EECE 5639 Computer Vision I

Lecture 12

Snakes, Region Segmentation

Project 2 is out …

Next Class

VIDEO ONLY!!

Homographies

Active Contours

Deformable Contours

They are also called
- Snakes
- Active contours

Think of a snake as an elastic band:
- of arbitrary shape
- sensitive to image gradient
- that can wiggle in the image
- represented as a necklace of points

Active Contour Models

An important class of algorithms to find boundaries

Usually does not use prior knowledge of the shape

Poses the problem as an “optimization” problem

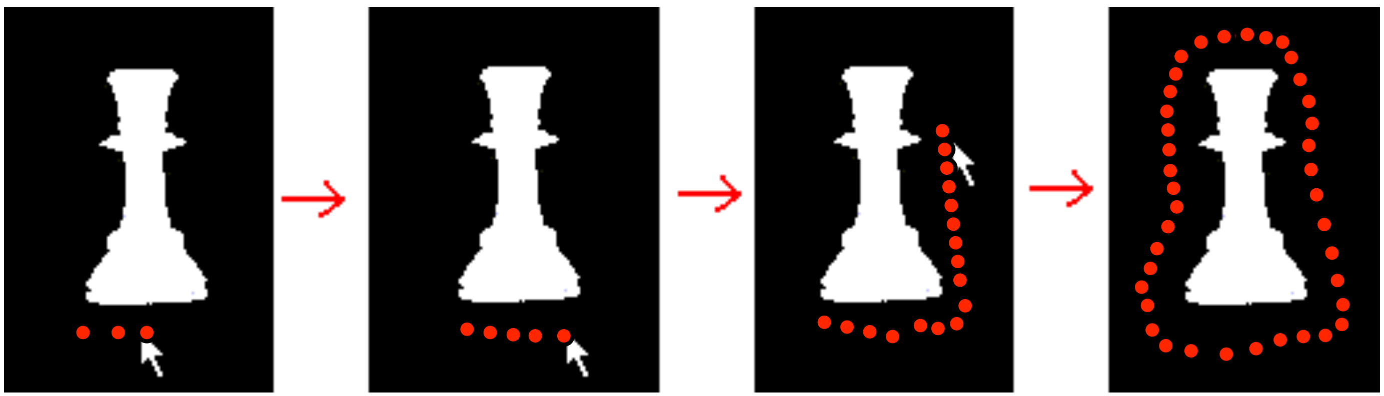
Introduction

How can we find the boundary of an object in an image?

- One approach could be:
  - Find edges
  - Link the edges
- Another possibility is to search for “smooth” boundaries:
  - The boundary should “match” the image
  - Can iteratively “improve”

Main Idea:

"Drop" a snake



Let the snake "wiggle", attracted by image gradient, until it glues itself against a contour

The Energy Functional

- Associate to each possible shape and location of the snake a value E.
  - Values should be s.t. the image contour to be detected has the <u>minimum</u> value.
  - E is called the energy of the snake.
- Keep wiggling the snake towards smaller values of E.

Energy Functional Design

We need a function that given a snake state, associates to it an Energy value E.

The function should be designed so that the snake moves towards the contour that we are seeking!

What moves the snake?

"Forces" applied to its points

Snake Energy

The total energy of the snake is defined as:

$$E_{total} = E_{internal} + E_{external}$$

The internal energy encourages smoothness
The external energy encourages closeness to edges

Forces moving the snake (External)

It needs to be attracted to contours:

- Edge pixels must “pull” the snake points.
- The stronger the edge, the stronger the pull.
- The force is proportional to $|\nabla I|$

Forces preserving the snake (Internal)

The snake should not break apart!

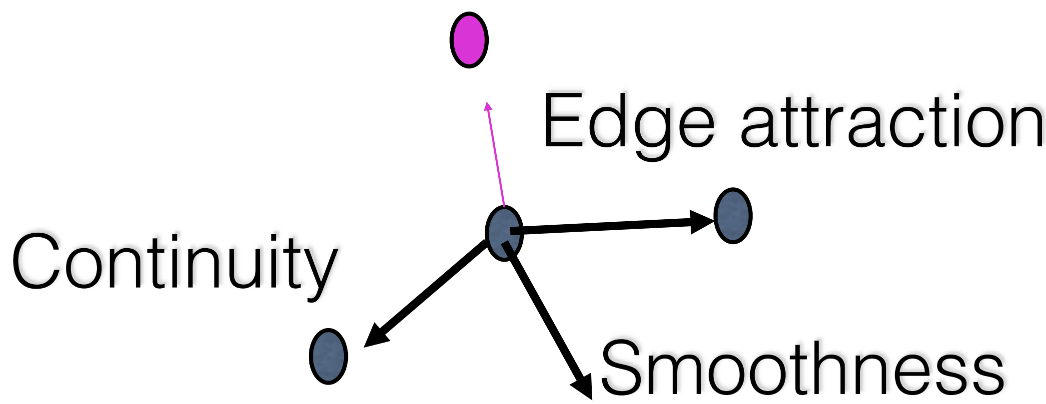
- Points on the snake must stay close to each other
- Each point on the snake pulls its neighbors
- The farther the neighbors, the stronger the force
- The force is proportional to the distance $|P_i - P_{i-1}|$

Forces preserving the snake (Internal)

The snake should avoid “oscillations”

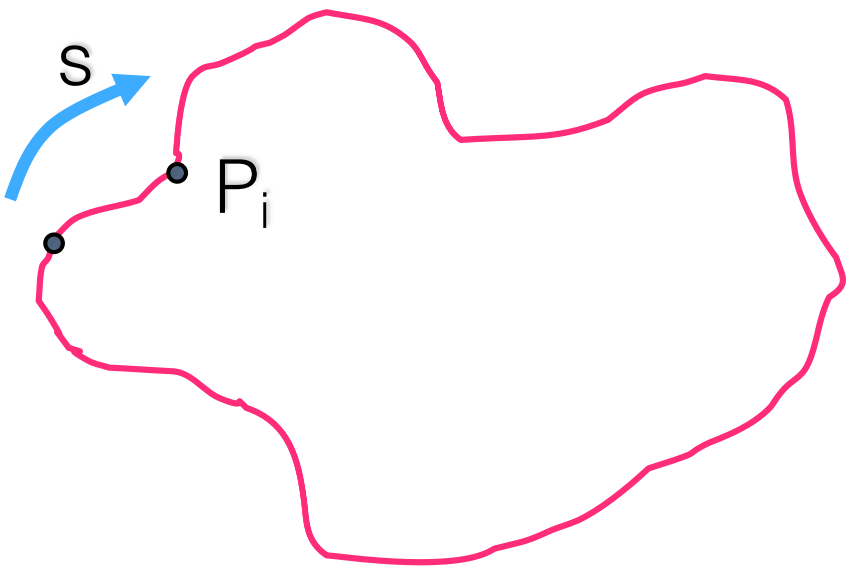
- Penalize high curvature
- Force proportional to snake curvature

Snake Forces



Snake "state": Contour Parametrization

Consider a contour parametrization c=c(s) where s is the "arc length"



Each point P_i on the contour has coordinates $(x_i(s), y_i(s))$

$$E = \int E_{int}(s) + E_{ext}(s) ds$$

Snake Energy Functional

Given a snake with N points $p_1, p_2, \ldots, p_N$

$$E = \sum_{i-1}^{N} a_i E_c(p_i) + b_i E_s(p_i) + c_i E_g(p_i)$$

"Continuity" "Smoothness" "Edgeness"

a_i, b_i, c_i are "weights" to control influence

Continuity Term

Given a snake with N points $p_1, p_2, \ldots, p_N$ $\quad p_i = [x_i \; y_i]$

Let d be the average distance between points

Distance between points should be kept close to average

Define the continuity term of the Energy Functional:

$$E_c(p_i) = (d - |p_i - p_{i-1}|)^2$$

$$E_c = \left(d - \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right)^2$$

Smoothness Term

Given a snake with N points $p_1, p_2, \ldots, p_N$

$$p_i = [x_i \ y_i]$$

Curvature should be kept small

Define the smoothness term of the Energy Functional:

$$E_s(p_i) = \underbrace{|p_{i-1} - 2p_i + p_{i+1}|^2}_{\text{Second derivative}}$$

$$E_s = (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2$$

Edgeness Term

Given a snake with N points $p_1, p_2, \ldots, p_N$

$$p_i = [x_i \; y_i]$$

Define the edgeness term of the Energy Functional:

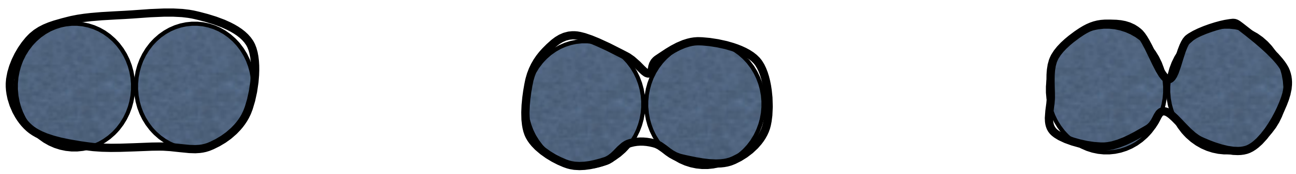
$$E_g(p_i) = -|\nabla I(p_i)|$$

$$\nabla I(p_i) = [G_x(p_i) \; G_y(p_i)]$$
$$|\nabla I(p_i)| = \sqrt{G_x(p_i)^2 + G_y(p_i)^2}$$

Magnitude of the gradient should be LARGE

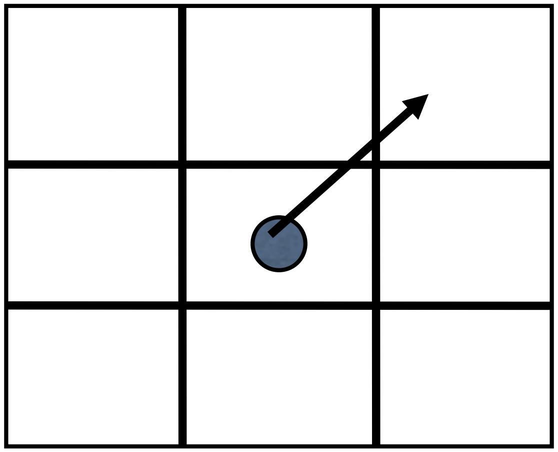
Relative Weighting

The weights control the smoothness and stiffness of the snake



Greedy Algorithm

Each point moves within a small window to minimize the energy



Compute the new energy for each candidate location
Move the point to the one with the minimum value

Keeping corners …

Before starting a new iteration:

- Search for “corners”:
  - max curvature
  - large gradient
- Corner points should not contribute to the energy (set $b_i = 0$)

Implementation Considerations

To avoid numerical problems, the terms of the energy function should be normalized.

- E_c and E_s are normalized by their maximum in the neighborhood
- E_g is normalized as $|(\nabla I - m)| / (M - m)$
  - M and m are the max and min value of the gradient magnitude in the neighborhood

Snake Algorithm

Input:

gray scale image I

a chain of points $p_1,p_2,\ldots,p_N$

f is the fraction of points that must move to start a new iteration

$U(p)$ is a neighborhood around p

d is the average distance between snake points.

Snake Algorithm

1. While the fraction of moved points > f
   1. For $i=1,2,\ldots,N$
      1. find a point in $U(p_i)$ s.t. the energy is minimum,
      2. move p_i to this location
   2. For $i=1,2,\ldots,N$
      1. Estimate the curvature $k=|p_{i-1}-2p_i+p_{i+1}|$
      2. Look for local max, and set $b_{max} = 0$
   3. Update d

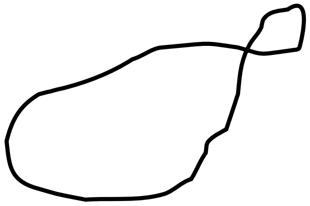
Problems with Snakes

Smoothness does not always capture all prior knowledge

User must define the weights

Snakes might oversmooth boundaries

Not trivial to prevent curve self intersecting



Region Segmentation

Regions and Edges

Ideally, regions are bounded by closed contours

- We could "fill" closed contours to obtain regions
- We could "trace" regions to obtain edges

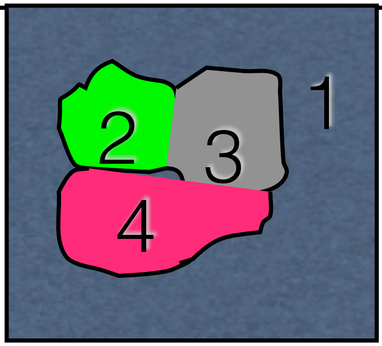
Unfortunately, these procedures rarely produce satisfactory results.

Regions and Edges

Edges are found based on DIFFERENCES between values of adjacent pixels.

Regions are found based on SIMILARITIES between values of adjacent pixels.

Region Segmentation



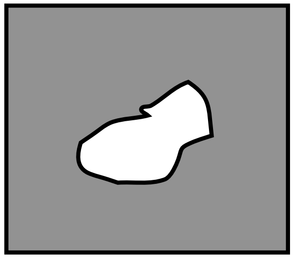
A segmentation is a **partition** $R_1, R_2, \ldots R_n$ s.t.:

- $\bigvee_{i=1}^{n} R_I = I$
- $R_i \wedge R_j = \emptyset$ If $i \neq j$
- $Pred(R_i)$ = TRUE For all i
- $Pred(R_i \vee R_j)$ = FALSE for all R_i adjacent R_j

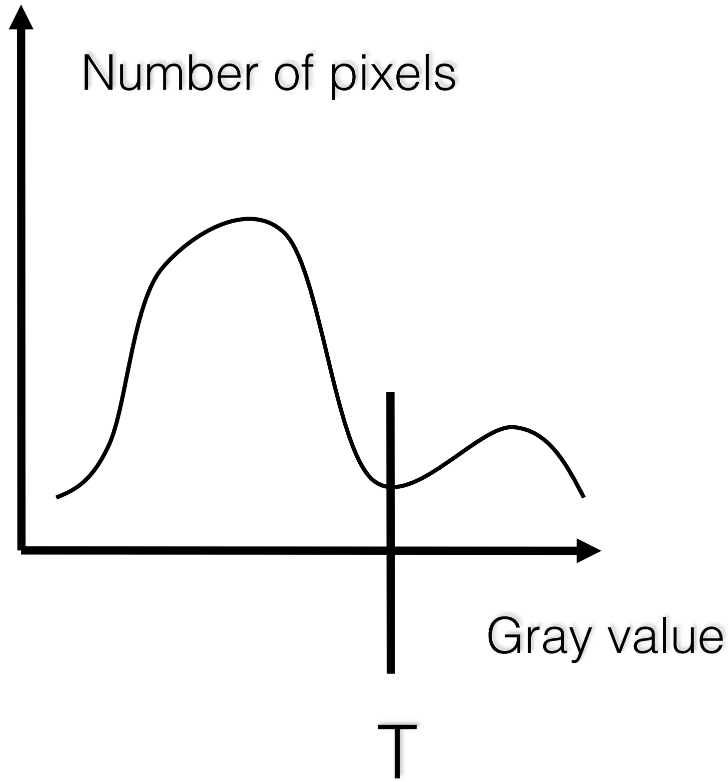
Where "Pred" is a function that evaluates similarities of the pixels in the region

Histogram-based Segmentation

Ex: bright object on dark background:



Histogram



Select threshold

Create binary image:

I(x,y) < T -> O(x,y) = 0

I(x,y) > T -> O(x,y) = 1

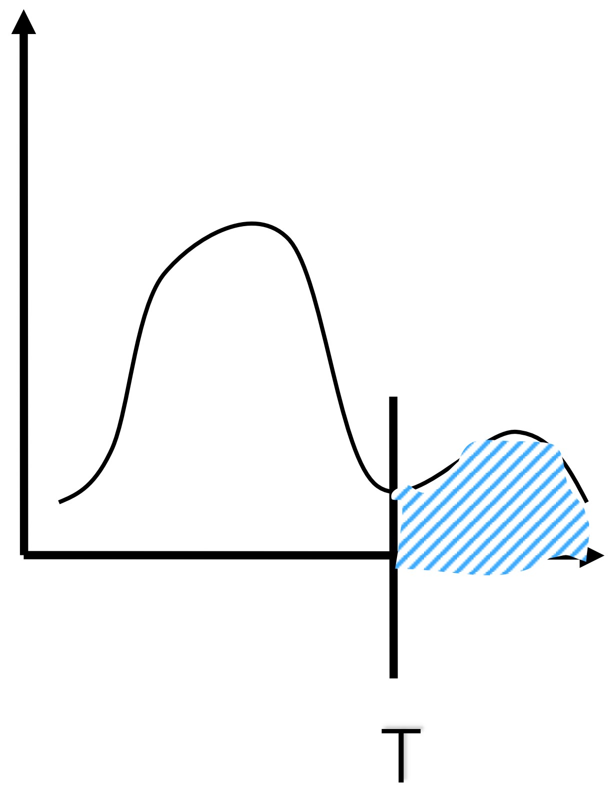
How do we select a Threshold?

Automatic thresholding

- P-tile method
- Mode method
- Peakiness detection
- Iterative algorithm

P-Tile Method

If the *size* of the object is approx. known, pick T s.t. the area under the histogram corresponds to the size of the object:

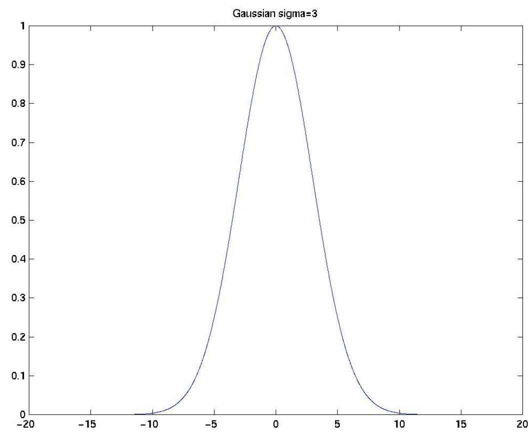


Mode Method

Model each region as “constant” + N(0,σ_i):

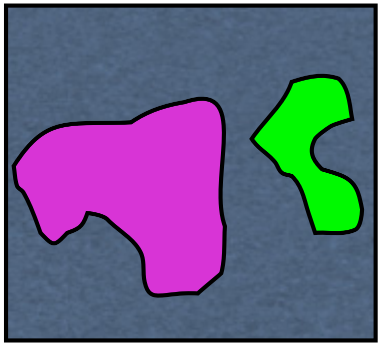
$$\text{If } (x,y) \in R_i \text{ then, } I(x,y) = \mu_i + n_i(x,y)$$

$$p(n_i) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{1}{2}\frac{n_i^2}{\sigma_i^2}}$$

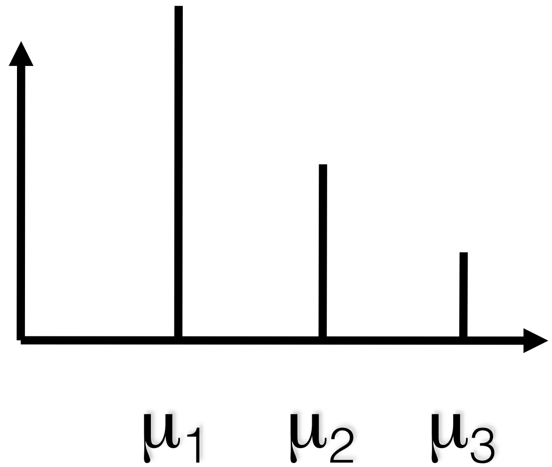


$$E(n_i) = 0 \qquad E(n_i^2) = \sigma_i^2$$

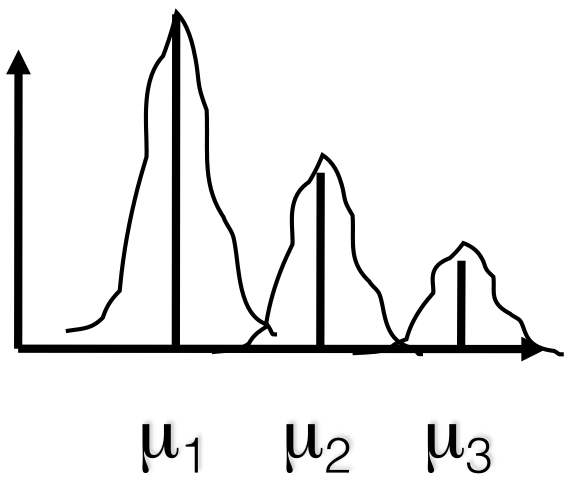
Example: Image with 3 regions



Ideal histogram:



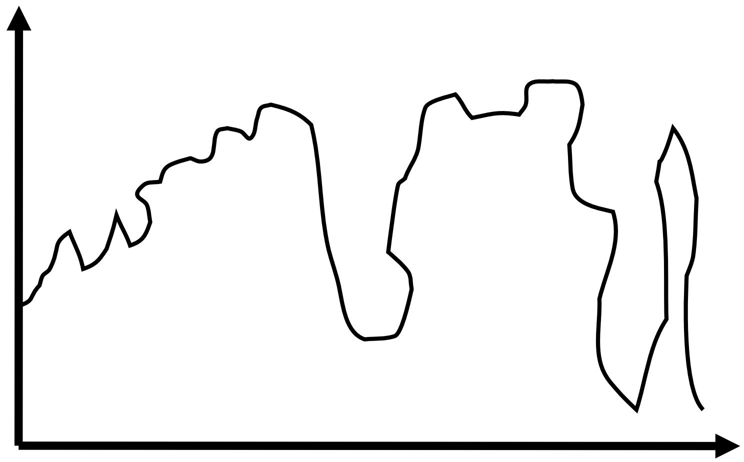
Add noise:



The valleys are good places for thresholding to separate regions.

Finding the peaks and valleys

It is a not trivial problem:



“Peakiness” Detection Algorithm

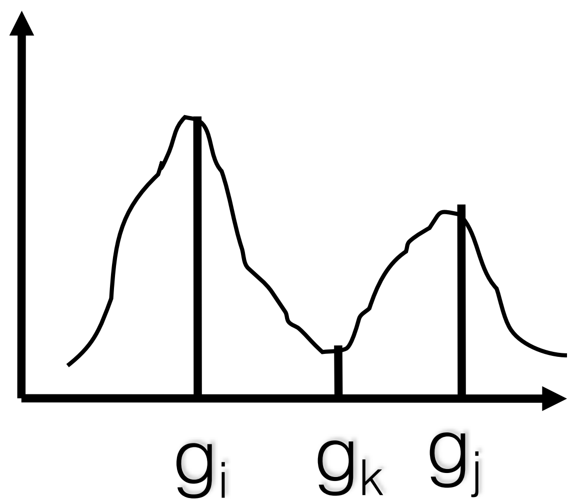
Find the two **HIGHEST LOCAL MAXIMA** at a **MINIMUM DISTANCE APART**: g_i and g_j

Find **lowest point** between them: g_k

Measure “peakiness”:

$\min(H(g_i),H(g_j))/H(g_k)$

Find (g_i,g_j,g_k) with highest peakiness



Iterative Threshold Algorithm

1. Select initial $T = T_o$

   Ex: T_o = average intensity

2. Partition image into regions R_1 and R_2 using T
3. Calculate the mean values of R_1 and R_2, μ_1 and μ_2

   Select a new threshold $T = ½(\mu_1 + \mu_2)$

1. Repeat 2-4 until μ_1 and μ_2 do not change.

Algorithm MEAN SHIFT

A non-parametric technique

Finds the peak of a given histogram

It is based on Robust Statistics

See: "**Robust Analysis of Feature Space: Color Image Segmentation,**" by D. Comaniciu and P.Meer, CVPR 1997, pp. 750-755.

Algorithm MEAN SHIFT to find histogram PEAK

1. Choose a window size
2. Choose the initial location of the search window
3. Compute the mean location in the search window
4. Center the window at the location computed in 3
5. Repeat steps 3 and 4 until convergence.

Mathematical Justification

Assume first that the histogram is unimodal

Let y be the possible gray values

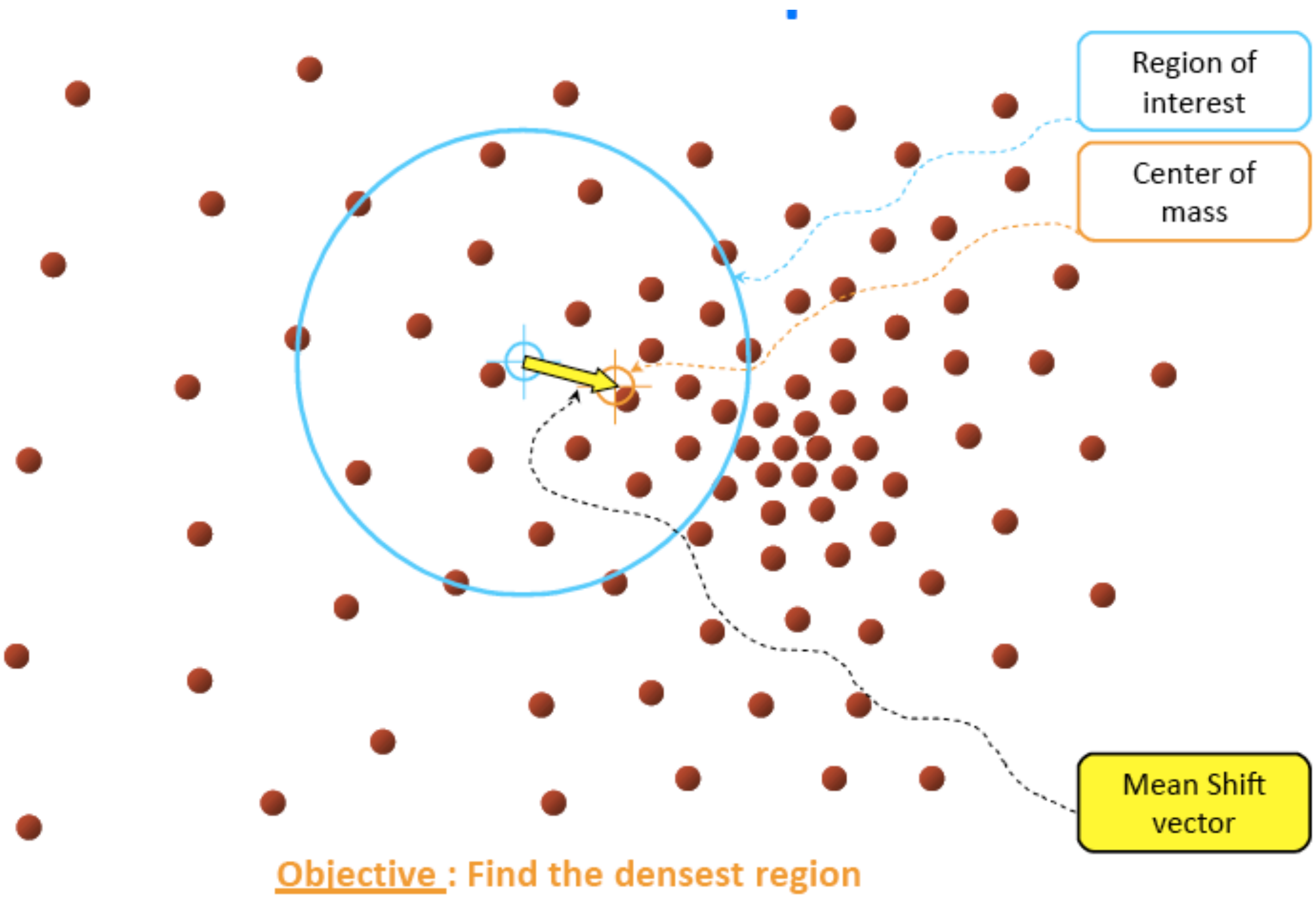
Let p(y) be the normalized histogram ~ pdf

$$p(y) = \frac{\text{pixels with value y}}{\text{pixels in image}}$$

Mean Shift

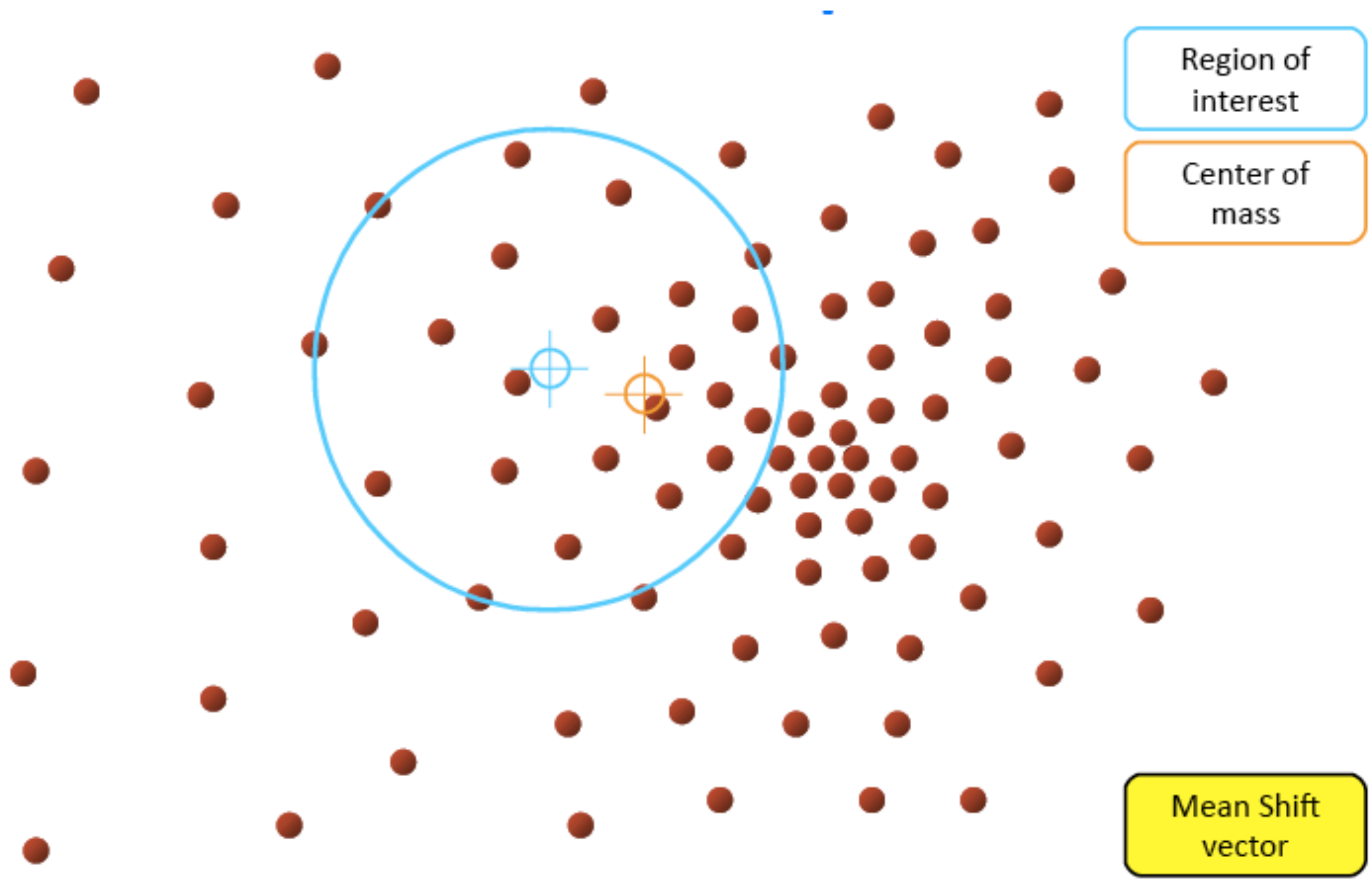
It is a hill-climbing algorithm that seeks modes of a non-parametric density represented by samples.

Intuitive Description



Ukrainitz&Sarel, Weizmann

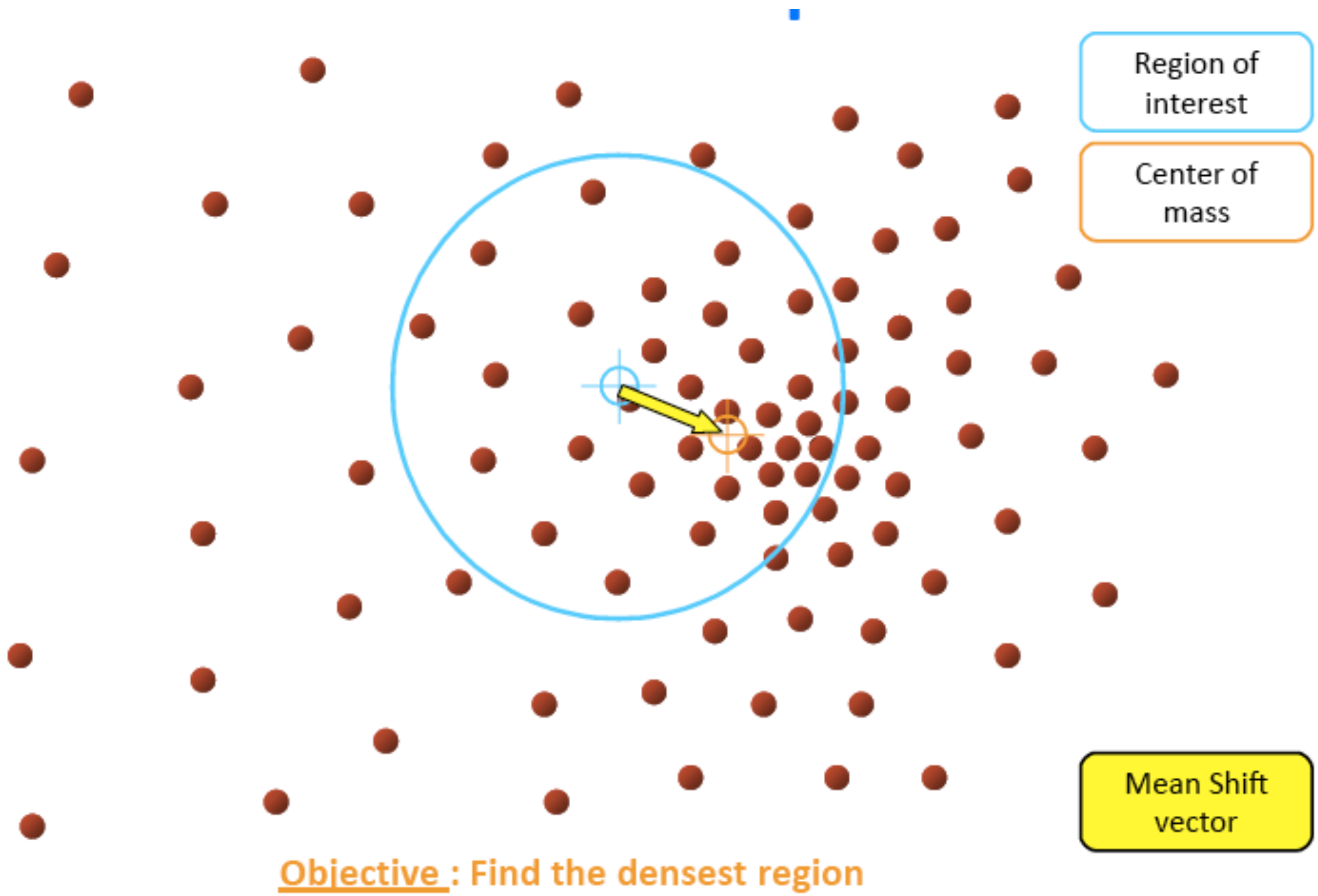
Intuitive Description



Objective : Find the densest region

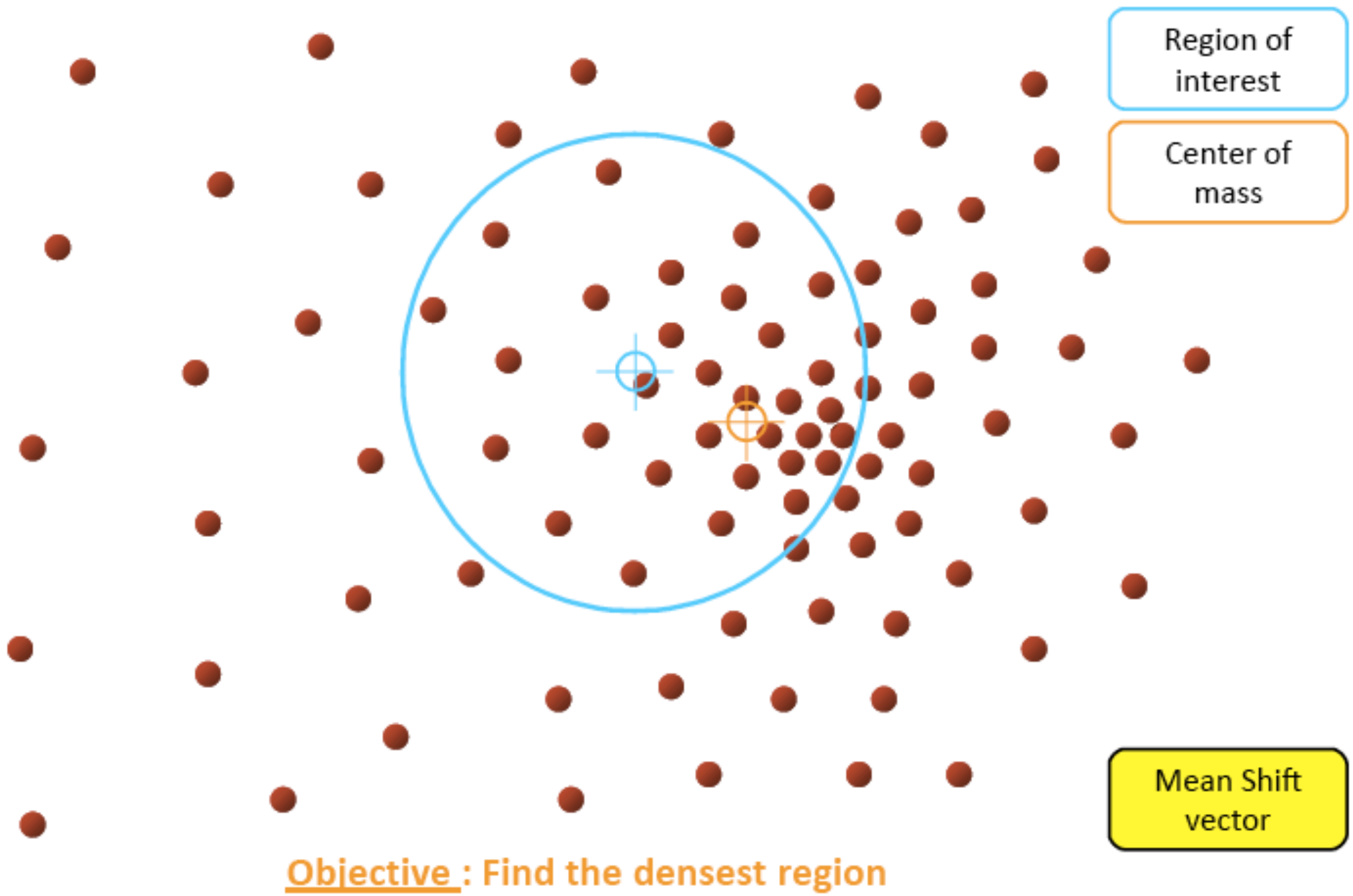
Ukrainitz&Sarel, Weizmann

Intuitive Description



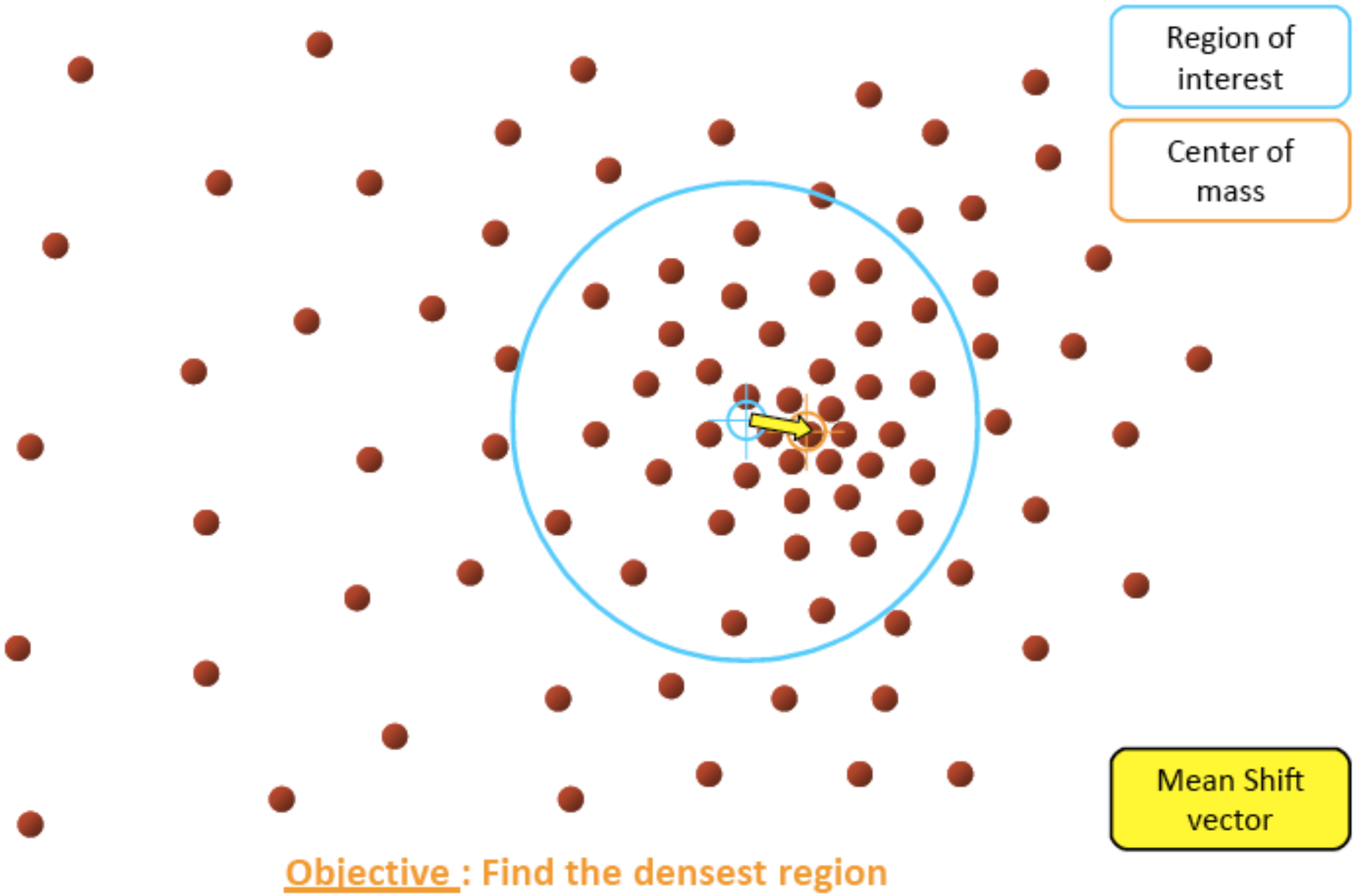
Ukrainitz&Sarel, Weizmann

Intuitive Description



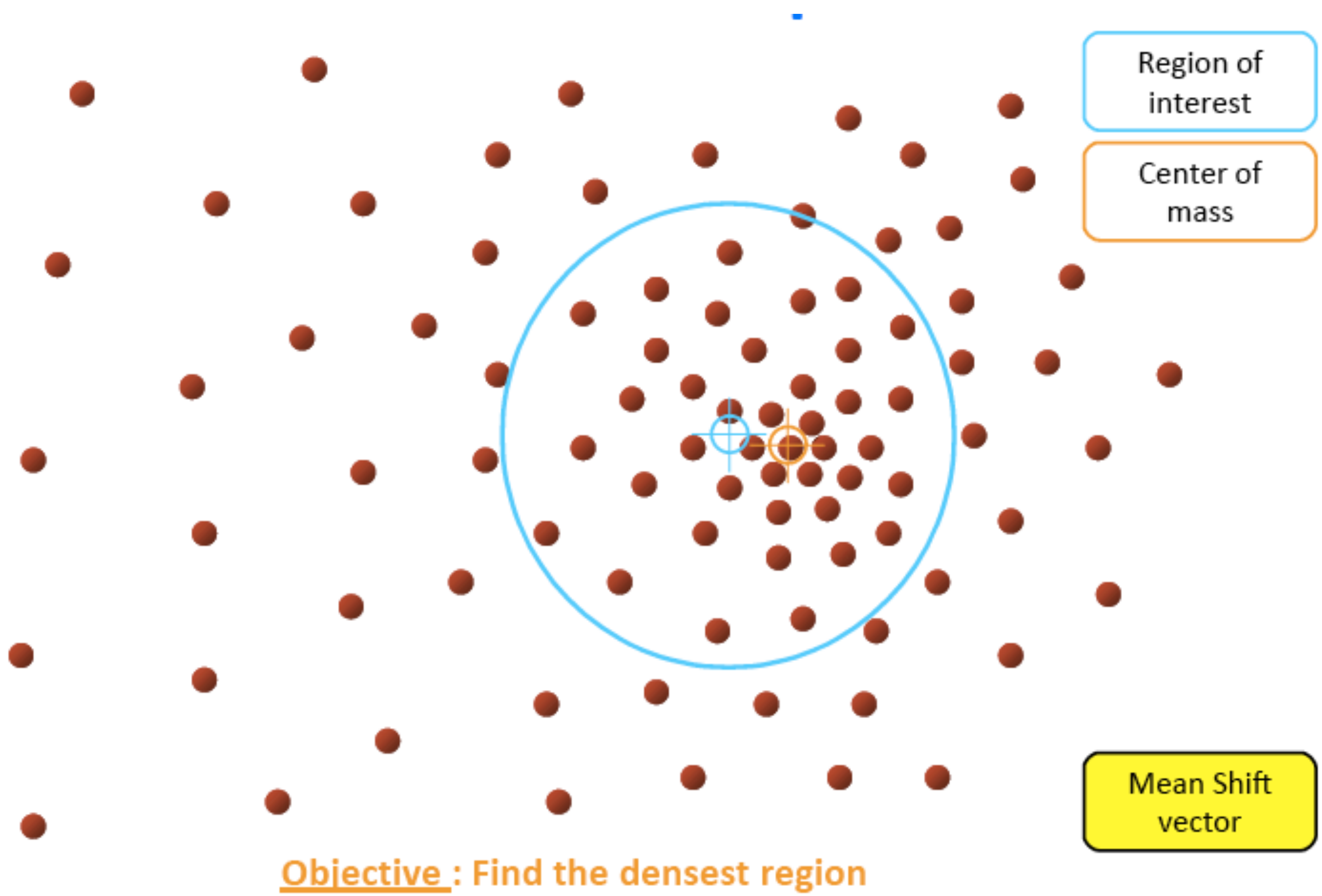
Ukrainitz&Sarel, Weizmann

Intuitive Description



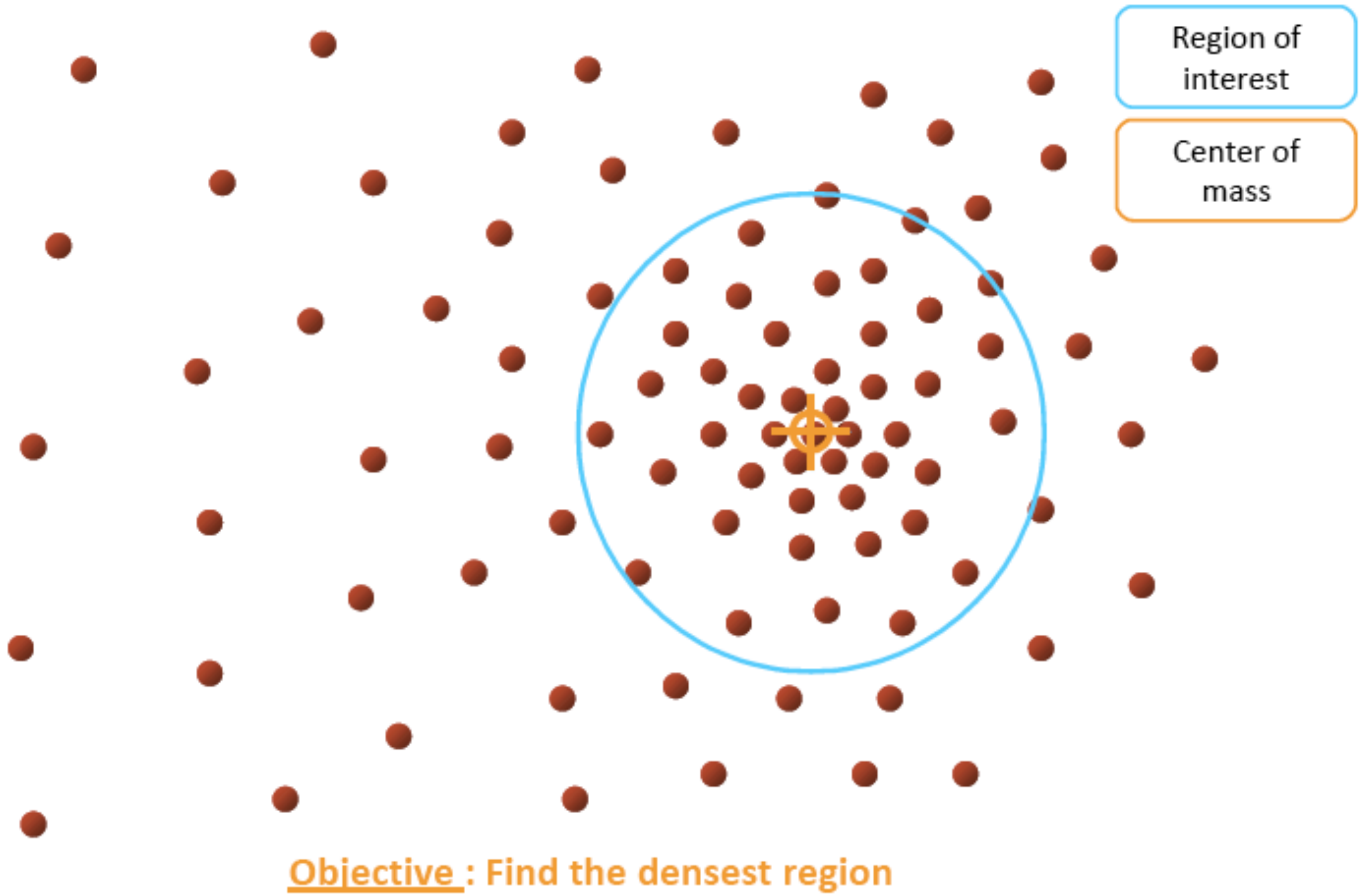
Ukrainitz&Sarel, Weizmann

Intuitive Description

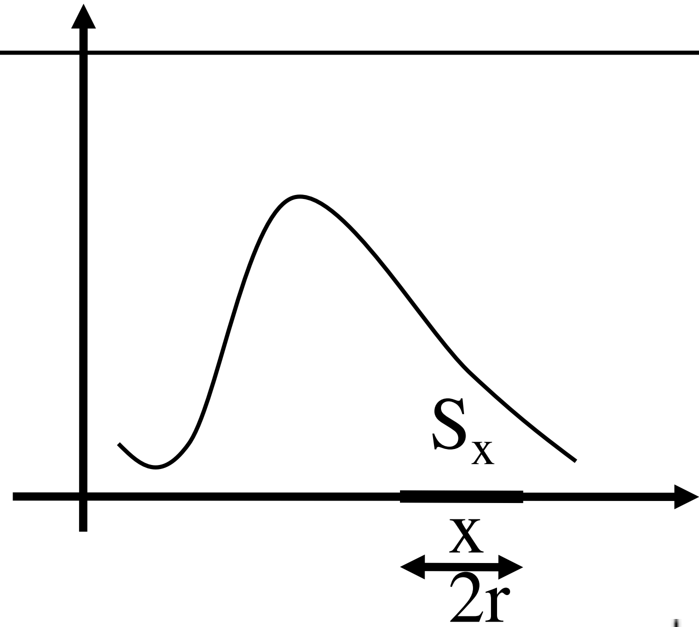


Ukrainitz&Sarel, Weizmann

Intuitive Description



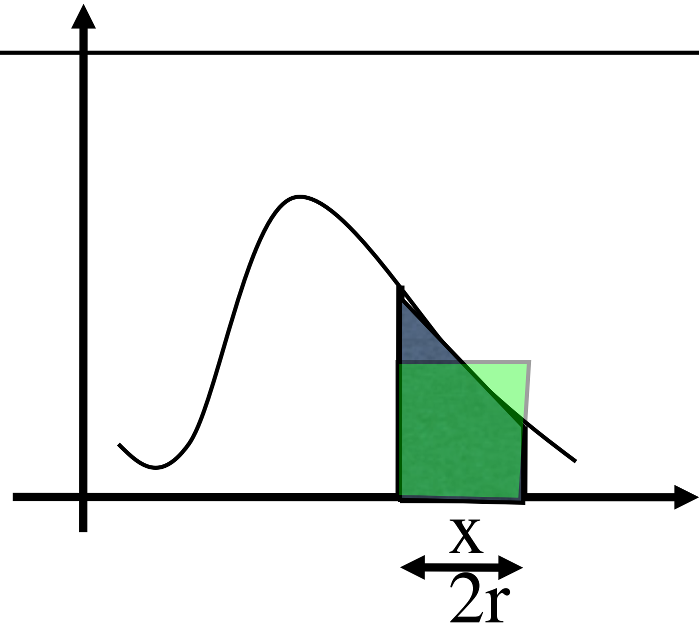
Ukrainitz&Sarel, Weizmann



Define $z = y - x$

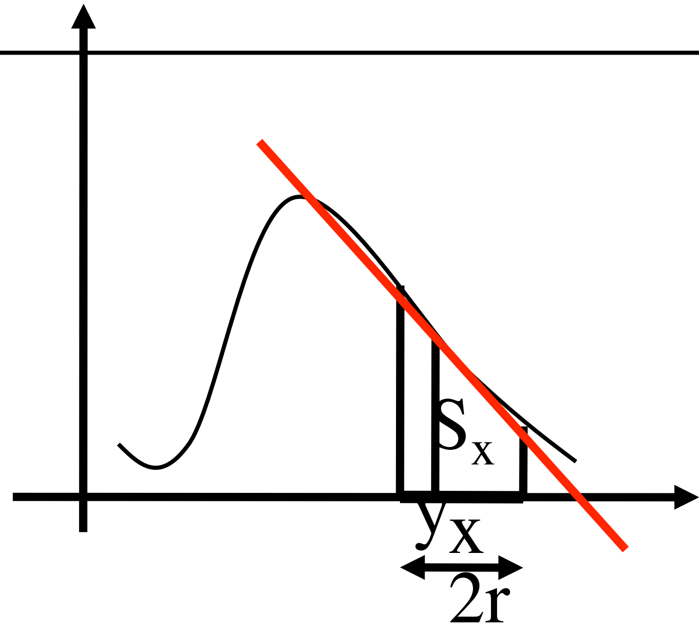
$$E[z|S_x] = \int_{x-r}^{x+r} (y - x) p(y|S_x) dy$$

$$E[z|S_x] = \int_{x-r}^{x+r} (y - x) \frac{p(y)}{p(y \in S_x)} dy$$



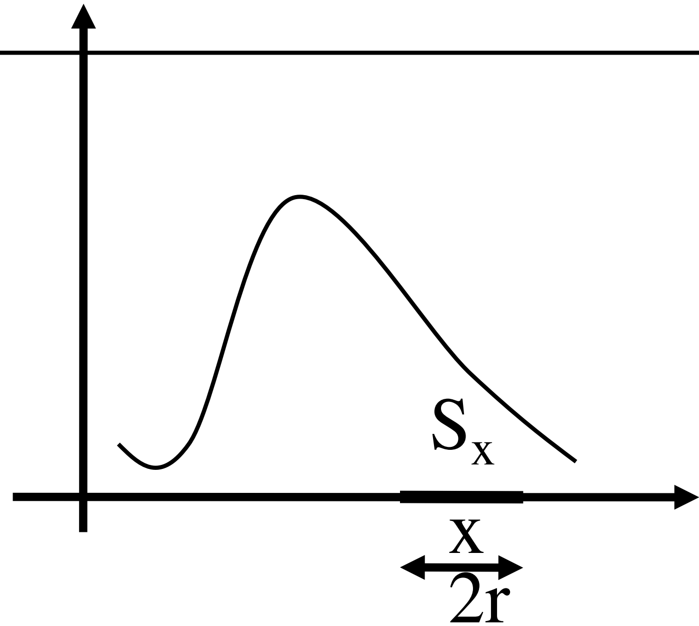
Define $z = y - x$

$$p(y \in S_x) = \int_{x-r}^{x+r} p(y)dy \approx p(x)2r$$



Define $z = y - x$

$$p(y) \approx p(x) + (y - x)p'(x)$$

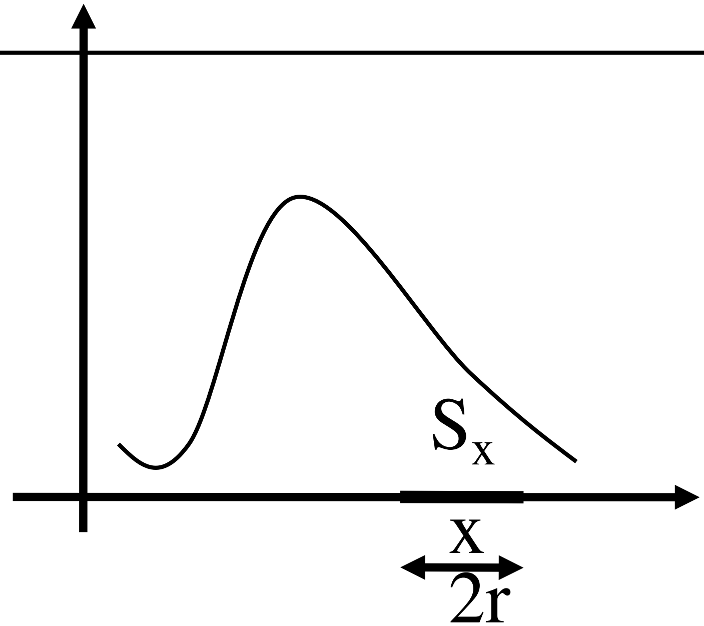


Define $z = y - x$

$$E[z|S_x] = \int_{x-r}^{x+r} (y - x) \frac{p(y)}{p(y \in S_x)} dy$$

$$p(y \in S_x) \approx p(x) 2r$$

$$E[z|S_x] \approx \frac{1}{2rp(x)} \int_{x-r}^{x+r} (y - x) p(y) dy$$

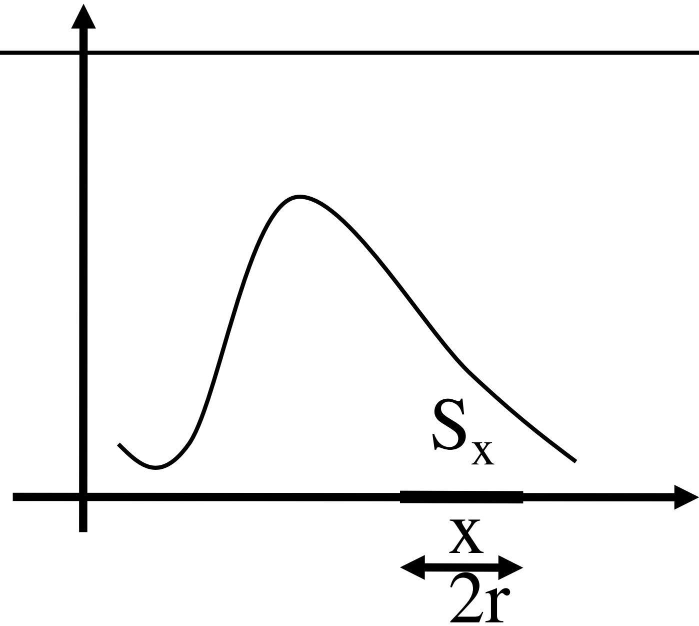


Define $$z = y - x$$

$$E[z|S_x] \approx \frac{1}{2rp(x)} \int_{x-r}^{x+r} (y-x)p(y)dy$$

$$p(y) \approx p(x) + (y-x)p'(x)$$

$$E[z|S_x] \approx \frac{1}{2rp(x)} \int_{x-r}^{x+r} (y-x)[p(x) + (y-x)p'(x)]dy$$



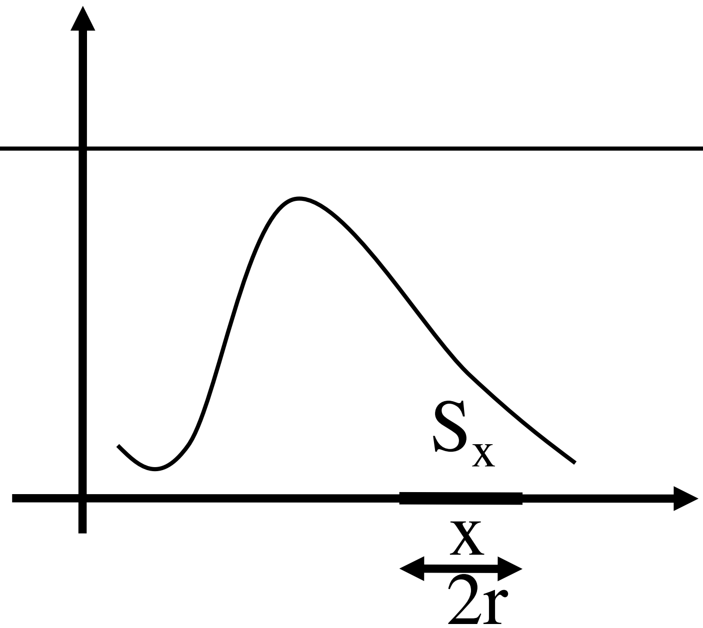
Define $z = y - x$

$$E[z|S_x] \approx \frac{1}{2rp(x)} \int_{x-r}^{x+r} (y-x)[p(x) + (y-x)p'(x)]dy$$

$$= \frac{1}{2rp(x)} \int_{-r}^{+r} (z)[p(x) + (z)p'(x)]dz$$

$$= \frac{1}{2rp(x)} [\frac{1}{2}z^2 p(x) + \frac{1}{3}z^3 p'(x)]_{-r}^{+r}$$

$$= \frac{1}{2rp(x)} \frac{2}{3} r^3 p'(x) = \frac{r^2 p'(x)}{3p(x)}$$



Define $z = y - x$

$$E(z|S_x) = \mu \approx \frac{r^2 p'(x)}{3p(x)}$$

μ is the "mean shift vector": the vector between the local mean E[y|y in S_x] and the center of the window x

- μ is proportional to p'(x)
- The factor of proportionality is reciprocal to p(x)
- Peaks have large p(x) but small p'(x)
- If not at the peak, μ is in the direction of the peak

Example: Comaniciu & Meer

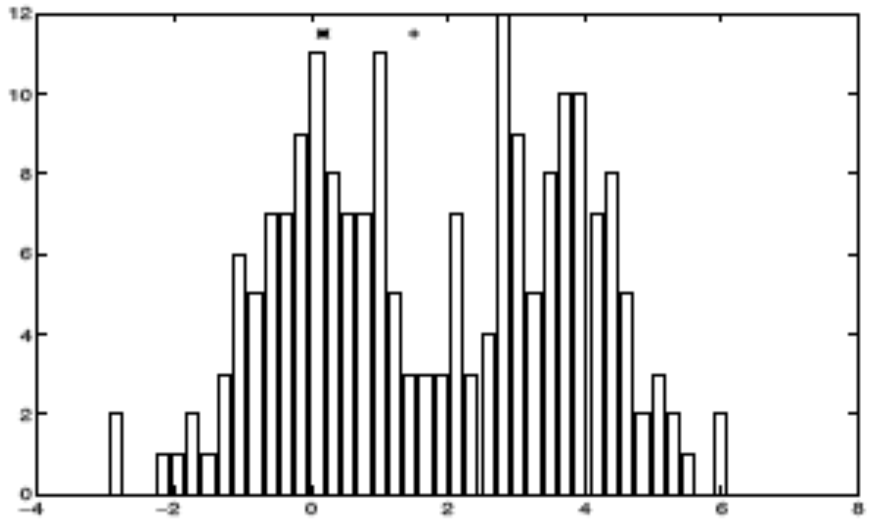


Figure 1: An example of the mean shift algorithm.

200 data pts
N(0,1) & N(3.5,1)

Mode detector:
Center of shortest window with half of the data points

Initial Mode	Initial Mean	Final Mean
1.5024	1.4149	0.1741

Initial window size = 3.2828

Example

#	0	1	3	5	3	1	1	1
gray	0	1	2	3	4	5	6	7

N=0+1+3+5+3+1+1+1=15

R =1.5, x0 = 5

4<= y <= 6

μ = (4*3+5*1+6*1)/(3+1+1) - 5 = 4.6-5 = -0.4

R=1.5, x1 = 4.6

Example

#	0	1	3	5	3	1	1	1
gray	0	1	2	3	4	5	6	7

N=0+1+3+5+3+1+1+1=15

R=1.5, x1 = 4.6

3<= y <=6

μ = (3*5+4*3+5*1+6*1)/(5+3+1+1) - 4.6 = 3.8 - 4.6 = -0.8

R =1.5, x2 = 3.8

Example

#	0	1	3	5	3	1	1	1
gray	0	1	2	3	4	5	6	7

N=0+1+3+5+3+1+1+1=15

R =1.5, x2 = 3.8

2<= y <= 5

μ = (2*3 +3*5+4*3+5*1)/(3+5+3+1) - 3.8 = 3.16-3.8 = -0.3

R=1.5, x3 = 3.16

Example

#	0	1	3	5	3	1	1	1
gray	0	1	2	3	4	5	6	7

N=0+1+3+5+3+1+1+1=15

R=1.5, x3 = 3.16

2<= y <=5

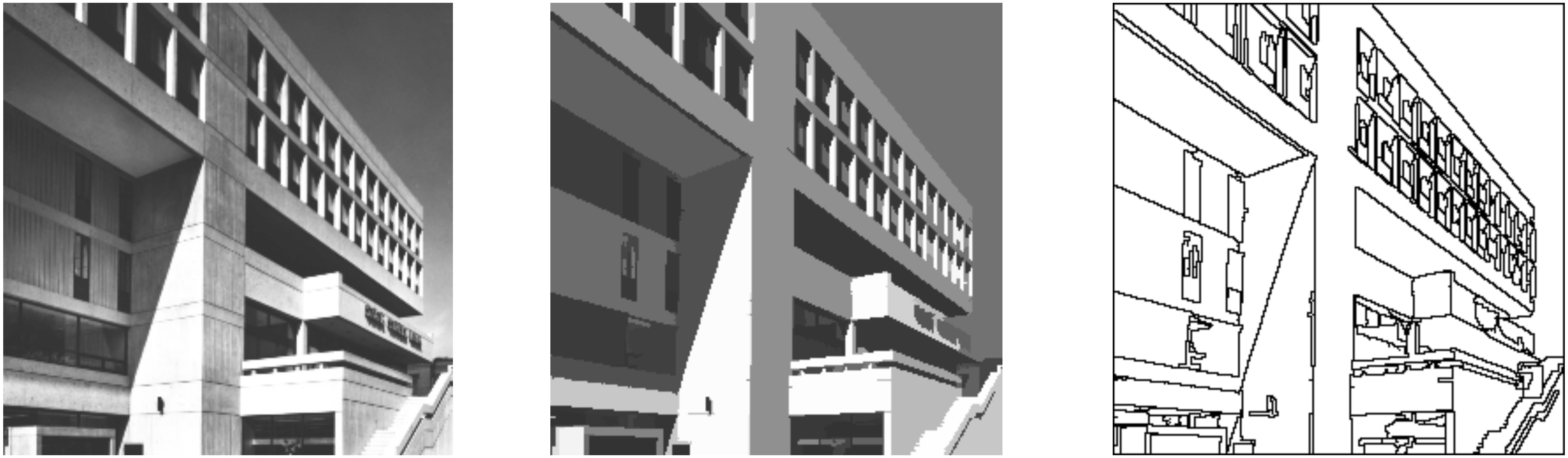
μ = (2*3+3*5+4*3+5*1)/(3+5+3+1) - 3.16 = 3.16 - 3.16 = 0

Algorithm MEAN SHIFT for Image Segmentation

1. Find image histogram, choose window size
2. Choose initial location of search window:
    1. Randomly select a number M of image pixels
    2. Find the average value in a 3x3 window for each of these pixels
    3. Set the center of the window to the value with largest histogram count.
3. Apply mean shift to find the window peak.
4. Remove pixels in the window from the image and the histogram.
5. Repeat steps 2 to 4 until no pixels are left.

Example of Mean Shift Segmentation

by D. Comaniciu and P. Meer



Limitations of histogram methods:

Use **GLOBAL** information

Ignore **SPATIAL** relationships among pixels.

SPLIT & MERGE Algorithms

Simple intensity algorithms usually result in too many regions.

- Reasons:
  - high frequency noise
  - Gradual transitions between regions

After segmentation, regions might need refinement:

- Interactively or automatically
- May use domain and or image process knowledge

Merging Algorithm

Merge **ADJACENT, SIMILAR** regions

What does “similar” mean?

“similar” average values : $|\mu_i - \mu_j| < T$

“small” spread of gray values: $|g_{max} - g_{min}| < T$

$g_{max} = \max\{g(x,y) \mid (x,y) \text{ in } R_i \cup R_j\}$

$g_{min} = \min\{g(x,y) \mid (x,y) \text{ in } R_i \cup R_j\}$

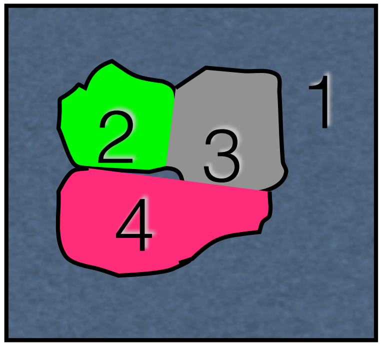
Note:

A similar to B, and B similar to C does not imply that A is similar to C.

Merging Algorithm

Start with an initial segmentation

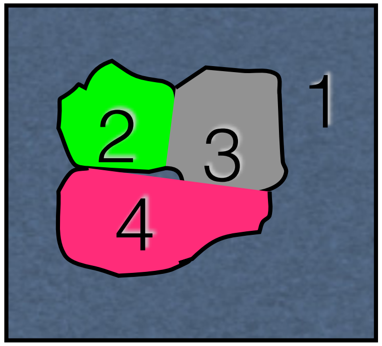
- Ex:
  - By thresholding,
  - nxn (5x5, 7x7, etc) regions
  - manually selected
- Each region has a unique “label”

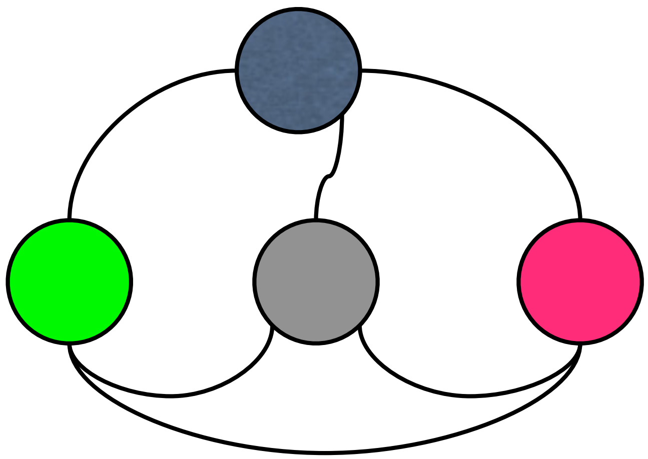


Merging Algorithm

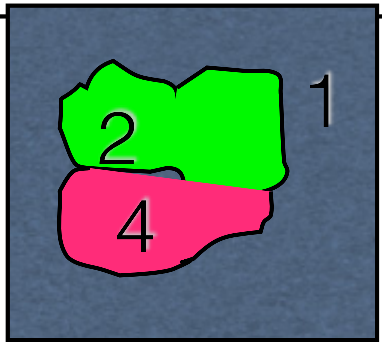
Form the Region Adjacency Graph

- Regions are the nodes
- Adjacency relations are the links



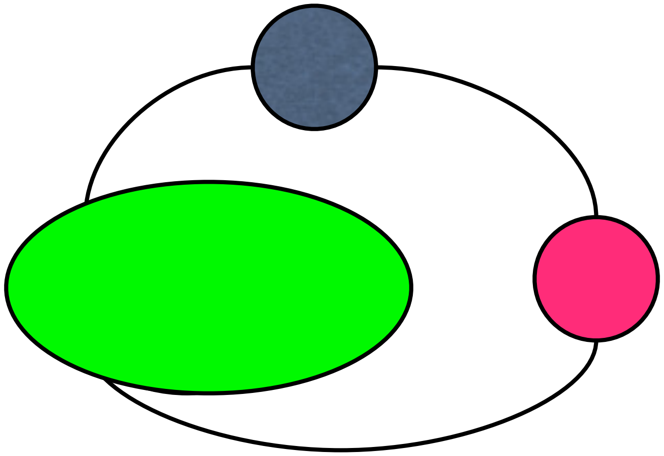


Merging Algorithm



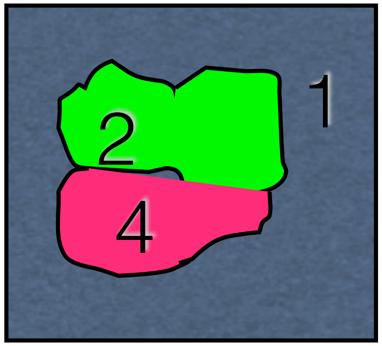
For each region in the image do:

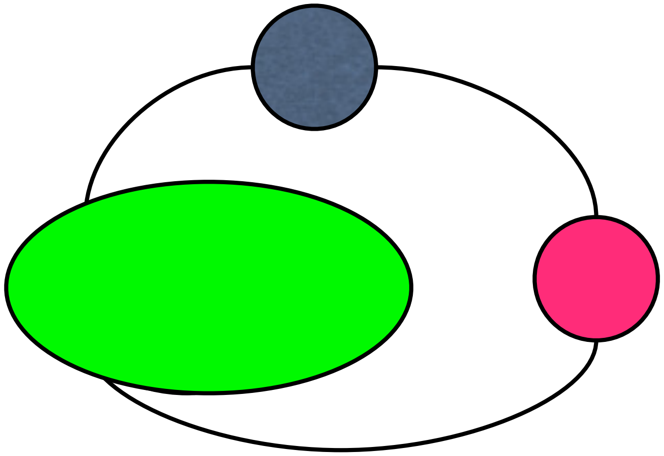
- Consider its adjacent regions and test if they are similar
- If they are similar, merge them and update the RAG



Merging Algorithm

Repeat the previous step until there are no more merges.





Using Similarity Measures

Compare mean intensities (zero order model)

Compare surface fitting (higher order model)

- Model regions as polynomial surfaces:
  - $F(x,y;a,m) = \sum_{i+j<=m} a_{ij}x^i y^j$
    - m is the polynomial degree (usually at most 2)
- Compute the fitting error:
  - $E^2(R,a,m) = \sum_{(x,y) \text{ in } R} [I(x,y) - F(x,y;a,m)]^2$
- Merge regions if it decreases the fitting error.

Using Statistics

Model pixels as drawn from probability distributions.

Different regions have different distributions.

Merge regions if their pixels came from the same distribution.

Example: One or Two Regions

Consider two regions:

Let $g_1, g_2, \ldots . g_{m1}$ be the gray values in R_1

Let $g_{m1+1}, g_{m1+2}, \ldots, g_{m1+m2}$ be the gray values in R_2

Assume regions are:

constant + uncorrelated zero mean Gaussian noise

Question:

Should the two regions be merged or not?

Two possible HYPOTHESES:

H_o:

Both regions belong to the same object and the gray values have distribution $N(\mu_o,\sigma_o)$

H_1:

Each region belongs to a different object and their gray values have distributions $N(\mu_1,\sigma_1)$ and $N(\mu_2,\sigma_2)$

Assume H_o is TRUE:

Estimate μ_o and σ_o:

$$\mu_o = \frac{1}{m_1 + m_2} \sum_{g_i \in R_1 \cup R_2} g_i$$

$$\sigma_o^2 = \frac{1}{m_1 + m_2} \sum_{g_i \in R_1 \cup R_2} (g_i - \mu_o)^2$$

"likelihood" of H_o being true:

Compute the probability of independently drawing $g_1, g_2 \ldots, g_{m1+m2}$ with distribution $N(\mu_o, \sigma_o)$:

$$P(g_1, g_2, ..., g_{m1+m2}|H_o) = P(g_1|H_o)P(g_2|H_o)...p(g_{m1+m2}|H_o)$$

$$P(g_i|H_o) = \frac{1}{\sqrt{2\pi}\sigma_o} e^{\frac{-(g_i-\mu_o)^2}{2\sigma_o^2}}$$

$$\begin{aligned}
P(g_1, g_2, ..., g_{m1+m2}|H_o) &= \prod_{i=1}^{m1+m2} \frac{1}{\sqrt{2\pi}\sigma_o} e^{\frac{-(g_i-\mu_o)^2}{2\sigma_o^2}} \\
&= \frac{1}{(\sqrt{2\pi}\sigma_o)^{m1+m2}} e^{\frac{-1}{2\sigma_o^2}\sum_{i=1}^{m1+m2}(g_i-\mu_o)^2} \\
&= \frac{1}{(\sqrt{2\pi}\sigma_o)^{m1+m2}} e^{\frac{-(m1+m2)}{2}}
\end{aligned}$$

Assume H_1 is TRUE:

Estimate μ_1 and σ_1, μ_2 and σ_2

$$\mu_j = \frac{1}{m_j} \sum_{g_i \in R_j} g_i$$

$$\sigma_j^2 = \frac{1}{m_j} \sum_{g_i \in R_j} (g_i - \mu_j)^2$$

$$j = 1, 2$$

"likelihood" of H_1 being true:

Compute the probability of independently drawing $g_1, g_2, \ldots, g_{m1}$ with distribution $N(\mu_1, \sigma_1)$, and $g_{m1+1}, g_{m1+2}, \ldots, g_{m1+m2}$ with distribution $N(\mu_2, \sigma_2)$:

$$P(g_1, g_2, ..., g_{m1+m2}|H_1) = P(g_1|H_1)P(g_2|H_1)...p(g_{m1+m2}|H_1)$$

$$g_i \in R_1 \rightarrow P(g_i|H_1) = \frac{1}{\sqrt{2\pi}\sigma_1} e^{\frac{-(g_i-\mu_1)^2}{2\sigma_1^2}} \qquad g_i \in R_2 \rightarrow P(g_i|H_1) = \frac{1}{\sqrt{2\pi}\sigma_2} e^{\frac{-(g_i-\mu_2)^2}{2\sigma_2^2}}$$

$$\begin{aligned}
P(g_1, g_2, ..., g_{m1+m2}|H_1) &= \prod_{i=1}^{m1} \frac{1}{\sqrt{2\pi}\sigma_1} e^{\frac{-(g_i-\mu_1)^2}{2\sigma_1^2}} \prod_{i=m1+1}^{m1+m2} \frac{1}{\sqrt{2\pi}\sigma_2} e^{\frac{-(g_i-\mu_2)^2}{2\sigma_2^2}} \\
&= \frac{1}{(\sqrt{2\pi})^{m1+m2}\sigma_1^{m1}\sigma_2^{m2}} e^{\frac{-1}{2\sigma_1^2}\sum_{i=1}^{m1}(g_i-\mu_1)^2} e^{\frac{-1}{2\sigma_2^2}\sum_{i=m1+1}^{m1+m2}(g_i-\mu_2)^2} \\
&= \frac{1}{(\sqrt{2\pi})^{m1+m2}\sigma_1^{m1}\sigma_2^{m2}} e^{\frac{-m1}{2}} e^{\frac{-m2}{2}}
\end{aligned}$$

Choose the most "likely" Hypothesis

$$L = \frac{P(g_1, ..., g_{m1+m2}|H_1)}{P(g_1, ...g_{m1+m2}|H_o)} = \frac{\sigma_o^{m1+m2}}{\sigma_1^{m1}\sigma_2^{m2}}$$

If $L < 1$, H_o is more likely than H_1:

Merge the regions!

Splitting Algorithms

When are regions split?

- Split a region if:
  - A property is not “constant”
  - A predicate is not TRUE
- Deciding to split is fairly straight-forward.

Where to split?

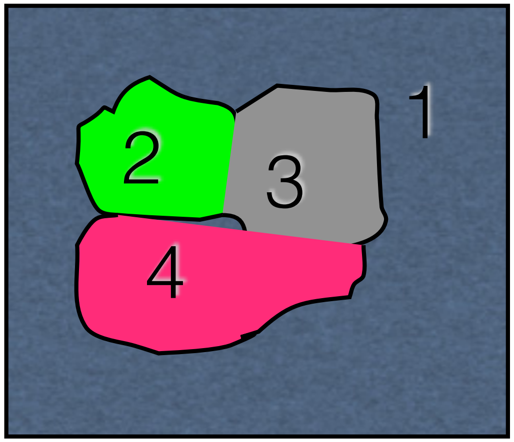
- This is a difficult problem.
- Some approaches:
  - Divide it into equal parts along image dimensions.
  - Look for strong edges to create boundaries.

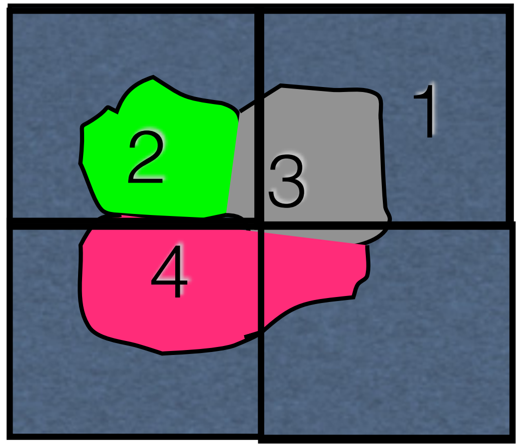
Split and Merge Algorithms

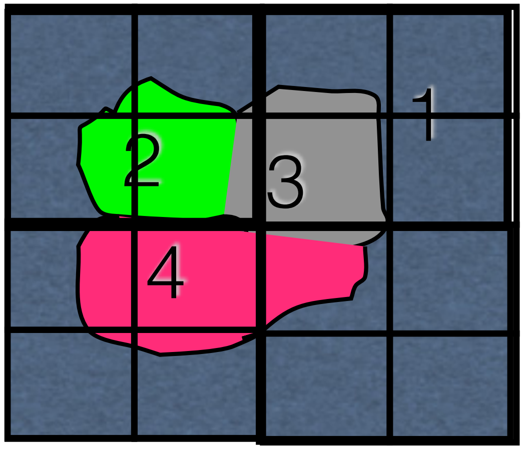
Split and merge are often used together:

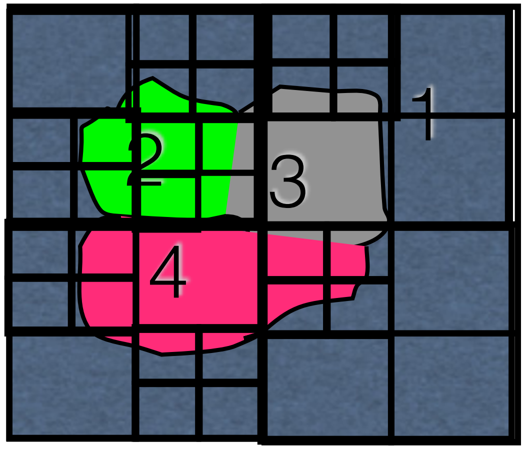
- Start with the initial image and a “predicate”
- Test the image with the predicate:
  - If it doesn’t satisfy it, split image into quarters;
  - Repeat for each sub-region until there are no more splits.
- Test adjacent regions with the predicate:
  - If they satisfy it: merge them.

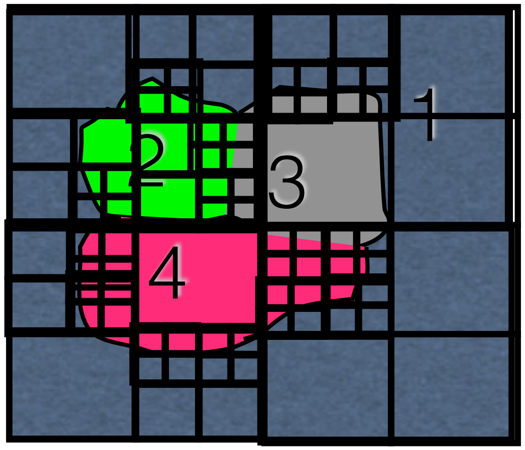
Split ...



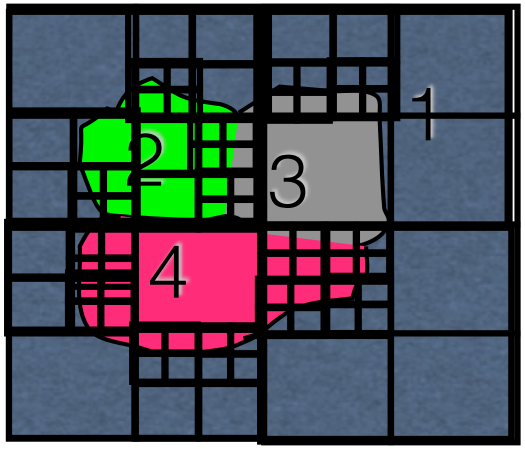


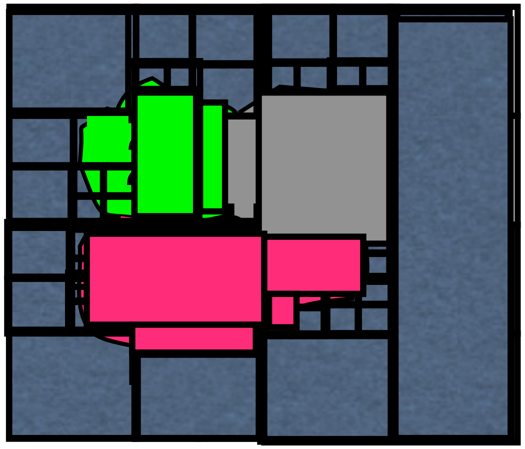


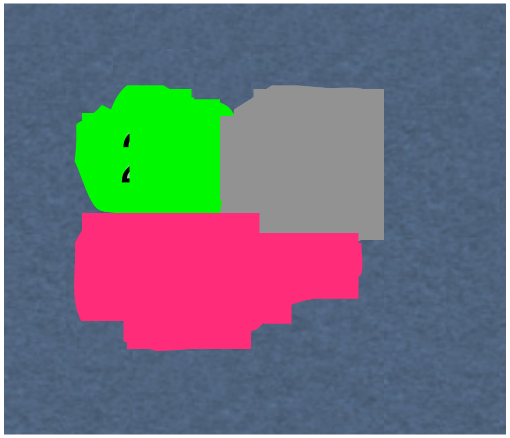




Merge …







Quadtree Representation

Quadtrees:

- Trees where nodes have 4 children

Build quadtree:

- Nodes represent regions
- Every time a region is split, it’s node gives birth to 4 children
- Leaves are nodes for uniform regions

Merging:

- Siblings that are “similar” can be merged.

Quadtree Representation