IMAGE PROCESSING & ANALYSIS

References

- E.Trucco, A. Verri: Introductory Techniques for 3-D Computer Vision, 1998
- R. Szeliski, Computer Vision: Algorithms and Applications, http://szeliski.org/Book/, 2010
- G. Bradski and A. Kaehler, Learning OpenCV, 2008
- A. Keil and N. Navab, Image Processing and Segmentation slides, 2006
- http://homepages.inf.ed.ac.uk/rbf/HIPR2
- and others ...

IMAGE PROCESSING & ANALYSIS

Image segmentation

Basic concepts

Goals and Difficulties

- The goal of segmentation is:
  - to find groups of pixels that “go together” (Szeliski)
  - to partition an image into a set of disjoint regions; the union of all the regions must correspond to the whole image.
- The result of segmentation - the segments – must have some meaning for the concrete application:
  - Object Recognition: separate the objects from the background
  - Visual Inspection: detect defects
  - Robotic Vision: locate objects / road boundaries
  - Road Traffic Control: detect moving objects
  - Medical Imaging: separate pathological/non-pathological regions.
- The results of segmentation are very important in determining the eventual success or failure of image analysis
- Segmentation is **very a very difficult problem**, in general.

The goals of segmentation

- Separate image into coherent “objects”
  - “Bottom-up” or “top-down” process?
  - Supervised or unsupervised?

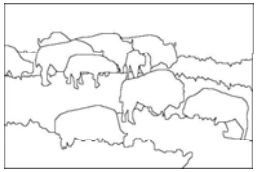
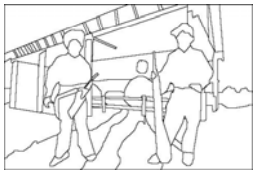



image | human segmentation

Source: S. Lazebnik

Berkeley segmentation database: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench

The goals of segmentation

- A **region** is
  - a group of connected pixels with similar properties.
- The **goal** of segmentation is
  - to divide the image into regions, each of which is homogeneous in some sense, but the union of no two adjacent regions is homogeneous in the same sense.

Increasing accuracy and robustness

- Introduce enough knowledge about the application domain
- Control the environment
  - e.g.: in industrial applications
- Select type of sensors to enhance the objects of interest
  - e.g.: use infrared imaging for target recognition applications

Segmentation approaches

- **Edge-based** approaches:
  - Use the boundaries of regions to segment the image
  - Detect abrupt changes in intensity (discontinuities)
  - Gaps in edges have to be bridged
- **Region-based** approaches:
  - Use similarity and spatial proximity among pixels to find different regions


- Edges are found based on DIFFERENCES between values of adjacent pixels.
- Regions are found based on SIMILARITIES between values of adjacent pixels.
- Theoretically, both approaches should give identical results but this is not true in practice

Classifications (1)

Classification often not unique

- Pixel-based
  - Only based on pixel intensities
  - Only applicable in simple cases or as a preprocessing step
- Region-based
  - Allows definition of region properties
  - Takes connectivity properties into account
  - Results in connected components
- Edge-based
  - Searching for surrounding edges (often based on gradient values)
  - Gaps in edges have to be bridged
- Model-based
  - Given model is deformed to fit region
  - Explicit or implicit representation of the model
- Atlas-based
  - Pre-segmented data is mapped onto actual image by registration

Classification (2)

- Basic algorithms
  - Thresholding
  - Region growing
  - Clustering
- Advanced algorithms
  - Morphological
    - Split and Merge
    - Watersheds
  - Deformable models
    - Snakes
    - Level Sets
  - Model-based
    - Hough Transform
    - Active Shape Models
    - Atlas-Based
  - Graph-based methods
    - Live Wire
    - Graph Cuts

Classifications (3)

- Image domain
  - Boundary-based
    - edge-based
    - deformable models
  - Region-based
    - region splitting
    - region merging
    - split-and-merge
    - watersheds
- Feature domain
    - thresholding
    - clustering
    - mean-shift
    - graph-based

Segmentation in the feature domain

- The idea is to measure several **features**, organize them into a **feature vector** and use **clustering** methods to segment the data.
- Depending on the problem many features can be used, such as:
  - Intensity
  - Colour
  - Texture
  - Position
  - Geometrical features (size, orientation, shape …)
  - ...
- Each feature component may characterize a single pixel (as the intensity) or a region (as the area).
- The choice of features and how they are quantified implies a **feature space** in which each token is represented by a point.
- Token similarity is thus measured by distance between points (**feature vectors**) in feature space.

Classification (Szeliski)

- **Active contours**
  - Snakes
  - Dynamic snakes and CONDENSATION
  - Scissors
  - Level sets
- **Split and merge**
  - Watershed
  - Region splitting (divisive clustering)
  - Region merging (agglomerative clustering)
  - Graph-based segmentation
  - Probabilistic aggregation
- **Mean shift and mode finding**
  - K-means and mixtures of Gaussians
  - Mean shift
- **Normalized cuts**
- **Graph cuts** and **energy-based methods**

IMAGE PROCESSING & ANALYSIS

Thresholding

Region growing/splitting

Split-and-merge

Basic segmentation algorithms

- Basic segmentation algorithms presented on the next slides:
  - Thresholding
  - Region growing
- Useful for
  - easy segmentation tasks
    - e.g.: bone in CT
  - initializing or autoseeding more sophisticated algorithms

Thresholding

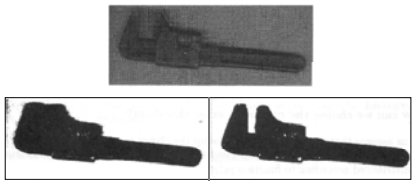
- Regions with uniform intensity give rise to strong peaks in the histogram !
- Select one or more intensity values (**thresholds**)
- Partition the image domain according to the intensity being higher or lower than those thresholds

$$R_i = \{x \in D_I : t_i \le I(x) < t_{i+1}\}$$

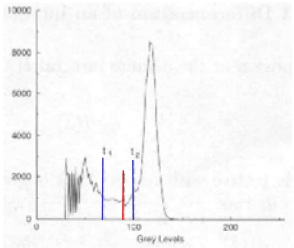
- **Selection of thresholds**
  - manually
  - automatically
    - at local minima (requires smoothing of histogram first)
    - at intersection of fitted Gaussian functions
    - from prior knowledge
      - intensity characteristics of the objects
      - sizes of the objects
      - fractions of an image occupied by the objects
      - number of different types of objects appearing in an image
- In general, good thresholds can be selected if the histogram peaks are
  - tall,
  - narrow,
  - symmetric,
  - and separated by deep valleys.

Hysteresis thresholding

- If there is no clear valley in the histogram of an image, it means that there are several background pixels that have similar gray level value to object pixels and vice-versa.
- Two thresholds, one at each side of the valley can be used in this case.
- Pixels above the high threshold are classified as **object** and below the low threshold as **background**.
- Pixels between the low and high thresholds are classified as object only if they are adjacent to other object pixels.

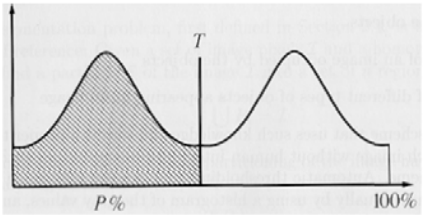


Single thresholding Hysteresis thresholding



P-Tile (percentile) method

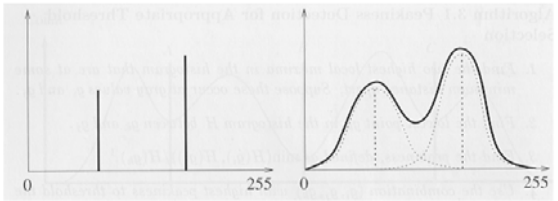
- Requires previous knowledge about the area or size of the objects present in the image
- Let us assume that we have dark objects against a light background.
- If, for example, the objects occupy p% of the image area, an appropriate threshold can be chosen by partitioning the histogram



Optimal thresholding

- Suppose that an image contains only two principal regions (e.g., object and background)
- We can minimize the number of misclassified pixels if we have some prior knowledge about the distributions of the gray level values that make up the object and the background.
- Assume that the distribution of gray-level values in each region follows a Gaussian distribution

- Drawbacks:
  - Prior probabilities might not be known.
  - Object/Background distributions might not be known.



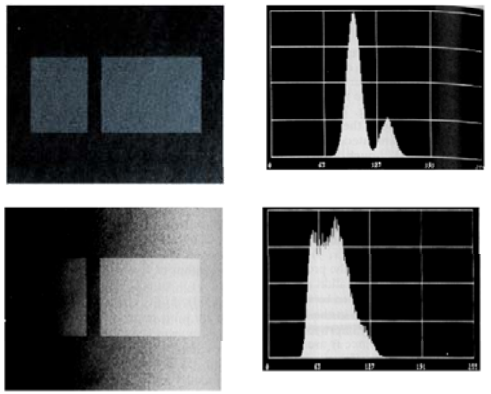
Thresholding

Otsu method

- A measure of region homogeneity is variance (i.e., regions with high homogeneity will have low variance).
- Otsu's method selects the threshold, T, such that
  - the intra-class variance is minimized and
  - the between-class variance is maximized .
- Iterative procedure
  - => calculate between-class variance for every possible T
- It does not depend on modeling the probability density functions, however, it assumes a bimodal distribution of gray-level values (i.e., if the image approximately fits this constraint, it will do a good job).
- Drawbacks:
  - Assumes that the histogram of the image is bimodal (i.e., two classes).
  - Breaks down when the two classes are very unequal (i.e., the classes have very different sizes).

Effect of illumination on segmentation

- A single threshold will not work well when we have **uneven illumination** due to shadows or due to the direction of illumination.

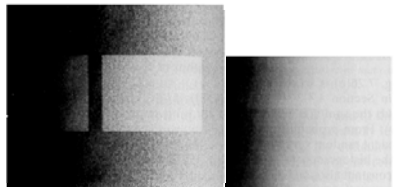


non-uniform illumination

Effect of illumination on segmentation

Handling nonuniform illumination: laboratory solution

- Obtain an image of just the illumination field.
  - Suppose that $I_1(x, y) = r(x, y)\, i(x, y)$, where i(x, y) is non-uniform
  - Project the illumination pattern on a surface with uniform reflectance (e.g., a white surface)
    - $I_2(x, y) = k\, i(x, y)$
  - Normalize $I_1(x, y)$:
    - $I_{1N}(x, y) = I_1(x, y) / I_2(x, y)$
- Or, better
  - make illumination uniform !



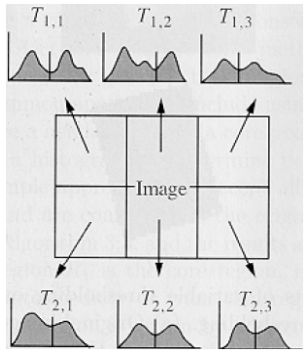
I_1

I_2

Effect of illumination on segmentation

Handling nonuniform illumination: local thresholding

- The idea is to
  - partition the image into m x m subimages and
  - then choose a threshold $T_{i,j}$ for each subimage.
- This approach might lead to subimages having simpler histogram (e.g., bimodal)
- Drawback:
  - object contours not continuous in the transition between regions



http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

Thresholding

Drawbacks of thresholding:

- Pixels assigned to a single class need not form coherent regions as the spatial locations of pixels are completely ignored
  - only hysteresis thresholding considers some form of spatial proximity.
- Threshold selection is not always straightforward.

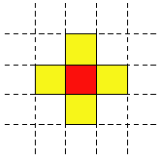
Region growing

- Region-growing approaches exploit the important fact that pixels which are close together have similar gray values.
- General algorithm:
  - Start with a single pixel (seed) and add new pixels iteratively
  1. Choose the seed pixel
  2. Check the neighboring pixels and add them to the region if they are similar to the seed
  3. Repeat step 2 for each of the newly added pixels; stop if no more pixels can be added.
- Crucial definitions:
  - Neighborhood
  - Homogeneity (aggregation criterion)
  - Stopping criterion

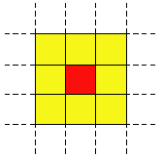
Region growing

Neighborhood

- The region around the center pixel
- 2D: 4 pixel or 8 pixel neighborhood
- 3D: 6, 18, or 26 voxel neighborhood



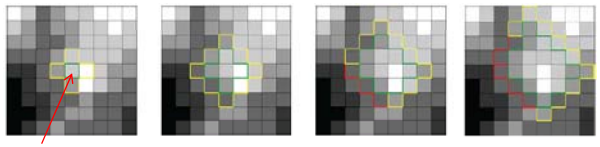
- Behavior of algorithms depends on the type of neighborhood chosen



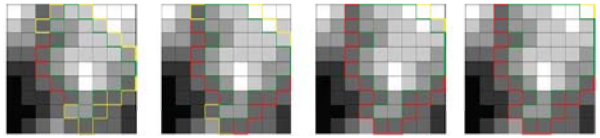
Region growing

- How to choose the seed(s) in practice ?
  - It depends on the nature of the problem.
    - e.g.: if targets need to be detected using infrared images, choose the brightest pixel(s).
  - Without a-priori knowledge, compute the histogram and choose the gray-level values corresponding to the strongest peaks
- How to choose the similarity criteria (predicates)?
  - The homogeneity predicate can be based on any characteristic of of the regions in the image such as
    - average intensity
    - variance
    - color
    - texture
    - motion
    - shape
    - ...

Region growing

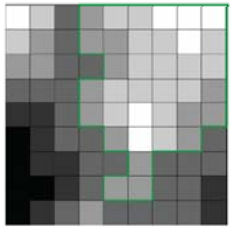


seed point



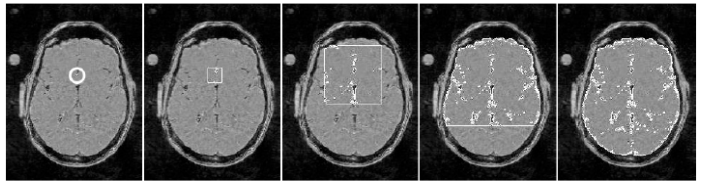
Similarity criterion: gray value difference to seed point <= 1,
neighborhood = 4

Segmentation result



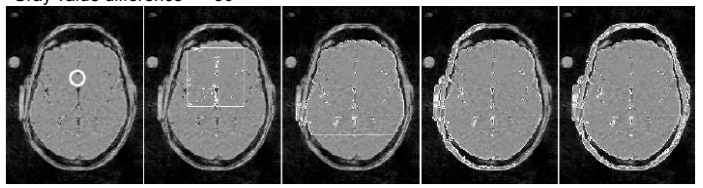
Region growing

Gray value difference <= 20



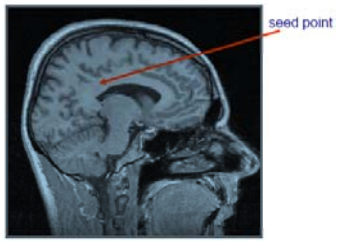
seed point | 10 iterations | 40 iterations | 70 iterations | Result: 117 iterations

Gray value difference <= 50



seed point | 40 iterations | 80 iterations | 200 iterations | Result: 320 iterations

Region growing vs. Thresholding



Thresholding

Region growing
Gray value difference = 40

Region Merging / Splitting

- **Region Merging:**
  - Image is separated into very small regions (like pixels, or 2x2 squares)
  - Neighboring regions are merged if the merged region remains homogeneous or if the regions are similar
  - What does "similar" mean?
    - "similar" average values : $|\mu_i - \mu_j| < T_\mu$
    - "small" spread of gray values: $|I_{max} - I_{min}| < T_I$
    - surface fitting
      - Model regions as polynomial surfaces
      - Compute the fitting error
      - Merge regions if the fitting error is low enough
    - other ...?

Region Merging / Splitting

- **Region Splitting:**
  - Whole image is one region
  - Homogeneity is computed for each region
  - Regions are successively split, until homogeneity is fulfilled for every region
  - Problem: over-segmentation
- Deciding when to split is fairly straight-forward
  - A property is not "constant"
  - A predicate is not TRUE
- Where to split is the difficulty
  - Some approaches:
    - Divide it into equal parts along image dimensions.
    - Look for strong edges to create boundaries.

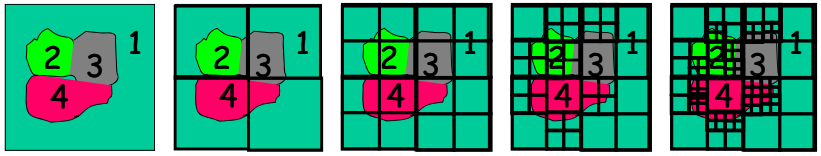
Split-and-Merge

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-and-Merge

Split …



…and merge

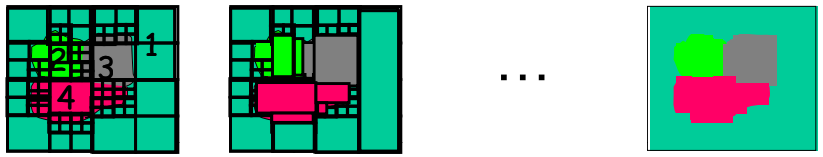
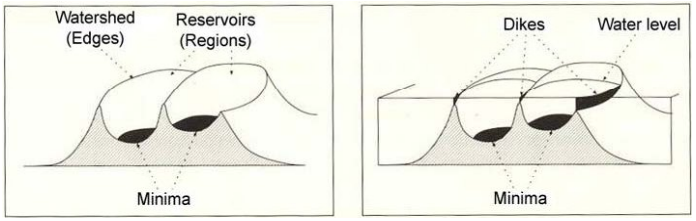


IMAGE PROCESSING & ANALYSIS

Watersheds

Watersheds: idea

- Interpretation of image as height map
- Height map consists of basins and ridges
- Often applied to gradient magnitude images
- Idea: dropping water onto gray value height map
- Flooding the “landscape” would fuse regions which can be prevented by building dikes
- Result: one region for every local minimum



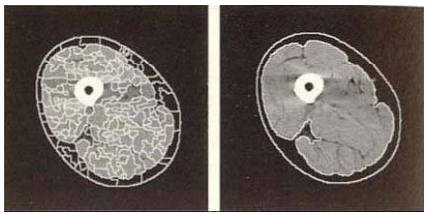
http://cmm.ensmp.fr/~beucher/wtshed.html

Watersheds: algorithms

- Rainfall algorithm (top-down, gradient descent)
  - *Two points are in the same region if they drain to the same point.*
  - Drop a raindrop on every unlabeled pixel
  - Let the raindrop flow to a local minimum using gradient descent
  - Assign label of local minimum found hereby to the starting pixel
- Flooding (bottom-up, morphological)
  - *Flooding the “landscape” would fuse regions which can be prevented by building dikes*
  - Start from smallest intensity value and go up level by level, flooding the landscape
  - Build dike when separate basins would merge during flooding
  - Detect merging regions by connected component analysis and dilation operator and erect dike by setting highest gray value plus one
- Flooding algorithm is level-based (e.g. integer values), faster than rainfall

Watersheds: properties

- Boundaries are always closed around regions
- One region for every local minimum
- Raw watershed segmentation may produce a severely oversegmented image with hundreds or thousands of catchment basins.
  - Pre-processing: smoothing to prevent over-segmentation
  - Post-Processing: region merging; edge information; ...
- Other possibilities to prevent over segmentation:
  - Minimum basin depth
  - Minimum basin size



OpenCV

- **void watershed(InputArray image, InputOutputArray markers)**
- Performs a marker-based image segmentation using the watershed algrorithm.
- Parameters:
- image – Input 8-bit 3-channel image.
- **markers** – Input/output 32-bit single-channel image (map) of markers. It should have the same size as image.
- The function implements one of the variants of watershed, non-parametric marker-based segmentation algorithm, described in [Meyer92].
- Before passing the image to the function, you have to roughly outline the desired regions in the image **markers** with positive (>0) indices. So, every region is represented as one or more connected components with the pixel values 1, 2, 3, and so on.
- Such markers can be retrieved from a binary mask using findContours() and drawContours() *(see the watershed.cpp demo)*.
- The markers are “seeds” of the future image regions. All the other pixels in markers, whose relation to the outlined regions is not known and should be defined by the algorithm, should be set to 0’s.
- In the function output, each pixel in markers is set to a value of the “seed” components or to -1 at boundaries between the regions.

IMAGE PROCESSING & ANALYSIS

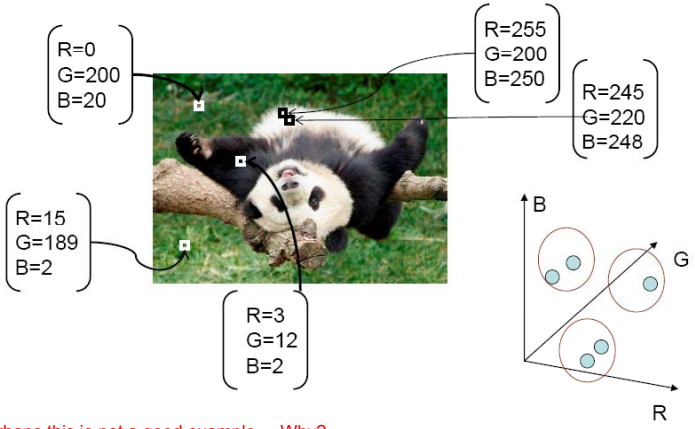
Clustering

Clustering

- Clustering:
  - unsupervised classification methodology for grouping the **feature space** into a set of meaningful groups or classes.
    - *supervised classification:*
      - *use training data to infer model (the classes)*
      - *apply model to test data*
    - *unsupervised classification*
      - *no training data*
      - *model inference and application both rely on the test data exclusively*
- Two main approaches:
  - Divisive clustering (or clustering by splitting)
    - the entire data set is regarded as cluster and the clusters are recursively split to yield a good clustering
  - Agglomerative clustering (or clustering by merging)
    - each data item is regarded as a cluster and clusters are recursively merged until a good set of clusters is obtained.

Clustering

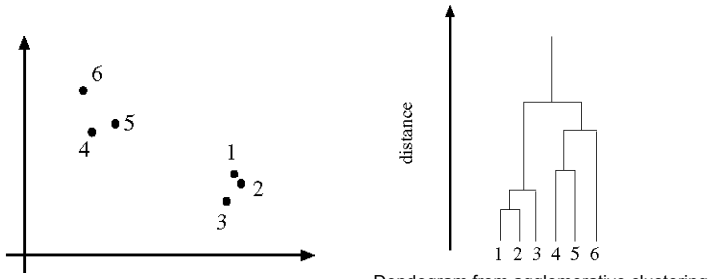
- Cluster similar pixels (features) together



Perhaps this is not a good example ... Why?

Clustering

- Dendograms
  - yield a picture of output as clustering process continues
  - instead of a fixed number of clusters, the dendrogram represents a hierarchy of clusters



Dendogram from agglomerative clustering

Divisive clustering

- Splitting the image into successively finer regions is one of the oldest techniques in computer vision.
- Ohlander, Price, and Reddy (1978) present such a technique
  - first compute a histogram for the whole image
  - then find a threshold that best separates the large peaks in the histogram.
  - repeat until regions are either fairly uniform or below a certain size.
- More recent splitting algorithms often optimize some metric of intra-region similarity and inter-region dissimilarity.
  - ex: graph cuts, normalized cuts

Agglomerative clustering

Agglomerative clustering

- Algorithms can link clusters together based on
  - the distance between
    - their closest points (single-link clustering),
    - their farthest points (complete-link clustering),
    - or something in between
- A very simple version of pixel-based merging combines adjacent regions whose average color difference is below a threshold
  or
  whose regions are too small.
- Segmenting the image into such *superpixels* which are not semantically meaningful, can be a useful pre-processing stage
  - to make higher-level algorithms such as stereo matching, optic flow, and recognition both faster and more robust.

Mode finding

K-means (Szeliski)

- implicitly models the probability density as a superposition of spherically symmetric distributions,
  - it does not require any probabilistic reasoning or modeling (Bishop 2006)
  - the algorithm is given the number of clusters k it is supposed to find;
  - it then iteratively updates the cluster center location based on the samples that are closest to each center.
  - The algorithm can be initialized by randomly sampling k centers from the input feature vectors. Techniques have also been developed for splitting or merging cluster centers
- based on their statistics, and for accelerating the process of finding the nearest mean center (Bishop 2006).

Mixtures of Gaussians

- each cluster center is augmented by a covariance matrix whose values are re-estimated from the corresponding samples.
- Instead of using nearest neighbors to associate input samples with cluster centers, a Mahalanobis distance

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k) = \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k^{-1}} = (\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_k)$$

where x_i are the input samples, μ_k are the cluster centers, and Σ_k are their covariance estimates. Samples can be associated with the nearest cluster center (a *hard assignment* of membership) or can be *softly assigned* to several nearby clusters.

K-means clustering

K-means algorithm

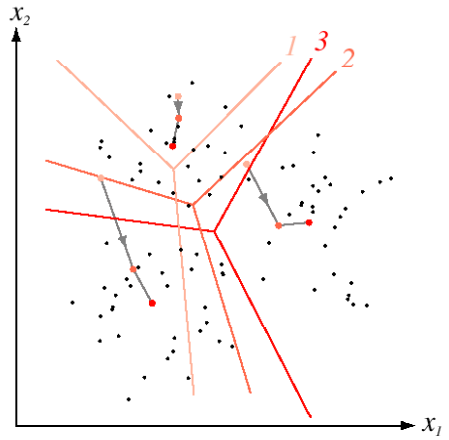
- Initialization: given
  - N points in feature space
  - K categories.
- Pick K points randomly
  - these are initial cluster centers (means): $\mu_1, \ldots, \mu_K$.
- Repeat :
  - assign each of the N points, x_j, to clusters by nearest μ_i
    - make sure that every cluster has at least one data point;
      - possible techniques for doing this include supplying empty clusters with a point chosen at random from points far from their cluster center.
  - recompute mean μ_i of each cluster from its member points

  Until no mean has changed.
- Effectively carries out gradient descent to minimize:

$$\sum_{i \in \text{clusters}} \left\{ \sum_{j \in \text{elements of i'th cluster}} \|x_j - \mu_i\|^2 \right\}$$

How to find K ?
- Use prior knowledge about image.
- Use different K's and test for goodness of clusters.
- Analyze image histograms.

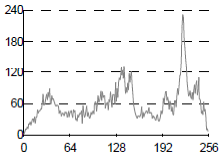
Example: 3-means clustering

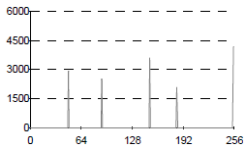


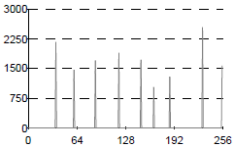

original image


K=5 clusters


K=9 clusters







- K-means clustering based on intensity or color
  - Clusters don't have to be spatially coherent

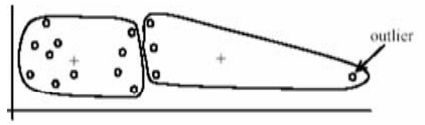

Image


Intensity-based clusters

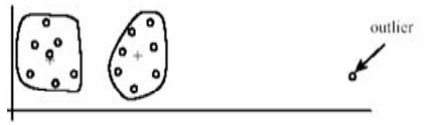

Color-based clusters

- Clustering based on (color_components & X,Y) values enforces more spatial coherence

- Pros
  - Very simple method
  - Converges to a local minimum of the error function
- Cons
  - Need to pick K
  - Sensitive to initialization
  - Sensitive to outliers
  - Only finds "spherical" clusters



(A): Undesirable clusters



(B): Ideal clusters

Mixture of Gaussians

- Each cluster center is augmented by a covariance matrix *(see next slides)* whose values are re-estimated from the corresponding samples.
- Instead of using nearest neighbors to associate input samples with cluster centers,
  a Mahalanobis distance *(see next slides)* is used:

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k) = \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k^{-1}} = (\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_k)$$

  where
  - x_i are the input samples,
  - μ_k are the cluster centers, and
  - Σ_k are their covariance estimates.
- Samples can be associated with
  the nearest cluster center (a *hard assignment* of membership)
  or can be *softly assigned* to several nearby clusters.

- Tutorial on clustering algorithms: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/index.html

(Covariance matrix)

- Covariance is the generalization of the variance
- Let x and y be random variables
  with respective expectations $E[x]$ and $E[y]$

$$\mathrm{E}[x] = \sum_{i \in \text{values}} x_i p(x_i)$$

$$\begin{aligned} Var(x) &= E[(x_i - E[x])^2] \\ &= E[x^2] - E[x]^2 \\ &= E[xx] - E[x]E[x] \end{aligned}$$

$$\begin{aligned} Cov(x, y) &= E[(x - E[x])(y - E[y])] \\ &= E[xy] - E[x]E[y] \end{aligned}$$

- Let X=(X_1, X_2, ... , X_n) be a multivariate random variable.
- Then the Covariance Matrix is defined as

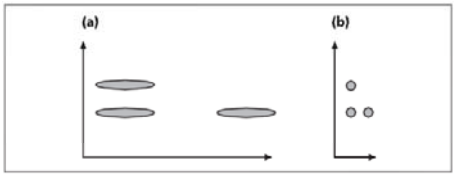
$$COV(X) = \begin{bmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) & \dots & Cov(X_1, X_n) \\ Cov(X_2, X_1) & Cov(X_2, X_2) & \dots & Cov(X_2, X_n) \\ \vdots & \vdots & & \vdots \\ Cov(X_n, X_1) & Cov(X_n, X_2) & \dots & Cov(X_n, X_n) \end{bmatrix}$$

This matrix is symmetric.
Diagonal entries are the variance of components of x, and must be non-negative.
Off-diagonal elements measure the extent to which two variables co-vary.
For independent variables, the covariance must be zero.
For two random variables that generally have different signs, the covariance can be negative.

(Mahalanobis distance)

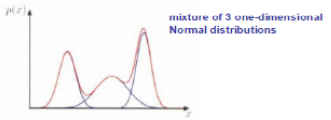
Mahalanobis distance:

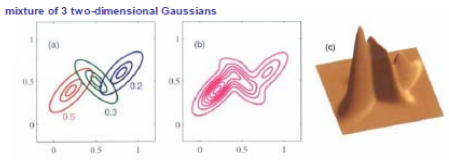
- A distance measure that takes into account the variance of the data.
- In particular,
  distance from the mean along a direction where there is little variance has a large weight and
  distance from the mean along a direction where there is a large variance has little weight.
- If the covariance is the identity matrix (identical variance),
  then this measure is identical to the Euclidean distance measure.

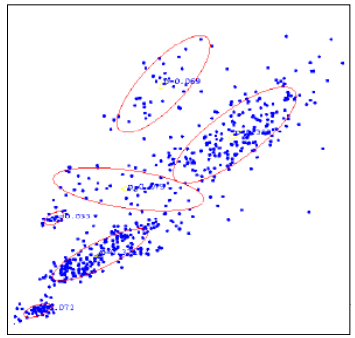


The Mahalanobis computation allows us to reinterpret the data's covariance as a "stretch" of the space:
(a) the vertical distance between raw data sets is less than the horizontal distance;
(b) after the space is normalized for variance,
the horizontal distance between data sets is less than the vertical distance

Mixture of Gaussians







A Tutorial on Clustering Algorithms
http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Active contours

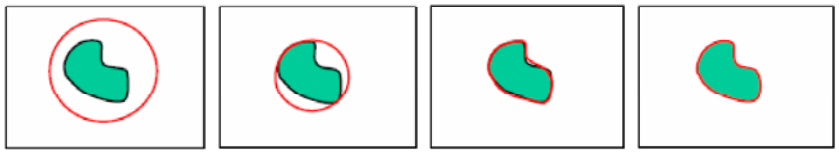
Active contours:

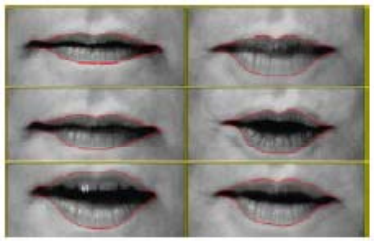
- Snakes
  - An energy-minimizing, two-dimensional spline curve that evolves (moves) towards image features such as strong edges - *(Kass, Witkin, and Terzopoulos, 1988)*
- Level sets
  - Region boundaries are implicitly represented as the zero level set of a level set function
  - Overcomes some of the limitations of snakes.
- Active Shape Models
  - A *point distribution model* that describes the average contour of a class of objects and the allowable displacement of the points the contour.
  - The *point distribution model* is aligned with image object by searching for image edge points along the normal to the contour.
- Intelligent scissors
  - As the user draws a rough outline, the system computes and draws a better curve that clings to high-contrast edges, in real-time – *(Mortensen and Barrett,1995)*

Snakes

- Segmentation using a deformable model which represents the interface between background and object
- The snake is a contour represented parametrically as $c(s) = (x(s), y(s))$ where x(s) and y(s) are the coordinates along the contour and $s \in [0,1]$
- Snake minimizes energy composed of *(see Trucco book)*
  - internal energy:
    - only depends on snake itself, not on image intensities
    - usually curvature-dependant leading to some regularization
  - external energy:
    - image-dependant
    - usually gradient-dependant attracting snake to edges
  - Possibly some "balloon force" expanding or shrinking the model
- Stationary solution for the minimization problem gives desired segmentation
- User-interaction:
  - drawing initial boundary curve
- Comments:
  - This approach is simple and has low computational requirements.
  - It does not guarantee convergence to the global minimum of the energy functional.
  - Works very well as far as the initial snake is not too far from the desired solution.

Snakes



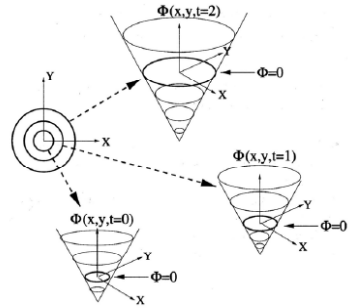


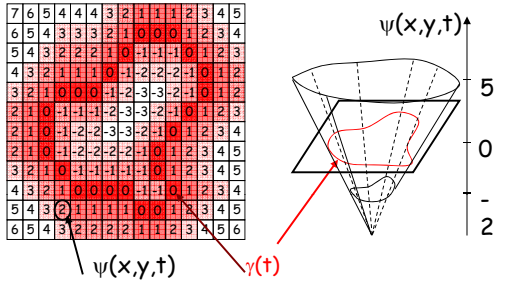
Source: Szeliski book

Lip tracking

Level sets

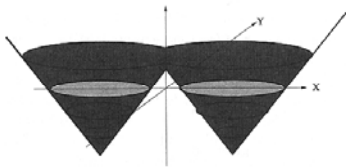
- Level sets
  - Drawbacks of snakes:
    - fixed topology (number of connected components and possible holes)
    - self-intersection possible
  - Solution:
    - Implicit representation of interfaces or boundaries (curves in 2D, surfaces in 3D)



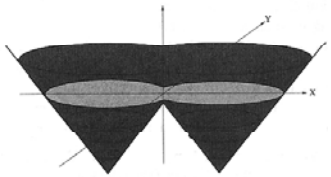


Level sets

- Region boundaries are implicitly represented as the zero level set of the level set function
- Evolution of the boundaries by deformation of the level set function
- The key idea is: let an initial curve evolve with a speed that depends on the curve and the image itself
  - local properties: curvature and normal of the curve
  - global properties: position and shape of the curve
  - image properties: gradients, intensity, texture
- Computation leads to partial differential equations



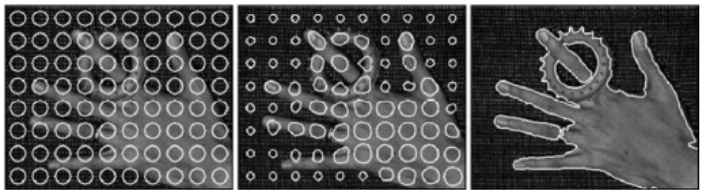
The level set surface ϕ (dark gray): Two separate initial fronts (in light gray).



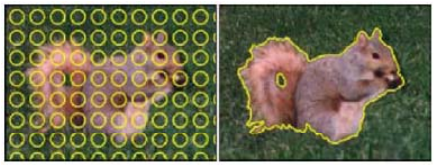
Later in time: the interface topology has changed, yielding a single curve as the zero level set.

Tutorial on level sets: http://step.polymtl.ca/~rv101/levelset/

Level sets



(a)



(b)

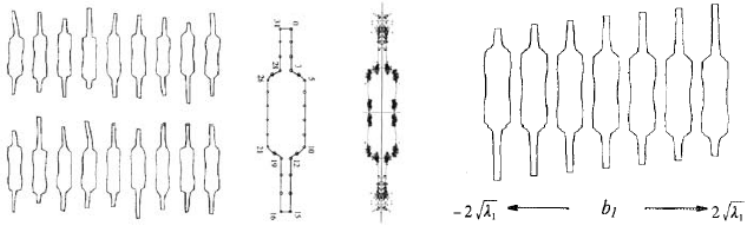
Source: Szeliski book

Level set segmentation (Cremers, Rousson, and Deriche 2007)
(a) grayscale and (b) color image segmentation.
The initial circles evolve towards an accurate segmentation of foreground and background, adapting their topology as they evolve.

Active Shape Models (ASM's)

- **Active shape models** (ASM's)
  - are statistical models of the shape of objects which iteratively deform to fit to an example of the object in a new image.
  - The shapes are constrained by the PDM (point distribution model) Statistical Shape Model to vary only in ways seen in a training set of labelled examples.
  - The shape of an object is represented by a set of points (controlled by the shape model).
  - The ASM algorithm aims to match the model to a new image. It works by alternating the following steps:
    - Look in the image around each point for a better position for that point
    - Update the model parameters to best match to these new found positions
  - To locate a better position for each point one can:
    - look for strong edges, or
    - a match to a statistical model of what is expected at the point.
- Application examples:
  - analyse images of faces,
  - medical image segmentation (in 2D and 3D; ex: lung segmentation).
- It is also known as a "Smart Snakes" method, since it is analog to an active contour model which would respect explicit shape constraints
- It is closely related to the **Active Appearance Model**.
- *T.F. Cootes and C.J. Taylor and D.H. Cooper and J. Graham (1995). "Active shape models – their training and application". Computer Vision and Image Understanding (61): 38–59.*

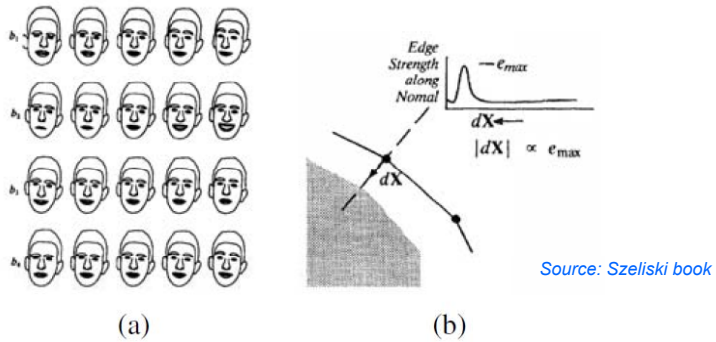
Active Shape Models (ASM's)



Point distribution model for a set of resistors (Cootes, Cooper, Taylor et al.1995)
(a) set of input resistor shapes;
(b) assignment of **control points** to the boundary;
(c) distribution (scatter plot) of point locations;
(d) first (largest) mode of variation in the ensemble shapes: λ_1.

Source: Szeliski book

Active Shape Models (ASM's)

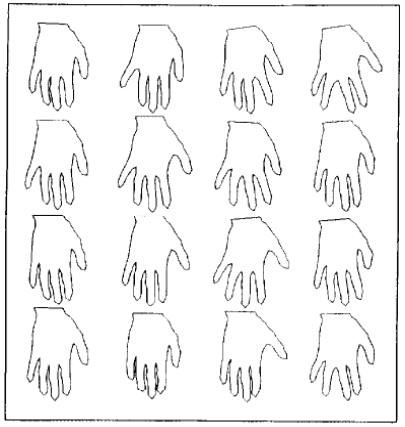


Source: Szeliski book

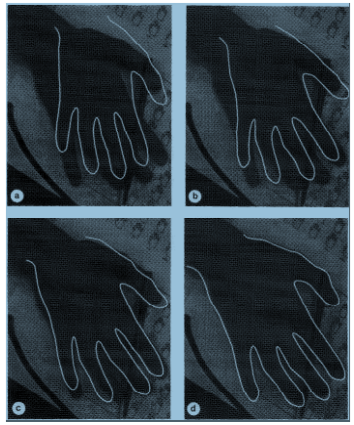
Active Shape Model - ASM (Cootes, Taylor, Lanitis et al. 1993):
(a) the effect of **varying the first four shape parameters** of a set of faces
(b) searching for the strongest gradient along the normal to each control point

Computer Vision

Active Shape Models (ASM's)



Training set of hand shapes each defined by 72 points



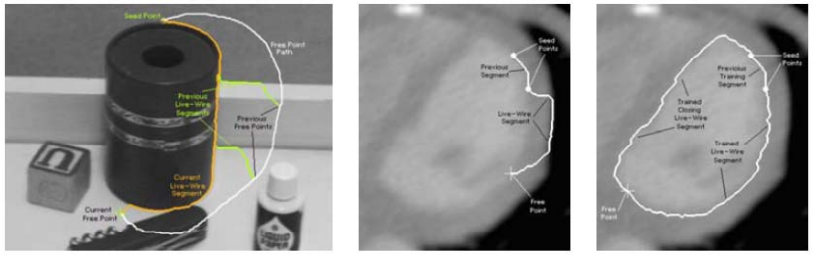
Hand with hand model superimposed; initial position and location after 100, 200 and 350 iterations

Source:
T.F. Cootes and C.J. Taylor and D.H. Cooper and J. Graham (1995)."Active shape models – their training and application". Computer Vision and Image Understanding (61): 38–59

Computer Vision

Intelligent scissors / Livewire

- Approach answers a basic question
  - Q: how to find a path from seed to mouse that follows object boundary as closely as possible ?
  - A: Instead of connecting user-selected anchor points with straight lines, a minimum cost path is chosen. Low costs can be found along pixels that exhibit strong edge features.



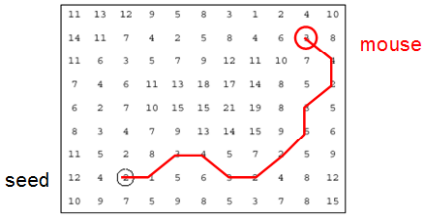
(a) (b) (c)

(a) as the mouse traces the white path, the scissors follow the orange path along the object boundary (the green curves show intermediate positions) (Mortensen and Barrett,1995)
(b) regular scissors can sometimes jump to a stronger (incorrect) boundary
(c) after training to the previous segment, similar edge profiles are preferred (Mortensen and Barrett,1998)

Computer Vision

Intelligent scissors

- Basic Idea
  - Define edge score for each pixel
    - edge pixels have low cost
  - Find lowest cost path from seed to mouse



- Questions
  - How to define costs?
  - How to find the path?

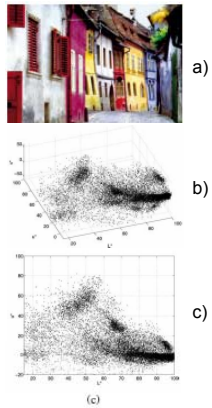
- *see: E. N. Mortensen and W. A. Barrett, Intelligent Scissors for Image Composition, ACM Computer Graphics (SIGGRAPH '95), pp. 191-198, 1995*

IMAGE PROCESSING & ANALYSIS

Mean-shift clustering

Mean shift & Mode finding

- **Mean-shift** and **mode finding** techniques,
  such as **k-means** and **mixtures of Gaussians**,
  model the feature vectors associated with each pixel
  (e.g., color and position)
  as samples from an unknown probability density function and
  then try to find clusters (modes) in this distribution.
- Consider the color image shown in Figure a.
  How would you segment this image based on color alone?
- Figure b shows the distribution of pixels in L*u*v* space,
  which is equivalent to what
  a vision algorithm that ignores spatial location would see.
- To make the visualization simpler,
  let us only consider the L*u* coordinates, as shown in Figure c.
- How many obvious (elongated) clusters do you see?
  How would you go about finding these clusters?
- The **k-means** and **mixtures of Gaussians** techniques
  use a **parametric model** of the density function to answer this question, i.e.,
  they assume the
  density is the superposition of a small number of simpler distributions
  (e.g., Gaussians)
  whose locations (centers) and shape (covariance) can be estimated.
- **Mean shift**, on the other hand,
  smoothes the distribution and finds its peaks
  as well as the regions of feature space that correspond to each peak
  using a non-parametric technique.

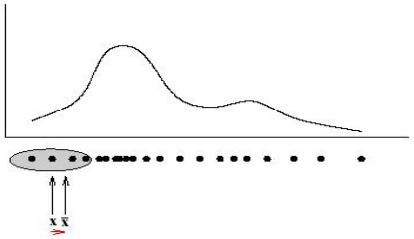


Mean-shift algorithm

- The mean shift algorithm seeks the “mode”
  or point of highest density of a data distribution
  - using a non-parametric technique
  - based on Robust Statistics
    - "robust" is used in its formal statistical sense:
      that is, mean-shift ignores outliers in the data
      - this means that it ignores data points that are far away
        from the peaks in the data
    - it does so by processing only those points
      that are within a local window of the data
      and then moving the window
- The key to mean shift is
  a technique for efficiently finding peaks
  in this high-dimensional data distribution
  without ever computing the complete function explicitly

Mean-shift algorithm

- What is mean-shift ?
  - Given any density function,
    the mean of a set of samples taken about **x**
    will be biased towards a local mode (i.e. local maximum)
  - The mean-shift vector **x−x**
    points in the direction of the gradient
- 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.



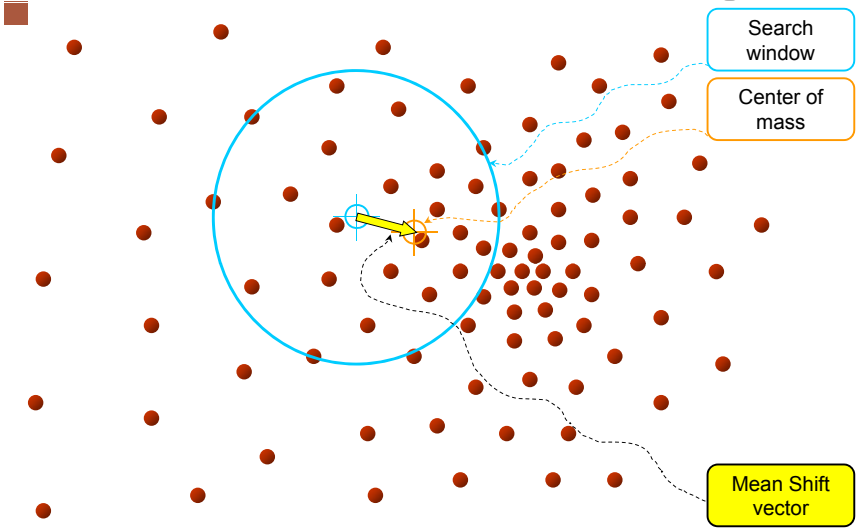
Mean-shift algorithm

1. Choose a search window
   1. its initial location
   2. its type
      - uniform, polynomial, exponential, or Gaussian
   3. its shape
      - symmetric or skewed, possibly rotated, rounded or rectangular
   4. its size
      - extent at which it rolls off or is cut off
2. Compute the window's (possibly weighted) center of mass
3. Center the window at the center of mass
4. Return to step 2 until the window stops moving
   (it will always will ...!)

source: Bradski book

Computer Vision

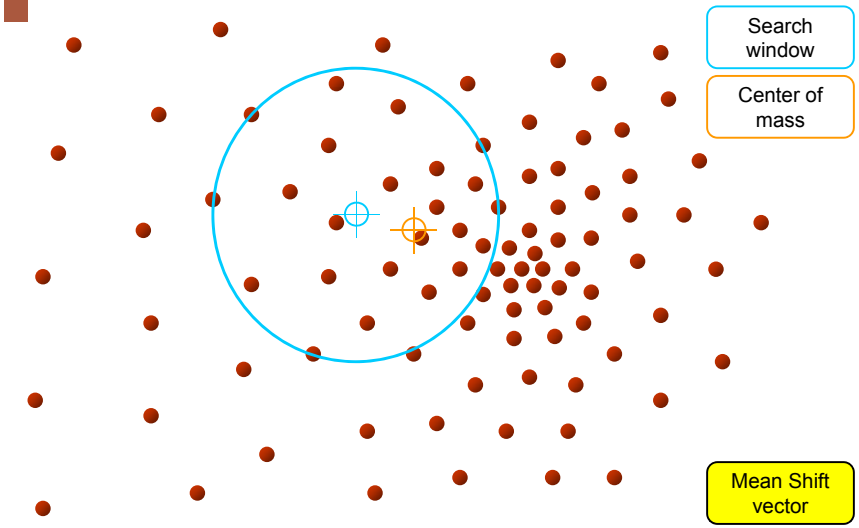
Mean-shift algorithm



Source: Y. Ukrainitz & B. Sarel

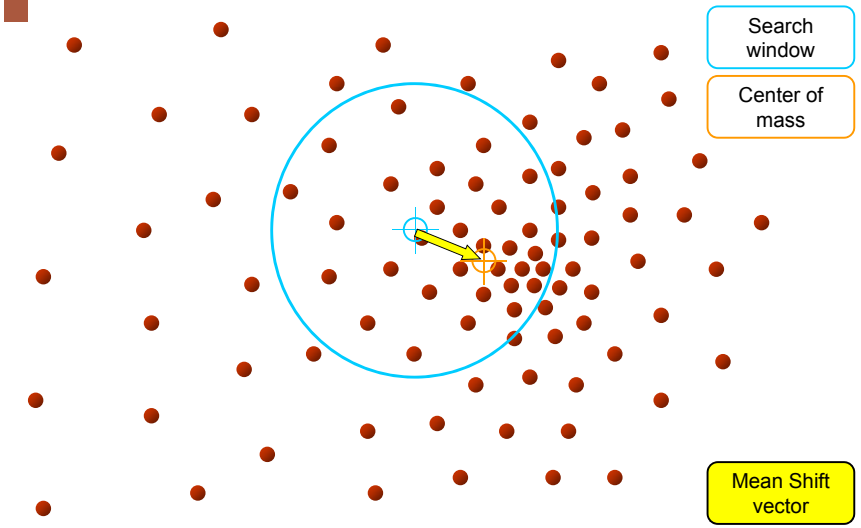
Computer Vision

Mean-shift algorithm

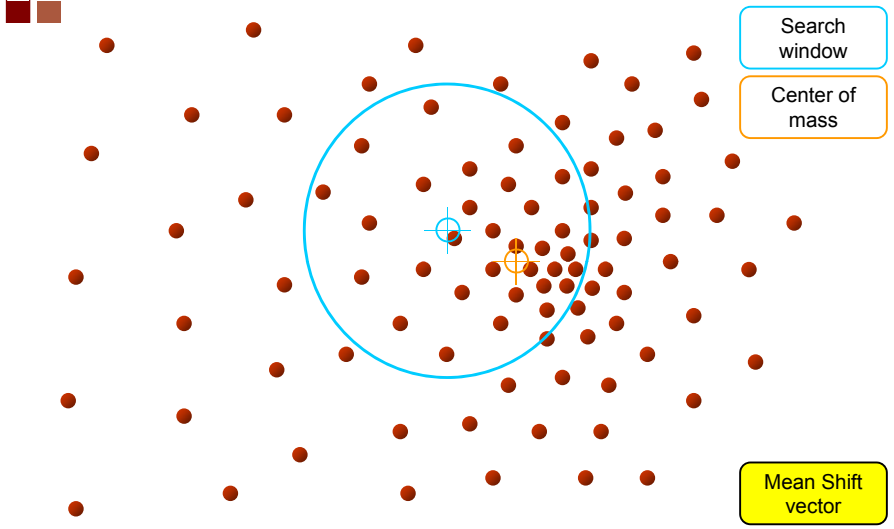


Computer Vision

Mean-shift algorithm

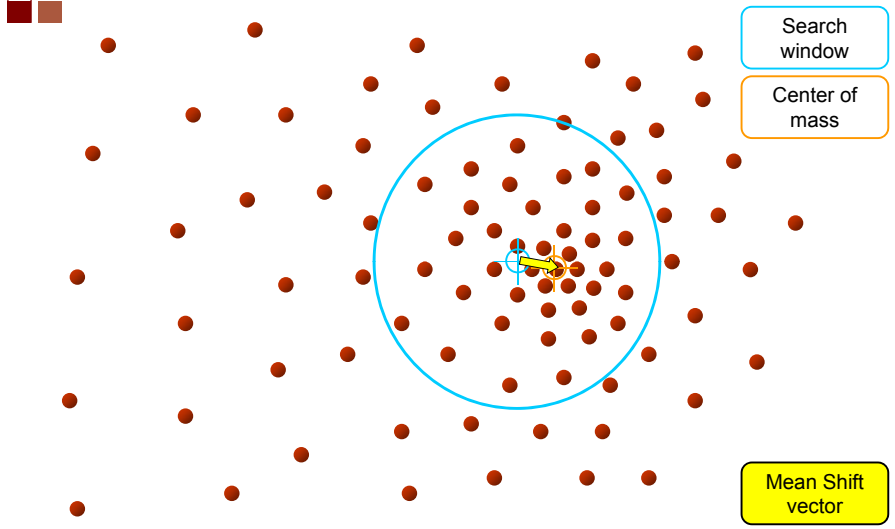


Mean-shift algorithm



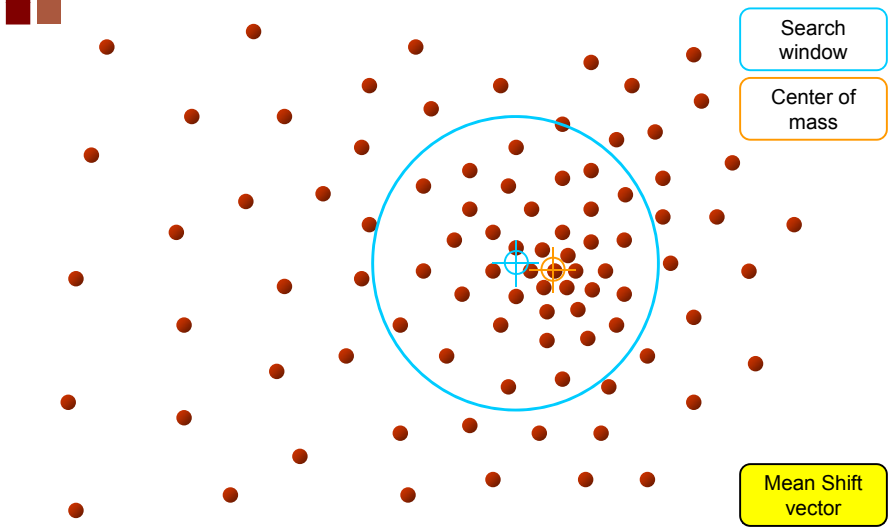
Computer Vision

Mean-shift algorithm



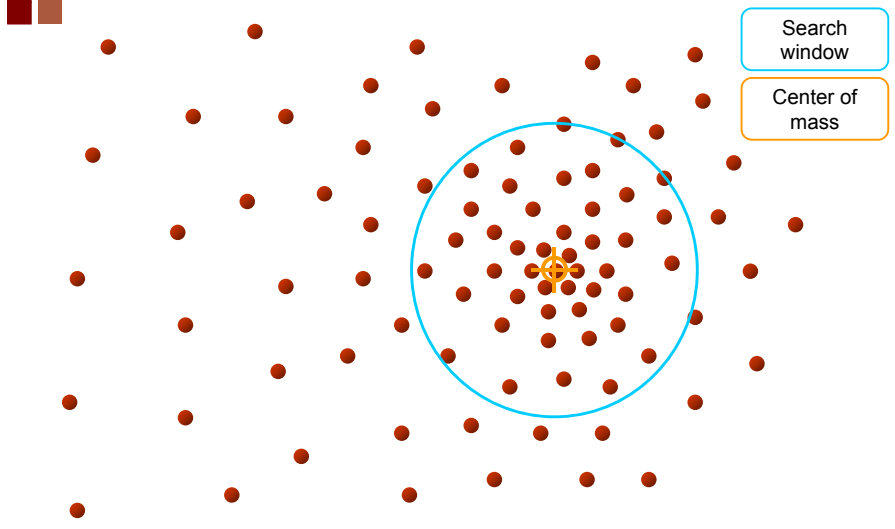
Computer Vision

Mean-shift algorithm

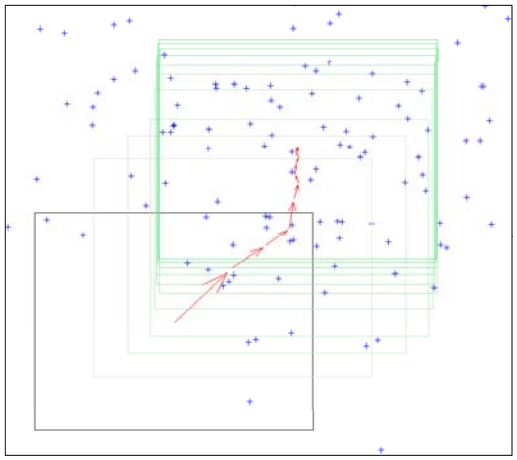


Computer Vision

Mean-shift algorithm



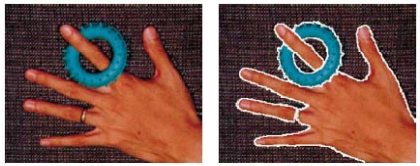
Mean-shift algorithm



Mean-shift algorithm in action: an initial window is placed over a 2D array of data points and is successively recentered over the mode (or local peak) of its data distribution until convergence

Mean-shift for Image Segmentation

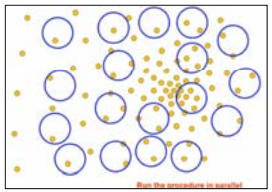
- Mean-shift for IMAGE SEGMENTATION
  - Convert the image into tokens (via color, gradients, texture measures, etc).
  - Choose initial search window locations uniformly in the data.
    - *you could place a window at each data point ... (heavy!)*
  - Compute the mean shift window location for each initial position.
  - Merge windows that end up on the same "peak" or mode.
  - The data these merged windows traversed are clustered together.



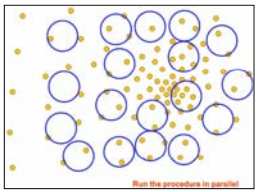
Mean-shift color image segmentation, Comaniciu and Meer 2002

Mean-shift for Image Segmentation

- Window initialization and peak finding

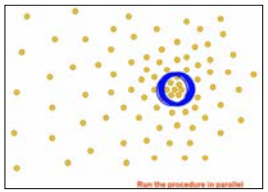


Initialization
(*could place a window at each data point ... (heavy!)*

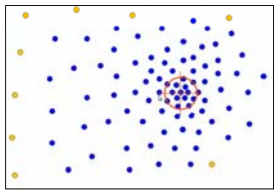


Movement

...



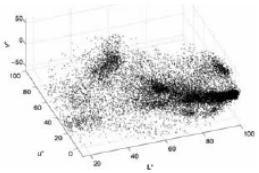
Peak found



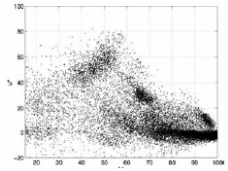
The blue data points were traversed by the windows towards the mode

Mean-shift for Image Segmentation

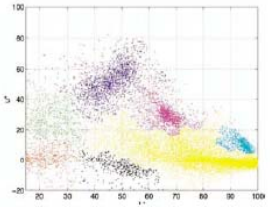
a) input color image



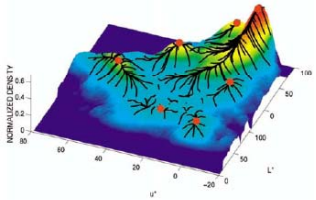
b) pixels plotted in L*u*v* space



c) L*u* space distribution



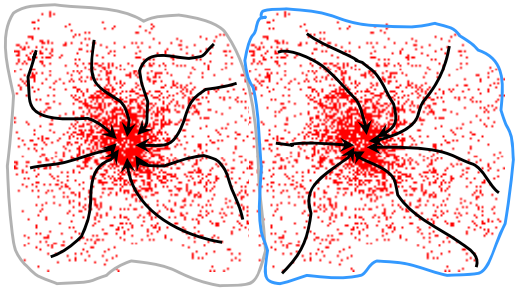
d) clustered results after 159 mean-shift procedures;



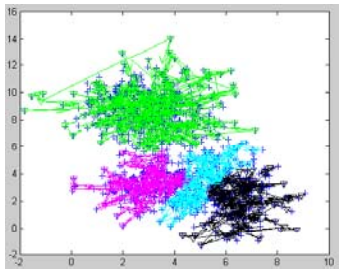
e) corresponding trajectories with peaks marked as red dots

Mean-shift image segmentation (Comaniciu and Meer 2002).

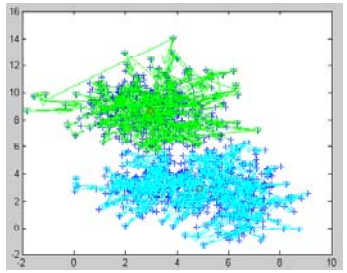
- Cluster:
  - all data points in the attraction basin of a mode
- Attraction basin:
  - the region for which all trajectories lead to the same mode



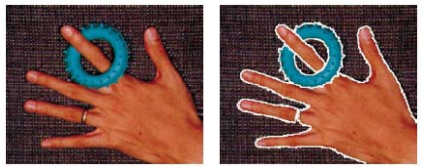
- Mean-shift is scale (search window size) sensitive.
  - Solution, use several scales.



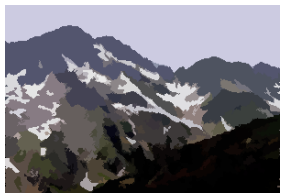
Best 4 clusters



Best 2 clusters



Mean-shift
color image segmentation,
Comaniciu and Meer 2002



http://www.caip.rutgers.edu/~comaniciu/MSPAMI/msPamiResults.html

Perhaps one of the best segmentation techniques ...

Mean-shift pros & cons

- Pros
  - Does not assume spherical clusters
  - Just a single parameter (window size)
  - Finds variable number of modes
  - Robust to outliers
- Cons
  - Computationally expensive
  - Does not scale well with dimension of feature space
  - Output depends on window size

Camshift

- Camshift (Continuously adaptive mean-shift) is a tracking algorithm related with mean-shift
- If differs from mean-shift in that the search window adjusts itself in size.
- If you have well-segmented distributions (e.g. face features that stay compact), then this algorithm will automatically adjust itself for the size of face as the person moves closer to and further from the camera.

- https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_video/py_meanshift/py_meanshift.html

OpenCV

- **cv::pyrMeanShiftFiltering**
  - Performs initial step of mean-shift segmentation of an image *(the previously described mean-shift algorithm)*
  - see *meanshift_segmentation.cpp* for an example
- **cv::meanShift**
  - **Finds an object on a back projection image (see next slides).**
  - The function implements the iterative object search algorithm.
  - It takes the input back projection (*see calcBackProject*) of an object and the initial position.
  - The mass center in window of the back projection image is computed and the search window center shifts to the mass center.
  - The procedure is repeated until the specified number of iterations criteria.maxCount is done or until the window center shifts by less than criteria.epsilon .
  - The algorithm is used inside CamShift() and, unlike CamShift() , the search window size or orientation do not change during the search.
  - You can simply pass the output of calcBackProject() to this function. But better results can be obtained if you pre-filter the back projection and remove the noise. For example, you can do this by retrieving connected components with findContours() , throwing away contours with small area (contourArea()), and rendering the remaining contours with drawContours() .
- **cv::CamShift**
  - Finds an object center, size, and orientation.
    - The function implements the CAMSHIFT object tracking algrorithm [Bradski98].
    - First, it finds an object center using meanShift() and then adjusts the window size and finds the optimal rotation.
    - Returns the rotated rectangle structure that includes the object position, size, and orientation.
  - See *camshiftdemo.cpp* that tracks colored objects

OpenCV

- **int meanShift(InputArray probImage, Rect& window, TermCriteria criteria)**
- Finds an object on a back projection image.
- Parameters:
- probImage – Back projection of the object histogram; see calcBackProject() for details.
- window – Initial search window.
- criteria – Stop criteria for the iterative search algorithm.
- The function implements the iterative object search algorithm.
- It takes the input back projection of an object and the initial position.
- The mass center in window of the back projection image is computed and the search window center shifts to the mass center.
- The procedure is repeated until the specified number of iterations criteria.maxCount is done or until the window center shifts by less than criteria.epsilon .
- The algorithm is used inside CamShift() and, unlike CamShift() , the search window size or orientation do not change during the search.
- You can simply pass the output of calcBackProject() to this function. But better results can be obtained if you pre-filter the back projection and remove the noise. For example, you can do this by retrieving connected components with findContours() , throwing away contours with small area (contourArea()), and rendering the remaining contours with drawContours() .

Histogram backprojection

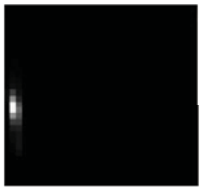
- What is Back Projection?
  - Back Projection is a way of recording how well the pixels of a given image fit the distribution of pixels in a histogram model.
  - To make it simpler: for Back Projection, you calculate the histogram model of a feature and then use it to find this feature in an image.
  - Application example: If you have a histogram of flesh color (say, a Hue-Saturation histogram), then you can use it to find flesh color areas in an image.

Computer Vision

Histogram backprojection

- How does it work?
  - We explain this by using the skin example.
  - Let's say you have gotten a skin histogram (Hue-Saturation) based on the image on the right. The histogram besides is going to be our model histogram (which we know represents a sample of skin tonality).
    - You applied some mask to capture only the histogram of the skin area.
  - What we want to do is to use our model histogram (that we know represents a skin tonality) to detect skin areas in our Test Image.

Hand ("model" of tonality that one wants to detect)



Hue-Saturation histogram of the hand

Computer Vision

Histogram backprojection

- Here are the steps:
  - In each pixel of our Test Image (i.e. p(i, j)), collect the data and find the correspondent bin location for that pixel - ($h_{i,j}$, $s_{i,j}$).
  - Lookup the model histogram in the correspondent bin – ($h_{i,j}$, $s_{i,j}$) - and read the bin value.
  - Store this bin value in a new image (BackProjection).
  - Also, you may consider to normalize the model histogram first, so the output for the Test Image can be visible for you.
- Applying the steps above, we get the BackProjection image for our Test Image:
- In terms of statistics, the values stored in BackProjection represent the probability that a pixel in Test Image belongs to a skin area, based on the model histogram that we use.
- For instance in our Test image, the brighter areas are more probable to be skin area (as they actually are), whereas the darker areas have less probability (notice that these "dark" areas belong to surfaces that have some shadow on it, which in turns affects the detection).

Test image

BackProjection image

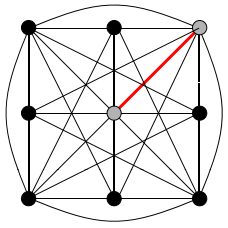
Computer Vision

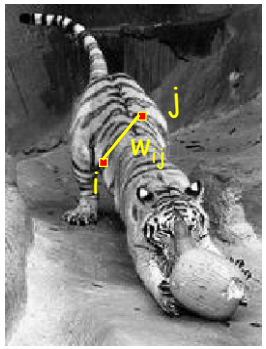
IMAGE PROCESSING & ANALYSIS

Graph cuts

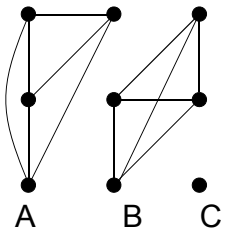
- Represent image as a graph
  - A vertex (node) for each pixel (!)
  - Edges between pixels
- Weights on edges reflect similarity (affinity) in:
  - Brightness
  - Color
  - Texture
  - Distance
  - …
- Connectivity:
  - Fully connected: edges between every pair of pixels
  - Partially connected: edges between neighboring pixels

Excellent introduction by Prof. Guillermo Sapiro (Duke University)
http://www.youtube.com/watch?v=HMGX8HXskKk





- Node for every pixel
- Edge between every pair of pixels
  (or every pair of "sufficiently close" pixels)
- Each edge is weighted by
  the *affinity* or similarity of the two nodes

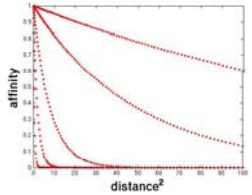


- Break graph into segments
  - Delete links that cross between segments
  - Easiest to break links that have low affinity
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

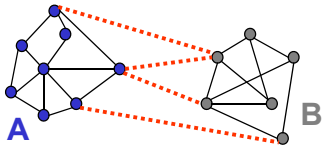
- Suppose we represent each pixel by a feature vector x,
  and define a distance function
  appropriate for this feature representation
- Then we can convert the distance between two feature vectors into an affinity with the help of a generalized Gaussian kernel:

$$\exp\left(-\frac{1}{2\sigma^2}\mathrm{dist}(\mathbf{x}_i,\mathbf{x}_j)^2\right)$$

- Scale, σ , affects affinity:
  - Small σ: group only nearby points
  - Large σ: group far-away points



Graph cut



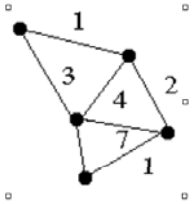
- Graph cut:
  - set of edges whose removal makes a graph disconnected
- Cost of a cut - *cut(A,B)*:
  - sum of weights of cut edges

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$

- A graph cut gives us a segmentation
  - What is a "good" graph cut and how do we find one?

Weighted graphs & Minimum cuts

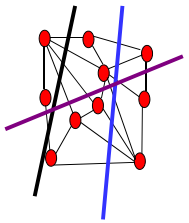
Weighted graphs and their representations



$$\begin{bmatrix} 0 & 1 & 3 & \infty & \infty \\ 1 & 0 & 4 & \infty & 2 \\ 3 & 4 & 0 & 6 & 7 \\ \infty & \infty & 6 & 0 & 1 \\ \infty & 2 & 7 & 1 & 0 \end{bmatrix}$$

Weight Matrix: W

Minimum Cut



A **cut** of a graph G is
the set of edges S such that
removal of S from G disconnects G.

Minimum cut is the cut of minimum weight

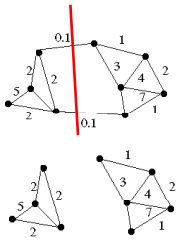
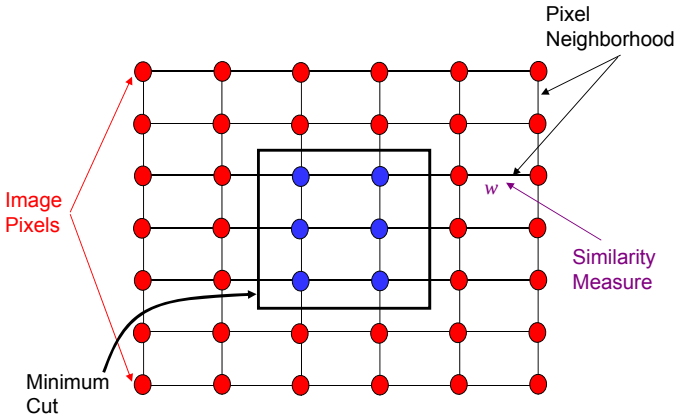
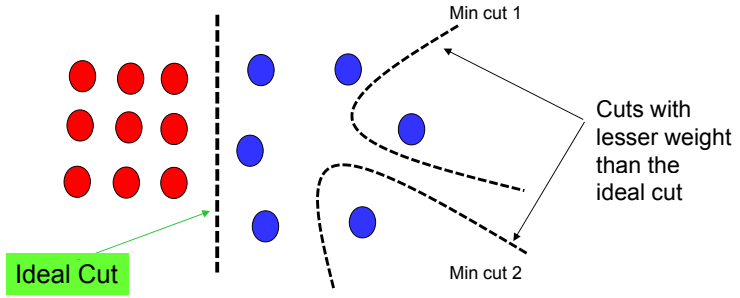


Image Segmentation & Minimum Cut



Minimum cuts

- There can be more than one minimum cut in a given graph.
- Min cut is not always the best cut
- Using a **minimum cut** as a segmentation criterion,
  does not usually result in reasonable clusters,
  since the smallest cuts usually involve isolating a single pixel.



Normalized cuts

- A better measure of segmentation is the **normalized cut**, which is defined as

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

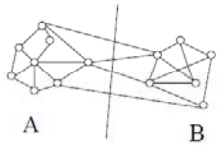
V – the whole graph
A and B – the clusters

where $assoc(A,A) = \sum_{i \in A, j \in A} w_{ij}$

is the association (sum of all the weights) within a cluster

and $assoc(A,V) = assoc(A,A) + cut(A,B)$

is the sum of all the weights associated.



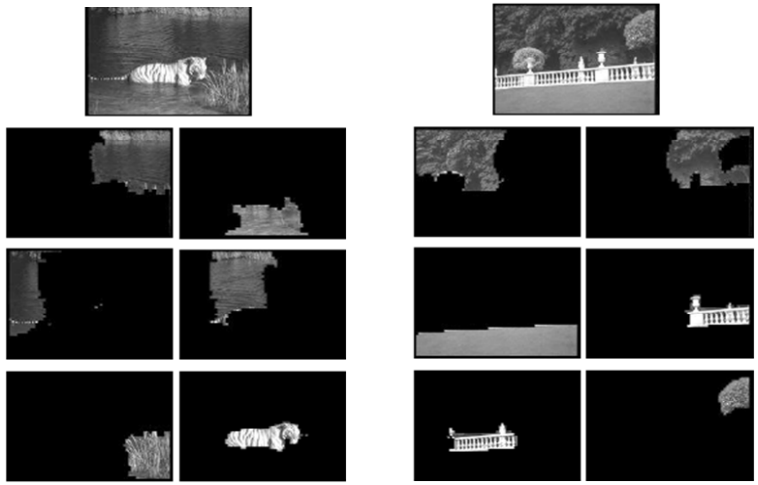
- The cut cost is computed as a fraction of the total edge connections to all nodes in the graph
- $N_{cut}(A,B)$ is the measure of dissimilarity of sets A and B.
- Minimizing $N_{cut}(A,B)$ maximizes a measure of similarity within the sets A and B.

J. Shi and J. Malik, "Normalized Cuts & Image Segmentation," IEEE Trans. on PAMI, Aug/2000

Normalized cuts

- Pros
  - Generic framework, can be used with many different features and affinity formulations
- Cons
  - Huge storage requirement and time complexity
    - Requires the solution of large sparse eigenvalue problems
    - Minimizing a normalized cut is NP-complete
  - Bias towards partitioning into equal segments
  - Problems with textured backgrounds

Normalized cuts: results



Graph cuts

- Approach from graph theory: represent image as undirected graph
  - One vertex per pixel / voxel
  - Neighbors are connected by edges in graph, whose weights correspond to cutting cost (e.g. inverse of gradient magnitude)
- User-interaction: setting seed points or regions (belonging to object or background) which will be labeled as terminals
- Minimum cost cut in graph provides an optimal segmentation
- Application of algorithms from graph theory
- *Boykov, Y. and Jolly, M.-P. (2001). Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In Eighth International Conference on Computer Vision (ICCV 2001), pp. 105–112, Vancouver, Canada.*

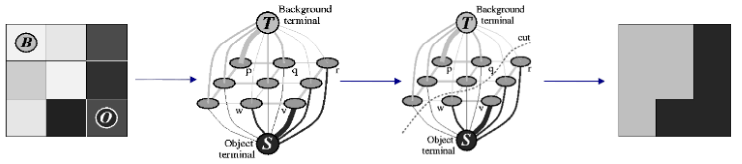
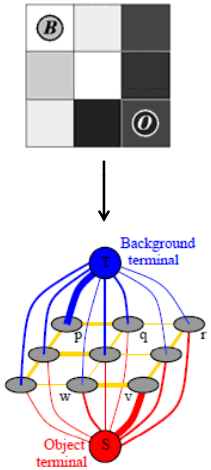


Image with seeds **Graph** **Cut** **Final segmentation**

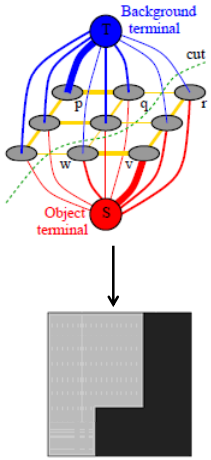
GRAPH CONSTRUCTION

- We define a **graph** (undirected): **G=<V,E>**
- The **vertices** represent image pixels / voxels
- Two special **nodes** S(source) and T(sink) that represent "object" and "background" labels: **terminals**
- Neighboring pixels are interconnected by graph edges: **n-links**
- Another type of edges are used to connect pixels to terminals (object and background): **t-links**
- All edges are assigned some non-negative weight: **cost**



Find a cut:

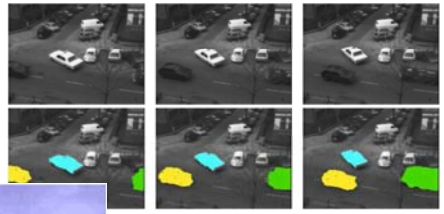
- **n-links**(neighboring links): cost of segmentation boundary
- **t-links**(terminal links): cost of regional properties
  - t-links with infinit cost make it possible to impose hard constraints
- A **minimum cost cut** may correspond to segmentation with desirable balance of boundary and regional properties
- Based on combinatorial optimization of computing a **minimum s-t cut**

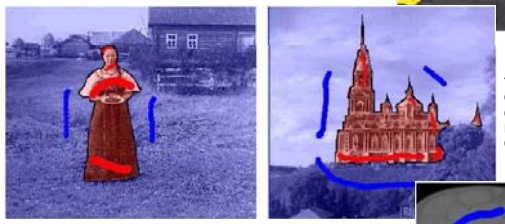


- A **s-t cut** is a subset of edges **C∈ E** such that the terminals S and T become completely separated on the induced graph **G(C) = <V,E\C>**
- Any cut corresponds to some binary partitioning of an image into"object"and "background"
- The **goal** is to compute the **best cut** that gives an "optimal"segmentation (i.e. a cut with minimal cost)

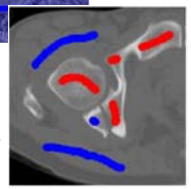
- RESULTS

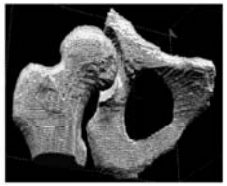
Yuri Boykov, Marie-Pierre Jolly: Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images. International Conference on Computer Vision (ICCV), 2001





Yuri Boykov, Gareth Funk-Lea. Optimal Object Extraction via Constrained Graph-Cuts. International Journal of Computer Vision, VISI 666-NP





Boykov and Funka-Lea, 3D volumetric medical image segmentation using graph cuts , 2006
(a) computed tomography (CT) slice with some seeds;
(b) recovered 3D volumetric bone model

- The basic binary segmentation algorithm of Boykov and Jolly (2001) has been extended in a number of directions.
- The GrabCut system of Rother, Kolmogorov, and Blake (2004) iteratively re-estimates the region statistics, which are modeled as a mixtures of Gaussians in color space.
- This allows their system to operate given minimal user input, such as a single bounding box *(see next slide)*
  - the background color model is initialized from a strip of pixels around the box outline.
  - The foreground color model is initialized from the interior pixels, but quickly converges to a better estimate of the object.
- The user can also place additional strokes to refine the segmentation as the solution progresses.
- In more recent work, Cui, Yang, Wen et al. (2008) use color and edge models derived from previous segmentations of similar objects to improve the local models used in GrabCut.

GrabCut

- GrabCut is computationally more expensive than watershed, but it generally produces better results
- It is the best algorithm to use when one wants to extract a foreground object in a still image (e.g., to cut and paste an object from one picture to another)

(a) (b) (c)

source: Szeliski book

(a) the user draws a bounding box in red;
(b) the algorithm guesses color distributions for the object and background and performs a binary segmentation;
(c) the process is repeated with better region statistics.

OpenCV

cv::grabCut

- Input:
  - an image +
  - + a rectangle where the objects are contained
  - OR
  - +a set of pixels that certainly belong to the background/foreground
  - number of iterations
- Procedure
  - A foreground label (cv::GC_PR_FGD) is tentatively assigned to all unmarked pixels
  - Based on the current classification, the algorithm groups the pixels into clusters of similar colors
  - A background/foreground segmentation is done by introducing boundaries between background and foreground pixels
    - done through an optimization process (GrabCut algorithm) that
    - tries to connect pixels with similar labels and
    - imposes a penalty for placing a boundary in regions of relatively uniform intensity
  - The obtained segmentation produces new labels for the pixels
  - The clustering process is repeated and a new optimal segmentation is found again, and so on
  - *Depending on the complexity of the scene, a good solution can be found in more or less iterations (in easy cases, one iteration can be enough)*
- Output:
  - a labelled image (cv::GC_PR_BGD and cv::GC_PR_FGD labels) from which binary images representing the background and the foreground pixels can be extracted

IMAGE PROCESSING & ANALYSIS

Other segmentation methods

Background subtraction

Atlas-based segmentation

Background subtraction

- If we know what the background looks like, it is easy to segment out new regions
- Applications
  - Detecting person in an office
  - Tracking cars on a road
  - Surveillance
  - Video game interfaces
- Approach
  - estimate background image
  - subtract from current frame
  - large absolute values are interesting pixels

Background estimation

- Offline average
  - Pixel-wise mean values are computed during training phase
- Adjacent frame difference
  - Each image is subtracted from previous image in sequence
- Moving average (running average)
  - Background model is a weighted sum of previous frames

To be studied later .

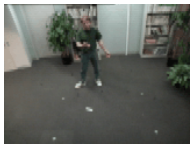
Background subtraction

from C. Stauffer and W. Grimson

Current image

Background image

Foreground pixels



Pfinder, MIT

(recognize human figures and their movements and gestures)

Atlas based segmentation

- Registration of pre-segmented data set (atlas) to actual image, e.g. by deformable **registration**
- Mapping of labels from the atlas to the data set
- Segmentation task is converted into registration task

IMAGE PROCESSING & ANALYSIS

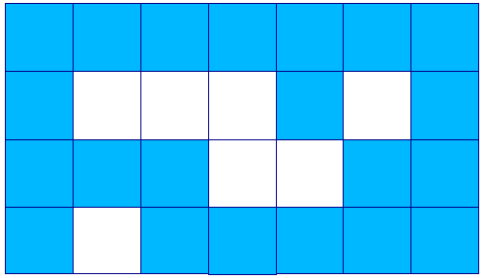
Post Processing Steps

Post-processing

- After segmenting an image, we usually want to:
  - Clean up
    the binary image or
    each one of the resulting regions
  - Know something
    about the regions found ...
    - How many objects are in the image?
    - What the the area, perimeter, position, ...
      of the distinct object components?
    - ...

Post-processing

- Ex: thresholding results in segmented pixels,
  which are totally unrelated
- Post processing by morphological operations and/or
  analyzing connected components necessary



Post-processing

Post-processing steps

- Morphologic operations, for
  - closing gaps and holes
  - removing “noses”
  - computing centerlines
  - ...
- Quantitative analysis
  - center, area, permiter, volume, ...
  - principal axes (for orientation)
  - ...
- ...

Connected components

- Left to right, top to bottom:
  - If pixel is connected to one other, already labeled, pixel
    then copy the label
  - If pixel is connected to two other, already labeled, pixels
    then unify the two corresponding regions
    by adapting one label for every connected pixel
    else assign a new label

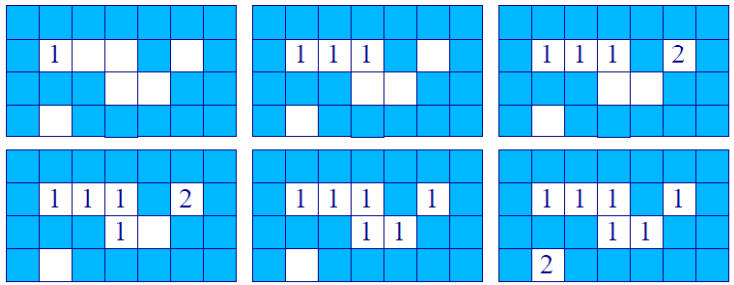


IMAGE PROCESSING & ANALYSIS

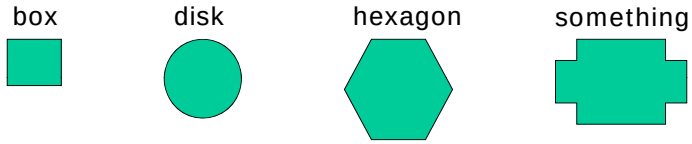
Morphological image processing

Morphological operations

- Morphological image processing
  - a set of operations by which the spatial structure of objects within an image is modified.
- May be applied to:
  - binary images
  - gray-level images
- Binary mathematical morphology consists of two **basic operations**
  - **erosion**
  - **dilation**
- and several compound operations:
  - opening
  - closing
  - hit or miss
  - ...
- When used correctly, morphological operations preserve the essential features of shape of an object, while removing the irrelevant details.

Morphological operations

- The arguments to dilation and erosion are:
  - a binary image **B**
  - a structuring element **S**
- **Structuring element** (or **kernel**)
  - A structuring element is a shape mask used in the basic morphological operations.
  - They can be any shape and size that is digitally representable, and each has an origin.
  - It can even correspond to a particular shape that is being sought for in the image.

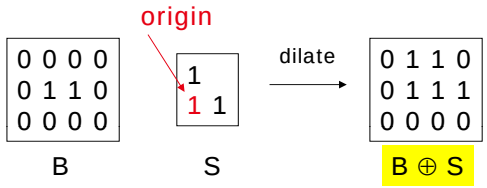


Dilation

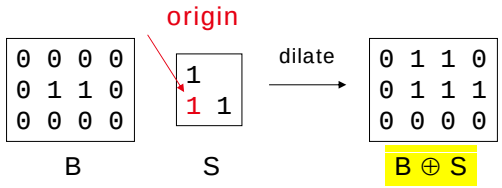
dilate(B,S)

- takes binary image B,
- places the origin of structuring element S over each 1-pixel, and
- ORs the structuring element S into the output image at the corresponding position.

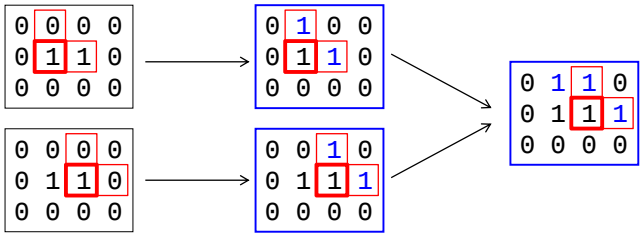
- Dilation is both associative and commutative
  - this fact allows breaking a complex shape into several simpler shapes which can be combined as a sequence of dilations



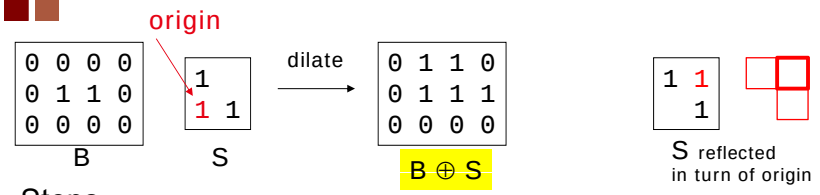
Dilation



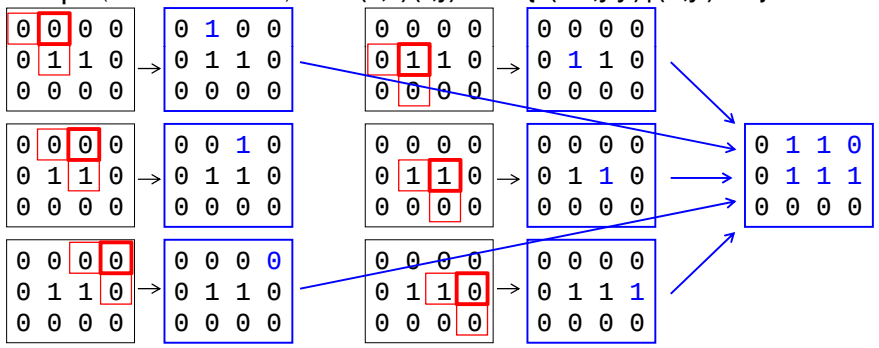
Steps:



Dilation



Steps (alternative formulation): **dilate (B,S) (x,y) = max {B(x-x´,y-y') | (x´,y´) ∈ S }**

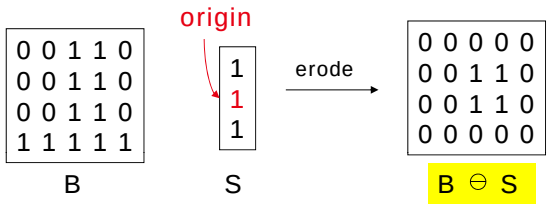


Erosion

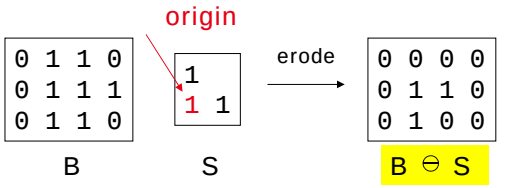
erode(B,S)

- takes a binary image B,
- places the origin of structuring element S over every pixel position, and
- ORs a binary 1 into that position of the output image only if every position of S (with a 1) covers a 1 in B.

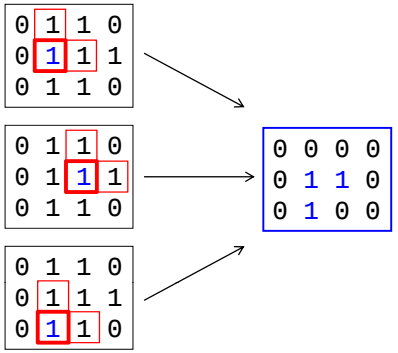
- Note:
  erosion gives all the locations where the structuring element is contained in the image



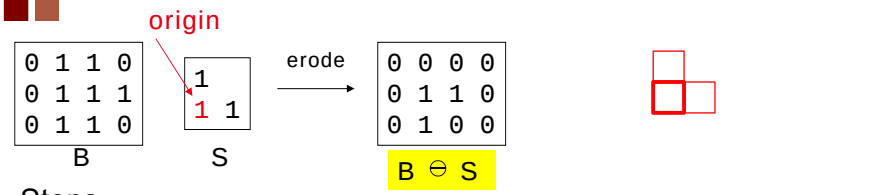
Erosion



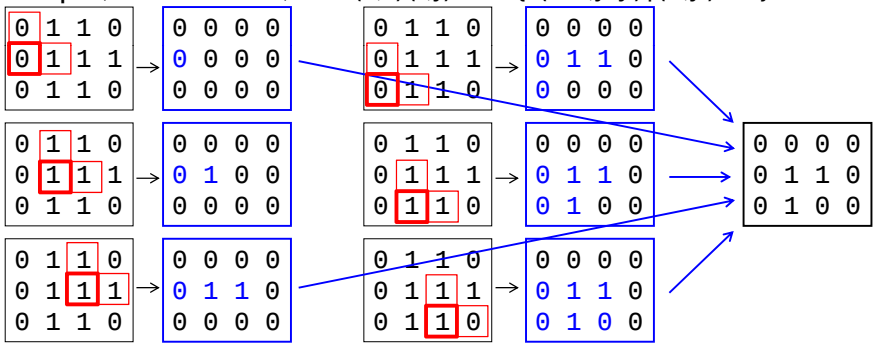
Steps:



Erosion



Steps (alternative formulation): **erode(B,S) (x,y) = min {B(x+x´,y+y') | (x´,y´) ∈ S }**



Dilation and erosion

- Dilation
  - expands the connected sets of 1s of a binary image.
  - it can be used for
    - growing features
    - filling holes and gaps
- Erosion
  - shrinks the connected sets of 1s of a binary image.
  - it can be used for
    - shrinking features
    - removing bridges, branches and small protrusions
- Dilation and erosion arise in a wide variety of contexts such as:
  - removing noise,
  - isolating individual elements, and
  - joining disparate elements in an image.
- Morphological operators can also be used
  - to find intensity bumps or holes in an image and
  - to find image gradients.

Opening and closing

Opening

- is the compound operation of erosion followed by dilation (with the same structuring element)
- **open(B,S)** = dilate(erode(B,S),S)

Closing

- is the compound operation of dilation followed by erosion (with the same structuring element)
- **close(B,S)** = erode(dilate(B,S),S)

Morphological operations: results

original dilation erosion opening closing

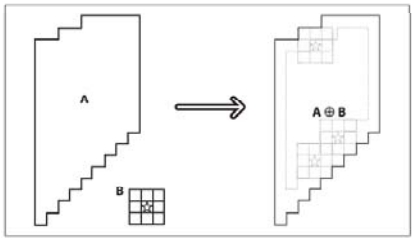
NOTE: in this example objects are black (=0) and background is white (=1)

- Dilation
  - $(g \oplus s)(x,y) = \max \{g(x-x',y-y') + s(x',y') | (x',y') \in D_s\}$
- Erosion
  - $(g \ominus s)(x,y) = \min \{g(x+x',y+y') - s(x',y') | (x',y') \in D_s\}$
- being
  - g(x,y) a graylevel image
  - s(x,y) a structuring element (graylevel)
    - D_s a binary matrix that defines the locations in the neighborhood that must be included in the max/min operations

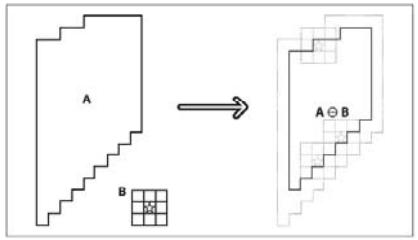
- When the structuring element is flat one can define s(x',y') = 0 for $(x',y') \in D_s$
- So ...
  - Dilation
    - $(g + s)(x,y) = \max \{g(x-x',y-y') | (x',y') \in D_s\}$
  - Erosion
    - $(g - s)(x,y) = \min \{g(x+x',y+y') | (x',y') \in D_s\}$
- ...and if the structuring element is symmetric ...
  - Dilation
    - $(g + s)(x,y) = \max \{g(x+x',y+y') | (x',y') \in D_s\}$
  - Erosion
    - $(g - s)(x,y) = \min \{g(x+x',y+y') | (x',y') \in D_s\}$

$$\text{erode}(x,y) = \min_{(x',y') \in \text{kernel}} \text{src}(x+x', y+y')$$
$$\text{dilate}(x,y) = \max_{(x',y') \in \text{kernel}} \text{src}(x+x', y+y')$$

kernel ⇔ structuring element

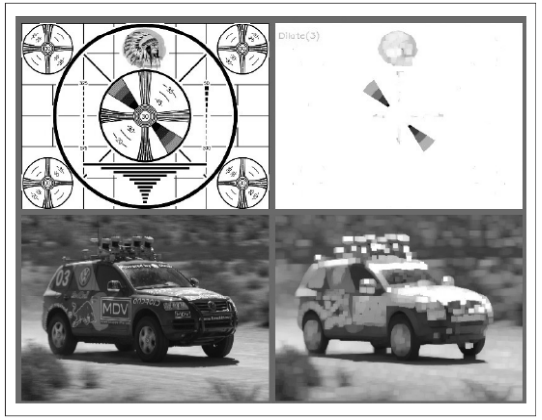


Dilation: take the **maximum** under the kernel B



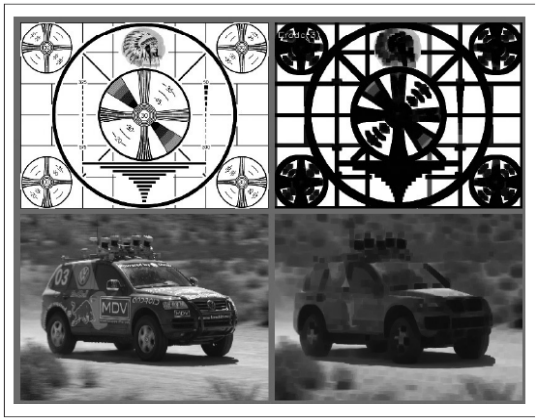
Erosion: take the **minimum** under the kernel B

(source Bradski book)



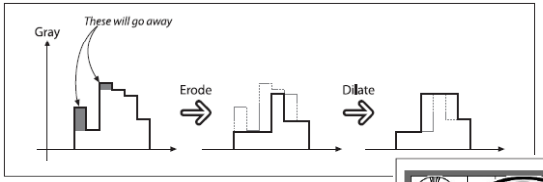
Results of the dilation, or "max", operator: bright regions are expanded and often joined
(source: Bradski book)

Erosion

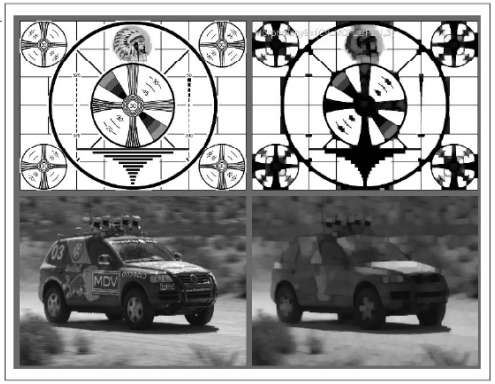


Results of the erosion, or “min”, operator: bright regions are isolated and shrunk
(source: Bradski book)

Opening

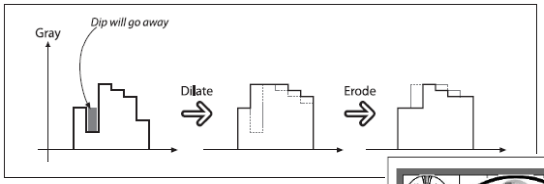


Morphological opening: the upward outliers are eliminated as a result

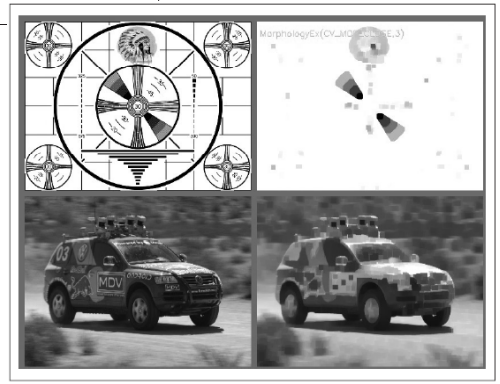


Results of opening on an image: small bright regions are removed, and the remaining bright regions are isolated but retain their size
(source: Bradski book)

Closing



Morphological closing: the downward outliers are eliminated as a result



Results of closing on an image: bright regions are joined but retain their basic size
(source: Bradski book)

Other morphological operations

- Thinning
- Thickening
- Hit-and-miss transform
  - can be used to search an image for particular instances of a shape or other characteristic image feature
- Skeletonization / Medial Axis Transform

http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm

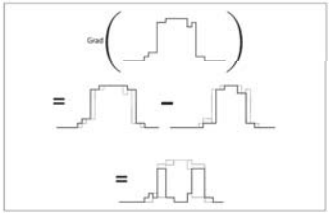
Other morphological operators

- Morphological gradient
  - **gradient(src) = dilate(src)–erode(src)**
    - the effect of this operation on a binary image would be simply to isolate perimeters of existing blobs.
- Top-hat *
  - **TopHat(src) = src–open(src)**
    - reveals areas that are lighter than the surrounding region
- Black-hat **
  - **BlackHat(src) = close(src)–src**
    - reveals areas that are darker than the surrounding region

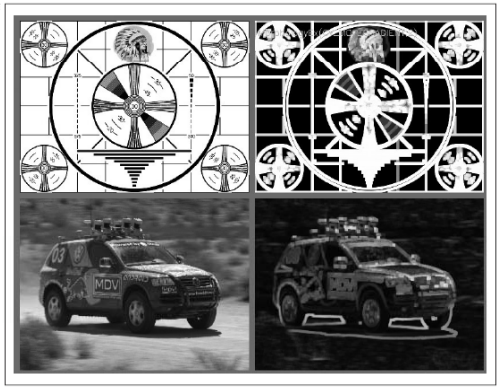
*, ** - Bradski book ("Learning OpenCV") designations;
"Black-hat" is also known as "Bottom-hat"

Morphological gradient

Morphological gradient



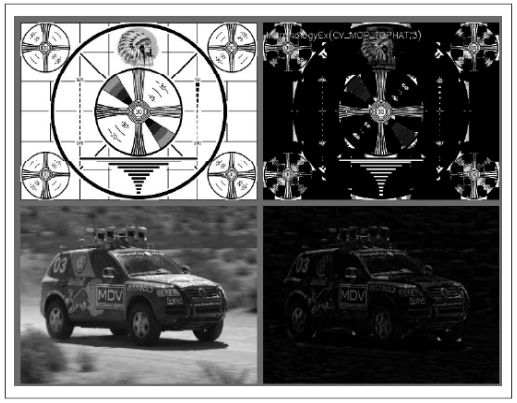
Morphological gradient
applied to a grayscale image:
as expected,
the operator has its highest values
where the grayscale image
is changing most rapidly



Results of the morphological gradient operator:
bright perimeter edges are identified

Top-hat

Top-hat



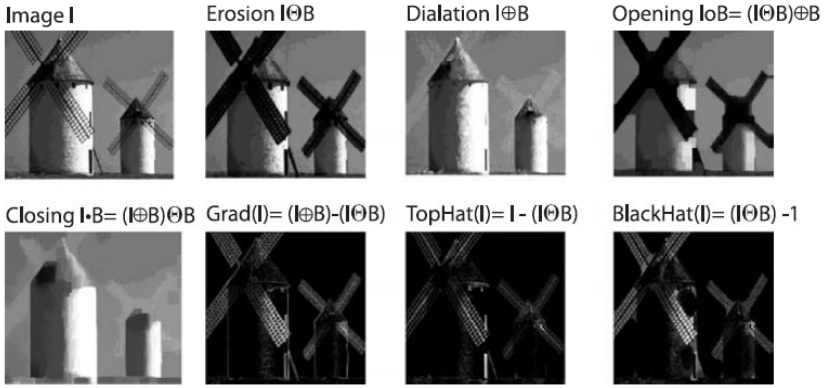
Results of Top-Hat:
bright local peaks are isolated

Black-hat

Black-hat

Results of Black-Hat operation:
dark holes are isolated

Morphological operations examples



Morphological operators usage

- Opening
  - often used before counting regions in a binary image.
    - Ex: a thresholded image of cells on a microscope slide, to separate out cells that are near each other before counting the regions.
- Closing
  - used in most of the more sophisticated connected-component algorithms to reduce unwanted or noise-driven segments.
- Note:
  - open and close tend to preserve the area of connected regions.
- Morphological gradient
  - often used when we want to isolate the perimeters of bright regions so we can treat them as whole objects (or as whole parts of objects).
  - The complete perimeter of a region tends to be found because an expanded version is subtracted from a contracted version of the region, leaving a complete perimeter edge.
  - This differs from calculating a gradient, which is much less likely to work around the full perimeter of an object.