Computer Vision I

Segmentation - 10.07.2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Segmentation

- What do we mean by segmentation and why do we need it?
  - Segmentation can roughly be described as the grouping of similar information in an image.
  - Instead of having to work with all the pixels, a segmentation allows us to work with a much more compact representation.
  - This is useful in practice, because this compact representation can make it easier to carry out certain tasks.
    - Scene understanding, object recognition, ...
  - Sometimes, we are interested in the segmentation itself.
    - Especially in medical image analysis (e.g., segmenting out a tumor)

Some Examples

[Ren & Malik, 03]

Figure-Ground Separation

- One very useful way of thinking about segmentation is that it enables the separation of the figure (i.e., foreground) from the background.
- Example:

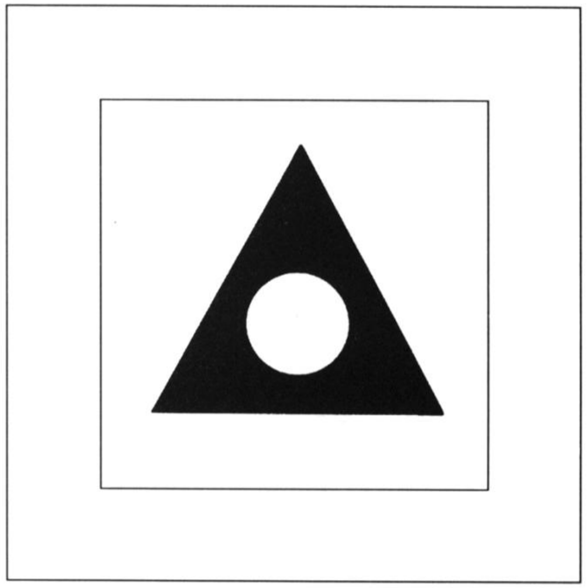
Full image

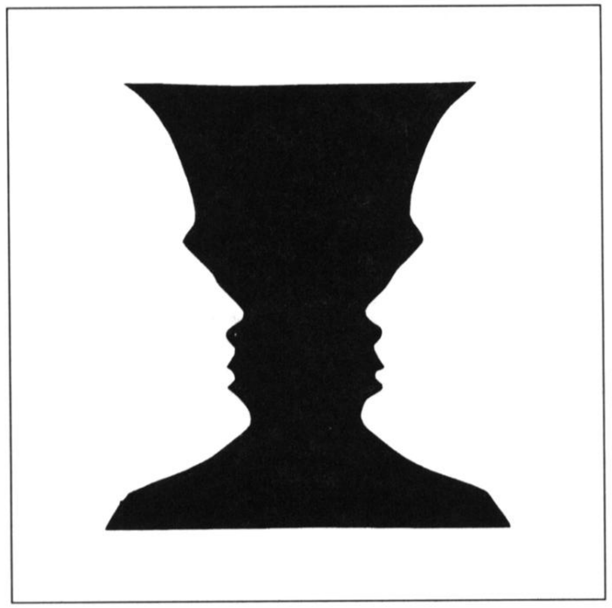
Figure (foregound) portion

[Ren et al., 05]

Figure-Ground Separation

- This separation may be ambiguous, i.e. while we may be able to separate figure and ground, we may not be able to decide which is which:



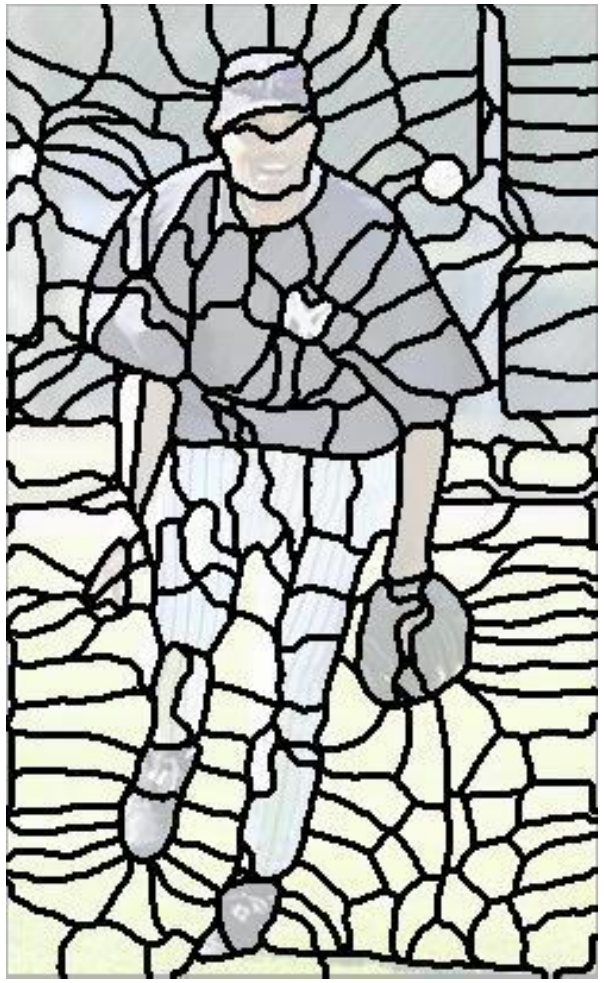


  - The white circle may be the figure on a black ground, or a black mask with a hole is the figure on a white ground.
  - Vase or faces?

[Gordon]

Superpixels

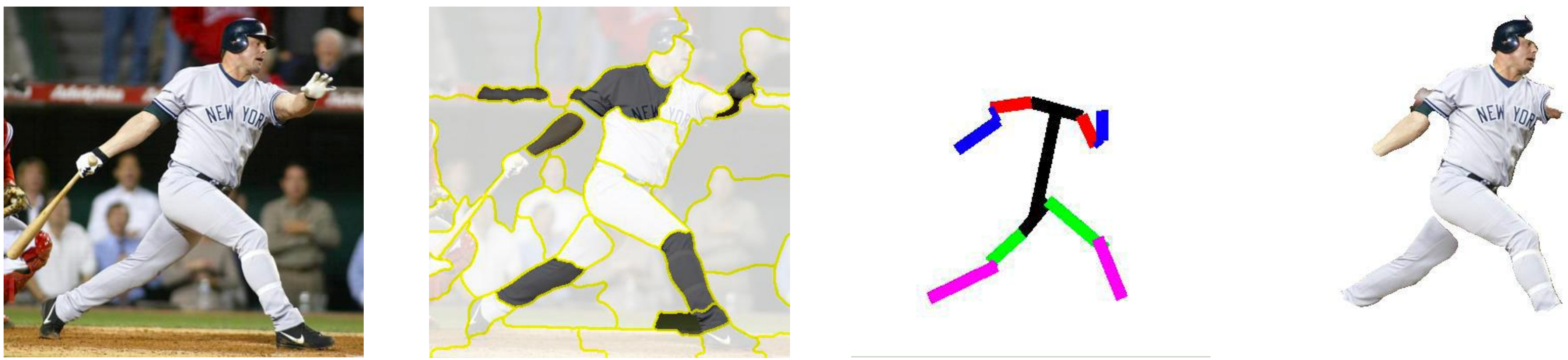
- Superpixels are a form of segmentation, in which the goal is to find many small segments that can substitute using the actual pixels:



[Mori et al., 04]

Superpixels

- Each superpixel is not necessarily a meaningful image part, but instead really like a big pixel.
  - There should be many fewer superpixels than actual pixels.
  - For certain tasks, this can amount to substantial computational savings.
  - E.g., pose estimation:



[Mori et al., 04]

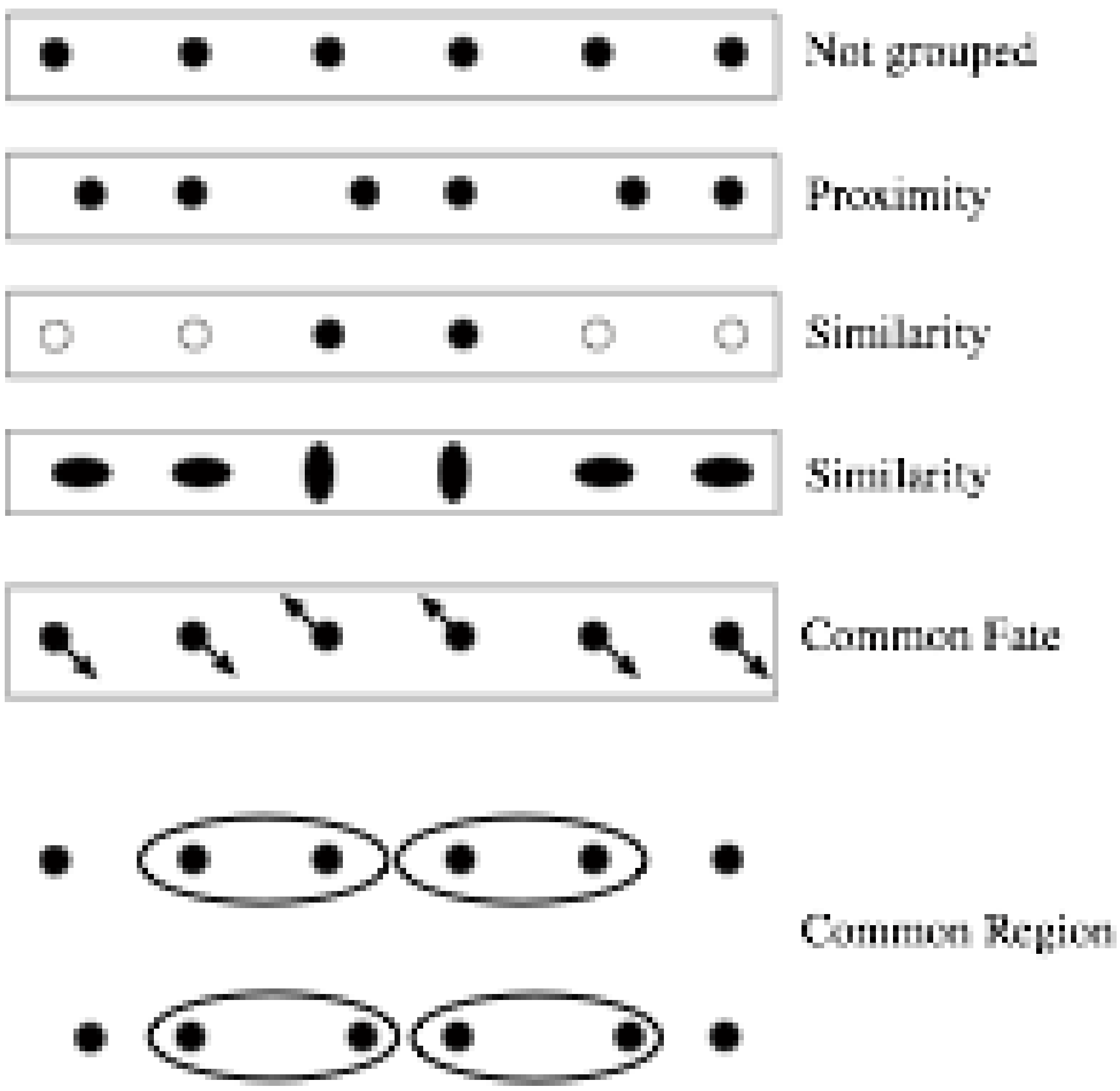
Superpixels

- Problem 1:
  - How many superpixels (segments) do we need?
  - This is a general problem in segmentation.
- Problem 2 (related):
  - What if the superpixels group together things that shouldn't be grouped?
- Superpixels can significantly ease certain problems, but we cannot blindly trust them.
  - We may need to "go back" to the actual pixels.
  - Of course, this does not prevent us from using them to obtain good and quick initializations.

What belongs together?

- In order to perform image segmentation, we need to decide which parts of the image belong together.
- We can draw inspiration from various sources.
  - As before, we can try to think about what makes us humans believe parts of an image belong together.
  - Early work: Gestalt psychology in the early 20th century.
    - Max Wertheimer was one of the leading figures.

Gestalt Factors

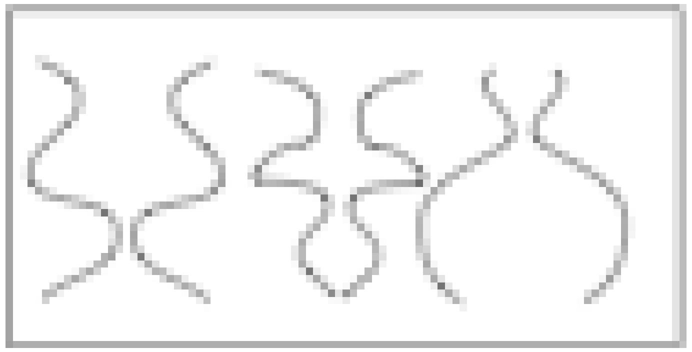


[Gordon]

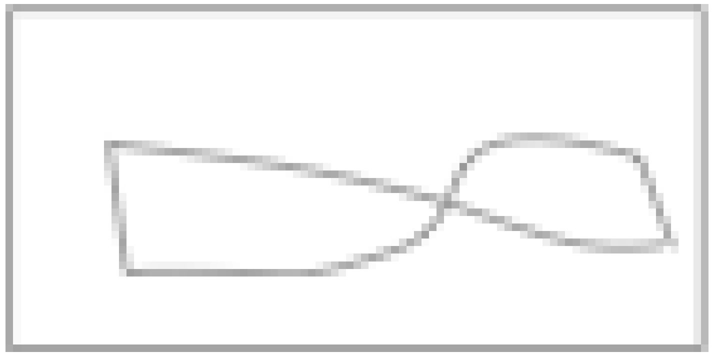
Gestalt Factors

Parallelism

Continuity



Symmetry



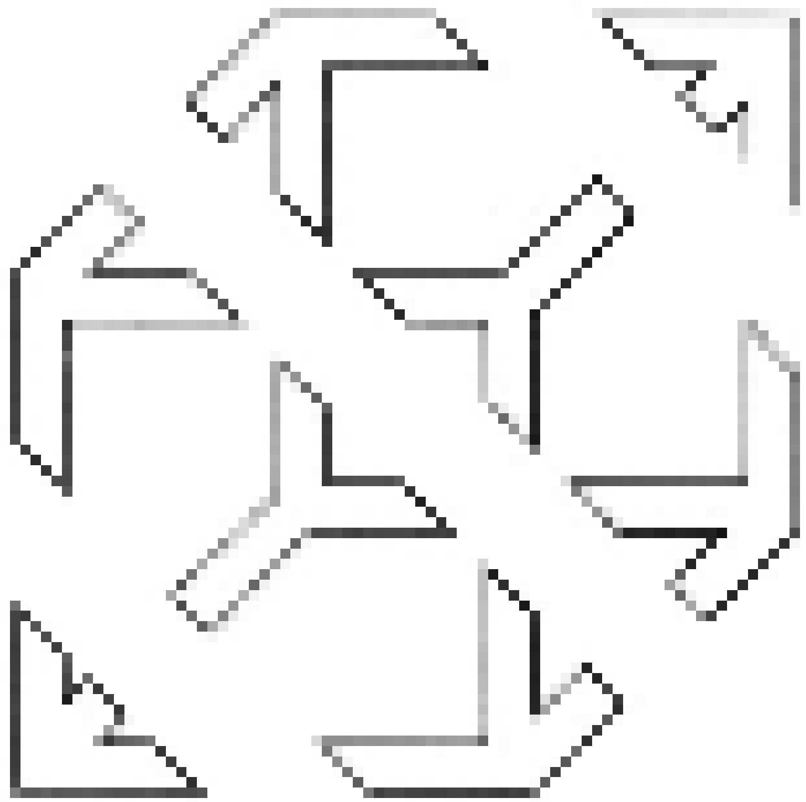
Closure

[Gordon]

- These factors offer some insights as to what may be useful from a computer vision point of view.
- Turning them into an algorithm is difficult, however.

Importance of Occlusion

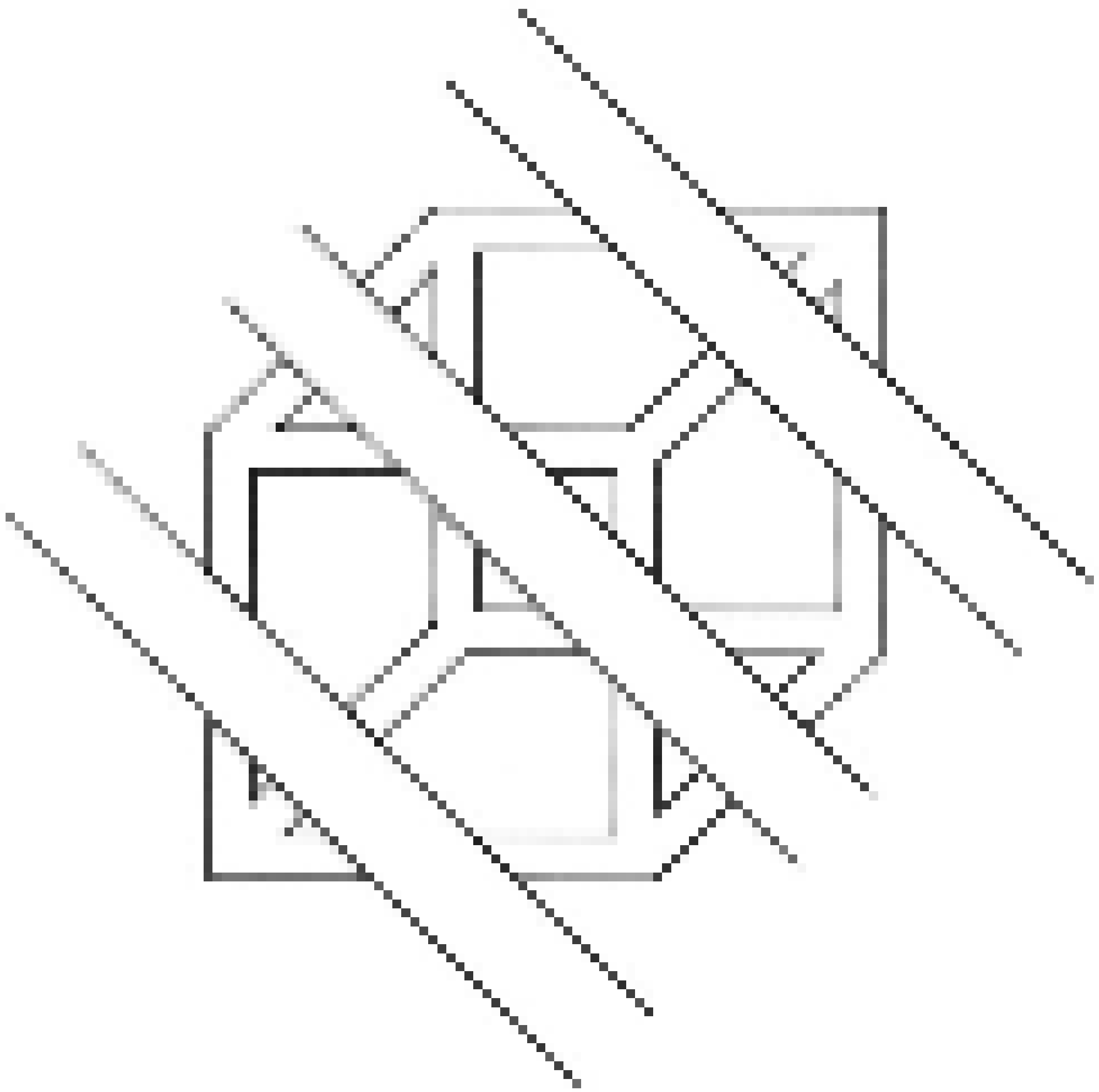
- What do you see?



[Gordon]

Importance of Occlusion

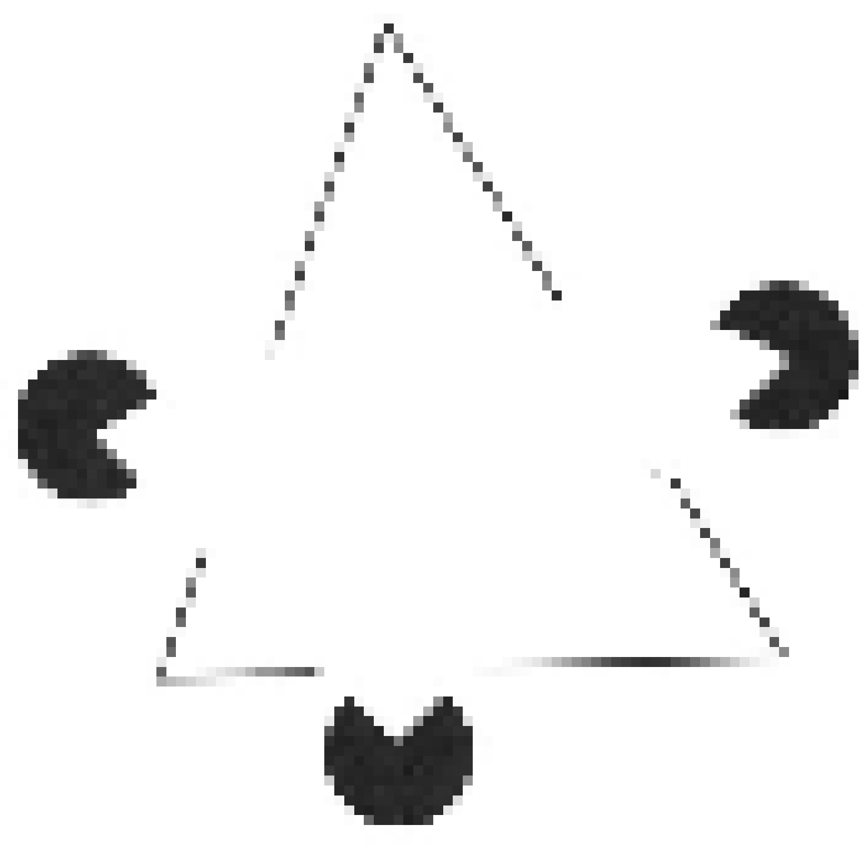
- What do you see now?



[Gordon]

Illusory Contours

- Kanisza triangle (similar):



[Marr]

“Ultimate Challenge”

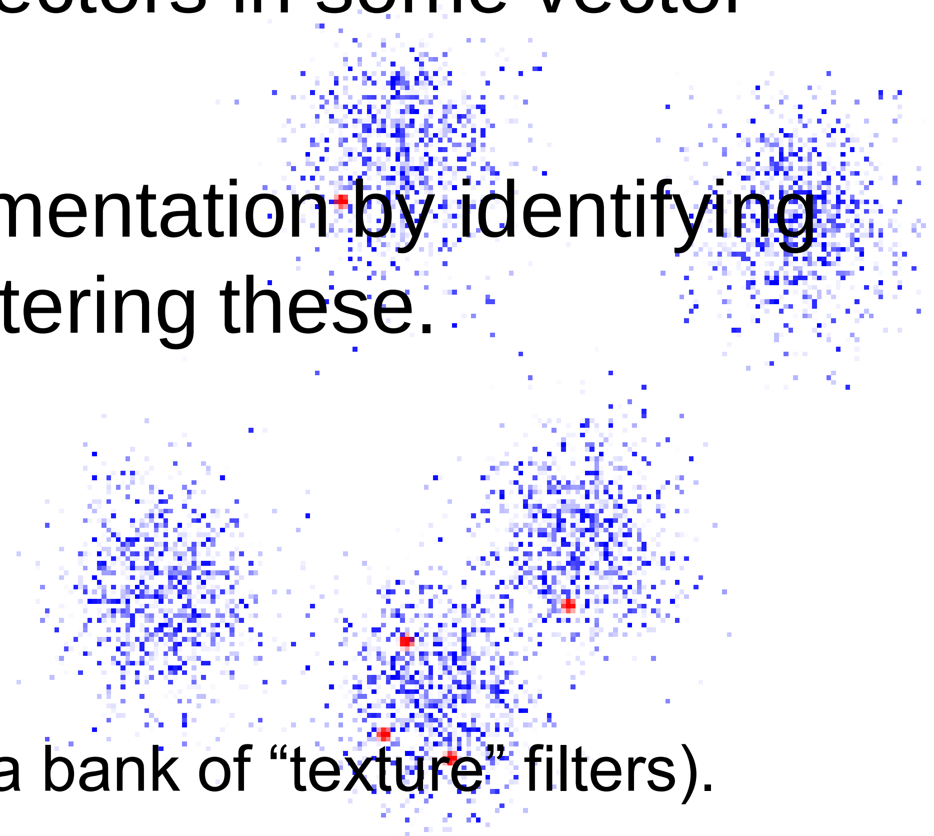
[Marr]

Conclusions so far

- From these examples & rules we can see that:
  - Segmentation is generally a quite difficult problem.
  - It is hard to even characterize what it is.
  - We humans seem to be very good at it, which suggests that it is somehow important for our visual processing.
- Most of these cases are very very challenging to implement on a computer:
  - We will only be able to do something rather simple.
  - In particular, we will not be able to solve these examples.
  - But what we can do is still useful.

Segmentation by Clustering

- One simple way of performing segmentation is to use clustering algorithms:
  - Clustering (a problem from machine learning) tries to group data points together. The points are usually vectors in some vector space.
  - We can apply this to the problem of segmentation by identifying each pixel with a feature vector and clustering these.
  - This feature vector may include:
    - The pixel's position
    - Pixel intensity or color
    - A description of the local texture (e.g. output of a bank of "texture" filters).



Simple Clustering Methods

- Agglomerative clustering:

```
Make each point a separate cluster

Until the clustering is satisfactory

    Merge the two clusters with the
    smallest inter-cluster distance

end
```

- Divisive clustering:

```
Construct a single cluster containing all points

Until the clustering is satisfactory

    Split the cluster that yields the two
    components with the largest inter-cluster distance

end
```

[FP]

Simple Clustering Methods

- Both of these techniques may be applied, but can be slow and may need "hacking" to produce good results.
- Better technique: K-means clustering

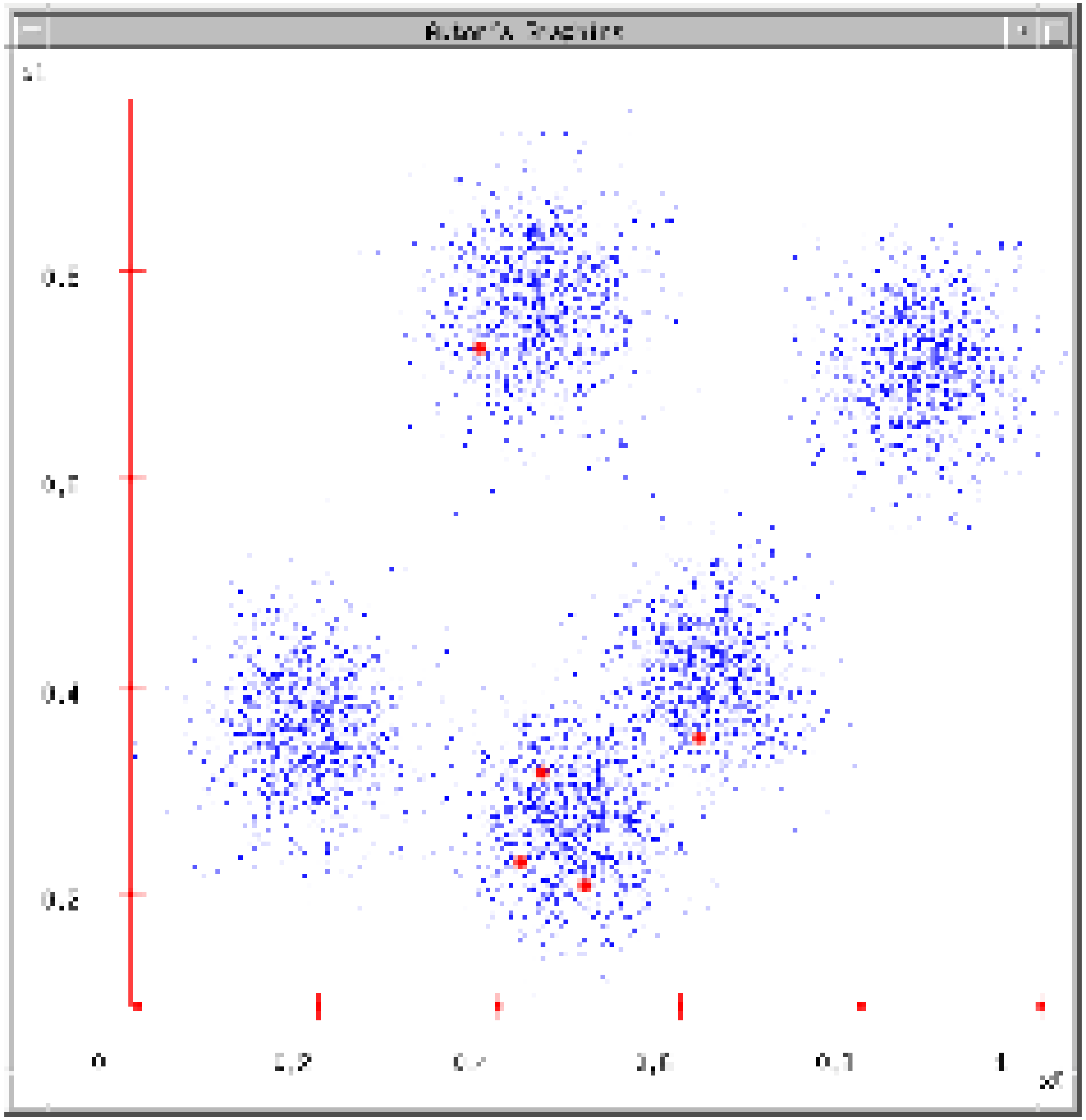
```
Choose k data points to act as cluster centers

Until the cluster centers are unchanged

    Allocate each data point to cluster whose center is nearest

    Now ensure that every cluster has at least
    one data point; possible techniques for doing this include .
    supplying empty clusters with a point chosen at random from
    points far from their cluster center.

    Replace the cluster centers with the mean of the elements
    in their clusters.

end
```

[FP]

K-Means

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations

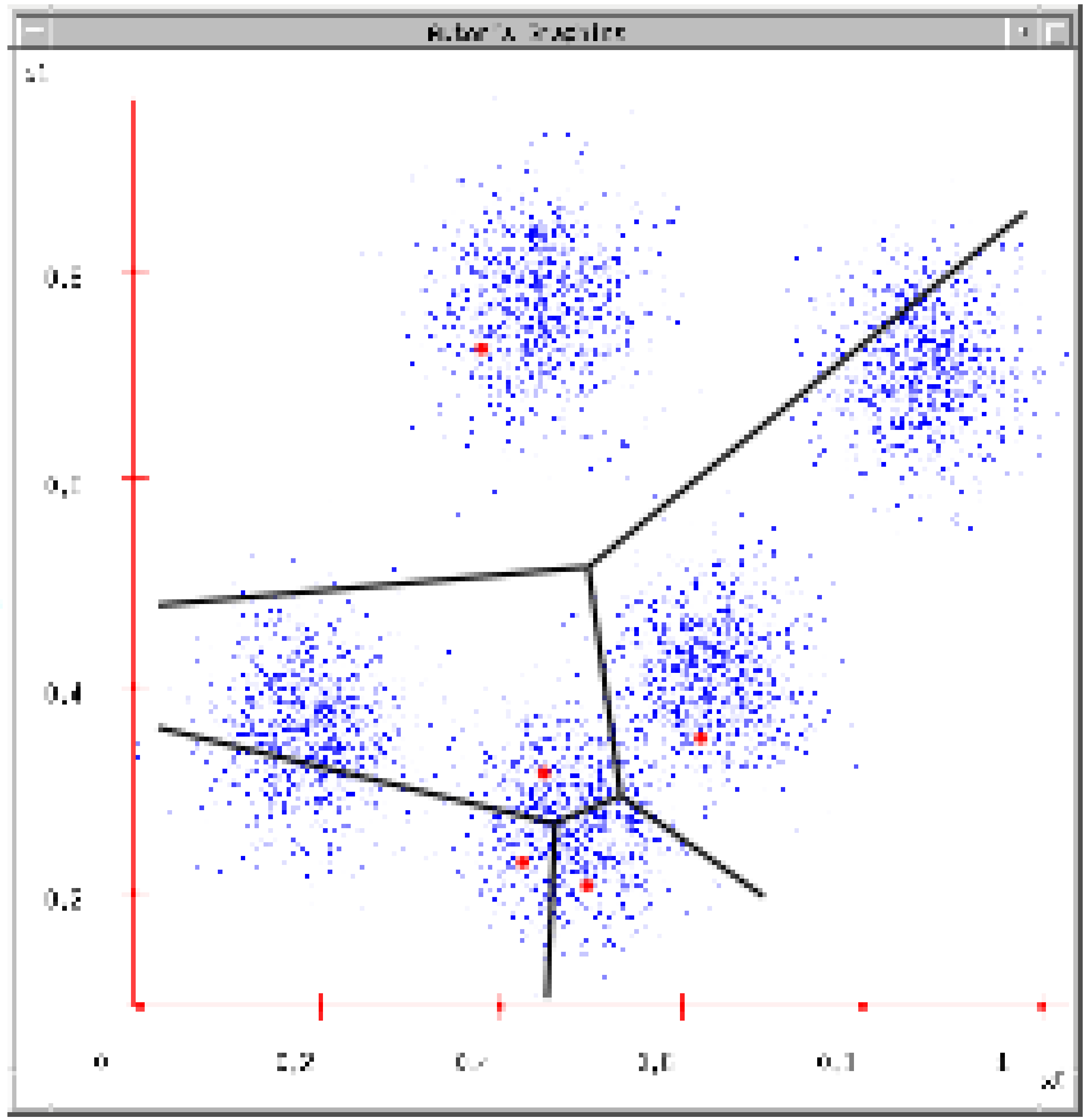


 K-means and Hierarchical Clustering: Slide 7

K-Means

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)

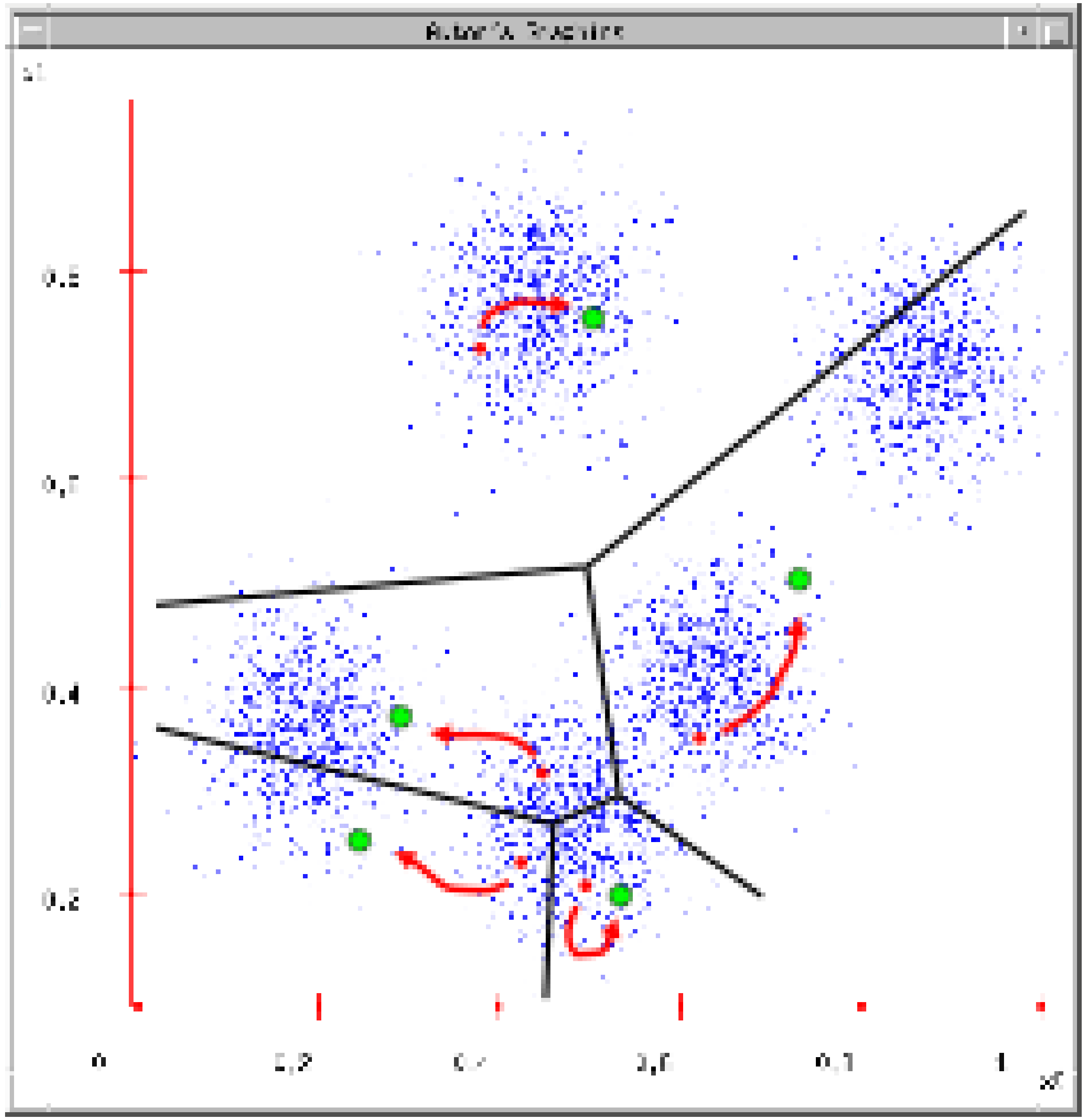


K-means and Hierarchical Clustering: Slide 8

K-Means

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns

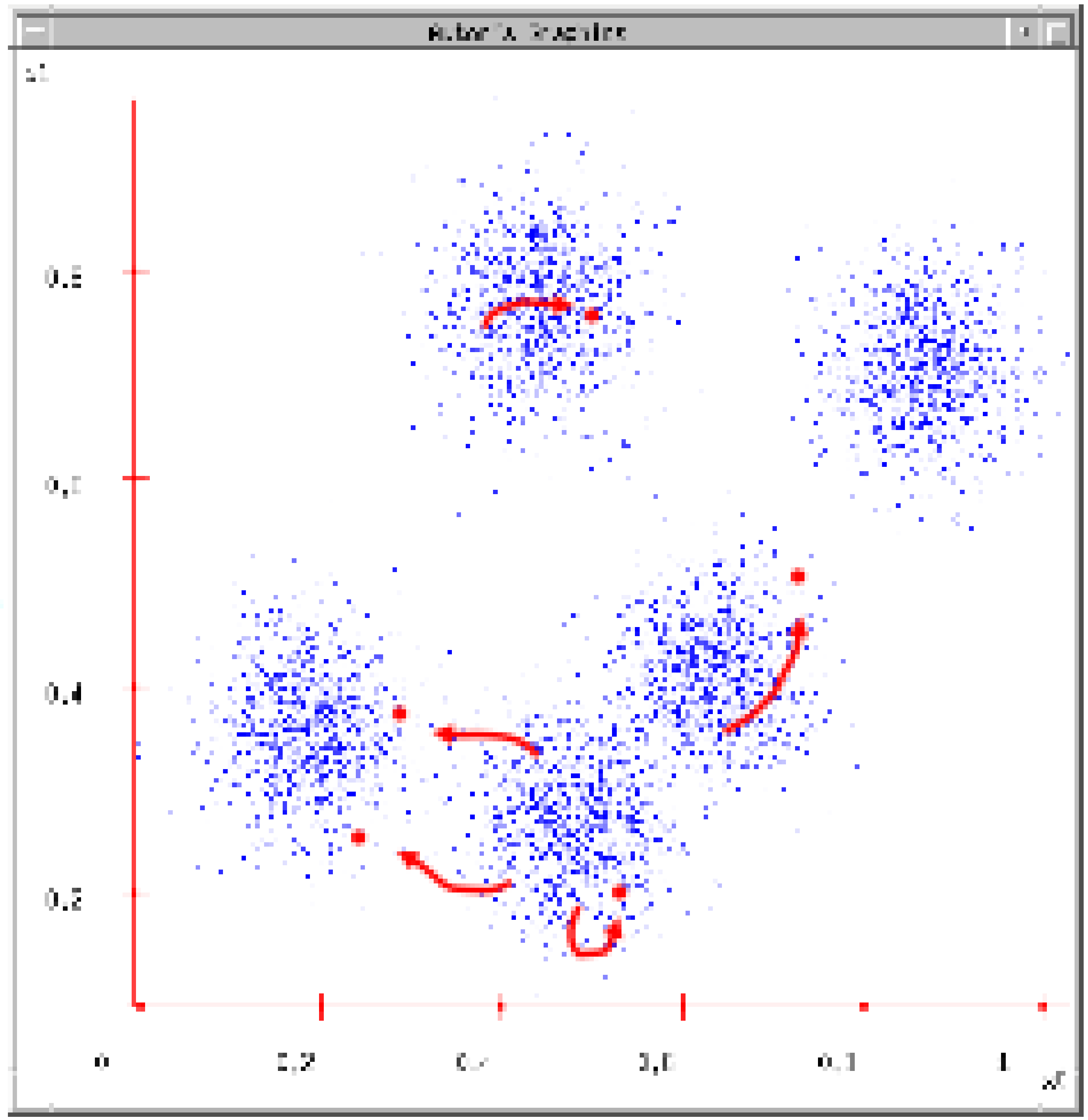


K-means and Hierarchical Clustering: Slide 9

K-Means

K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



K-means and Hierarchical Clustering: Slide 10

K-Means

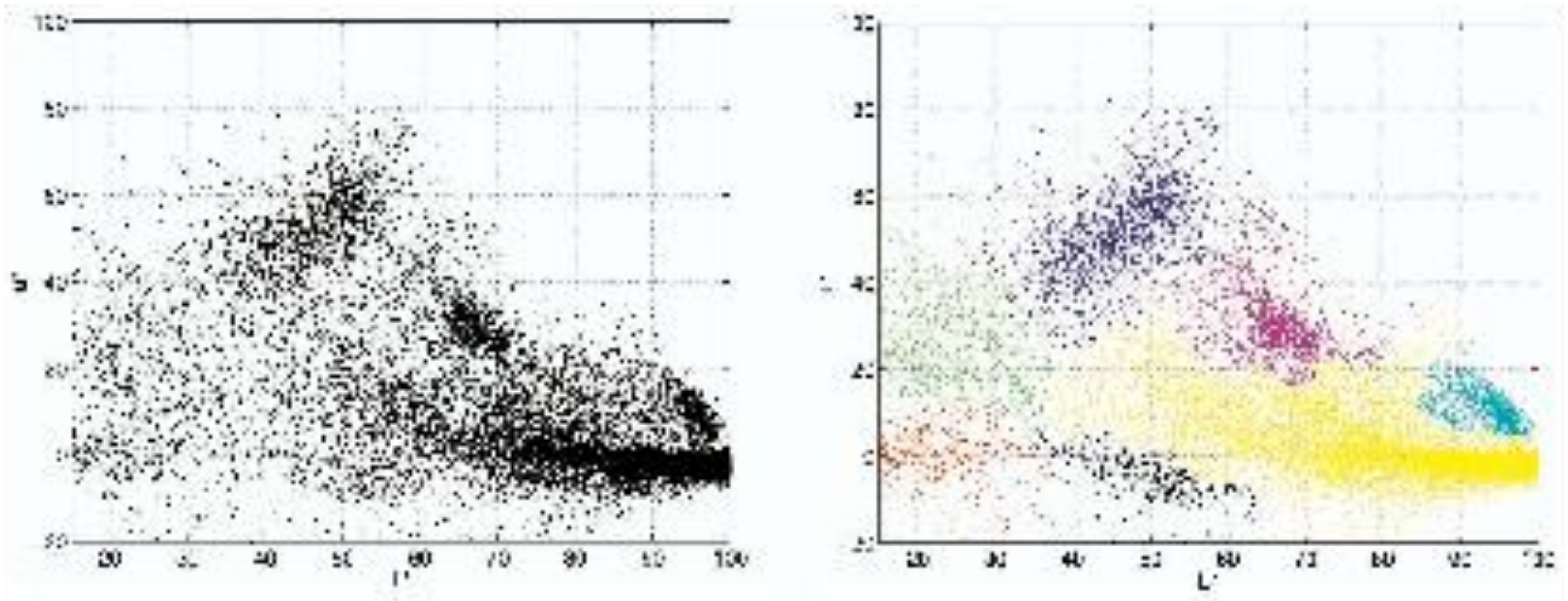
- K-Means is quite easily to implement and reasonably fast.
- Other nice property: We can understand it as the local optimization of an objective function:

$$\Psi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{-th cluster}} ||\boldsymbol{x}_j - \boldsymbol{c}_i||^2 \right\}$$

- Problem (of many segmentation approaches):
  - How do we know which is the right number of clusters k?

Mean Shift Clustering

- Mean shift is a method for finding modes in a cloud of data points where the points are most dense.

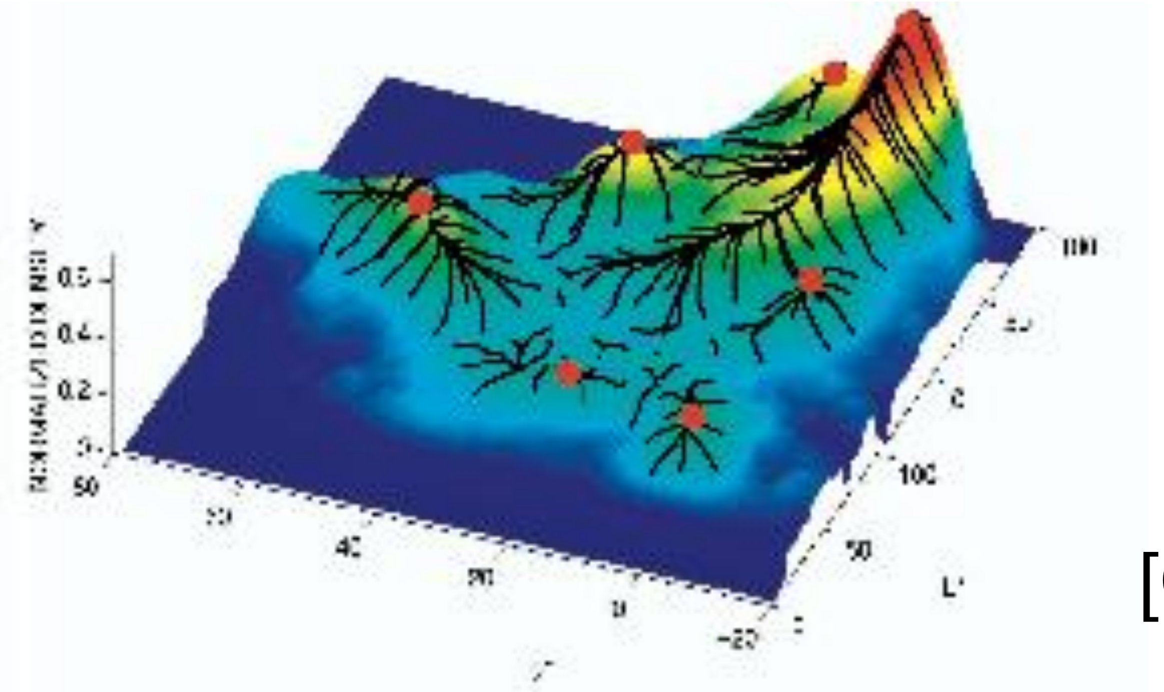


- To use this for segmentation, we use feature vectors to describe the pixels, just as before.

[Comaniciu & Meer, 02]

Mean Shift

- The mean shift procedure estimates a density out of the data points and finds various modes of the density through local search.



[Comaniciu & Meer, 02]

  - The black lines indicate various search paths starting at different points.
  - Paths that converge at the same point get assigned the same label.

Kernel Density Estimate

- If we have given a set of points $\mathbf{x}_i$, we can estimate the probability density from which they were sampled using a so-called kernel density estimate:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh^d} \sum_{i=1}^{N} k\left(\left\| \frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right\|^2 \right)$$

  - Here, $k(\cdot)$ is a kernel function with width h .
  - This is a so-called non-parametric density estimate.
- We can derive the mean shift procedure by taking the gradient of this estimate.
  - We will skip this here and only look at the final result...

Mean Shift

- **Procedure:**
  - Start at a random data point.
  - Compute the mean shift vector:

$$m_{h,g}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} \boldsymbol{x}_i g\left(\left\|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{N} g\left(\left\|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right\|^2\right)} - \boldsymbol{x}$$

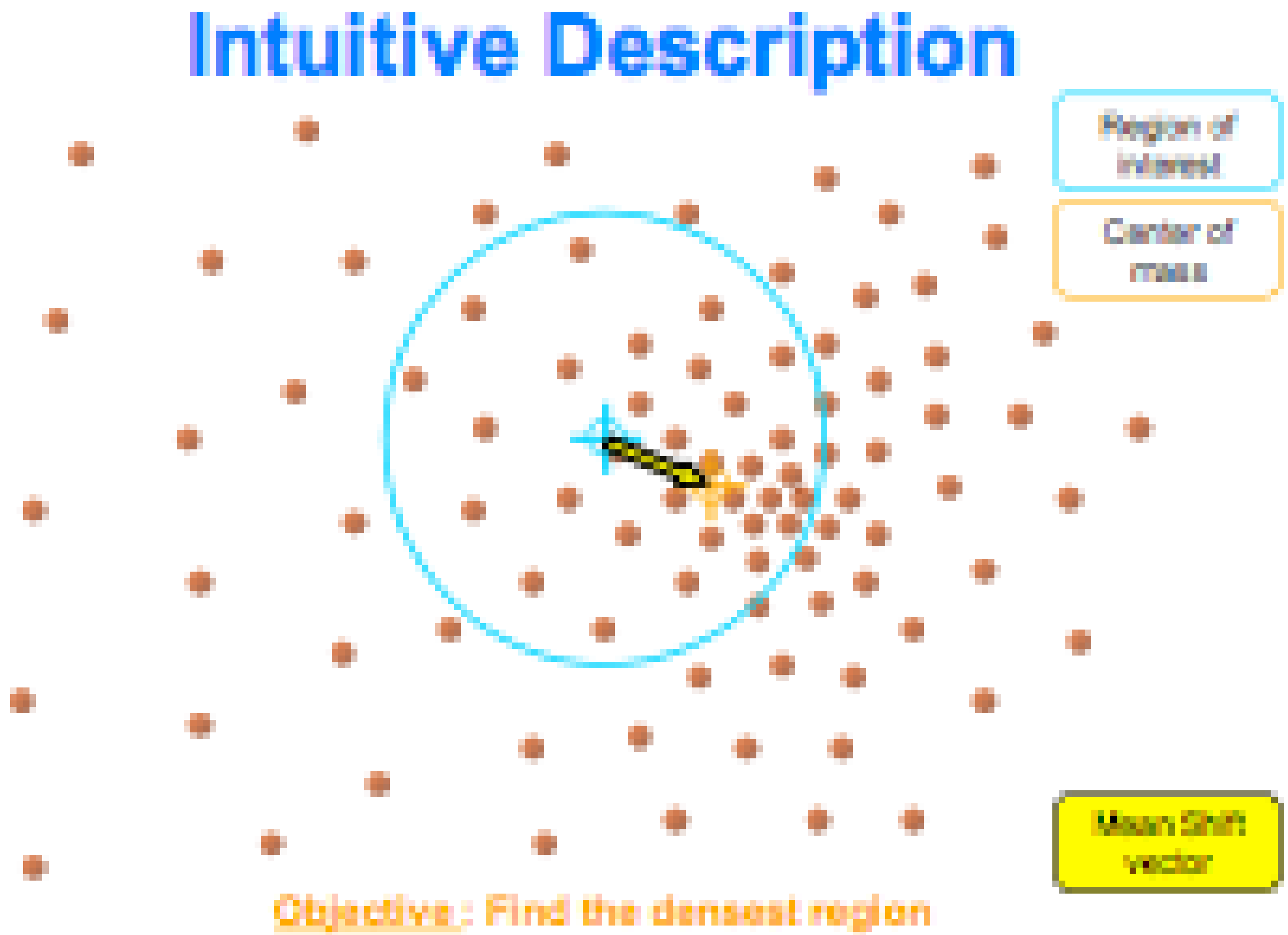
    - Here $g(y) = -k'(y)$
  - Move the current point by the mean shift vector:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{m}_{h,g}(\boldsymbol{x})$$

  - Repeat until convergence.

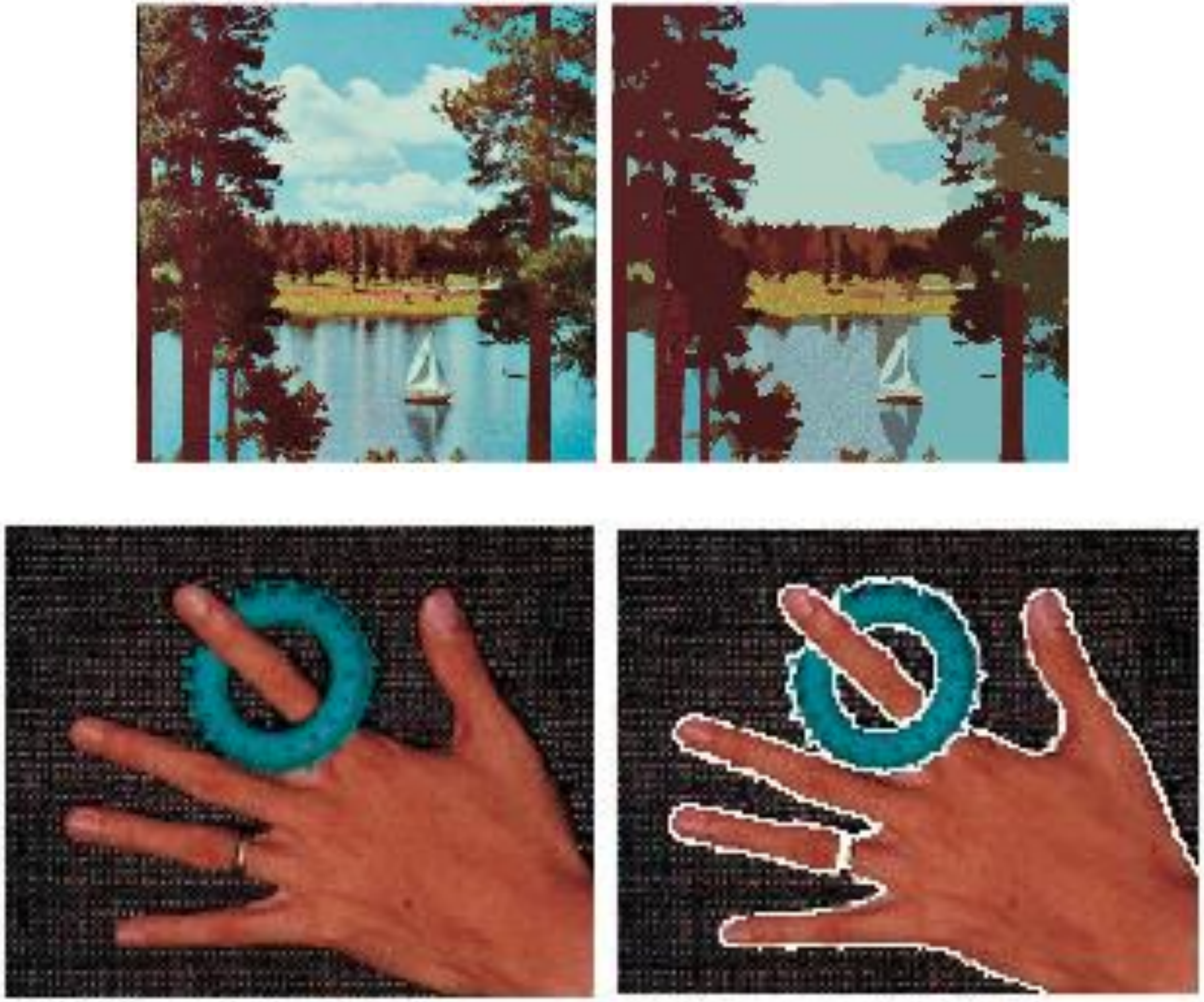
[Comaniciu & Meer, 02]

Illustration



From Ukrainitz & Sarel

Mean Shift Segmentations



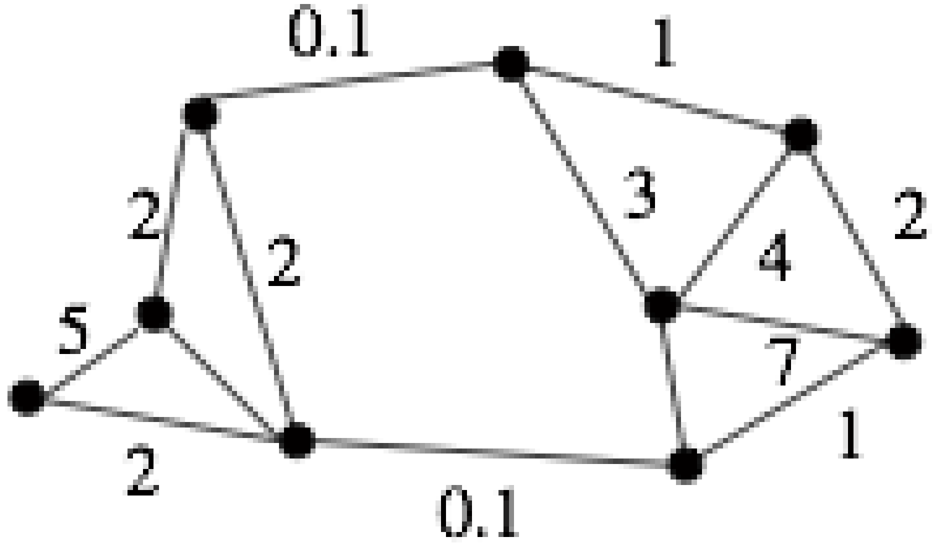
[Comaniciu & Meer, 02]

One more thing...

- ... about mean shift.
- You might have noticed that we did not have to select the number of segments.
- So do we no longer have to choose that number by hand?
  - Yes and no. We don't have to choose it directly, but the number of segments varies depending on the kernel width.
  - So we have just shifted the problem to a different place.

Graph-based Clustering

- Clustering can be interpreted as cutting a graph in which each node represents a pixel into pieces.
  - (Note: This graph is not a graphical model!)
- For this we define affinities between the pixels that encode how similar they are.
- These give the edge weights:



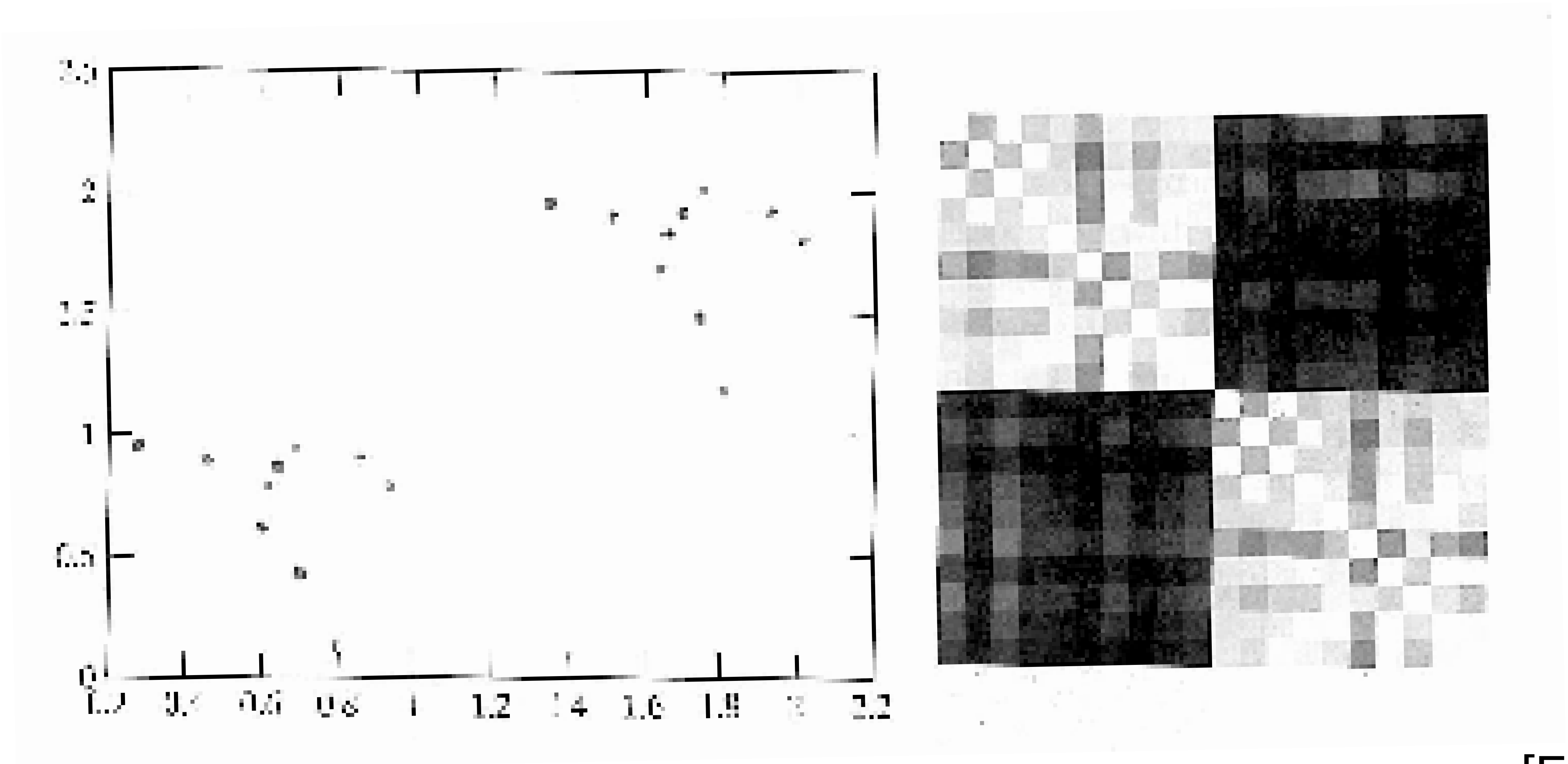
Note: Spatial arrangement is arbitrary!

Simple Affinity Criteria

- Define affinities of pixels:
  - Affinity by distance
    $$\operatorname{aff}(\boldsymbol{x},\boldsymbol{y}) = \exp\left\{-\|\boldsymbol{x}-\boldsymbol{y}\|^2/(2\sigma_D^2)\right\}$$
  - Affinity by intensity
    $$\operatorname{aff}(\boldsymbol{x},\boldsymbol{y}) = \exp\left\{-(I(\boldsymbol{x})-I(\boldsymbol{y}))^2/(2\sigma_D^2)\right\}$$
  - Affinity by color
    $$\operatorname{aff}(\boldsymbol{x},\boldsymbol{y}) = \exp\left\{-\operatorname{dist}(c(\boldsymbol{x}),c(\boldsymbol{y}))^2/(2\sigma_D^2)\right\}$$
  - Affinity by texture
    $$\operatorname{aff}(\boldsymbol{x},\boldsymbol{y}) = \exp\left\{-\|\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{f}(\boldsymbol{y})\|^2/(2\sigma_D^2)\right\}$$
    texture descriptor
  - ...

Affinity Matrix

- From this we can build an affinity matrix with all pairwise affinities:



[FP]

Graph-based Clustering

- Affinity matrix: $\mathbf{A}$, where A_{ij} is the affinity (weight) between pixels i and j
- Assignment to clusters: $\boldsymbol{w}_n$, where w_{ni} denotes that pixel i is assigned to cluster n with a certain weight ("certainty").
- What is a good cluster? Define a simple objective function:

$$\boldsymbol{w}_n^{\mathrm{T}} \mathbf{A} \boldsymbol{w}_n$$

  - The objective is large, when the cluster n contains pixels with high affinity to each other.
  - We need to ensure that weights cannot grow unboundedly, e.g.:

$$\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{w}_n = 1$$

Graph-based Clustering

- Constrained optimization problem:

$$\max \boldsymbol{w}_n^{\mathrm{T}} \mathbf{A} \boldsymbol{w}_n \quad \text{s.t.} \quad \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{w}_n = 1$$

- Methods of Lagrange multipliers:

$$\boldsymbol{w}_n^{\mathrm{T}} \mathbf{A} \boldsymbol{w}_n + \lambda (\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{w}_n - 1)$$

- Differentiate and set to zero:

$$\mathbf{A} \boldsymbol{w}_n = \lambda \boldsymbol{w}_n$$

- This is an eigenvalue problem!
  - Hence also called spectral clustering.
  - We know how to compute eigenvectors...

Graph-based Clustering

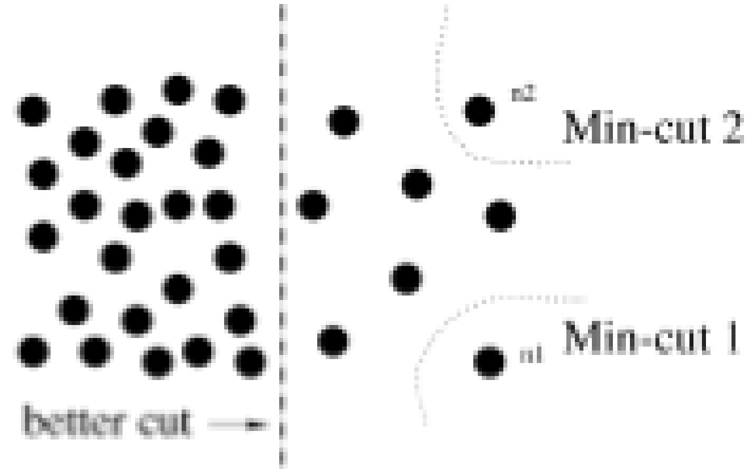
- We still need to extract segments from the eigenvector.
- This gets a bit hacky:

```
Construct an affinity matrix

Compute the eigenvalues and eigenvectors of the affinity matrix

Until there are sufficient clusters

    Take the eigenvector corresponding to the
    largest unprocessed eigenvalue; zero all components corresponding
    to elements that have already been clustered, and threshold the
    remaining components to determine which element
    belongs to this cluster, choosing a threshold by
    clustering the components, or
    using a threshold fixed in advance.

    If all elements have been accounted for, there are
    sufficient clusters

end
```

[FP]

Graph-Cut Based Segmentation

- This approach has its problems, because it is not always the case that there is a single eigenvector with a large value for each pixel.
- Another popular graph-based approach to segmentation is to find the min-cut on the graph:
  - The problem with this is that it favors small segments:



[Shi & Malik, 00]

Normalized Cuts

- A better approach is to normalize the cut to remove this bias:

$$\frac{\mathrm{cut}(A,B)}{\mathrm{assoc}(A)} + \frac{\mathrm{cut}(A,B)}{\mathrm{assoc}(B)}$$

  - Here $\mathrm{cut}(A,B)$ are the weights that are cut by separating the segments A and B.
  - $\mathrm{assoc}(A)$ is the weight of all edges going into segment A
- Unfortunately, optimizing this objective is NP hard:
  - But there is an efficient approximation as a generalized eigenvalue problem [Shi & Malik, 00].

Some Results

[Shi & Malik, 00]

Summary of Methods

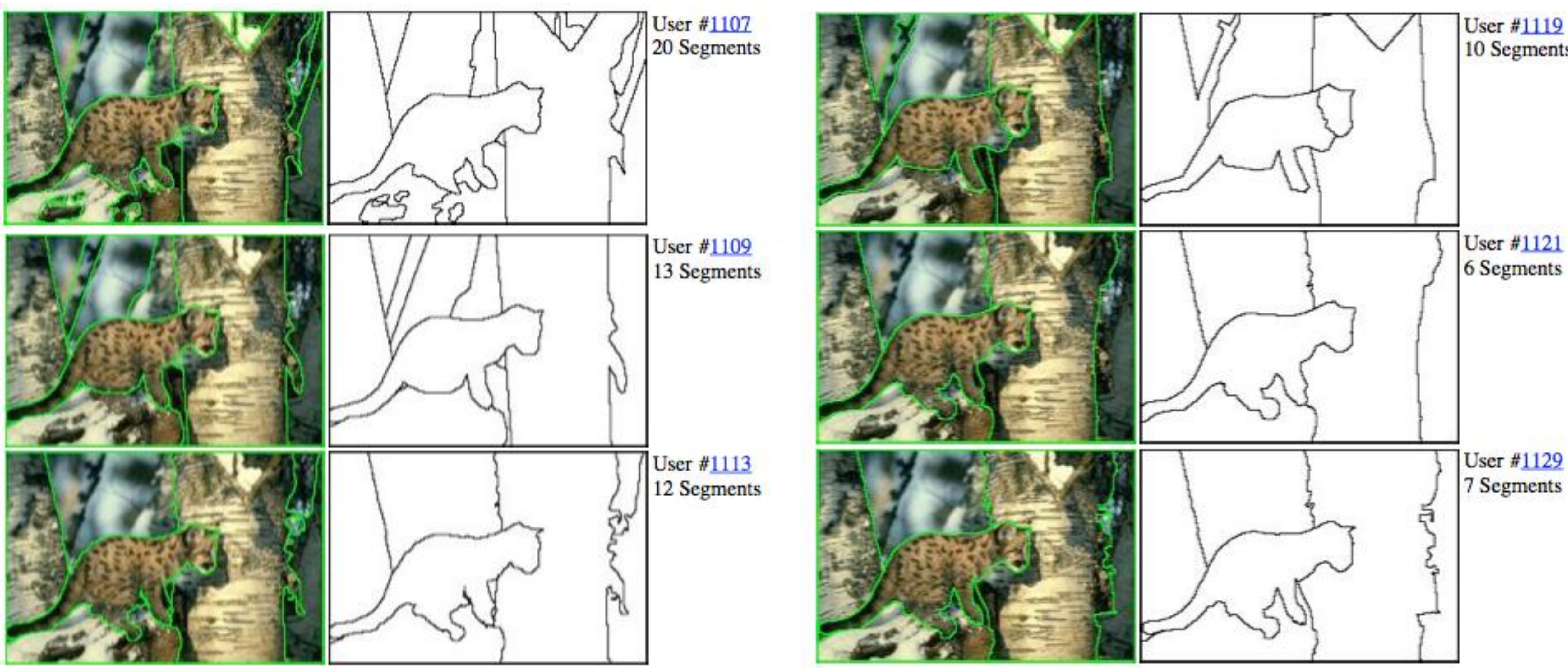
- We have seen a whole set of different segmentation techniques:
  - Agglomerative and divisive clustering
  - K-Means
  - Mean Shift
  - Graph-based or spectral clustering methods
    - Simple approach
    - Min-cut
    - Normalized cuts
  - CV 2: Energy-based methods & probabilistic methods
- So far, no “golden standard” has been established.

Is there a correct segmentation?

- Unfortunately not!
- A segmentation can only be right for a certain purpose...
  - Say, if we care about finding a person in an image, we may want the segmentation to separate people from the background.
  - But what if we wanted to know what cash register the person goes to? Then we want to also segment the image into the various cash registers.
  - Segmentation can mean a lot of different things!

Is there a correct segmentation?

- If you ask different people to segment an image, you will get many different results:



  - This makes it also hard to evaluate how good a segmentation is.

What can we hope for?

- We cannot hope that segmentation does all that we might want it to do.
  - Segmentation is normally “dumb” in the sense that it doesn’t know what we want to do with the result.
  - Chicken-and-egg problem: A good segmentation helps a lot with various vision tasks (e.g. object recognition), but unless we have solved this task already, we can’t hope to get the perfect segmentation.
  - Segmentation should somehow be coupled to the task we want to solve with it. How?
- Of course, all of this doesn’t mean that we should abandon it.