may need user to select starting points and different choices of seeds may give different segmentation results
- Problems can occur if the seed point lies on an edge
- May yield misleading results if connectivity properties are not used
- **Stopping rule** difficult to be defined
- Algorithm is very **sensitive to noise**

Region splitting and merging

- Region splitting:

  Starts with the whole image as a single region and subdivides it into subsidiary regions recursively while a condition of homogeneity is not satisfied.

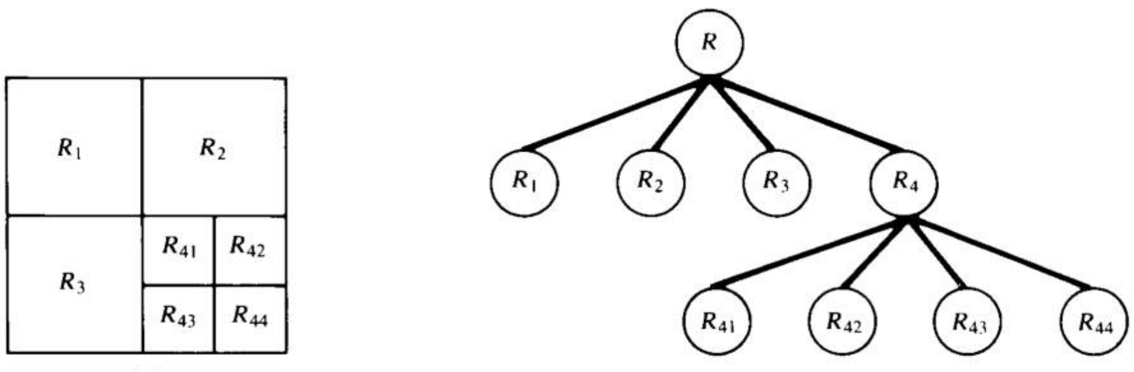


- Region merging:
  - Is the opposite of region splitting, and works as a way of **avoiding over-segmentation**
  - Starts with small regions (e.g. 2x2 or 4x4 regions) and merge the regions that have similar characteristics (such as gray level, variance, …).

Splitting and Merging

Algorithm

1. Split into four disjoint quadrants any region R_i for which $P(R_i) = FALSE$



2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $P(R_j \bigcup R_k) = TRUE$

3. Stop when no further splitting or merging is possible

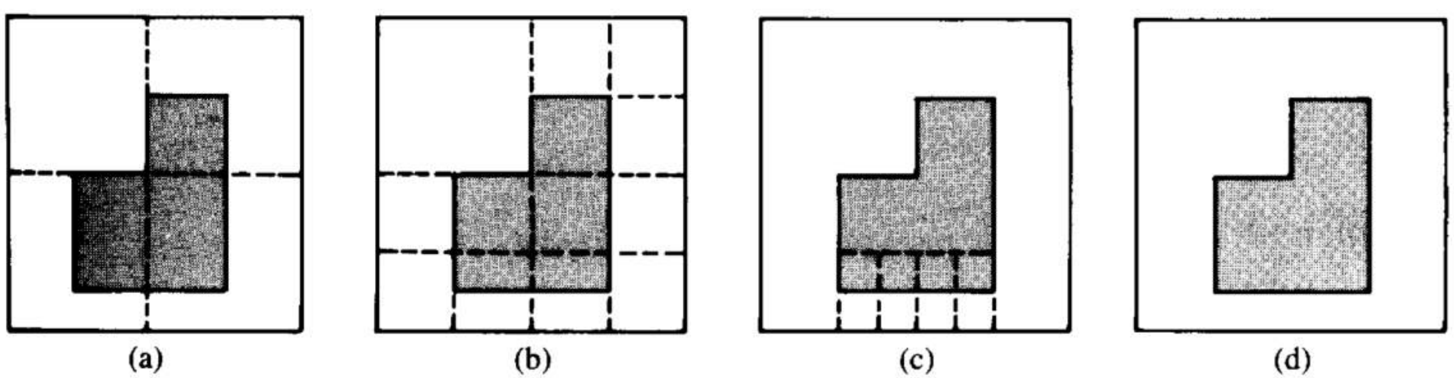
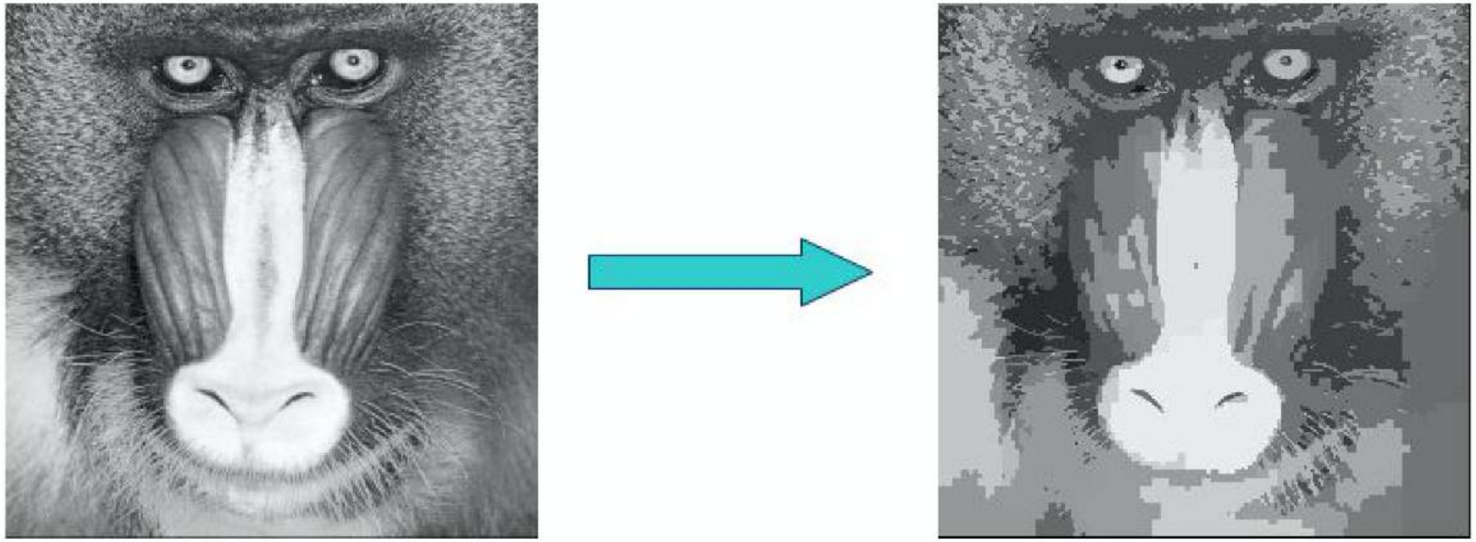


Figure 7.38 *Example of split-and-merge algorithm. (From Fu, Gonzalez, and Lee [1987].)*

Observations on splitting and merging



- Tries to eliminate the need for seeds

  Sort of an all purpose algorithms

- Still requires a predicate *P*

  $P(R_i)$ needs to be fairly generic

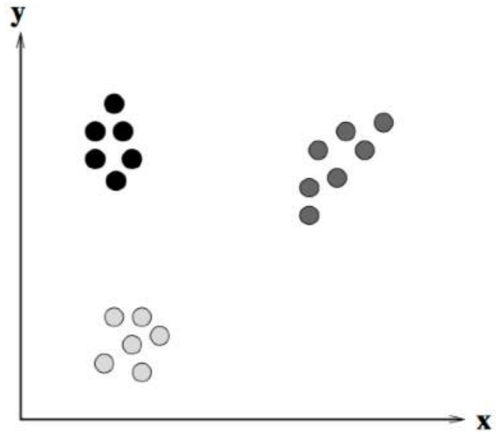
- The key to this algorithm is how to merge the regions!

Clustering

Process of partitioning a set of pattern vectors into subsets called ***clusters***.

- Principles:
  - Used any feature that can be associated to a pixel to group them (intensity values, RGB values, texture measurements, etc. …)
  - Once pixels have been grouped into clusters, find connected regions using connected components labeling.
- Least square error measure to find how "close" are the pixels

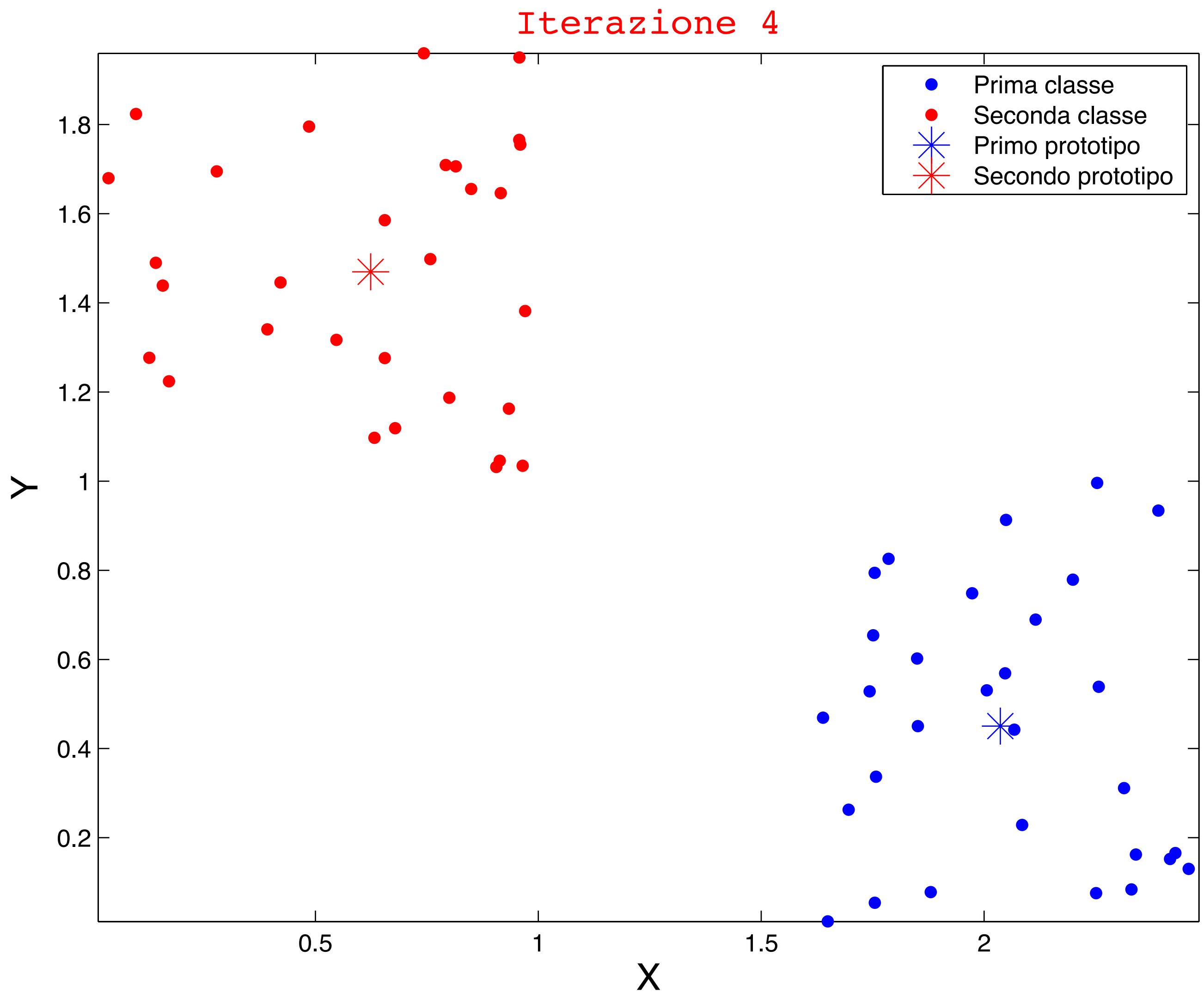
$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - m_k||^2.$$



Clustering example

- Using Euclidean distance

$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - m_k||^2.$$



K-Means algorithm

- Classical algorithm: **K-Means**
  1. Set ic (iteration count) to 1
  2. Choose randomly a set of K means $m_1(1), \ldots, m_K(1)$
  3. For each vector x_i compute $D(x_i, m_k(ic))$ for each $k = 1, \ldots, K$ and assign x_i to the cluster C_j with the nearest mean
  4. Increment ic by 1 and update the means to get a new set $m_1(ic), \ldots, m_K(ic)$
  5. Repeat steps 3 and 4 until $C_k(ic) = C_k(ic+1) \quad \forall k$

Figure 10.4: Football image (left) and K=6 clusters resulting from a K-means clustering procedure (right) shown as distinct gray tones. The six clusters correspond to the six main colors in the original image: dark green, medium green, dark blue, white, silver, and black.

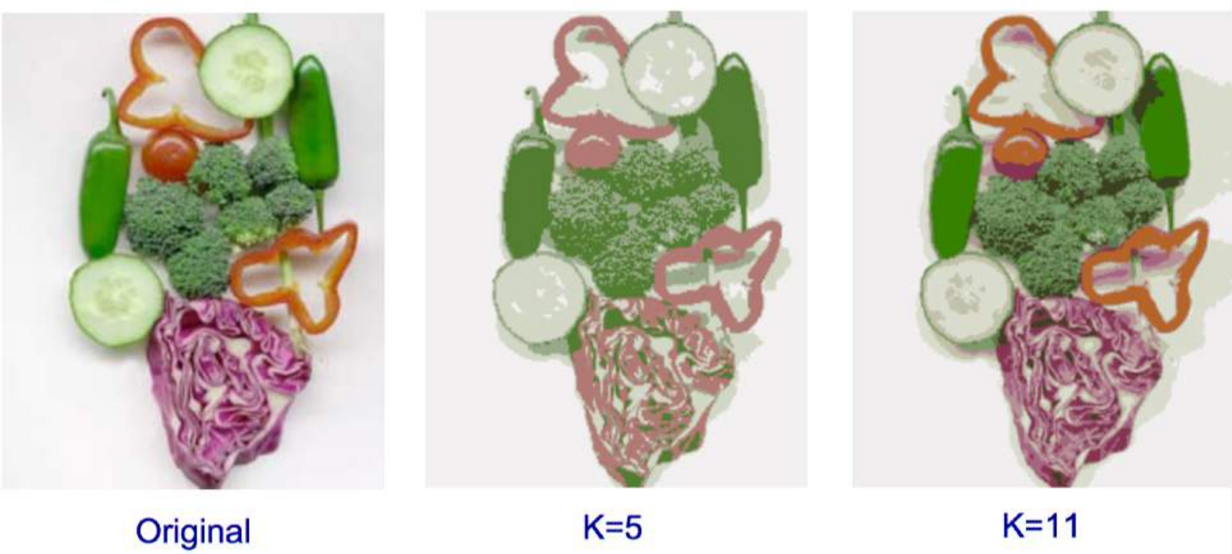
K-Means Limits

Advantages

- It is a **fast** method
- It is conceptually **simple**
- **Convergence** is guaranteed

Disadvantages

- **Exact number** of **clusters** must be provided
- Features with larger scales dominate clustering



Possible solution

Introduce the concept of "**uncertainty**" using a *probabilistic* (*fuzzy* or *probabilistic*) formulation

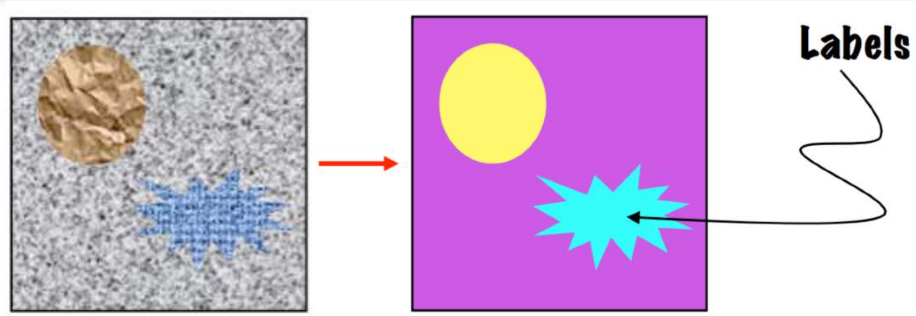
Segmentation

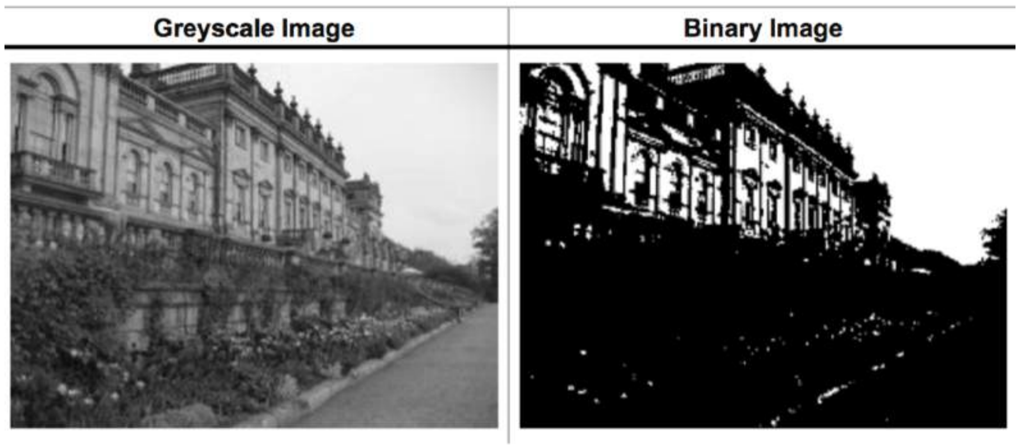
Part II

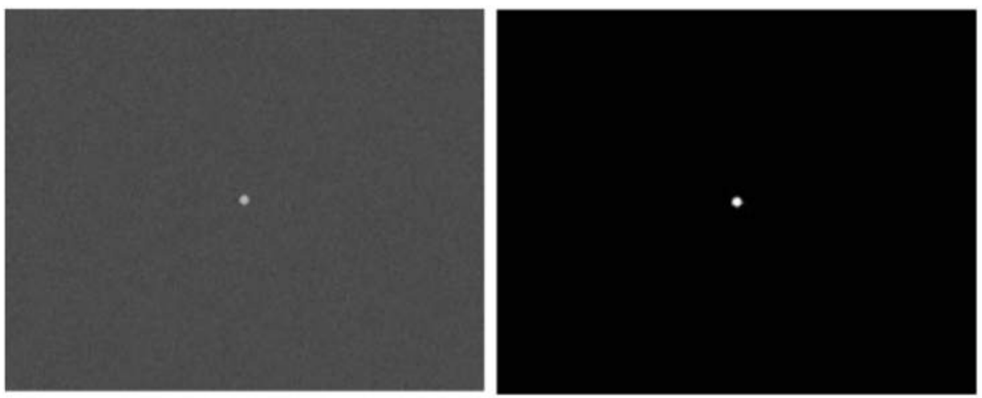
Summary Part I

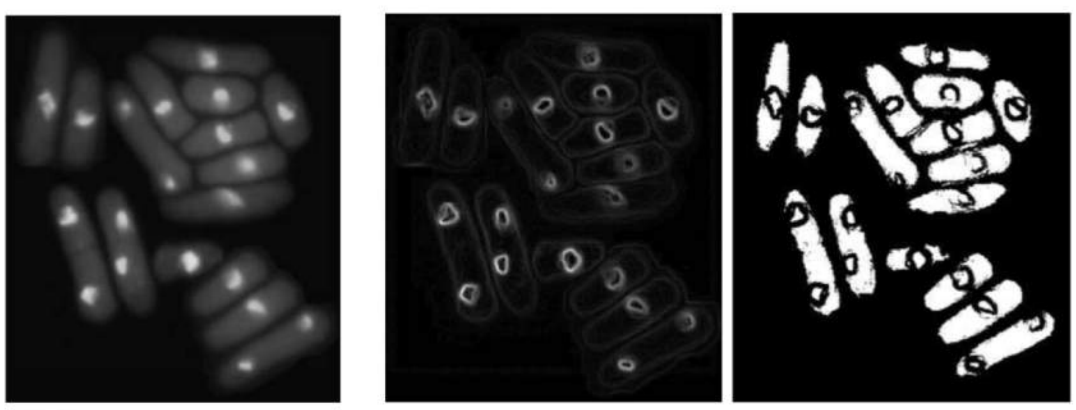
Threshold based approaches

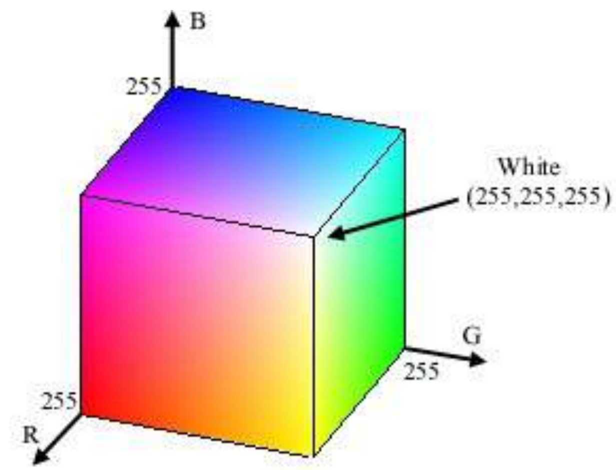
- Otsu’s method
- Otsu’s method with edge detection
- Variable thresholding
- Multivariable thresholding





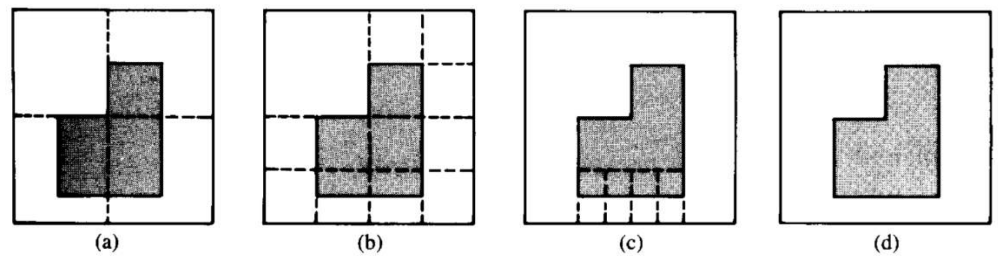


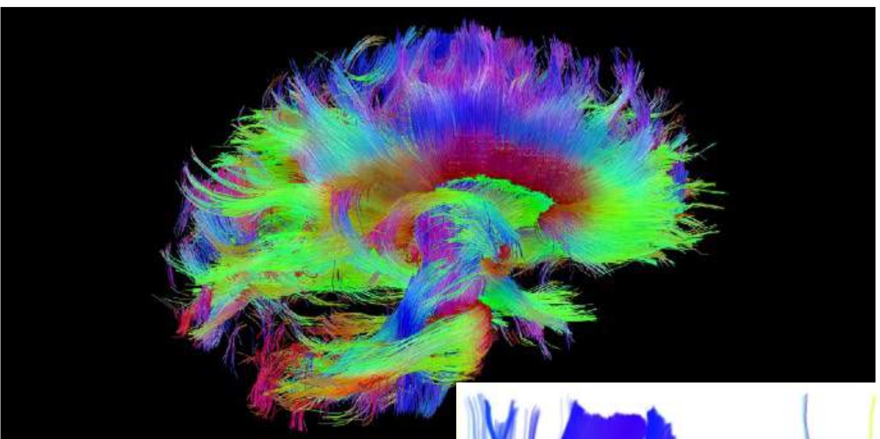


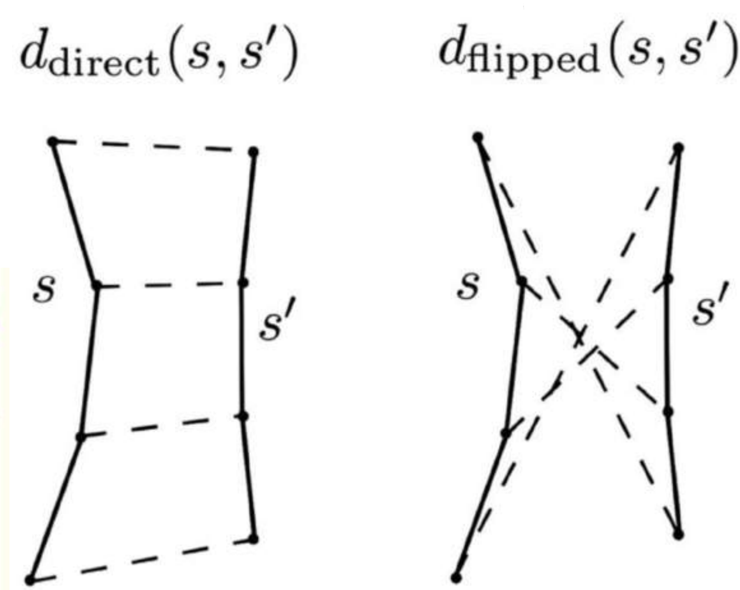


Summary Part I

- Region-based approaches
  - Region growing
  - Split and Merge
  - Clustering



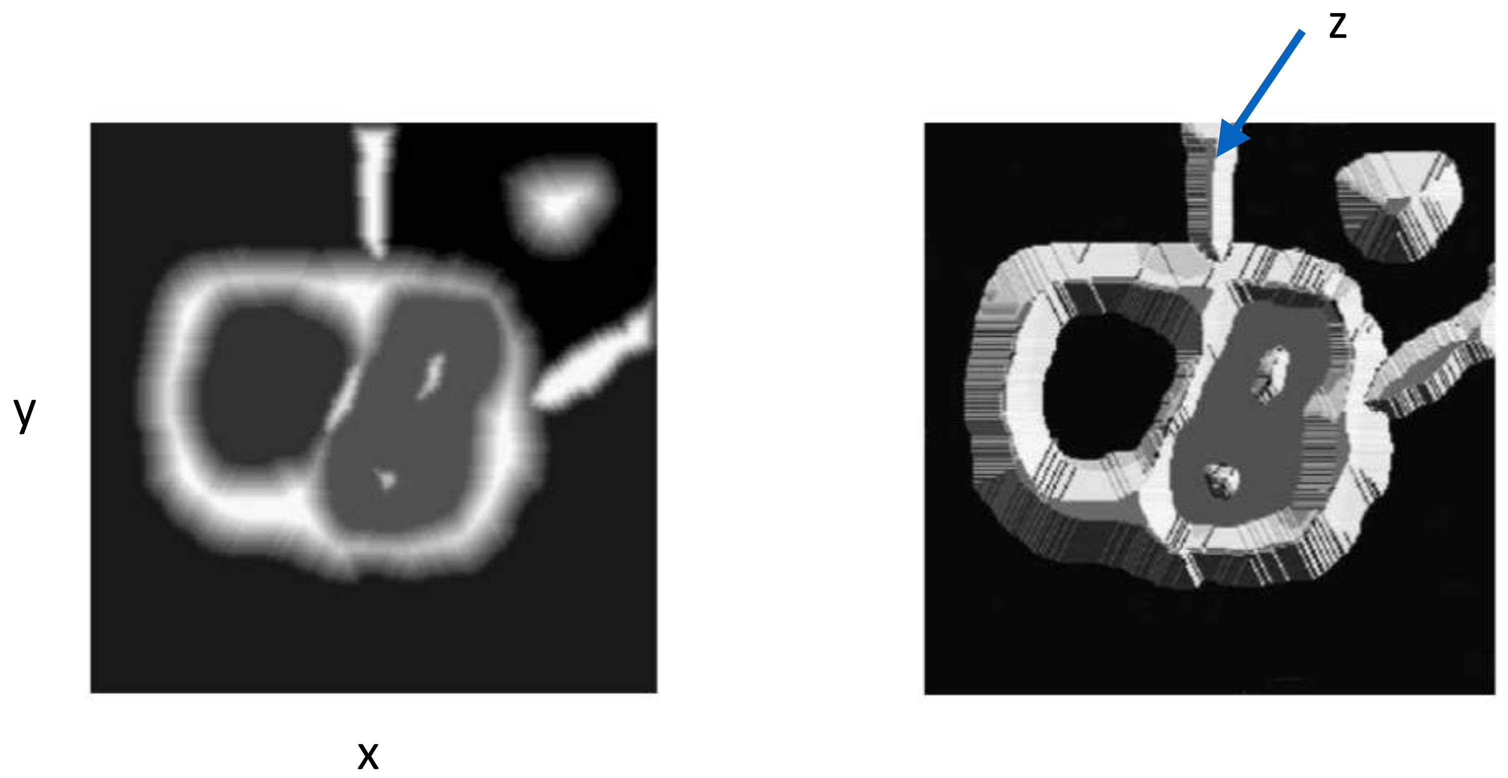




Morphological watershed

Morphological watershed

- Visualize a 2D image in 3-dimensions:
  - 2 spatial coordinates
  - intensity of the pixel as third coordinate



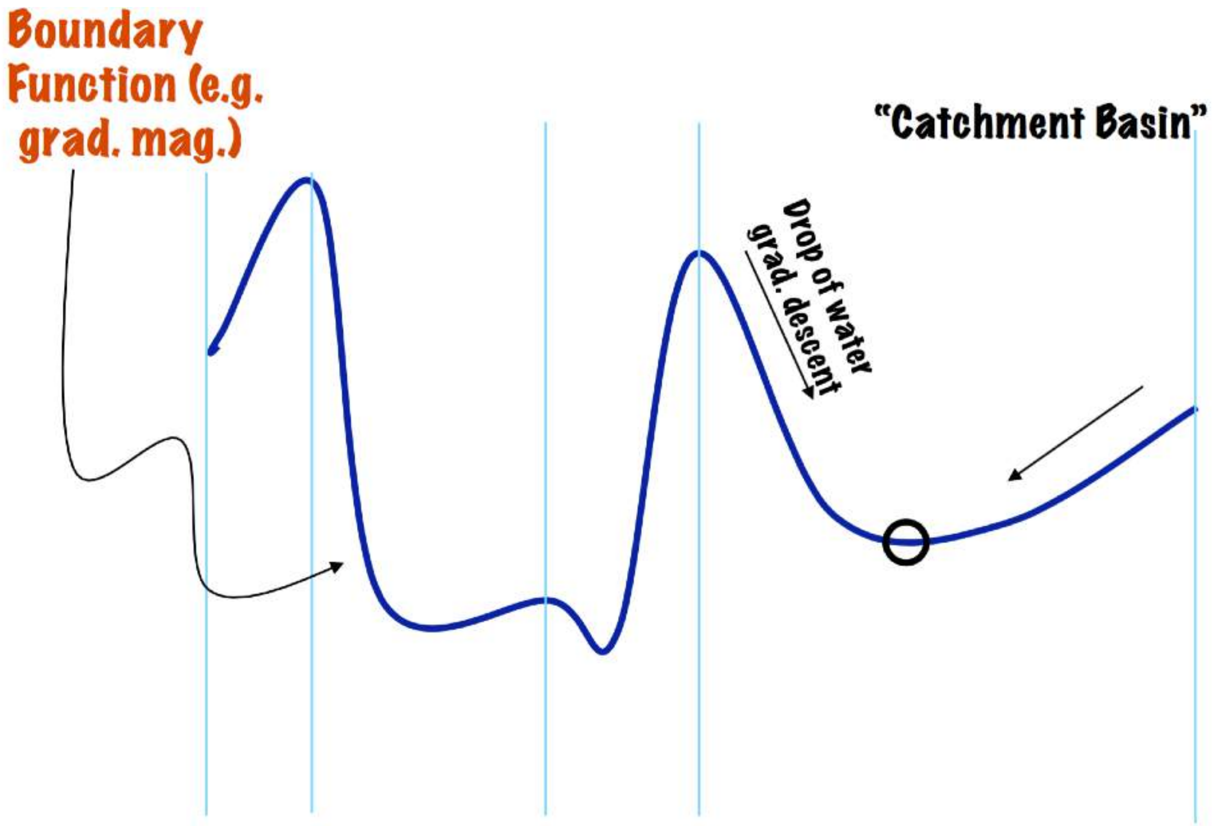
Watershed

Topographic interpretation

- points belong to a regional ***minimum***
- points at which a drop of water would fall with certainty to a single minimum
  watershed
- points at which water would be equally likely to fall to more than one such minimum
  watershed lines

Aim

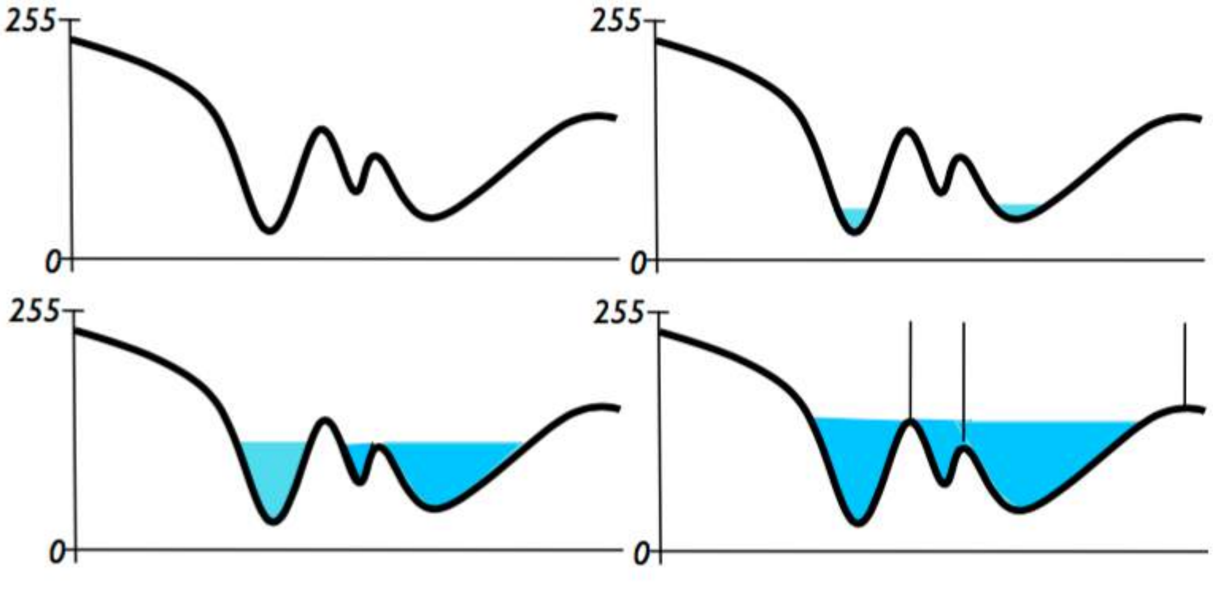
Identify all the watershed lines to get a segmentation

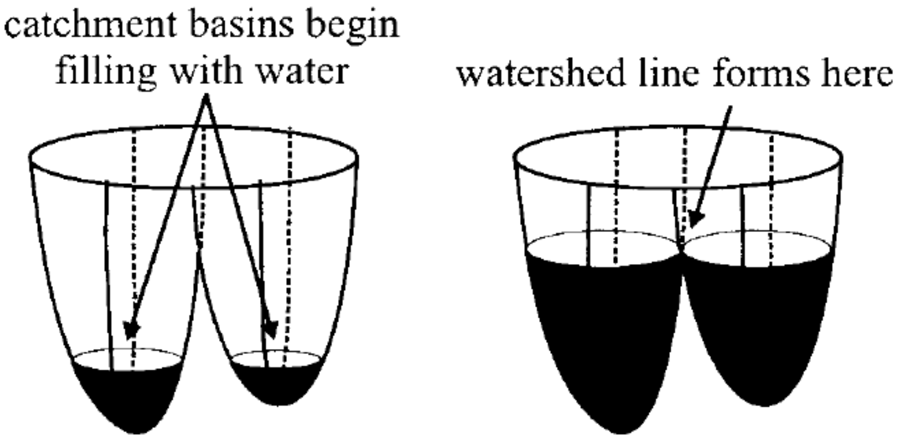


Watershed principles

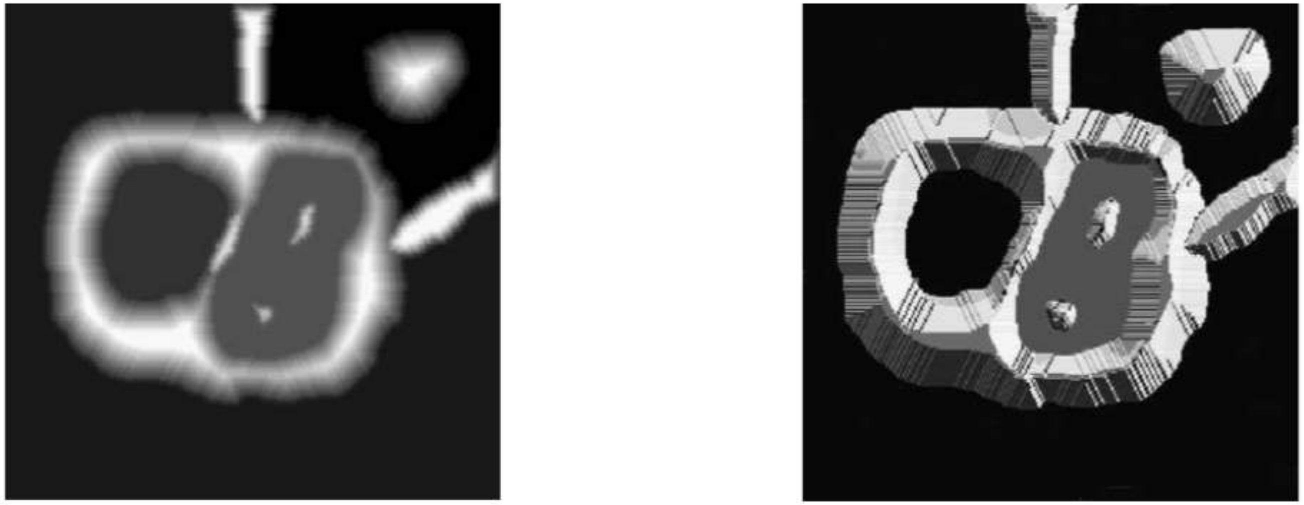
Underlying idea

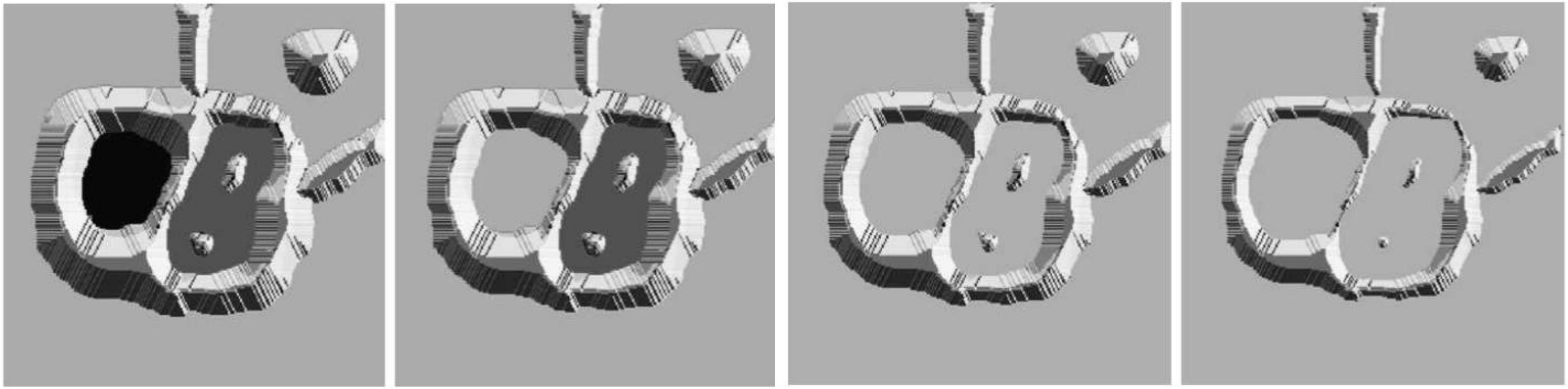
- Immagine that a hole is done through each local minimum so that the entire topography is flooded with water rising through the holes at uniform rate
- When rising water in adjacent catchment basis is about to merge, a dam is built up to prevent merging. These dam boundaries correspond to watershed lines





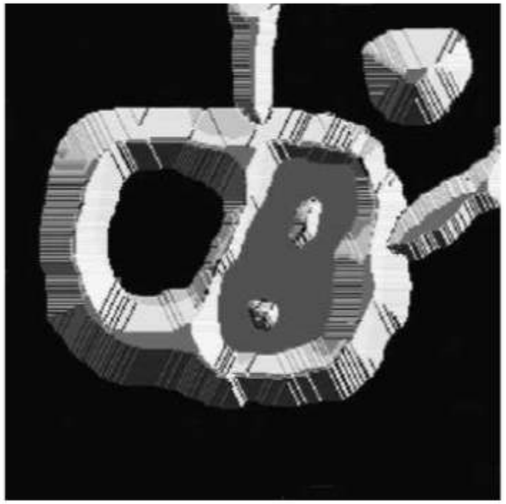
Watershed principles

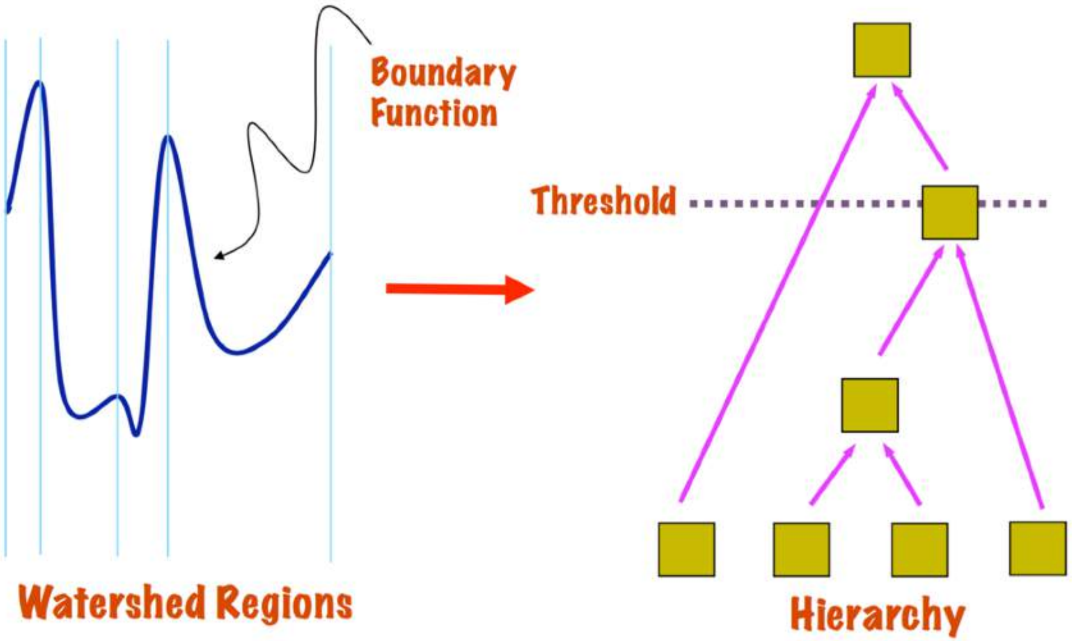




Watershed algorithm: Basic steps

- Start with all the pixels with the lowest possible values
  - These pixels are the *local minima* “through which we start to flood water”
- For each group of pixels of intensity k
  - **If** the point is adjacent to exactly one existing region, add this pixel to that region
  - **Else**
    - **if** the point is adjacent to more than one existing regions, mark as boundary
    - **Else** start a new region

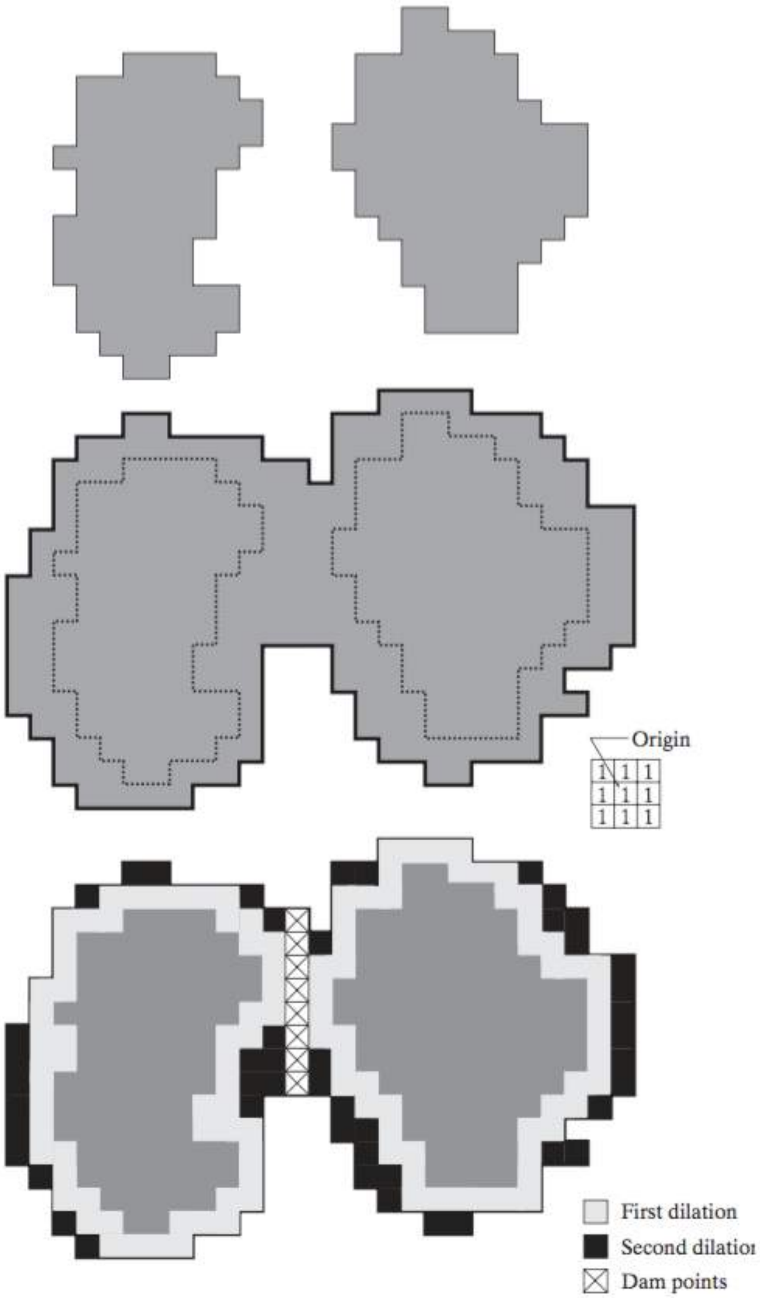




Watershed algorithm: dam construction

- Principle
  - Prevent the merging of water from two catchments basins

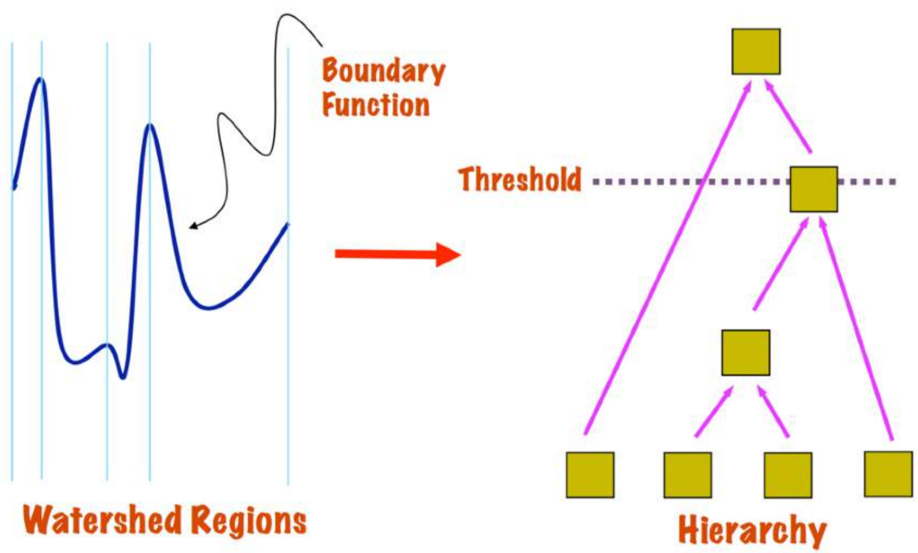
- Underlying operation
  - Binary morphological ***dilation***



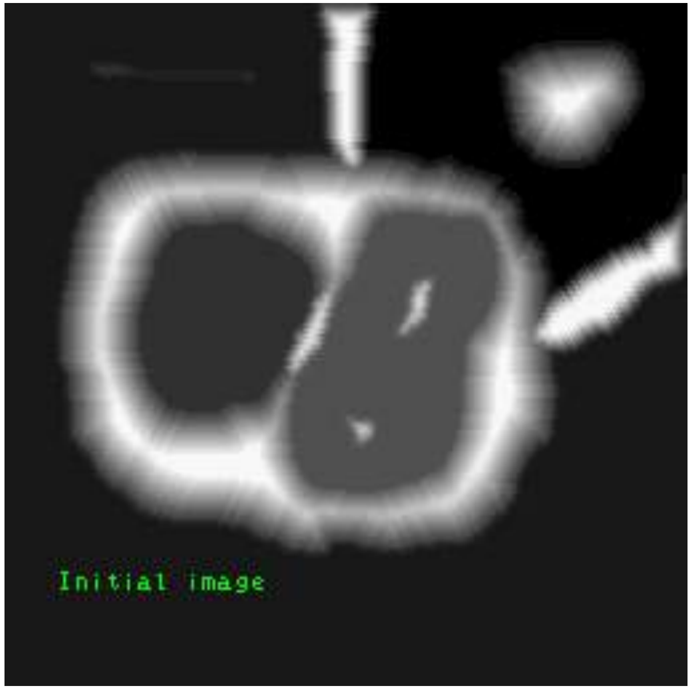
Watershed algorithm: dam construction

Dam construction sub-algorithm

- Set pixels with minimum gray level to 1 and the rest to 0
- At each iteration we flood the 3D topography from below and the pixels covered by the rising water are set to 1 and the other 0
- If at flooding step n-1 there are two connected components $(C_1 \text{ and } C_2)$ and at step n there is only one connected component C, then
  - Compute $q = C \bigcap (C_1 \bigcup C_2)$ to find the points that may have caused the merging
  - For each point on the boundary of C_1 and C_2 re-perform the dilation and find which of them gives the same point in q
  - Mark the the found points with a number greater then the maximum intensity value of the image



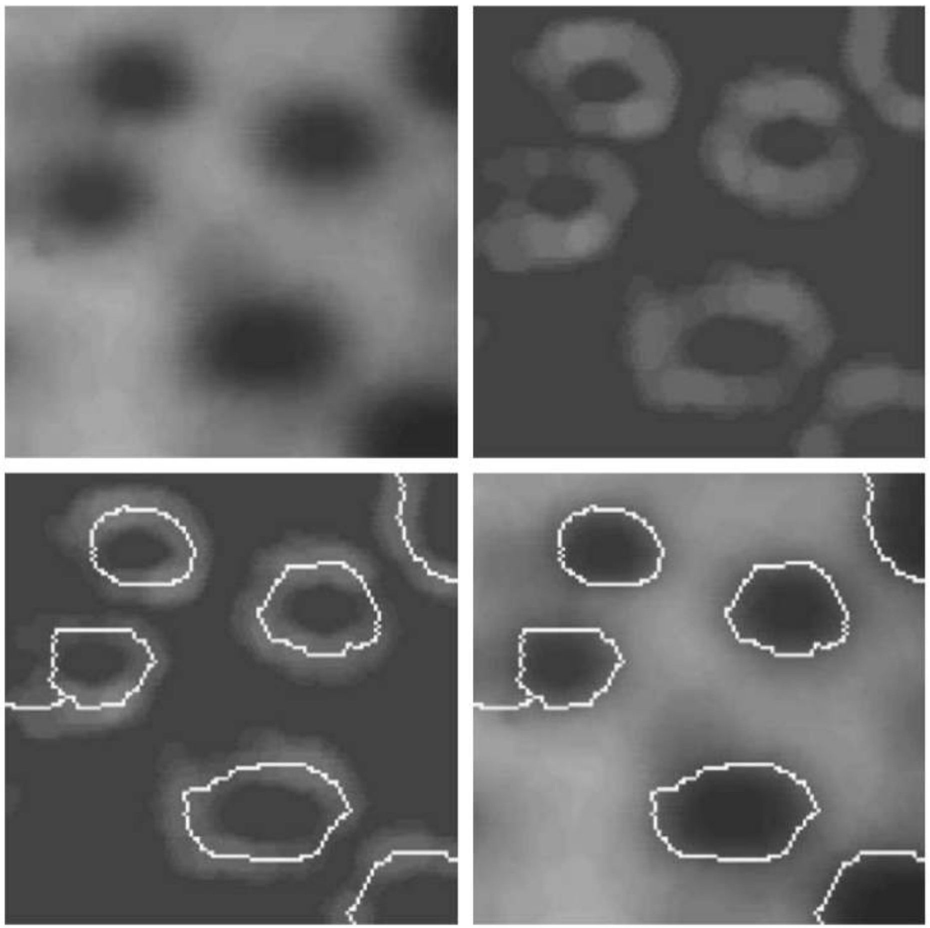
Watershed animation

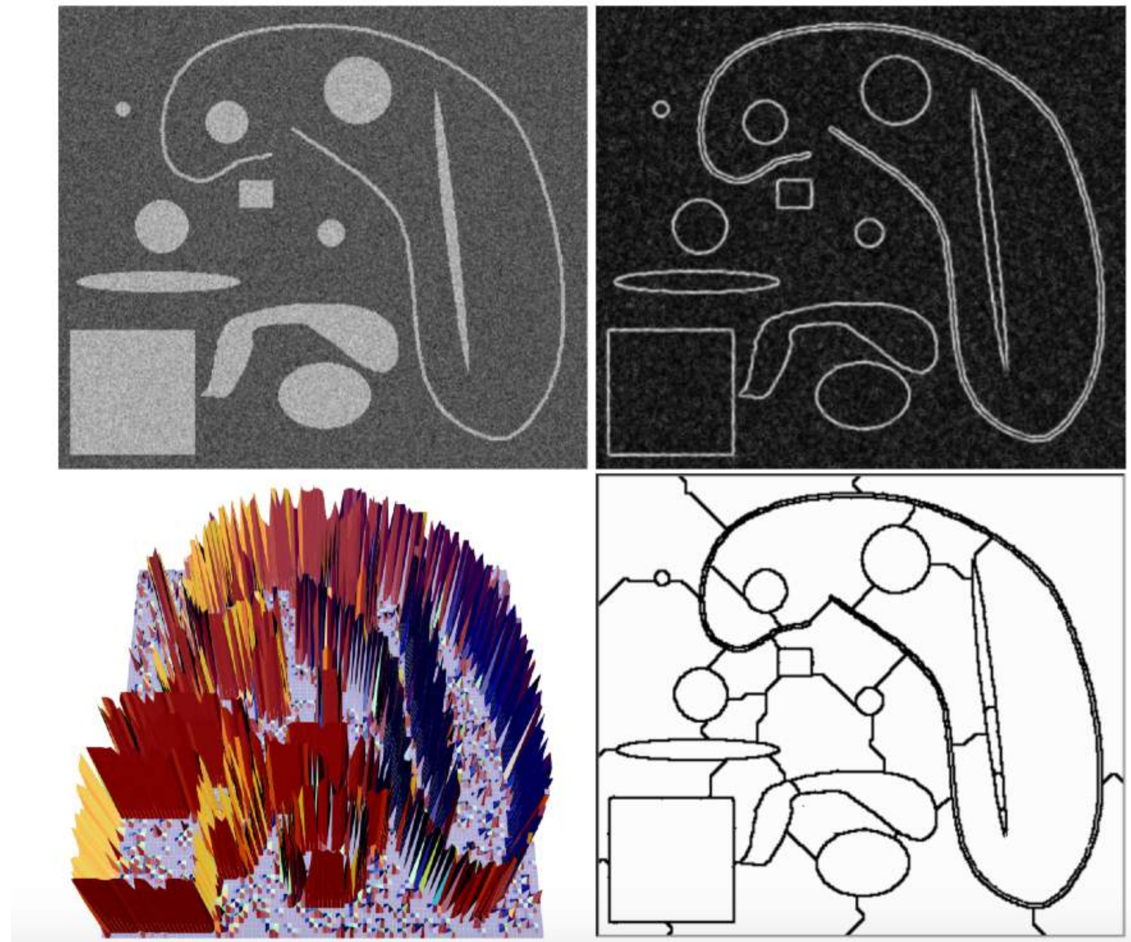


Taken from http://cmm.ensmp.fr/~beucher/wtshed.html

Example of watershed segmentation

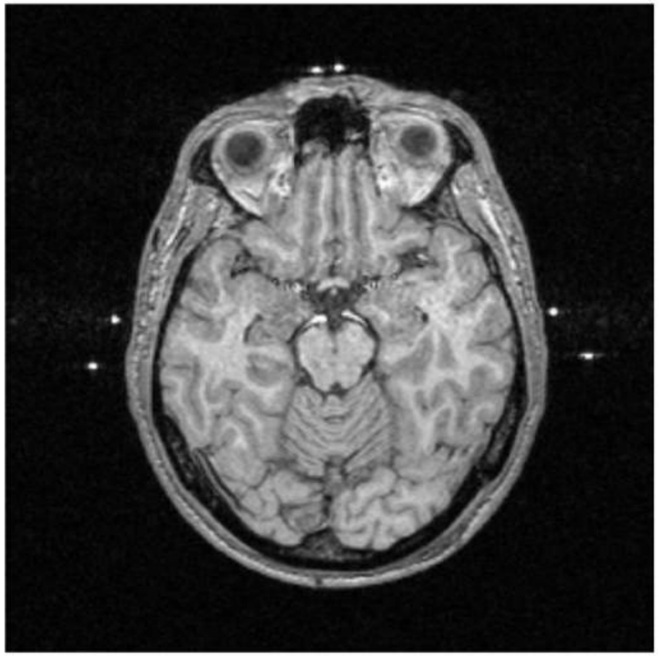
- Highly used with gradient images

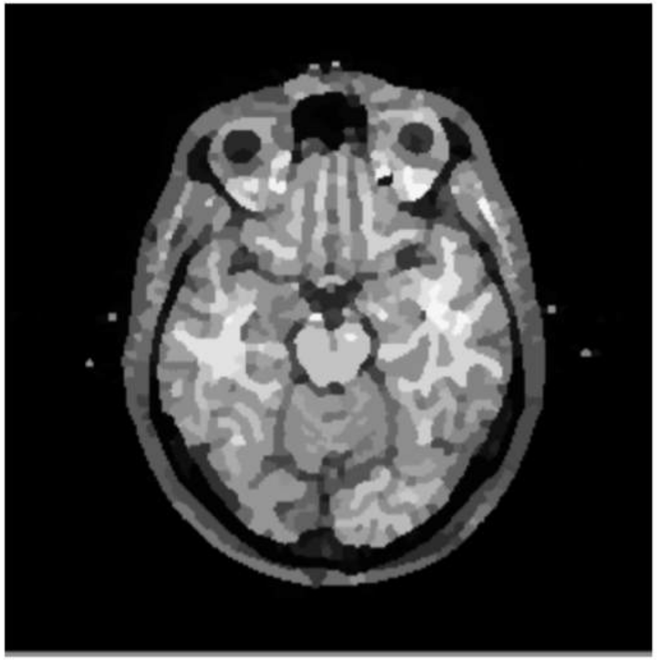




Watershed segmentation properties

- Non local (regions can leak!)
- Boundary based
  - Poor in low contrast data
  - Very sensitive to noise
- Low level (pixel based)
  - Lack of shape model
  - May leed to over segmentation
- Preprocessing step
  - Necessary for reliable boundary measure





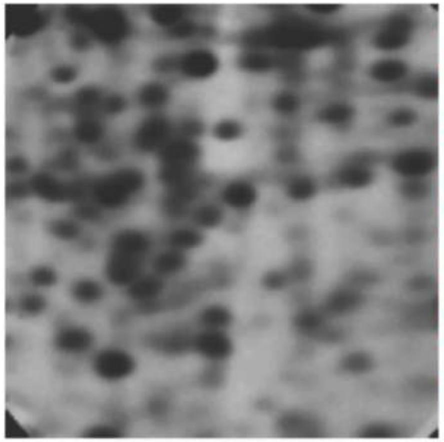
Marker-controlled water segmentation

Problem

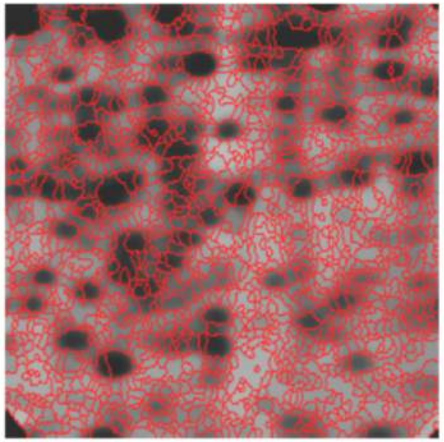
Due to noise and other local irregularities of the gradient, ***over segmentation*** may occur

Solution

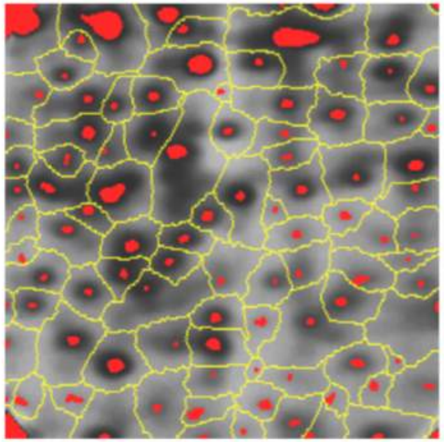
- **Exclude** a number of **non-significant minima**
- Do the exclusion implicitly **using markers** on the blobs to specify the only allowed regional minima (like seeds for region growing algorithm)
- For example, gray level values can be used as markers



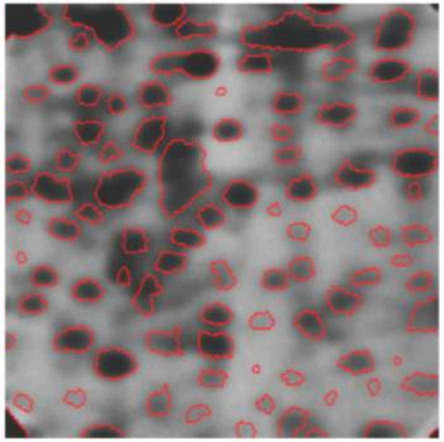
original image



over-segmented image

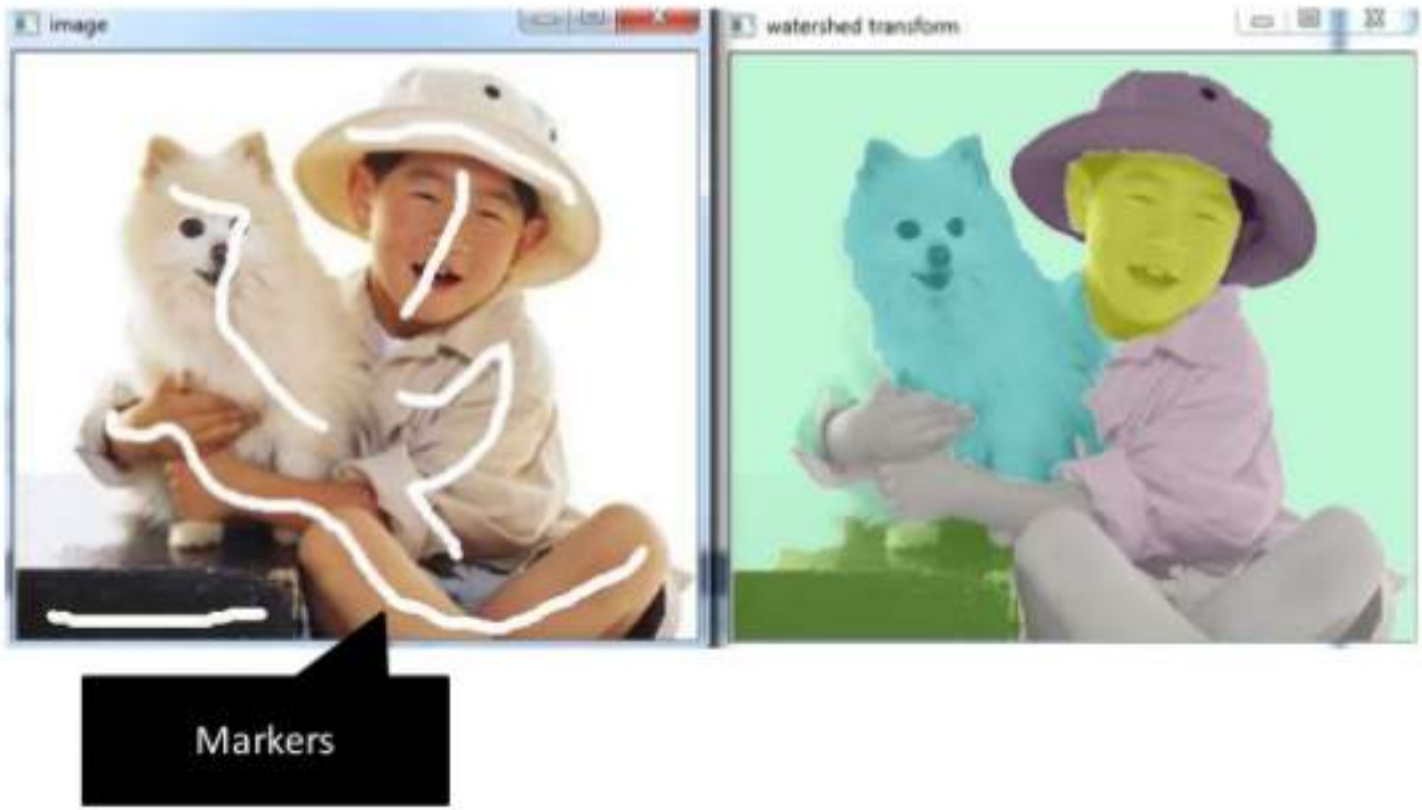


markers of the blobs and of the background



marker-controlled watershed of the gradient image

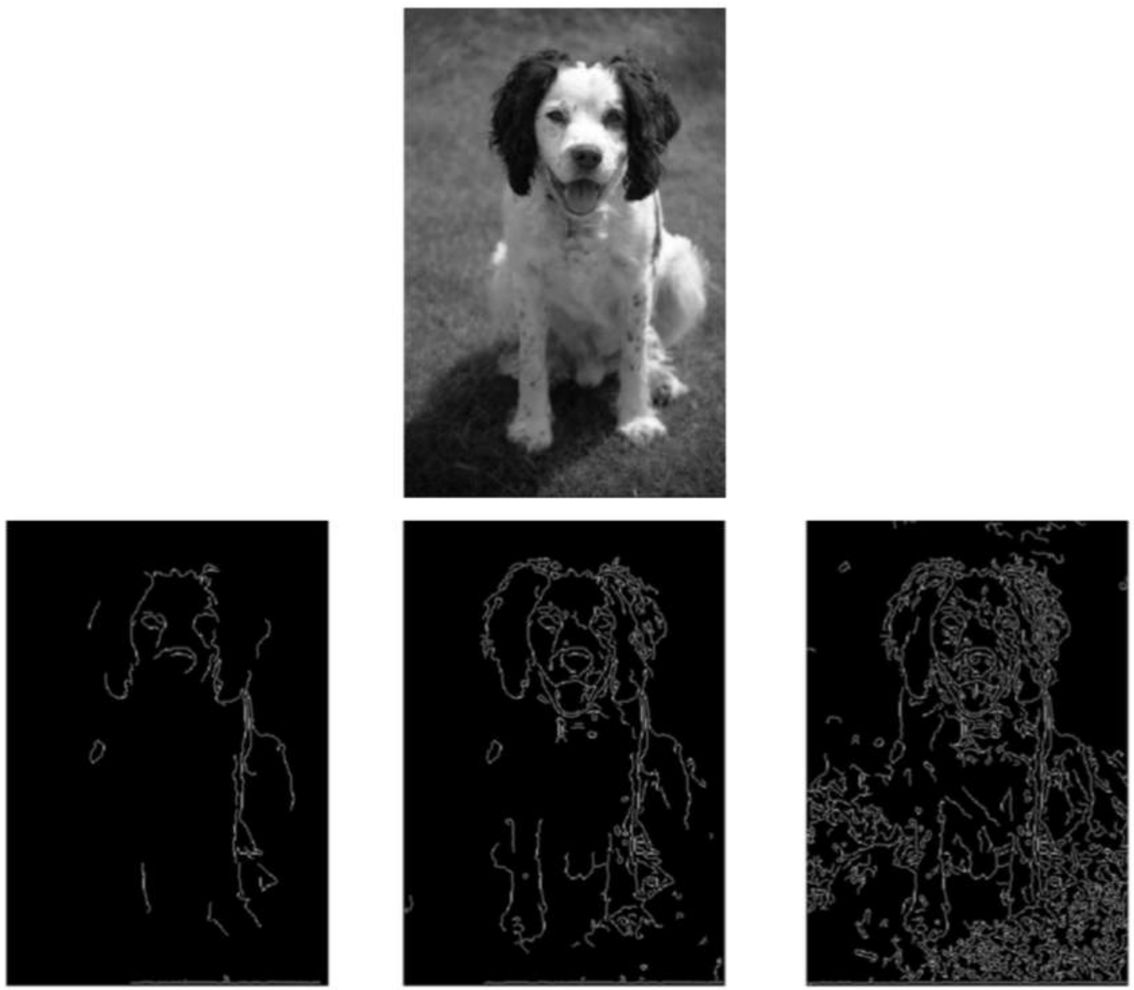
Marker controlled water segmentation



Active contours

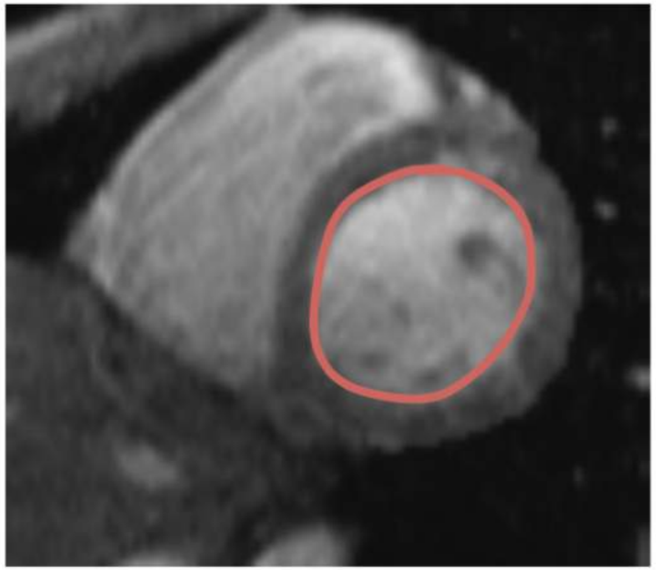
Active contours: Introduction

- Why?
  - When **edge-segmentation** is **fragmented**

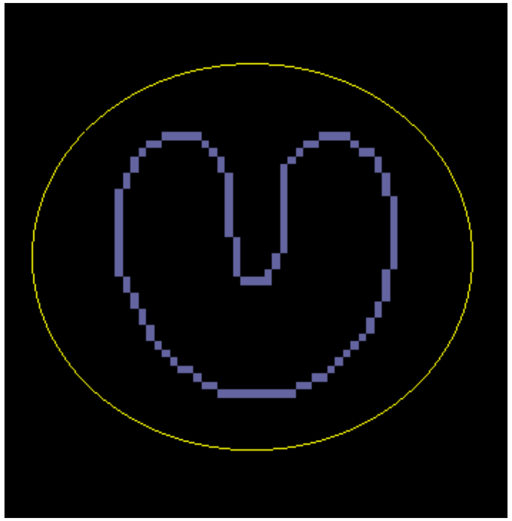


Active contours: Introduction

- The active contour model (also called ***snakes***) extract the object contours of an image
  - It is based on the variational theory
  - The active contour evolves like a ***"snake"***



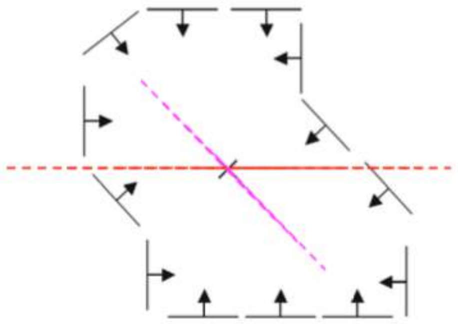
- Goal
  - **Given initial contour** near the desired object **evolve** the contour **to fit** exact **object boundary**

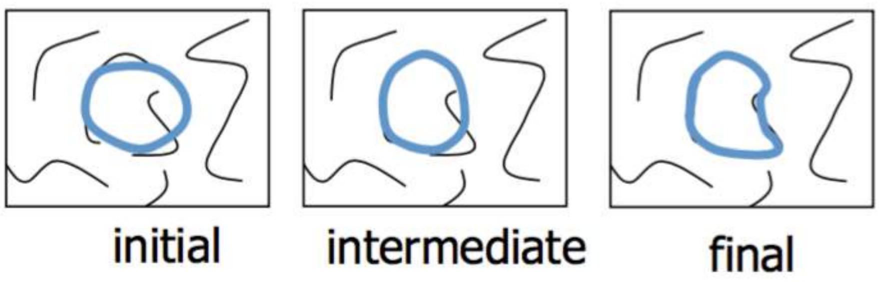


- Elastic band is iteratively adjusted so as to
  - be near image positions with *high gradient*
  - satisfy *contours priors*

Snakes vs Hough transformation

- Like Hough transformation is useful for shape fitting, but





Hough

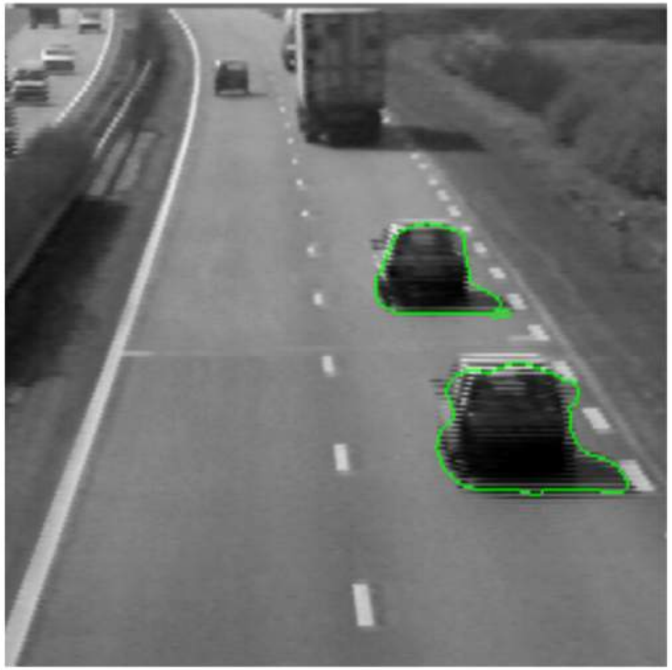
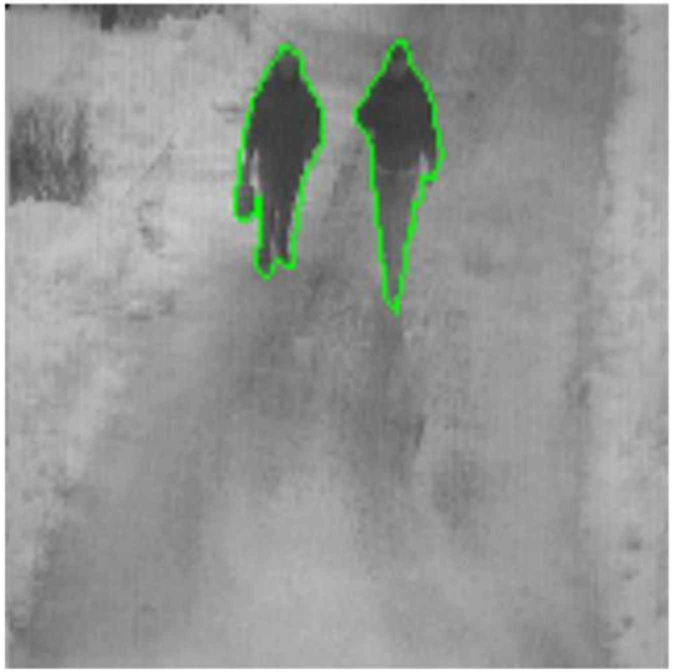
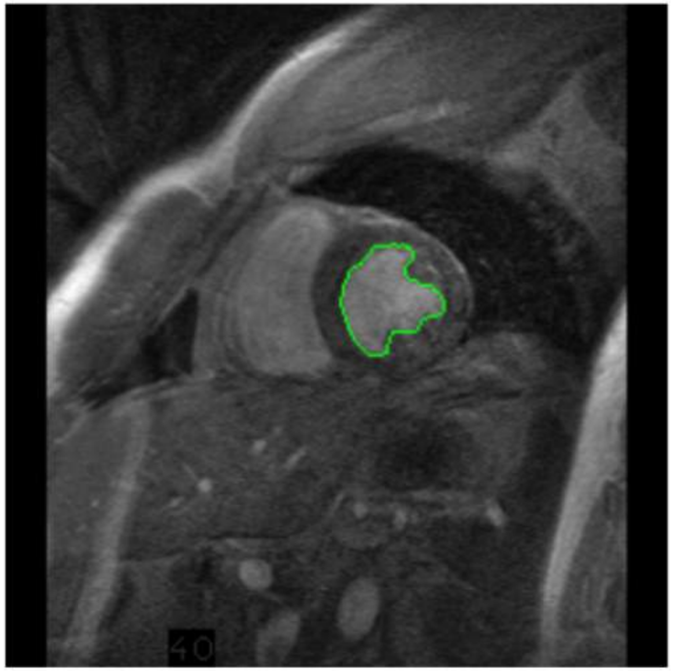
- Rigid model shape
- Single voting pass can detect multiple instances

Snakes

- Prior on shape types, but shape iteratively adjusted (*deformed*)
- Requires initialization nearby
- One optimization "pass" to fit a single contour

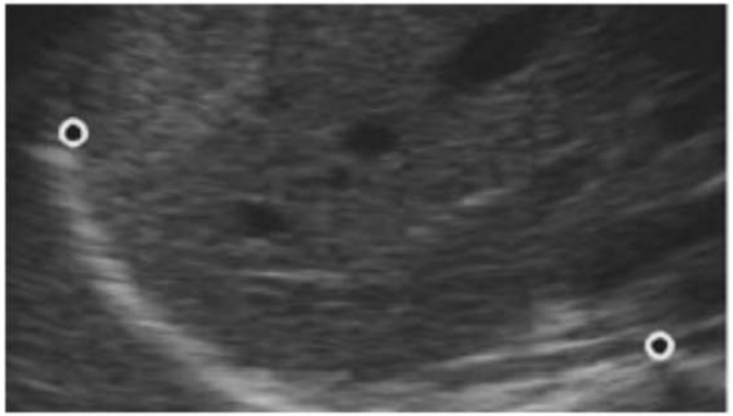
Why do we want to fit deformable shape?

- Non-rigid deformable objects can change their shape over time

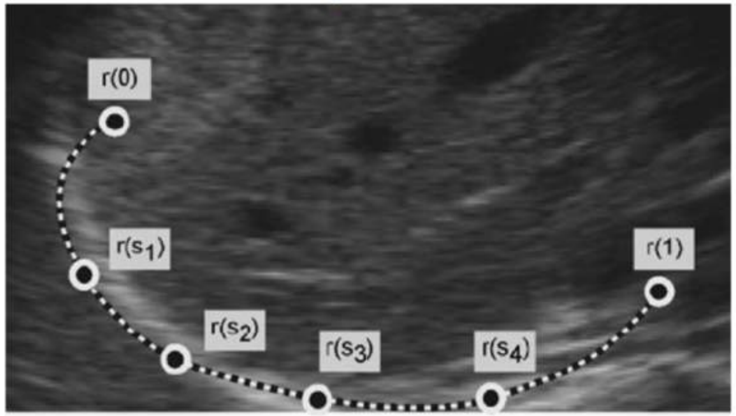
  

Main idea

- Represents an object boundary or some other salient image feature as a **parametric curve**

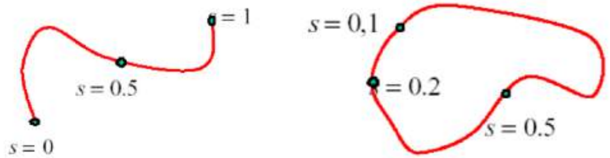


- An ***energy functional*** *E* is associated with the curve
- The problem of finding object boundary is cast as an **energy minimization problem**
- At each iteration we can move each vertex to another nearby location ("state")



Energy minimization

- A snake *C* is a ***curve*** $C = \{\nu(s) = (x(s), y(s)) \mid s \in [0,1]\}$



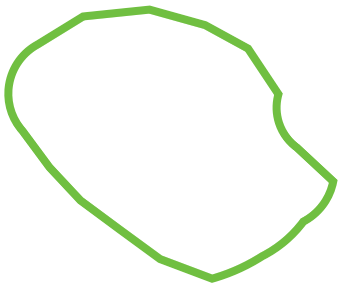
- The movements of a snake is modelled as an ***energy minimization*** process

$$E = E_i + E_e + E_c = \int_0^1 \left(E_i(\nu(s)) + E_e(\nu(s)) + E_c(\nu(s))\right) ds$$

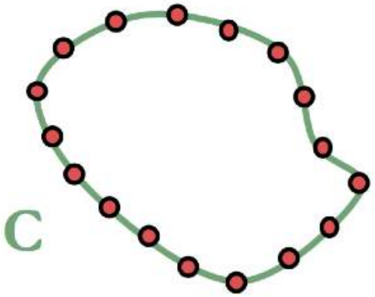
  - E_i ***internal forces***, increases if the snake is stretched or bent
  - E_e ***external forces***, decreases if the snake moves closer to the border of the object we want to segment
  - E_c ***additional constraints*** such as penalizing the creation of loops (for many applications it is set to 0)

Discretizing …

- A snake is a curve
  - ▶ Continuous case $C = \{\nu(s) = (x(s), y(s)) \mid s \in [0, 1]\}$



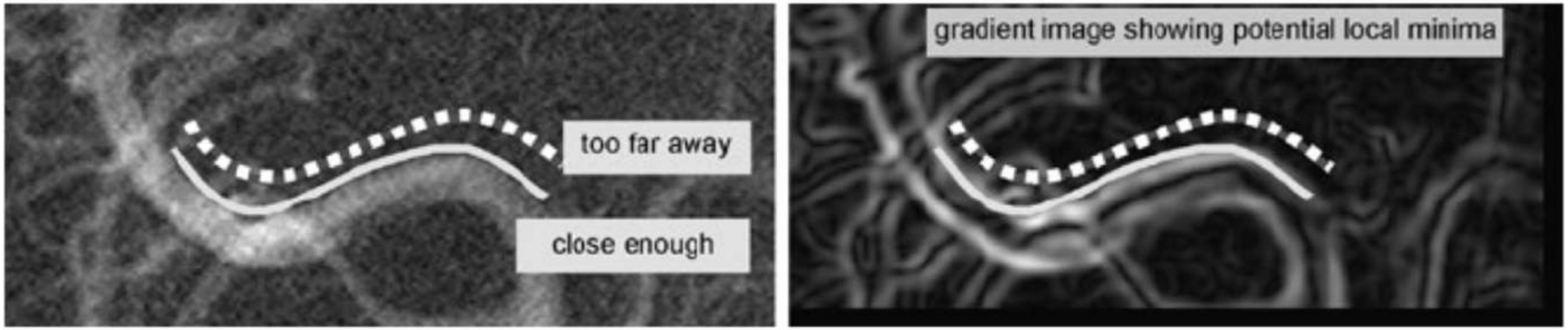
  - ▶ Discrete case $C = \{\nu_i = (x_i, y_i) \mid 0 \leq i \leq 1\}$



- The movement is modelled as an energy minimization process
  - ▶ Continuous case $E = \int_0^1 (E_i(\nu(s)) + E_e(\nu(s)) + E_c(\nu(s)))\, ds$
  - ▶ Discrete case $E = \sum_{j=0}^{n-1} (E_i(\nu_j) + E_e(\nu_j) + E_c(\nu_j))$

External (Image) energy

- Encourage contour to fit on places where image structure exist
  - Measure **how well** the curve matches the image data locally
  - “**Attract**” the curve toward different image features (edge, lines, etc., …)



- Given the image *I* $E_e(\nu(s)) = -c_3 \, ||\nabla I(\nu(s))||^2$
  - c_3 is a constant that sets the relative influence of the edge attraction force
  - For the entire snake $E_e = \int_0^1 E_e(\nu(s))ds$ → (discretization) $E_e = \sum_{j=0}^{n-1} (-c_3 \, ||\nabla I(\nu_j)||^2$

Internal energy

- Encourage ***prior*** shape preferences: e.g., smoothness, elasticity, particular known shape
  - A priori we want to favour **smooth shapes**, contours with **low curvature**, contours similar to a **known shape**, etc., to balance what is actually observed in the gradient image

- Given the curve *C*

$$E_i = c_1 \left|\left| \frac{d\,\nu(s)}{ds} \right|\right|^2 + c_2 \left|\left| \frac{d^2\,\nu(s)}{ds^2} \right|\right|^2$$

Tension, Elasticity

Stiffness, Curvature

  - c_1 is a constant that controls the **elasticity**
  - c_2 is a constant that controls **how much the snake can bend**

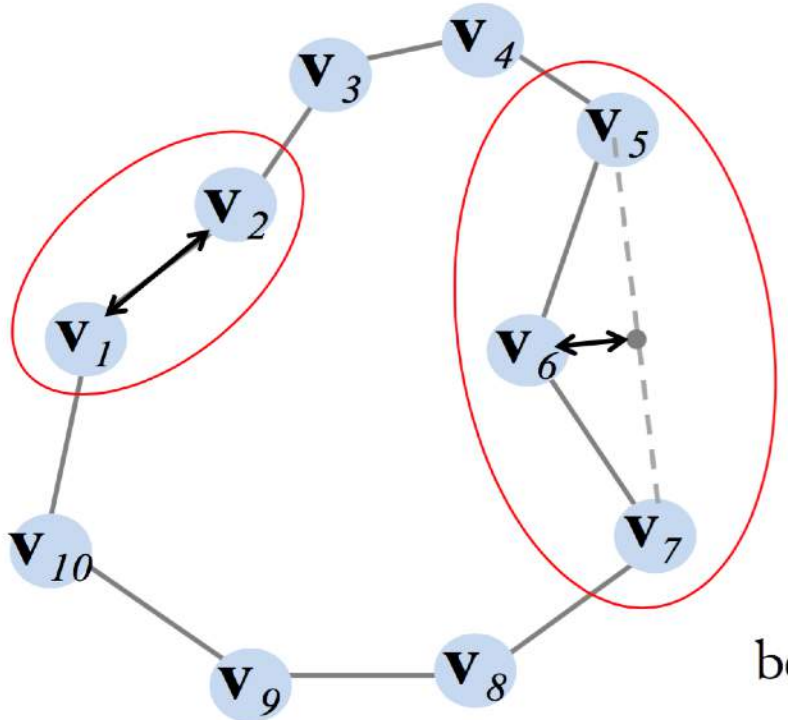
Discrete internal energy

$$\mathbf{C} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{n-1}) \in \Re^{2n}$$

$$\mathbf{v}_i = (x_i, y_i)$$

elastic energy
(elasticity)

$$\frac{dv}{ds} \approx v_{i+1} - v_i$$



bending energy
(stiffness)

$$\frac{d^2 v}{ds^2} \approx (v_{i+1} - v_i) - (v_i - v_{i-1}) = v_{i+1} - 2v_i + v_{i-1}$$

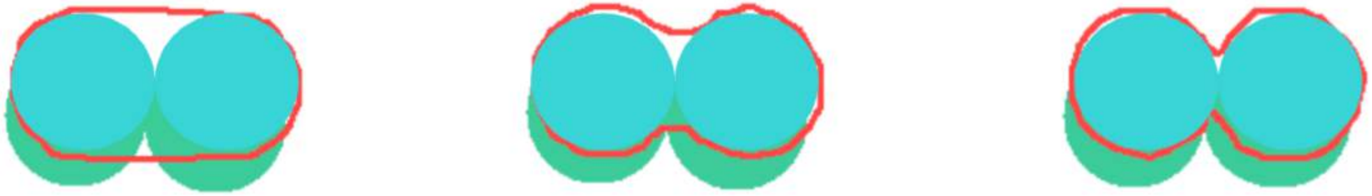
- For the entire snake

$$E_i = \sum_{j=0}^{n-1} c_1 \left| \nu_{j+1} - \nu_j \right|^2 + c_2 \left| \nu_{j+1} - 2\nu_j + \nu_{j-1} \right|^2$$

Elasticity — Stiffness, Curvature

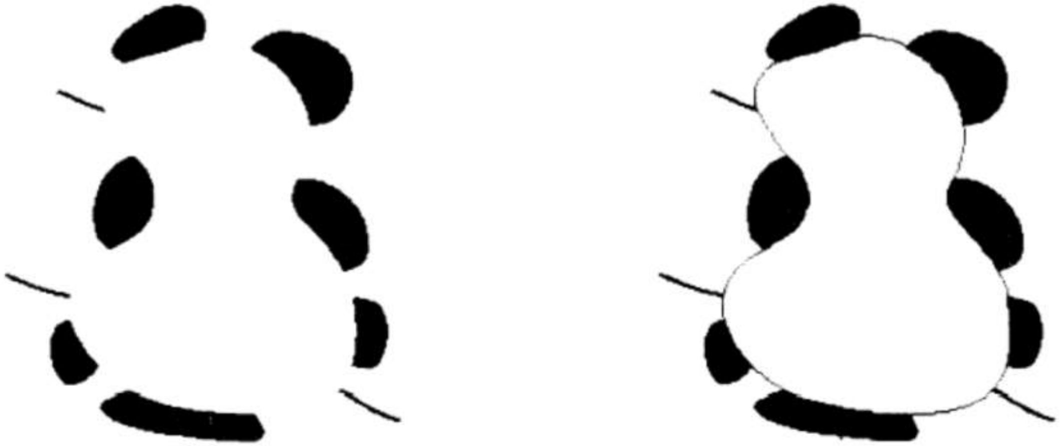
Internal energy properties

- Elasticity term measures how much the snake is stretched locally
  - small c_1 will result in high stretch values having no impact on the energy (the snake may stretch infinitely)
  - large c_1 makes that the snake can stretch very little



large c_1 medium c_1 small c_1

- The preferences for low-curvature, smoothness help deal with missing data



Total Energy

- We need to minimize the total energy

$$E_{total} = \int_0^1 (E_i(\nu(s)) + E_e(\nu(s)) + E_c(\nu(s)))\, ds$$

$$= \int_0^1 \left(c_1 \left|\left| \frac{d\nu(s)}{ds} \right|\right|^2 + c_2 \left|\left| \frac{d^2\nu(s)}{ds^2} \right|\right|^2 - c_3 \left|\left| \nabla I(\nu(s)) \right|\right|^2 + E_c(\nu(s)) \right) ds$$

- Numerically this is done by solving its Euler-Lagrange form

$$-\frac{d}{ds}\left(c_1 \left|\left| \frac{d\nu(s)}{ds} \right|\right|^2 \right) + \frac{d^2}{ds^2}\left(c_2 \left|\left| \frac{d^2\nu(s)}{ds^2} \right|\right|^2 \right) + \nabla \left(E_e(\nu(s)) + E_c(\nu(s)) \right) = 0$$

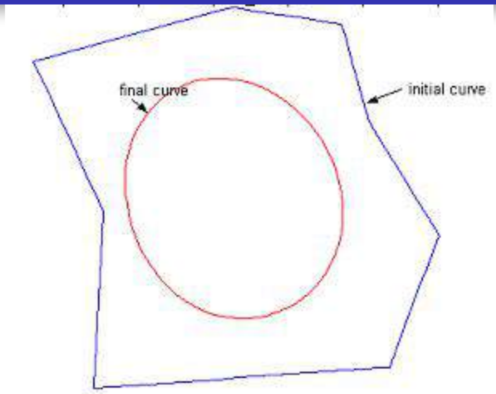
- Of course the discretized version!
    - This equation can be interpreted as a force balance equation
    - The contour deforms under the action of these forces

Forces

Elastic force

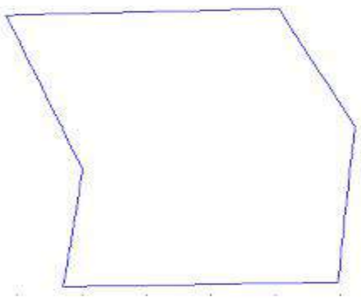
- Generated by elastic potential energy of the curve

$$F_{elastic} = -\frac{d}{ds}\left(c_1 \left\|\frac{d\nu(s)}{ds}\right\|^2\right)$$

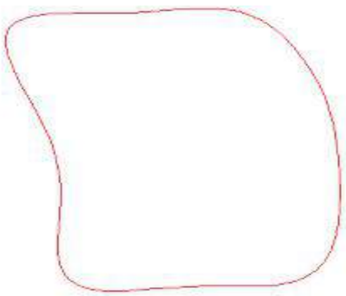


Bending force

- Generated by the bending energy of the contour
- Tries to smooth out the curve



Initial curve
(High bending energy)



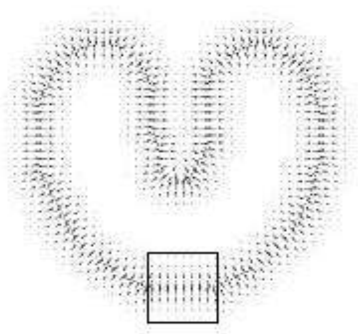
Final curve deformed by bending force. (low bending energy)

External force

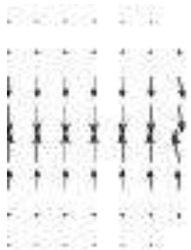
- Minimizes E_e

$$F_e = -\nabla E_e$$

Image



External force

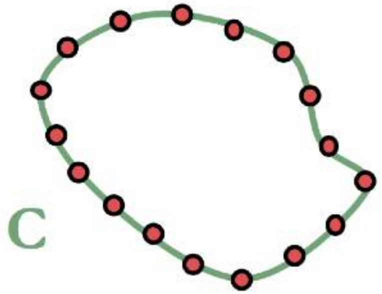


Zoomed in

Discretizing …

- The curve is **piecewise linear** obtained by joining each control point

$$C = \{\nu_i = (x_i, y_i) \mid 0 \leq i \leq 1\}$$

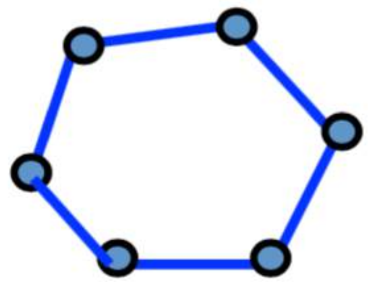


- Force equations applied to **each control point separately**

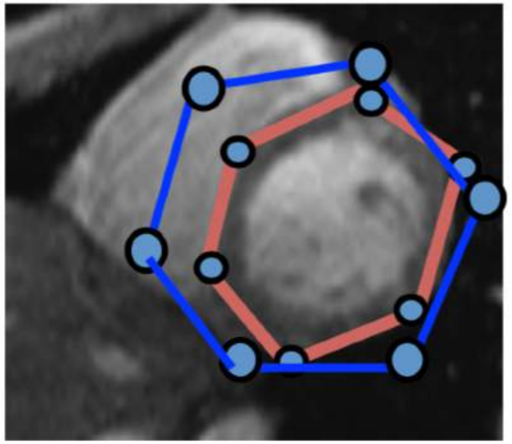
- Each control point allowed to move freely under the influence of the forces

- The energy and **force terms** are **converted to discrete form** with the derivatives substituted by finite differences
  - as we have seen for energies

Elastic snake example



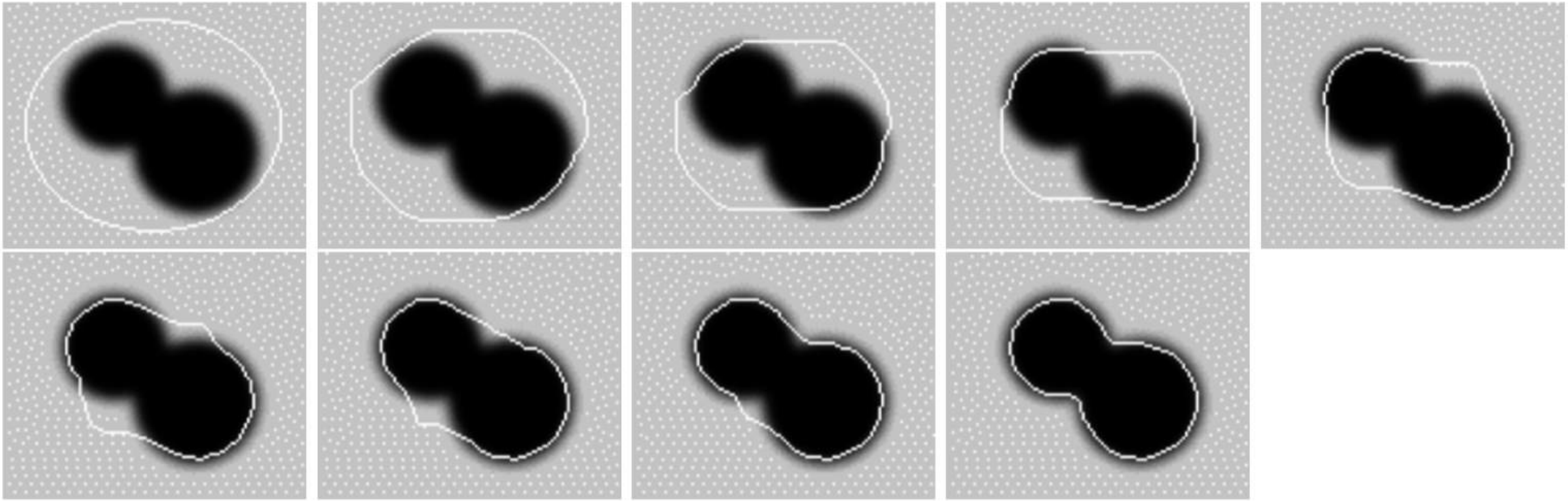
- A simple elastic snake is defined by
  - A set of n points
  - An internal energy term (tension, bending) $E_i = c_1 \sum_{j=0}^{n-1} |\nu_{j+1} - \nu_j|^2$
  - An external energy term (gradient-based) $E_e = -\sum_{j=0}^{n-1} |\nabla I(\nu_j)|^2$
- To use to segment an object
  - Initialize in the vicinity of the desired object
  - Modify the points to minimize the total energy

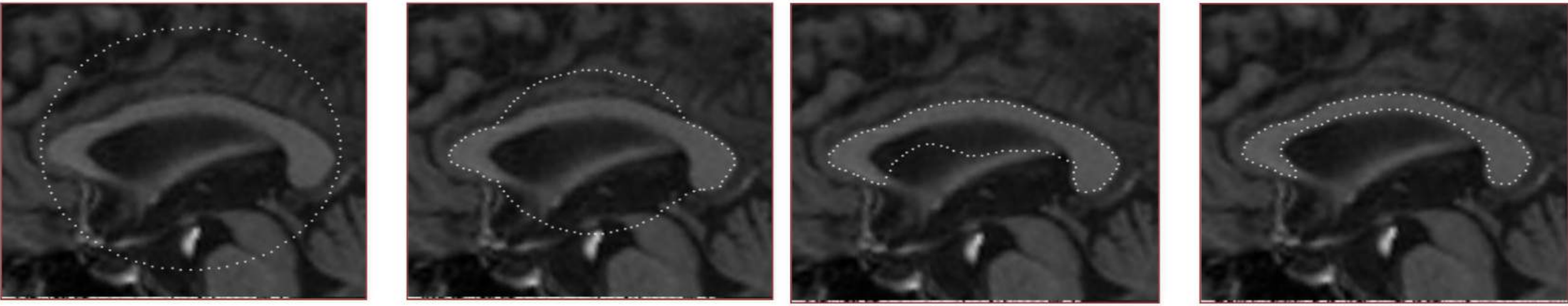


$$E = \sum_{j=0}^{n-1} c_1 |\nu_{j+1} - \nu_j|^2 - |\nabla I(\nu_j)|^2$$

$$= \sum_{j=0}^{n-1} c_1 \left((x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2 - \left(|I_x(x_j, y_j)|^2 + |I_y(x_j, y_j)|^2\right)\right)$$

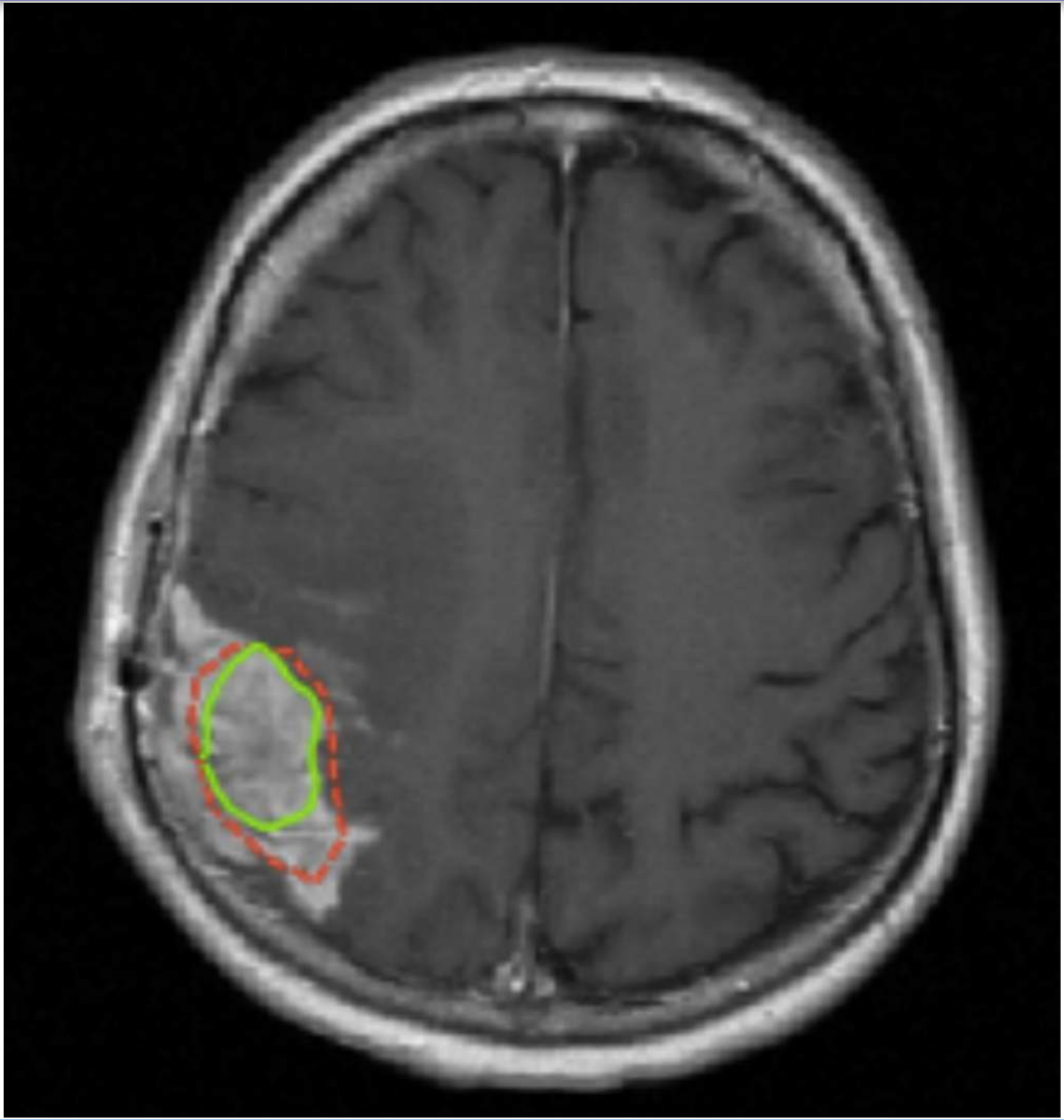
Examples without constraints





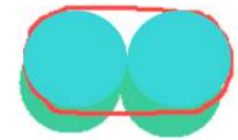
Example using only external force

Red: Initial contour
Green: Final contour



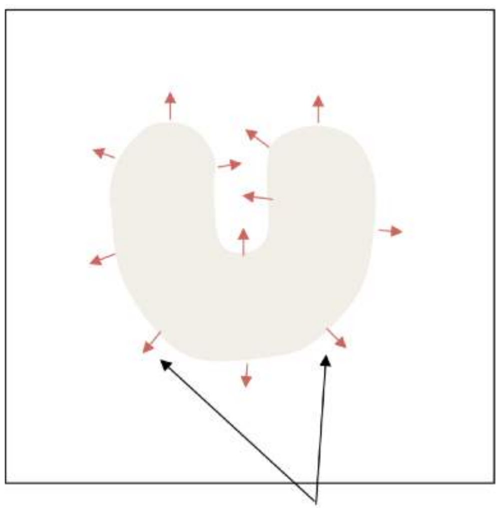
Limitations

- May over-smooth the boundary



- Cannot follow topological changes of objects
  - can be overcome using ***level sets***

- snake does not really "see" object boundaries in the image unless it gets very close to it!



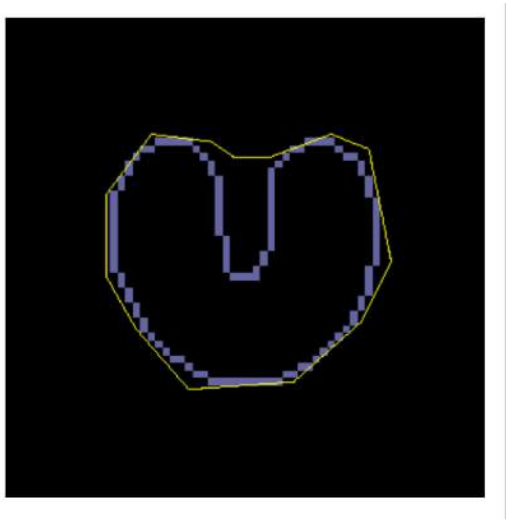
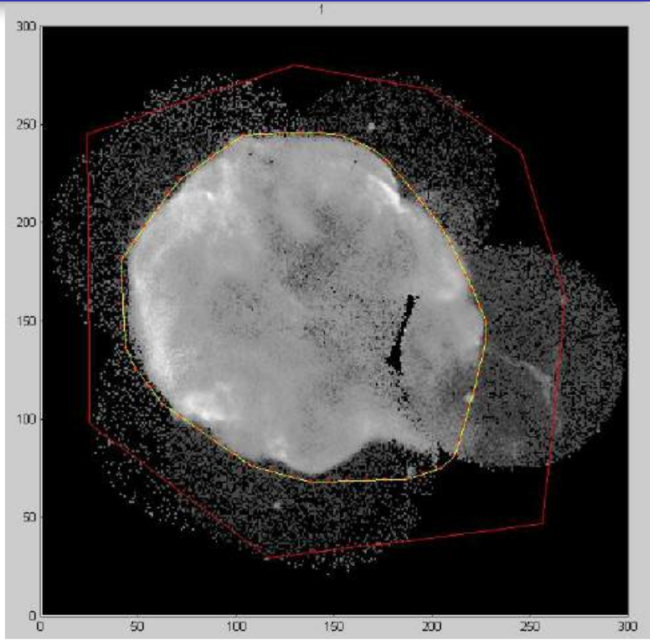


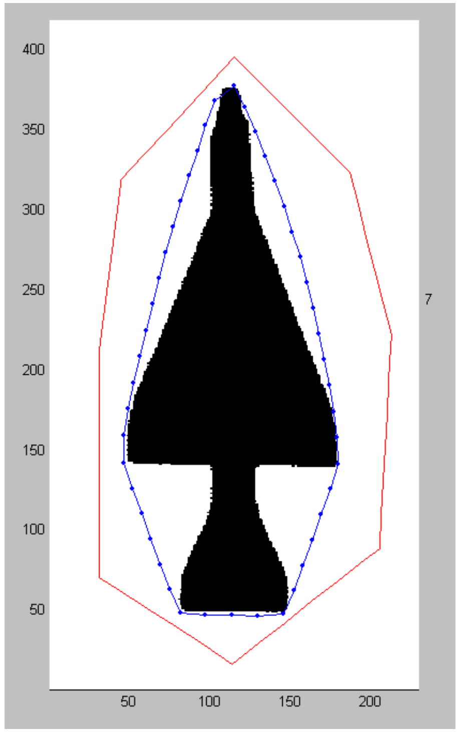
image gradients are large only directly on the boundary!

Limitations

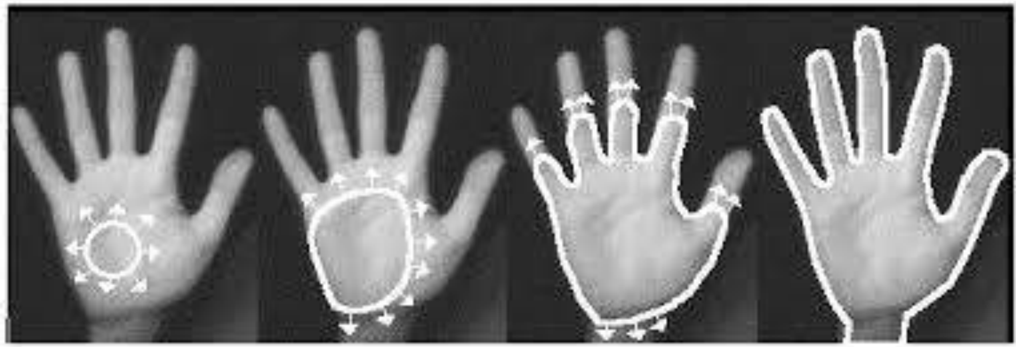
- Snakes are very sensitive to false local minima which leads to wrong convergence



- Fails to detect ***concave*** boundaries. External force cant pull control points into boundary concavity
    - Can be solved using ***Gradient Vector Flow***



Active contours comments



Advantages

- Useful to track and fit non-rigid shapes
- Contour remains connected
- Possible to fill in "subjective contours"
- Flexibility in how energy function is defined, weighted

Disadvantages

- Depends on number and spacing of control points
- Must have a decent **initialization** near true boundary
- May get stuck in **local minimum**
- Parameters of energy function must be set well based on prior information